Джон Дакетт

Все, что нужно знать
для разработки веб-приложений

PHP и MYSQL

Серверная веб-разработка

МИРОВОЙ
КОМПЬЮТЕРНЫЙ
БЕСТСЕЛЛЕР


МИРОВОЙ
КОМПЬЮТЕРНЫЙ
БЕСТСЕЛЛЕР

John Duckett



PHP & MYSQL

Server-side Web Development

WILEY

Джон Дакетт



PHP и MYSQL

Серверная веб-разработка



МОСКВА 2023

УДК 004.43
ББК 32.973.26-018.1
Д14

PHP & MySQL: Server-side Web Development
Jon Duckett

© 2022 by John Wiley & Sons, Inc., Indianapolis, Indiana.
All rights reserved. This translation published under license with the original publisher
John Wiley & Sons, Inc.

Дакетт, Джон.

Д14 PHP и MYSQL. Серверная веб-разработка / Джон Дакетт ; [перевод с английского М. А. Райтмана]. — Москва : Эксмо, 2023. — 688 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-04-171951-7

Перед вами практическое руководство по созданию веб-сайтов с помощью языка программирования PHP и системы управления базами данных MySQL. Джон Дакетт представляет читателю простой и понятный подход к созданию функциональных систем, начиная с базовых концепций и заканчивая разработкой сложных проектов. Книга содержит множество примеров кода, упражнений и советов по оптимизации производительности. Это необходимое руководство для начинающих и опытных веб-разработчиков, желающих создавать высококачественные веб-сайты и приложения.

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-04-171951-7

© Райтман М. А., перевод на русский язык, 2023
© Оформление. ООО «Издательство «Эксмо», 2023

ОГЛАВЛЕНИЕ

Введение ... 7

Часть А **Основы программирования** ... 23
Глава 1 Переменные, выражения и операторы ... 35
Глава 2 Управляющие конструкции ... 73
Глава 3 Функции ... 109
Глава 4 Классы и объекты ... 149

Часть Б **Динамические веб-страницы** ... 183
Глава 5 Встроенные функции ... 207
Глава 6 Получение данных из браузера ... 237
Глава 7 Изображения и файлы ... 291
Глава 8 Дата и время ... 315
Глава 9 Cookie и сессии ... 335
Глава 10 Обработка ошибок ... 355

Часть В **Сайты на основе баз данных** ... 387
Глава 11 Язык структурированных запросов (SQL) ... 403
Глава 12 Получение и вывод информации из базы данных ... 439
Глава 13 Изменение информации в базе данных ... 489

Часть Г **Расширение функционала и модернизация учебного приложения** ... 527
Глава 14 Рефакторинг и внедрение зависимостей ... 539
Глава 15 Пространства имен и библиотеки ... 563
Глава 16 Регистрация пользователей ... 609
Глава 17 Добавление нового функционала ... 639

Предметный указатель ... 669
Примечания ... 677

Загрузить код для этой книги можно на странице http://addons.eksmo.ru/it/phpbook.zip

ВВЕДЕНИЕ

Изучив эту книгу, вы научитесь создавать веб-сайты на языке программирования PHP и хранить информацию в базе данных MySQL.

PHP — это язык программирования, который был создан для работы на веб-сервере, чтобы генерировать HTML-страницы по запросу. Это означает, что страницы сайта не являются статичными, а могут отображать разную информацию для разных пользователей. Это обязательное требование для любого сайта, который позволяет пользователям выполнять такие задачи, как:

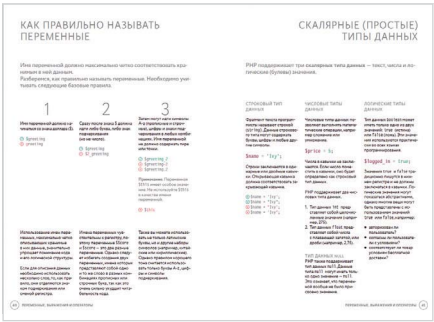
- **регистрация и вход в систему**. Имя, адрес электронной почты и пароль каждого пользователя уникальны;
- **совершение покупки**. Детали заказа, оплаты и доставки каждого клиента уникальны;
- **поиск на сайте**. Результаты поиска зависят от поисковой фразы.

Язык программирования PHP создан для работы с базами данных, например MySQL, которые могут хранить такие данные, как содержимое страниц, сведения о продаваемых товарах или посетителях сайта. Используя PHP, вы научитесь создавать веб-страницы, позволяющие пользователям обновлять информацию, хранящуюся в базе данных. Например:

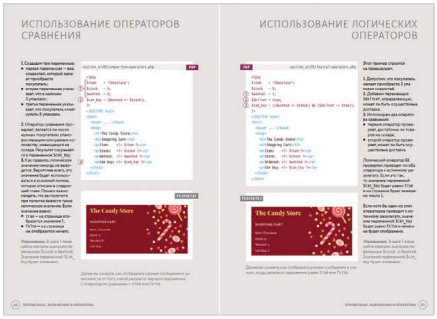
- **системы управления контентом** позволяют владельцам (администраторам) сайта обновлять с помощью формы содержимое сайта. Эти обновления отображаются для пользователей без написания нового кода;
- **интернет-магазины** позволяют владельцам выставлять товары для продажи, а покупателям — делать покупки;
- **социальные сети** позволяют пользователям регистрироваться и входить в систему, создавать свои профили, загружать собственный контент и просматривать страницы в зависимости от интересов.

Поскольку отображаемая на страницах сайтов информация хранится в базе данных, такие сайты известны как **веб-сайты на основе баз данных.**

Ниже представлены описанные в книге типы страниц. Различный дизайн страниц передает разную информацию.



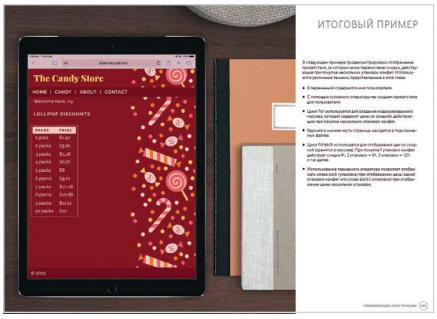
Информационные страницы отображаются на белом фоне. На информационных страницах обычно освещаются какие-либо темы, описывается контекст.



Страницы с кодом отображаются на бежевом фоне. Они представляют собой примеры кода по каждой изученной теме.



Диаграммы и инфографика отображаются на темном фоне.



Итоговые примеры в конце некоторых глав содержат код, обобщающий всю информацию, изученную в этой главе.



В конце каждой главы тезисно повторяется весь изученный в ней материал. На этих страницах кратко приведены ключевые понятия по теме.

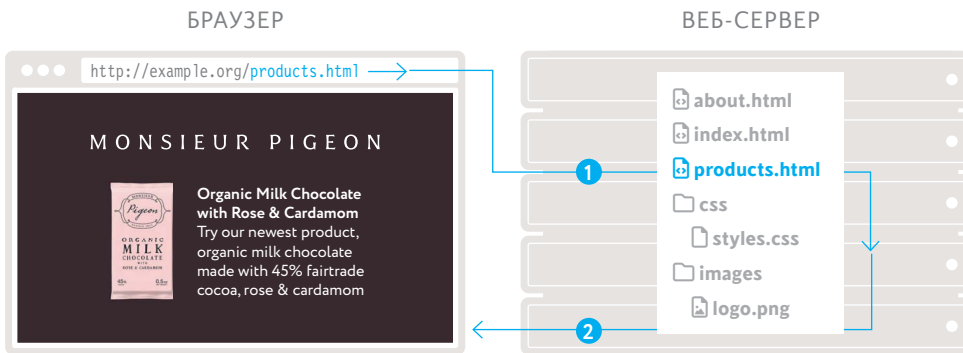
СТАТИЧЕСКИЕ И ДИНАМИЧЕСКИЕ ВЕБ-САЙТЫ

Когда сайт разработан только с использованием HTML и CSS, каждый пользователь будет видеть одно и то же содержимое, так как всем отправляются одни и те же HTML- и CSS- файлы.

1. Когда браузер запрашивает страницу сайта, созданного с использованием только HTML и CSS, запрос отправляется на **веб-сервер**, на котором размещен этот сайт.
2. Веб-сервер находит запрошенный браузером HTML-файл и возвращает его браузеру. Затем браузер запрашивает все файлы, которые упомянуты в HTML, такие как CSS для стилизации страницы, мультимедиа (например, изображений), JavaScript и другие используемые страницей файлы.

Поскольку всем пользователям отправляются одни и те же HTML-файлы, все они будут видеть один и тот же контент. Такой тип сайта известен как **статический сайт**.

Владельцам статических веб-сайтов необходимо владеть навыками HTML и CSS, чтобы иметь возможность обновлять информацию на сайте. Если владелец (администратор) такого сайта захочет обновить текст на странице, то HTML-код придется обновить вручную и загрузить на веб-сервер.



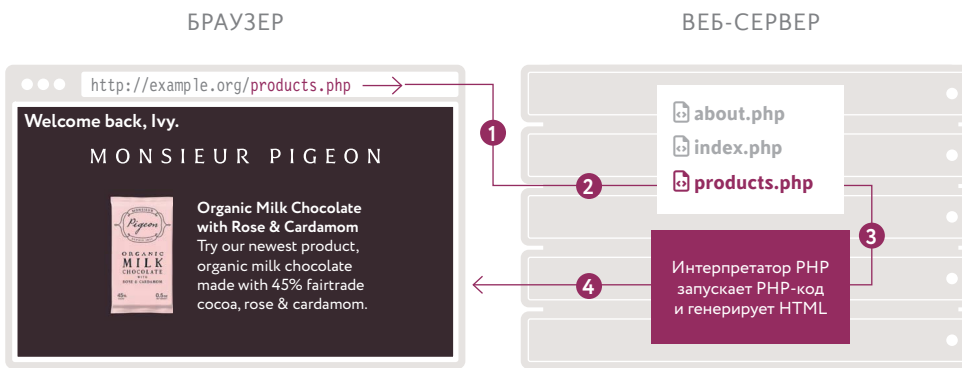
Материал этой книги предполагает базовое знакомство с HTML и CSS. Если вы только изучаете HTML и CSS, рекомендуем ознакомиться с нашей книгой: https://book24.ru/product/html-i-css-razrabotka-i-dizayn-veb-saytov-5136049/.

Когда веб-сайт разработан с использованием PHP, каждый пользователь может видеть разный контент, так как страница PHP каждый раз заново создает HTML-код, который отправляется пользователю.

Такие сайты, как eBay, Facebook* и новостные порталы, практически при каждом новом посещении отображают новую информацию. Если вы посмотрите на исходный код полученной страницы в браузере, то увидите HTML-код, однако разработчик не будет постоянно вручную обновлять его.

Такой тип сайта известен как **динамический веб-сайт**, поскольку HTML-код, который отправляется в браузер, каждый раз создается кодом PHP в ответ на запрос пользователя.

1. Когда браузер запрашивает страницу сайта, созданного на PHP, запрос отправляется на веб-сервер.
2. Веб-сервер находит файл PHP.
3. Весь имеющийся в файле PHP-код выполняется с помощью программы, называемой **интерпретатором PHP**, и новый HTML-код создается для конкретного пользователя, запросившего страницу.
4. Веб-сервер отправляет HTML-код, созданный для этого пользователя, в браузер. Копия файла не сохраняется. При следующем запросе PHP-файла HTML-код создается для этого же пользователя заново.



HP-код в веб-браузер не отправляется. Он используется для создания HTML-кода, который затем направляется в браузер. Поскольку PHP-код выполняется на стороне веб-сервера, он относится к **серверному программированию**.

PHP можно использовать для создания HTML-страниц, адаптированных или персонализированных для каждого отдельного пользователя. На страницах может отображаться имя пользователя, интересующие его темы или сообщения его друзей.

* Принадлежит компании Meta, которая признана экстремистской и запрещена на территории РФ.

PHP: ЯЗЫК И ИНТЕРПРЕТАТОР

Интерпретатор PHP — это программа, которая запускается на веб-сервере. Она выполняет программный код, написанный на языке PHP.

Программное обеспечение позволяет использовать компьютер для выполнения определенных задач. При этом нет необходимости досконально знать, как именно компьютер выполняет эту задачу. Например:

- программы электронной почты позволяют отправлять и получать электронные письма. При этом пользователю не нужно понимать, как именно компьютеры хранят или передают электронные письма;
- программа Adobe Photoshop позволяет редактировать изображения, но от пользователя не требуется разбираться, каким образом компьютеры манипулируют изображениями.

С помощью программы мы каждый раз выполняем одни и те же задачи, но при этом манипулируем разными данными:

- программа электронной почты может использоваться для создания, отправки, получения и хранения электронных писем. Но содержание и получатели каждого электронного письма могут быть разными;
- программа Adobe Photoshop выполняет такие задачи, как добавление фильтра, изменение размера или обрезка изображения. Программа может выполнять задачи с любым изображением.

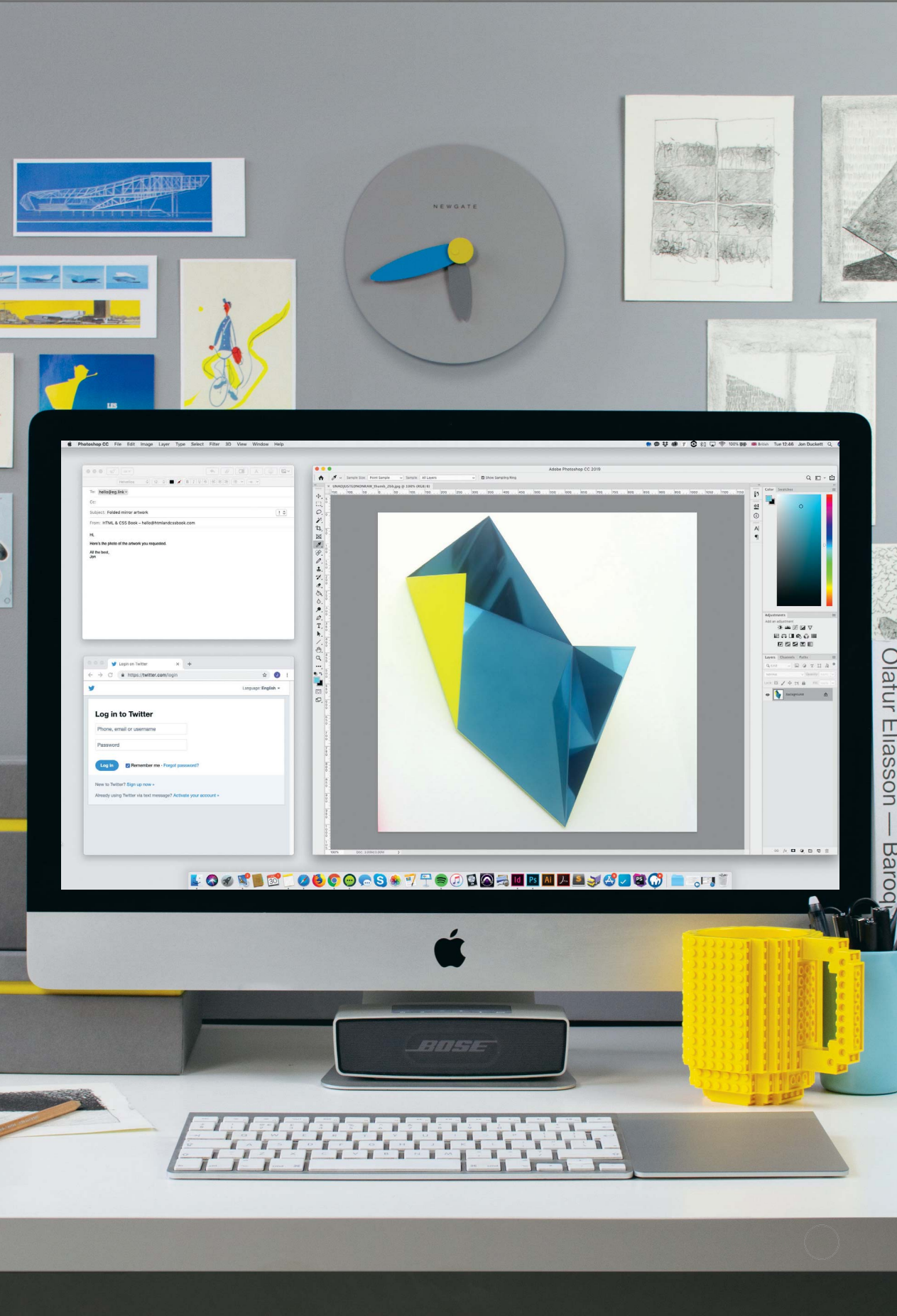
Обе эти программы имеют графический пользовательский интерфейс, с которым вы взаимодействуете для выполнения каких-либо задач.

Интерпретатор PHP — это тоже программа, которая работает на веб-сервере. Однако вы сообщаете интерпретатору PHP задачи, которые ему необходимо выполнить, не с использованием графического пользовательского интерфейса, а с помощью кода, написанного на языке PHP.

Когда вы создаете веб-страницу, используя PHP, эта страница всегда будет выполнять одни и те же задачи. Однако она может выполнять их, используя разные данные при каждом запросе страницы. Например, сайт, написанный на PHP, может содержать следующие страницы:

- одна и та же страница входа в систему, которую каждый пользователь использует для входа, несмотря на то что адрес электронной почты и пароль каждого пользователя уникальны;
- одна и та же страница учетной записи пользователя, на которой каждый пользователь видит детали своего личного кабинета. Даже если этой страницей одновременно пользуются сотни разных пользователей, они будут видеть только собственные данные.

Это становится возможным благодаря тому, что программа использует одни и те же инструкции при выполнении этих задач, но данные пользователей будут каждый раз разные.


NEWGATE
Olafur Eliasson — Baroq

ВЫПОЛНЕНИЕ ОДНОЙ И ТОЙ ЖЕ ЗАДАЧИ С РАЗНЫМИ ДАННЫМИ

Языки программирования позволяют создавать правила, сообщающие компьютеру, как выполнять ту или иную задачу. При этом данные, которые использует программа при каждом выполнении этой задачи, могут быть разными.

Работая с каким-либо языком программирования, вы указываете компьютеру точные инструкции для выполнения задачи. Эти инструкции значительно отличаются от тех, которые вы даете человеку, когда просите выполнить какое-либо задание.

Представьте, что вам необходимо купить пять плиток шоколада и вычислить их общую стоимость. Для этого необходимо умножить цену одной плитки на необходимое количество. Это правило можно записать следующим образом:

```
total = price *  quantity
// итого = цена * количество
```

Теперь на основе этой формулы можно рассчитать конечную стоимость покупки.

- Если шоколад стоит $1, а вы покупаете 5 плиток, общая сумма будет составлять $5.
- Если цена одной плитки составляла $1,50, правило остается тем же, но общая сумма будет равна $7,50.
- Если вы хотите купить 10 плиток по $2 каждая, правило остается тем же, но общая сумма будет равна $20.

Значения, используемые вместо слов `total` (итого), `price` (цена) и `quantity` (количество), могут изменяться, но правило, используемое для расчета общей стоимости шоколадных плиток, остается прежним.

При использовании PHP для создания веб-страницы вам сначала необходимо определить следующее:

- какую задачу необходимо выполнить;
- какие данные могут изменяться при выполнении этой задачи.

Затем вы сообщаете интерпретатору PHP подробные инструкции, как выполнить задачу и как назвать те данные, которые будут изменяться.
Если вы определите следующие данные:

```
price = 3 // цена
quantity = 5 // количество
```

а затем воспользуетесь следующим правилом:

```
total = price * quantity
// итого = цена * количество
```

то значение `total` (итого) будет равно 15.
В следующий раз при выполнении того же кода вы можете задать для цены или количества другие значения, и программа, используя то же правило, сможет вычислить новую сумму.

Слова, которые обозначают изменяющиеся значения, программисты называют **переменными**, так как эти значения могут изменяться каждый раз при запуске программы.

total = price x quantity

// итого = цена * количество



$9 = 3 x 3

ЧТО ПРЕДСТАВЛЯЕТ СОБОЙ СТРАНИЦА НА PHP?

Страница PHP в простейшем случае[1] представляет собой смесь кода HTML и PHP. Она используется для генерации HTML-кода в ответ на запрос браузера.

Ниже представлен пример страницы на PHP, содержащей смесь языков HTML и PHP.

- HTML- код обозначен синим цветом;
- код PHP выделен фиолетовым цветом.

Что делает интерпретатор PHP, когда открывает файл:

- отправляет любой HTML-код как есть;
- исполняет все команды PHP-кода, который обычно генерирует контент для HTML-страницы.

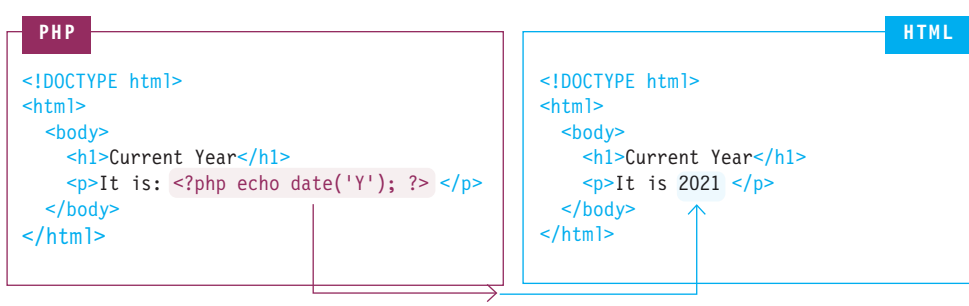
Представленный здесь PHP-код определяет, какой сейчас год, и выводит его между открывающим и закрывающим тегами.

PHP-код может выполнять как простейшие операции — арифметические вычисления или вычисление текущей даты, так и более сложные, например обновление сведений в базе данных на основе информации, отправленной через HTML-форму.

Когда интерпретатор PHP завершает обработку файла PHP, он отправляет сгенерированную на лету HTML-страницу в браузер.

Ниже представлен пример HTML-страницы, которая будет отправлена в браузер после выполнения интерпретатором PHP приведенного выше кода.

Интерпретатор PHP получает значение текущего года и отображает его на созданной им HTML-странице.



Интерпретатор PHP получает значение текущего года и выводит его внутри тегов абзаца.

Каждая страница обычно выполняет одну и ту же задачу, но способна отображать различную информацию при каждом запросе.

Веб-сайт, разработанный на PHP, состоит из набора страниц, каждая из которых выполняет определенную задачу. Например, сайт, который позволяет пользователям входить в систему, может содержать следующие страницы:

- страница входа в систему. С ее помощью пользователь заходит на сайт;
- страница профиля. Отображает личный кабинет пользователя.

Каждый раз, когда запрашивается одна из этих страниц, она должна уметь работать с данными каждого пользователя. Следовательно, ей потребуется:

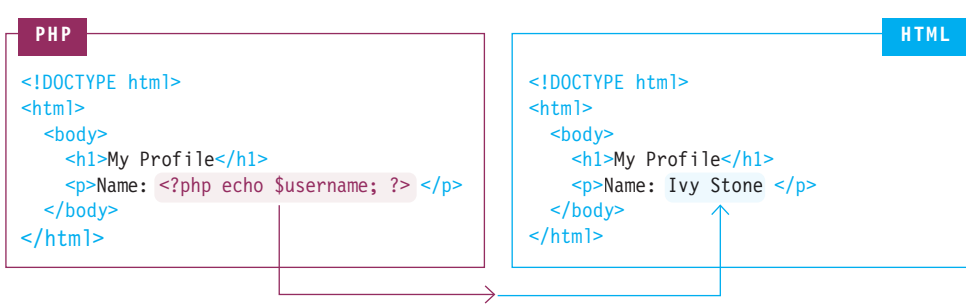
- содержать инструкции, как выполнить задачу, для которой она предназначена;
- определить именованные фрагменты данных, которые могут изменяться при каждом запросе.

В PHP каждая переменная имеет свое имя, которое описывает суть представляемого ей значения. При этом само значение может изменяться при каждом запросе страницы. Код PHP сообщает интерпретатору следующее:

- какое имя переменной использовать для фрагмента данных, который может изменяться при *каждом* запросе;
- какое значение использовать при обработке *данного* конкретного запроса.

После отправки HTML-страницы пользователю, интерпретатор PHP удалит все значения, хранящиеся в переменных, поэтому эта же задача может быть выполнена для следующего пользователя, запрашивающего страницу, но с другими значениями.

Для более продолжительного хранения данных их необходимо поместить в базу данных, например MySQL.



Интерпретатор PHP считывает значение переменной `$username` и выводит его внутри тегов абзаца.

ЗНАКОМСТВО С MYSQL

MySQL — это система управления базами данных (СУБД). Базы данных (БД) хранят информацию в структурированном виде, поэтому вы можете легко прочитать или обновить данные, которые в ней хранятся.

Программа для работы с электронными таблицами, например Excel, хранит информацию в виде таблицы, состоящей из столбцов и строк. Затем она может использовать хранящиеся в электронной таблице данные для вычислений или манипулировать ими с помощью формул.

MySQL — это программа, которая хранит информацию примерно таким же образом: в виде **таблиц**, которые точно так же состоят из строк и колонок. При этом вы можете использовать PHP для получения информации из базы данных, а также для ее обновления.

Одна база данных может содержать несколько таблиц. Каждая таблица обычно содержит данные одного типа. Ниже представлены два примера таблиц БД, которые содержат:

- данные пользователей, зарегистрированных на сайте;
- отображаемые на сайте публикации.

В каждой таблице **имена столбцов** описывают тип информации, которую содержит каждый столбец таблицы:

- таблица `member` (пользователь) содержит столбцы, в которых для каждого пользователя хранится имя, фамилия, адрес электронной почты, пароль, дата регистрации и изображение профиля;
- таблица `article` (публикация) содержит столбцы, в которых для каждой публикации хранятся заголовок, краткое содержание, полный текст, дата создания и некоторые другие данные.

Каждая **строка** содержит данные, описывающие одну из тех сущностей, для хранения которых была создана таблица:

- в таблице `member` каждая строка представляет одного пользователя;
- в таблице `article` каждая строка содержит одну публикацию.

ИМЯ ТАБЛИЦЫ — ИМЯ СТОЛБЦА — СТОЛБЕЦ

member

id	forename	surname	email	password	joined	picture
1	Ivy	Stone	ivy@eg.link	$2y$10$MAdTTCAOMiOw	2021-01-01 20:28:47	ivy.jpg
2	Luke	Wood	luke@eg.link	$2y$10$NN5HEAD3atar	2021-01-02 09:17:21	NULL
3	Emiko	Ito	emi@eg.link	$2y$10$/RpRmiUMStji	2021-01-02 10:42:36	emi.jpg

article

id	title	summary	content	created	category_id	member_id	image_id	published
1	Systemic	Brochure	<p>This	2021-01-01	1	2	1	1
2	Polite	Poster	<p>These	2021-01-02	1	1	2	1
3	Swimming	Architect	<p>This	2021-01-02	4	1	3	1

СТРОКА

Используя PHP, вы научитесь:

- **извлекать данные из базы данных** и отображать эту информацию на веб-странице;
- **добавлять в нее новые строки**. Чтобы создать новую публикацию, необходимо добавить строку в таблицу `article` и внести в нее данные для каждого столбца;
- **удалять строки**. Чтобы удалить публикацию, необходимо удалить всю строку, представляющую публикацию;
- **изменять данные в существующей строке**. Чтобы обновить адрес электронной почты пользователя, необходимо найти нужную строку в таблице `member` (пользователь), а затем обновить значение в столбце `email` (электронная почта) этой строки.

Обратите внимание, что в обеих таблицах первый столбец — `id` (идентификатор). Каждая строка таблицы содержит в этом столбце уникальное значение (в данном примере значения в столбцах начинаются с 1 и увеличиваются на 1 для каждой строки). Значения в столбце `id` позволяют указать базе данных, с какой строкой данных вам необходимо работать. Например, вы можете получить пользователя с идентификатором 2 или публикацию с идентификатором 1.

MySQL — **реляционная база данных**, так как она позволяет устанавливать связи между данными, хранящимися в разных таблицах.

Например, публикации в таблицах ниже написаны разными пользователями сайта. В таблице `article` значение в столбце `member_id` указывает, какой пользователь написал публикацию. Столбец `member_id` содержит число, соответствующее одному из значений в столбце `id` таблицы пользователей.

Первую публикацию разместил пользователь, у которого в столбце `id` стоит 2 (Luke Wood). Вторая и третья публикации размещены пользователем, у которого в столбце `id` стоит 1 (Ivy Stone).

Эти взаимосвязи:

- структурируют данные так, чтобы каждая таблица содержала информацию только об одном конкретном типе данных (пользователь или публикация);
- помогают избежать дублирования одних и тех же данных в нескольких таблицах (экономия места в базе данных);
- упрощают обновление данных. Если пользователь меняет свое имя, его необходимо обновить только в таблице `member`, а не в каждой размещенной им публикации.

member

id	forename	surname	email	password	joined	picture
1	Ivy	Stone	ivy@eg.link	$2y$10$MAdTTCAOMiOw	2021-01-01 20:28:47	ivy.jpg
2	Luke	Wood	luke@eg.link	$2y$10$NN5HEAD3atar	2021-01-02 09:17:21	NULL
3	Emiko	Ito	emi@eg.link	$2y$10$/RpRmiUMStji	2021-01-02 10:42:36	emi.jpg

article

id	title	summary	content	created	category_id	member_id	image_id	published
1	Systemic	Brochure	<p>This	2021-01-01	1	2	1	1
2	Polite	Poster	<p>These	2021-01-02	1	1	2	1
3	Swimming	Architect	<p>This	2021-01-02	4	1	3	1

ИСТОРИЯ РАЗВИТИЯ PHP

За время существования PHP и MySQL было выпущено множество версий этих программ. Новые версии работают быстрее, и в них добавляются новые функции.

Язык программирования PHP создан в 1994 году Расмусом Лердорфом (Rasmus Lerdorf). Затем в 1995 году он сделал код общественным достоянием, чтобы кто угодно мог использовать и улучшать его. В то время аббревиатура PHP обозначала Personal Home Page («Персональная домашняя страница»). С тех пор она поменяла значение и сейчас читается как PHP: Hypertext Processor (PHP: препроцессор гипертекста).

В настоящее время PHP применяется на 80% динамических веб-сайтов.

Facebook*, Etsy, Flickr и Wikipedia изначально были разработаны на PHP. Однако в настоящее время некоторые могли перейти на использование других технологий.

На PHP написаны такие популярные программные продукты с открытым исходным кодом, как WordPress (на котором работает более 35% веб-сайтов в мире), Drupal, Joomla и Magento. Изучение PHP поможет вам освоить их.

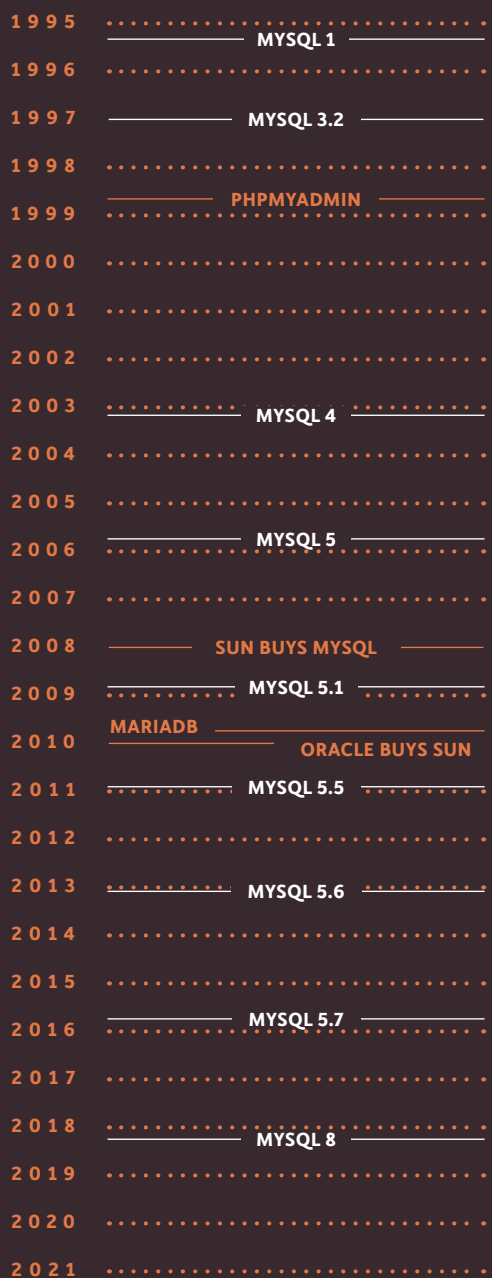
По мере выхода каждой новой версии PHP добавляются дополнительные функции. В этой книге представлены функции до версии PHP 8 включительно, выпущенной в ноябре 2020 года.

Поскольку хостинг-провайдеры не всегда используют последнюю версию PHP, пример сайта во второй части книги разработан для работы на серверах, на которых работает PHP версии 7.3 и не поддерживаются новейшие функции PHP 8.

	1995
PHP 1	
	1996
	1997
PHP 2	1998
PHP 3	
	1999
	2000
PHP 4	
	2001
	2002
	2003
	2004
PHP 5	
	2005
PHP 5.1	2006
PHP 5.2	2007
	2008
	2009
PHP 5.3	
	2010
	2011
PHP 5.4	2012
	2013
PHP 5.5	
	2014
PHP 5.6	
	2015
PHP 7	2016
PHP 7.1	2017
PHP 7.2	2018
PHP 7.3	2019
PHP 7.4	2020
PHP 8	2021

* Принадлежит компании Meta, которая признана экстремистской и запрещена на территории РФ.

ИСТОРИЯ РАЗВИТИЯ MYSQL



Система управления базами данных MySQL впервые выпущена в 1995 году. Аббревиатура SQL произносится как ess-queue-el или sequel и означает Structured Query Language — язык структурированных запросов. SQL — это язык, используемый в качестве эффективного способа получения информации в реляционных базах данных и для её обновления.

СУБД MySQL разработана шведской компанией MySQL AB, которая предоставила бесплатный доступ к MySQL. Микаэль Видениус (Michael Widenius), один из создателей MySQL, назвал MySQL в честь своей дочери Мю (My).

Затем в январе 2008 года компания MySQL AB была продана Sun Microsystems, а в 2010 году компания Oracle приобрела Sun Microsystems.

Когда разработчики MySQL узнали, что Oracle собирается приобретать Sun Microsystems, а значит, получит права на MySQL, они забеспокоились, что MySQL не останется бесплатной, поэтому была создана версия базы данных с открытым исходным кодом MariaDB, названная в честь младшей дочери Микаэля Видениуса Марии (Maria).

Facebook*, YouTube, Twitter, Neftlix, Spotify и Wordpress используют СУБД MySQL или MariaDB.

phpMyAdmin — это бесплатное веб-приложение, которое используется для управления базами данных MySQL и MariaDB. Оно выпущено в 1998 году.

Приведенный в книге программный код работает с MySQL или MariaDB версии 5.5 и выше. Для работы с этими СУБД используется веб-инструмент phpMyAdmin.

Последняя версия MySQL — 8. MySQL 6 никогда не выпускалась, а версия 7 не показана, так как была предназначена для работы на кластерах серверов, а не для персонального компьютера.

ЧТО ВЫ УЗНАЕТЕ ИЗ ЭТОЙ КНИГИ

Эта книга состоит из четырех разделов.
Ниже представлен обзор основных тем каждого раздела.

А: ОСНОВЫ ПРОГРАММИРОВАНИЯ

В первом разделе этой книги вы познакомитесь с основами написания программного кода на PHP. Вы узнаете:

- основные принципы программирования;
- как выполнять тот или иной код в зависимости от ситуации. Например, один код запускается, если пользователь авторизован на сайте, а другой — если нет;
- как с помощью функций группировать код, отвечающий за какую-то одну операцию;
- как с помощью классов и объектов описывать в коде понятия и явления из реального мира.

Б: ДИНАМИЧЕСКИЕ ВЕБ-СТРАНИЦЫ

Во втором разделе этой книги представлен набор инструментов, позволяющий создавать динамические веб-страницы. Вы научитесь:

- получать данные, отправленные из браузера;
- проверять формат предоставленных пользователем данных;
- работать с любыми отправленными данными;
- работать с загруженными пользователями файлами;
- манипулировать датой и временем в PHP;
- временно хранить данные в файлах cookie и сессиях;
- находить и устранять ошибки в коде.

В: САЙТЫ НА ОСНОВЕ БАЗ ДАННЫХ

В третьем разделе описаны способы получения информации из базы данных и ее отображения на веб-страницах, а также способы обновления информации в базе данных. Вы изучите следующие темы:

- хранение информации в базе данных;
- использование SQL для получения или обновления информации, хранящейся в базе данных;
- отображение информации из базы данных на странице;
- использование HTML-форм для обновления пользователями информации, хранящейся в базе данных.

Г: РАСШИРЕНИЕ ФУНКЦИОНАЛА И МОДЕРНИЗАЦИЯ УЧЕБНОГО ПРИЛОЖЕНИЯ

В четвертом разделе продемонстрированы практические приемы создания веб-сайтов и веб-приложений на PHP. В качестве учебного приложения вы создадите простую систему управления контентом с функциями социальной сети. Вы научитесь:

- улучшать структуру кода;
- использовать код, разработанный другими программистами;
- отправлять электронные письма с помощью PHP;
- разрешать пользователям регистрироваться и входить на сайт;
- создавать страницы, адаптированные для отдельных пользователей;
- добавлять социальные функции, например лайки и комментарии;
- создавать более удобные (как для человека, так и для поисковых систем) URL-адреса.

A

ОСНОВЫ ПРОГРАММИРОВАНИЯ

В первом разделе книги вы познакомитесь с основами написания программного кода на PHP.

Программирование — это процесс написания инструкций, следуя которым, компьютер может выполнить ту или иную задачу. Этот процесс можно сравнить с пошаговым кулинарным рецептом, которого вы придерживаетесь для приготовления блюда. Каждая отдельная команда в PHP называется **инструкцией (statement)**[2].

Поскольку PHP был разработан для создания веб-сайтов, которые динамически генерируют HTML-страницы по запросу, то команды и операторы, которые вы изучите в первом разделе книги, сконцентрированы на использовании PHP для создания HTML-страниц.

Готовый веб-сайт, как правило, состоит из тысяч строк программного кода, поэтому важно очень внимательно отнестись к структурированию кода, разбиению его на модули. В этом разделе будут описаны две концепции, которые позволяют группировать инструкции по смыслу.

- **Функции**. Группируют инструкции, необходимые для выполнения определенной задачи.
- **Объекты**. Объединяют код и данные, которые относятся к какому-то одному явлению. Например, это могут быть статьи, товары на сайте или зарегистрированные пользователи.

Темы этого раздела — основа всей дальнейшей информации книги.

Прежде чем углубиться в первую главу, необходимо освоить несколько базовых понятий, которые помогут вам в дальнейшем обучении.

УСТАНОВКА НЕОБХОДИМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И ПРИМЕРОВ КОДА

Для создания сайтов с использованием PHP и MySQL на вашем стационарном компьютере или ноутбуке необходимо установить программное обеспечение. После установки необходимо загрузить примеры кода для книги, которые вы найдете на нашем сайте http://addons.eksmo.ru/it/phpbook.zip.

PHP И HTML В ОДНОМ ФАЙЛЕ

На начальном этапе обучения PHP-файлы будут представлять собой смесь языков HTML и PHP. Поэтому необходимо знать, как интерпретатор PHP определяет разницу между кодами HTML и PHP.

ВЫВОД ИНФОРМАЦИИ ИЗ PHP В HTML

Одна из самых частых команд, которые вы будете использовать в PHP, это вывод определенной информации посреди HTML-кода. Вывод данных между тегов HTML с помощью PHP используется в каждом примере этого раздела.

ДОБАВЛЕНИЕ КОММЕНТАРИЕВ В PHP-КОД

Комментарии игнорируются интерпретатором PHP и нужны только для разработчика. Комментарии помогут вам (и другим) понять работу кода, поэтому важно научиться их добавлять. В этой книге комментарии добавлены практически по всему коду, чтобы помочь в изучении каждого примера.

УСТАНОВКА НЕОБХОДИМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И ФАЙЛОВ

С помощью приведенных ниже программных средств вы установите программное обеспечение, необходимое для создания веб-сайтов на основе баз данных на вашем стационарном компьютере или ноутбуке.

Для работы с книгой вам необходимо установить:

- **веб-сервер**, на котором работает интерпретатор PHP. В книге используется **Apache**;
- **MySQL** или **MariaDB** — программное обеспечение для базы данных;
- **phpMyAdmin** — веб-приложение для работы с базами данных.

Чтобы не загружать и не устанавливать каждую из этих программ по отдельности, ниже представлены программные средства, которые сделают все автоматически.

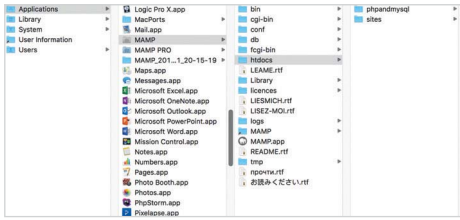
Мы также рекомендуем установить один из редакторов кода, приведенных на странице **http://notes.re/php/editors**.

УСТАНОВКА В MACOS

Пользователям Mac мы рекомендуем установить необходимое программное обеспечение с помощью локальной среды разработки MAMP. Ссылка для загрузки MAMP и инструкции по установке можно найти на сайте **http://notes.re/php/mamp**.

В процессе установки MAMP на Mac (с настройками по умолчанию) создается папка `/Applications/MAMP`.

Внутри имеется еще одна папка — `htdocs`. Все веб-страницы, которые создаются с помощью PHP, должны находиться в этой папке. Папка `htdocs` — это **корневой каталог** вашего сайта (document root).

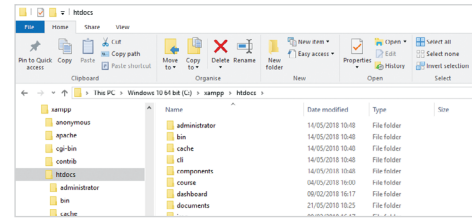


УСТАНОВКА В WINDOWS/LINUX

Пользователям компьютеров под управлением Windows или Linux мы рекомендуем установить необходимое программное обеспечение с помощью программы XAMPP. Ссылку для загрузки и инструкции по установке можно найти на сайте `http://notes.re/php/xampp`.

В процессе установки XAMPP на компьютер (с настройками по умолчанию) создается папка `c:\xampp\`.

Внутри имеется еще одна папка — `htdocs`. Все веб-страницы, которые создаются с помощью PHP, должны находиться в этой папке. Папка `htdocs` — это **корневой каталог** вашего сайта (document root).



ЗАГРУЗКА ПРИМЕРОВ КОДА

Примеры кода можно загрузить с сайта http://addons.eksmo.ru/it/phpbook.zip.

Все приводимые в книге примеры кода находятся в папке `phpbook`. Она содержит подкаталоги для каждого раздела книги, а также для каждой отдельной главы. Папку `phpbook` необходимо поместить в папку `htdocs` вашего веб-сервера.

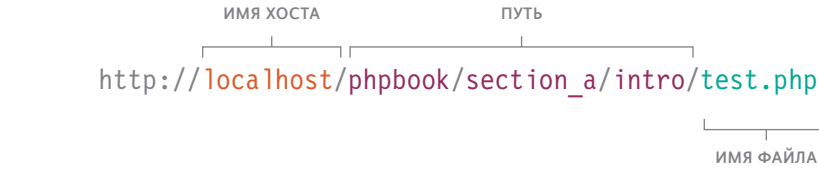
Для выполнения PHP-файлов вам необходимо в адресную строку браузера ввести URL-адрес. PHP-код **не запустится**, если вы откроете файл с помощью команды **File (Файл) → Open (Открыть)**, дважды кликнете по нему или перетащите его в браузер.

Также папка с примерами снабжена удобным оглавлением, с помощью которого легко можно найти нужный пример. Оно находится в коневой папке с загруженными файлами. Далее по ссылкам можно найти нужный раздел, главу и конкретный пример.

Читать эту книгу удобнее всего, открыв на компьютере браузер, в котором загружена страница из архива с примерами, таким образом сразу можно будет выполнить каждый пример, приведенный в книге. Кроме того, рекомендуется открывать примеры кода в редакторе, поскольку в книге они часто приводятся в сокращенном виде.

Попробуйте открыть приведенный ниже URL, и вы увидите тестовую страницу. В адресной строке своего браузера в качестве адреса сайта введите `localhost`. Далее следует путь от папки `htdocs` к файлу, который необходимо открыть.

Указанный ниже путь дает команду серверу искать в папке `phpbook/section_a/intro` файл с именем `test.php`.





УСТРАНЕНИЕ ОШИБОК

Если вы не видите страницу, расположенную слева, проверьте следующее:

- `http://notes.re/php/mamp` для MAMP;
- `http://notes.re/php/xampp` для XAMPP.

При установке с помощью MAMP иногда требуется указать **номер порта**. MAMP использует порт 8888, поэтому в браузере необходимо написать `http://localhost:8888/`.

Номера портов помогают различным установленным программам использовать один и тот же интернет-адрес аналогично тому, как в офисах установлен один телефонный номер, за которым следуют добавочные цифры для связи с конкретным сотрудником.

КАК PHP ВСТРАИВАЕТСЯ В HTML

Исходно язык PHP был разработан для того, чтобы встраиваться в код HTML. Поэтому код PHP всегда пишется между открывающим и закрывающим тегами PHP. **Блок PHP** — это все, что находится между тегами PHP.

ОТКРЫВАЮЩИЙ ТЕГ

<?php

Открывающий тег указывает, что интерпретатор PHP должен начать обработку кода перед отправкой любого содержимого в браузер.

ЗАКРЫВАЮЩИЙ ТЕГ

?>

Закрывающий тег указывает, что интерпретатор PHP прекращает обработку кода, пока не найдет другой открывающий тег `<?php`.

PHP — это скриптовый язык программирования, или, иначе, — язык сценариев. **Языки сценариев** предназначены для работы в определенной среде. Такой средой для PHP является интерпретатор PHP. Страницу, созданную на PHP, часто называют **скриптом**.

Ко всему PHP-коду необходимо относиться так, как будто он чувствителен к регистру (то есть не менять регистр букв произвольно).

Даже если не все элементы языка чувствительны к регистру, то использование этого правила уменьшает вероятность ошибок.

Страница на PHP — это текстовый файл (как, например, и HTML-страница). У PHP-файла должно быть расширение. php, которое указывает веб-серверу на то, что файл должен быть отправлен на обработку интерпретатору PHP.

Ниже продемонстрирована страница, которая создана на PHP и содержит следующее:

- **PHP-код между тегами PHP обозначен фиолетовым цветом**. Интерпретатор PHP обрабатывает любой код, написанный внутри тегов PHP;
- **HTML-код за пределами тегов PHP обозначен белым цветом**. Он автоматически отправляется в браузер (поскольку интерпретатор PHP его игнорирует).

Код внутри тегов PHP разбивается на **инструкции**. Большинство инструкций начинаются с новой строки и заканчиваются точкой с запятой. В следующих случаях вы можете опустить точку с запятой в конце:

- в последней строке блока PHP;
- если блок PHP содержит только одну инструкцию.

Добавление точки с запятой в конце каждой инструкции помогает избежать ошибок.

В следующем примере рассчитывается стоимость 5 упаковок конфет ценой 3 доллара каждая. Итоговая стоимость сохраняется в переменной $total. Затем это значение выводится на HTML-странице. О том, как PHP это делает, вы узнаете в главе 1.

```
<?php
  $price    = 3;
  $quantity = 5;
  $total    = $price * $quantity;
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Cost of Candy</title>
  </head>
  <body>
    <p>Total: $ <?php echo $total ?></p>
  </body>
</html>
```

ОТКРЫВАЮЩИЙ ТЕГ PHP
ИНСТРУКЦИЯ
ИНСТРУКЦИЯ
ИНСТРУКЦИЯ
ЗАКРЫВАЮЩИЙ ТЕГ PHP
БЛОК PHP
БЛОК PHP

КАК PHP ВЫВОДИТ ТЕКСТ И HTML-КОД

Команда echo говорит интерпретатору PHP вывести текст или разметку в браузер.

Любой текст (включая разметку HTML), который пишется в инструкции после команды echo, будет выведен на странице. Текст в PHP-коде должен заключаться в кавычки. Вы можете использовать одинарные или двойные, при этом открывающая и закрывающая кавычки должны **совпадать**.

Первая кавычка указывает интерпретатору PHP начало текста, а вторая — его конец. Текст, заключенный в кавычки, называется **строковым литералом**. Точка с запятой в конце строки указывает интерпретатору PHP на конец инструкции.

```
echo '<b>Hello!</b>';
```

ВЫВОД В БРАУЗЕР — ТЕКСТ И РАЗМЕТКА ДЛЯ ОТОБРАЖЕНИЯ

Чтобы добавить кавычку внутри текста, непосредственно перед кавычкой добавьте обратную косую черту. Обратная косая черта указывает интерпретатору PHP не воспринимать следующую за ней кавычку как ограничитель. Программисты называют это **экранированием кавычек**.

В примере ниже для написания HTML-ссылки используются двойные кавычки, поскольку значения атрибутов всегда должны заключаться в кавычки. Таким образом с помощью экранирования можно вывести следующий HTML-код:

```
echo "<a href=\"http://notes.re/php\">PHP</a>";
```

ОТКРЫВАЮЩАЯ КАВЫЧКА В КОМАНДЕ ECHO — ЭКРАНИРОВАННЫЕ КАВЫЧКИ ОБРАМЛЯЮТ ЗНАЧЕНИЕ АТРИБУТА — ЗАКРЫВАЮЩАЯ КАВЫЧКА В КОМАНДЕ ECHO

Вы также можете отображать двойные кавычки, помещая любой текст в одинарные кавычки.

Это работает, так как интерпретатор PHP ищет подходящую одинарную кавычку для указания конца текста.

```
echo '<a href="http://notes.re/php">PHP</a>';
```

ОТКРЫВАЮЩАЯ КАВЫЧКА — HTML-АТРИБУТ В ДВОЙНЫХ КАВЫЧКАХ — ЗАКРЫВАЮЩАЯ КАВЫЧКА

ВЫВОД КОНТЕНТА НА СТРАНИЦУ

PHP section_a/intro/echo.php

```php
<!DOCTYPE html>
<html>
  <head>
    <title>echo Command</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
    <h2><?php echo 'Ivy\'s'; ?> page</h2>
    <?php echo '<p class="offer">Offer: 20% off</p>' ?>
  </body>
</html>
```

РЕЗУЛЬТАТ



Примечание. Если вы используете двойные кавычки, интерпретатор PHP проверяет, содержит ли текст имена переменных. Если это так, интерпретатор PHP подставит в это место значение, которое содержит переменная. С одинарными кавычками это не сработает.

Упражнение. В шаге 1 измените имя `Ivy` на свое и сохраните файл. При обновлении страницы в приветствии будет отображаться ваше имя.

Рассмотрим структуру приведенного примера, поскольку все примеры в книге будут приводиться по такой же схеме.

- Путь к файлу соответствует файлу в загружаемом коде. Соответственно, вы можете открыть его в браузере, как было показано выше, и выполнить.
- Цифры соответствуют действиям, описывающим приведенный ниже код.

1. В команде `echo` в имени пользователя используются одинарные кавычки, которые обрамляют имя пользователя, за которым следуют символы `'s`. Символ обратной косой черты используется для экранирования символа `'` между именем и буквой `s`.
2. Команда `echo` выводит HTML-тег абзаца на страницу. Элемент `<p>` содержит атрибут `class`.

Поскольку текст и разметка помещаются в одинарные кавычки, HTML-атрибут может заключаться в двойные кавычки.

Хотя вы можете использовать одинарные или двойные кавычки по желанию, рекомендуется выбрать один вариант и придерживаться его. В книге в основном используются одинарные кавычки, чтобы значения атрибутов HTML можно было заключать в двойные, так же как показано в этом примере.

КОММЕНТАРИИ

В код PHP рекомендуется добавлять комментарии. Они помогут вам вспомнить, что делает ваш код, когда вы вернетесь к нему после некоторого перерыва, а также поможет другим программистам понять его.

Однострочные комментарии начинаются со следующих символов:

- две косые черты //
- символ решетки #

При интерпретации кода комментарий игнорируется (при этом нужно быть внимательным: закрывающий тег PHP прекращает действие однострочного комментария. То есть если внутри комментария встретятся символы ?>, то весь текст после них будет выведен в браузер.

```
echo «Welcome»; // Отображение приветствия
echo «Welcome»; # Отображение приветствия
```

ОДНОСТРОЧНЫЙ КОММЕНТАРИЙ

Многострочные комментарии — это комментарии, которые состоят из нескольких строк. С их помощью можно добавлять более подробные пояснения к коду.

Многострочный комментарий начинается символами /* и продолжается до тех пор, пока не встретит закрывающие символы */.

```
echo «Welcome»;
/*
После приветственного сообщения необходимо:
- добавить изображение профиля рядом
с именем пользователя
- добавить ссылку на страницу профиля
пользователя
*/
```

МНОГОСТРОЧНЫЙ КОММЕНТАРИЙ

ДОБАВЛЕНИЕ КОММЕНТАРИЕВ К ВАШЕМУ КОДУ

PHP section_a/intro/comments.php

```php
<?php
  /*
  На странице отображается имя пользователя
  и информация о текущем предложении
  */
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Adding Comments to Your Code</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
    <h2><?php echo 'Welcome Ivy'; // Отображение имени
?></h2>
    <?php echo '<p class="offer">Offer: 20% off</p>' ?>
  </body>
</html>
```

РЕЗУЛЬТАТ



Следующий пример аналогичен предыдущему, но в него добавлены комментарии.

1. Код начинается с многострочного комментария, описывающего работу кода.

2. Однострочный комментарий поясняет, что именно отображается на странице. Но при этом его действие заканчивается с закрывающим тегом PHP, а закрывающий тег HTML выводится на страницу.

Комментарии не добавляются в отправляемый браузеру код HTML. Они видны только в коде PHP.

Упражнение. В шаге 1 добавьте в комментарий еще одну текстовую строку.

Упражнение. В шаге 2 замените символы // на символ #.

Примечание. Для улучшения читабельности и понимания работы программного кода в книге используется большое количество комментариев. Однако опытные программисты пишут комментарии не так часто, описывая только сложные и неочевидные участки кода.

ПЕРЕЧЕНЬ ГЛАВ РАЗДЕЛА А
ОСНОВЫ ПРОГРАММИРОВАНИЯ

1

ПЕРЕМЕННЫЕ, ВЫРАЖЕНИЯ И ОПЕРАТОРЫ

Каждый раз, когда PHP-скрипт запускается для выполнения какой-либо задачи, данные, с которыми он работает, могут принимать разные значения. Поэтому важно научиться работать с этими данными в коде с помощью **переменных**. Вы также научитесь использовать **выражения** и **операторы** для работы с этими значениями.

2

УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ

PHP-скрипт не всегда выполняет одни и те же команды в одном и том же порядке. **Управляющие конструкции** позволяют создавать правила, которые интерпретатор PHP использует для определения следующего шага при выполнении кода.

3

ФУНКЦИИ

Отдельные команды можно объединять с помощью **функций**. Они не только позволяют сгруппировать вместе инструкции, выполняющие какую-то одну задачу, но и помогают избежать повторения кода при многократном выполнении одних и тех же действий.

4

КЛАССЫ И ОБЪЕКТЫ

В коде могут быть представлены такие сущности, как пользователи сайта, товары для продажи или статьи. Чтобы объединить код и данные, представляющие каждое из этих различных понятий, программисты используют **классы** и **объекты**.

1

ПЕРЕМЕННЫЕ, ВЫРАЖЕНИЯ И ОПЕРАТОРЫ

В этой главе показано, как с помощью переменных можно использовать данные, которые могут принимать различные значения при каждом запросе страницы PHP, а также как работать с этими данными с помощью выражений и операторов.

В PHP переменные используются для хранения отдельных значений, которые могут изменяться во время работы.

- **Имя** переменной описывает суть данных, которые она содержит.
- **Значение** — это сами данные, которые переменная содержит при данном конкретном запросе.

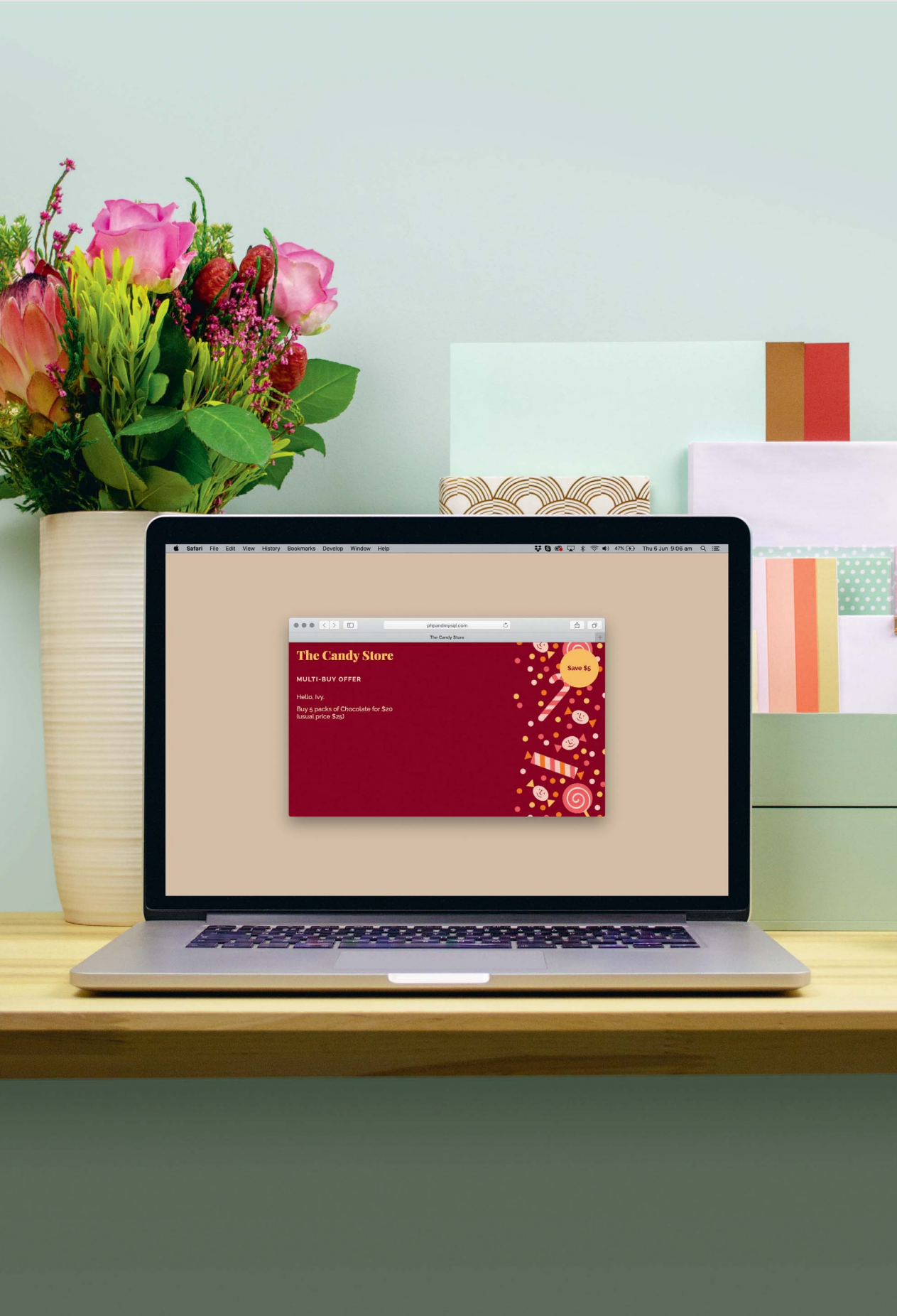
После завершения работы со страницей интерпретатор PHP «забывает» все переменные. При следующем обращении к странице переменная может получить другое значение.

PHP умеет работать со значениями различных типов (такими как текст или числа), то есть с различными **типами данных**.

- Строка или тип `string` — это набор символов.
- Тип данных `integer` представляет собой целочисленные значения.
- Тип данных `float` представляет собой числа с плавающей запятой.
- Логические значения (тип данных `boolean`) принимают только значение `true` или `false`.
- Именованные и индексированные коллекции можно хранить в массиве (`array`).

Изучив переменные, вы с легкостью научитесь использовать **выражения** для создания новых значений. Например, текст, содержащийся в двух переменных, может быть объединен в одно предложение. Или число, хранящееся в одной переменной, может быть умножено на число, хранящееся в другой переменной.

Выражения используют **операторы** для получения результата. Например, оператор «+» используется для сложения двух значений, а оператор «-» используется для вычитания одного значения из другого.


Safari File Edit View History Bookmarks Develop Window Help
Thu 6 Jun 9:06 am
phpandmysql.com
The Candy Store
The Candy Store
MULTI-BUY OFFER
Hello, Ivy.
Buy 5 packs of Chocolate for $20
(usual price $25)
Save $5

ПЕРЕМЕННЫЕ

Все данные хранятся в памяти в виде переменных. **Имя переменной** используется для доступа к этим данным (**значению** переменной). Переменные могут менять свое значение.

Для того чтобы создать переменную и присвоить ей значение, вам понадобится следующее.

- **Имя переменной** должно начинаться со знака $, за которым следует словесное описание данных, хранящихся в переменной.
- Присвоение значения происходит с помощью **знака равенства**, который также называется **оператором присваивания**.
- После знака равенства пишется **значение**, которое необходимо присвоить переменной.

Если переменной надо присвоить текстовое значение, то его всегда заключают в кавычки. Можно использовать одинарные или двойные, но надо следить, чтобы они всегда совпадали. То есть нельзя начинать текст одинарной кавычкой, а завершать — двойной, это вызовет ошибку.

Числовые и логические значения (`true` или `false`) в кавычки не заключаются.

Создание переменной программисты также называют **объявлением** переменной. Когда переменная получает какое-то значение, это называется **присваиванием**.



После того как переменная была объявлена и ей было присвоено значение, имя переменной можно использовать в программном коде везде, где необходимо работать со значением, которое хранит данная переменная.

Когда интерпретатор PHP находит имя переменной, он заменяет его значением, которое содержит эта переменная. В примере ниже команда `echo` используется для вывода значения переменной `$name`.



СОЗДАНИЕ ПЕРЕМЕННЫХ И РАБОТА С НИМИ

PHP section_a/c01/variables.php

```
<?php
(1) $name  = 'Ivy';
(2) $price = 5;
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Variables</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
(3)  <h2>Welcome <?php echo $name; ?></h2>
    <p>The cost of your candy is
(4)     $<?php echo $price; ?> per pack.</p>
  </body>
</html>
```

РЕЗУЛЬТАТ



В этой главе значения переменным присваиваются прямо в коде PHP. В следующих главах значения будут также присваиваться на основе данных, которые получены из отправленных пользователями HTML-форм, из URL-адресов и из базы данных.

В данном примере вверху страницы создаются две переменные и им присваиваются значения.

1. Переменная `$name` содержит имя посетителя сайта. Эта переменная содержит текст, а текст заключается в кавычки.
2. Переменная `$price` содержит цену одной упаковки конфет. Эта переменная содержит число, поэтому ее значение в кавычки не берется.

Далее идет HTML-код, который выводится в браузер.

3. Имя пользователя отображается на странице с помощью команды `echo`.
4. Также на странице отображается стоимость конфет.

Упражнение. В шаге 1 присвойте переменной `$name` свое имя. Сохраните файл, а затем обновите страницу в своем браузере. В результате на странице отобразится ваше имя.

Упражнение. В шаге 2 переменной `$price` присвойте значение 2. Сохраните файл и обновите страницу. В результате на странице отобразится новая цена.

КАК ПРАВИЛЬНО НАЗЫВАТЬ ПЕРЕМЕННЫЕ

Имя переменной должно максимально четко соответствовать хранимым в ней данным.
Разберемся, как правильно называть переменные. Необходимо учитывать следующие базовые правила.

1

Имя переменной должно начинаться со знака доллара ($).

✓ `$greeting`
✗ `greeting`

2

Сразу после знака $ должна идти либо буква, либо знак подчеркивания
(но не число).

✓ `$greeting`
✗ `$2_greeting`

3

Затем могут идти символы A-z (прописные и строчные), цифры и знаки подчеркивания в любых комбинациях. Имя переменной не должно содержать тире или точки.

✓ `$greeting_2`
✗ `$greeting-2`
✗ `$greeting.2`

Примечание. Переменная `$this` имеет особое значение. Не используйте `$this` в качестве имени переменной.

✗ `$this`

Использование имен переменных, максимально четко описывающих хранимые в них данные, значительно упрощает понимание кода и его логической структуры.

Если для описания данных необходимо использовать несколько слов, то, как правило, они отделяются знаком подчеркивания или сменой регистра.

Имена переменных чувствительны к регистру, поэтому переменные `$Score` и `$score` — это две разные переменные. Однако следует избегать создания двух переменных, имена которых представляют собой одно и то же слово в разных комбинациях прописных или строчных букв, так как это очень сильно ухудшит читабельность кода.

Также вы можете использовать не только латинские буквы, но и другие наборы символов (например, китайские или кириллические). Однако правилом хорошего тона считается использовать только буквы A-z, цифры и символы подчеркивания.

СКАЛЯРНЫЕ (ПРОСТЫЕ) ТИПЫ ДАННЫХ

PHP поддерживает три **скалярных типа данных** — текст, числа и логические (булевы) значения.

СТРОКОВЫЙ ТИП ДАННЫХ

Фрагмент текста программисты называют строкой (`string`). Данные строкового типа могут содержать буквы, цифры и любые другие символы.

```
$name = 'Ivy';
```

Строки заключаются в одинарные или двойные кавычки. Открывающая кавычка должна соответствовать закрывающей кавычке.

```
✓ $name = 'Ivy';
✓ $name = "Ivy";
✗ $name = "Ivy';
✗ $name = 'Ivy";
```

ЧИСЛОВЫЕ ТИПЫ ДАННЫХ

Числовые типы данных позволяют выполнять математические операции, например сложение или умножение.

```
$price = 5;
```

Числа в кавычки не заключаются. Если число поместить в кавычки, оно будет определено как строковый тип данных.

PHP поддерживает два числовых типа данных.

1. Тип данных `int` представляет собой целочисленные значения (например, 275).
2. Тип данных `float` представляет собой числа с плавающей запятой, или дроби (например, 2,75).

ТИП ДАННЫХ NULL

PHP также поддерживает тип данных `null`. Данные типа `null` могут иметь только одно значение — `null`. Это означает, что переменной вообще не было присвоено значение.

ЛОГИЧЕСКИЕ ТИПЫ ДАННЫХ

Тип данных `boolean` может иметь только одно из двух значений: `true` (истина) или `false` (ложь). Эти значения используются практически во всех языках программирования.

```
$logged_in = true;
```

Значения `true` и `false` традиционно пишутся в нижнем регистре и не должны заключаться в кавычки. Логические значения могут показаться абстрактными, однако многие вещи могут быть представлены с использованием значений `true` или `false`, например:

- авторизован ли пользователь?
- согласны ли пользователи с условиями?
- соответствует ли товар условиям бесплатной доставки?

ИЗМЕНЕНИЕ ЗНАЧЕНИЯ ПЕРЕМЕННОЙ

Вы можете изменить или перезаписать хранящееся в переменной значение, присвоив ей новое. Это выполняется аналогично тому, когда вы присваиваете значение переменной при ее создании.

Рассмотрим следующий пример.

1. Инициализация переменной `$name`. Это означает, что переменная объявлена и ей присвоено начальное значение, которое и будет использоваться, если не будет изменено далее в коде.

Начальное значение — `Guest`. Значение записано в кавычках, так как представляет собой текст.

2. Затем переменной `$name` присваивается новое значение `Ivy`.
3. Переменной `$price` присваивается цена одной упаковки конфет.

Затем идет HTML-код, который выводится в браузер.

4. Имя посетителя отображается на странице с помощью команды `echo`. Отображается обновленное значение, которое в шаге 2 было присвоено переменной `$name`.
5. Также на странице отображается стоимость конфет.

section_a/c01/updating-variables.php **PHP**

```
<?php
(1) $name  = 'Guest';
(2) $name  = 'Ivy';
(3) $price = 5;
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Updating Variables</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
(4)   <h1>The Candy Store</h1>
    <h2>Welcome <?php echo $name; ?></h2>
(5)   <p>The cost of your candy is
        $<?php echo $price; ?> per pack.</p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 2 поменяйте значение переменной `$name`, чтобы она теперь содержала ваше имя. Сохраните файл, а затем обновите страницу в своем браузере. В результате на странице отобразится ваше имя.

Упражнение. Добавьте еще одну строку и присвойте переменной `$name` еще чье-нибудь имя. Сохраните файл и обновите страницу. В результате на странице отобразится новое имя.

МАССИВЫ

Переменная также может содержать **массив**, представляющий собой упорядоченный набор элементов. Массивы относятся к **составному типу данных**, поскольку они могут хранить более одного значения.

Массив — это своеобразный контейнер, содержащий упорядоченный набор переменных. Каждое значение в массиве называется **элементом**. Аналогично переменным, каждый элемент в массиве имеет:

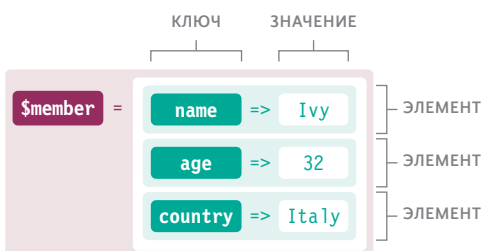
- **ключ** (`key`), который работает как имя переменной;
- **значение** (`value`) — данные, которые хранятся под этим индексом.

PHP поддерживает два типа массивов.

- **Ассоциативный массив** — это именованный список, где ключ описывает содержащееся значение.
- **Индексированный массив** — это массив с непрерывно возрастающими **числовыми индексами**.

АССОЦИАТИВНЫЙ МАССИВ

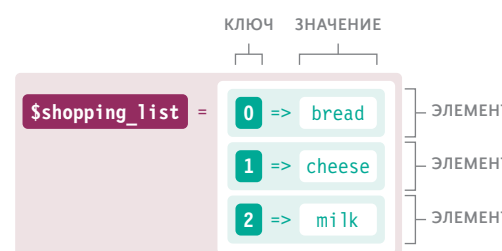
Приведенный ниже массив предназначен для хранения данных пользователя вебсайта. Обычно структура ассоциативного массива остается низменной.



В данных примерах значения, хранящиеся в массиве, имеют скалярный тип данных (единичные значения).

ИНДЕКСИРОВАННЫЙ МАССИВ

Приведенный в примере ниже массив предназначен для хранения списка покупок. Подобные списки при каждом использовании могут содержать разное количество элементов. Ключ — это целое число. Ключ может назначаться автоматически (всегда начинается с 0).



Примечание. Нумерация элементов массива начинается с 0. Первый элемент имеет индекс с номером 0, второй — 1 и так далее. Номер индекса обычно используется для определения порядка элементов в списке.

АССОЦИАТИВНЫЕ МАССИВЫ

Для создания ассоциативного массива необходимо каждому элементу массива присвоить **ключ**, описывающий хранимые в нем данные.

Для создания переменной, содержащей ассоциативный массив, необходимо сделать следующее:

- придумать имя переменной, описывающее набор содержащихся в массиве значений;
- написать оператор присваивания;
- заключить значения массива в квадратные скобки.

Внутри квадратных скобок необходимо сделать следующее:

- написать имя ключа, которое обычно является строкой (и, следовательно, заключается в кавычки);
- добавить оператор =>;
- указать желаемое значение. Строки всегда заключаются в кавычки, числа и логические значения в кавычках не пишутся;
- разделять элементы массива с помощью запятой.

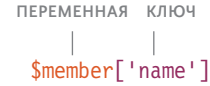


При создании ассоциативного массива также может использоваться следующий синтаксис, где вместо квадратных скобок используется конструкция `array()`.

```
$member = array(
  'name'    => 'Ivy',
  'age'     => 32,
  'country' => 'Italy',
);
```

Для доступа к элементам ассоциативного массива понадобится:

- имя переменной, содержащей массив;
- далее следуют квадратные скобки;
- внутри которых указывается ключ элемента, значение которого необходимо получить.



СОЗДАНИЕ АССОЦИАТИВНОГО МАССИВА И ДОСТУП К ЕГО ЭЛЕМЕНТАМ

PHP section_a/c01/associative-arrays.php

```
<?php
$nutrition = [
    'fat'   => 16,
    'sugar' => 51,
    'salt'  => 6.3,
];
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Nutrition (per 100g)</h2>
    <p>Fat:   <?php echo $nutrition['fat']; ?>%</p>
    <p>Sugar: <?php echo $nutrition['sugar']; ?>%</p>
    <p>Salt:  <?php echo $nutrition['salt']; ?>%</p>
  </body>
</html>
```

РЕЗУЛЬТАТ



1. В данном примере создается ассоциативный массив, который сохраняется в переменной `$nutrition`.

При создании массива используются квадратные скобки. Каждый элемент содержит пару «ключ — значение». Для присваивания значений элементам используется оператор `=>`.

2. При отображении данных понадобится следующее:

- команда `echo` указывает, что следующее значение будет выведено на веб-страницу;
- затем следует имя переменной, содержащей массив;
- затем следуют квадратные скобки и заключенное в кавычки имя ключа, к которому необходимо получить доступ.

Например, с помощью следующей команды можно отобразить на странице содержание сахара: `echo $nutrition['sugar'];`.

Упражнение. В шаге 1 измените значения элементов массива и присвойте им следующие значения:

- `Fat` (жиры) — `42`;
- `Sugar` (сахар) — `60`;
- `Salt` (соль) — `3,5`.

Затем сохраните и обновите страницу. Вы увидите, что информация на странице изменилась.

Упражнение. В шаге 1 добавьте еще один элемент в массив. Используйте ключ `protein` и присвойте ему значение `2,6`. Затем в шаге 2 покажите значение элемента `protein` на странице.

ИНДЕКСИРОВАННЫЕ МАССИВЫ

Если при создании массива элементам не присвоен ключ, интерпретатор PHP назначает его автоматически. Такие присвоенные номера называются **индексами**. Номера индексов начинаются с нуля (0), а не с единицы (1).

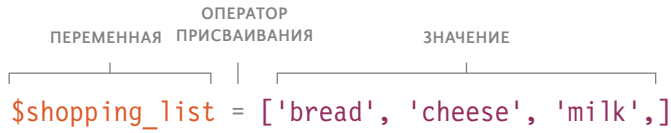
Для хранения индексированных массивов в переменной понадобится следующее:

- имя переменной, описывающее набор содержащихся в массиве значений;
- оператор присваивания;
- квадратные скобки.

Внутри квадратных скобок (или конструкции `array()`):

- перечисляются значения массива (строки всегда заключаются в кавычки, числа и логические значения пишутся без кавычек);
- элементы массива разделяются запятыми.

Каждому элементу будет присвоен индекс автоматически.



В примере выше элементу `bread` (хлеб) будет присвоен индекс 0, элементу `cheese` (сыр) — 1, элементу `milk` (молоко) — 2. Индексные номера, как правило, используются для определения порядка элементов в массиве.

При создании индексированного массива также может использоваться следующий синтаксис, где вместо квадратных скобок используется конструкция `array()`:

```
$shopping_list = array('bread',
                       'cheese',
                       'milk');
```

Каждое добавляемое в массив значение может находиться в той же или новой строке, как показано выше[3].

Для доступа к элементам индексированного массива понадобится:

- имя переменной, содержащей массив;
- далее следуют квадратные скобки;
- затем в квадратных скобках указывается индекс элемента, к которому необходимо получить доступ.

Ниже в примере выполняется доступ к третьему элементу массива, т. е. элементу `milk` (молоко).



СОЗДАНИЕ ИНДЕКСИРОВАННОГО МАССИВА И ДОСТУП К ЕГО ЭЛЕМЕНТАМ

PHP section_a/c01/indexed-arrays.php

```php
<?php
$best_sellers = ['Chocolate', 'Mints', 'Fudge',
    'Bubble gum', 'Toffee', 'Jelly beans',];
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Best Sellers</h2>
    <ul>
      <li><?php echo $best_sellers[0]; ?></li>
      <li><?php echo $best_sellers[1]; ?></li>
      <li><?php echo $best_sellers[2]; ?></li>
    </ul>
  </body>
</html>
```

РЕЗУЛЬТАТ



Рассмотрим пример создания индексированного массива.

1. Сначала создадим переменную `$best_sellers`. Этой переменной присвоим массив, содержащий список самых продаваемых товаров на сайте.

При создании такого массива используются квадратные скобки, а добавляемые элементы помещаются внутрь этих скобок. Поскольку элементы массива — это текст, они заключены в кавычки. Числа и логические значения в кавычки не заключаются. Все элементы массива разделяются запятыми.

2. Затем на странице будут отображены три самых продаваемых товара:

- команда `echo` используется для вывода значения;
- затем следует имя содержащей массив переменной;
- затем в квадратных скобках указан порядковый номер элемента, к которому необходимо получить доступ. Помните, что нумерация индексов начинается с 0, а не с 1.

Упражнение. В шаге 1 в массив после элемента `Fudge` (мягкий ирис) добавьте еще один элемент `Licorice` (лакричные конфеты). Затем перейдите к шагу 2 и добавьте в массив 4-й и 5-й элементы.

ОБНОВЛЕНИЕ МАССИВОВ

После того как массив создан, вы можете добавлять в него новые элементы или менять их значения.

Чтобы изменить значение, хранящееся в ассоциативном массиве, необходимо написать следующее:

- имя переменной, содержащей массив;
- квадратные скобки;
- строковое имя ключа заключается в кавычки;
- оператор присваивания;
- новое значение, которое должен получить элемент массива.

```
$member['name'] = 'Tom';
```

ПЕРЕМЕННАЯ КЛЮЧ НОВОЕ ЗНАЧЕНИЕ

Для добавления нового элемента в ассоциативный массив выполните все вышеперечисленные действия, но используйте новое имя ключа (не то, которое использовалось ранее в массиве).

Имя ключа заключается в кавычки, если это значение — строка, так как строки пишутся в кавычках.

Чтобы изменить значение, лежащее в индексированном массиве, необходимо написать следующее:

- имя переменной, содержащей массив;
- квадратные скобки;
- порядковый номер (не заключается в кавычки);
- оператор присваивания;
- новое значение.

```
$shopping_list[2] = 'butter';
```

ПЕРЕМЕННАЯ ИНДЕКС НОВОЕ ЗНАЧЕНИЕ

Тут было продемонстрировано добавление элементов в индексированные массивы. Обратите внимание на отличие: вы можете указать позицию нового элемента в массиве.

Номера индексов никогда не заключаются в кавычки, так как это числовой тип данных.

В КАКИХ СЛУЧАЯХ ИСПОЛЬЗОВАТЬ ТОТ ИЛИ ИНОЙ ТИП МАССИВА

Ассоциативные массивы предпочтительнее всего использовать в следующих случаях:

- когда известно, какую информацию будет хранить массив. Это необходимо для присвоения определенных имен каждому элементу;
- когда требуется произвольный доступ к отдельным элементам массива по ключу.

Индексированные массивы используются, как правило, в следующих случаях:

- когда неизвестно количество данных, хранимых в массиве. В этом случае номера индексов увеличиваются по мере добавления элементов в список;
- когда необходимо сохранить набор однородных значений в определенном порядке.

ИЗМЕНЕНИЕ ЗНАЧЕНИЙ ЭЛЕМЕНТОВ МАССИВА

PHP section_a/c01/updating-arrays.php

```php
<?php
$nutrition = [
    'fat'   => 38,
    'sugar' => 51,
    'salt'  => 0.25,
];
$nutrition['fat']   = 36;
$nutrition['fiber'] = 2.1;
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Nutrition (per 100g)</h2>
    <p>Fat:   <?php echo $nutrition['fat']; ?>%</p>
    <p>Sugar: <?php echo $nutrition['sugar']; ?>%</p>
    <p>Salt:  <?php echo $nutrition['salt']; ?>%</p>
    <p>Fiber: <?php echo $nutrition['fiber']; ?>%</p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Рассмотрим следующий пример.

1. Сохраним массив в переменную `$nutrition`.

Ключи и значения необязательно должны быть на новой строке (как показано в примере), однако, чтобы улучшить читабельность кода, предпочтительнее записывать их с новой строки.

2. Значение, соответствующее элементу `fat` (жиры), меняется с 38 на 36.
3. Затем в массив добавляется новый элемент `fiber` (пищевые волокна) со значением `2,1`.
4. Все значения, хранящиеся в массиве, выводятся на странице.

Упражнение. Выполнив шаг 3, добавьте еще один ключ `protein` (белки) и присвойте ему значение `7,3`.

КАК ХРАНИТЬ МАССИВЫ В МАССИВЕ

Каждый элемент массива может содержать в качестве значения другой массив. Таким образом можно создавать **многомерные массивы**. Ниже приведен пример использования многомерного массива.

Иногда возникает необходимость сохранить в элементе массива упорядоченный набор значений, например данные из таблицы. В качестве примера рассмотрим таблицу, содержащую следующие параметры: три пользователя, их возраст и местонахождение (страны).

NAME	AGE	COUNTRY
Ivy	32	UK
Emi	24	Japan
Luke	47	USA

Каждая строка этой таблицы (каждый пользователь) может быть представлена с помощью элемента индексированного массива. Эти элементы, в свою очередь, будут являться ассоциативными массивами, содержащими имя, возраст и страну пользователя.

Индексные номера для индексированного массива автоматически присваиваются интерпретатором PHP. Запятая в конце каждого ассоциативного массива указывает на конец значения для этого элемента.

```
$members = [
    ['name' => 'Ivy',  'age' => 32, 'country' => 'UK',],
    ['name' => 'Emi',  'age' => 24, 'country' => 'Japan',],
    ['name' => 'Luke', 'age' => 47, 'country' => 'USA',],
];
```

Чтобы получить массив, содержащий данные об Эми (Emi), напишите следующее:

- имя переменной, содержащей индексированный массив;
- индексный номер элемента, к которому необходимо получить доступ, должен записываться в квадратных скобках (всегда помните, что индексированные массивы начинаются с 0 и что числа не заключаются в кавычки).

```
$members[1];
```

Чтобы узнать возраст Люка (Luke), напишите следующее:

- имя переменной, содержащей индексированный массив;
- порядковый номер элемента, содержащего массив данных о Люке, должен записываться в квадратных скобках;
- ключ элемента, к которому необходимо получить доступ, в массиве, касающемся Люка, также должен записываться во втором наборе квадратных скобок (поскольку ключ — это строка, его необходимо заключить в кавычки).

```
$members[2]['age'];
```

МНОГОМЕРНЫЕ МАССИВЫ

PHP section_a/c01/multidimensional-arrays.php

```php
<?php
$offers = [
    ['name' => 'Toffee', 'price' => 5, 'stock' => 120,],
    ['name' => 'Mints',  'price' => 3, 'stock' => 66,],
    ['name' => 'Fudge',  'price' => 4, 'stock' => 97,],
];
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Offers</h2>
    <p><?php echo $offers[0]['name']; ?> -
      $<?php echo $offers[0]['price']; ?> </p>
    <p><?php echo $offers[1]['name']; ?> -
      $<?php echo $offers[1]['price']; ?> </p>
    <p><?php echo $offers[2]['name']; ?> -
      $<?php echo $offers[2]['price']; ?> </p>
  </body>
</html>
```

РЕЗУЛЬТАТ



1. Данный пример начинается с присвоения массива переменной `$offers`.

Каждый элемент массива содержит ассоциативный массив, содержащий название, цену и уровень запасов предлагаемого товара.

2. Выводится название первого товара, индексный номер которого равен 0.
3. Выводится цена первого товара.
4. Выводится название и цена второго товара.
5. Выводится название и цена третьего товара.

Упражнение. В шаге 1 добавьте в массив еще один товар `chocolate` (шоколад). Присвойте элементу `price` (цена) значение 2, а элементу `stock` (количество на складе) — 83. Выполнив шаг 5, выведите название и цену только что добавленного нового товара.

В следующей главе мы будем использовать циклы, чтобы выводить названия и цену для каждого товара, хранящегося в массиве `$offer`, независимо от количества товаров в массиве.

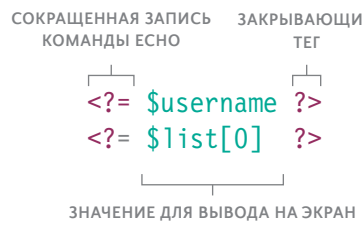
СОКРАЩЕННАЯ ФОРМА ЗАПИСИ КОМАНДЫ ECHO

Когда блок PHP используется только для вывода значения в браузер, вы можете использовать сокращенную запись вместо записи <?php echo?>.

Вместо записи <?php echo $name;?> вы можете использовать сокращенную запись <?= $name?>. Это единственный случай, когда нет необходимости использовать полный открывающий тег <?php.

Нет необходимости писать:

- символы php в открывающем теге;
- команду echo;
- точку с запятой перед закрывающим тегом.



Во многих примерах этой книги вы заметите, что каждый файл PHP состоит из двух частей.

- В первой части код PHP присваивает значения переменным или массивам. Этот код может выполнять различные операции с этими переменными.
- Затем располагается HTML-код, который выводится в браузер. Во второй части страницы эти переменные выводятся с использованием сокращенной формы записи.

Если каждая страница начинается с создания переменных, значения которых должны быть отображены на странице, то это поможет создать четкое разделение между кодом получения данных и кодом их отображения, что в дальнейшем позволит легко менять формат вывода, не затрагивая основной код.

Вторая же часть скрипта посвящена исключительно формированию HTML-кода, причем в этой части необходимо использовать минимально возможное количество кода PHP. В частности, в нескольких ближайших примерах код PHP в этой части страницы будет использоваться только для вывода значений переменных.

ИСПОЛЬЗОВАНИЕ СОКРАЩЕННОЙ ФОРМЫ ЗАПИСИ КОМАНДЫ ECHO

PHP section_a/c01/echo-shorthand.php

```
<?php
(1) $name      = 'Ivy';
(2) $favorites = ['Chocolate', 'Toffee', 'Fudge',];
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Echo Shorthand</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
(3)  <h2>Welcome <?= $name ?></h2>
    <p>Your favorite type of candy is:
(4)     <?= $favorites[0] ?>.</p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Рассмотрим следующий пример. В верхней части страницы создадим две переменные и присвоим им значения.

1. Переменная `$name` содержит имя пользователя сайта. Поскольку это текст, то он заключается в кавычки.
2. Переменная `$favorites` содержит набор любимых видов сладостей.
3. Имя выводится на страницу с использованием сокращенной формы записи команды `echo`.
4. Один из видов сладостей также выводится на страницу с использованием сокращенной формы записи команды `echo`.

Упражнение. В шаге 1 измените значение переменной `$name` и присвойте ей свое имя. В шаге 2 добавьте в начало массива вашу любимую сладость. Затем сохраните файл и обновите страницу. Вы увидите, что информация на странице изменилась.

ВЫРАЖЕНИЯ И ОПЕРАТОРЫ

Для создания одного нового значения, как правило, используется два (или более) исходных значения.
Выражение — это любая единица исходного программного кода, которая может быть вычислена для получения значения. Для вычисления значения используются **операторы**.

В простейших математических операциях (сложение, вычитание, умножение и деление) два исходных операнда используются для получения одного результата. В следующем примере представлено выражение, в котором, чтобы получить значение 15, число 3 умножается на число 5.

```
3 * 5
```

Как говорят программисты, выражения **выполняются** для получения результата. В примере ниже новое вычисленное значение сохраняется в переменной `$total`.

```
$total = 3 * 5;
```

Символы +, -, *, /, = называются **операторами**.

С помощью **оператора конкатенации** вы можете соединить две или более строк, чтобы получить один более длинный фрагмент текста. В следующем примере представлено выражение, которое объединяет значения `'Hi '` и `'Ivy'` в одну строку.

```
$greeting = 'Hi ' . 'Ivy';
```

В результате объединения этих двух строк получается одно значение `Hi Ivy`, которое сохраняется в переменной `$greeting`.

В оставшейся части этой главы вы будете изучать операторы.

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

Арифметические операторы позволяют работать с числами, выполняя такие действия, как сложение, вычитание, умножение и деление.

Например, при решении задачи, в которой необходимо вычислить общую стоимость трех пакетов, если стоимость каждого пакета равна $5, вы можете использовать оператор умножения.

ОПЕРАТОРЫ СРАВНЕНИЯ

Операторы сравнения сравнивают два значения и возвращают результат в виде логического значения `true` (истина) или `false` (ложь).

Например, взяв числа 3 и 5, вы можете проверить следующие утверждения и выяснить, являются ли они истинными или ложными:

- 3 больше, чем 5 (`false`);
- 3 равно 5 (`false`);
- 3 меньше, чем 5 (`true`).

Также вы можете сравнивать строки.

- `'Apple'` больше, чем `'Banana'` (`false`);
- `'A'` равно `'B'` (`false`);
- `'A'` меньше, чем `'B'` (`true`).

СТРОКОВЫЕ ОПЕРАТОРЫ

Строковые операторы позволяют работать с текстом.

В PHP существует два оператора для работы со строками, которые используются для объединения отдельных фрагментов текста в одну строку.

Например, одна переменная — это имя пользователя, а вторая — его фамилия. Вы можете объединить эти две переменные, чтобы в результате получить полное имя.

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

PHP поддерживает стандартные логические операторы: `and` (логическое И), `or` (логическое ИЛИ) и `not` (логическое отрицание). Логические операторы возвращают одно логическое значение — `true` или `false`. Рассмотрим следующий пример, в котором общий ответ на оба нижеследующих вопроса получает значение `true` или `false`.

Высокая ли температура воздуха? Солнечная ли погода?

- Оператор `and` проверяет, верны ли ответы на оба вопроса: являются ли температура высокой **и** погода — солнечной.
- Оператор `or` проверяет, является ли верным ответ хотя бы на один из вопросов: высокая ли температура **или** солнечная ли погода.
- Оператор `not` инвертирует логическое значение, превращая `false` в `true`, а `true` — в `false`. Например, погода не солнечная?

Каждое из этих утверждений возвращает значение `true` или `false`.

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

PHP поддерживает все основные арифметические операторы. В качестве операндов используются целые и вещественные числа.

ОПЕРАТОР	ОБОЗНАЧЕНИЕ	ОПИСАНИЕ	ПРИМЕР	РЕЗУЛЬТАТ
Сложение	`+`	Сложение двух значений	`10 + 5`	`15`
Вычитание	`-`	Вычитание одного значения из другого	`10 - 5`	`5`
Умножение	`*`	Перемножение двух значений (примечание: используйте символ звездочки)	`10 * 5`	`50`
Деление	`/`	Деление одного значения на другое	`10 / 5`	`2`
Остаток от деления	`%`	Деление одного значения на другое и возвращение остатка (деление по модулю)	`10 % 3`	`1`
Возведение в степень	`**`	Оператор возведения в степень возвращает результат возведения первого операнда в степень, равную второму	`10 ** 5`	`100000`
Инкремент[4]	`++`	Оператор инкремента возвращает текущее значение, а потом увеличивает операнд на единицу	`$i = 10; $i++;`	`10, но переменная получает значение 11`
Декремент	`--`	Операция декремента возвращает текущее значение и уменьшает операнд на 1	`$i = 10; $i--;`	`10, но переменная получает значение 9`

ПРИОРИТЕТ ОПЕРАТОРОВ

Если в одном выражении используется несколько операторов одновременно, то важно понимать порядок, в котором они будут выполняться. Например, умножение и деление выполняются перед сложением и вычитанием.

Порядок вычисления операторов влияет на результат вычисления. Например, в следующем примере операторы вычисляются слева направо, и получается результат — 16:

```
$total = 2 + 4 + 10;
```

Однако результат вычисления следующего выражения — 42, а не 60:

```
$total = 2 + 4 * 10;
```

Для изменения порядка выполнения операторов служат круглые скобки. Например, результат вычисления следующего выражения — 60.

```
$total = (2 + 4) * 10;
```

Скобки указывают, что сначала выполняется операция сложения чисел 2 и 4, а затем выполняется умножение на число 10.

ИСПОЛЬЗОВАНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАТОРОВ

PHP section_a/c01/arithmetic-operators.php

```php
<?php
$items    = 3;
$cost     = 5;
$subtotal = $cost * $items;
$tax      = ($subtotal / 100) * 20;
$total    = $subtotal + $tax;
?>
<!DOCTYPE html>
  <html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Shopping Cart</h2>
    <p>Items: <?= $items ?></p>
    <p>Cost per pack: $<?= $cost ?></p>
    <p>Subtotal: $<?= $subtotal ?></p>
    <p>Tax: $<?= $tax ?></p>
    <p>Total: $<?= $total ?></p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Рассмотрим использование арифметических операторов на примере расчета стоимости заказа. Сначала создадим две переменные.

1. Переменная `$items` — итоговое количество заказанных товаров.
2. Переменная `$cost` — стоимость одной упаковки.

Затем выполняются вычисления, и результаты сохраняются в переменных до создания HTML-кода. Это помогает отделить код PHP от вывода HTML.

3. Стоимость заказа рассчитывается путем умножения количества товаров на стоимость одной упаковки сладостей.
4. Затем необходимо добавить налог в размере 20%. Чтобы выполнить это, промежуточный итог необходимо разделить на 100 (это выражение необходимо заключить в круглые скобки, чтобы гарантировать, что оно будет вычислено первым). Затем результат необходимо умножить на 20.
5. Наконец, к промежуточной сумме добавим налог, чтобы определить общую стоимость.
6. Результаты, которые были сохранены в переменных, будут считаны и отображены на HTML-странице.

Упражнение. В шаге 1 измените стоимость товаров, а в шаге 2 — их количество.

СТРОКОВЫЕ ОПЕРАТОРЫ

Часто возникает необходимость объединить две или более строки для получения одного строкового значения. Объединение двух или более строк в одну называется **конкатенацией**.

ОПЕРАТОР КОНКАТЕНАЦИИ

Оператор конкатенации — это символ точки `.`, который возвращает строку, представляющую собой объединение двух строковых аргументов. Ниже в примере переменная `$name` будет содержать строку `'Ivy Stone'`:

```
$forename = 'Ivy';
$surname = 'Stone';
$name = $forename. ' '. $surname;
```

Обратите внимание, что между переменными `$forename` и `$surname` добавлен пробел. Если бы пробела не было, то переменная `$name` содержала бы значение `IvyStone`.

Вы можете объединить любое количество строк. При этом оператор конкатенации должен использоваться между всеми строками.

Вы можете объединять строки, которые хранятся в переменных, не используя оператор конкатенации. Если значение заключено в двойные кавычки (а не в одинарные), интерпретатор PHP заменит имена переменных в двойных кавычках значениями, которые они содержат. В примере ниже переменная `$name` будет содержать значение `Ivy Stone`.

```
$name = «$forename $surname»;
```



ОПЕРАТОР ПРИСВАИВАНИЯ С КОНКАТЕНАЦИЕЙ

Оператор присваивания с конкатенацией используется, чтобы добавить текст к существующей переменной. Этот способ считается сокращенной формой записи при объединении строк.

```
$greeting = 'Hello ';
$greeting .= 'Ivy';
```

В данном примере строка `'Hello '` хранится в переменной `$greeting`. В следующей строке, с помощью оператора присваивания с конкатенацией, после значения переменной `$greeting` добавляется строка `'Ivy'`.

Теперь переменная `$greeting` содержит значение `'Hello Ivy'`. В данном примере программный код содержит на одну строку меньше, чем в предыдущем.

ОБЪЕДИНЕНИЕ СТРОК

PHP section_a/c01/string-operator.php

```php
<?php
$prefix  = 'Thank you';
$name    = 'Ivy';
$message = $prefix . ', ' . $name;
?>
<!DOCTYPE html>
<html>
  <head>
    <title>String Operator</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>
    <h2><?= $name ?>'s Order</h2>
    <p><?= $message ?></p>
  </body>
</html>
```

РЕЗУЛЬТАТ



В данном примере рассмотрим вывод персонализированного сообщения.

1. Для хранения начала сообщения создадим переменную `$prefix`, содержащую текст `'Thank you'`.
2. Для хранения имени пользователя создадим переменную `$name`, содержащую значение `Ivy`.
3. Создадим персонализированное сообщение, используя конкатенацию (объединение) трех значений, и сохраним новое значение в переменной `$message`:

- сначала берется значение переменной `$prefix`;
- затем к нему добавляется запятая и пробел;
- далее к ним добавляется значение переменной `$name`;
- и в завершение конечный результат всех этих операций присваивается переменной `$message`.

Упражнение. В шаге 2 измените значение переменной `$name`, присвоив ей свое имя.

Упражнение. В шаге 3 присвойте значение переменной `$message`, используя двойные кавычки (при этом не используя оператор конкатенации).
`$message = «$prefix $name»;`

ОПЕРАТОРЫ СРАВНЕНИЯ

Операторы сравнения позволяют сравнивать между собой два значения. Если условие выполнено, возвращается значение `true` (истина), а если нет — `false` (ложь).



ОПЕРАТОР РАВЕНСТВА

Этот оператор проверяет, что значения равны.
Выражение `'Hello' == 'Hello'` возвращает значение `true`, так как строки слева и справа совпадают.
Выражение `'Hello' == 'Goodbye'` возвращает значение `false`, так как операнд слева и операнд справа — это разные строки.

Операторы, показанные выше, позволяют интерпретатору PHP определять, равны ли эти два значения без учета типов. Те же операторы, которые приведены ниже, являются более строгими, поскольку они проверяют на равенство не только значения, но и тип данных.



ОПЕРАТОР НЕРАВЕНСТВА

Этот оператор проверяет, что значения не равны.
Выражение `'Hello' != 'Hello'` возвращает значение `false`, так как строки слева и справа совпадают.
Выражение `'Hello' != 'Goodbye'` возвращает значение `true`, так как операнд слева и операнд справа — это разные строки.

Например, используя приведенные выше операторы, мы выясним, что числа 3 (целое число) и 3,0 (число с плавающей запятой) равны. Используя приведенные ниже операторы, узнаем, что числа 3 и 3,0 не равны. 0 можно интерпретировать в качестве логического значения `false`, а 1 — `true`.



ОПЕРАТОР ИДЕНТИЧНОСТИ

Этот оператор проверяет, что значения операндов равны и относятся к одному и тому же типу данных.
Выражение `'3' === 3` возвращает значение `false`, так как операнды относятся к разному типу данных.
Выражение `'3' === '3'` возвращает значение `true`, так как оба операнда имеют один и тот же тип данных и значение.



ОПЕРАТОР НЕИДЕНТИЧНОСТИ

Этот оператор проверяет, что значения операндов не равны или не относятся к одному и тому же типу данных.
Выражение `3.0 !== 3` возвращает значение `true`, так как операнды относятся к разному типу данных.
Выражение `3.0 !== 3.0` возвращает значение `false`, так как оба операнда имеют один и тот же тип данных и значение.

При выводе логического значения командой echo вместо true будет отображаться значение 1, а вместо false не будет выведено ничего.

< и >

ОПЕРАТОРЫ МЕНЬШЕ И БОЛЬШЕ

Оператор < (меньше) проверяет, что значение операнда слева меньше значения операнда справа.
Выражение `4 < 3` возвращает значение `false`.
Выражение `3 < 4` возвращает значение `true`.
Оператор > (больше) возвращает значение true, когда значение операнда слева больше значения операнда справа.
Выражение `z > a` возвращает значение `true`.
Выражение `a > z` возвращает значение `false`.

<= и >=

ОПЕРАТОРЫ МЕНЬШЕ ИЛИ РАВНО И БОЛЬШЕ ИЛИ РАВНО

Оператор <= (меньше или равно) проверяет, что значение операнда слева меньше или равно значению операнда справа.
Выражение `4 <= 3` возвращает значение `false`.
Выражение `3 <= 4` возвращает значение `true`.
Оператор >= (больше или равно) проверяет, что значение операнда слева больше или равно значению операнда справа.
Выражение `z >= a` возвращает значение `true`.
Выражение `z >= z` возвращает значение `true`.

<=>

ОПЕРАТОР ТРЕХСТОРОННЕГО СРАВНЕНИЯ

Оператор трехстороннего сравнения (его еще называют spaceship, космический корабль) сравнивает значения операндов слева и справа и возвращает следующие целочисленные значения:

- 0, если оба значения равны;
- 1, если значение слева больше;
- –1, если значение справа больше.

Оператор трехстороннего сравнения был введен в PHP 7 и не работает с более ранними версиями.

Выражение `1 <=> 1` возвращает значение `0`.
Выражение `2 <=> 1` возвращает значение `1`.
Выражение `2 <=> 3` возвращает значение `–1`.

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

Операторы сравнения возвращают только одно логическое значение — `true` или `false`. Логические операторы могут использоваться с несколькими операторами сравнения, чтобы получить результат для нескольких выражений.

В примере ниже представлена строка кода, в которой имеются три выражения, каждое из которых возвращает значение `true` или `false`.

В выражении 1 (слева) и выражении 2 (справа) используются операторы сравнения, и оба выражения возвращают значение `false`.

В выражении 3 используется логический оператор, а не оператор сравнения.

Оператор логическое И (`AND` и `&&`) проверяет, возвращают ли оба выражения (с любой стороны) значение `true`. В нашем случае это не так, поэтому все выражение в результате вернет значение `false`.

В первую очередь выполняются выражения 1 и 2, затем выполняется выражение 3.

Каждое выражение заключено в отдельные круглые скобки. Это улучшает читабельность кода и показывает, что отдельный набор круглых скобок — это каждое отдельное выражение, которое возвращает одно значение. Данный программный код работает и без скобок, однако использование круглых скобок значительно улучшает его читабельность.



&&

ЛОГИЧЕСКОЕ И

Этот оператор проверяет по крайней мере один операнд.

```
((2 < 5) && (3 >= 2))
```
Данное выражение возвращает значение `true`.

Если оба выражения вернут значение `true`, то все выражение также вернет значение `true`. Если одно из выражений вернет значение `false`, то все выражение также вернет значение `false`.

Выражение `true && true` возвращает значение `true`
Выражение `true && false` возвращает значение `false`
Выражение `false && true` возвращает значение `false`
Выражение `false && false` возвращает значение `false`

Вместо символов `&&` можно использовать слово `and`.

||

ЛОГИЧЕСКОЕ ИЛИ

Этот оператор проверяет по крайней мере один операнд.

```
((2 < 5) || (2 < 1))
```
Данное выражение возвращает значение `true`.

Если значение одного из выражений равно `true`, то все выражение также возвращает `true`. Если оба выражения возвращают значение `false`, все выражение также возвращает `false`.

Выражение `true || true` возвращает значение `true`
Выражение `true || false` возвращает значение `true`
Выражение `false || true` возвращает значение `true`
Выражение `false || false` возвращает значение `false`

Вместо символов `||` можно использовать слово `or`.

!

ЛОГИЧЕСКОЕ НЕ

Этот оператор инвертирует логическое значение единственного операнда.

```
!(2 < 1)
```
Данное выражение возвращает значение `true`.

Знак !, расположенный перед операндом, также называют отрицанием. Если утверждение ложно, то в результате вернется значение `true`. Если утверждение верно, то в результате вернется значение `false`.

Выражение `!true` возвращает значение `false`.
Выражение `!false` возвращает значение `true`.

Нельзя использовать слово `not` вместо знака !.

УПРОЩЕННОЕ (СОКРАЩЕННОЕ) ВЫЧИСЛЕНИЕ

Логические выражения вычисляются слева направо. После того как первое выражение уже вычислено и известен логический оператор, интерпретатор PHP может не вычислять результат второго выражения, как описано в примерах справа.

```
((5 < 2) && (2 >= 2))
```
Обнаружено значение `false`.

PHP не будет продолжать проверку и вычислять второе условие, так как выражение в любом случае не вернет `true` (поскольку в случае `&&` для этого оба операнда должны вернуть `true`).

```
((2 < 5) || (2 >= 2))
```
Обнаружено значение `true`.

PHP не будет продолжать проверку и вычислять второе условие, так как выражение в любом случае вернет `true` (поскольку в случае `||` для этого достаточно, чтобы `true` вернул любой из операндов).

ИСПОЛЬЗОВАНИЕ ОПЕРАТОРОВ СРАВНЕНИЯ

1. Создадим три переменные:

- первая переменная — вид сладостей, который желает приобрести покупатель;
- вторая переменная указывает, что в наличии 5 упаковок;
- третья переменная указывает, что покупатель хочет купить 8 упаковок.

2. Оператор сравнения проверяет, является ли число нужных покупателю упаковок меньшим или равным количеству, имеющемуся на складе. Результат сохраняется в переменной `$can_buy`.

3. Как правило, логическое значение никогда не выводится. Вероятнее всего, это значение будет использоваться в условной логике, которая описана в следующей главе. Однако важно увидеть, что вы получите при попытке вывести такое логическое значение. Если значение равно:

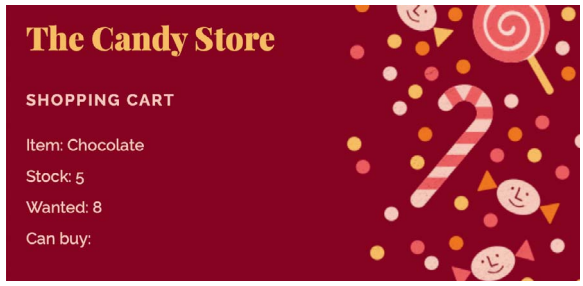
- `true` — на странице отобразится значение 1;
- `false` — на странице не отобразится ничего.

Упражнение. В шаге 1 поменяйте местами значения переменных `$stock` и `$wanted`. Значение переменной `$can_buy` будет изменено.

section_a/c01/comparison-operators.php **PHP**

```
<?php
$item    = 'Chocolate';
$stock   = 5;
$wanted  = 8;
$can_buy = ($wanted <= $stock);
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Shopping Cart</h2>
    <p>Item:    <?= $item ?></p>
    <p>Stock:   <?= $stock ?></p>
    <p>Wanted:  <?= $wanted ?></p>
    <p>Can buy: <?= $can_buy ?></p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Далее вы узнаете, как отображать разные сообщения в зависимости от того, какой результат вернуло выражение с оператором сравнения — `true` или `false`.

ИСПОЛЬЗОВАНИЕ ЛОГИЧЕСКИХ ОПЕРАТОРОВ

PHP section_a/c01/logical-operators.php

```
<?php
$item    = 'Chocolate';
$stock   = 5;
$wanted  = 3;
$deliver = true;
$can_buy = (($wanted <= $stock) && ($deliver == true));
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Shopping Cart</h2>
    <p>Item:    <?= $item ?></p>
    <p>Stock:   <?= $stock ?></p>
    <p>Ordered: <?= $wanted ?></p>
    <p>Can buy: <?= $can_buy ?></p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Далее вы узнаете, как отображать разные сообщения в случаях, когда результат выражения равен `true` или `false`.

Этот пример строится на предыдущем.

1. Допустим, что покупатель желает приобрести 3 упаковки сладостей.
2. Добавим переменную `$deliver`, определяющую, может ли быть осуществлена доставка.
3. Используем два оператора сравнения:

- первый оператор проверяет, достаточно ли товаров на складе;
- второй оператор проверяет, может ли быть осуществлена доставка.

Логический оператор `&&` проверяет, приводят ли оба оператора к истинному результату. Если это так, то значение переменной `$can_buy` будет равно `true` и на странице будет выведено число 1.

Если хотя бы один из этих операторов приводит к истинному результату, значение переменной `$can_buy` будет равно `false` и ничего не будет отображено.

Упражнение. В шаге 1 поменяйте местами значения переменных `$stock` и `$wanted`. Значение переменной `$can_buy` будет изменено.

ЖОНГЛИРОВАНИЕ, ИЛИ ПРИВЕДЕНИЕ ТИПОВ

Интерпретатор PHP может в любой момент привести переменную к другому типу. Это известно как **жонглирование типами** или **приведение типов** и может иногда приводить к непредвиденным результатам.

PHP — это язык программирования **со слабой типизацией**, так как при создании переменной нет необходимости указывать для нее тип данных. В примере ниже переменная `$title` сначала содержит строку, а затем целое число:

```
$title = 'Ten'; //Строковый тип данных
$title = 10; //Целочисленный тип
```

В этом отличие PHP от языков со строгой типизацией (таких, как C++ или C#), требующих от программиста указания типа данных, который будет содержать каждая переменная при ее объявлении и который в дальнейшем нельзя будет изменить.

Когда интерпретатор PHP обнаруживает значение, тип которого не подходит для использования в текущей операции, интерпретатор может попытаться преобразовать тип этого значения к ожидаемому. Этот процесс называется **жонглированием типами**.

Различные манипуляции с типами могут вызвать путаницу, так как в некоторых случаях это может приводить к непредвиденным результатам или ошибкам. Однако в большинстве случаев такая конверсия приводит к ожидаемому результату. Например, число 1 относится к целочисленному типу, а 2 — к строковому (поскольку заключено в кавычки).

```
$total = 1 + '2';
```

В данном случае, чтобы выполнить арифметическую операцию, интерпретатор PHP попытается преобразовать строку в число. В результате переменная `$total` будет содержать число 3.

Далее вы можете ознакомиться с правилами преобразования из одного типа данных в другой. Примеры, демонстрирующие преобразование типов, можно найти на сайте: `http://notes.re/php/type-juggling`.

Когда один тип данных меняется на другой, программисты говорят, что тип данных **преобразуется** (type casting) из одного типа в другой. Также такое поведение называется **неявным приведением** типов, поскольку выполняется интерпретатором автоматически, без непосредственного участия программиста.

Когда программист явно изменяет тип данных значения с помощью кода, это называется **явным приведением типов**.

ЧИСЛА

Когда интерпретатор PHP ожидает два числа, он может выполнить с ними арифметические операции.

В примерах ниже вы можете увидеть, что происходит, когда:

- к числу добавляется строка;
- к числу добавляется логическое значение.

ЧИСЛО + СТРОКА	ИНТЕРПРЕТИРУЕТСЯ КАК...	РЕЗУЛЬТАТ	ОПИСАНИЕ
`1 + '1'`	`1 + 1`	`2 (int)`	Строка содержит целое число. Интерпретируется как целое число
`1 + '1.2'`	`1 + 1.2`	`2.2 (float)`	Строка содержит вещественное число. Интерпретируется как вещественное число
`1 + '1.2e + 3'`	`1 + 1200`	`1201 (float)`	Строка содержит число с плавающей запятой в экспоненциальной записи. Интерпретируется как число с плавающей запятой
`1 + '5star'`	`1 + 5`	`6 (int)`	Строка содержит целое число, за которым следуют другие символы. Интерпретируется как целое число. Последующие символы игнорируются[5]
`1 + '3.5star'`	`1 + 3.5`	`4.5 (float)`	Строка содержит число с плавающей запятой, за которым следуют другие символы. Интерпретируется как число с плавающей запятой. Последующие символы игнорируются
`1 + 'star9'`	`1 + 0`	`1 (int)`	Строка начинается с любого символа, кроме числа. Интерпретируется как 0[6]

ЧИСЛО + ЛОГИЧЕСКОЕ ЗНАЧЕНИЕ	ИНТЕРПРЕТИРУЕТСЯ КАК...	РЕЗУЛЬТАТ	ОПИСАНИЕ
`1 + true`	`1 + 1`	`2 (int)`	Логическое значение true интерпретируется как целое число 1
`1 + false`	`1 + 0`	`1 (int)`	Логическое значение false интерпретируется как целое число 0

СТРОКИ

При использовании оператора конкатенации интерпретатор PHP будет следовать следующим правилам.

В примерах ниже вы можете увидеть, что происходит, когда:

- строка объединяется с числом;
- строка объединяется с логическим значением.

СТРОКА. ЧИСЛО	ИНТЕРПРЕТИРУЕТСЯ КАК...	РЕЗУЛЬТАТ	ОПИСАНИЕ
`'Hi '.1`	`'Hi  '. '1'`	`Hi 1` (строка)	Целое число интерпретируется как строка
`'Hi '.1.23`	`'Hi'. '1.23'`	`Hi 1.23` (строка)	Число с плавающей запятой интерпретируется как строка

СТРОКА. ЛОГИЧЕСКОЕ ЗНАЧЕНИЕ	ИНТЕРПРЕТИРУЕТСЯ КАК...	РЕЗУЛЬТАТ	ОПИСАНИЕ
`'Hi '.true`	`'Hi '. '1'`	`Hi 1` (строка)	Логическое значение true интерпретируется как целое число 1
`'Hi '.false`	`'Hi'. ''`	`Hi` (строка)	Логическое значение false интерпретируется как пустая строка

ЛОГИЧЕСКИЕ ЗНАЧЕНИЯ

Когда интерпретатор PHP ожидает логическое значение, все значения, приведенные в таблице справа, будут интерпретироваться как ложные.

Любые другие значения будут интерпретированы как `true`: любой текст (даже один пробел), любые числа кроме нуля, любой непустой массив (даже с пустыми значениями) и собственно логическое значение `true`.

ЗНАЧЕНИЕ	ТИП ДАННЫХ	ИНТЕРПРЕТИРУЕТСЯ КАК...
`false`	Boolean (логический тип данных)	`false`
`0`	Integer (целочисленный тип)	`false`
`0.0`	Float (число с плавающей запятой)	`false`
`'0'`	Строка со значением 0	`false`
`''`	Пустая строка	`false`
`array[]`	Пустой массив	`false`
`null`	Null	`false`

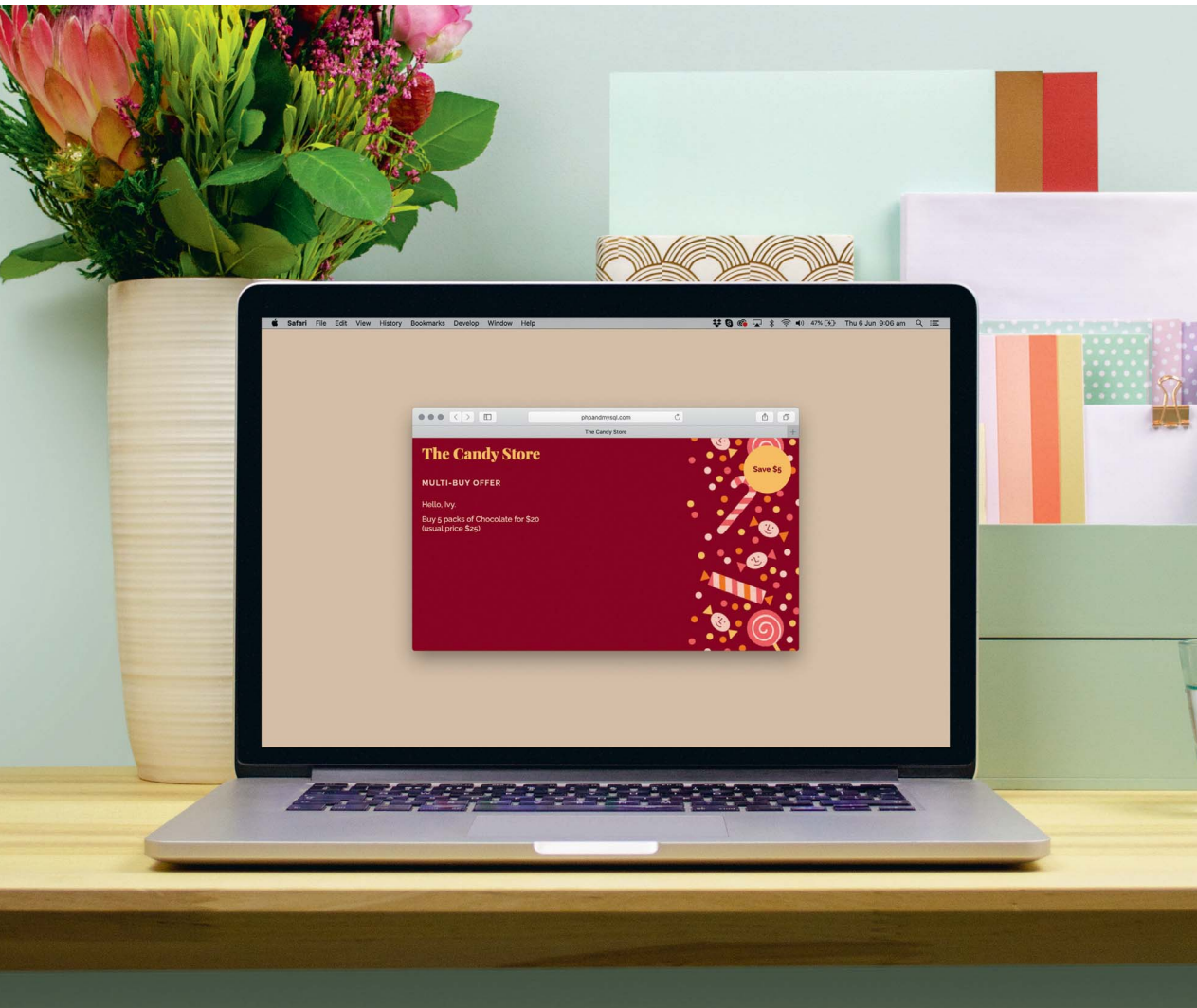
ПРОСТЕЙШАЯ СТРАНИЦА НА PHP

Здесь мы соберем вместе все те технологии, с которыми познакомились в этой главе. С помощью PHP мы сгенерируем HTML-страницу, сообщающую пользователям о скидках при покупке нескольких упаковок сладостей.

Вы увидите единый пример, в котором показано, как:

- сохранять информацию в скалярных переменных и массивах;
- использовать оператор конкатенации для объединения строк, например, чтобы создать персонализированное сообщение для пользователя;
- использовать арифметические операторы для вычисления цен, которые будут показаны на странице;
- выводить новые значения, созданные с помощью PHP, в HTML-код.

Кроме того, если значения в переменных будут изменены, то страница автоматически будет отображать новые товары и цены.



ОБРАБОТКА И ОТОБРАЖЕНИЕ ДАННЫХ

Когда вы только начинаете работать с PHP, файлы часто содержат и код HTML, и код PHP. Но при этом рекомендуется максимально разделять обработку данных и их вывод.

- В начале файла рекомендуется писать код, который создает все те значения, которые будут в дальнейшем выводиться, и присваивает эти значения переменным. На странице справа это показано над горизонтальной пунктирной линией.
- Затем, в нижней части страницы, будет располагается код HTML. PHP в этой части тоже используется, но только вывода заранее подготовленных переменных (в примере справа — ниже горизонтальной пунктирной линии).

Рассмотрим PHP-код, расположенный в начале страницы.

1. В первую очередь объявим переменную `$username`, в которой будет храниться имя пользователя. Все имена переменных начинаются с символа $ и должны описывать суть тех данных, которые они содержат.

2. Переменная `$greeting` будет содержать строку, которая приветствует пользователя. Для этого используется строковый оператор для соединения строки `Hello` и имени пользователя.

3. Переменная `$offer` содержит сведения о специальном предложении. Значение переменной `$offer` — это массив из четырех элементов:

- предлагаемый товар;
- количество товара, которое надо приобрести для получения скидки;
- цена упаковки без скидки;
- цена упаковки со скидкой.

Первый элемент, описывающий предлагаемый товар, содержит строковый тип данных. Остальные значения — целые числа.

4. Переменная `$usual_price` будет содержать стоимость товара без скидки. Она рассчитывается путем перемножения двух значений, хранящихся в массиве, — количества и цены.

5. Переменная `$offer_price` будет содержать стоимость товаров с учетом скидки. Она рассчитывается путем умножения количества на цену со скидкой.

6. Переменная `$saving` обозначает сэкономленные средства пользователя. Это значение рассчитывается путем вычитания значения переменной `$offer_price` (объявленной в шаге 5) из значения переменной `$normal_price` (объявленной в шаге 4).

Во второй части страницы (ниже пунктирной горизонтальной линии) создается HTML-код, который будет выведен в браузер. HTML-код начинается с объявления HTML DOCTYPE. PHP используется только для вывода значений, которые были сохранены в переменных в предыдущих шагах.

7. Приветствие `Hello`, за которым следует имя пользователя, выводится с использованием сокращенной формы команды `echo`.

8. Сэкономленные средства — значение переменной `$saving` (объявленной в шаге 6) — обозначены желтым кружком. С помощью CSS он выводится в правом верхнем углу окна браузера.

9. Далее выводятся детали предложения: количество конфет, которое необходимо купить для получения скидки, и их название.

10. Затем следует цена со скидкой (переменная `$offer_price`) и первоначальная цена (переменная `$normal_price`).

PHP section_a/c01/example.php

```php
<?php
$username = 'Ivy'; //Объявление переменной, содержащей имя пользователя

$greeting = 'Hello, '. $username. '.'; //Приветствие 'Hello, ' + имя

$offer = [//Массив, содержащий специальное предложение
    'item' => 'Chocolate', //Название товара со скидкой
    'qty' => 5, //Количество
    'price' => 5, //Первоначальная цена за упаковку
    'discount' => 4, //Цена со скидкой за упаковку
];

$usual_price = $offer['qty'] * $offer['price']; //Цена без скидки
$offer_price = $offer['qty'] * $offer['discount']; //Цена со скидкой
$saving = $usual_price – $offer_price; //Сэкономленные средства
?>
```

```html
<! DOCTYPE html>
<html>
  <head>
    <title>The Candy Store</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>The Candy Store</h1>

    <h2>Multi-buy Offer</h2>

    <p><?= $greeting?></p>

    <p class="sticker">Save $<?= $saving?></p>

    <p>Buy <?= $offer['qty']?>packs of <?= $offer['item']?>
       for $<?= $offer_price?><br>(usual price $<?= $usual_price?>)</p>
  </body>
</html>
```

Упражнение. В шаге 1 измените имя пользователя на свое имя.

В шаге 2 обновите приветствие (вместо `Hello` поставьте `Hi`).

В шаге 3 измените количество упаковок (ключ `qty` массива `$offer`) на значение 3.

В шаге 4 измените цену конфет на значение 6.

ЗАКЛЮЧЕНИЕ

ПЕРЕМЕННЫЕ, ВЫРАЖЕНИЯ И ОПЕРАТОРЫ

- В переменных хранятся данные, которые могут изменяться при каждом запуске сценария.
- К скалярным типам данных относятся текст, целые числа, числа с плавающей запятой, а также логические значения `true` или `false`.
- Массив — это составной тип данных, используемый для хранения набора связанных друг с другом значений.
- Массивы состоят из элементов. Элементы ассоциативного массива имеют ключ и значение. Элементы индексированного массива имеют номер индекса и значение.
- Строковые операторы конкатенируют (объединяют) текст в одну большую строку.
- Арифметические операторы выполняют вычисления с числами.
- Операторы сравнения сравнивают одно значение с другим, чтобы выяснить, равны они или одно из них больше или меньше другого. Они возвращают логическое значение `true` (истина) или `false` (ложь).
- Логические операторы позволяют комбинировать результат нескольких выражений, используя стандартные логические операторы: `and` (логическое И), `or` (логическое ИЛИ) и `not` (логическое отрицание). Логические операторы возвращают одно логическое значение — `true` или `false`.

2

УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ

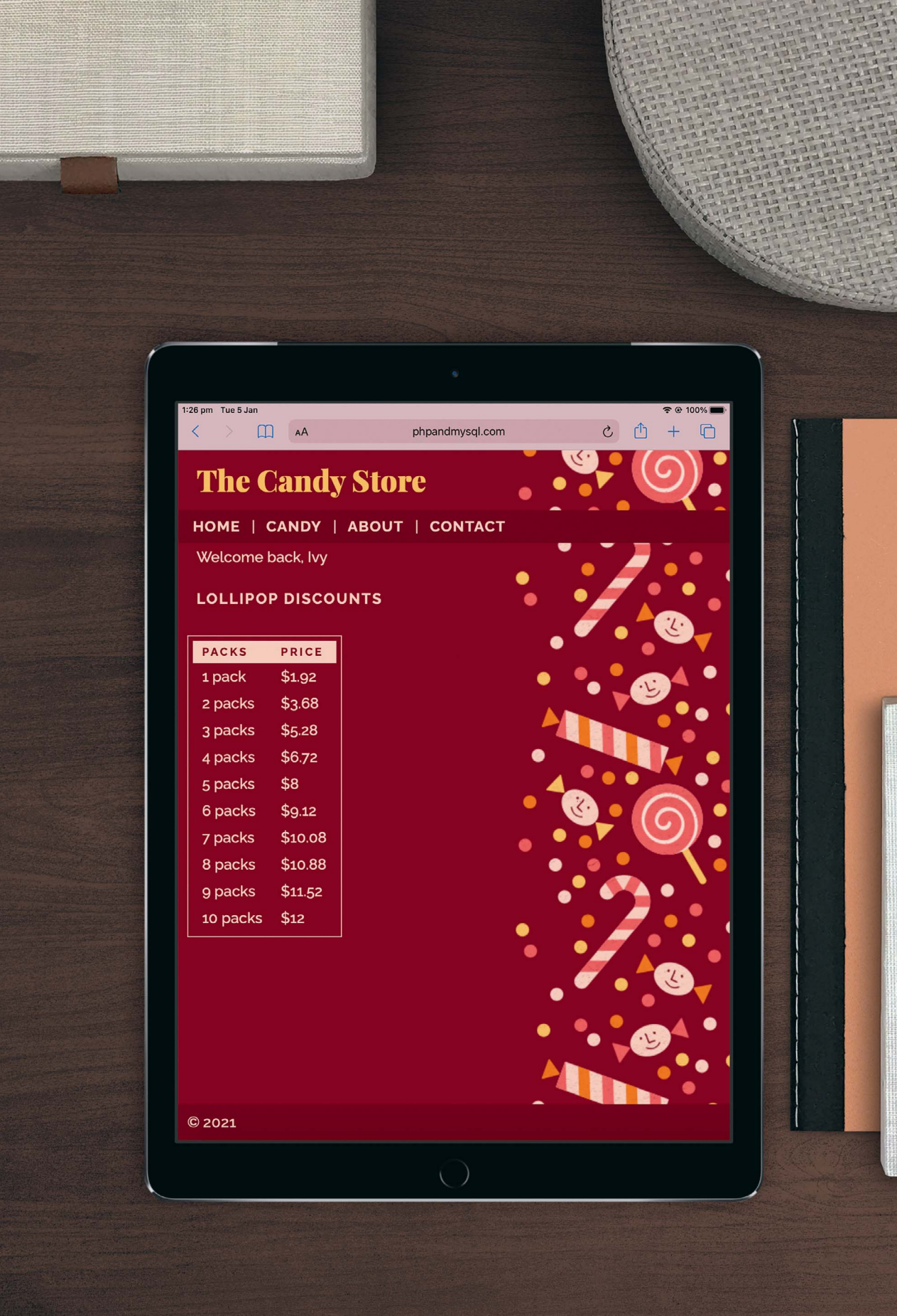
В этой главе описано, как объяснить интерпретатору PHP, стоит ли выполнять определенный блок кода; как несколько раз повторить набор из нескольких инструкций; и когда необходимо подключить код из другого файла.

Существует три способа задать порядок выполнения инструкций в коде PHP.

- **Последовательность.** Интерпретатор обрабатывает операторы в том порядке, в котором они написаны. Строка 1, строка 2, строка 3 и так далее, пока не достигнет последней строки. Во всех примерах, которые вы видели до сих пор, код выполняется последовательно.
- **Выбор.** Интерпретатор проверяет условие и в зависимости от результата определяет, какой блок кода надо выполнить. Например, условие может быть следующее: «Пользователь вошел в систему?» Если ответ «Нет», может быть отображена ссылка на страницу входа. Если ответ «Да», может быть отображена ссылка на страницу профиля пользователя. Программисты называют такие инструкции **условными операторами**, так как они в зависимости от условия выбирают, какой набор инструкций выполнять.
- **Повторение/итерирование.** Интерпретатор может повторять несколько раз один и тот же набор кода. Например, если массив содержит список покупок, один и тот же набор кода может выполняться для каждого элемента в списке независимо от того, содержит он 1 или 100 элементов. **Циклы** используются для повторения наборов инструкций.

Когда вы изменяете очередность исполнения инструкций в коде, это называется изменением **порядка (или потока) управления (control flow)**.

Изучив эту главу, вы также научитесь применять **подключаемые файлы**, которые содержат код, использующийся на нескольких страницах. Это метод позволяет включать один файл в другие скрипты, вместо того чтобы повторять в каждом файле один и тот же код снова и снова.


1:26 pm Tue 5 Jan
100%
AA
phpandmysql.com
The Candy Store
HOME | CANDY | ABOUT | CONTACT
Welcome back, Ivy
LOLLIPOP DISCOUNTS
PACKS PRICE
1 pack $1.92
2 packs $3.68
3 packs $5.28
4 packs $6.72
5 packs $8
6 packs $9.12
7 packs $10.08
8 packs $10.88
9 packs $11.52
10 packs $12
© 2021

УСЛОВНЫЕ КОНСТРУКЦИИ

Условные конструкции проверяют **условие**, чтобы определить, выполнять блок кода или нет. Это все равно что сказать: «Если условие верно, выполни задачу 1, в противном случае (при необходимости) — задачу 2».

Некоторые задачи выполняются только при соблюдении некоторого **условия**. Рассмотрим веб-сайт, на котором пользователи могут авторизоваться.

- Если пользователь вошел в систему, отображается ссылка на страницу его профиля.
- Если пользователь не вошел в систему, отображается ссылка на страницу входа.

Рассмотрим следующее условие: «Пользователь вошел в систему?» От этого условия зависит, какую ссылку отображать.

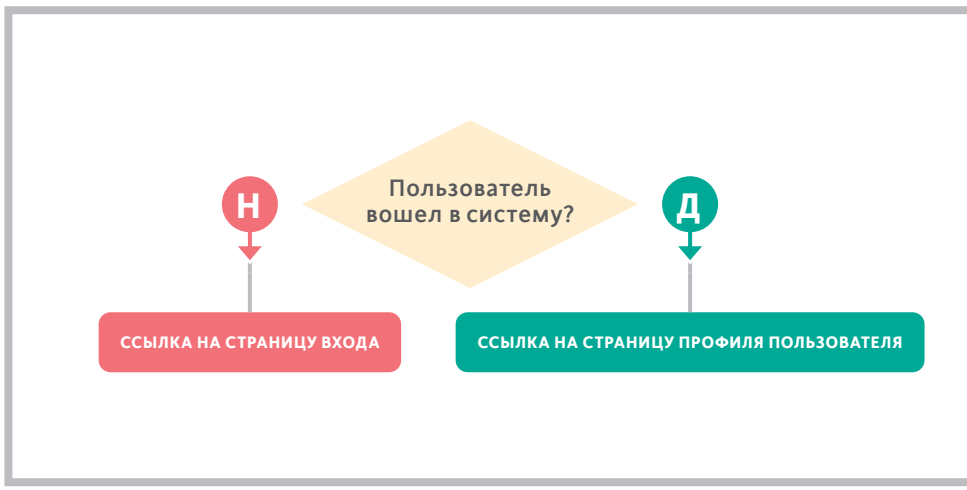
Условия — это выражения, которые всегда приводят к значению `true` (истина) или `false` (ложь). Как правило, в условиях для сравнения двух значений используются операторы сравнения.

Если переменная `$logged_in` будет содержать значение `true`, когда пользователь вошел в систему, и `false` — когда пользователь не вошел в систему, следующее выражение можно будет использовать в качестве условия:

```
($logged_in === true)
```

Если условие приводит к следующему значению:

- `true` — выполняются инструкции, отображающие ссылку на страницу профиля пользователя;
- `false` — выполняются инструкции, отображающие ссылку на страницу регистрации.



КОНСТРУКЦИЯ IF

Стр. 74

Конструкция `if` исполняет набор инструкций только при выполнении условия. Если условие истинно, то будет исполнен следующий за конструкцией `if` блок кода (заключенный в фигурные скобки). Если условие ложно, этот блок кода исполнен не будет, а интерпретатор PHP перейдет к следующей строке кода.

```
if ($logged_in === true) {
    //Исполняемый код (если условие
    //выполняется)
}
```

КОНСТРУКЦИИ IF...ELSE

Стр. 75–77

Конструкция `else` может дополнять конструкцию `if` и позволяет выполнить некоторый код, если условие ложно. Если условие возвращает значение `true`, то выполняется набор инструкций в блоке `if`. В противном случае выполняется набор инструкций в блоке `else`. В дальнейшем вы познакомитесь также с тернарным оператором, который по действию похож на `if...else`, но при этом является не управляющей конструкцией, а оператором — то есть вычисляет значение и возвращает результат.

```
if ($logged_in === true) {
    //Исполняемый код (если условие
    //выполняется)
} else {
    //Исполняемый код (если условие
    //не выполняется)
}
```

КОНСТРУКЦИИ IF... ELSEIF...

Стр. 78

На случай, если условие в конструкции `if` не выполнится, можно добавить еще одно, с помощью конструкции `elseif`. Инструкции, следующие за вторым условием, исполняются, только если не было выполнено первое условие и выполнено второе.

Также с помощью конструкции `else` можно добавить код для выполнения по умолчанию, если ни одно из условий не выполняется.

```
if ($logged_in === true) {
    //Исполняемый код (если условие 1
    //истинно)
} elseif ($time > 12) {
    //Исполняемый код (если условие 1 ложно,
    // а 2 истинно)
} else {
    //Исполняемый код для всех остальных
    //случаев
}
```

КОНСТРУКЦИЯ SWITCH

Стр. 79

Конструкции `switch` не требуется условие. В нее передаются переменная и список значений, с которыми она сравнивается (используя нестрогое сравнение).

Если ни один из вариантов не соответствует, то исполняется набор инструкций по умолчанию (в блоке `default`).

Если совпадений не было и блок для выполнения по умолчанию отсутствует, то интерпретатор PHP переходит к строке, следующей после оператора `switch`.

```
switch ($option) {
    case 'option_1':
        //Выполняемые операторы
        break;
    case 'option_2':
        //Выполняемые операторы
        break;
    default:
        //Выполняемые операторы
}
```

ВЫРАЖЕНИЕ MATCH

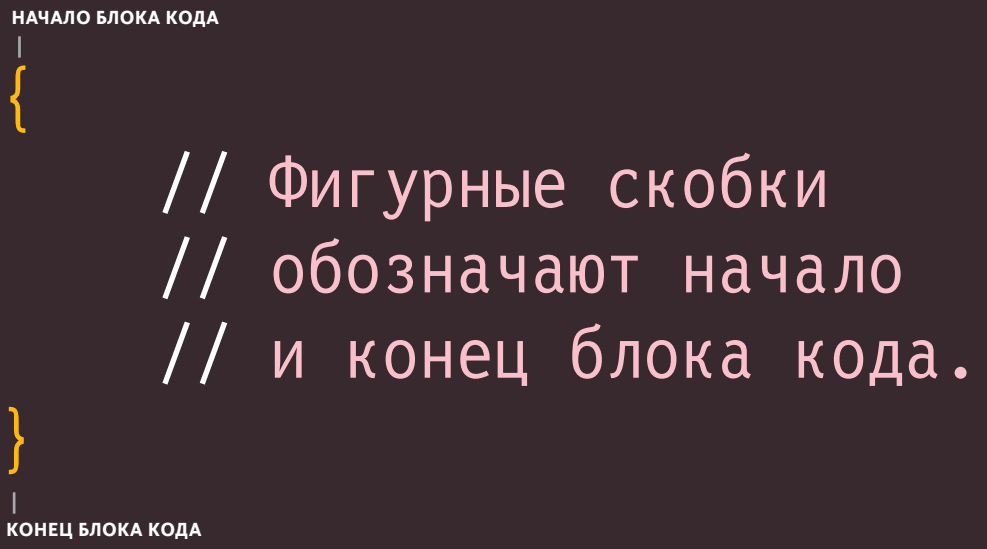
Стр. 80

В PHP 8 было добавлено выражение `match` (похожее на конструкцию `switch`). Если обнаружено точное совпадение для переменной (значение и тип данных), то выражение выполняется и возвращается созданное этим выражением значение. Также вы можете указать в одной строке несколько параметров, а также указать значение по умолчанию, если ни одно значение не совпадет. Однако, если нет совпадения и нет значения по умолчанию, будет вызвано сообщение об ошибке.

```
$result = match($option) {
    'option_1'              =>//Выражение,
    'option_2', 'option_3' =>//Выражение,
    'default'               =>//Выражение,
};
```

ФОРМИРОВАНИЕ БЛОКА КОДА — ФИГУРНЫЕ СКОБКИ

Для группировки логически связанных инструкций в PHP используются фигурные скобки.
Блок кода — это набор логически связанных инструкций, заключенных в фигурные скобки. Причем такой блок также представляет собой инструкцию.



Фигурные скобки указывают интерпретатору PHP начало и конец блока кода.

- левая фигурная скобка обозначает начало блока кода;
- правая фигурная скобка обозначает конец кодового блока.

Количество инструкций внутри фигурных скобок может быть любым.

Блоки кода позволяют интерпретатору PHP исполнять, пропускать или повторять содержащиеся в них инструкции.

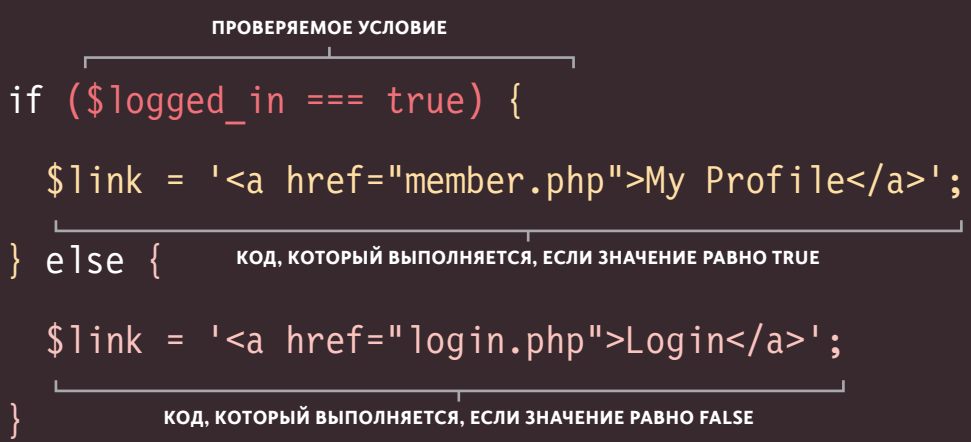
После закрывающей фигурной скобки} в конце блока кода не нужна точка с запятой.

СТРУКТУРА УСЛОВНЫХ КОНСТРУКЦИЙ

Результатом вычисления **условия** всегда является логическое значение `true` или `false`. Затем, в зависимости от результата, выполняется определенный блок кода.

Следующее условие проверяет, равно ли содержимое переменной `$logged_in`: логическому значению `true`:

- если условие выполняется, оно вернет логическое значение `true`;
- если условие не выполняется, оно вернет логическое значение `false`.



Когда результат условного выражения равен `true`, выполняется первый блок кода. Затем интерпретатор PHP игнорирует ключевое слово `else` и пропускает второй блок кода. После этого интерпретатор PHP переходит к строке кода, следующей за условным оператором.

Когда результат условного выражения равен `false`, интерпретатор PHP пропускает первый блок кода и переходит к ключевому слову `else`, выполняя операторы в следующем за ним блоке. После этого интерпретатор переходит к первой строке кода, следующей после условного оператора.

ИСПОЛЬЗОВАНИЕ КОНСТРУКЦИИ IF

В следующем примере при входе пользователя в систему отображается настраиваемое под него приветствие. Сначала создадим две переменные и присвоим им значения.

1. Переменная `$name` содержит имя пользователя.
2. Также **инициализируем** переменную `$greeting`. Присвоим этой переменной начальное значение, которое будет использоваться, если оно не будет изменено в шагах 3 и 4. Этим значением будет сообщение `Hello`.
3. В конструкции `if` используется условие для проверки, не является ли значение переменной `$name` пустой строкой.

Если переменная `$name` — не пустая строка, выполняется следующий блок кода.

4. Значение переменной `$greeting` меняется на другое, и на странице отобразится сообщение `Welcome back`, за которым следует имя пользователя.
5. Значение переменной `$greeting` будет отображаться на странице.

Упражнение. В шаге 1 измените значение переменной `$name` на пустую строку. Обновите страницу, и в приветствии будет отображено `Hello`.

section_a/c02/if-statement.php — PHP

```php
<?php
(1) $name     = 'Ivy';
(2) $greeting = 'Hello';

(3) if ($name !== '') {
(4)     $greeting = 'Welcome back, ' . $name;
}
?>
<!DOCTYPE html>
<html>
 <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
(5)    <h2><?= $greeting ?></h2>
  </body>
</html>
```

РЕЗУЛЬТАТ



Примечание. Условие может содержать только имя переменной, например:

```php
if ($name) {
    $greeting = 'Hi, '.
$name;
}
```

Данное условие проверяет, будет ли значение переменной `$name` рассматриваться как значение `true` после преобразования в логический тип данных.

Строка, содержащая любой текст или число, отличное от 0, будет трактоваться как значение `true`.

ИСПОЛЬЗОВАНИЕ КОНСТРУКЦИЙ IF...ELSE

PHP section_a/c02/if-else-statement.php

```
<?php
1 $stock = 5;

2 if ($stock > 0) {
3     $message = 'In stock';
4 } else {
5     $message = 'Sold out';
}
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Chocolate</h2>
6   <p><?= $message ?></p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Рассмотрим следующий пример, в котором необходимо проверить уровень запасов товара и отобразить соответствующее сообщение.

1. Переменная `$stock` — количество товаров на складе.
2. В конструкции оператора `if` используется условие, которое проверяет, превышает ли значение переменной `$stock` значение 0.
3. Если условие приводит к результату `true`, переменной `$message` присваивается значение `In stock` (В наличии). Затем интерпретатор PHP пропускает ключевое слово `else` и последующий блок кода.
4. Если в результате выполнения условия в шаге 2 возвращается значение `false`, интерпретатор PHP выполняет блок кода, следующий за ключевым словом `else`.
5. Переменной `$message` присвоится значение `Sold out` (Продано).
6. Значение переменной `$message` выводится на странице.

Упражнение. В шаге 1 измените значение переменной `$stock` на значение 0.

В шаге 5 измените сообщение на `More stock coming soon` (Скоро в продаже).

ТЕРНАРНЫЙ ОПЕРАТОР

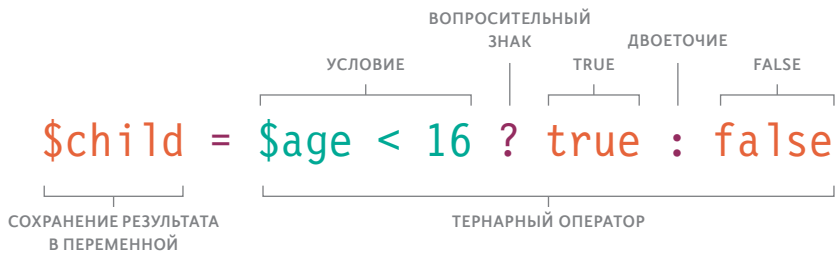
Тернарный оператор проверяет результат условного выражения, а затем, если условие вернет значение `true`, возвращает одно значение, а если `false` — другое. Обычно применяется как сокращенная форма записи конструкций `if…else`, если они используются для присвоения значения переменной.

В приведенных справа конструкциях `if…else` используется условное выражение, которое проверяет, меньше ли возраст пользователя, чем 16 лет. Если результат условного выражения равен:

- `true`, тогда переменной `$child` присваивается значение `true`;
- `false`, тогда переменной `$child` присваивается значение `false`.

Ниже рассмотрим пример работы тернарного оператора, который получит тот же результат, используя всего одну строку кода.

```
if ($age < 16) {
    $child = true;
} else {
    $child = false;
}
```



Вопросительный знак отделяет проверяемое условие от возвращаемых значений.

Двоеточие разделяет значение, возвращаемое, если результат условия равен `true`, от значения, возвращаемого, если результат условия равен `false`.

Затем результат, возвращаемый тернарным оператором, сохраняется в переменной `$child`.

Иногда условие заключается в круглые скобки для улучшения читабельности (см. пример далее), однако использование круглых скобок необязательно[7].

ИСПОЛЬЗОВАНИЕ ТЕРНАРНОГО ОПЕРАТОРА

PHP section_a/c02/ternary-operator.php

```php
<?php
① $stock = 5;

② $message = ($stock > 0) ? 'In stock' : 'Sold out';
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Chocolate</h2>
③   <p><?= $message ?></p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Следующий пример аналогичен предыдущему, но с использованием тернарного оператора (вместо конструкций `if…else`).

1. Переменная `$stock` — количество товаров, имеющихся на складе.
2. Тернарный оператор используется для присвоения значения переменной `$message`. Условное выражение проверяет, превышает ли значение переменной `$stock` значение 0. Если значение условного выражения равно:

- `true`, тогда переменной `$message` присваивается значение `In stock` (В наличии);
- `false`, тогда переменной `$message` присваивается значение `Sold out` (Продано).

3. Значение переменной `$message` выводится на странице.

Упражнение. В шаге 1 измените значение переменной `$stock` на значение 0.

В шаге 2 измените сообщение на `More stock coming soon` (Скоро в продаже).

ИСПОЛЬЗОВАНИЕ КОНСТРУКЦИЙ IF... ELSEIF...

Следующий пример основан на предыдущих. В нем отображаются различные сообщения в зависимости от того, есть ли товар на сладе, ожидается ли доставка на склад новой партии или товара нет совсем.

1. В переменной `$ordered` содержится количество товара, доставка которого на склад уже заказана.
2. В конструкции `if` используется условие, которое проверяет, превышает ли значение переменной `$stock` значение 0. Если превышает, то переменной `$message` будет присвоено значение `In stock` (В наличии), а интерпретатор PHP переместится в конец конструкций `if...elseif...else`.
3. Если первое условие не выполнится, то будет проверяться условие в конструкции `elseif` (значение переменной `$ordered` больше 0). Если это условие выполняется, переменной `$message` будет присвоено значение `Coming soon` (Товар скоро будет), а интерпретатор PHP переместится в конец конструкций `if...elseif...else`.
4. Если ни одно из условий не вернуло значение `true`, интерпретатор PHP перейдет к конструкции `else` и выполнит следующий за ней блок кода, в котором переменной `$message` будет присвоено значение `Sold out` (Продано).
5. Вывод переменной `$message`.

section_a/c02/if-else-if-statement.php **PHP**

```
<?php
$stock   = 5;
$ordered = 3;

if ($stock > 0) {
    $message = 'In stock';
} elseif ($ordered > 0) {
    $message = 'Coming soon';
} else {
    $message = 'Sold out';
}
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Chocolate</h2>
    <p><?= $message ?></p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 измените значение переменной `$stock` на значение 0. После обновления страницы появится сообщение `Coming soon` (Товар скоро будет).

ИСПОЛЬЗОВАНИЕ КОНСТРУКЦИИ SWITCH

PHP section_a/c02/switch-statement.php

```
<?php
(1) $day = 'Monday';

(2) switch ($day) {
(3)     case 'Monday':
(4)         $offer = '20% off chocolates';
(5)         break;
(3)     case 'Tuesday':
(4)         $offer = '20% off mints';
(5)         break;
(6)     default:
            $offer = 'Buy three packs, get one free';
    }
    ?>
    <!DOCTYPE html>
    <html>
      <head> ... </head>
      <body>
        <h1>The Candy Store</h1>
        <h2>Offers on <?= $day; ?></h2>
(7)     <p><?= $offer ?></p>
      </body>
    </html>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 измените значение переменной на Wednesday (Среда). После выполнения шага 5 добавьте в оператор switch вариант для значения Wednesday (Среда).

1. Переменная $day — значение дня недели.
2. Конструкция switch начинается с команды switch и имени переменной, заключенной в круглые скобки. Переменная является так называемым **значением-переключателем**.
Затем внутри фигурных скобок следует набор вариантов ветвления, где каждый параметр сравнивается со значением-переключателем (используя нестрогое сравнение).
3. Здесь показано два варианта ветвления. В обоих:

- блок начинается с ключевого слова case;
- за ним следует значение параметра;
- затем двоеточие.

4. Если значение-переключатель совпадает с параметром, то выполняются все следующие за ним инструкции. Здесь это присвоение значения переменной $offer.
5. После того как совпадение будет найдено, интерпретатор PHP начнет исполнять все инструкции до конца блока switch либо до тех пор, пока не обнаружит оператор break.
6. Последним идет вариант, обозначенный командой default, он выполняется по умолчанию. За ним следуют инструкции, которые исполняются, если ни в одном из вариантов ветвления параметр не совпал со значением-переключателем. После параметра по умолчанию оператор break не пишется.
7. Вывод значения переменной $offer.

ИСПОЛЬЗОВАНИЕ ВЫРАЖЕНИЯ MATCH

Примечание. Этот пример работает только для версий PHP 8 и выше.

section_a/c02/match.php — PHP

```php
<?php
$day = 'Monday';

$offer = match($day) {
    'Monday'             => '20% off chocolates',
    'Saturday', 'Sunday' => '20% off mints',
    default              => '10% off your entire order',
}
?>
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Offers on <?= $day ?></h2>
    <p><?= $offer ?></p>
  </body>
</html>
```

1. Переменная `$day` содержит значение дня недели.
2. Выражение `match` используется для присвоения значения переменной `$offer`. Код начинается с ключевого слова `match`, за которым следуют круглые скобки, содержащие имя проверяемой переменной, а затем открывающая фигурная скобка.
3. Выражение `match` предназначено для выбора одного из вариантов на основании проверки совпадения значения с заданным условием. Внутри фигурных скобок содержится несколько вариантов выбора, каждый из которых начинается со значения, которое будет проверяться на соответствие значению переменной `$day`.
Если совпадение найдено, то выполняется выражение, расположенное справа от оператора двойной стрелки =>.
Каждый вариант содержит только одно выражение и заканчивается запятой.
Обратите внимание, что в выражении `match` используется строгое сравнение типов. Различные манипуляции с типами не выполняются.
4. Последним идет условие с ключевым словом `default`, которое используется по умолчанию. Его выражение выполнится, если совпадение не было обнаружено выше. Если условие по умолчанию отсутствует и при этом совпадение не было найдено, то возникает ошибка.
5. Вывод значения переменной `$offer`.

РЕЗУЛЬТАТ



Упражнение. В шаге 1 измените значение переменной на `Tuesday` (Вторник). В шаге 3 добавьте предложение к выражению `match` для значения `Tuesday` (Вторник).

Упражнение. В шаге 1 измените значение переменной на `Wednesday` (Среда). Затем удалите параметр по умолчанию. Будет выведено сообщение об ошибке.

ЦИКЛЫ

Цикл — это управляющая конструкция, позволяющая выполнить блок кода либо определенное количество раз, либо пока выполняется определенное условие.

Например, если вы хотите попросить человека выполнить какое-то задание 10 раз, то вы не будете десять раз писать одну и ту же инструкцию — вы можете написать задание один раз, а затем попросить выполнить его десять раз. В PHP циклы позволяют:

- создать набор инструкций, которые выполняют необходимую задачу один раз, и заключить его в фигурные скобки;
- добавить условие, которое определяет, выполнять эти инструкции или нет (так же, как это делает конструкция `if`).

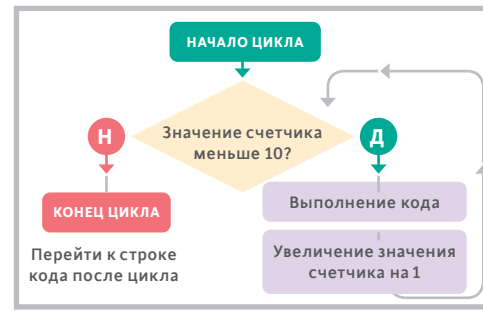
Чтобы выполнить задачу десять раз, можно использовать переменную-счетчик, которой надо присвоить начальное значение 0, а затем будет происходить следующее.

1. Выполняется проверка, что значение счетчика меньше 10.
2. Если это так, то будут выполнены инструкции в блоке кода.
3. Значение счетчика увеличивается на 1.
4. Интерпретатор PHP возвращается к шагу 1.

Если условие возвращает значение `true`, блок кода выполняется, а если условие возвращает значение `false` — то прекращается выполнение.

- После выполнения блока инструкций условие снова проверяется. Если условие возвращает значение `true`, выполнение блока кода повторяется, а затем условие снова проверяется.

Когда условие возвращает значение `false`, интерпретатор переходит к строке кода, следующей после цикла.



ЦИКЛ WHILE

Цикл `while` исполняет код, пока выполняется условие в круглых скобках.

ЦИКЛ DO WHILE

Цикл `do…while` работает аналогично циклу `while`, но позволяет выполнить действие минимум один раз, даже если условие в `while` не выполняется (возвращает значение `false`), поскольку условие проверяется не до, а после выполнения блока кода.

ЦИКЛ FOR

Цикл `for` позволяет повторять блок кода определенное количество раз. Помимо условия, в нем пишутся инструкции, которые создают счетчик и обновляют его каждый раз при прохождении цикла.

ЦИКЛ FOREACH

Цикл `foreach` представляет собой перебор элементов массива, повторяя для каждого из них один и тот же набор инструкций. Цикл `foreach` также может работать со свойствами объектов, которые вы изучите в главе 4.

ЦИКЛ WHILE

Цикл `while` исполняет код, пока выполняется условие в круглых скобках. Если условие возвращает значение `true`, блок кода выполняется. Затем условие проверяется снова. Если условие возвращает значение `true`, снова выполняется блок кода. Цикл будет повторяться до тех пор, пока условие не вернет значение `false`.

СИНТАКСИС ЦИКЛА

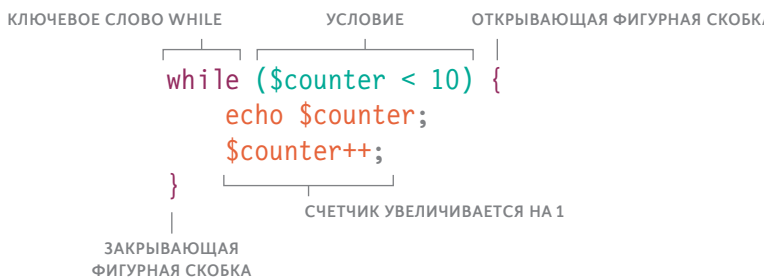
Все циклы начинаются с ключевого слова, указывающего интерпретатору PHP, какой цикл используется. Цикл while начинается с ключевого слова `while`.

ПРОВЕРКА УСЛОВИЯ

Условие проверяет значения в коде. В примере ниже условие проверяет, что значение переменной `$counter` меньше 10. Если условие истинно (возвращает значение `true`), выполняются операторы, заключенные в фигурные скобки.

ВЫПОЛНЯЕМЫЕ ИНСТРУКЦИИ

Блок кода, выполнение которого необходимо повторить несколько раз, помещается в фигурные скобки. Цикл повторяет (итерирует) код в фигурных скобках, пока условие не вернет значение `false`.



Пример выше демонстрирует следующее: пока значение переменной `$counter` меньше 10, необходимо повторить выполнение инструкций, заключенных в фигурные скобки.

Код, заключенный в фигурные скобки:

- выводит значение, хранящееся в переменной `$counter`;
- оператор ++ увеличивает значение переменной `$counter` на 1.

Например, если бы при запуске данного кода переменная `$counter` содержала значение 1, в результате отобразилось бы 123456789 (поскольку при выполнении кода значение переменной `$counter` выводится, пока не достигнет значения 10).

ИСПОЛЬЗОВАНИЕ ЦИКЛА WHILE

PHP section_a/c02/while-loop.php

```
<?php
1 $counter = 1;
2 $packs   = 5;
3 $price   = 1.99;
?>
...
<h2>Prices for Multiple Packs</h2>
<p>
  <?php
4   while ($counter <= $packs) {
5       echo $counter;
6       echo ' packs cost $';
7       echo $price * $counter;
8       echo '<br>';
9       $counter++;
    }
  ?>
</p>
```

РЕЗУЛЬТАТ



Рассмотрим следующий пример — он демонстрирует отображение стоимости нескольких упаковок конфет.

1. Переменной `$counter` присвоим значение 1.
2. Переменной `$packs` — количество упаковок.
3. Переменной `$price` — цену одной упаковки.
4. Цикл `while` начинается с определения условия, которое проверяет, меньше значение переменной `$counter` или равно значению переменной `$packs`. Если условие выполняется, выполняется код, заключенный в фигурные скобки.
5. Выводится значение счетчика.
6. Выводится текст: `packs cost $`.
7. Выводится результат умножения переменной `$price` на значение переменной `$counter`.
8. Добавляется перевод строки.
9. Используя оператор инкремента, значение переменной `$counter` увеличивается на 1.

После выполнения шага 9 интерпретатор PHP снова проверяет условие, определенное в шаге 4. Процесс будет повторяться, пока это условие не вернет значение `false`.

Упражнение. В шаге 2 увеличьте количество пачек до 10.

Упражнение. В шаге 4 измените оператор <= на <.

ЦИКЛ DO WHILE

Цикл `do while` перед проверкой условия запускает набор операторов, заключенный в фигурные скобки. Цикл всегда выполняется по крайней мере один раз.

СИНТАКСИС ЦИКЛА

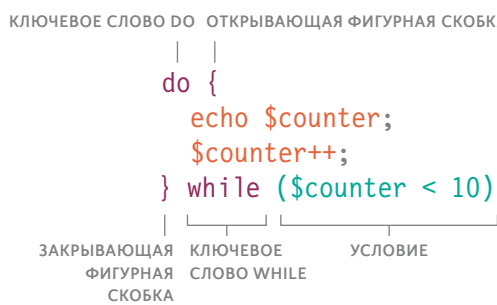
Цикл `do while` начинается с ключевого слова `do`. Ключевое слово `while` следует после закрывающей фигурной скобки, содержащей выполняемые инструкции.

ВЫПОЛНЯЕМЫЕ ИНСТРУКЦИИ

Код, который необходимо повторить, заключается в фигурные скобки. Данный блок кода выполняется хотя бы один раз, так как условие проверяется после фигурных скобок.

ПРОВЕРКА УСЛОВИЯ

Условие проверяет текущие значения в коде. Если условие возвращает значение `true`, интерпретатор PHP возвращается к началу цикла и повторяет блок кода снова.



Код в этом цикле выводит значение переменной `$counter`, а затем, используя оператор `++`, значение переменной `$counter` увеличивается на 1.

Поскольку блок кода выполняется перед условием, то вывод переменной `$counter` и увеличение ее на 1 произойдет по крайней мере один раз.

Если значение переменной `$counter` равно 3, то в результате мы получим 3456789. Если значение переменной `$counter` равно 1, то в результате мы получим 123456789.

ИСПОЛЬЗОВАНИЕ ЦИКЛА DO WHILE

PHP section_a/c02/do-while-loop.php

```
<?php
$packs = 5;
$price = 1.99;
?>
...
<h2>Prices for Multiple Packs</h2>
<p>
  <?php
  do {
      echo $packs;
      echo ' packs cost $';
      echo $price * $packs;
      echo '<br>';
      $packs--;
  } while ($packs > 0);
  ?>
</p>
```

РЕЗУЛЬТАТ



В данном примере заключенный в фигурные скобки код запускается до проверки условия, поэтому блок кода выполняется один раз, даже если условие возвращает значение `false`.

1. Объявим и проинициализируем две переменные. Переменная `$packs` — количество упаковок конфет. Переменная `$price` — цена одной упаковки.
2. Цикл `do while` начинается с ключевого слова `do` и открывающей фигурной скобки. Блок кода находится перед условием, поэтому он выполняется хотя бы один раз независимо от того, выполняется условие или нет.
3. Выводится количество упаковок (значение переменной `$packs`), за которым будет отображаться следующий текст: `packs cost $`.
4. Стоимость выводится путем умножения значения переменной `$packs` на значение переменной `$price`. После этого следует перевод строки.
5. При использовании оператора декремента `–`, значение переменной `$packs` уменьшается на 1.
6. Блок кода заканчивается закрывающей фигурной скобкой. Затем следует ключевое слово `while` и условие. Условие проверяет, превышает ли значение переменной `$packs` значение 0.

Упражнение. В шаге 1 измените значение переменной `$packs` на значение 10 и измените значение переменной `$price` на значение `2,99`.

Примечание. Будьте внимательны: после закрывающей фигурной скобки (перед ключевым словом `while`) точка с запятой не ставится, но ставится после условия.

ЦИКЛ FOR

Цикл `for` повторяет блок кода определенное количество раз. Для этого цикл создает счетчик и обновляет его каждый раз при запуске цикла.

СИНТАКСИС ЦИКЛА

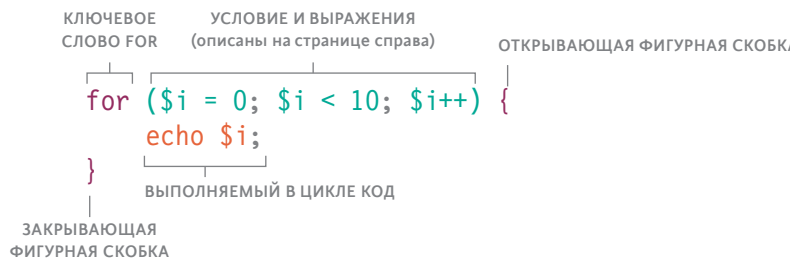
Цикл `for` начинается с ключевого слова `for`.

ПРОВЕРКА УСЛОВИЯ

В цикле `for` условие пишется рядом с кодом, который создает и обновляет счетчик. Подробнее об этом написано на странице справа.

ВЫПОЛНЯЕМЫЕ ИНСТРУКЦИИ

Код, выполняющий задачу, которую необходимо повторить, заключается в фигурные скобки. Этот блок кода выполняется заданное количество раз.



Имя переменной `$i` или `$index`, как правило, применяется для счетчиков.

Код, заключенный в фигурные скобки, выводит значение переменной `$i`.

В результате отобразится следующее: 0123456789.

ТРИ ВЫРАЖЕНИЯ ЦИКЛА FOR

Цикл `for` содержит два дополнительных выражения помимо условия. Первое создает счетчик, а другое обновляет его.

ВЫРАЖЕНИЕ 1
ИНИЦИАЛИЗИРУЮЩЕЕ ВЫРАЖЕНИЕ

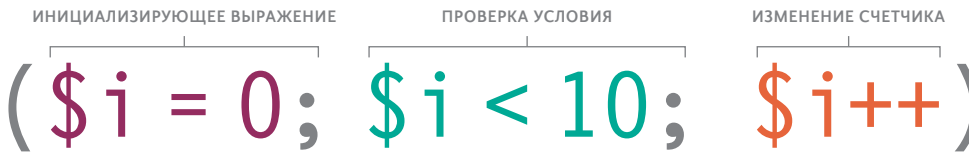
Первое выражение всегда выполняется только один раз в начале цикла. На этом этапе создается переменная-счетчик, которой присваивается значение.

ВЫРАЖЕНИЕ 2
ПРОВЕРКА УСЛОВИЯ

Второе выражение — это условие. Операторы в цикле `for` будут повторяться, пока условие не вернет значение `false`.

ВЫРАЖЕНИЕ 3
ИЗМЕНЕНИЕ СЧЕТЧИКА

После выполнения блока кода в фигурных скобках выполняется третье выражение, которое используется для изменения значения счетчика[8].



В приведенном выше примере переменная `$i` используется как счетчик. Ей присваивается начальное значение 0.

Условие проверяет, меньше ли значение переменной `$i` значения 10. Пока условие истинно, выполняются инструкции в блоке кода.

Вместо числа 10 можно использовать переменную, которая содержит нужное значение, например `$i < $max;`.

При каждом обороте цикла счетчик обновляется, используя оператор инкремента ++ для увеличения счетчика (переменная `$i`) на 1.

ИСПОЛЬЗОВАНИЕ ЦИКЛА FOR

В данном примере цикл `for` выполняет операцию десять раз.

1. Переменная `$price` — стоимость одной упаковки конфет.
2. Ключевое слово `for` определяет тип цикла, за которым в круглых скобках следуют три выражения.

- Первое выражение: `$i = 1;` Инициализируется счетчик со значением 1.
- Второе выражение: `$i <= 10;` Данное условие означает, что код должен повторяться, пока значение счетчика меньше или равно 10.
- Третье выражение: `$i++` При каждом обороте цикла значение счетчика увеличивается на 1.

3. Код, выполняемый при каждой итерации цикла, заключается в фигурные скобки. Как и ранее, в этом примере отображается количество упаковок (значение счетчика) и цена (значение счетчика, умноженное на значение переменной `$price`).

После выполнения кода, заключенного в фигурные скобки, третье выражение (шаг 2) обновляет счетчик, увеличивая значение переменной `$i` на 1.

section_a/c02/for-loop.php **PHP**

```
<?php
① $price = 1.99;
?>
...
<h2>Prices for Multiple Packs</h2>
<p>
  <?php
② for ($i = 1; $i <= 10; $i++) {
      echo $i;
      echo ' packs cost $';
③     echo $price * $i;
      echo '<br>';
  }
  ?>
</p>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 установите значение цены равным `2,99`.

Упражнение. В шаге 2 повторите цикл 20 раз.

PHP section_a/c02/for-loop-higher-counter.php

```php
<?php
$price = 1.99;
?>
...
<h2>Prices for Large Orders</h2>
<p>
  <?php
  for ($i = 10; $i <= 100; $i = $i + 10) {
      echo $i;
      echo ' packs cost $';
      echo $price * $i;
      echo '<br>';
  }
  ?>
</p>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 2 обновите условное выражение, чтобы отобразить цены до 200 упаковок конфет.

В данном примере выводятся цены при покупке нескольких упаковок конфет.

1. Переменная `$price` — стоимость одной упаковки конфет.
2. Ключевое слово `for` определяет тип цикла, за которым в круглых скобках следуют три выражения.

- Первое выражение:
  `$i = 10;`
  Инициализируется счетчик со значением 10.

- Второе выражение:
  `$i <= 100;`
  Это условие означает, что код должен повторяться, пока значение счетчика меньше или равно 100.

- Третье выражение:
  `$i = $i + 10`
  При каждом запуске цикла значение счетчика увеличивается на 10.

3. Код, выполняемый при каждой итерации цикла, заключается в фигурные скобки. Данный блок кода выполняется так же, как и в предыдущем примере.

ЦИКЛ FOREACH

Цикл `foreach` используется для работы с составными типами данных, такими как массивы. Он перебирает все элементы массива по одному и выполняет для каждого элемента один и тот же блок кода.

Составные типы данных, такие как массивы, содержат набор упорядоченных элементов. Каждый элемент состоит из пары «ключ — значение» (key/value). В ассоциативном массиве ключ — это строка. В индексированном массиве ключ — это порядковый номер.

Цикл `foreach` перебирает все элементы массива по одному, пока не будет достигнут последний, а затем управление передается следующим за циклом коду.

При каждом выполнении блока кода можно получить доступ к ключу и значению текущего элемента в массиве и использовать их внутри блока кода. Для этого в скобках после ключевого слова `foreach` указывается следующее:

- имя переменной, содержащей массив;
- имя переменной для текущего ключа;
- имя переменной для текущего значения.

УКАЗАНИЕ ТИПА ЦИКЛА

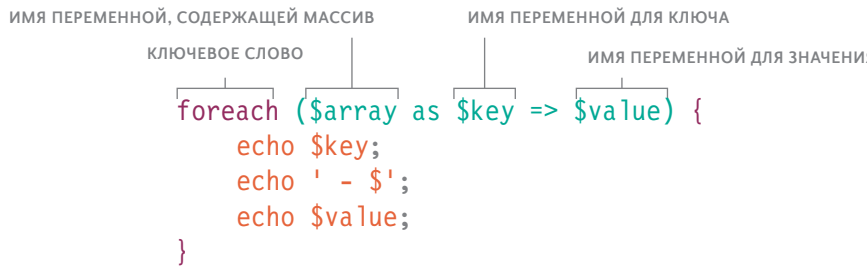
Цикл `foreach` начинается с ключевого слова `foreach`.

ИМЕНА ПЕРЕМЕННЫХ

В скобках содержится три переменных (более подробно описано на странице справа).

ВЫПОЛНЯЕМЫЕ ИНСТРУКЦИИ

Инструкции, выполняющие задачу, которую необходимо повторять, заключены в фигурные скобки.



Внутри фигурных скобок находится три инструкции, которые выводят:

- ключ;
- тире и символ доллара;
- значение элемента.

Если необходимо использовать только значения массива, то ключ можно опустить.

```
foreach ($array as
$value) {
    //Ваш код
}
```

Когда выполнится обход всех элементов массива и будет достигнут последний, интерпретатор PHP после выполнения цикла переходит к следующей строке кода.

Ниже представлен массив `$products`, который содержит названия товаров и их цену.

Используя цикл `foreach`, можно отображать название и цену для каждого элемента.

Цикл работает независимо от количества элементов в массиве.

```
$products = ['toffee' => 2.99, 'mints' => 1.99, 'fudge' => 3.40,];
```

ПЕРЕМЕННАЯ — КЛЮЧ — ЗНАЧЕНИЕ — КЛЮЧ — VALUE — КЛЮЧ — ЗНАЧЕНИЕ

В примере ниже цикл начинается с ключевого слова `foreach`.

Затем внутри скобок пишется следующее:

- переменная `$products` — имя переменной, содержащей массив;
- далее следует ключевое слово `as`;
- переменная `$item` — имя переменной, содержащей **ключ** текущего элемента в массиве;
- далее следует оператор двойной стрелки;
- переменная `$price` — имя переменной, содержащей **значение** текущего элемента в массиве.

В блоке кода переменные `$item` и `$price` будут использоваться для представления ключа и значения текущего элемента в массиве. Сначала будет отображаться название товара (переменная `$item`), затем ставится знак доллара и тире, а затем будет отображаться цена (переменная `$price`).

КЛЮЧЕВОЕ СЛОВО AS — ОПЕРАТОР ДВОЙНОЙ СТРЕЛКИ

ПЕРЕМЕННАЯ, СОДЕРЖАЩАЯ МАССИВ — ИМЯ ПЕРЕМЕННОЙ ДЛЯ КЛЮЧА — ИМЯ ПЕРЕМЕННОЙ ДЛЯ ЗНАЧЕНИЯ

```
foreach ($products as $item => $price) {
    echo $item;
    echo ' - $';
    echo $price;
}
```

Часто циклы используются в той части страницы, которая генерирует HTML (см. следующую страницу).

Если это так, первая и последняя строки указанного выше цикла пишутся в отдельных блоках PHP.

Затем данные, содержащиеся в ключах и значениях массива, можно вывести, используя сокращенную запись `echo`.

```
<?php foreach ($products as $item => $price) { ?>
  <li>
    <b><?= $item ?></b> - $<?= $price ?>
  </li>
<?php } ?>
```

ОБХОД ПО КЛЮЧАМ И ЗНАЧЕНИЯМ

Рассмотрим пример отображения таблицы с названиями и ценами на товары, хранящимися в массиве.

1. Переменная `$products` содержит ассоциативный массив со списком товаров и их ценами.
2. В части страницы, содержащей код HTML, прописываются заголовок и начало HTML-таблицы.
3. Создадим цикл `foreach`. Далее в скобках содержится:

- переменная `$products` — имя переменной, содержащей массив;
- ключевое слово `as`;
- переменная `$item` — имя переменной, используемое для ключа текущего элемента массива;
- символ двойной стрелки;
- переменная `$price` — имя переменной, используемое для значения текущего элемента массива.

Затем следует открывающая фигурная скобка, обозначающая начало блока кода.

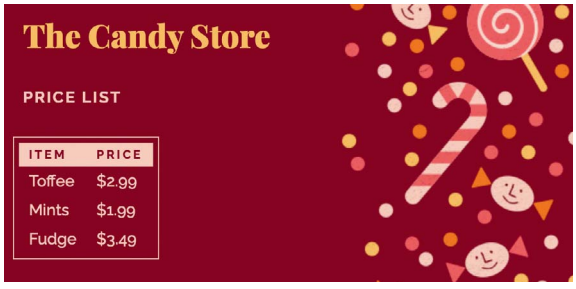
4. Для каждого элемента массива выводится строка таблицы с указанием названия и цены.
5. Закрываем блок кода.

Упражнение. В шаге 1 добавьте в массив два или более элементов.

section_a/c02/foreach-loop.php — PHP

```
<?php
$products = [
    'Toffee' => 2.99,
    'Mints'  => 1.99,
    'Fudge'  => 3.49,
];
?>
...
<h2>Price List</h2>
<table>
  <tr>
    <th>Item</th>
    <th>Price</th>
  </tr>
  <?php foreach ($products as $item => $price) { ?>
    <tr>
      <td><?= $item ?></td>
      <td>$<?= $price ?></td>
    </tr>
  <?php } ?>
</table> ...
```

РЕЗУЛЬТАТ



PHP section_a/c02/foreach-loop-just-accessing-values.php

```php
<?php
① $best_sellers = ['Toffee', 'Mints', 'Fudge',];
?>
...
<h2>Best Sellers</h2>
② <?php foreach ($best_sellers as $product) { ?>
③   <p><?= $product ?></p>
④ <?php } ?>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 добавьте в массив еще два элемента. Затем на шагах 2 и 3 измените имя переменной `$product` на `$candy`.

В индексированном массиве номера индексов соответствуют порядку элементов в массиве. Цикл `foreach` можно использовать, чтобы вывести значения в том порядке, в котором они расположены в массиве.

1. Переменная `$best_sellers` содержит индексированный массив, представляющий самые продаваемые товары.
2. Используя цикл `foreach`, выведем самые продаваемые товары. Далее в скобках пишется:

- переменная `$best_sellers` — имя переменной, содержащей массив;
- ключевое слово `as`;
- переменная `$product` — имя переменной, которая будет использоваться для представления значения текущего элемента в массиве;
- для имени ключа (порядкового номера) переменная отсутствует, поскольку он не используется в блоке кода.

Затем следует открывающая фигурная скобка, начинающая блок кода.

3. Значение текущего элемента отображается в тегах <p>.
4. Закрываем блок кода.

ПОДКЛЮЧЕНИЕ ФАЙЛОВ ДЛЯ ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ КОДА

Большинству веб-сайтов необходимо повторять один и тот же код на нескольких страницах. Например, верхняя и нижняя часть оформления (часто их называют хидер и футер[9]), как правило, на каждой странице одинаковы. **Подключение файлов** позволяет избежать необходимости повторять один и тот же код в нескольких файлах.

Чтобы не дублировать код заголовка сайта на каждой странице, вы можете выполнить следующее.

- Поместите код верхней части оформления сайта в отдельный PHP-файл, известный как подключаемый файл.
- Для подключения файла необходимо использовать конструкцию `include`.

При обнаружении конструкции `include` интерпретатор PHP считывает содержимое подключаемого файла и выполняет содержащийся в нем код.

В примере ниже слева файл с именем `candy.php` подключает следующие два файла:

- файл `header.php` содержит заголовок сайта;
- файл `footer.php` содержит нижнюю часть оформления сайта.

Между двумя конструкциями `include` размещается основное содержимое страницы. Преимущества использования подключаемых файлов:

- предотвращается дублирование или повторение одного и того же кода;
- значительно упрощается сопровождение кода, поскольку при изменении подключаемого файла обновляются сразу все страницы, которые его используют.

candy.php

```
<?php include 'includes/header.php'; ?>

<h1>The Candy Store</h1>
<h2>Welcome</h2>
<p>A wide selection of delicious candy
   handmade in our kitchen...</p>

<?php include 'includes/footer.php'; ?>
```

includes/header.php

```
<h1>The Candy Store</h1>
<nav>
  <a href="index.php">Home</a> |
  <a href="candy.php">Candy</a> |
  <a href="about.php">About</a> |
  <a href="contact.php">Contact</a>
</nav>
```

includes/footer.php

```
<footer>
  &copy; <?php echo date('Y')?>
</footer>
```

КОМАНДЫ, ИСПОЛЬЗУЕМЫЕ ДЛЯ ПОДКЛЮЧЕНИЯ ФАЙЛОВ

Для добавления кода из подключаемого файла используются четыре разные управляющие конструкции.
Они работают немного по-разному, однако используют один и тот же синтаксис.

Конструкция `include` подключает файл и обрабатывает его так, как если бы содержимое этого файла было записано на месте оператора `include`.

После ключевого слова `include` следует путь к файлу. Иногда имя файла пишут внутри круглых скобок, но скобки необязательны. Подключаемый файл обычно имеет расширение.php.

КОНСТРУКЦИЯ INCLUDE

```
<?php include 'includes/filename.php'; ?>
```

ОТНОСИТЕЛЬНЫЙ ПУТЬ К ФАЙЛУ

КОНСТРУКЦИИ INCLUDE И REQUIRE

Конструкции `include` и `require` подключают и выполняют указанный файл.

Разница между ними заключается в поведении интерпретатора PHP в случае, если подключаемый файл не может быть найден или прочитан:

- `include`: выдается предупреждение, затем обрабатывается остальная часть кода;
- `require`: генерируется ошибка и действие программы прекращается.

КОНСТРУКЦИИ INCLUDE_ONCE И REQUIRE_ONCE

Конструкции `include_once` и `require_once` работают аналогично конструкциям `include` и `require`, но подключают файл только один раз.

После подключения файла с использованием конструкции `include_once` или `require_once`, даже если в дальнейшем для подключения того же файла будут использоваться те же операторы, второй раз файл подключен не будет.

Интерпретатор PHP тратит лишние ресурсы, чтобы проверить, был ли подключен файл, поэтому данные конструкции следует использовать только в случае риска дублирования[10].

СОЗДАНИЕ ПОДКЛЮЧАЕМЫХ ФАЙЛОВ

На этой странице рассматривается создание двух подключаемых файлов:

- `header.php` содержит начальную разметку HTML, имя сайта и элементы навигации, которые отображаются в верхней части каждой страницы сайта;
- `footer.php` содержит уведомление об авторских правах с указанием текущего года и закрывающего HTML-кода для каждой страницы.

Оба файла имеют расширение .php.

Подключаемые файлы часто хранятся в папке с именем `includes` (как показано в примере).

Как показано в примере, при необходимости обновить навигацию нужно будет обновить только код файла `header.php`. Затем автоматически обновится каждая страница, подключающая этот файл.

Примечание. Ссылки в файле `header.php` используются для демонстрации создания панели навигации. Код в загружаемых примерах не содержит соответствующих страниц для каждой ссылки.

section_a/c02/includes/header.php — PHP

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Candy Store</title>
    <link rel="stylesheet" href="css/styles.css" />
  </head>
  <body>
    <h1>The Candy Store</h1>
    <nav>
      <a href="index.php">Home</a> |
      <a href="candy.php">Candy</a> |
      <a href="about.php">About</a> |
      <a href="contact.php">Contact</a>
    </nav>
```

section_a/c02/includes/footer.php — PHP

```
    <footer>&copy; <?php echo date('Y')?></footer>
  </body>
</html>
```

Последний закрывающий тег ?> в PHP-файле не ставится, так как пробелы после закрывающего тега могут привести к появлению нежелательных пробелов в браузере. Это также может привести к слишком ранней отправке заголовков HTTP в браузер.

В конце подключаемого файла вы также можете обнаружить пустую строку. Иногда она добавляется для использования инструментами, которые анализируют изменения в разных версиях файла (они часто используются группами разработчиков и в репозиториях кода, таких как GitHub).

ИСПОЛЬЗОВАНИЕ ПОДКЛЮЧАЕМЫХ ФАЙЛОВ

PHP section_a/c02/include-and-require-files.php

```php
<?php
$stock = 25;

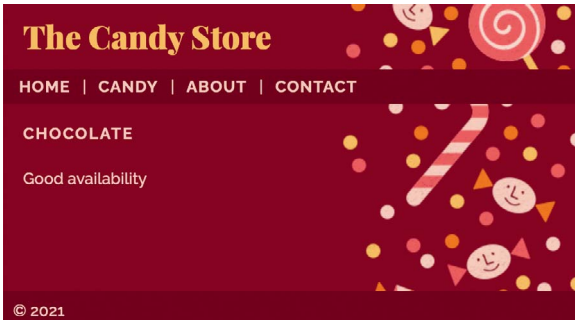
if ($stock >= 10) {
    $message = 'Good availability';
}
if ($stock > 0 && $stock < 10) {
    $message = 'Low stock';
}
if ($stock == 0) {
    $message = 'Out of stock';
}
?>

<?php require_once 'includes/header.php'; ?>

<h2>Chocolate</h2>
<p><?= $message ?></p>

<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



Упражнение. Добавьте новую ссылку в строке навигации файла `header.php`.

В данном случае мы будем использовать подключаемые файлы, приведенные в предыдущем примере.

1. Создадим сообщение о количестве товара на складе и сохраним его в переменной `$message`. Если количество:

- больше или равно 10, сообщение `Good availability` (Товар в наличии);
- находится в диапазоне от 1 до 9, отображается сообщение `Low stock` (Скоро закончится);
- равно 0, отобразится сообщение `Out of stock` (Нет в продаже).

2. Часть страницы с кодом HTML начинается с подключения файла заголовка (данный код используется в верхней части каждой страницы). Конструкция `require_once` указывает, что файл должен быть включен только один раз. Конструкция `require_once` размещается там, где должен отображаться заголовок, а интерпретатор PHP обрабатывает его так, как если бы содержимое этого файла было записано на месте оператора.

3. Далее следует содержание страницы, где отображается сообщение об уровне запасов.

4. Конструкция `include` указывает интерпретатору PHP добавить код из файла `footer.php`.


1:26 pm Tue 5 Jan
100%
AA
phpandmysql.com
The Candy Store
HOME | CANDY | ABOUT | CONTACT
Welcome back, Ivy
LOLLIPOP DISCOUNTS
PACKS PRICE
1 pack $1.92
2 packs $3.68
3 packs $5.28
4 packs $6.72
5 packs $8
6 packs $9.12
7 packs $10.08
8 packs $10.88
9 packs $11.52
10 packs $12
© 2021



ИТОГОВЫЙ ПРИМЕР

В следующем примере продемонстрировано отображение приветствия, за которым ниже перечислены скидки, действующие при покупке нескольких упаковок конфет. Используются различные техники, представленные в этой главе.

- В переменной содержится имя пользователя.
- С помощью условного оператора мы создаем приветствие для пользователя.
- Цикл `for` используется для создания индексированного массива, который содержит цены со скидкой, действующие при покупке нескольких упаковок конфет.
- Верхняя и нижняя часть страницы находятся в подключаемых файлах.
- Цикл `foreach` используется для отображения цен со скидкой (хранятся в массиве). При покупке 1 упаковки конфет действует скидка 4%, 2 упаковок — 8%, 3 упаковок — 12% и так далее.
- Использование тернарного оператора позволяет отображать слово `pack` (упаковка) при отображении цены одной упаковки конфет или слово `packs` (упаковки) при отображении цены нескольких упаковок.

КОД ПРИМЕРА

В самом начале программного кода создадим значения, которые будут отображаться на странице, и сохраним их в переменных.

1. Переменная `$name` — имя пользователя.
2. Объявим и проинициализируем переменную `$welcome`, в которой будет храниться сообщение `Hello`.
3. Условие в конструкции `if` проверяет, содержит ли переменная `$name` какое-либо значение.
4. Если это так, значение переменной `$greeting` обновится и будет содержать персональное сообщение с именем пользователя.
5. Переменная `$product` — название товара.
6. Переменная `$cost` — стоимость одной упаковки.
7. Цикл `for` создает массив с ценами за несколько упаковок конфет. Значение счетчика показывает количество упаковок конфет. Далее в скобках:

- начальное значение счетчика равно 1 (соответствует 1 упаковке конфет);
- условие проверяет, меньше ли счетчик значения 10 либо равен ему (соответствует 10 упаковкам конфет);
- при каждом проходе цикла счетчик увеличивается на 1.

Внутри цикла.

8. Переменная `$subtotal` — цена отдельной упаковки конфет, умноженная на значение счетчика (который представляет текущее количество упаковок).
9. Переменная `$discount` — скидка, действующая при покупке указанного количества упаковок. Скидка рассчитывается путем деления стоимости на 100, а затем умножения полученного значения на значение счетчика, умноженного на 4.
10. Переменная `$totals` представляет собой массив. В него добавляются элементы с индексом, равным текущему значению счетчика (которое представляет собой количество упаковок), и значением, равным цене этого количества упаковок конфет за вычетом скидки.
11. Закрываем цикл `for`.

Рассмотрим часть страницы, генерирующую HTML-код.

12. Используя оператор `require`, подключим файл с верхней частью оформления сайта страницы. Он необходим для правильного отображения остальной части страницы.
13. На странице отображается приветствие.
14. Название товара также отображается на странице.
15. Создадим HTML-таблицу, в первой строке которой выводятся заголовки столбцов.
16. Цикл `foreach` используется для отображения данных, которые были сохранены в массиве (шаги 7-8). В результате в таблицу будет добавлена новая строка данных для каждого элемента массива. Далее в скобках:

- в переменной `$totals` содержится массив;
- переменная `$amount` представляет текущий ключ;
- переменная `$price` представляет текущее значение.

17. Выводится ключ массива (это количество упаковок конфет).
18. Выводится слово `pack` (упаковка). Затем условие в тернарном операторе проверяет, равно ли значение переменной `$amount` значению 1. Если условие выполняется, к слову `pack` ничего не добавляется. В случае любого другого значения к слову `pack` добавляется окончание `s` (поэтому будет выводиться `packs` (упаковки), а не `pack` (упаковка).
19. Выводится цена на определенное количество конфет (со скидкой).
20. Закрываем цикл `foreach`.
21. Нижняя часть страницы подключается с помощью оператора `include`.

Упражнение. В шаге 6 измените значение переменной `$cost` на значение 10. В шаге 7 обновите цикл, чтобы он запускался 20 раз, отображая цены до 20 упаковок конфет.

PHP section_a/c02/example.php

```php
<?php
$name = 'Ivy';                                  // Имя пользователя

$greeting = 'Hello';                            // Начальное значение для приветствия
if ($name) {                                    // Если переменная $name имеет значение
    $greeting = 'Welcome back, ' . $name;       // Создаем персональное приветствие
}

$product = 'Lollipop';                          // Название товара
$cost    = 2;                                   // Цена за одну упаковку

for ($i = 1; $i <= 10; $i++) {
    $subtotal   = $cost * $i;                   // Общая стоимость
    $discount   = ($subtotal / 100) * ($i * 4); // Скидка
    $totals[$i] = $subtotal - $discount;        // Добавление цены со скидкой
                                                // в индексированный массив
}
?>

<?php require 'includes/header.php'; ?>

  <p><?= $greeting ?></p>
  <h2><?= $product ?> Discounts</h2>
  <table>
    <tr>
      <th>Packs</th>
      <th>Price</th>
    </tr>
    <?php foreach ($totals as $quantity => $price) { ?>
    <tr>
      <td>
        <?= $quantity ?>
        pack<?= ($quantity === 1) ? '' : 's'; ?>
      </td>
      <td>
        $<?= $price ?>
      </td>
    </tr>
    <?php } ?>
  </table>

<?php include 'includes/footer.php' ?>
```

ЗАКЛЮЧЕНИЕ
УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ

- Фигурные скобки используются для обозначения начала и конца группы инструкций для формирования блока кода.
- Условные конструкции используют условие, чтобы определить, выполнять или не выполнять блок кода.
- Выражения, проверяющие условие, возвращают значения `true` или `false`.
- Существуют четыре типа условных управляющих конструкций: `if`, `else`, `elseif`, `switch`, а также выражение `match`.
- Циклы позволяют повторять один и тот же блок кода несколько раз, пока условие возвращает значение `true` или пока не кончатся элементы в массиве.
- Существует четыре типа циклов: `while`, `do…while`, `for` и `foreach`.
- Подключение файлов позволяет избежать необходимости повторять один и тот же код в нескольких файлах.

3

ФУНКЦИИ

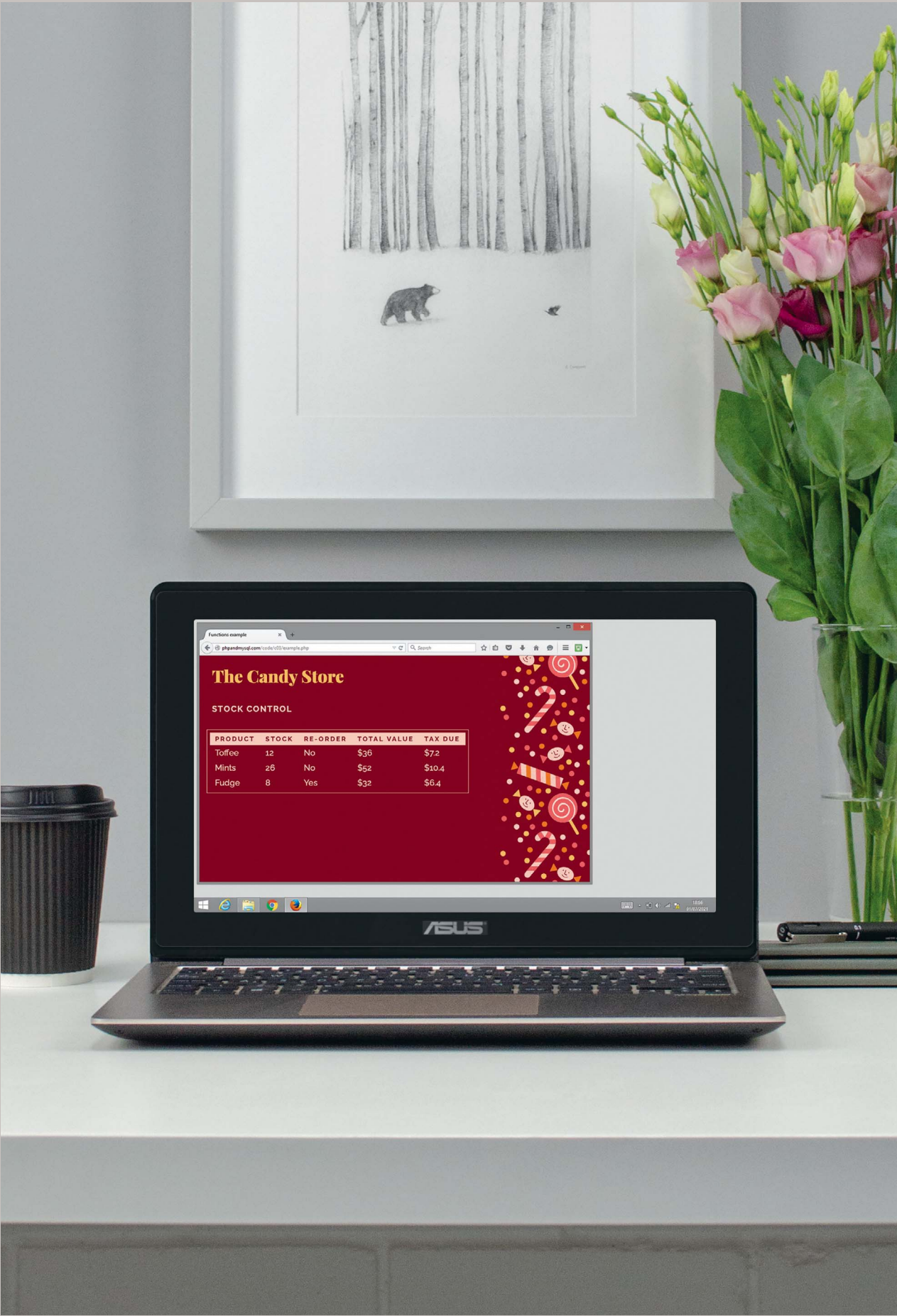
Часто одна веб-страница выполняет множество задач. С помощью функций можно объединять команды, необходимые для выполнения отдельной задачи.

Каждая страница на PHP может содержать сотни строк кода и выполнять несколько разных задач. Поэтому очень важно уделять внимание структурированию кода, разбиению его на отдельные блоки, чтобы вы (или другой программист) могли легко в нем разобраться.

В предыдущей главе вы увидели, как несколько связанных между собой инструкций можно сгруппировать, используя фигурные скобки, которые указывают начало и конец блока, который может быть пропущен (используя условные конструкции) или повторно выполнен (при использовании цикла).

Используя функции, можно сгруппировать в блок операции, необходимые для выполнения определенной задачи. **Имя функции** описывает задачу, выполняемую этим блоком кода (чтобы легко находить нужную функцию под определенную задачу). Открывающая фигурная скобка указывает интерпретатору PHP на начало блока кода, закрывающая фигурная скобка — на конец блока кода.

Когда интерпретатор PHP обнаруживает функцию, он не запускает код сразу, а ждет, пока другая инструкция не **вызовет** эту функцию. Только тогда он будет выполнять инструкции, заключенные в блоке кода функции. Также вы можете вызывать функцию несколько раз, чтобы избежать повторения одних и тех же строк кода.


Functions example
phpandmysql.com/code/c03/example.php
Search
The Candy Store
STOCK CONTROL
PRODUCT STOCK RE-ORDER TOTAL VALUE TAX DUE
Toffee 12 No $36 $7.2
Mints 26 No $52 $10.4
Fudge 8 Yes $32 $6.4
ASUS

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

Выполнение некоторых задач может потребовать написания множества команд PHP. Инструкции, необходимые для выполнения отдельной задачи, могут быть сохранены внутри функции, а затем при необходимости вызваны.

ОБЪЯВЛЕНИЕ И ВЫЗОВ ФУНКЦИИ

Функции присваивается имя, которое описывает выполняемую задачу. Затем следуют инструкции, необходимые для выполнения этой задачи. Программисты называют это **объявлением функции**.

Для вызова функции используется имя функции, за которым следуют круглые скобки. При вызове функции исполняются инструкции, расположенные в теле функции. Программисты называют это **вызовом** функции.

ПОЛУЧЕНИЕ РЕЗУЛЬТАТА РАБОТЫ ФУНКЦИИ

При выполнении своей задачи функция, как правило, возвращает значение, содержащее результат выполнения этой задачи. Например.

- Если функция используется для входа пользователя на сайт, то при успешном входе она может возвращать значение `true`, а при некорректном входе на сайт — значение `false`.
- Если функция используется для расчета общей стоимости заказа, то она вернет эту сумму.

ПЕРЕДАЧА ДАННЫХ В ФУНКЦИЮ

Часто возникает необходимость передать внутрь функции информацию извне.

Например, при регистрации пользователя на сайте потребуются следующие данные: адрес электронной почты пользователя и его пароль.

Параметры (подобно переменным) представляют каждый фрагмент данных, необходимый функции для выполнения своей задачи. **Аргументы** — фактические значения, используемые при вызове функции.

Функции, описанные в этой главе, выполняют очень простые задачи, просто чтобы показать, как создаются и используются функции. В следующих главах они будут использоваться для решения более сложных задач.

КАК ФУНКЦИИ ВЗАИМОДЕЙСТВУЮТ С ПЕРЕМЕННЫМИ

Код внутри функции не может получить доступ к переменным, объявленным вне функции. Поэтому любые данные должны быть переданы в функцию с помощью параметра.

Точно так же объявленные внутри функции переменные не могут быть доступны извне. Поэтому код в функции обычно пишется так, чтобы после вызова вернуть результат своей работы тому коду, который ее вызвал.

УКАЗАНИЕ ТИПОВ ДАННЫХ

Объявления типов данных указывают интерпретатору PHP ожидаемый тип для данных, которые:

- передаются в функцию (как аргумент);
- возвращают значение из функции.

Использование объявления типов гарантирует, что функция получит данные, которые она может применять для выполнения своей задачи. Это также помогает отслеживать в коде проблемные места.

НЕОБЯЗАТЕЛЬНЫЕ ПАРАМЕТРЫ И ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

При создании функции вы можете определить один или несколько необязательных параметров (фрагментов информации, необходимых для выполнения задачи).

Для того чтобы сделать параметр необязательным, вам необходимо указать для него значение по умолчанию — PHP использует его, когда функция будет вызвана без этого параметра.

ОБЪЯВЛЕНИЕ И ВЫЗОВ ФУНКЦИИ

Объявление функции состоит из имени, которое описывает выполняемую функцией задачу, и блока кода в фигурных скобках, который эту задачу выполняет. Затем функция **вызывается** при необходимости выполнить описанную в функции задачу.

Чтобы объявить (или создать) функцию, необходимо написать следующее.

- Ключевое слово `function` (это означает, что вы определяете новую функцию).
- Имя функции, описывающее суть выполняемой задачи, за которым следует пара круглых скобок.
- Фигурные скобки для указания начала и конца блока кода.

После закрывающей фигурной скобки точка с запятой не ставится.

В примере ниже функция содержит две инструкции.

- Первая присваивает переменной `$year` значение текущего года (как это работает, мы рассмотрим в главе 8).
- Вторая отображает на странице уведомление об авторских правах, используя значение, хранящееся в переменной `$year`.

При объявлении функции код не выполняется, а просто сохраняется в определении функции для дальнейшего использования.




```
function write_copyright_notice()
{
    $year = date('Y');
    echo '&copy; ' . $year;
}
```

Если необходимо выполнить задачу, для которой функция была создана, напишите имя функции, за которым идут круглые скобки. Так PHP-интерпретатор узнает, что здесь вы хотите выполнить код, написанный внутри функции.

После того как функция определена, вы можете вызывать ее сколько угодно раз. Как только функция выполнила свою задачу, интерпретатор PHP переходит к строке кода, следующей после вызова функции.




```
write_copyright_notice();
```

ДВЕ ПРОСТЫЕ ФУНКЦИИ

PHP section_a/c03/basic-functions.php

```
<?php
function write_logo()
{
    echo '<img src="img/logo.png" alt="Logo">';
}

function write_copyright_notice()
{
    $year = date('Y');
    echo '&copy; ' . $year;
}
?> ...
  <header>
    <h1><?php write_logo(); ?> The Candy Store</h1>
  </header>
  <article>
    <h2>Welcome to the Candy Store</h2>
  </article>
  <footer>
    <?php write_logo(); ?>
    <?php write_copyright_notice(); ?>
  </footer>
```

РЕЗУЛЬТАТ



Рассмотрим пример использования следующих двух функций. Первая функция отображает логотип, вторая выводит уведомление об авторских правах.

1. Определяем функцию `write_logo()`.
2. Заключенный в фигурные скобки блок кода содержит одну строку, которая выводит логотип.
3. Определяем еще одну функцию с именем `write_copyright_notice()`. Заключенный в фигурные скобки блок кода содержит две инструкции.
4. Переменной `$year` присвоим значение текущего года (более подробно, как это работает, вы узнаете в главе 8).
5. На страницу выводится знак авторского права ©, за которым следует год.
6. Первая функция вызывается, чтобы добавить логотип вверху страницы.
7. Она же вызывается, чтобы добавить логотип в нижнюю часть страницы.
8. Вторая функция выводит сведения об авторских правах в нижней части страницы.

Упражнение. После выполнения шага 5 в теле функции после сведений об авторских правах напишите название компании, The Candy Store (Кондитерская).

КОД НЕ ВСЕГДА ВЫПОЛНЯЕТСЯ ПОСЛЕДОВАТЕЛЬНО

Определение функции содержит код, который необходим для выполнения задачи. Этот код выполняется только при вызове функции. Это означает, что код не всегда работает в порядке его написания.

Впервые глядя на PHP-код, можно подумать, что операторы должны выполняться так, как они записаны, один за другим. Однако на практике интерпретатор PHP может выполнять операторы в совершенно другом порядке.

Как правило, вы будете видеть функции, объявленные в верхней части страницы. Если в верхней части страницы также объявляются переменные, то эти переменные обычно объявляются перед определениями функций.

Позже функции вызываются в коде, в котором необходимо выполнить задачу.

Даже если функции написаны в верхней части страницы, определение функции только сохраняет код, выполняющий задачу. Интерпретатор PHP не выполняет этот код до тех пор, пока функция не будет вызвана. Это означает, что инструкции могут выполняться в совершенно ином порядке, чем они написаны в коде.

```
<?php
(1) function write_logo()
{
(2)     echo '<img src="img/logo.png" alt="Logo" />';
}

(3) function write_copyright_notice()
{
(4)     $year = date('Y');          //Получение значения
                                    //текущего года
(5)     echo '&copy; ' . $year;     //Отображение сведений
                                    //об авторских правах
}
?>
<!DOCTYPE html>
<html>
  <header>
(6)     <h1><?php write_logo(); ?> The Candy Store</h1>
  </header>
  <article>
    <p>Welcome to The Candy Store</p>
  </article>
  <footer>
(7)     <?php write_logo(); ?>
(8)     <?php write_copyright_notice(); ?>
  <footer>
</html>
```

На странице слева находится точно такой же пример кода, что и на предыдущей. Но здесь он показывает порядок, в котором выполняются инструкции. Первая строка, которую фактически выполняет интерпретатор PHP, находится на шаге 6.

ШАГ	ДЕЙСТВИЯ ИНТЕРПРЕТАТОРА
6	Первая строка, которая будет исполнена, находится на шаге 6. На нем вызывается функция `write_logo()`
1, 2	Интерпретатор PHP переходит к шагу 1, на котором была определена функция. Затем выполняет шаг 2
6	После выполнения функции интерпретатор возвращается к строке кода, где была вызвана функция
7	Теперь интерпретатор переходит к следующей строке PHP-кода. В шаге 7 снова будет вызвана функция `write_logo()`
1, 2	Интерпретатор PHP переходит к шагу 1, на котором была определена функция. Затем выполняет шаг 2
7	После выполнения функции интерпретатор возвращается к строке кода, где была вызвана функция
8	Теперь интерпретатор переходит к следующей строке PHP-кода. В шаге 8 будет вызвана функция `write_copyright_notice()`
3, 4, 5	Интерпретатор переходит к шагу 3, где была объявлена эта функция. Затем выполняются шаги 4–5
8	По завершении функции интерпретатор возвращается к строке, в которой она была вызвана

Обратите внимание, что функция может быть вызвана до своего определения. Однако предпочтительнее сначала определить функции, а затем вызывать их.

Если возникает необходимость использовать одни и те же функции на нескольких страницах, тогда определения функций можно сохранить в подключаемом файле.

ПОЛУЧЕНИЕ РЕЗУЛЬТАТА РАБОТЫ ФУНКЦИИ

Функции обычно используются для получения новых значений. При использовании ключевого слова return такое значение может быть отправлено обратно в код, вызвавший функцию.

Функция, приведенная в примере ниже, аналогична функциям, приведенным на предыдущих страницах. Однако в отличие от предыдущего примера эта функция не отображает сведения об авторских правах, а сохраняет их в переменной $message.

Затем функция возвращает значение переменной $message (вместо того чтобы сразу выводить HTML прямо на странице).

```
function create_copyright_notice()
{
    $year    = date('Y');
    $message = '&copy; ' . $year;
    return $message;
}
```

КЛЮЧЕВОЕ СЛОВО — ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

Когда функция возвращает значение и вам необходимо отобразить эти данные на странице, напишите команду echo (или сокращенную форму записи), а затем вызовите функцию.

Хорошей практикой считается возвращать значение из функции, а затем отображать его на странице, а не использовать команду echo внутри функции.

```
<?= create_copyright_notice() ?>
```

Вы также можете сохранить возвращенное функцией значение в переменной.

Для этого напишите имя переменной, затем оператор присваивания и затем вызовите функцию.

```
$copyright_notice = create_copyright_notice();
```

ВОЗВРАТ ЗНАЧЕНИЙ: ПРИМЕР

PHP section_a/c03/functions-with-return-values.php

```
<?php
1 function create_logo()
  {
2     return '<img src="img/logo.png" alt="Logo" />';
  }

3 function create_copyright_notice()
  {
4     $year    = date('Y');
5     $message = '&copy; ' . $year;
6     return $message;
  }
?> ...
    <header>
7     <h1><?= create_logo() ?>The Candy Store</h1>
    </header>
    <article>
      <h2>Welcome to The Candy Store</h2>
    </article>
    <footer>
8     <?= create_logo() ?>
9     <?= create_copyright_notice() ?>
    </footer>
```

РЕЗУЛЬТАТ



В следующем примере рассмотрим функции PHP, возвращающие значения.

1. Объявим функцию `create_logo()`.
2. Далее используем оператор `return`, за которым следует HTML-код, необходимый для создания изображения.
3. Объявим функцию `create_copyright_notice()`. В фигурных скобках заключены три оператора.
4. В переменную `$year` запишем значение текущего года.
5. В переменную `$message` запишем знак авторского права ©, за которым будет следовать значение текущего года.
6. Затем вернем значение переменной `$message`.
7. Вызовем первую функцию. При использовании сокращенной формы записи команды `echo` на странице будет отображаться возвращаемое значение.
8. Чтобы повторно отобразить логотип, снова вызовем первую функцию.
9. Затем вызовем вторую функцию. При использовании сокращенной формы записи команды `echo` на странице будет отображаться возвращаемое значение.

Упражнение. В шаге 5 в переменную `$message` добавьте название компании.

ФУНКЦИИ, КОТОРЫМ ТРЕБУЮТСЯ ДАННЫЕ

Параметры (подобно переменным) представляют элементы данных, необходимых функции для выполнения своей задачи. Значения, определяемые параметрами, могут изменяться при каждом вызове функции.

При создании функции, которой для работы требуются данные:

- составьте список необходимых элементов данных;
- придумайте для каждого параметра имя. Оно должно начинаться с символа $ и описывать суть содержимого;
- заключите эти имена в круглые скобки;
- разделите каждое имя запятой.

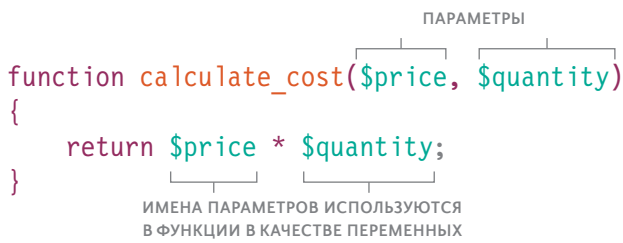
Это и есть **параметры** функции.
Параметры работают подобно переменным. Параметры используются только кодом, заключенным в фигурные скобки при определении функции. Код вне функции не может получить доступа к этим параметрам.

Приведенная ниже функция `calculate_cost()` вычисляет общую сумму при покупке одного или нескольких одинаковых товаров. Для выполнения такой задачи необходимы следующие два параметра:

- `$price` — цена одной единицы товара;
- `$amount` — количество товара.

Параметры `$price` и `$amount` внутри данной функции работают как переменные. Они представляют собой значения, которые передаются в функцию при ее вызове.

Код внутри определения функции вычисляет общую стоимость, умножая цену на количество. Затем это значение отправляется обратно в код, где была вызвана функция с помощью оператора `return`.



Когда выполнение функции завершается, интерпретатор PHP «забывает» все использовавшиеся в ней значения. Это важно, так как функция может быть вызвана на странице несколько раз, каждый раз используя новые данные.

Примечание. В PHP 8 в определении функции после имени последнего параметра (а не только между параметрами) можно добавлять запятую. В результате код будет выглядеть более однородным. Однако в более ранних версиях PHP это вызовет ошибку.

ВЫЗОВ ФУНКЦИИ С ПАРАМЕТРАМИ

При вызове функции с параметрами значение каждого параметра указывается в скобках после имени функции. Фактические значения, используемые при вызове функции, называются **аргументами**.

АРГУМЕНТЫ КАК ЯВНО ЗАДАННЫЕ ЗНАЧЕНИЯ

Ниже в коде при вызове функции `calculate_cost()` ей передаются значения, которые она должна использовать. Значения указываются в том же порядке, в котором параметры указаны в определении функции.

Число 3 используется для указания цены товара, а число 5 — для количества приобретенных товаров, поэтому функция `calculate_cost()` вернет число 15. Затем это значение будет присвоено переменной `$total`.

```
$total = calculate_cost(3, 5);
```

АРГУМЕНТЫ КАК ПЕРЕМЕННЫЕ

В данном случае в вызове функции `calculate_cost()` используются переменные, а не значения:

- `$cost` — цена товара;
- `$units` — количество приобретенных товаров.

Если в качестве аргументов используются переменные, их имена необязательно должны совпадать с именами параметров.

При вызове указанной ниже функции интерпретатор PHP отправляет в функцию значения переменных `$cost` и `$units`.

Внутри функции эти значения представлены как имена параметров `$price` и `$quantity` (как они указаны в скобках в первой строке определения функции).

```
$cost  = 4;
$units = 6;
$total = calculate_cost($cost, $units);
```

ПАРАМЕТРЫ И АРГУМЕНТЫ

Обычно пользователи понимают термины «параметр» и «аргумент» как синонимы, но между ними есть небольшая разница. В примере слева в определении функции вы можете увидеть следующие имена: `$price` и `$amount`. Заключенные в фигурные скобки функции, эти слова действуют как переменные. Эти имена — **параметры**.

В примере на этой странице код при вызове функции указывает либо числа, которые будут использоваться для выполнения вычислений, либо переменные, содержащие числа. Эти значения, которые передаются в функцию (информация, необходимая для расчета стоимости этого конкретного типа сладостей), называются **аргументами**.

ИСПОЛЬЗОВАНИЕ ФУНКЦИИ С ПАРАМЕТРАМИ

1. В верхней части кода определим функцию `calculate_total()`. Функция рассчитывает общую сумму при покупке одного или нескольких одинаковых товаров, а затем добавляется НДС 20%. Поскольку ей необходимы два элемента данных, в определении функции содержится два параметра:

- `$price` — цена одной единицы товара;
- `$amount` — количество приобретаемого товара.

2. В определении функции переменная `$cost` обозначает стоимость необходимого количества товара. Она вычисляется путем умножения значения переменной `$price` на значение переменной `$amount`.
3. Переменная `$tax` — НДС, подлежащий уплате за эти товары. НДС рассчитывается путем умножения значения переменной `$cost` (созданной в шаге 2) на значение `0,2`.
4. Для получения общей суммы необходимо сложить значения переменных `$cost` и `$tax`.
5. Значение общей суммы возвращается из функции в код, который ее вызвал.
6. Функция вызывается трижды. При каждом вызове в функции в качестве параметров содержатся разные значения цен и количества. Возвращаемые функцией значения будут отображаться на странице.

section_a/c03/function-with-parameters.php **PHP**

```
<?php
1 function calculate_total($price, $quantity)
{
2     $cost  = $price * $quantity;
3     $tax   = $cost * (20 / 100);
4     $total = $cost + $tax;
5     return $total;
}
?> ...
<h1>The Candy Store</h1>
<p>Mints:  $<?= calculate_total(2, 5) ?></p>
6 <p>Toffee: $<?= calculate_total(3, 5) ?></p>
<p>Fudge:  $<?= calculate_total(5, 4) ?></p>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 6 снова вызовите функцию, чтобы отобразить цену четырех упаковок жевательной резинки стоимостью $1,50 каждая.

КАК ПРАВИЛЬНО НАЗЫВАТЬ ФУНКЦИИ

Имя функции должно четко передавать ее назначение. Как правило, имя функции состоит из слова, описывающего то, что она делает, и вида информации, с которой она работает.

Рекомендации, как правильно называть функции, такие же, как и для переменных. С точки зрения синтаксиса, имя функции начинается с буквы или символа подчеркивания, за которым следует любая комбинация букв, цифр или знаков подчеркивания. На одной странице PHP не должно быть двух функций с одинаковыми именами.

В этой книге все имена функций написаны в нижнем регистре. Если имя функции состоит из двух и более слов, то для их разделения используется знак подчеркивания. Однако в дальнейшем вы можете встретить функции, следующие другим правилам именования. Единого стандарта на формат имен функций нет, но при этом в рамках одного проекта важно придерживаться одного и того же принципа именования.

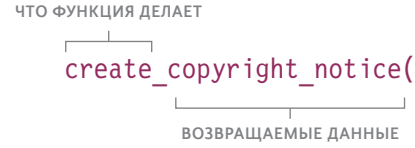
Чтобы составить имя функции, описывающее определенную задачу, необходимо учитывать следующие рекомендации:

- укажите, что именно делает функция (например, вычисляет, получает или обновляет);
- укажите вид информации, которую функция возвращает или обрабатывает (например, дата, итоговая сумма или сообщение).

Ниже приведены два примера описательных имен функций, которые уже встречались в этой главе:

- `calculate_total()` вычисляет общую стоимость товаров;
- `create_copyright_notice()` создает уведомление об авторских правах.





ОБЛАСТЬ ВИДИМОСТИ

При вызове функции ее код запускается в своей собственной **области видимости**. Он не может получить доступ к переменным вне функции.

Код функции выполняется независимо от остальной части страницы.

- Любая информация, необходимая функции для выполнения задачи, должна быть передана ей с использованием параметров. Внутри функции параметры работают как переменные.
- При вызове функции код внутри нее может создавать переменные и присваивать им значения.
- Затем функция возвращает значение коду, вызвавшему ее.
- По завершении работы функции все ее параметры и созданные в ней переменные удаляются.

Поскольку код в функции выполняется отдельно от остальной части страницы, это значит:

- функция не может обращаться к переменным или обновлять их вне функции, поэтому информация передается как параметры;
- последующий код не может получить доступ к переменным, созданным внутри функции, так как они удаляются, как только функция выполнит свою работу.

Программисты говорят, что каждый раз при вызове функции интерпретатор PHP запускает код этой функции в **локальной области видимости**. Код вне функции в основной части страницы находится в **глобальной области видимости**.

Объявление переменной в локальной или глобальной области указывает, может ли другой код получить к ней доступ.

На иллюстрации ниже рассматриваются две переменные `$tax`. Каждая переменная работает в своей области видимости.

А: Первая переменная `$tax` создана в глобальной области видимости, т. е. вне функции.

Б: Вторая переменная `$tax` создана внутри определения функции, т. е. в локальной области видимости.

```
<?php
Ⓐ$tax = 20;
 function calculate_total($price, $quantity)
 {
     $cost  = $price * $quantity;
Ⓑ    $tax   = $cost  * (20 / 100);
     $total = $cost  + $tax;
     return $total;
 }
 ?>
```

ОБЛАСТЬ ВИДИМОСТИ: ГЛОБАЛЬНАЯ ЛОКАЛЬНАЯ

В идеале две переменные не должны иметь в одном коде одинаковые имена. Однако этот пример демонстрирует, как переменные обрабатываются совершенно независимо друг от друга.

ОБЛАСТЬ ВИДИМОСТИ: ПРИМЕР

PHP section_a/c03/global-and-local-scope.php

```
<?php
① $tax = '20';

② function calculate_total($price, $quantity)
{
③     $cost  = $price * $quantity;
④     $tax   = $cost  * (20 / 100);
⑤     $total = $cost  + $tax;
⑥     return $total;
}
?> ...
<h1>The Candy Store</h1>
<p>Mints:  $<?= calculate_total(2, 5) ?></p>
⑦ <p>Toffee: $<?= calculate_total(3, 5) ?></p>
<p>Fudge:  $<?= calculate_total(5, 4) ?></p>
⑧ <p>Prices include tax at: <?= $tax ?>%</p>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 измените ставку налога на значение 25. В результате изменится ставка налога, отображаемая в нижней части страницы, но не итоговые суммы, указанные в шаге 7.

Значение переменной `$total` в шаге 7 остается прежним, поскольку в шаге 4 в функции используется ставка налога 20%. Это демонстрирует, как две переменные `$tax` работают независимо друг от друга.

1. Переменная `$tax` объявлена в глобальной области видимости, поэтому она доступна для кода вне функции.

2. Определим функцию `calculate_total()`. В функцию также необходимо передать цену и количество товара. Переменные, созданные внутри этой функции, находятся в локальной области видимости.

3. Стоимость рассчитывается путем умножения цены одной единицы товара на его количество. Результат сохраняется в переменной `$cost`.

4. Налог рассчитывается путем умножения стоимости товара в долларах США на ставку налога (20 делится на 100). Результат сохраняется в переменной `$tax`.

Примечание. В данном случае значение переменной `$tax`, созданной в шаге 1, не перезаписывается.

5. Чтобы получить итоговую сумму, значение переменной `$cost` добавляется к значению переменной `$tax`.

6. В результате возвращается значение переменной `$total`. При вызове функции интерпретатор PHP удалит все созданные в функции параметры и переменные.

7. Функция вызывается трижды, каждый раз с новыми значениями.

8. Отображается значение, присвоенное переменной `$tax` (в глобальной области видимости в шаге 1).

ГЛОБАЛЬНЫЕ И СТАТИЧЕСКИЕ ПЕРЕМЕННЫЕ

Только в редких случаях коду в функции можно дать доступ к глобальной переменной. Также можно указать, чтобы значение, которое было присвоено переменной внутри функции, не уничтожалось после завершения работы функции.

ДОСТУП К ГЛОБАЛЬНЫМ ПЕРЕМЕННЫМ ВНУТРИ ФУНКЦИИ

Код внутри функции может получить доступ к переменной, объявленной в глобальной области видимости. Но для этого интерпретатору PHP нужно сообщить, что функция может получить к ней доступ.

В начале блока кода в функции (перед использованием переменной) добавьте ключевое слово `global`, за которым следует имя переменной. Это позволит коду функции получить доступ к глобальной переменной.

Следует помнить, что использование параметров для передачи переменных в функцию является предпочтительным, а данный пример приводится в качестве демонстрации этой возможности языка на случай, если вы встретите его в существующем коде.

```
global $cost;
```

КЛЮЧЕВОЕ СЛОВО — ПЕРЕМЕННАЯ

СОХРАНЕНИЕ ЗНАЧЕНИЯ ПЕРЕМЕННОЙ В ФУНКЦИИ ПОСЛЕ ЕЕ ВЫЗОВА

После завершения работы функция, как правило, удаляет все локальные переменные, созданные внутри функции.

При необходимости интерпретатор PHP может запомнить значение, хранящееся в созданной в функции переменной, если эта переменная объявлена как **статическая**.

Для объявления статической переменной необходимо следующее:

- ключевое слово `static`;
- за ним следует имя переменной;
- затем переменной присваивается начальное значение, которое переменная будет иметь при первом вызове функции.

После завершения работы функции эта переменная и сохраненное в ней значение не будут удалены. Они будут доступны только для кода внутри функции.

```
static $quantity = 10;
```

КЛЮЧЕВОЕ СЛОВО — ПЕРЕМЕННАЯ — НАЧАЛЬНОЕ ЗНАЧЕНИЕ

ДОСТУП К ПЕРЕМЕННЫМ ВНЕ ФУНКЦИИ

PHP section_a/c03/global-and-static-variables.php

```php
<?php
$tax_rate = 0.2;

function calculate_running_total($price, $quantity)
{
    global $tax_rate;
    static $running_total = 0;
    $total = $price * $quantity;
    $tax   = $total * $tax_rate;
    $running_total = $running_total + $total + $tax;
    return $running_total;
}
?> ...
<h1>The Candy Store</h1>
<table>
  <tr><th>Item</th><th>Price</th><th>Qty</th>
    <th>Running total</th></tr>
  <tr><td>Mints:</td><td>$2</td><td>5</td>
    <td>$<?= calculate_running_total(2, 5); ?></td></tr>
  <tr><td>Toffee:</td><td>$3</td><td>5</td>
    <td>$<?= calculate_running_total(3, 5); ?></td></tr>
  <tr><td>Fudge:</td><td>$5</td><td>4</td>
    <td>$<?= calculate_running_total(5, 4); ?></td></tr>
</table>
```

РЕЗУЛЬТАТ



1. В глобальной области видимости создадим переменную `$tax_rate`.

2. Функция `calculate_running_total()` вычисляет итоговую стоимость.

3. Ключевое слово `global` позволяет функции получить доступ к глобальной переменной `$tax_rate`.

4. Ключевое слово `static` означает, что переменная `$running_total` (накопительный результат) сохранит свое значение после завершения работы функции. При объявлении переменной ей присваивается начальное значение 0.

5. Переменная `$total` — цена одной единицы товара, умноженная на необходимое количество.

6. Переменная `$tax` — сумма налога, при вычислении которого используется значение глобальной переменной `$tax_rate`, созданной в шаге 1.

7. Переменная `$running_total` получает свое значение из суммы трех переменных:

- `$running_total`;
- `$total`;
- `$tax`.

8. Значение переменной `$running_total` возвращается, но не удаляется, так как `$running_total` — статическая переменная.

9. Функция вызывается трижды. Каждый раз сумма для следующего элемента добавляется к предыдущему результату.

ФУНКЦИИ И СОСТАВНЫЕ ТИПЫ ДАННЫХ

Составные типы данных (например, массивы) могут хранить несколько значений. Функции могут как принимать составные типы данных в качестве аргументов, так и возвращать их как результат своей работы.

ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ ТИПОВ ДАННЫХ В КАЧЕСТВЕ АРГУМЕНТА

При определении функции параметры могут принимать как скалярные, так и составные типы данных.

- Скалярные типы содержат один элемент данных: строку, число или логическое значение.
- Составные типы содержат несколько элементов. В главе 1 вы изучили массивы, а в главе 4 познакомитесь еще с одним составным типом данных — объектами.

На странице справа приведен пример, в котором массив, содержащий значения трех разных обменных курсов, передается в функцию как один параметр.

ИСПОЛЬЗОВАНИЕ СОСТАВНОГО ТИПА ДАННЫХ В КАЧЕСТВЕ ВОЗВРАЩАЕМОГО ЗНАЧЕНИЯ

Хотя каждая функция должна выполнять только одну задачу, а не несколько, в результате решения даже одной задачи может потребоваться вернуть больше одного значения.

При необходимости вернуть из функции более одного значения вам нужно создать в функции массив или объект, а затем вернуть его. Это объясняется тем, что функция может возвращать только одно значение, скалярное или составное.

Как только интерпретатор PHP выполняет оператор `return`, он прекращает выполнение кода в функции и возвращается к строке кода, вызвавшей функцию (даже если в определении функции имеются другие инструкции, которые в итоге не будут выполнены).

В примере на странице справа функция `calculate_prices()` вычисляет для переданной суммы ее цену в трех валютах и возвращает эти значения в виде массива.

ПОЛУЧЕНИЕ И ВОЗВРАТ НЕСКОЛЬКИХ ЗНАЧЕНИЙ

PHP section_a/c03/functions-with-multiple-values.php

```php
<?php
$us_price = 4;
$rates = [
    'uk' => 0.81,
    'eu' => 0.93,
    'jp' => 113.21,
];

function calculate_prices($usd, $exchange_rates)
{
    $prices =  [
        'pound' => $usd * $exchange_rates['uk'],
        'euro'  => $usd * $exchange_rates['eu'],
        'yen'   => $usd * $exchange_rates['jp'],
    ];
    return $prices;
}
$global_prices = calculate_prices($us_price, $rates);
?> ...
<h2>Chocolates</h2>
<p>US $<?= $us_price ?></p>
<p>(UK &pound; <?= $global_prices['pound'] ?> |
    EU &euro;  <?= $global_prices['euro'] ?> |
    JP &yen;   <?= $global_prices['yen'] ?>)</p>
```

РЕЗУЛЬТАТ



1. Переменная `$us_price` — цена товара в долларах США.
2. Переменная `$rates` хранит ассоциативный массив из трех обменных курсов.
3. Функция `calculate_prices()` вычисляет цены на товар в трех валютах, а затем возвращает значения цен в виде массива. Функция использует два параметра: цена в долларах США и массив, содержащий курсы валют.
4. Создадим массив и сохраним его в переменной `$price`. Первый элемент — цена в Великобритании, которая рассчитывается путем умножения цены в США на обменный курс Великобритании. Затем в массив добавляются цены ЕС и Японии.
5. Функция возвращает массив, содержащий три новые цены.
6. Вызовем функцию. Возвращаемый массив сохраняется в переменной `$global_prices`.
7. Выводится цена в долларах США, используя переменную из шага 1.
8. Так же отобразим другие цены из массива, созданного на шаге 6.

Упражнение. Установите обменный курс для австралийских долларов 1,32.

УКАЗАНИЕ ТИПА ДЛЯ АРГУМЕНТОВ И ВОЗВРАЩАЕМЫХ ЗНАЧЕНИЙ

При определении функции вы можете указать тип данных для каждого аргумента, а также какого типа данные она возвращает.

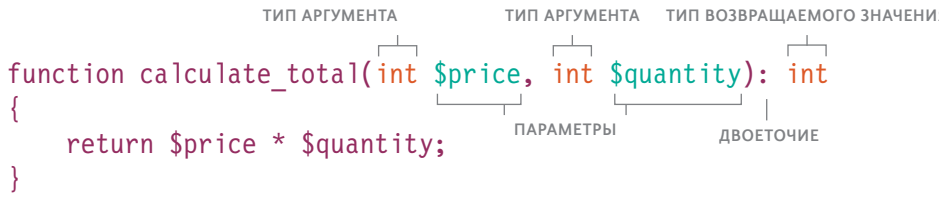
Большинство задач требуют данные строго определенного типа. Например, функциям, выполняющим арифметические действия, в качестве аргументов подходят только числа, а функциям, обрабатывающим текст, — строки.

При определении функции можно указать тип данных для каждого параметра и тип данных, которые она возвращает. Это помогает программистам, так как в первой же строке определения функции четко видно, какого типа должен быть каждый аргумент[11] и какого типа данные функция возвращает.

В примере ниже в первой строке определения функции:

- внутри круглых скобок перед каждым именем параметра указывается **тип аргумента**;
- после круглых скобок стоит двоеточие, за которым пишется **тип возвращаемого значения**, чтобы указать тип данных, которые вернет функция.

В нашем случае оба аргумента, а также возвращаемое значение должны быть целыми числами.



В таблице справа приведены типы данных, используемые в объявлениях аргументов и возвращаемых типов. В PHP 8 добавлены следующие типы данных.

- Объединение типов (union types) означает, что аргумент или возвращаемое значение может быть одного из нескольких типов, разделенных символом `|`. Например, `int|float` обозначает целое число или число с плавающей запятой.
- Тип `mixed` говорит о том, что параметр или возвращаемое значение может быть любого типа (также такой тип называется псевдотипом, так как переменных такого типа не бывает).

ТИП ДАННЫХ	ОПИСАНИЕ
`string`	Строка
`int`	Целое число
`float`	Число с плавающей запятой
`bool`	Логический тип данных (`true` или `false` / 0 или 1)
`array`	Массив
`имяКласса`	Объект определенного класса (см. главу 4)
`mixed`	Любой тип данных (доступен в PHP 8 и выше)

ПРИМЕР ОБЪЯВЛЕНИЯ ТИПОВ ДАННЫХ

PHP section_a/c03/type-declarations.php

```php
<?php
$price    = 4;
$quantity = 3;

function calculate_total(int $price, int $quantity): int
{
    return $price * $quantity;
}

$total = calculate_total($price, $quantity);
?>
<h1>The Candy Store</h1>
<h2>Chocolates</h2>
<p>Total $<?= $total ?></p>
```

РЕЗУЛЬТАТ



1. В примере первая строка определения функции указывает следующее.

- Объявление типа для параметров `$price` и `$amount`, чтобы показать, что их значения должны быть целыми числами.
- Объявление типа возвращаемого значения, показывающее, что функция должна возвращать целое число.

В этом примере объявление типов не вызовет ошибку, если передать в качестве аргументов числа в переменных строкового типа. Оно только подсказывает, какого типа должны быть аргументы и возвращаемые значения. На следующей странице показано, как сделать эту проверку строгой.

Упражнение. Измените тип переменной `$price` на строку, например:

```
$price = '1';
```

При обновлении страницы вы должны увидеть тот же результат, поскольку для появления ошибки вам понадобится включить строгую проверку типов (см. следующую страницу).

Примечание. Если аргумент или возвращаемый тип может принимать значение `null`, то перед типом данных можно использовать вопросительный знак.

Например, `?int` указывает, что значение может принимать тип данных `integer` или `null`. В PHP 8 также можно использовать объединенный тип `int|null`.

Упражнение. Если вы запускаете код с помощью PHP 8, то используйте объединение типов, чтобы указать, что значения могут быть либо целыми, либо числами с плавающей запятой.

СТРОГАЯ ПРОВЕРКА ТИПОВ

Если в функции используется указание типов для аргументов и возвращаемых значений, то в случае совсем неверных данных, например если будет передана строка 'абв' вместо целого числа, PHP в любом случае будет выдавать ошибку. Однако интерпретатор может «смотреть сквозь пальцы», если передаются правильные данные, но неправильного типа, например строка '1' там, где ожидается число. Но вы можете указать интерпретатору генерировать ошибку и в таких случаях.

Дело в том, что для некоторых скалярных типов интерпретатор PHP может попытаться преобразовать значение автоматически. Например, при передаче аргумента в функцию или возврате значения из нее, в следующих случаях интерпретатор PHP будет преобразовывать:

- строку `'1'` в целое число `1`;
- логическое значение `true` — в целое число `1`;
- логическое значение `false` — в целое число `0`;
- целое число `1` — в логическое значение `true`;
- целое число `0` — в логическое значение `false`.

Но вы можете указать интерпретатору PHP включить **строгую проверку типов**, чтобы он не пытался их преобразовывать, а сразу вызывал ошибку в случаях, если функция:

- вызывается с аргументом неверного типа (если он был указан);
- возвращает значение неверного типа данных (если был указан тип возвращаемого значения).

Возникающие при этом ошибки могут помочь отследить источник проблем в PHP-коде. Такое поведение необходимо включать специально, с помощью команды, приведенной ниже. Причем делать это надо на той странице, где функция **вызывается**. И это **обязательно** должен быть первый оператор на странице.

СТРОГАЯ ПРОВЕРКА ТИПОВ ВКЛ.

```
declare(strict_types = 1);
```

Изучив главу 10, вы научитесь обрабатывать ошибки и устранять неполадки. Однако базовые настройки отображения ошибок уже включены в загружаемых файлах, поэтому вы должны видеть ошибки прямо на экране. Возможно, вы заметили, что в коде для каждой главы есть файл `.htaccess`. Этот файл используется для управления настройками веб-сервера, и, в частности, в нем можно указать, сообщать ли об ошибках прямо на HTML-странице, которую веб-сервер отправляет в браузер. Если вы не видите этот файл, то потому, что операционные системы определяют его как скрытый и не отображают.

Часто определения функций пишутся в отдельном подключаемом файле, так чтобы одни и те же функции можно было использовать на нескольких страницах сайта (см. главу 6). При этом включать строгую проверку типов в таких файлах не обязательно[12]. Чтобы интерпретатор PHP выдавал ошибку при использовании неверного типа данных, строгая проверка типов должна быть включена на тех страницах, где функции **вызываются**.

ИСПОЛЬЗОВАНИЕ СТРОГОЙ ТИПИЗАЦИИ

PHP section_a/c03/strict-types.php

```php
<?php
declare(strict_types = 1);

$price    = 4;
$quantity = 3;

function calculate_total(int $price, int $quantity): int
{
    return $price * $quantity;
}

$total = calculate_total($price, $quantity);
?>
<h1>The Candy Store</h1>
<h2>Chocolates</h2>
<p>Total $<?= $total ?></p>
```

РЕЗУЛЬТАТ



Этот пример подобен предыдущему. Отличие в следующем: чтобы показать ошибку в случае, если аргументы или возвращаемое значение имеют неправильный тип данных, в первой строке включается строгая проверка типов.

1. Конструкция `declare` включает строгую типизацию на данной странице.
2. Создаются две переменные для цены и количества.
3. Объявляется функция `calculate_total()`:

- объявление типа для аргументов указывает, что оба параметра — целые числа;
- объявление типа возвращаемого значения указывает, что функция возвращает целое число.

4. Код функции умножает цену на количество и возвращает полученное значение.
5. Функция вызывается с использованием переменных, созданных в шаге 2, а возвращаемое в результате значение сохраняется в переменной `$total`.
6. Выводится значение переменной `$total`.

Упражнение. В шаге 2 установите значение переменной `$price` строкой:

```
$price='4';
```

После обновления страницы вы увидите сообщение об ошибке.

Упражнение. При использовании PHP 8 в шаге 2 в качестве цены используйте дробное значение `4,5`, а в шаге 3 используйте объединение типов, чтобы указать, что аргументы и возвращаемые значения могут быть `int` или `float`.

НЕСКОЛЬКО КОНСТРУКЦИЙ RETURN В ОДНОЙ ФУНКЦИИ

Функции могут возвращать разные значения в зависимости от результата выполнения кода внутри функции.

Чтобы определить, какое значение должно быть возвращено, функция может использовать условные конструкции. В примере ниже функция возвращает разные сообщения в зависимости от значения, переданного в качестве аргумента.

Как только в коде функции выполняется конструкция return, интерпретатор PHP возвращается к строке кода, вызвавшей функцию. Ни один из последующих операторов в этой функции выполняться не будет.

В приведенной ниже функции конструкция `return` вызывается три раза.

1. В условии проверяется, равно ли или больше 10 значение параметра `$stock`. Если условие выполняется, то обрабатывается первая конструкция `return`, а последующие инструкции в функции выполняться не будут.

2. Если значение параметра `$stock` больше 0 и меньше 10, срабатывает вторая конструкция `return`, и последующий код функции выполняться не будет.

3. Если функция все еще выполняется, это значит, что параметр `$stock` меньше либо равен нулю. В этом случае будет выполнена последняя конструкция `return`.

```
function get_stock_message($stock)
{
    if ($stock >= 10) {
        return 'Good availability';
    }
    if ($stock > 0 && $stock < 10) {
        return 'Low stock';
    }
    return 'Out of stock';
}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ НЕСКОЛЬКИХ КОНСТРУКЦИЙ RETURN

PHP section_a/c03/multiple-return-statements.php

```php
<?php
$stock = 25;

function get_stock_message($stock)
{
    if ($stock >= 10) {
        return 'Good availability';
    }
    if ($stock > 0 && $stock < 10) {
        return 'Low stock';
    }
    return 'Out of stock';
}
?>
<h1>The Candy Store</h1>
<h2>Chocolates</h2>
<p><?= get_stock_message($stock) ?></p>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 измените количество товара на значение 8. Вы увидите сообщение `Low stock` (Скоро закончится).

1. Переменная `$stock` — количество товара на складе.
2. Функция `get_stock_message()` проверяет количество товара на складе и возвращает одно из трех сообщений.
3. В условной конструкции проверяется, является ли количество товара больше или равным 10. Если условие выполняется, то вызывается первая конструкция `return`. Она возвращает сообщение `Good availability` (Товар в наличии), и код функции дальше не выполняется.
4. Если запасы товара меньше, то функция продолжит выполнение и будет проверено следующее условие — находится ли количество товара в промежутке между 0 и 10. Если условие выполняется, то будет вызвана вторая конструкция `return`. Она возвращает сообщение `Low stock` (Скоро закончится), и код в функции прекратит выполнение.
5. Если код в функции все еще продолжает выполняться, это означает, что товар закончился, поэтому последняя конструкция `return` выдает сообщение `Out of stock` (Нет в наличии).
6. Функция вызывается, и возвращаемое значение отображается на странице.

НЕОБЯЗАТЕЛЬНЫЕ ПАРАМЕТРЫ И ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

Параметры функции могут быть необязательными. Для этого необходимо присвоить параметру значение по умолчанию, которое будет использоваться, когда значение не указано. Необязательные параметры обычно пишутся после обязательных.

При выполнении некоторых задач может использоваться дополнительная информация, которая не является обязательной, однако может потребоваться в отдельных случаях.

Чтобы сделать параметр необязательным, ему необходимо присвоить **значение по умолчанию**, которое будет использоваться при вызове функции, если аргумент для этого параметра не был передан.

Значение по умолчанию указывается в определении функции после имени параметра. Синтаксис такой же, как если бы вы присваивали значение переменной.

В примере ниже функция вызывается с использованием только двух аргументов, поэтому последнему параметру необходимо добавить значение по умолчанию, оно будет равно 0.

Необязательные параметры следуют после обязательных, так при стандартном вызове функции аргументы должны указываться ровно в том же порядке, в котором параметры перечислены в определении функции.

Далее в этой главе вы узнаете, как с помощью именованных параметров можно указывать аргументы в произвольном порядке, но все равно традиционно необязательные параметры пишутся после обязательных.

```
function calculate_cost($cost, $quantity, $discount = 0)
{
    $cost = $cost * $quantity;
    return $cost - $discount;
}

$cost = calculate_cost(5, 3);
```

НЕОБЯЗАТЕЛЬНЫЙ ПАРАМЕТР

При написании документации необязательные параметры обычно выделяются с помощью квадратных скобок. В примере ниже показан **не** пример кода, а пример описания функции в документации к коду.

Примечание. Запятая перед необязательными параметрами также находится в квадратных скобках, так как запятая после последнего аргумента при вызове функции вызывала ошибку вплоть до версии PHP 8.

```
calculate_cost($cost, $quantity[, $discount])
```

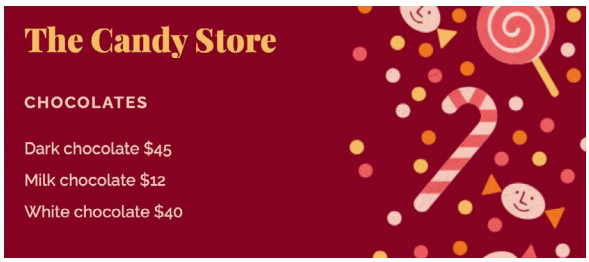
НЕОБЯЗАТЕЛЬНЫЙ ПАРАМЕТР ЗАКЛЮЧЕН В КВАДРАТНЫЕ СКОБКИ

ИСПОЛЬЗОВАНИЕ ЗНАЧЕНИЙ ПАРАМЕТРОВ ПО УМОЛЧАНИЮ

PHP section_a/c03/default-values-for-parameters.php

```php
<?php
function calculate_cost($cost, $quantity, $discount = 0)
{
    $cost = $cost * $quantity;
    return $cost - $discount;
}
?>
<h1>The Candy Store</h1>
<h2>Chocolates</h2>
<p>Dark chocolate $<?= calculate_cost(5, 10, 5) ?></p>
<p>Milk chocolate $<?= calculate_cost(3, 4) ?></p>
<p>White chocolate $<?= calculate_cost(4, 15, 20) ?></p>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 измените скидку по умолчанию на значение 2. В шаге 2 измените значение скидки на 7.

1. Функция `calculate_cost()` вычисляет стоимость одного или нескольких единиц товара на основе следующей информации:

- стоимость;
- количество;
- скидка.

При вызове функции последний аргумент будет необязательным, так как параметру по умолчанию присваивается значение 0. Затем функция вызывается трижды.

2. При первом вызове функции передаются следующие параметры: стоимость — 5, количество — 10, скидка — 5. Функция вычтет значение 5 из значения общей суммы 50 и в результате вернет значение 45.

3. При втором вызове функции передаются следующие параметры: стоимость — 3, количество — 4, скидка не предоставляется, поэтому ей присваивается значение по умолчанию 0. В результате функция вернет значение стоимости 12.

4. При третьем вызове функции передаются следующие параметры: стоимость — 4, количество — 15, скидка — 20. Аналогично шагу 2 значение скидки (в данном случае 20) вычитается из стоимости (которая равна 60), и в результате возвращается значение 40.

ИМЕНОВАННЫЕ АРГУМЕНТЫ

В PHP 8 при вызове функции вы можете указывать имя параметра перед аргументом. В этом случае не нужно указывать аргументы в том порядке, в котором они прописаны в определении функции.

Некоторые функции содержат много параметров, и при вызове бывает сложно понять, какой за что отвечает. В PHP 8 при вызове функции вы можете добавлять имена параметров перед аргументами. Эти параметры называются **именованными аргументами** (или именованными параметрами). Они позволяют:

- четко указывать, что делает каждый аргумент;
- пропускать необязательные аргументы без указания значения по умолчанию.

Определение функции не меняется, меняется только способ передачи аргументов при ее вызове. В примере на странице справа используются следующие четыре параметра:

- `$cost` (обязательный) — стоимость товара;
- `$quantity` (обязательный) — количество товара;
- `$discount` (необязательный) — величина скидки;
- `$tax` (необязательный) — налог.

При вызове функции без именованных аргументов аргументы должны писаться в том же порядке, что и параметры в определении функции.

Но если в функции используется два параметра по умолчанию, а нам нужно передать значение только для второго (например, если мы хотим оставить значение по умолчанию для параметра `$discount`, но при этом указать свое значение для параметра `$tax`), то для параметра `$discount` придется задать значение вручную, поскольку он стоит перед параметром `$tax`.

```
calculate_cost(5, 10, 0, 5);  or  calculate_cost(5, 10, '', 5);
```

При использовании именованных аргументов имя отделяют от аргумента двоеточием, и аргументы могут стоять в любом порядке.

Если аргумент использует значение по умолчанию (указанное в определении функции), значение для него указывать не нужно.

```
calculate_cost(quantity: 10, cost: 5, tax: 5);
```

Обычные (неименованные) аргументы можно использовать перед именованными, если они пишутся в том же порядке, что и при объявлении функции.

В примере ниже первые два значения будут использоваться для определения стоимости и количества. Затем следует налог. Обратите внимание, что размер скидки не указан.

```
calculate_cost(5, 10, tax: 5);
```

ИСПОЛЬЗОВАНИЕ ИМЕНОВАННЫХ АРГУМЕНТОВ

PHP section_a/c03/named-arguments-in-php-8.php

```php
<?php
function calculate_cost($cost, $quantity, $discount = 0, $tax = 20,)
{
    $cost = $cost * $quantity;
    $tax  = $cost * ($tax / 100);
    return ($cost + $tax) - $discount;
}
?>
<h1>The Candy Store</h1>
<h2>Chocolates</h2>
<p>Dark chocolate $<?= calculate_cost(quantity: 10, cost: 5, tax: 5, discount: 2); ?></p>
<p>Milk chocolate $<?= calculate_cost(quantity: 10, cost: 5, tax: 5); ?></p>
<p>White chocolate $<?= calculate_cost(5, 10, tax: 5); ?></p>
```

1. Функция `calculate_cost()` вычисляет стоимость одного или нескольких товаров на основе следующей информации:

- стоимость (обязательно);
- количество (обязательно);
- скидка (необязательно, по умолчанию равна 0);
- налог (необязательно, по умолчанию равен 20%).

Поскольку этот пример разработан для PHP 8, **конечную запятую** в определении функции можно добавить после последнего параметра, а не только между параметрами. Это улучшает единообразие и читабельность кода, поскольку запятая ставится после каждого параметра.

После определения функция вызывается трижды.

РЕЗУЛЬТАТ



2. Функция вызывается с четырьмя именованными аргументами. Поскольку аргументам присвоены имена, аргументы могут передаваться в любом порядке.

3. Именованные аргументы используются для передачи в функцию значений стоимости, количества и налога. Значение скидки берется по умолчанию.

4. Первые два значения передаются традиционным способом, а это значит, что они следуют порядку параметров в определении функции, то есть это значения для параметров `$cost` и `$amount`. Чтобы не указывать значение для параметра `$discount`, последний аргумент для параметра `$tax` сделан именованным.

РЕКОМЕНДАЦИИ ПО НАПИСАНИЮ ФУНКЦИЙ

Следующие рекомендации помогут вам в написании функций.

1. КРАТКО ОПИШИТЕ ЗАДАЧУ

Используйте описание выполняемой функцией задачи (например, получение, вычисление, обновление или сохранение), с указанием типа данных, с которыми функция работает. Это станет именем функции.
Имя функции никогда не меняется.

2. РЕШИТЕ, КАКИЕ ДАННЫЕ НЕОБХОДИМЫ ДЛЯ ВЫПОЛНЕНИЯ ЗАДАЧИ

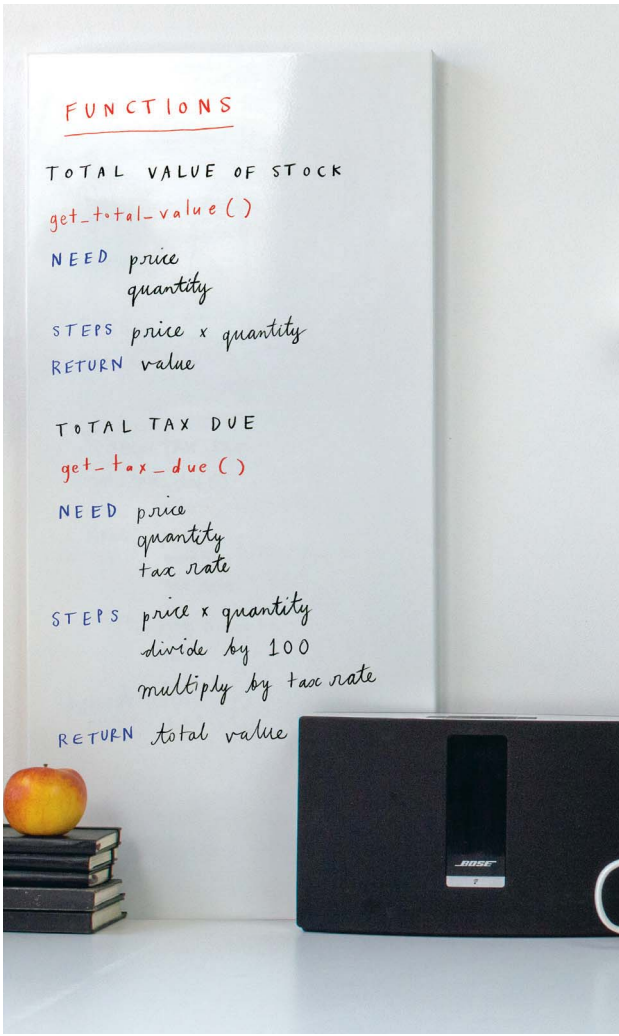
Каждый необходимый фрагмент данных становится параметром.
Значения, переданные в функцию (аргументы), могут изменяться каждый раз при вызове функции.

3. КАКОЙ КОД ПОНАДОБИТСЯ ДЛЯ ВЫПОЛНЕНИЯ ЗАДАЧИ?

Код, выполняющий задачу, пишется внутри фигурных скобок.
При каждом вызове функции выполняемые инструкции будут одинаковыми.

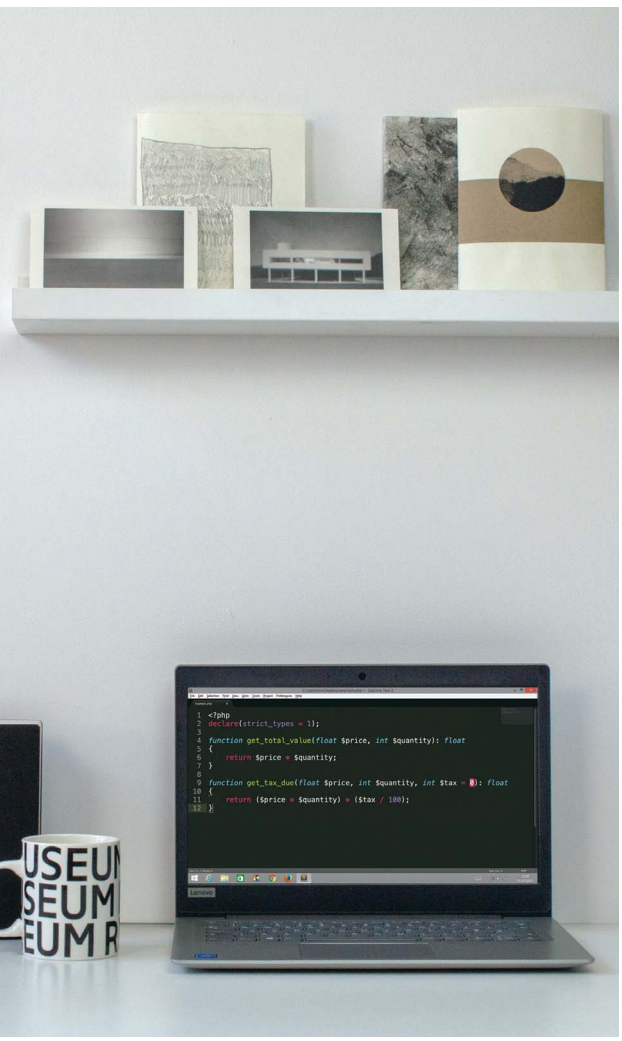
4. КАКОЙ РЕЗУЛЬТАТ ОЖИДАЕТСЯ?

Результатом будет возвращаемое функцией значение. Хорошей практикой считается, когда функция возвращает значение. Если вы выполняете задачу, которая не вычисляет новое значение или не извлекает информацию, то функция, как правило, возвращает значения `true` или `false`, чтобы определить, сработала она или нет.
Возвращаемое значение может меняться каждый раз при изменении передаваемых в функцию значений.



ЗАЧЕМ НУЖНЫ ФУНКЦИИ?

Помещая выполняющий определенную задачу код внутрь функции, вы получаете определенные преимущества.



ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ

Как рассматривалось ранее в этой главе, если на странице выполняется одна и та же задача несколько раз (например, вычисление стоимости товаров), код для выполнения этой задачи нужно написать только один раз. И далее всякий раз, когда требуется выполнить эту задачу, вам нужно всего лишь вызвать функцию и передать ей необходимые для выполнения работы значения.

МОДЕРНИЗАЦИЯ

Если не использовать функции и решать какую-либо задачу дублированием одного и того же кода в разных местах, то в случае, если логика выполнения задачи изменится, придется вносить изменения в каждом из них. В случае же использования функции вы можете изменить код только в одном месте — в определении функции. И после этого при каждом вызове функции сразу будет использоваться обновленный код.

СТРУКТУРА

Когда вы помещаете определенные инструкции в функцию, становится проще находить код, выполняющий ту или иную задачу, а сам код становится гораздо компактнее и читабельнее.

ТЕСТИРОВАНИЕ

Разбивая код на функции, каждая из которых выполняет определенную задачу, вы можете тестировать каждую задачу в отдельности, что упрощает выявление проблемных участков.

ДОКУМЕНТИРОВАНИЕ ФУНКЦИЙ

Программистам часто приходится использовать функции, написанные другими людьми. Например, при работе в команде над большим сайтом. Документация помогает программистам осваивать эти функции.

Чтобы использовать функцию на своей странице, вам необязательно понимать, как работает код у нее внутри. Все, что вам надо знать, это:

- что должна делать функция;
- название функции;
- необходимые параметры;
- что функция должна вернуть.

На странице справа вы можете увидеть страницу веб-сайта `PHP.net`. Это официальный сайт PHP, который содержит документацию по языку программирования PHP. Это очень полезный ресурс для изучения языка.

Там приведена страница документации для функции, определяющей количество символов в строке. Это типичный пример документирования функций. Как правило, в документации пишется следующее:

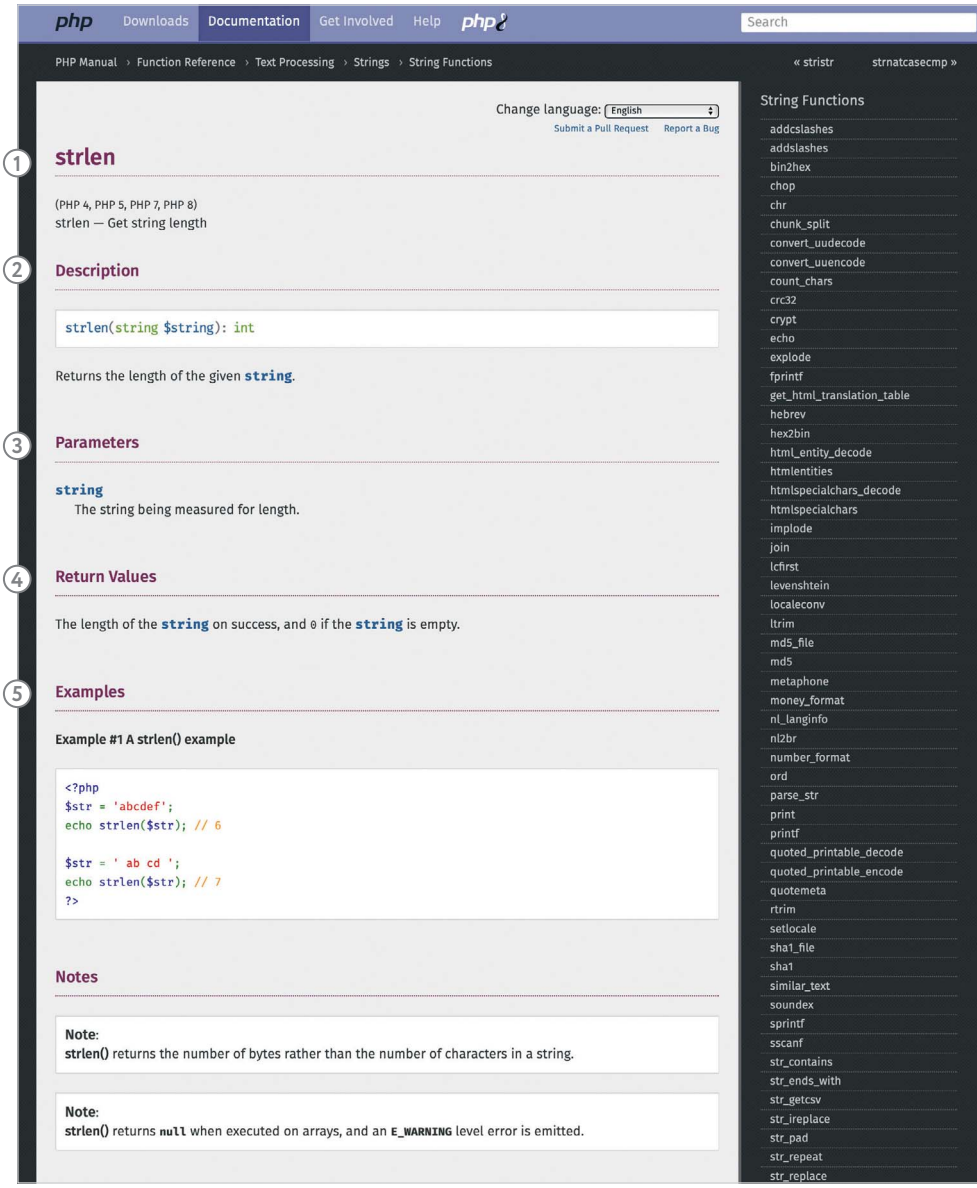
1) имя функции и ее описание;
2) синтаксис вызова функции и ее параметров (могут отображаться типы аргументов и возвращаемых значений);
3) описание параметров;
4) возвращаемые функцией значения;
5) пример использования функции.

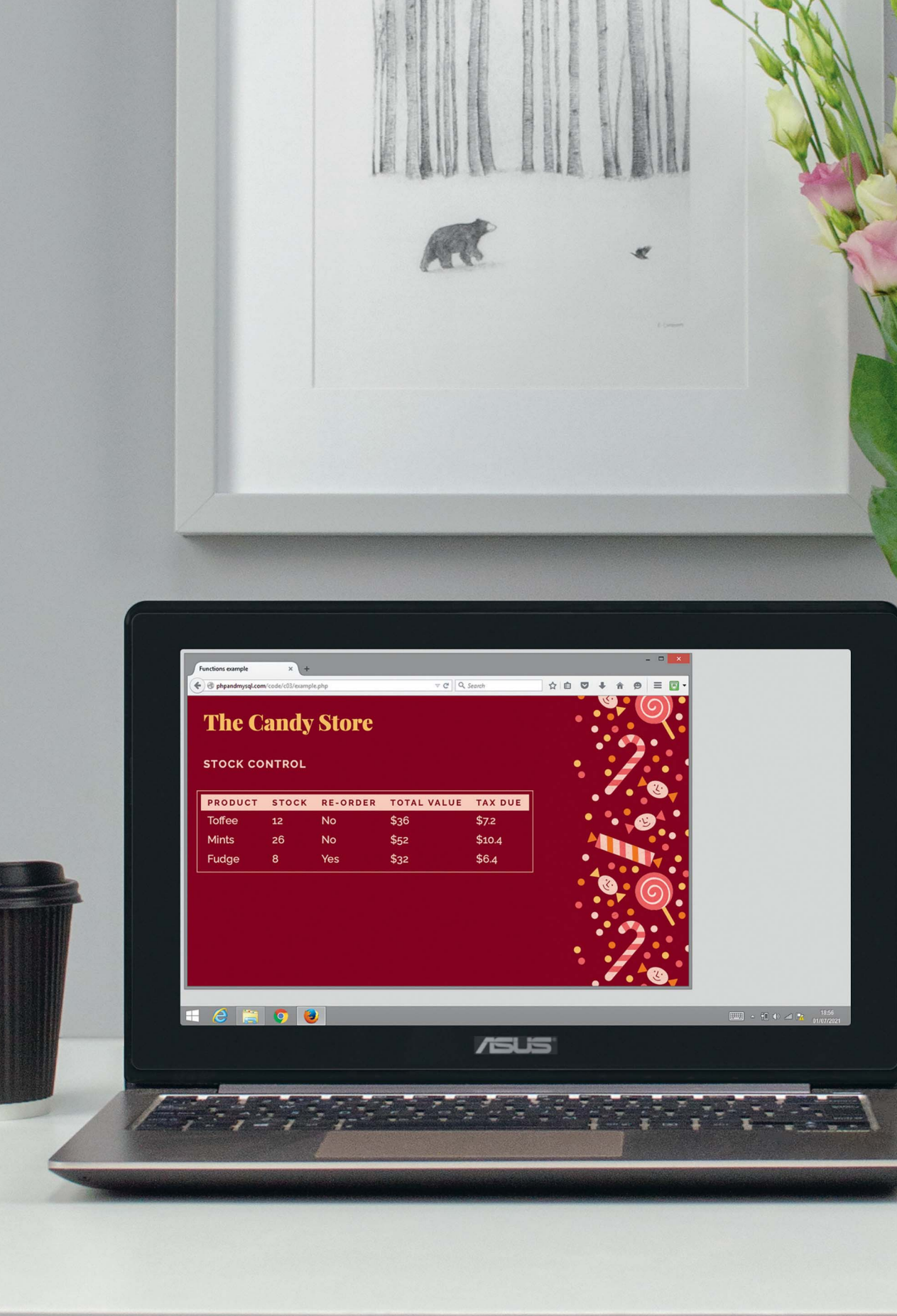
Важно различать два типа функций.

- **Пользовательские функции** — это функции, которые определены программистом в файле на языке PHP. Все функции в этой главе определены пользователем.

- **Встроенные функции** — это функции, которые уже написали создатели языка программирования. Это означает, что кто угодно может просто использовать их в любом месте кода.

Встроенные функции позволяют выполнять наиболее распространенные задачи, которые приходится решать программистам при написании кода PHP. Таким образом, встроенные функции избавляют разработчиков от необходимости писать собственный код. Более подробно о встроенных функциях вы узнаете в главе 5.


php
Downloads
Documentation
Get Involved
Help
Search
PHP Manual › Function Reference › Text Processing › Strings › String Functions
« stristr
strnatcasecmp »
Change language: English
Submit a Pull Request
Report a Bug
1
strlen
(PHP 4, PHP 5, PHP 7, PHP 8)
strlen — Get string length
2
Description
strlen(string $string): int
Returns the length of the given string.
3
Parameters
string
The string being measured for length.
4
Return Values
The length of the string on success, and 0 if the string is empty.
5
Examples
Example #1 A strlen() example
<?php
$str = 'abcdef';
echo strlen($str); // 6
$str = ' ab cd ';
echo strlen($str); // 7
?>
Notes
Note:
strlen() returns the number of bytes rather than the number of characters in a string.
Note:
strlen() returns null when executed on arrays, and an E_WARNING level error is emitted.
String Functions
addcslashes
addslashes
bin2hex
chop
chr
chunk_split
convert_uudecode
convert_uuencode
count_chars
crc32
crypt
echo
explode
fprintf
get_html_translation_table
hebrev
hex2bin
html_entity_decode
htmlentities
htmlspecialchars_decode
htmlspecialchars
implode
join
lcfirst
levenshtein
localeconv
ltrim
md5_file
md5
metaphone
money_format
nl_langinfo
nl2br
number_format
ord
parse_str
print
printf
quoted_printable_decode
quoted_printable_encode
quotemeta
rtrim
setlocale
sha1_file
sha1
similar_text
soundex
sprintf
sscanf
str_contains
str_ends_with
str_getcsv
str_ireplace
str_pad
str_repeat
str_replace


Functions example
phpandmysql.com/code/c03/example.php
Search
The Candy Store
STOCK CONTROL
PRODUCT STOCK RE-ORDER TOTAL VALUE TAX DUE
Toffee 12 No $36 $7.2
Mints 26 No $52 $10.4
Fudge 8 Yes $32 $6.4
ASUS



ИТОГОВЫЙ ПРИМЕР

Рассмотрим пример создания страницы для отслеживания количества товаров в кондитерской.

Создадим ассоциативный массив для хранения названий товаров, продаваемых в кондитерской, и имеющегося на складе количества для каждого из них.

Затем для генерации выводимых на странице значений создадим три функции.

- Первая отслеживает количество товара на складе и выводит сообщение о необходимости заказать дополнительный товар.
- Вторая определяет общую стоимость каждого проданного товара.
- Третья рассчитывает сумму подлежащего уплате налога, когда все оставшиеся запасы будут распроданы.

section_a/c03/example.php PHP

```php
<?php
declare(strict_types = 1);
$candy = [
    'Toffee' => ['price' => 3.00, 'stock' => 12],
    'Mints'  => ['price' => 2.00, 'stock' => 26],
    'Fudge'  => ['price' => 4.00, 'stock' => 8],
];
$tax = 20;

function get_reorder_message(int $stock): string
{
    return ($stock < 10) ? 'Yes' : 'No';
}

function get_total_value(float $price, int $quantity): float
{
    return $price * $quantity;
}

function get_tax_due(float $price, int $quantity, int $tax = 0): float
{
    return ($price * $quantity) * ($tax / 100);
}
?>
```

1. Включим строгую типизацию.
2. Создадим многомерный массив и сохраним его в переменной `$candy`:
- ключи — это названия продаваемых сладостей;
- значения — это массивы, содержащие цену и количество на складе для данного товара.

3. Объявим глобальную переменную для хранения значения налоговой ставки.
4. Определим функцию `get_reorder_message()`. Функция использует один параметр — текущее количество товара на складе (`int`) и возвращает сообщение (строку) о том, следует ли дозаказать товар.
5. Для возврата сообщения используется тернарный оператор. В условии проверяется, меньше ли количество товара, чем 10:
- если условие выполняется, функция возвращает `Yes` (Да);
- в противном случае функция возвращает `No` (Нет).

6. Определим функцию `get_total_value()`. Функция будет использовать следующие параметры:
- цена товара (`float`);
- количество данного товара в наличии (`int`).

Функция возвращает число с плавающей запятой, указывающее общую стоимость запасов для данного товара (в данном случае `int` также будет допустимо).
7. Функция возвращает цену товара, помноженную на количество товара, имеющегося в наличии.
8. Определим функцию `get_tax_due()`. Она использует следующие параметры:
- цена товара (`float`);
- количество данного товара в наличии (`int`);
- налоговая ставка в процентах со значением по умолчанию 0% (`int`).

Функция возвращает число с плавающей запятой, указывающее общую сумму налога, которая будет уплачена после продажи товара.

PHP section_a/c03/example.php

```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>The Candy Store</h1>
    <h2>Stock Control</h2>
    <table>
      <tr>
        <th>Candy</th><th>Stock</th><th>Re-order</th><th>Total value</th><th>Tax due</th>
      </tr>
⑩     <?php foreach ($candy as $product_name => $data) { ?>
⑪       <tr>
          <td><?=  $product_name ?></td>
⑫         <td><?=  $data['stock'] ?></td>
⑬         <td><?=  get_reorder_message($data['stock']) ?></td>
⑭         <td>$<?= get_total_value($data['price'], $data['stock']) ?></td>
⑮         <td>$<?= get_tax_due($data['price'], $data['stock'], $tax) ?></td>
        </tr>
⑯     <?php } ?>
    </table>
  </body>
</html>
```

9. Возвращается итоговая сумма налога. Чтобы ее рассчитать, общая стоимость товара на складе (цена одного товара, умноженная на доступное количество) умножается на процент налога (ставка налога, разделенная на 100).

10. Цикл `foreach` перебирает товары из массива, лежащего в переменной `$candy`. В скобках указываются следующие параметры:

- `$candy` — переменная, хранящая массив, созданный в шаге 2;
- `$product` — имя переменной, которая будет содержать ключ для текущего элемента массива (название товара `toffee`, `mints` или `fudge`);
- `$data` — переменная, которая будет представлять значение для текущего элемента (это массив, в котором хранится цена и количество товара).

11. Создадим в HTML-таблице строку. Название товара, который в данный момент обрабатывается в цикле, будет выводиться в элементе `<td>`.

12. Переменная `$data` — массив, содержащий цену и количество товара. Они будут выводиться в следующей ячейке таблицы.

13. Вызов функции `get_reorder_message()`. Количество товара будет передаваться в качестве аргумента. Возвращенное значение будет выведено в таблицу.

14. Вызов функции `get_total_value()`. Первый параметр — цена товара. Второй параметр — доступное количество товара. Возвращенное значение будет выведено в таблицу.

15. Вызов функции `get_tax_due()`. Первый параметр — цена товара. Второй параметр — доступное количество товара. Третий параметр — налоговая ставка, сохраненная в шаге 3. Возвращенное значение будет записано в таблицу.

16. Закрывающая фигурная скобка указывает на конец блока кода. Цикл повторяется для каждого элемента массива.

ЗАКЛЮЧЕНИЕ
ФУНКЦИИ

- При определении функции ей присваивается имя и создается блок кода для выполнения задачи.
- Вызов функции говорит интерпретатору PHP выполнить код, содержащийся в этом блоке.
- Ключевое слово `return` возвращает данные из функции.
- Параметры — это данные, которые необходимы функции для выполнения задачи. Переданные параметры становятся локальными переменными в функции.
- После завершения работы функции все параметры и переменные, объявленные в функции, удаляются.
- При вызове функции значения параметров называются аргументами.
- Объявление типов задает тип данных для аргументов.
- Тип возвращаемых данных определяет тип возвращаемого функцией значения.
- Если параметр необязательный, ему присваивается значение по умолчанию.

4

КЛАССЫ И ОБЪЕКТЫ

Объекты группируют вместе переменные и функции, представляющие понятия или сущности, с которыми вы сталкиваетесь в повседневной жизни, — новостные статьи, товары для продажи или пользователи веб-сайта.

- В главе 2 вы узнали, как переменные могут содержать отдельные фрагменты информации. При использовании переменной в объекте она называется **свойством** объекта.
- В главе 3 вы узнали, как функции могут представлять выполняемую вашим кодом задачу. При использовании функции в объекте она вызывается **методом** объекта.

Веб-сайтам часто приходится работать с множеством сущностей одного типа. Новостной сайт публикует много статей, магазин продает множество товаров, а сайт, позволяющий посетителям регистрироваться, будет хранить большой список пользователей. Каждая из этих вещей может быть представлена в коде с помощью **объекта**.

Шаблон для создания объектов называется **классом**. Он представляет собой определенный тип данных. Например, PHP может использовать один класс для создания объектов, представляющих продукты, а другой класс — для создания объектов, представляющих учетные записи пользователей. Каждому объекту, созданному с использованием класса, автоматически присваиваются свойства и методы, определенные в этом классе.

Объекты и классы помогают упорядочить ваш код и облегчить его понимание. Кроме того, узнать, как работают объекты, важно еще и потому, что в PHP есть несколько встроенных классов. С ними вы познакомитесь в разделе Б этой книги.

Первые несколько страниц этой главы описывают концепции, лежащие в основе объектов, и то, как они используются. После этого вы изучите код, необходимый для создания и использования объектов и классов.


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon 13 Sep 15:09
localhost:8888/phpbook/section_a/c04/example.php
NEO BANK
NAME: IVY STONE
ACCOUNT NUMBER
ACCOUNT TYPE
BALANCE
20489446
Checking
$ -20
20148896
Savings
$ 380

ВЕБ-САЙТ КАК МОДЕЛЬ

Модели — это представления вещей из окружающего нас мира. Программисты создают модели, используя данные. Затем они применяют код для выполнения задач, которые управляют данными, хранящимися в этих моделях.

Веб-сайты используют данные для создания моделей, которые представляют сущности из повседневной жизни. Программисты часто называют эти *сущности* различными **типами объектов**:

- люди (клиенты или зарегистрированные пользователи сайта);
- товары или услуги, которые могут приобрести посетители (книги, автомобили, банковские счета или подписки на телевидение);
- документы, которые раньше печатались на бумаге (новостные статьи, календари или билеты).

Например, банку могут потребоваться определенные фрагменты данных для представления каждого клиента:

- имя;
- фамилия;
- адрес электронной почты;
- пароль.

Для этого потребуется единая структура данных с информацией о каждом человеке, в которой имена, адреса электронной почты и пароли для каждого клиента будут разными.

Также потребуется знать одни и те же фрагменты данных о каждом банковском счете, но значения, используемые для представления каждого счета, будут разными. Например:

- номер счета;
- тип счета;
- баланс.

Банк может выполнять различные задачи, используя эти данные. Например, операции, выполняемые с помощью банковского счета, могут включать:

- проверка баланса;
- внесение депозита;
- вывод средств.

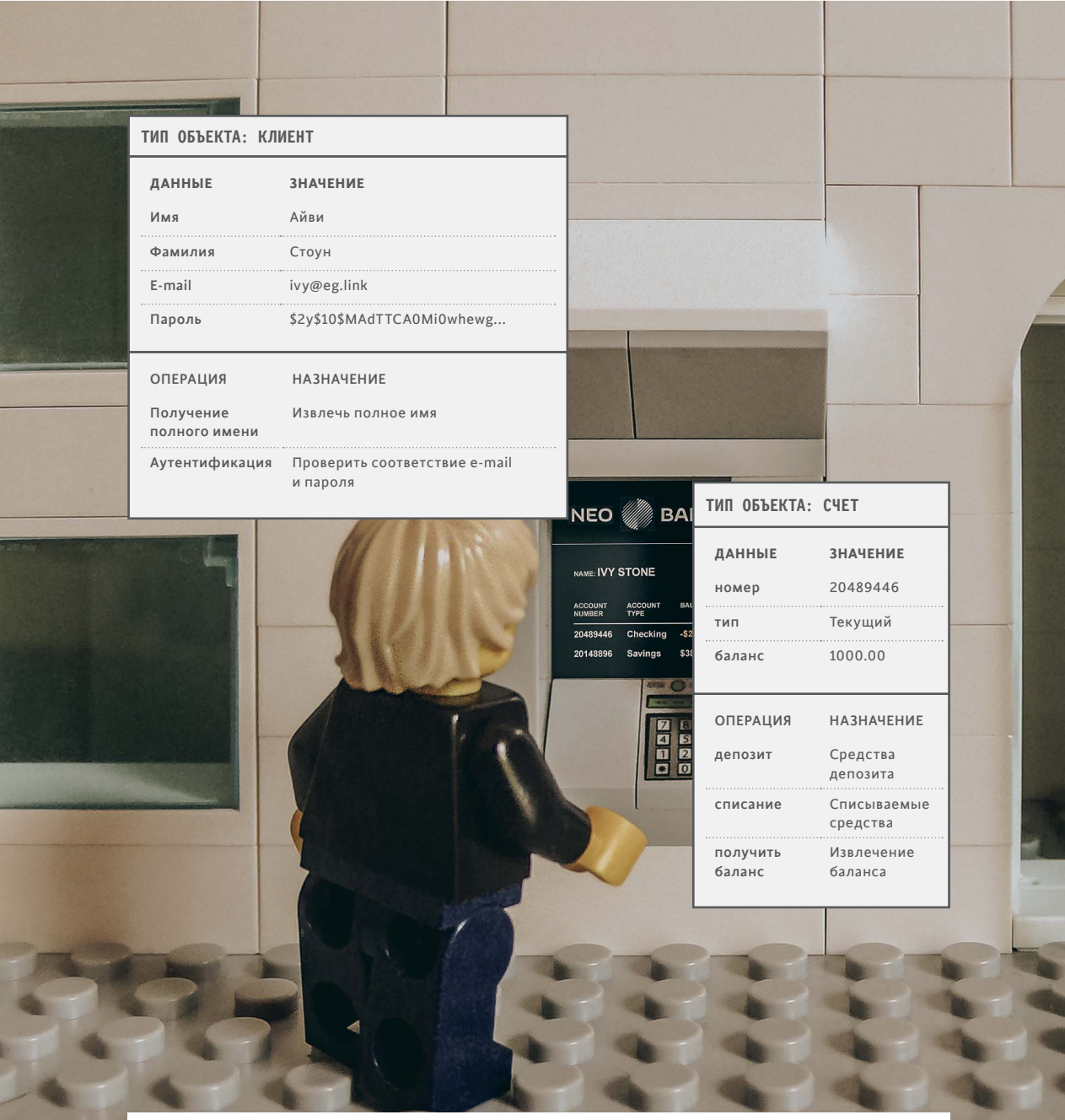
Подобные задачи считывают или обновляют данные, хранящиеся в переменных. Например, если вы снимете деньги или внесете депозит, это изменит сумму, сохраненную на балансе этого счета. Аналогично этому связанные с клиентом задачи могут включать в себя:

- аутентификация пользователя (подтверждение того, что он тот, за кого себя выдает). Проверяется, что указанный им адрес электронной почты и пароль соответствуют сохраненным для него данным;
- получение полного имени пользователя (путем объединения имени и фамилии).

Объект группирует:

- переменные, которые хранят данные, необходимые для создания модели для определенной сущности, например клиента или учетной записи;
- функции, представляющие задачи, которые может выполнять объект этого типа.

Если убрать иллюстрацию справа, вы все равно сможете многое рассказать о представленных на ней объектах, посмотрев на информацию в таблицах: типы объектов, данные, необходимые для представления каждого из них, и задачи, которые могут выполнять объекты.



ТИП ОБЪЕКТА: КЛИЕНТ

ДАННЫЕ	ЗНАЧЕНИЕ
Имя	Айви
Фамилия	Стоун
E-mail	ivy@eg.link
Пароль	$2y$10$MAdTTCA0Mi0whewg...

ОПЕРАЦИЯ	НАЗНАЧЕНИЕ
Получение полного имени	Извлечь полное имя
Аутентификация	Проверить соответствие e-mail и пароля

ТИП ОБЪЕКТА: СЧЕТ

ДАННЫЕ	ЗНАЧЕНИЕ
номер	20489446
тип	Текущий
баланс	1000.00

ОПЕРАЦИЯ	НАЗНАЧЕНИЕ
депозит	Средства депозита
списание	Списываемые средства
получить баланс	Извлечение баланса

В информационных таблицах представлено два типа объектов: **клиент** и **банковский счет**. Для каждого типа объекта веб-сайт должен делать две вещи.

1. **Содержать фрагменты данных, используемые для представления объектов**. Структура данных, которые он хранит, одинакова для всех клиентов или счетов, но значения, представляющие этих клиентов или счета, различны.

2. **Выполнять одинаковые операции с этим типом объекта**. Вы можете выполнять одни и те же операции с каждым клиентом либо счетом.

Эти операции могут получать доступ к данным, хранящимся для каждого клиента или счета, либо изменять их. Например, когда на счет вносится депозит, то меняется значение, в котором хранится баланс счета.

СВОЙСТВА И МЕТОДЫ

В объекте переменные называются **свойствами**, а функции — **методами**. Свойства хранят данные, необходимые для создания модели той или иной сущности. Методы представляют задачи, которые может выполнять объект данного типа.

ПЕРЕМЕННЫЕ: СВОЙСТВА ОБЪЕКТА

В главе 1 вы узнали, что переменные могут содержать данные, изменяющиеся при каждом запросе страницы. Когда переменные используются внутри объекта, они называются свойствами объекта.

Планируя создание класса, программист должен решить, какие данные объект этого типа должен содержать, чтобы выполнять свою работу.

Например, если объекты используются для представления клиентов, каждый объект клиента будет содержать в себе:

- *одинаковые* свойства для хранения имени, фамилии, адреса электронной почты и пароля; но
- *отличающиеся* значения этих свойств для представления каждого клиента.

Если вы используете объект для представления учетной записи, каждый объект учетной записи также будет содержать:

- *одинаковые* свойства для хранения номера счета, типа счета и баланса; но
- *отличающиеся* значения этих свойств для представления каждого счета.

Свойства объекта — это набор переменных, которые описывают индивидуальные *характеристики*, общие для всех этих объектов. Значения, сохраняющиеся для каждого свойства, это то, что отличает один объект от другого.

ФУНКЦИИ: МЕТОДЫ ОБЪЕКТА

В главе 3 вы узнали, что PHP может использовать функции, чтобы сгруппировать все операторы, необходимые для выполнения определенной задачи. Когда функции используются внутри объекта, они называются методами объекта.

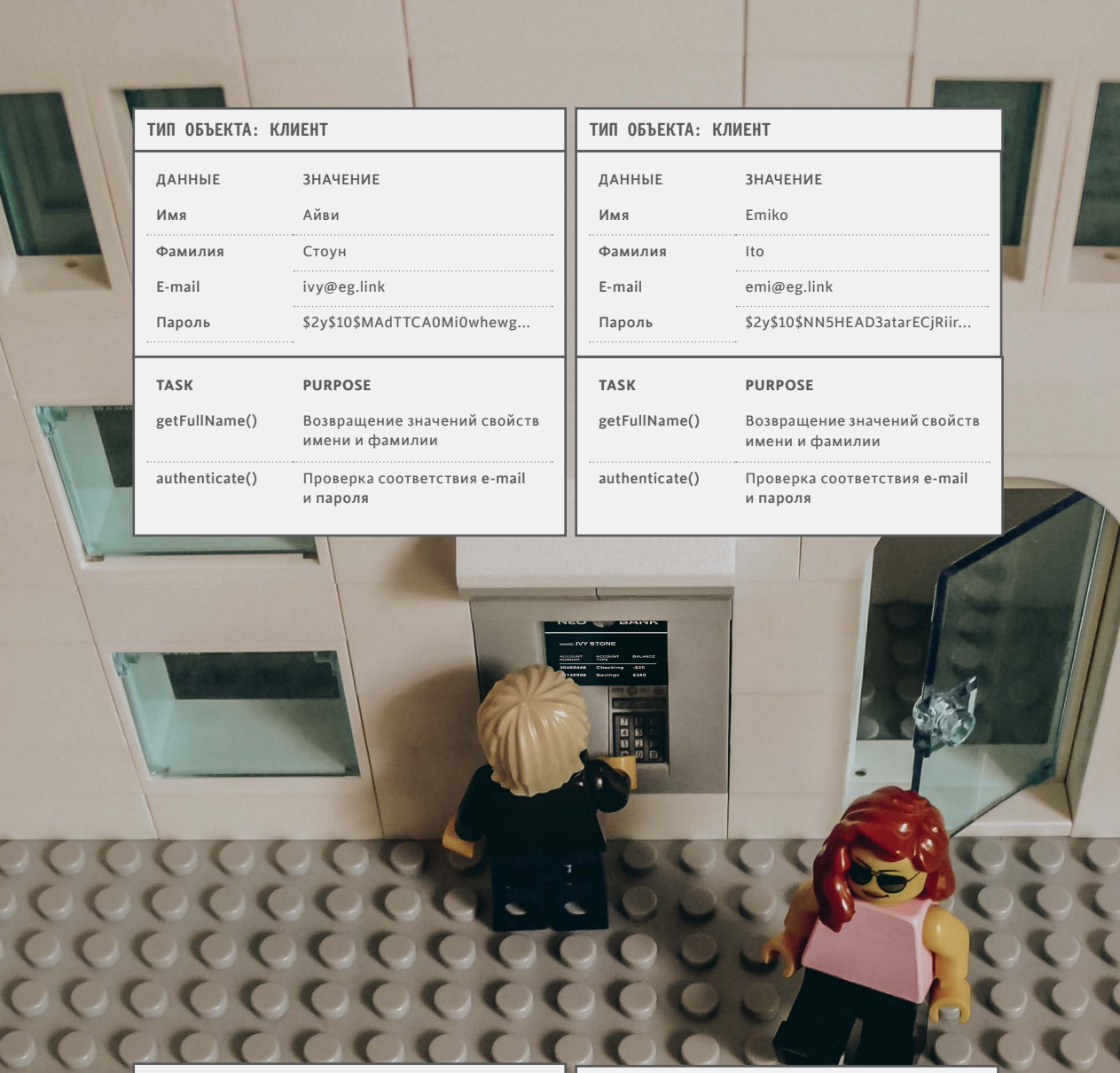
Планируя создание класса, программист решает, какие операции пользователи веб-сайта могут выполнять с каждым объекта этого типом. Эти задачи обычно включают:

- запросы для получения информации об объекте (используются данные, содержащиеся в его свойствах);
- изменение значений, содержащихся в одном или нескольких свойствах этого объекта.

Задачи, которые вы можете выполнять со счетом (внести депозит, снять деньги или проверить баланс), применимы к каждому счету. Поэтому все объекты, представляющие счет, содержат одинаковый набор методов.

Аналогично вам нужно будет выполнять одни и те же задачи для каждого клиента (аутентифицировать его, получить полное имя), поэтому каждый объект, представляющий клиента, будет содержать одни и те же методы.

Справа вы можете увидеть схему, аналогичную схеме на предыдущей странице, но на этот раз на ней показаны имена свойств и методов для двух объектов «Клиент» и двух объектов «Счет».



ТИП ОБЪЕКТА: КЛИЕНТ

ДАННЫЕ	ЗНАЧЕНИЕ
Имя	Айви
Фамилия	Стоун
E-mail	ivy@eg.link
Пароль	$2y$10$MAdTTCA0Mi0whewg...

TASK	PURPOSE
getFullName()	Возвращение значений свойств имени и фамилии
authenticate()	Проверка соответствия e-mail и пароля

ТИП ОБЪЕКТА: КЛИЕНТ

ДАННЫЕ	ЗНАЧЕНИЕ
Имя	Emiko
Фамилия	Ito
E-mail	emi@eg.link
Пароль	$2y$10$NN5HEAD3atarECjRiir...

TASK	PURPOSE
getFullName()	Возвращение значений свойств имени и фамилии
authenticate()	Проверка соответствия e-mail и пароля

ТИП ОБЪЕКТА: СЧЕТ

ДАННЫЕ	ЗНАЧЕНИЕ
Номер	20489446
Тип	Текущий
Баланс	1000.00

ОПЕРАЦИЯ	НАЗНАЧЕНИЕ
deposit()	Увеличение значения свойства баланса
withdraw()	Уменьшение значения свойства баланса
getBalance()	Возвращение значения свойства баланса

ТИП ОБЪЕКТА: СЧЕТ

ДАННЫЕ	ЗНАЧЕНИЕ
Номер	10937528
Тип	Сбережения
Баланс	2346.00

ОПЕРАЦИЯ	НАЗНАЧЕНИЕ
deposit()	Увеличение значения свойства баланса
withdraw()	Уменьшение значения свойства баланса
getBalance()	Возвращение значения свойства баланса

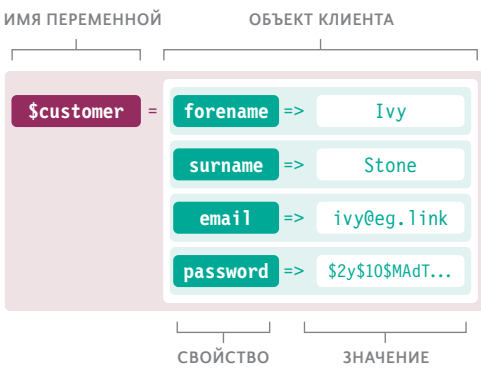
ТИП ДАННЫХ «ОБЪЕКТ»

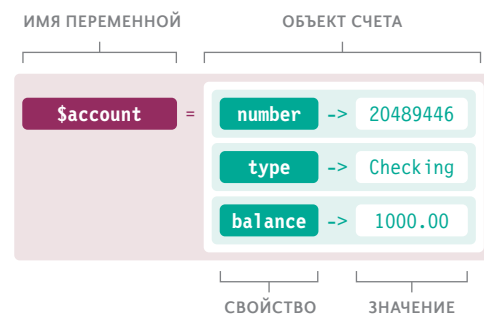
Объект — это пример составного типа данных, поскольку он может хранить несколько значений.

Как вы уже знаете, PHP поддерживает разные типы данных:

- скалярные типы данных содержат отдельные значения (строки, целые числа, числа с плавающей запятой, логические значения);
- составные типы данных, которые содержат несколько значений (массивы и объекты).

Ниже вы можете увидеть две схемы объектов, представляющие клиента и его счет.





Переменная, содержащая объект, может быть названа так же, как и любая другая переменная (в нижнем регистре, с подчеркиванием для разделения каждого слова, если в имени используется несколько слов). Например:

- переменная с именем `$customer` может содержать объект, представляющий клиента;
- переменная с именем `$account` может содержать объект, представляющий счет.

Люди часто говорят, что объект содержится в переменной, но переменная на самом деле только содержит **ссылку** на то место в памяти интерпретатора PHP, где был создан объект.

Когда скрипт завершает работу и интерпретатор PHP отправляет HTML-страницу в браузер, он «забывает» объект (удаляет его из своей памяти точно так же, как «забывает» все значения, хранящиеся в переменных).

КЛАССЫ — ШАБЛОНЫ ДЛЯ СОЗДАНИЯ ОБЪЕКТОВ

Для создания объектов используется шаблон, называемый классом. Определение класса содержит имена свойств и методы, которыми обладает объект данного типа.

Определение класса задает:

- имена свойств, описывающие данные, которые требуются в объектах этого типа;
- методы, определяющие операции, которые могут быть выполнены с помощью такого объекта.

При каждом создании объекта с помощью класса:

- вы можете указать значения для некоторых свойств (эти значения отличают один объект от другого);
- объект автоматически получает все методы, которые были определены в классе.

Каждый отдельный объект, созданный с использованием класса, называется **экземпляром** этого класса.

Например, при создании объекта для представления банковского счета (как показано в этой главе) вы должны указать значения для следующих свойств:

- `$number` (номер);
- `$type` (тип);
- `$balance` (баланс).

И объект автоматически получит методы:

- `deposit();`
- `withdraw();`
- `getBalance().`

Некоторые программисты используют термины «класс» и «объект» как взаимозаменяемые, но, строго говоря, класс — это шаблон, который используется для создания объекта.

КАК СОЗДАВАТЬ И ИСПОЛЬЗОВАТЬ ОБЪЕКТЫ

Ниже представлены шаги, которые вам необходимо освоить для создания и использования классов и объектов.

СОЗДАНИЕ КЛАССА КАК ШАБЛОНА ДЛЯ ОБЪЕКТА

Класс — это шаблон, который используется для создания объектов одного типа.

Класс определяет:

- свойства, в которых хранятся данные, используемые в объектах этого типа;
- методы, содержащие инструкции для выполнения операций, доступных объекту такого типа.

Для каждого типа объектов, с которыми должен работать веб-сайт, создаются свои классы.

СОЗДАНИЕ ОБЪЕКТА И СОХРАНЕНИЕ ЕГО В ПЕРЕМЕННОЙ

Чтобы создать объект, нужно:

- указать имя класса, используемого в качестве шаблона для объекта;
- предоставить значения для его свойств.

Объект автоматически получит методы, определенные в классе.

Объект при создании обычно сохраняется в переменной, чтобы его можно было использовать в остальной части кода на странице.

ПРИСВОЕНИЕ ЗНАЧЕНИЙ СВОЙСТВАМ И ДОСТУП К НИМ

После создания объекта можно:

- задать значения для его свойств;
- получить доступ к значениям, хранящимся в его свойствах, и использовать их в остальной части кода.

Разные объекты одного класса будут содержать собственные значения в своих свойствах, поскольку они представляют разные экземпляры данного класса (например, другого клиента или другой счет).

ОПРЕДЕЛЕНИЕ И ВЫЗОВ МЕТОДОВ ОБЪЕКТА

Метод содержит инструкции, необходимые для выполнения тех операций, которые должен выполнять объект данного типа. Определение метода пишется так же, как определение функции, но является частью класса. Методы обычно возвращают результат своей работы — так же, как это делают функции.

Когда вызывается метод объекта, ему часто требуется получить доступ к значениям, хранящимся в свойствах этого объекта, или обновить их.

В PHP есть специальная переменная с именем `$this`. Она позволяет методам обращаться к значениям, хранящимся в свойствах этого объекта.

ПРИСВОЕНИЕ ЗНАЧЕНИЙ СВОЙСТВАМ ПРИ СОЗДАНИИ ОБЪЕКТОВ

Вместо того чтобы создавать объект и затем задавать значения для каждого из его свойств по отдельности, вы можете создать объект и присвоить значения свойствам в одной строке кода. Для этого в класс добавляется функция под названием **конструктор**.

В PHP 8 эта функция также может определять свойства объекта (и тогда вам не придется определять их отдельно).

УПРАВЛЕНИЕ ДОСТУПОМ К СВОЙСТВАМ ИЗ КОДА

Иногда требуется запретить коду PHP получать доступ к свойствам объекта напрямую. Вместо этого вы можете создавать методы для получения или обновления значений, хранящихся в этих свойствах.

Например, вы можете скрыть свойство `balance` объекта `account`, а затем использовать методы `getBalance()`, `deposit()` и `withdraw()` для работы со значением, хранящимся в свойстве `balance`.

КЛАССЫ: ШАБЛОНЫ ДЛЯ ОБЪЕКТОВ

Когда создается **определение класса**, свойства и методы, которыми будет обладать объект, пишутся внутри фигурных скобок.

В определении класса применяется:

- ключевое слово `class`;
- имя, описывающее тип создаваемого объекта. В его создании должен использоваться CamelCase (**ГорбатыйРегистр**, стиль верблюда или верблюжий **регистр**[13]), где каждое слово будет начинаться с заглавной буквы (и не используется подчеркивание для разделения слов);
- пара фигурных скобок для создания блока кода. Фигурные скобки показывают, где начинается и заканчивается класс. Каждая фигурная скобка должна находиться на новой строке.

Примечание. После закрывающей фигурной скобки, завершающей определение класса, точка с запятой не ставится.

Внутри фигурных скобок класса свойства для этого типа объектов задаются с помощью:

- ключевого слова, определяющего видимость свойства. Те, что приведены ниже, используют ключевое слово `public` (общедоступная область видимости);
- типа данных для этого свойства (это было добавлено в PHP 7.4 и необязательно);
- имени свойства, начинающегося с символа $.

Методы используют тот же синтаксис, что и определения функций, но им предшествует ключевое слово, определяющего видимость метода. В приведенных ниже методах используется `public`.

```
class Account
{
    public int    $number;
    public string $type;
    public float  $balance;

    public function deposit(float $amount): float
    {
        // Code to deposit money here
    }
    public function withdraw(float $amount): float
    {
        // Code to withdraw money here
    }
}
```

СВОЙСТВА

МЕТОДЫ

СОЗДАНИЕ ОБЪЕКТА С ИСПОЛЬЗОВАНИЕМ КЛАССА

Чтобы создать объект, используйте ключевое слово `new`, за которым следует имя класса и пара круглых скобок.

Для создания объекта используйте:

- ключевое слово `new`;
- имя класса, который будет шаблоном для объекта;
- круглые скобки. Они могут содержать имена параметров (так же, как функция получает параметры в круглых скобках). С их помощью при создании объекта передаются данные в конструктор.

Ссылка на объект часто содержится в переменной, чтобы ее можно было использовать в остальном коде страницы PHP. Чтобы сделать это:

- создайте переменную для хранения объекта. Ее имя должно описывать тип объекта, который она содержит;
- добавьте оператор присваивания `=`;
- создайте объект (как описано слева).



В приведенном выше примере переменная `$account` будет содержать ссылку на объект, созданный с использованием класса `Account`. Он показан на левой странице.

Этот объект будет содержать три свойства: `$number`, `$type` и `$balance`. Ни в одном из них еще нет значений. Вы увидите, как присвоить им значения, на следующей странице.

Объект автоматически будет содержать два метода, которые были добавлены в определении класса.

Для создания второго объекта, представляющего другую учетную запись, вы должны сделать то же самое, что показано выше, но с другим именем переменной (в противном случае второй объект перезапишет первый).

Определение класса должно быть на любой странице, которая создает объект с использованием класса. Если определение класса используется более чем на одной странице, его стоит поместить в отдельный файл, который затем может быть подключен на обеих страницах. Этому файлу присваивается то же имя, что и классу (например, `Account.php`).

ДОСТУП К СВОЙСТВАМ И ИХ ИЗМЕНЕНИЕ

Прочитать значение свойства объекта или изменить его можно так же, как и при работе с переменными. Если объект содержится в переменной, сначала укажите имя переменной, а затем используйте объектный оператор и после него имя свойства, с которым вы хотите работать.

ДОСТУП К СВОЙСТВАМ

Чтобы получить доступ к значению, хранящемуся в свойстве, используйте:

- имя переменной, содержащей объект;
- оператор объекта -> без пробелов вокруг;
- имя свойства (обратите внимание, что имя свойства здесь не начинается с символа $).

Объектный оператор указывает, что свойство справа от оператора принадлежит объекту в переменной слева от оператора.

Приведенный ниже код показывает, как отобразить значение баланса счета на странице.



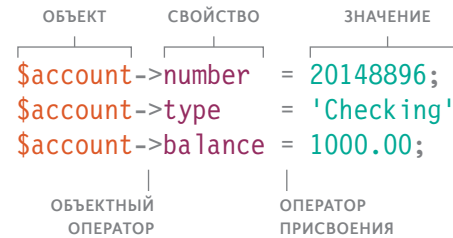
Если для свойства был задан тип данных и вы пытаетесь получить доступ к свойству до того, как ему было присвоено значение, интерпретатор PHP выдаст ошибку, которая остановит дальнейшее выполнение кода.

УСТАНОВКА И ИЗМЕНЕНИЕ ЗНАЧЕНИЯ СВОЙСТВА

Чтобы изменить значение, содержащееся в свойстве, используйте:

- имя переменной, содержащей объект;
- объектный оператор -> без пробелов вокруг;
- имя свойства, которое хотите обновить;
- оператор присваивания =;
- новое значение (если значение — это строка, оно заключено в кавычки; числа и логические значения в кавычки не заключаются).

Если вы попытаетесь установить значение свойства, которого не было в определении класса, свойство будет добавлено к одному этому объекту[14]. Ни к каким другим объектам, созданным с помощью этого класса, оно не будет добавлено.



На стр. 166 вы узнаете, что можно указать значения по умолчанию для свойств в методе `__construct()`. Это гарантирует, что каждое свойство будет содержать значение при создании объекта с использованием класса.

ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ И СВОЙСТВ

PHP section_a/c04/objects-and-properties.php

```php
<?php
class Customer
{
    public string $forename;
    public string $surname;
    public string $email;
    public string $password;
}

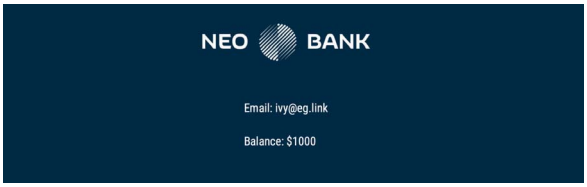
class Account
{
    public int    $number;
    public string $type;
    public float  $balance;
}

$customer = new Customer();
$account  = new Account();
$customer->email  = 'ivy@eg.link';
$account->balance = 1000.00;
?>
<?php include 'includes/header.php'; ?>
<p>Email: <?= $customer->email ?></p>
<p>Balance: $<?= $account->balance ?></p>
<?php include 'includes/footer.php'; ?>
```

1. Создается класс `Customer` (клиент) и его свойства.
2. Создается класс `Account` (счет) и его свойства.
3. Создается экземпляр класса `Customer`, и этот объект присваивается переменной с именем `$customer`.
4. Создается экземпляр класса `Account`, и этот объект присваивается переменной с именем `$account`.
5. Свойству `email` объекта `Customer` присваивается значение.
6. Свойству `balance` (баланс) объекта `Account` присваивается значение.
7. Подключаемый файл добавляет заголовок на страницу.
8. Отображаются два свойства, которые только что были заданы.
9. Подключаемый файл добавляет HTML-теги, необходимые для закрытия страницы.

Упражнение. После шага 5 задайте имя и фамилию клиента в объекте `Customer`. Затем, в шаге 8, отобразите их содержимое перед адресом электронной почты.

РЕЗУЛЬТАТ



ОПРЕДЕЛЕНИЕ И ВЫЗОВ МЕТОДОВ

Метод — это функция, которая записана внутри определения класса. Для вызова метода используйте имя переменной, содержащей объект, объектный оператор, а затем имя метода.

ОПРЕДЕЛЕНИЕ МЕТОДА

Чтобы добавить метод в класс, используйте ключевое слово видимости метода; в приведенном ниже примере используется `public`), за которым следует определение функции. Если внутри метода необходимо получить доступ к свойству объекта, используйте:

- специальную переменную с именем `$this` (известную как **псевдопеременная**), указывающую, что вы хотите получить доступ к свойству данного объекта;
- оператор объекта `->`;
- имя свойства, к которому вы хотите получить доступ.

Приведенный ниже метод `deposit()` содержит параметр с именем `$amount`. При вызове метода значение, используемое для `$amount`, добавляется к значению в свойстве `balance`. Новое значение возвращается в `balance`.

```
class Account
{
  public int    $number;
  public string $type;
  public float  $balance;

  public function deposit($amount)
  {
    $this->balance += $amount;
    return $this->balance;
  }
}
```

$this — ПСЕВДОПЕРЕМЕННАЯ

ВЫЗОВ МЕТОДА

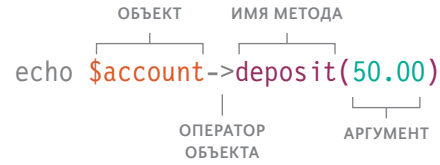
Для вызова метода используйте:

- имя переменной, содержащей объект;
- оператор объекта `->`;
- имя метода;
- аргументы для параметров метода.

В приведенном ниже примере на счет вносится 50 долларов.

Метод `deposit()` (показан в левом столбце) добавляет эту сумму к значению в свойстве `balance`; затем он возвращает новое значение `balance`.

Команда `echo` используется для вывода нового значения баланса на страницу[15].



ИСПОЛЬЗОВАНИЕ МЕТОДОВ ОБЪЕКТОВ

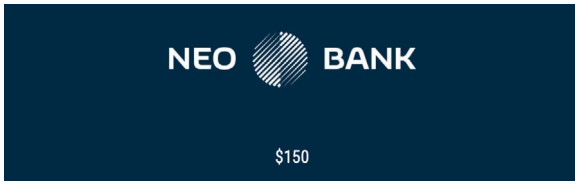
PHP section_a/c04/objects-and-methods.php

```php
<?php
class Account
{
    public int    $number;
    public string $type;
    public float  $balance;

    public function deposit(float $amount): float
    {
        $this->balance += $amount;
        return $this->balance;
    }
    public function withdraw(float $amount): float
    {
        $this->balance -= $amount;
        return $this->balance;
    }
}

$account = new Account();
$account->balance = 100.00;
?>
<?php include 'includes/header.php'; ?>
<p>$<?= $account->deposit(50.00) ?></p>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



1. Определение класса `Account` и его свойств.
2. Добавление метода `deposit()`. Параметр `$amount` — это сумма, которую нужно добавить к балансу.
3. Сумма, переданная в функцию, добавляется к значению, сохраненному в свойстве `balance`:

- `$this->balance` означает обращение к свойству `balance` этого объекта;
- оператор `+=` добавляет переданную сумму к балансу.

4. Возврат нового значения, сохраненного в свойстве `balance`.
5. Метод `withdraw()` устроен точно так же, но вычитает сумму из свойства `balance`.
6. Создается объект с использованием класса `Account` и сохраняется в переменной с именем `$account`.
7. Свойство `balance` объекта задается равным 100.00.
8. Метод `deposit()` вызывается для добавления $50.00 на счет. Он возвращает обновленный баланс, и это значение выводится на страницу с помощью сокращенной команды `echo`.

Упражнение. После шага 8 использовать метод `withdraw()`, чтобы списать $75.

ОСОБЕННЫЙ МЕТОД: КОНСТРУКТОР

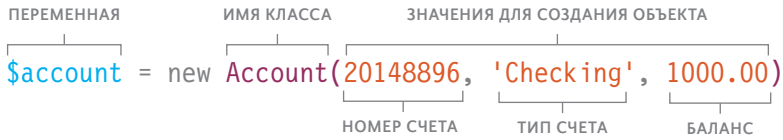
Метод `__construct()` известен как **конструктор**. Он выполняется автоматически при создании объекта.

Если вы добавите метод с именем `__construct()` в определение класса (его имя должно начинаться с двух знаков подчеркивания), то содержащийся в нем код будет автоматически выполнен при создании объекта.

Используйте метод `__construct()`, чтобы получить возможность создать объект и добавить значения к его свойствам в одной строке кода, вместо того чтобы создавать объект, а затем задавать каждое свойство по отдельности, используя отдельную команду для каждого.

Ниже создается объект класса `Account` и присваивается переменной с именем `$account`. В момент создания объекта, интерпретатор PHP ищет в классе метод с названием `__construct()`.

Аргументы в круглых скобках после имени класса передаются методу `__construct(`. Внутри метода `__construct()` эти значения используются для установки свойств объекта.



Примечание. Не начинайте имя ваших собственных функций с двух знаков подчеркивания. Это соглашение об именовании должно применяться только для тех элементов, которые в PHP называются **магическими методами**.

Магические методы автоматически вызываются интерпретатором PHP — вам не нужно вызывать их в собственном коде.

Приведенный ниже метод `__construct()` класса `Account` содержит три параметра: `$type`, `$number` и `$balance`, которые соответствуют свойствам этого класса.

Три инструкции в методе `__construct()` принимают значения параметров и используют их для установки значений свойств объекта.

Псевдопеременная `$this` позволяет вам получать доступ к свойствам этого объекта.

Если объект был создан с использованием кода на левой странице, метод `__construct()`, показанный ниже, автоматически запустится и присвоит:

- свойству `$number` — значение 20148896;
- свойству `$type` — значение Checking (Текущий);
- свойству `$balance` — значение 100.00.

В примере на следующей странице вы увидите, как задать значения по умолчанию для любого из этих свойств.

```
class Account
{
    public int    $number;
    public string $type;
    public float  $balance;

    public function __construct($number, $type, $balance)
    {
        $this->number  = $number;
        $this->type    = $type;
        $this->balance = $balance;
    }

    function deposit($amount) {...}
    function withdraw($amount) {...}
    function getBalance() {...}
}
```

В PHP 8 был добавлен более простой способ написания определений классов, позволяя вам объявлять свойства класса внутри круглых скобок метода `__construct()`.

Когда объект создается с использованием класса, аргументы, переданные в конструктор, автоматически присваиваются в качестве значений для соответствующих свойств. Это называется **определением свойств объекта в конструкторе (constructor property promotion)**.

Если свойство необязательное, ему можно задать значение по умолчанию (см. свойство `$balance` справа), и это значение будет использоваться, если аргумент не указан.

```
class Account
{
    public function __construct(
        public int    $number,
        public string $type,
        public float  $balance = 0.00,
        ) {}

    function deposit($amount) {...}
    function withdraw($amount) {...}
    function getBalance() {...}
}
```

ИСПОЛЬЗОВАНИЕ КЛАССОВ С КОНСТРУКТОРОМ

1. Страница PHP начинается с включения строгой типизации, поскольку в методах используется объявление типов.
2. Задается имя класса и его свойства.

3. Для установки значений свойств добавлен метод `__construct()` с предыдущей страницы. Объявления типов аргументов добавляются к параметрам. Если при создании объекта баланс не задан, используется значение по умолчанию `0.00`.

4. Методы `deposit()` и `withdraw()` изменяют значение, содержащееся в свойстве `balance`. В обоих используются аргументы и возвращаемые значения типа `float` (когда тип данных относится к `float`, использование `int` не вызовет ошибки).

section_a/c04/constructor-methods.php **PHP**

```php
<?php
declare(strict_types = 1);
class Account
{
    public int    $number;
    public string $type;
    public float  $balance;

    public function __construct(int $number, string $type, float $balance = 0.00)
    {
        $this->number  = $number;
        $this->type    = $type;
        $this->balance = $balance;
    }

    public function deposit(float $amount): float
    {
        $this->balance += $amount;
        return $this->balance;
    }

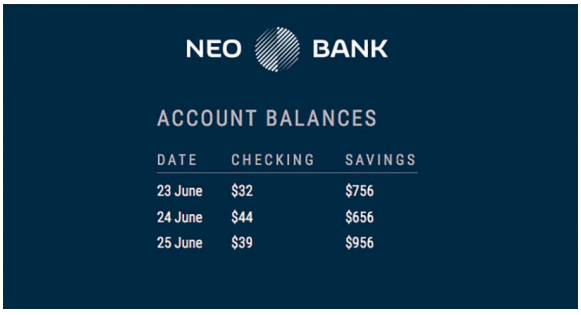
    public function withdraw(float $amount): float
    {
         $this->balance -= $amount;
         return $this->balance;
    }
}
```

PHP section_a/c04/constructor-methods.php

```php
$checking = new Account(43161176, 'Checking', 32.00);
$savings  = new Account(20148896, 'Savings', 756.00);
?>

<?php include 'includes/header.php'; ?>
<h2>Account Balances</h2>
<table>
  <tr>
    <th>Date</th>
    <th><?= $checking->type ?></th>
    <th><?= $savings->type  ?></th>
  </tr>
  <tr>
    <td>23 June</td>
    <td>$<?= $checking->balance ?></td>
    <td>$<?= $savings->balance  ?></td>
  </tr>
  <tr>
    <td>24 June</td>
    <td>$<?= $checking->deposit(12.00)  ?></td>
    <td>$<?= $savings->withdraw(100.00) ?></td>
  </tr>
  <tr>
    <td>25 June</td>
    <td>$<?= $checking->withdraw(5.00) ?></td>
    <td>$<?= $savings->deposit(300.00) ?></td>
  </tr>
</table>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



5. Создаются два объекта для представления текущего и сберегательного счетов.

Конструктор берет аргументы, переданные внутри круглых скобок, и присваивает свойствам объектов.

6. На странице отображается HTML-таблица. В первой строке выводятся заголовки, использующие свойство `type` обоих объектов. Для получения доступа к свойству используется:

- имя переменной, содержащей объект;
- оператор объекта;
- имя свойства.

7. В следующей строке таблицы выводится свойство `balance` объектов.

8. В третьей строке таблицы баланс каждого счета обновляется путем вызова методов `deposit()` или `withdraw()`.

Эти методы возвращают новое значение свойства `balance`, и оно выводится на страницу. Для вызова метода используется:

- имя переменной, содержащей объект;
- оператор объекта;
- имя метода с его аргументами в круглых скобках.

9. В четвертой строке таблицы повторяется предыдущий шаг с использованием других значений.

Упражнение. В шаге 6 создайте еще один объект, представляющий счет с повышенной процентной ставкой ($highInterest).

В шагах 7–9 добавьте строки, чтобы показать, что его баланс обновляется.

ОБЛАСТЬ ВИДИМОСТИ СВОЙСТВ И МЕТОДОВ

Вы можете запретить коду вне объекта получать доступ к значениям, содержащимся в его свойствах, а также вызывать методы этого объекта.

Свойства и методы класса называются **членами класса**. При создании класса вы можете указать, может ли код вне объекта, созданного с использованием этого класса:

- получать доступ к определенным свойствам этого объекта;
- вызывать определенные метод.

Это делается путем настройки **области видимости** при объявлении свойства или метода.

До сих пор в этой главе всем именам свойств и методов предшествовало слово `public`. Оно означает, что любой другой код может работать со свойствами и методами объекта.

Бывают случаи, когда вам требуется разрешить только коду внутри объекта читать или изменять значения свойств и вызывать методы этого объекта. Чтобы сделать это, замените слово `public` (общедоступный) на `protected` (защищенный).

Например, класс `Account` имеет свойство под названием `balance`. Если оно объявлено с использованием ключевого слова области видимости `public`, то любой код, работающий с объектом этого класса, может получить доступ к этому свойству.

Чтобы запретить любому другому коду читать или изменять значение, хранящееся в свойстве `balance`, область видимости этого свойства может быть изменена на `protected`.

Если вы попытаетесь получить доступ к свойству `protected` с помощью кода, который находится за пределами класса, то PHP выдаст сообщение об ошибке.

Когда коду вне объекта необходимо получить значение, содержащееся в свойстве, определенном как `protected`, необходимо добавить в класс метод, возвращающий его значение. Этот метод называют **геттер** (**getter**) (от английского get — получить).

В примере справа в класс `Account` добавлен новый метод с именем `getBalance()`. Его задача — вернуть значение, содержащееся в свойстве `$balance`.

Чтобы изменить значение, содержащееся в свойстве `protected`, добавьте в класс метод, обновляющий его значение. Он известен как **сеттер** (**setter**) (set — установить).

Методы `deposit()` и `withdraw()` в классе Account уже используются для изменения значения, содержащееся в свойстве `$balance`.

Эти изменения гарантируют, что `balance` обновляется только с помощью методов `deposit()` или `withdraw()`. Это свойство не может быть обновлено никаким другим кодом напрямую.

Если вы не укажете область видимости свойства или метода в определении класса, по умолчанию им будет установлена область видимости `public`. Но явное указание области видимости считается хорошим тоном и облегчает понимание вашего кода.

Существует еще один параметр области видимости, называемый `private` (закрытый). Он используется в более продвинутом объектно-ориентированном коде, что выходит за рамки книги для начинающих[16]. Эти настройки также называются **модификаторами доступа**.

ПРИМЕНЕНИЕ ГЕТТЕРОВ И СЕТТЕРОВ

PHP section_a/c04/getters-and-setters.php

```php
<?php
declare(strict_types = 1);

class Account {
    public    int    $number;
    public    string $type;
    protected float  $balance;

    public function __construct() {...}
    public function deposit() {...}
    public function withdraw() {...}

    public function getBalance(): float
    {
        return $this->balance;
    }
}

$account = new Account(20148896, 'Savings', 80.00);
?>

<?php include 'includes/header.php'; ?>
<h2><?= $account->type ?> Account</h2>
<p>Previous balance: $<?= $account->getBalance() ?></p>
<p>New balance: $<?= $account->deposit(35.00) ?></p>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



1. Свойство `balance` ранее было определено как общедоступное, а теперь изменено на защищенное, так что его не видно за пределами класса.
2. Существующие методы `deposit()` и `withdraw()` действуют как сеттеры для обновления баланса (код для них тот же, что и в предыдущем примере).
3. Геттер с именем `getBalance()` добавляется в класс, на случай, если понадобится получить значение защищенного свойства `balance`.
4. Создание объекта с использованием класса `Account`. Он присваивается переменной `$account`.
5. Свойство, содержащее тип счета, выводится напрямую, поскольку оно объявлено как `public`.
6. Метод `getBalance()` вызывается для отображения значения свойства `$balance`.
7. Вызывается метод `deposit()`. Он добавляет 35 долларов к свойству `$balance`. Этот метод также возвращает новый баланс, чтобы его можно было вывести на страницу.

Упражнение. После шага 6 используйте метод `withdraw()`, чтобы списать 50 долларов со счета.

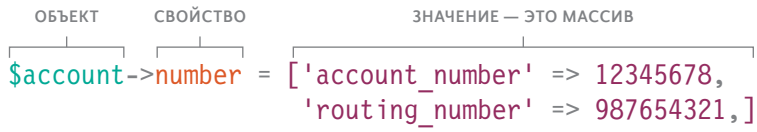
ХРАНЕНИЕ МАССИВА В СВОЙСТВЕ ОБЪЕКТА

Свойство объекта может хранить массив. Это позволяет получить доступ к отдельным элементам массива с помощью стандартного синтаксиса обращения к массивам.

Свойства объектов, которые вы видели до сих пор, содержали скалярные типы данных (строки, числа и логические значения). Свойство объекта также может содержать и составные типы данных, например массивы. В примере ниже задается свойство `number` объекта `$account`, относящегося к классу `Account`.

Присвоенное свойству `number` значение представляет собой ассоциативный массив, содержащий два отдельных значения:

- номер счета;
- номер банка (в некоторых странах известный как код сортировки, или BSB).

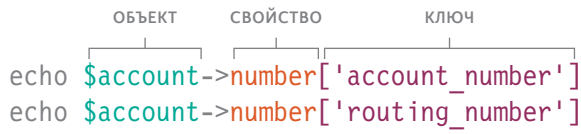



```
$account->number = ['account_number' => 12345678,
                    'routing_number' => 987654321,];
```

Чтобы получить доступ к элементу массива, сохраненного в свойстве `number`, используйте:

- имя переменной, содержащей объект;
- объектный оператор;
- имя свойства, содержащего массив;
- ключ элемента в массиве, к которому вы хотите получить доступ.

Ниже номер счета и номер банка, сохраненные в виде ассоциативного массива в свойстве `number` объекта `Account`, извлекаются с использованием их ключей и выводятся на страницу с помощью команды `echo`. Если бы массив был индексированным, ключом было бы значение индекса элемента, к которому вы хотите получить доступ.




```
echo $account->number['account_number'];
echo $account->number['routing_number'];
```

ИСПОЛЬЗОВАНИЕ МАССИВА В СВОЙСТВЕ ОБЪЕКТА

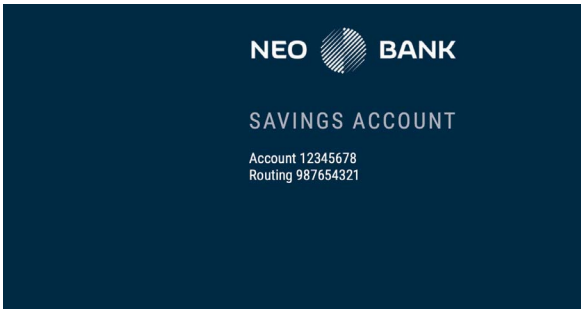
PHP section_a/c04/array-in-object.php

```php
<?php
declare(strict_types = 1);

class Account {...}
// Свойство number представляет собой массив,
// а не целое число
//Создаем массив для хранения в свойстве
$numbers = ['account_number' => 12345678,
            'routing_number' => 987654321,];

//Создаем экземпляр класса и задаем свойства
$account = new Account($numbers, 'Savings', 10.00);
?>
<?php include 'includes/header.php'; ?>
<h2><?= $account->type ?> account</h2>
Account <?= $account->number['account_number'] ?><br>
Routing <?= $account->number['routing_number'] ?>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



В этом примере создается объект для представления счета. Номера счета и банка будут сохранены в свойстве `$number`.

1. Класс `Account` такой же, как был в предыдущем примере, но объявление типа аргумента для параметра `$number` в методе `__construct()` указывает, что его значение будет массивом.

2. Объявляется переменная с именем `$number`. Она содержит ассоциативный массив с двумя ключами:

- `account_number`;
- `routing_number`.

3. Объект создается с использованием класса `Account`. Первый аргумент — это переменная `$numbers`, созданная в шаге 2, таким образом этот массив присваивается свойству `$number` объекта.

4. На странице выводится тип счета.

5. Отображение номера счета.

6. Отображение номера банка.

Упражнение. В шаге 2 изменить номер счета и номер банка.

ХРАНЕНИЕ ОБЪЕКТА В СВОЙСТВЕ ОБЪЕКТА

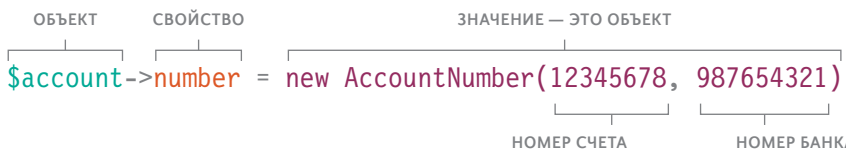
Свойство объекта может хранить другой объект. Можно читать или изменять отдельные свойства обоих объектов и вызвать их методы.

Вы видели, что свойство объекта может содержать массив, но точно так же оно может содержать и другой объект.

Ниже значение, присвоенное свойству `$number`, представляет собой новый объект, созданный с помощью класса `AccountNumber`.

Класс `AccountNumber` — это шаблон для объекта, представляющего номера счетов. Он обладает двумя свойствами:

- `$AccountNumber` содержит номер счета;
- `$routingNumber` содержит номер банка (в некоторых странах известный как код сортировки, или BSB).



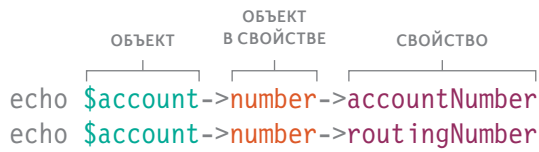
Чтобы получить доступ к свойству или методу объекта, содержащегося в свойстве `$number` другого объекта, используйте:

- имя переменной, содержащей объект класса `Account`;
- объектный оператор;
- название свойства, содержащего объект с номерами счетов;
- объектный оператор (для доступа к этому объекту);
- свойство или метод, который вы хотите использовать.

Ниже переменная `$account` содержит объект, представляющий банковский счет.

В его свойстве `$number` хранится второй объект, созданный с использованием класса `AccountNumber`.

Свойства `$AccountNumber` и `$routingNumber` этого второго объекта выводятся с помощью команды `echo`.



ИСПОЛЬЗОВАНИЕ ОБЪЕКТА В КАЧЕСТВЕ СВОЙСТВА ОБЪЕКТА

PHP section_a/c04/object-in-object.php

```php
<?php
declare(strict_types = 1);
class Account {...}
// Как ранее, но с типом данных свойства number
//будет имя класса AccountNumber

class AccountNumber
{
    public int $accountNumber;
    public int $routingNumber;

    public function __construct(int $accountNumber,
                                int $routingNumber)
    {
        $this->accountNumber = $accountNumber;
        $this->routingNumber = $routingNumber;
    }
}

$numbers = new AccountNumber(12345678, 987654321);
$account = new Account($numbers, 'Savings', 10.00);
?>
<?php include 'includes/header.php';?>
<h2><?= $account->type ?> Account</h2>
Account <?= $account->number->accountNumber ?><br>
Routing <?= $account->number->routingNumber ?>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



1. Класс `Account` такой же. Отличие: объявление типа аргумента для параметра `$number` в методе `__construct()` показывает, что значение для этого параметра должно быть объектом, созданным с использованием класса `AccountNumber`.
2. Добавляется определение для класса `AccountNumber`. У него есть два общедоступных свойства:

- `$accountNumber;`
- `$routingNumber.`

3. Значения свойствам присваиваются в конструкторе при создании объекта.
4. Создается объект с использованием класса `AccountNumber`. Он присваивается переменной `$numbers`.
5. Создается объект для представления счета с использованием класса `Account`. Первый аргумент — это переменная. Она содержит объект, представляющий номера счетов.
6. В заголовке страницы выводится тип счета.
7. Отображение номера счета.
8. Отображение номера банка.

ПРЕИМУЩЕСТВА ИСПОЛЬЗОВАНИЯ ОБЪЕКТОВ

Применение объектов помогает упорядочить ваш код, избавляет от повторения одного и того же кода на разных страницах, упрощает его обслуживание и понимание кода коллегами.

УЛУЧШЕННАЯ ОРГАНИЗАЦИЯ КОДА

Если на странице PHP содержатся сотни строк кода, может быть трудно понять, что делает каждая отдельная строка.

Объединение в одном классе переменных и функций, используемых для представления определенной концепции или сущности, такой как клиент или его счет, помогает сохранить весь связанный код в одном месте.

Увидев код, в котором создается какой-либо объект, программисты могут просмотреть определение его класса. Что они увидят:

- доступ к каким данным предоставляют его свойства;
- какие операции могут быть выполнены с использованием его методов.

В последнем примере этой главы на следующей странице показано, что определения классов часто хранятся в отдельных файлах (которые называются **файлами классов**). Это упрощает поиск кода для класса.

ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ КОДА

На сайте может быть несколько страниц, которые должны выводить одну и ту же информацию. Например, несколько страниц могут показывать данные клиента или пользователя сайта.

Вместо того чтобы на каждой странице повторять объявление переменных (для хранения данных, представляющих клиента) и определение функций (для представления операций, которые они могут выполнять), можно использовать готовое определение класса в качестве шаблона для создания объекта, содержащего информацию о клиенте и методы для работы с ней. Любая страница, на которой потребуется информация о клиенте, может подключить файл класса и создать объект, используя это определение класса в качестве шаблона.

Программисты иногда ссылаются на принцип «**Не повторяйся**», или **DRY** (**Don't Repeat Yourself**). Согласно этому принципу, при обнаружении повторов кода вам следует проверить, можно ли вместо повтора использовать функцию или метод объекта для выполнения задачи.

Программисты также ссылаются на **принцип единственной ответственности** (**SRP, Single responsibility principle**), который указывает, что каждая функция или метод должны отвечать только за что-то одно (а не выполнять несколько задач). Это помогает максимально эффективно повторно использовать код и облегчает его понимание.

ПРОСТОТА ОБСЛУЖИВАНИЯ

Тщательная организация и максимальное повторное использование кода помогают упростить его обслуживание.

Например.

- Если вам нужно сохранить некоторую дополнительную информацию о клиентах сайта, вы можете добавить свойство в определение класса, представляющее клиентов, и эта информация будет доступна для каждого объекта, представляющего клиента.
- Если вам нужно изменить способ выполнения программой определенной задачи (например, способ расчета процентов по счету), вам нужно только обновить код в одном классе, и он обновит каждый объект, созданный с использованием этого класса.

ПРОЩЕ ДЕЛИТЬСЯ КОДОМ

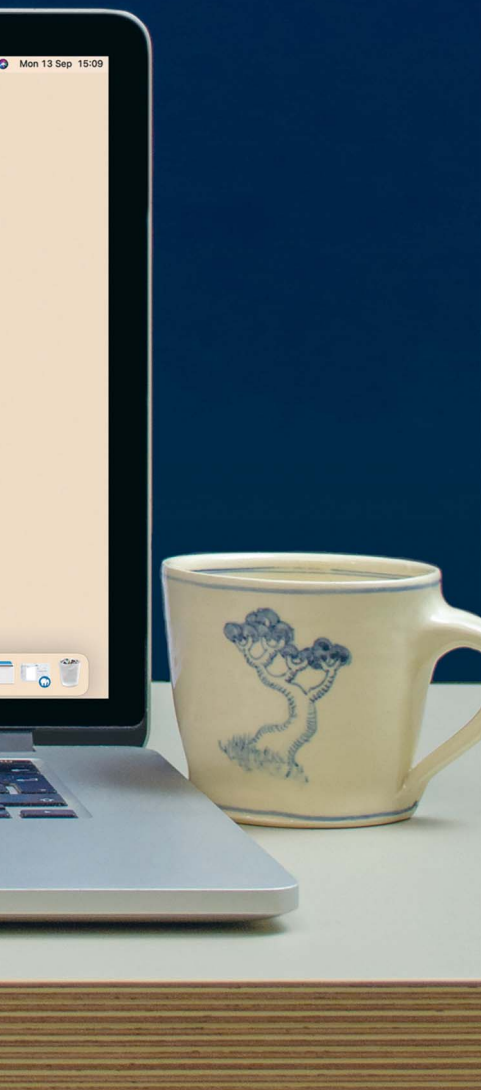
Если задуматься, то вам достаточно знать только имя класса, а также его свойства и методы, чтобы начать им пользоваться. При этом знать, как именно он выполняет свою работу, совершенно не обязательно. Вам только нужно знать:

- как создать объект, используя этот класс;
- какие данные можно получить из его свойств;
- какие задачи можно выполнять с помощью его методов.

Это помогает программистам, которые работают в команде, потому что разные программисты могут отвечать за разные определения классов.

В следующем разделе книги вы увидите, что интерпретатор PHP содержит множество встроенных функций и классов, которые помогают создавать веб-страницы. Вам не нужно знать, как именно они выполняют свои задачи. Вам достаточно знать, как их использовать.


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Macintosh HD
localhost:8888/phpbook/sectio
localhost:8888/phpbook/section_a/c04/example.php
NEO BANK
NAME: IVY STONE
ACCOUNT NUMBER
ACCOUNT TYPE
BALANCE
20489446
Checking
$ -20
20148896
Savings
$ 380



ИТОГОВЫЙ ПРИМЕР

В этом примере будет отображена информация о пользователе и банковские балансы для клиента с несколькими банковскими счетами.

Здесь реализованы два определения класса.

- Класс `Customer` используется для создания объекта, представляющего клиента банка.
- Класс `Account` — для создания объектов, представляющих различные счета, имеющиеся у каждого клиента.

Классы будут помещены в два отдельных файла с названиями `Customer.php` и `Account.php` соответственно, и эти файлы будут сохранены в папке `classes`.

На любой странице, которая использует классы для создания объекта, определения классов будут подключаться с помощью соответствующих конструкций PHP, точно так же, как хидер и футер подключаются к страницам в наших примерах.

На странице слева следующее.

- Объект, представляющий клиента, будет создан с использованием класса `Customer`.
- В него будет добавлено новое свойство под названием `$accounts`.
- Свойство `$accounts` будет содержать массив.
- Этот массив будет содержать два объекта класса `Account`, представляющие два счета клиента разных типов (созданных с использованием класса `Account`).

Это показывает, как вы можете создать иерархию объектов, где один объект содержит другой.

На странице отображается имя клиента и используется цикл `foreach` для того, чтобы перебрать все имеющиеся у клиента счета. Внутри цикла будут показаны номер счета, тип и баланс каждого счета.

С помощью условной конструкции будет сделана проверка, не допустил ли пользователь перерасход, и, если это так, баланс отобразится оранжевым, а не белым цветом.

КОД ПРИМЕРА

section_a/c04/classes/Account.php PHP

```php
<?php
class Account {...} // See p165
```

section_a/c04/classes/Customer.php PHP

```php
<?php
class Customer
{
    public  string $forename;
    public  string $surname;
    public  string $email;
    private string $password;
    public  array  $accounts;

    function __construct(string $forename, string $surname, string $email,
                         string $password, array $accounts)
    {
        $this->forename = $forename;
        $this->surname  = $surname;
        $this->email    = $email;
        $this->password = $password;
        $this->accounts = $accounts;
    }
    function getFullName()
    {
        return $this->forename . ' ' . $this->surname;
    }
}
```

Эти определения классов записываются в файлы `Account.php` и `Customer.php`, которые хранятся в папке `classes`. Таким образом, с помощью инструкции PHP `include` эти классы могут быть включены в любую страницу, на которой понадобится создать объект такого типа (см. шаг 5).

1. Класс `Account` был создан.

2. Класс `Customer` создан на основе класса. Новое свойство `$accounts` содержит массив объектов. Каждый объект представляет один из счетов клиента.

3. Свойство `$accounts` добавлено в конструктор.

4. Новый метод, который возвращает полное имя клиента.

5. На странице, которая отображают счета пользователей, подключаются определения классов `Account` и `Customer`, необходимые для создания этих объектов.

6. Создается индексированный массив и присваивается переменной с именем `$accounts`. Он содержит два объекта, созданных на основе класса `Account`. Каждый объект представляет собой один из банковских счетов клиента.

PHP section_a/c04/example.php

```php
<?php
include 'classes/Account.php';
include 'classes/Customer.php';

$accounts = [new Account(20489446, 'Checking', -20),
             new Account(20148896, 'Savings', 380),];

$customer = new Customer('Ivy', 'Stone', 'ivy@eg.link', 'Jup!t3r2684', $accounts);
?>
<?php include 'includes/header.php'; ?>
<h2>Name: <b><?= $customer->getFullName() ?></b></h2>

<table>
  <tr>
    <th>Account Number</th>
    <th>Account Type</th>
    <th>Balance</th>
  </tr>

  <?php foreach ($customer->accounts as $account) { ?>
    <tr>
      <td><?= $account->number ?></td>
      <td><?= $account->type ?></td>
      <?php if ($account->getBalance() >= 0) { ?>
        <td class="credit">
      <?php } else { ?>
        <td class="overdrawn">
      <?php } ?>
      $ <?= $account->getBalance() ?></td>
    </tr>
  <?php } ?>

</table>
<?php include 'includes/footer.php'; ?>
```

7. Создается новый объект `Customer`, представляющий клиента, и сохраняется в переменной с именем `$customer`. Последний аргумент — это массив счетов, созданных в шаге 7.

- Новый метод объекта Customer `getFullName()` возвращает полное имя клиента, которое выводится в заголовке.
- Цикл `foreach` проходит по массиву, лежащему в свойстве `$accounts` объекта `Customer`. В цикле каждый счет по очереди присваивается переменной, называемой `$account`.
- Номер и тип счета выводятся на странице.
- С помощью конструкции `if` проверяется, равен баланс 0 или больше.
- Если условие верно, то создается элемент `<td>` с CSS классом `credit`.
- Если нет, то элемент `<td>` создается с классом `overdrawn`.
- Вывод баланса.

Упражнение. В шаге 7 добавить в массив третий счет.

ЗАКЛЮЧЕНИЕ
КЛАССЫ И ОБЪЕКТЫ

- Объекты группируют переменные и функции, которые представляют какое-то явление или предмет окружающего нас мира.
- В объекте переменные называются свойствами, а функции называются методами.
- Класс используется в качестве шаблона для создания объектов.
- Определение класса задает свойства и методы, которыми будет обладать каждый объект, созданный на основе этого класса.
- Метод `__construct()` выполняется при создании объекта. Его можно использовать для присвоения значений свойствам объекта.
- `$this` позволяет обращаться в методах объекта к любым свойствам или методам этого объекта.
- Свойства и методы могут быть объявлены либо как общедоступные (`public`), к которым можно будет обращаться из кода вне объекта, либо как защищенные (`protected`), которые могут использоваться только кодом внутри объекта.
- Классы и объекты позволяют более эффективно структурировать код, организовывать его повторное использование и поддержку, а также помогают использовать код совместно с другими разработчиками.

Б

ДИНАМИЧЕСКИЕ ВЕБ-СТРАНИЦЫ

В этом разделе будет показано, как использовать PHP для создания динамических веб-страниц. Это страницы, на которых содержимое, представляемое пользователям, может изменяться, даже если программист вручную не изменял файл.

В разделе А вы ознакомились с синтаксисом языка PHP. В нем было показано:

- как переменные и массивы хранят данные;
- как операторы создают новое значение из нескольких фрагментов данных;
- как условия и циклы определяют, в каком порядке выполняется код;
- как функции и классы группируют связанные операторы.

В этом разделе вы узнаете, как применять эти знания для создания динамических веб-сайтов. Компьютер — это машина, позволяющая запускать программы, которые могут выполнять следующие базовые действия:

- принимать данные, известные как **входящие**;
- **обрабатывать** эти данные и выполнять задачи с ними;
- генерировать **исходящие** данные, которые пользователь может видеть или слышать;
- при необходимости **сохранять** данные для последующего использования.

PHP-скрипты, которые вы научитесь писать в этом разделе, следуют тем же базовым принципам. Они могут принимать входящие данные из веб-браузера, обрабатывать эти данные, а затем использовать их для вывода в браузер HTML-кода, адаптированного под конкретного посетителя. Вы узнаете:

- как использовать функции и классы, являющиеся частью PHP;
- как принимать и обрабатывать отправленные из браузера данные;
- как работать с изображениями и другими файлами, которые пользователи могут загружать;
- как сохранять данные о посетителях веб-сайта, используя файлы cookie и сессии;
- как устранять ошибки и неполадки в своем коде.

Перед тем как приступать к изучению данного раздела, вам нужно сначала получить общее представление о том, как интерпретатор PHP обрабатывает запросы и как он на них отвечает.

Для обработки запросов серверы следуют правилам, описанным в определенных протоколах, а также должны учитывать кодировку данных.

ЗАПРОСЫ И ОТВЕТЫ HTTP

Протокол передачи гипертекста (**HTTP**, **HyperText Transfer Protocol**) — это набор правил, регулирующих взаимодействие браузера с сервером. Вот почему URL-адреса веб-сайтов начинаются либо с «http://», либо с «https://». HTTP определяет:

- какие данные браузер должен отправить на сервер при запросе страницы;
- какие данные сервер должен отправить браузеру в ответ на этот запрос.

КОДИРОВКИ

Компьютеры представляют текст, изображения и аудио с использованием двоичных данных, состоящих из серий 0 и 1.

Системы кодирования (кодировки) — это используемые компьютерами правила для преобразования того, что вы видите и слышите, в 0 и 1, которые обрабатывает компьютер. Если вы не узнаете, как указать PHP, чтобы он использовал определенную кодировку, то он может неправильно обрабатывать или отображать данные.

Интерпретатор PHP содержит некоторое количество инструментов, которые помогают создавать динамические веб-страницы.

МАССИВЫ, ФУНКЦИИ, КЛАССЫ

Интерпретатор PHP содержит, в частности, такие инструменты.

- **Суперглобальные массивы**. Это массивы, создаваемые при каждом запросе к серверу.
- **Встроенные функции**. Они выполняют стандартные действия, необходимые программистам для решения типовых задач.
- **Встроенные классы**. Они необходимы для создания объектов, предоставляющих методы и свойства, позволяющие работать с определенным явлением или сущностью.

СООБЩЕНИЯ ОБ ОШИБКАХ

При возникновении проблем интерпретатор PHP создает **сообщения об ошибках**. Внимательное изучение этих сообщений поможет вам устранить проблемы в вашем коде.

НАСТРОЙКИ

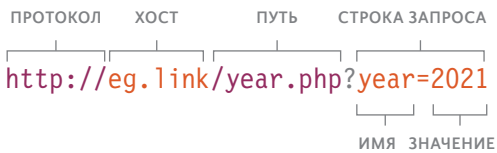
Как и многие другие программы, интерпретатор PHP и веб-сервер поддерживают набор доступных для программиста настроек. Вы узнаете, как изменять настройки для этих двух видов программ с помощью конфигурационных файлов.

ЗАПРОСЫ И ОТВЕТЫ HTTP

Протокол передачи гипертекста (HTTP) — это набор правил, определяющих, как браузер должен отправлять **запрос** для получения определенной страницы и как сервер должен формировать свой **ответ**. Изучение HTTP поможет вам понять, какие данные отправляются на каждом шаге.

Когда браузер **запрашивает** страницу PHP, в адресной строке браузера отображается URL-адрес, который содержит инструкции для браузера, как ему запрашивать эту страницу. Каждый URL содержит:

- **протокол** (для веб-страниц это HTTP или HTTPS);
- **хост** (сервер, на который отправляется запрос);
- **путь**, идентифицирующий запрошенный файл;
- необязательная **строка запроса** с дополнительными данными, которые могут понадобиться странице.



Когда в конец URL-адреса добавляется строка запроса, то это выглядит как передача пар «переменная — значение». Она содержит:

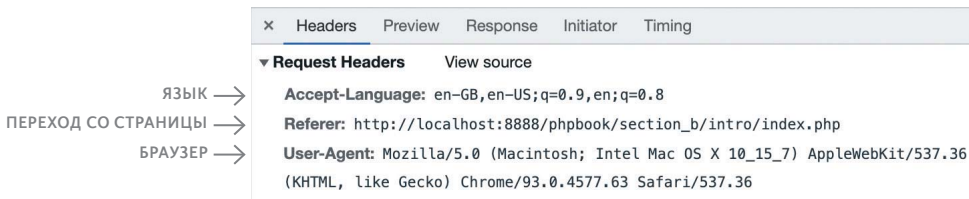
- **имя**, описывающее отправляемые данные. Оно не меняется при каждом использовании адреса;
- **значение** для этого фрагмента данных. Значение может меняться каждый раз при запросе страницы.

Когда браузер запрашивает веб-страницу, он также отправляет на сервер **заголовки HTTP-запроса**. Они не отображаются в главном окне браузера (как, например, URL-адрес), но их можно просмотреть в инструментах разработчика[17], предоставляемых большинством браузеров (см. снимок экрана ниже). Заголовки содержат данные, которые сервер может счесть полезными, и они также похожи на передачу переменных. У них есть:

- имя, описывающее, какие данные отправляются. Оно не меняется при каждом использовании URL-адреса;
- значение, содержащее передаваемые данные.

Заголовки на скриншоте ниже показывают:

- язык, на котором говорит посетитель (американский английский). На многоязычных сайтах это может быть использовано для выбора правильного языка для посетителя;



- информацию о том, что запрос пришел с другой веб-страницы вместе с URL-адресом этой страницы;
- информацию о браузере — Chrome на устройстве Mac, работает под управлением OSX. Это может быть использовано для определения, следует ли отправлять посетителя на полную либо мобильную версию сайта.

Когда веб-сервер получает запрос на страницу PHP, он **отвечает** на этот запрос следующим образом.

- Ищет запрошенный в URL-адресе PHP-файл.
- Запускает интерпретатор PHP для обработки содержащегося в файле PHP-кода.
- Отправляет результирующий HTML в браузер, который запросил эту страницу.

Когда сервер отправляет HTML (или другой тип контента) в браузер, он также передает **заголовки HTTP-ответа**. Они сообщают браузеру информацию о содержимом ответа. Как и заголовки запроса, каждый заголовок ответа содержит имя и значение (что делает это похожим на отправку переменных). Их можно просмотреть в инструментах разработчика в браузере. На скриншоте ниже сервер отправляет заголовки HTTP-ответов, сообщающих браузеру:

- тип контента и используемую кодировку (для правильного отображения ответа);
- дата и время отправки ответа;
- тип веб-сервера, используемого для отправки файла.

В PHP заголовки ответов могут быть отправлены с помощью:

- настроек интерпретатора PHP;
- встроенной функции под названием `header()`.

Когда браузер получает HTML-контент, он отображается так же, как и любая HTML-страница.

Каждый ответ также содержит строку состояния, которая показывает, был ли запрос успешным. Она состоит из двух элементов:

- трехзначный **код состояния** для интерпретации программой;
- **поясняющая фраза** — ее могут прочитать люди.

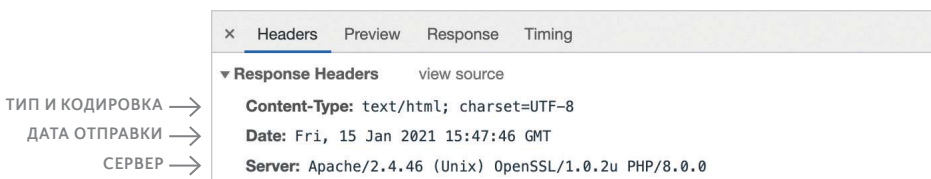
Для успешных запросов код состояния равен 200, а поясняющая фраза — OK. Если сервер не может найти файл, код состояния равен 404, а поясняющая фраза — Not found. При просмотре различных веб-сайтов вы наверняка встречали подобное сообщение, указывающее, что страница не найдена.

Not Found

The requested URL /code/section_b/c5/test.php was not found on this server.

В таблице ниже приведены наиболее распространенные коды состояния и поясняющие фразы. Такие коды, как 301 (перемещена навсегда) и 404 (не найдена), помогают поисковым системам индексировать сайты, когда они находят ссылки на удаленные или перенесенные на новый URL страницы.

КОД СОСТОЯНИЯ	ФРАЗА ПРИЧИНЫ
200	OK (В порядке)
301	Moved permanently (Перемещена навсегда)
307	Temporary redirect (Временное перенаправление)
403	Forbidden (Запрещена)
404	Not found (Не найдена)
500	Internal server error (Внутренняя ошибка сервера)



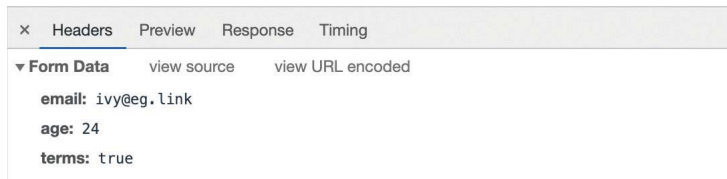
КАК ДАННЫЕ ОТПРАВЛЯЮТСЯ С ПОМОЩЬЮ HTTP GET И POST

HTTP определяет два метода[18], которыми браузеры могут отправлять данные на сервер.
HTTP GET содержит только заголовки. Данные при этом могут передаваться только в строке запроса.
HTTP POST помимо строки запроса может передавать данные после заголовков HTTP (эти данные еще называют телом запроса).

При отправке данных на сервер методом **HTTP GET** браузер может поместить данные в строку запроса и добавить ее в конец URL-адреса страницы[19]. Вопросительный знак отделяет URL-адрес страницы от строки запроса, которая может содержать несколько пар «имя — значение». Знак равенства отделяет каждое имя от его значения, а амперсанд (&) отделяет элементы строки запроса (пары «имя — значение») друг от друга.



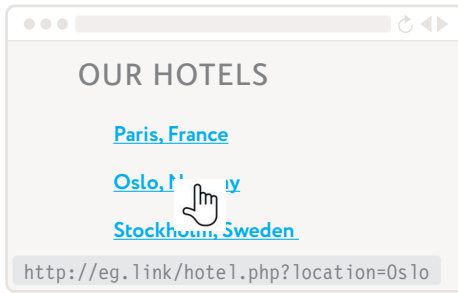
При отправке данных методом **HTTP POST** браузер добавляет дополнительные пары «имя — значение» после заголовков HTTP-запроса. Браузер может отправлять на сервер несколько пар «имя — значение» с каждым запросом. Заголовки и тело запроса не отображаются в главном окне браузера, но вы можете увидеть их в инструментах разработчика — их предоставляет большинство браузеров. Ниже вы можете увидеть три заголовка и соответствующие им значения.



КАК ОТПРАВЛЯТЬ ДАННЫЕ С ПОМОЩЬЮ ССЫЛОК И ФОРМ

В HTML используются ссылки и формы для отправки данных на сервер одновременно с запросом страницы.

В ссылке для передачи данных может использоваться строка запроса. Такой способ обычно используется, чтобы получить определенную информацию и отобразить ее на возвращаемой странице.

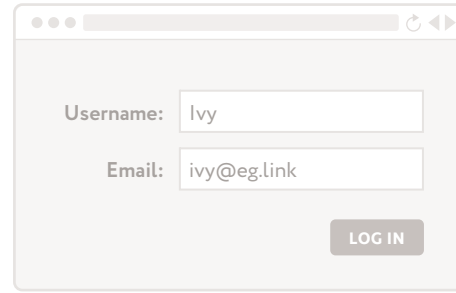


HTTP GET обычно используется, когда браузер хочет получить информацию с сервера, и эта информация будет одинаковой для каждого посетителя сайта. Например, когда посетители:

- нажимают ссылки, чтобы отобразить конкретную информацию;
- вводят поисковый запрос в форму.

Программисты иногда называют этот тип запроса **безопасным взаимодействием**, поскольку такие действия не накладывают на пользователя никаких обязательств (например, он не соглашается с условиями использования или не покупает продукт). Кроме того, запросы методом GET не изменяют состояние сервера (не меняют сохраненную на нем информацию).

У форм есть поля ввода, которые позволяют пользователям вносить текст или цифры, выбирать один из вариантов из списка или устанавливать флажок. Введенные таким образом данные должны быть либо добавлены в строку запроса, либо отправлены в теле запроса POST.



HTTP POST обычно используется, когда пользователь отправляет на сервер информацию, которая либо идентифицирует его, либо используется для обновления данных, хранящихся на сервере. Например, когда посетители:

- входят в свой личный кабинет;
- приобретают продукт;
- подписываются на услугу;
- соглашаются с правилами и условиями.

Часто такие действия влекут за собой какие-либо последствия для пользователя, которые он должен четко осознавать, поскольку заполнение и отправка формы могут привести к тому, что ему на почту будет приходить рассылка или с банковского счета спишутся деньги.

ЗАЩИТА ДАННЫХ, ОТПРАВЛЯЕМЫХ НА СЕРВЕР ИЛИ С СЕРВЕРА

При передаче между браузером и сервером конфиденциальные данные должны быть зашифрованы. **Шифрование** кодирует данные так, что их невозможно прочитать. **Расшифровка** снова преобразует данные в читаемый формат.

Когда данные передаются через интернет, они могут перемещаться по различным сетям, проходя через множество маршрутизаторов и серверов, чтобы достичь места назначения. Во время этого путешествия посторонние лица могут получить доступ к данным и попытаться их прочитать.

Любой веб-сайт, собирающий информацию о своих пользователях или отображающий их персональные данные на странице, несет ответственность за обеспечение безопасной передачи данных между браузером и сервером.

Для безопасной передачи данных между браузером и сервером веб-сайты используют **безопасный протокол передачи гипертекста** (**HTTPS**, **HyperText Transfer Protocol Secure**). HTTPS добавляет дополнительные правила к HTTP. Они определяют, как данные должны безопасно перемещаться между браузером и сервером.

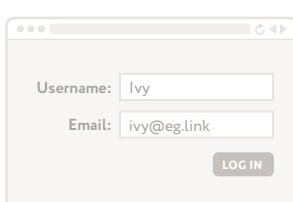
Чтобы безопасно отправлять данные через интернет, их зашифровывают. Это делается для того, чтобы, даже если они будут перехвачены во время перемещения, никто не смог их прочитать.

Сообщения шифруются путем замены исходных символов набором других символов. Это делается с помощью набора правил, известных как **шифр**.

Затем получателю сообщения необходимо расшифровать сообщение, чтобы снова сделать его читаемым. Чтобы расшифровать сообщение, получатель должен знать, как было зашифровано сообщение.

Данные, необходимые для расшифровки сообщения, называются **ключом**, поскольку они «отмыкают» сообщение.

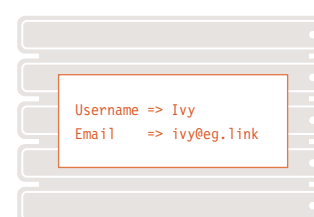
1. Когда пользователь отправляет форму, браузер шифрует данные.



2. Во время передачи зашифрованные данные невозможно прочесть.

```
NKFAyGCNYKdbNCDTA+XIwR698oP
pAdN1ghyUmRPtkE8y2evzf8LEMe
rOQ89N6XJN2AFt9l9bAr+qk/qSv
C6b/dRAbb6NqIYXqc6sOIZta/VZ
lUwJTUJHOIo6Qj68+paMgZX/6wX
XOf2VWLxxBM7XwU7ufVZ53VLQA+
mz/wA4jbAFevz8y2f8dbNCBW2wA
```

3. Сервер использует ключ для расшифровки данных.



Для шифрования и дешифрования данных, передаваемых между браузерами и серверами по протоколу HTTPS, на веб-сервере должен быть установлен **сертификат**. Он сообщает браузеру, как зашифровать отправляемую на этот сервер информацию.

Чтобы получить сертификат для установки на веб-сервере, необходимо выполнить следующие три шага.

1. Создайте **запрос на выпуск сертификата** (**CSR, Certificate signing request**). Он генерируется веб-сервером, на котором находится сайт. Запрос выглядит как набор случайных символов.

2. Приобретите сертификат у компании, называемой **центром сертификации** (**CA, Certificate authority**). Она запрашивает CSR, а также информацию о веб-сайте и его владельце. Плата CA за сертификат взимается за год. Список популярных центров сертификации приведен здесь http://notes.re/certificate-authorities/.

3. Установите сертификат (представляющий собой текстовый файл) на сервере, на котором работает сайт.

Примечание. Сертификаты не всегда выдаются немедленно, поэтому их необходимо получить до запуска сайта в эксплуатацию.

При локальной разработке сайта с использованием MAMP или XAMPP вы можете настроить веб-сервер для работы по протоколу HTTPS без покупки сертификата.

Ознакомиться с инструкциями о том, как это сделать, можно здесь: **http://notes.re/local-certificates/**.

Для получения CSR и установки сертификата на серверах хостинговой компании обратитесь к информации их службы поддержки.

После установки сертификата на веб-сервере, если браузер запрашивает URL-адрес сайта с использованием https://, а не http://, то:

- браузер шифрует запрос и заголовки HTTP-запросов;
- сервер шифрует возвращаемую страницу и заголовки HTTP-ответов.

Когда используется формат https://, браузеры обычно отображают значок замка в адресной строке.

Исторически сложилось так, что HTTPS использовал два разных протокола (набора правил) для добавления шифрования к запросам и ответам, отправляемым с использованием HTTP:

- **уровень защищенных сокетов** (**SSL**, **Secure Sockets Layer**);
- **безопасность транспортного уровня** (**TLS**, **Transport Layer Security**).

Часто люди используют термины SSL и TLS как взаимозаменяемые, но технически они разные.

Для простоты можно считать, что TLS это более поздняя версия SSL. TLS — это протокол, который вы должны использовать на своем сайте.

КОДИРОВКИ

Компьютеры выводят текст, изображения и аудио, используя **двоичные данные**, состоящие из последовательности нулей и единиц. Разные **кодировки** используются, чтобы преобразовать то, что вы видите или слышите, в 0 и 1, обрабатываемые компьютером.

Понимание роли схем кодирования важно, поскольку, если компьютер использует неправильную кодировку для преобразования между двоичными данными и текстом, изображениями и аудио, которые вы видите и слышите, данные не будут отображаться (воспроизводиться) правильно.

Все данные, которые компьютер обрабатывает и хранит, представлены с помощью **битов** (**двоичных цифр**). Бит — это 0 или 1. Итак, все на вашем компьютере (буквы, которые вы вводите, изображения, которые вы видите, звук, который вы слышите) представлено в 0 и 1. Ниже вы можете увидеть двоичный эквивалент каждой буквы в слове HELLO:

H	E	L	L	O
01001000	01000101	00101100	00101100	01001111

Это наглядно показывает, что даже для простых данных требуется много битов. Последовательность из 8 бит называется **байтом**. Схемы кодирования, или кодировки, — это правила, используемые компьютером для преобразования текста, изображений и аудио, а также двоичных данных (комбинаций 0 и 1), которые компьютер обрабатывает и сохраняет.

- Когда вы вводите текст, загружаете изображения или записываете аудио, схема кодирования используется для преобразования этого содержимого в 0 и 1.
- Когда компьютер показывает вам текст и изображения или воспроизводит аудиофайл, он использует определенную кодировку для преобразования 0 и 1 в то, что вы можете видеть или слышать.

Схемы кодирования изображений определяют, как представлять изображения с использованием битов. Компьютерные изображения состоят из точек, называемых пикселями. Ниже показано, как простой черно-белый значок сердечка может быть представлен с помощью 0 и 1. Каждый белый пиксель представлен цифрой 0, а каждый черный — цифрой 1.

0	1	1	1	0	1	1	1	0
1	0	0	0	1	0	0	0	1
1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	0

Чтобы воссоздать цветные изображения, компьютер должен знать, какого цвета каждый пиксель, для чего требуется больше данных. Различные форматы изображений (GIF, JPEG, PNG и WebP) используют разные схемы кодирования для представления цвета каждого пикселя с использованием 0 и 1. Чтобы управлять изображением, компьютер может изменять хранящиеся для каждого пикселя данные. Например, графический фильтр может использоваться для затемнения или осветления каждого пикселя изображения или изображение может быть обрезано путем удаления пикселей с краев.

Кодировки определяют, как представлять текст с использованием битов, и некоторые кодировки поддерживают больше символов, чем

другие. Чем больше символов поддерживает схема кодирования, тем больше байт данных ей требуется для обработки этих символов.
Когда вы создаете веб-сайт, рассчитанный на международную аудиторию, вы должны использовать кодировку, включающую символы из языков посетителей вашего сайта.

ТАБЛИЦА ASCII

ASCII — это одна из первых схем кодирования символов, использующая 7 бит данных для представления каждого символа.
Один из недостатков ASCII состоит в том, что существует только 128 возможных комбинаций 0 и 1 с использованием 7 двоичных цифр. Этого недостаточно для представления всех символов, используемых в каждом языке. Тем более что на самом деле ASCII поддерживает только 95 текстовых символов, а остальное отдано под знаки препинания и служебные символы.

СТАНДАРТ ISO 8859-1

Стандарт ISO 8859-1 использует 8 бит (1 байт) данных для представления каждого символа. Дополнительный бит данных означает, что существует достаточное количество комбинаций 0 и 1 для представления тех же символов, что и ASCII, а также символов, используемых в западноевропейских языках. Но он не поддерживает языки, использующие свои собственные наборы символов, такие как китайский, японский или русский.

СТАНДАРТ UTF-8

Стандарт UTF-8 содержит все символы из любого языка, это лучшая кодировка, которую можно использовать при создании веб-сайтов.
Для поддержки всех языков UTF-8 требуется до четырех байт данных для представления каждого символа (четыре набора из 8 комбинаций 0 и 1). Символы, представленные с использованием более чем одного байта данных, называются **многобайтовыми**. Например, ниже показаны двоичные эквиваленты для трех различных символов валюты:

СИМВОЛ	БИНАРНЫЙ	БАЙТОВ
$	00100100	1
£	11000010 10100011	2
€	11100010 10000010 10101100	3

Важно понимать, как работают кодировки, потому что:

- их необходимо указывать во множестве мест;
- они влияют на то, какие символы вы можете использовать;
- они определяют, какие встроенные функции следует использовать.

ВСТРОЕННЫЕ ФУНКЦИИ

Некоторые встроенные функции интерпретатора PHP поддерживают параметр, позволяющий указать используемую кодировку. Существуют также некоторые встроенные функции, работающие с определенной кодировкой. Например, в PHP есть функция, подсчитывающая количество символов в строке. Она была добавлена в PHP, когда кодировкой по умолчанию был стандарт ISO 8859-1, и работала путем подсчета количества байтов, используемых строкой (поскольку каждый символ использовал 1 байт данных). Когда PHP начал поддерживать UTF-8, эта функция начала давать неточные результаты для некоторых строк, поскольку каждый символ мог использовать более одного байта. Поэтому была добавлена новая встроенная функция с возможностью подсчета многобайтных символов.

НАСТРОЙКИ PHP

Отправляя страницу в браузер, интерпретатор PHP должен сообщать ему используемую кодировку, чтобы браузер мог правильно отобразить данные. В интерпретаторе PHP есть настройки, задающие используемую кодировку. Помимо прочего, они передают в браузер информацию о кодировке. Если кодировка указана неправильно, то браузер может отображать символ ◈ вместо некоторых символов или не отображать их вовсе.

РЕДАКТОРЫ КОДА

Поскольку сами файлы PHP являются текстовыми, редактору кода обычно можно указать кодировку, которую он должен использовать для сохранения файлов PHP.
По следующей ссылке показано, как настроить используемую кодировку в некоторых популярных редакторах кода: http://notes.re/editors/set-encoding.
В общем случае, если вы видите вариант выбора с надписью UTF-8 without BOM, вам следует выбрать его.
В разделе B вы увидите, что для базы данных также необходимо указывать кодировку, используемую веб-сайтом.

ВСТРОЕННЫЙ ИНСТРУМЕНТАРИЙ ИНТЕРПРЕТАТОРА PHP

В этом разделе книги вы познакомитесь с инструментами, встроенными в интерпретатор PHP. Они помогут вам создавать динамические веб-страницы.

Вы уже узнали, что интерпретатор PHP — это программа, работающая на веб-сервере.
Но она отличается от обычных программ, которые вы запускаете на настольном или портативном компьютере (например, текстовый процессор или редактор изображений). При запуске такой программы появляется графический пользовательский интерфейс (GUI), и для управления такой программой используются панели инструментов и пункты меню в графическом интерфейсе, а результаты отображаются на экране.
У интерпретатора PHP нет графического интерфейса, вместо него — набор встроенных массивов, функций и классов, которые мы можете использовать в своих скриптах для обработки данных и создания HTML-кода для отправки в браузер.
Кроме того, интерпретатор PHP может считывать различные настройки и параметры из специальных конфигурационных файлов, а также записывать любые возникающие ошибки в так называемые лог-файлы (от английского log — журнал).

СУПЕРГЛОБАЛЬНЫЕ МАССИВЫ

При каждом запросе браузером страницы PHP интерпретатор создает набор массивов, называемых суперглобальными. Они содержат различные данные, которые PHP-код может использовать на странице.
Все суперглобальные массивы ассоциативные, поэтому вам нужно знать:

- как называются массивы;
- какие ключи есть в каждом из массивов;
- что хранит каждый из этих ключей.

После того как интерпретатор PHP отправил результат своей работы в браузер, данные в этих массивах забываются, поскольку они, как и все остальные переменные в PHP, существуют только в рамках одного конкретного запроса.
При следующем запуске PHP-файла для него заново будет создан набор суперглобальных массивов с данными, относящимися уже к новому запросу.

ВСТРОЕННЫЕ ФУНКЦИИ

Встроенные функции можно сравнить с командами, находящимися в меню программы с графическим интерфейсом. Например, одна функция находит символы в строке и заменяет их другими символами аналогично функции поиска и замены текстового процессора.
Вместо того чтобы использовать пункт меню в графическом интерфейсе, вы вызываете функцию в своем PHP-коде.
В главе 3 было показано, как создать определение функции и вызвать эту функцию. Встроенные функции вызываются таким же образом, но нет необходимости включать определение функции в код, потому что оно уже встроено в интерпретатор PHP.
Чтобы использовать встроенные функции, вам необходимо знать:

- имя функции;
- запрашиваемые ею параметры;
- возвращаемое значение (или то, что она отображает на странице).

ВСТРОЕННЫЕ КЛАССЫ

Встроенные классы используются для создания объектов, представляющих сущности и понятия, с которыми программистам часто приходится иметь дело. Например, класс DateTime используется для создания объектов, представляющих дату и время. У класса есть свойства и методы, позволяющие вам работать с датой и временем, которые содержатся в созданном с помощью этого класса объекте.

В главе 4 было показано, как писать определения классов и использовать их для создания объектов. При создании объекта с использованием встроенного класса вам не потребуется добавлять определение этого класса на страницу, поскольку оно уже содержится в коде интерпретатора PHP. Чтобы использовать встроенные классы, вам нужно знать, как создать объект этого класса, а также:

- имеющиеся у него свойства;
- имеющиеся у него методы;
- параметры каждого метода;
- возвращаемое каждым методом значение.

СООБЩЕНИЯ ОБ ОШИБКАХ

Если интерпретатор PHP обнаружит проблему в запускаемом коде, он выдаст сообщение об ошибке.

Во время разработки сайта ошибки должны отображаться прямо в браузере. Это позволяет разработчикам сразу видеть любые ошибки, возникающие при попытке выполнить код.

Когда сайт начинает обслуживать посетителей, ошибки должны быть скрыты от пользователей. Разработчик сайта при этом не сможет видеть ошибки, поскольку они возникают время от времени (и невозможно заглядывать через плечо каждого пользователя). Вместо этого сообщения об ошибках сохраняются в текстовом файле, называемом **логом ошибок**. Этот файл хранится на сервере. Затем разработчик проверяет файлы логов, чтобы узнать, не были ли обнаружены какие-либо ошибки, пропущенные при разработке сайта (или возникшие из-за внешних факторов, например сбоя базы данных).

НАСТРОЙКИ

Программы для настольных компьютеров часто содержат пункт меню, который открывает окно с настройками, позволяющее пользователю изменять конфигурацию или параметры работы этой программы. Например, настройки текстового процессора могут позволить пользователю выбрать формат бумаги или язык по умолчанию для документа.

Поскольку у интерпретатора PHP и веб-сервера нет графического интерфейса, для управления настройками они используют специальные текстовые файлы, называемые конфигурационными. Например, существуют настройки, определяющие, должен ли интерпретатор PHP отображать сообщения об ошибках на экране или сохранять их в файл логов, а также где файл логов ошибок должен располагаться на сервере. Эти конфигурационные файлы можно редактировать с помощью того же редактора кода, который вы используете для создания PHP-скриптов.

СУПЕРГЛОБАЛЬНЫЕ МАССИВЫ

Суперглобальный массив $_SERVER — это пример одного из суперглобальных массивов, создаваемых интерпретатором PHP при каждом запросе страницы. Каждый суперглобальный массив содержит данные, доступные для использования в коде страницы.

Все суперглобальные массивы ассоциативные.
Вам нужно знать имя массива, ключи, находящиеся в каждом массиве, и данные, содержащиеся в них.
Ниже вы можете увидеть, какую информацию содержит суперглобальный массив $_SERVER[20]:

- данные о браузере (эта информация отправляется в заголовках HTTP);
- тип HTTP-запроса (GET или POST);
- запрошенный URL;
- расположение файла на сервере.

К суперглобальным массивам обращаются так же, как и к любым другим ассоциативным массивам. Например, чтобы сохранить IP-адрес компьютера, с которого пришел запрос, в переменной с именем $ip, следует использовать следующий код:



КЛЮЧ	НАЗНАЧЕНИЕ
$_SERVER['REMOTE_ADDR']	IP-адрес компьютера, с которого пришел запрос
$_SERVER['HTTP_USER_AGENT']	Тип браузера, используемого для запроса страницы
$_SERVER['HTTP_REFERER']	Если посетитель перешел на PHP-скрипт с другой страницы, браузер может отправить URL этой страницы. Не все браузеры отправляют эти данные
$_SERVER['REQUEST_METHOD']	Тип HTTP-запроса: GET или POST
$_SERVER['HTTPS']	Добавляется в массив только в том случае, если доступ к странице осуществляется по протоколу HTTPS. В этом случае будет иметь значение true
$_SERVER['HTTP_HOST']	Имя хоста (может быть именем домена, IP-адресом или localhost)
$_SERVER['REQUEST_URI']	URI (часть URL *после* имени хоста), который был использован для запроса этой страницы
$_SERVER['QUERY_STRING']	Строка запроса с данными (часть URI после знака вопроса)
$_SERVER['SCRIPT_NAME']	Путь от корневой папки сайта к выполняемому в данный момент файлу
$_SERVER['SCRIPT_FILENAME']	Путь от корневой папки файловой системы к выполняемому в данный момент файлу
$_SERVER['DOCUMENT_ROOT']	Путь от корневой папки файловой системы до корневой папки сайта

Разница между корневой папкой сайта (корнем сайта) и корневой папкой файловой системы описана здесь: **http://notes.re/php/filepaths**.

ДАННЫЕ В СУПЕРГЛОБАЛЬНОМ МАССИВЕ $_SERVER

Каждый элемент суперглобального массива $_SERVER содержит какую-либо информацию о запросе, запрашиваемом файле или окружении веб-сервера. Ниже отдельные значения, хранящиеся в суперглобальном массиве $_SERVER, выводятся путем обращения к различным ключам массива. Выполняя этот пример на своем компьютере, обратите внимание на различие в выводимых данных.

PHP section_b/intro/server-superglobal.php

```
<table>
  <tr><th colspan="2" class="title">Data About Browser Sent in HTTP Headers   </th></tr>
  <tr><th>Browser's IP address     </th><td><?= $_SERVER['REMOTE_ADDR'] ?>    </td></tr>
  <tr><th>Type of browser          </th><td><?= $_SERVER['HTTP_USER_AGENT'] ?></td></tr>
  <tr><th colspan="2" class="title">HTTP Request                             </th></tr>
  <tr><th>Host name                </th><td><?= $_SERVER['HTTP_HOST'] ?>      </td></tr>
  <tr><th>URI after host name      </th><td><?= $_SERVER['REQUEST_URI'] ?>    </td></tr>
  <tr><th>Query string             </th><td><?= $_SERVER['QUERY_STRING'] ?>   </td></tr>
  <tr><th>HTTP request method      </th><td><?= $_SERVER['REQUEST_METHOD'] ?> </td></tr>
  <tr><th colspan="2" class="title">Location of the File Being Executed      </th></tr>
  <tr><th>Document root            </th><td><?= $_SERVER['DOCUMENT_ROOT'] ?>  </td></tr>
  <tr><th>Path from document root  </th><td><?= $_SERVER['SCRIPT_NAME'] ?>    </td></tr>
  <tr><th>Absolute path            </th><td><?= $_SERVER['SCRIPT_FILENAME'] ?></td></tr>
</table>
```

РЕЗУЛЬТАТ

DATA ABOUT BROWSER SENT IN HTTP HEADERS	
BROWSER'S IP ADDRESS	::1
TYPE OF BROWSER	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.2 Safari/605.1.15
HTTP REQUEST	
HOST NAME	localhost:8888
URI AFTER HOST NAME	/phpbook/section_b/intro/server-superglobal.php
QUERY STRING	
HTTP REQUEST METHOD	GET
LOCATION OF THE FILE BEING EXECUTED	
DOCUMENT ROOT	/Users/Jon/Sites/localhost
PATH FROM DOCUMENT ROOT	/phpbook/section_b/intro/server-superglobal.php
ABSOLUTE PATH	/Users/Jon/Sites/localhost/phpbook/section_b/intro/server-superglobal.php

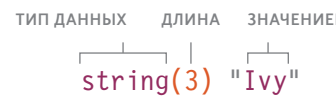
ВСТРОЕННЫЕ ФУНКЦИИ, ОТОБРАЖАЮЩИЕ ИНФОРМАЦИЮ О ПЕРЕМЕННЫХ

Функция `var_dump()` — это пример одной из встроенных функций PHP. Она используется во время разработки сайта для проверки, какое значение (или значения) содержит переменная и к какому типу данных оно относится.

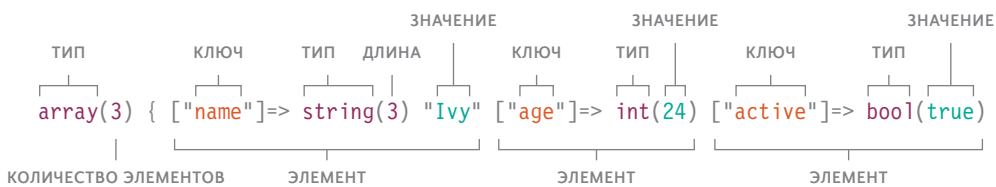
Чтобы использовать (вызвать) встроенную функцию, вам нужно знать ее имя, параметры и какие значения она вернет или отобразит на странице.
У функции `var_dump()` есть один параметр: имя переменной[21]. Она не возвращает значение — она сразу выводит содержимое переменной в браузер.

```
var_dump($variable);
```

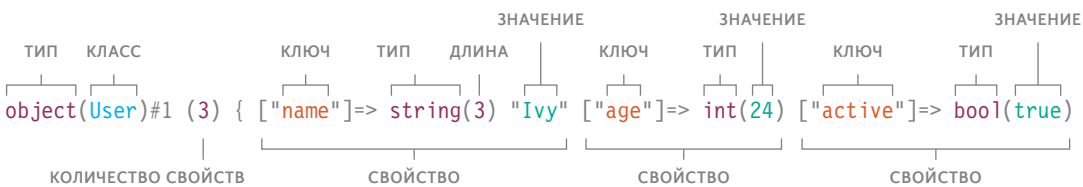
Если переменная содержит скалярное значение (строку, цифру или целое число), функция показывает тип данных и значение. Если значение представляет собой строку, то количество символов в строке указывается в круглых скобках после типа данных.



Если переменная содержит массив, то функция выводит слово `array` (массив) и количество элементов в нем внутри круглых скобок. Затем внутри фигурных скобок отображаются ключ, тип данных и значение для каждого элемента.



Если переменная содержит объект, функция выводит слово `object`, имя класса и количество свойств. Для каждого свойства выводится имя, тип данных и значение. Методы не отображаются.



ОТОБРАЖЕНИЕ СОДЕРЖИМОГО ПЕРЕМЕННОЙ

PHP section_b/intro/var-dump.php

```php
<?php
$username    = 'Ivy';

$user_array  = [
    'name'   => 'Ivy',
    'age'    => 24,
    'active' => true,
];

class User
{
    public $name;
    public $age;
    public $active;
    public function __construct($name, $age, $active) {
        $this->name   = $name;
        $this->age    = $age;
        $this->active = $active;
    }
}

$user_object = new User('Ivy', 24, true);
?>
...
<p>Scalar:   <?php var_dump($username); ?></p>
<p>Array:    <?php var_dump($user_array); ?></p>
<p>Object:   <?php var_dump($user_object); ?></p>
```

РЕЗУЛЬТАТ

```
Scalar: string(3) "Ivy"

Array: array(3) { ["name"]=> string(3) "Ivy" ["age"]=> int(24) ["active"]=> bool(true) }

Object: object(User)#1 (3) { ["name"]=> string(3) "Ivy" ["age"]=> int(24) ["active"]=> bool(true) }
```

В этом примере создаются переменные, содержащие скалярное значение, массив и объект, а затем используется функция `var_dump()` — она показывает, что содержит каждая переменная.

1. Переменной с именем `$username` присваивается строка с именем пользователя.
2. В переменной с именем `$user_array` содержится массив, в котором записана такая информация, как имя пользователя, возраст и информация об активации.
3. Класс под названием `User` создается в качестве шаблона для объектов, представляющих пользователей сайта. У него есть три свойства: имя пользователя, возраст и информация об активации.
4. Создается объект класса `User` и присваивается переменной с именем `$user_object`.
5. Значения переменных отображаются с помощью функции `var_dump()`.

Примечание. Если вы добавите теги HTML `<pre>` вокруг каждого блока PHP, информация выведется в отформатированном виде, что облегчит ее чтение.

```
<pre>
<?php var_dump($username)
?>
</pre>
```

СООБЩЕНИЯ ОБ ОШИБКАХ

Когда возникает проблема при выполнении программы, интерпретатор PHP генерирует сообщение об ошибке, помогающее вам исправить возникшие неполадки.

Если интерпретатор PHP сталкивается с проблемой при выполнении кода, он генерирует сообщение об ошибке. Существует два способа просмотреть эти сообщения. Они могут быть:

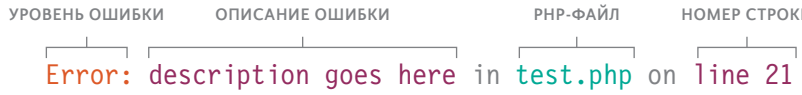
- отображены прямо в браузере;
- сохранены в текстовый файл, известный как **лог ошибок**.

Каждое сообщение об ошибке содержит четыре фрагмента данных, помогающих вам найти и устранить проблему:

- **уровень ошибки** (или серьезность ошибки; уровни описаны в таблице ниже);
- описание ошибки;
- файл, где произошла ошибка;
- номер строки, в которой была обнаружена ошибка.

При разработке сайта сообщения об ошибках должны отображаться прямо в браузере, чтобы разработчик мог их немедленно увидеть. Когда сайт «заступает на боевое дежурство», ошибки должны сохраняться в логе ошибок (текстовый файл на сервере), поскольку посетителям отображать системные ошибки PHP нельзя.

Код для этого раздела книги отображает ошибки прямо в браузере. Если вы попробуете выполнить упражнения, следующие за многими примерами в этом разделе, не расстраивайтесь встречающимся сообщениям об ошибках. Эти сообщения помогут вам узнать, где есть проблемы с вашим кодом и как их устранить. Вы изучите обработку ошибок более углубленно в главе 10.



УРОВЕНЬ	ОПИСАНИЕ
Парсинг (PARSE)	Ошибки в синтаксисе PHP-кода вообще не позволяют интерпретатору PHP выполнить код
Фатальные (FATAL)	Ошибка в коде PHP, останавливающая выполнение любого дальнейшего кода
Предупреждения (WARNING)	Ошибка в коде PHP, но интерпретатор попытается выполнить остальную часть кода
Уведомления (NOTICE)	Что-то, вероятно, указывает на проблему. Интерпретатор продолжит выполнять остальной код
Устаревания (DEPRECATED)	PHP-код, который, скорее всего, будет удален из будущих версий PHP
Строгие (STRICT)	Этот PHP-код можно было бы написать лучше, сделать его более строгим

ПРИМЕРЫ СООБЩЕНИЙ ОБ ОШИБКАХ

Сообщения об ошибках на первый взгляд могут показаться загадочными, но содержащаяся в них информация поможет вам выяснить, что не так с вашим кодом.

PHP section_b/intro/error1.php

```
<?php
① echo $name;
② echo ' welcome to our site.';
?>
```

РЕЗУЛЬТАТ

Warning: Undefined variable $name in **/Users/Jon/Sites/localhost/phpbook/section_b/intro/error1.php** on line **2**
welcome to our site.

PHP section_b/intro/error2.php

```
<?php
③ echo 'Hello ';
  username = 'Ivy';
?>
```

РЕЗУЛЬТАТ

Parse error: syntax error, unexpected token "=" in **/Users/Jon/Sites/localhost/phpbook/section_b/intro/error2.php** on line **3**

1. Страница пытается вывести несуществующую переменную. Это вызывает вывод предупреждения о том, что происходит обращение к неопределенной переменной, `Undefined variable $name` в файле `error1.php` на строке 2. Поскольку уровень ошибки — предупреждение, интерпретатор продолжает работать.

До версии PHP 7.4 эта ошибка создавала уведомление, а не предупреждение.

2. На страницу выводится текст `Welcome to our site` (его вы можете увидеть под сообщением об ошибке).

3. Конструкция `echo` *вывела бы* слово `Hello`, но следующая строка вызывает синтаксическую ошибку, не позволяющую интерпретатору PHP выполнить какой-либо код на странице.

Ошибка вызвана отсутствием символа $ в начале переменной `$username`.

Вы узнаете больше об устранении неполадок и сообщениях об ошибках в главе 10.

НАСТРОЙКИ ИНТЕРПРЕТАТОРА PHP

Управление настройками и предпочтениями в программах для настольных компьютеров обычно осуществляется с помощью меню в пользовательском интерфейсе. Управление настройками интерпретатора PHP и веб-сервера Apache производится с помощью текстовых файлов.

У веб-сервера Apache и интерпретатора PHP есть настройки, которые управляют такими параметрами, как используемая по умолчанию кодировка символов, отображение сообщений об ошибках пользователям или максимальный объем памяти, который разрешено использовать PHP-скрипту. Файлы, используемые для управления этими настройками, можно изменять в том же редакторе, что и код.

ФАЙЛ PHP.INI

Текстовый файл с названием `php.ini` управляет настройками интерпретатора PHP по умолчанию. Настройки в этом файле могут быть изменены, но их нельзя удалять. После внесения изменений в этот файл веб-сервер необходимо перезапустить, чтобы изменения вступили в силу.

Найти, где находится ваш файл `php.ini`, поможет встроенная функция PHP под названием `phpinfo()`. Она отображает настройки интерпретатора PHP в HTML-таблице (см. правую страницу). Путь к файлу `php.ini` находится в самом начале таблицы, в строке с заголовком **Loaded configuration file** (Загруженный файл конфигурации).

Некоторые хостинговые компании не предоставляют разработчикам доступ к файлу `php.ini`, поскольку он обычно управляет тем, как выполняются все файлы PHP на всем веб-сервере, и, следовательно, влияет на другие сайты на том же сервере. Если у вас нет доступа к `php.ini`, вы можете использовать файл `.htaccess` для управления многими из тех же настроек.

Если для одного PHP-файла требуются настройки, отличные от других, можно использовать встроенную функцию PHP, называемую `ini_set()`[22], чтобы переопределить некоторые настройки прямо в коде PHP.

ФАЙЛ HTTPD.CONF

Текстовый файл с названием `httpd.conf` управляет настройками по умолчанию для веб-сервера Apache. Некоторые настройки совпадают с настройками PHP, например кодировка страниц. После редактирования этого файла Apache необходимо перезапускать, чтобы изменения вступили в силу. Хосты не всегда предоставляют клиентам доступ к `httpd.conf`, поскольку он обычно управляет настройками всего веб-сервера.

ФАЙЛ .HTACCESS

Apache позволяет программисту добавлять файлы с именем `.htaccess` в любую папку в корневой папке сайта. Настройки в файле `.htaccess` будут применяться только к файлам в той же папке, что и файл `.htaccess`, и к ее дочерним папкам. Они будут переопределять настройки, указанные в файлах `httpd.conf` и `php.ini`.

Изменения в файлах `.htaccess` вступают в силу сразу после сохранения файла. Но использовать `.htaccess` стоит, только если вы не можете использовать `httpd.conf` или `php.ini`, поскольку использование файла `.htaccess` немного замедляет работу веб-сервера.

Большинство хостинговых компаний позволяют создавать файлы `.htaccess`, но они могут ограничивать доступные настройки (например, максимальный размер загружаемых файлов).

Операционные системы рассматривают файл `.htaccess` как скрытый, и поэтому вам может потребоваться указать проводнику файлов или FTP-программе, чтобы они отображали скрытые файлы. Чтобы узнать, как просматривать скрытые файлы, перейдите на http://notes.re/hidden_files.

В коде этой книги будет использоваться несколько разных файлов `.htaccess`, так что разные главы могут быть с разными настройками.

ПРОСМОТР НАСТРОЕК ИНТЕРПРЕТАТОРА PHP

Как и у большинства программ, у интерпретатора PHP есть настройки, управляющие его работой. Встроенная функция `phpinfo()` отображает несколько таблиц, в которых выводятся все конфигурационные параметры и их текущие значения.

PHP section_b/intro/phpinfo.php

```
<?php phpinfo(); ?>
```

РЕЗУЛЬТАТ

Core

PHP Version	8.0.0

Directive	Local Value	Master Value
allow_url_fopen	On	On
allow_url_include	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
auto_append_file	*no value*	*no value*
auto_globals_jit	On	On
auto_prepend_file	*no value*	*no value*
browscap	*no value*	*no value*
default_charset	UTF-8	UTF-8
default_mimetype	text/html	text/html
disable_classes	*no value*	*no value*
disable_functions	*no value*	*no value*

Функция `phpinfo()` выводит набор из нескольких таблиц, отображающих настройки интерпретатора PHP и их значения (текущие и по умолчанию), а также некоторую дополнительную информацию. Эти настройки влияют на каждый PHP-файл, запускаемый интерпретатором PHP.

Для изменения этих настроек можно использовать текстовые файлы, описанные на левой странице.

В таблице ниже представлены некоторые из настроек, которые вам понадобятся при изучении данного раздела. Все они находятся в таблице под заголовком *Core*, если не указано иное.

НАСТРОЙКА	ОПИСАНИЕ
`default_charset`	Кодировка символов по умолчанию. Это значение должно быть равно UTF-8
`display_errors`	Включение/выключение отображения ошибок на HTML-странице. Включено (On) при разработке. Выключено (Off) при запуске сайта в рабочем режиме
`log_errors`	Включение/выключение сохранения ошибок в файл логов. Включено (On) при запуске сайта в рабочем режиме
`error_log`	Путь к файлу логов, в который пишутся ошибки
`error_reporting`	Уровень ошибок, которые будет порождать PHP. E_ALL — порождать ошибки всех уровней
`upload_max_filesize`	Максимально допустимый размер одного файла для загрузки браузером на сервер
`max_execution_time`	Максимальное количество секунд, в течение которых может выполняться скрипт, прежде чем интерпретатор PHP остановит его
`date.timezone`	Часовой пояс по умолчанию, используемый сервером. Отображается под заголовком Date

ИЗМЕНЕНИЕ НАСТРОЕК ИНТЕРПРЕТАТОРА: php.ini

Файл `php.ini` позволяет вам редактировать настройки интерпретатора PHP. Вы должны только редактировать значения настроек, но не удаляйте ни одну из них[23].

Файл `php.ini` довольно большой, поскольку он содержит все настройки для интерпретатора PHP. В нем также много комментариев, объясняющих каждую настройку. Все, что стоит после точки с запятой, — комментарий.

Настройки управляются с помощью **директив**, которые подобны переменным. Редактируйте только значения директив (но не удаляйте сами директивы). Чтобы найти желаемый параметр, откройте файл в редакторе и выполните поиск этого параметра.

Полный список директив см. на **http://php.net/manual/ru/ini.list.php**.

Каждая директива начинается с новой строки. Она состоит из:

- имени параметра;
- оператора присвоения;
- значения, которое должно быть присвоено.

Когда значение:

- строковое, то поместите его в кавычки;
- числовое, то не используйте кавычки;
- логическое, то не используйте кавычки.

```
date.timezone   = "Europe/Rome"
display_errors = On
```

ПАРАМЕТР — ЗНАЧЕНИЕ

Пример php.ini (отсутствует в загружаемом примере) PHP

```
; Подборка настроек, которые могут быть изменены в файле PHP.ini с комментариями
default_charset     = "UTF-8"       ; Кодировка, используемая по умолчанию
display_errors      = On            ; Показывать ошибки на экране
log_errors          = On            ; Записывать ошибки в лог-файл
error_reporting     = E_ALL         ; Порождать ошибки всех уровней
upload_max_filesize = 32M           ; Максимально допустимый для загрузки размер файла
post_max_size       = 32M           ; Максимальный объем данных, передаваемых через HTTP POST
max_execution_time  = 30            ; Максимальное время выполнения каждого скрипта
memory_limit        = 128M          ; Максимальный объем памяти, потребляемый скриптом
date.timezone       = "Europe/Rome" ; Часовой пояс по умолчанию
```

ИЗМЕНЕНИЕ НАСТРОЕК СЕРВЕРА: .htaccess

Файл `.htaccess` может быть размещен в любой папке на веб-сервере. Это переопределит настройки интерпретатора PHP для файлов в этой папке, а также на все дочерние папки и содержащиеся в них файлы.

Этот файл служит для изменения настроек веб-сервера Apache, но также в некоторых случаях может использоваться для изменения настроек PHP[24], для чего применяются две специальные команды. В них указываются те же имена и значения, что и в файле `php.ini`, за исключением команды, которая пишется перед директивой:

- `php_flag` — если значение директивы логическое, т. е. представляет параметр, который может быть включен или выключен;
- `php_value` — для всех остальных значений.

Комментарии начинаются с символа # и должны отображаться в новой строке (не в той же строке, что и директива).

Добавляйте в `.htaccess` только те настройки, которые вы хотите переопределить, а не все подряд.

Приведенный ниже файл `.htaccess` включен в загружаемый код для этой книги. Это гарантирует правильное поведение примеров кода, в частности — что ошибки будут отображаться прямо в браузере.

Если вы не видите этот файл (или он выглядит серым), это связано с тем, что операционные системы рассматривают его как скрытый файл.

Чтобы узнать, как просматривать скрытые файлы, перейдите на http://notes.re/hidden_files.

```
php_value date.timezone  "Europe/London"
php_flag  display_errors On
```

ТИП — ПАРАМЕТР — ЗНАЧЕНИЕ

PHP

section_b/intro/.htaccess

```
# содержимое файла .htaccess, используемого в примерах кода
# (параметры описаны в примере файла PHP.ini)

php_value  default_charset     "UTF-8"
php_flag   display_errors      On
php_flag   log_errors          Off
php_value  error_reporting     -1
php_value  upload_max_filesize 32M
php_value  post_max_size       32M
php_value  max_execution_time  30
php_value  memory_limit        128M
php_value  date.timezone       "Europe/London"
```

В ЭТОМ РАЗДЕЛЕ
ДИНАМИЧЕСКИЕ ВЕБ-СТРАНИЦЫ

5 **ВСТРОЕННЫЕ ФУНКЦИИ**
Каждая из встроенных функций PHP выполняет определенную задачу, которую часто приходится решать программистам при работе с данными. Встроенные функции представлены первыми, поскольку они используются в каждой главе остальной части книги.

6 **ПОЛУЧЕНИЕ ДАННЫХ ИЗ БРАУЗЕРА**
В этой главе показано, как интерпретатор PHP может получить доступ к отправленным из браузера данным, проверить, что данные, необходимые для страницы, были отправлены, и убедиться, что они находятся в правильном формате. Вы также узнаете, как обеспечить безопасность при отображении данных на странице.

7 **ИЗОБРАЖЕНИЯ И ФАЙЛЫ**
Если вы разрешаете пользователям отправлять изображения или другие файлы на веб-сайт, вам нужно знать, как интерпретатор PHP обрабатывает эти файлы. В этой главе также показано, как выполнять такие задачи, как изменение размера изображений и создание уменьшенных версий изображений, известных как миниатюры.

8 **ДАТА И ВРЕМЯ**
Дата и время могут быть представлены во множестве разных форматов, поэтому вам нужно знать, как PHP может помочь вам последовательно форматировать даты и время. Вы также узнаете, как выполнять общие задачи, например, представление интервала времени и обработка повторяющихся событий.

9 **COOKIE И СЕССИИ**
В этой главе показано, как можно создавать cookie в браузере пользователя для хранения информации о посетителе. В ней также рассказывается, как можно использовать сессии для хранения информации на веб-сервере в течение короткого периода (например, при одном посещении сайта).

10 **ОБРАБОТКА ОШИБОК**
Каждый может ошибиться при написании кода. Кроме того, некоторые ошибки не зависят от программиста, а вызываются внешними факторами. В таких случаях интерпретатор PHP создает сообщения об ошибках. В этой главе показано, как читать такие сообщения, а также методы, помогающие находить и устранять ошибки в коде.

5

ВСТРОЕННЫЕ ФУНКЦИИ

В этой главе представлены некоторые функции, встроенные в интерпретатор PHP. Каждая функция выполняет определенную задачу.

Определения для встроенных функций встроены в интерпретатор PHP. Это означает, что их не нужно включать в страницу PHP перед вызовом. Они были разработаны для выполнения задач, которые часто приходится решать веб-разработчикам при создании динамических веб-страниц, и избавляют от необходимости писать собственные функции для выполнения таких задач.

Чтобы вызвать встроенную функцию, вам нужно знать ее имя, параметры, а также какие данные она возвращает. Поэтому в данной главе содержится несколько таблиц, в которых показаны имена и параметры функций, за которыми следуют их описания и возвращаемые значения.

Первый набор функций сгруппирован по типам данных, для работы с которыми они используются: строки, числа и массивы. Позже в этой главе вы также узнаете:

- как создавать константы (которые похожи на переменные, но их значения не могут измениться, после того как были заданы впервые);
- какая функция используется для управления HTTP-заголовками, отправляемыми в браузер, когда интерпретатор PHP возвращает запрошенную страницу;
- какие функции позволяют получать информацию о файлах на сервере.

Представленные в этой главе функции будут использоваться на протяжении всей остальной части этой книги.


Chapter 5 Examples
phpandmysql.com/code/section_b/ch05/array-sorting-functions.php
Search
MOUNTAIN ART SUPPLIES
Order
eraser, ink, notebook, pencil, scissors, washi tape
Classes
Knitting May 4th
Origami June 8th
Patchwork April 12th

ВЕРХНИЙ И НИЖНИЙ РЕГИСТРЫ, ПОЛУЧЕНИЕ ДЛИНЫ СТРОКИ

Эти функции преобразуют символы в тексте в верхний или нижний регистр, а также подсчитывают количество символов или слов в строке.

Следующие функции предназначены для работы с текстом (строковый тип данных, string). Они принимают в качестве аргумента строку, преобразуют ее и возвращают измененное значение.

Например, функция `strtolower()` принимает строку и преобразует весь текст в строчные буквы. Затем она возвращает обновленное значение.

ИЗМЕНЕНИЕ РЕГИСТРА СИМВОЛОВ

ФУНКЦИЯ	ОПИСАНИЕ
`strtolower($string)`	Возвращает строку со всеми символами в нижнем регистре
`strtoupper($string)`	Возвращает строку со всеми символами в верхнем регистре
`ucwords($string)`	Возвращает строку с первой буквой каждого слова в верхнем регистре

ПОДСЧЕТ КОЛИЧЕСТВА СИМВОЛОВ И СЛОВ

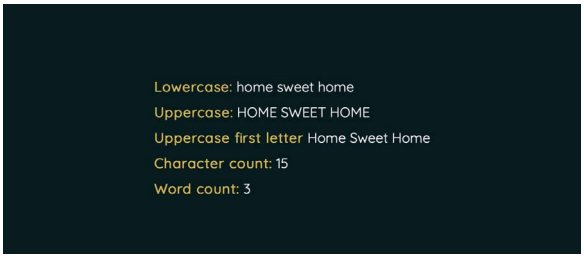
ФУНКЦИЯ	ОПИСАНИЕ
`strlen($string)`	Возвращает количество символов в строке Пробелы и знаки препинания считаются символами
`str_word_count($string)`	Возвращает количество слов в строке

ПРЕОБРАЗОВАНИЕ РЕГИСТРА И ПОДСЧЕТ СИМВОЛОВ

PHP section_b/c05/case-and-character-count.php

```
<?php
① $text = 'Home sweet home';
?>
<?php include 'includes/header.php'; ?>
<p>
  <b>Lowercase:</b>
② <?= strtolower($text) ?><br>
  <b>Uppercase:</b>
③ <?= strtoupper($text) ?><br>
  <b>Uppercase first letter:</b>
④ <?= ucwords($text) ?><br>
  <b>Character count:</b>
⑤ <?= strlen($text) ?><br>
  <b>Word count:</b>
⑥ <?= str_word_count($text) ?>
</p>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



1. Строка `Home sweet home` присваивается переменной с именем `$text`. Она будет использоваться в качестве аргумента при вызове каждой из функций.

2. Вызов функции `strtolower()`. Функция преобразует текст в нижний регистр и возвращает полученное значение. Оно выводится на странице с использованием сокращенной команды `echo`.

3. Функция `strtoupper()` возвращает строку в верхнем регистре.

4. Функция `ucwords()` возвращает строку с первой буквой каждого слова в верхнем регистре.

5. Функция `strlen()` подсчитывает количество символов в строке и возвращает это число.

6. Функция `str_word_count()` подсчитывает количество слов в строке и возвращает это число.

Упражнение. В шаге 1 измените строку на «`PHP и MySQL`», затем сохраните файл и обновите страницу.

ПОИСК СИМВОЛОВ В СТРОКЕ

Эти функции ищут один или несколько символов в строке. При обнаружении совпадения они возвращают позицию этого символа. Если совпадение не найдено, возвращается значение `false`[25].

Каждый символ в строке занимает определенную **позицию** — число, начинающееся с 0. Таким образом, первый символ находится в позиции 0, второй — в позиции 1 и так далее.

H	o	m	e		s	w	e	e	t		h	o	m	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Когда вы ищете набор символов внутри строки, эти символы называются **подстрокой** (**substring**).

Некоторые функции **чувствительны к регистру**, поэтому совпадение обнаруживается только в том случае, если строка и подстрока содержат одинаковую комбинацию символов верхнего и нижнего регистра.

ФУНКЦИЯ[26]	ОПИСАНИЕ
`strpos($string, $substring[, $offset])`	Возвращает позицию первого совпадения для подстроки (с учетом регистра). Если используется смещение (offset), то поиск начинается *после* указанной в смещении позиции
`stripos($string, $substring[, $offset])`	Версия strpos() без учета регистра
`strrpos($string, $substring[, $offset])`	Возвращает позицию последнего совпадения для подстроки (с учетом регистра)
`strripos($string, $substring[, $offset])`	Версия strrpos() без учета регистра
`strstr($string, $substring)`	Возвращает текст от первого совпадения подстроки (включая подстроку) до конца строки (с учетом регистра)
`stristr($string, $substring)`	Версия strstr() без учета регистра
`substr($string, $offset[, $characters])`	Возвращает символы от позиции, указанной в $offset, до конца строки. Если используется параметр $characters, он указывает количество символов, возвращаемых после $offset. Дополнительные параметры см. http://notes.re/php/substr
✲ `str_contains($string, $substring)`	Проверяет, найдена ли подстрока в строке, возвращает true/false
✲ `str_starts_with($string, $substring)`	Проверяет, начинается ли строка с подстроки, возвращает true/false
✲ `str_ends_with($string, $substring)`	Проверяет, заканчивается ли строка подстрокой, возвращает true/false

Последние три функции, отмеченные звездочкой, были добавлены в версии PHP 8. Все они чувствительны к регистру.

Примечание. Расширенные параметры указаны в квадратных скобках.

ПРОВЕРКА НАЛИЧИЯ СИМВОЛОВ В СТРОКЕ

PHP section_b/c05/finding-characters.php

```
<?php
① $text = 'Home sweet home';
?> ...
<b>First match (case-sensitive):</b>
② <?= strpos($text, 'ho') ?><br>
<b>First match (not case-sensitive):</b>
③ <?= stripos($text, 'me', 5) ?><br>
<b>Last match (case-sensitive):</b>
④ <?= strrpos($text, 'Ho') ?><br>
<b>Last match (not case-sensitive):</b>
⑤ <?= strripos($text, 'Ho') ?><br>
<b>Text after first match (case-sensitive):</b>
⑥ <?= strstr($text, 'ho') ?><br>
<b>Text after first match (not case-sensitive):</b>
⑦ <?= stristr($text, 'ho') ?><br>
<b>Text between two positions:</b>
⑧ <?= substr($text, 5, 5) ?><br>
```

РЕЗУЛЬТАТ

```
First match (case-sensitive): 11
First match (not case-sensitive): 13
Last match (case-sensitive): 0
Last match (not case-sensitive): 11
Text after first match (case-sensitive): home
Text after first match (not case-sensitive): Home sweet home
Text between two positions: sweet
```

Упражнение. В шаге 1 изменить строку на `Home and family`. Затем, в шаге 8, используйте `substr()`, чтобы вернуть слово `and`.

1. Строка `Home sweet home` присвоена переменной `$text`.
2. Функция `strpos()` вызывается для поиска позиции впервые встреченной в строке подстроки `ho`. Она возвращает значение 11.
3. Функция `stripos()` вызывается, чтобы найти позицию первого появления подстроки `me` после позиции 5. Она возвращает значение 13.
4. Функция `strrpos()` вызывается, чтобы найти позицию последнего появления подстроки `Ho`. Она возвращает 0, потому что функция чувствительна к регистру, а такая подстрока находится в самом начале строки.
5. Функция `strripos()` вызывается, чтобы найти позицию последнего появления подстроки `Ho`. Она возвращает 11, поскольку нечувствительна к регистру.
6. Функция `strstr()` вызывается для получения текста от первого появления подстроки `ho` до конца строки. Она возвращает значение `home`.
7. Функция `stristr()` вызывается для получения текста от первого появления `ho`. Она возвращает `Home sweet home`, потому что нечувствительна к регистру.
8. Вызывается функция `substr()`, которая возвращает пять символов начиная с символа в пятой позиции.

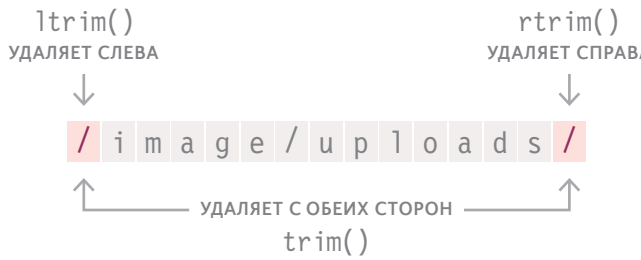
УДАЛЕНИЕ, ЗАМЕНА И ПОВТОРЕНИЕ СИМВОЛОВ

Эти функции могут удалять указанные символы (включая пробельные), заменять символы (как инструмент поиска и замены), а также повторять строку заданное количество раз.

Функции обрезки по краям (**trim**) удаляют символы в начале и конце строки. Они могут проверять наличие определенных символов в начале, конце или с обеих сторон строки и удалять их при наличии.

Если вы не указали символы для удаления, функции обрезки удаляют любые пробельные символы по краям строки, включая символы пробела, табуляции, вертикальной табуляции, возврата каретки, перевода строки и нулевого байта.

Функции замены (**replace**) ищут символы в строке. При обнаружении совпадения они заменяют эти символы другими. Функция повторения (**repeat**) повторяет строку фиксированное количество раз.



ФУНКЦИЯ	ОПИСАНИЕ
`ltrim($string[, $delete])`	Удаляет символы с левого края строки. Аргумент $delete, если определен, содержит набор символов, которые должны быть удалены, если обнаружены в начале строки. Иначе удаляются пробельные символы. Функция чувствительна к регистру
`rtrim($string[, $delete])`	Удаляет символы из правой части строки
`trim($string[, $delete])`	Удаляет символы с обеих сторон строки
`str_replace($old, $new, $string)`	Заменяет подстроку $old на подстроку из $new (с учетом регистра)
`str_ireplace($old, $new, $string)`	Заменяет подстроку $old на подстроку из $new (без учета регистра)
`str_repeat($string, $repeats)`	Повторяет строку заданное количество раз

ЗАМЕНА СИМВОЛОВ В СТРОКЕ

PHP section_b/c05/removing-and-replacing-characters.php

```
<?php
① $text = '/images/uploads/';
?> ...
<b>Remove '/' from both ends:</b><br>
② <?= trim($text, '/') ?><br>
<b>Remove '/' from the left of the string:</b><br>
③ <?= ltrim($text, '/') ?><br>
<b>Remove 's/' from the right of the string:</b><br>
④ <?= rtrim($text, 's/') ?><br>
<b>Replace 'images' with 'img':</b><br>
⑤ <?= str_replace('images', 'img', $text) ?><br>
<b>As above but case-insensitive:</b><br>
⑥ <?= str_ireplace('IMAGES', 'img', $text) ?><br>
<b>Repeat the string:</b><br>
⑦ <?= str_repeat($text, 2) ?></p>
```

РЕЗУЛЬТАТ

```
Remove '/' from both ends:
images/uploads
Remove '/' from the left of the string:
images/uploads/
Remove 's/' from the right of the string:
/images/upload
Replace 'images' with 'img':
/img/uploads/
As above but case-insensitive:
/img/uploads/
Repeat the string:
/images/uploads//images/uploads/
```

1. Путь `'/images/uploads/'` — это строка, содержащаяся в переменной с именем `$text`.
2. Функция `trim()` возвращает строку с удаленным символом `/` с левой и с правой сторон текста.
3. Функция `ltrim()` возвращает строку с удаленным символом `/` с левой стороны.
4. Функция `rtrim()` возвращает строку с удаленным символом `/` с правой стороны.
5. Функция `str_replace()` возвращает строку текстом `images`, замененным на текст `img`. Она чувствительна к регистру.
6. Функция `str_ireplace()` возвращает строку текстом `IMAGES`, замененным на текст `img`. Поскольку функция нечувствительна к регистру, она найдет как `IMAGES`, так и `images` и заменит их на `img`.
7. Функция `str_repeat()` возвращает строку, повторенную дважды.

Упражнение. В шаге 1 добавить пробел в самом начале и в самом конце строки, а затем обновить страницу. Прямые слеши `/` не будут удалены на этапах 2, 3 или 4, поскольку по краям строки теперь стоят пробелы, а не слеши.

МНОГОБАЙТОВЫЕ СТРОКОВЫЕ ФУНКЦИИ

Некоторые из строковых функций, которые встречались нам до сих пор, возвращают неправильный результат, если они используются с многобайтовыми символами. Приведенные ниже многобайтовые строковые функции поддерживают все символы в кодировке UTF-8.

Когда текст кодируется с использованием UTF-8, некоторые символы используют более одного байта данных. Например, символ £ использует два байта, а символ € — три.

Использование многобайтовых символов в качестве аргументов для некоторых строковых функций может привести к неправильному результату (примеры неточных результатов показаны на странице справа).

Представленные ниже многобайтовые строковые функции носят те же имена, что и описанные ранее в этой главе, но им предшествуют символы `mb_`. Для корректной работы этих функций должна быть указана кодировка в файлах `php.ini` или `.htaccess`. По умолчанию это UTF-8.

Для некоторых строковых функций нет многобайтового эквивалента. Например, для `str_replace()`, которая одинаково работает как с однобайтными кодировками, так и с UTF-8. Функция `trim()` также не имеет многобайтового эквивалента, и поэтому не следует использовать многобайтовые символы в ее втором параметре, поскольку каждый байт этого символа будет рассматриваться как отдельный символ для удаления.

ФУНКЦИЯ	ОПИСАНИЕ
`mb_strtoupper($string)`	Возвращает строку со всеми символами в верхнем регистре
`mb_strtolower($string)`	Возвращает строку со всеми символами в нижнем регистре
`mb_strlen($string)`	Возвращает количество символов в строке
`mb_strpos($string, $substring[, $offset])`	Возвращает позицию первого совпадения искомой подстроки (с учетом регистра). Если указать аргумент $offset, то поиск начинается с позиции, следующей за указанной в этом параметре
`mb_stripos($string, $substring[, $offset])`	Версия mb_strpos() без учета регистра
`mb_strrpos($string, $substring[, $offset])`	Возвращает позицию последнего совпадения для подстроки (с учетом регистра)
`mb_strripos($string, $substring[, $offset])`	Версия mb_strrpos() без учета регистра
`mb_strstr($string, $substring)`	Возвращает текст от первого совпадения подстроки (включая подстроку) до конца строки (с учетом регистра)
`mb_stristr($string, $substring)`	Версия mb_strstr() без учета регистра
`mb_substr($string, $start[, $characters])`	Возвращает символы от позиции, указанной в $start, до конца строки. Если указано значение $characters, функция возвращает количество символов после $start

ПРИМЕНЕНИЕ МНОГОБАЙТОВЫХ СТРОКОВЫХ ФУНКЦИЙ

PHP section_b/c05/multibyte-string-functions.php

```
<?php
① $text = 'Total: £444';
?> ...
<b>Character count using <code>strlen()</code>:</b>
② <?= strlen($text) ?><br>
<b>Character count using <code>mb_strlen()</code>:</b>
③ <?= mb_strlen($text) ?><br>
<b>First match of 444 <code>strpos()</code>:</b>
④ <?= strpos($text, '444') ?><br>
<b>First match of 444 <code>mb_strpos()</code>:</b>
⑤ <?= mb_strpos($text, '444') ?><br>
```

РЕЗУЛЬТАТ

```
Character count using strlen(): 12
Character count using mb_strlen(): 11
First match of 444 strpos(): 9
First match of 444 mb_strpos(): 8
```

В этом примере различные строковые функции используются для обработки строки в кодировке UTF-8, содержащей символ £, для кодирования которого требуется два байта данных.

1. Создается строка с использованием символа £ и присваивается переменной `$text`. Ее длина составляет 11 символов.
2. Функция `strlen()` работает путем подсчета количества байтов, используемых для представления строки, а не количества символов в ней. Вот почему она возвращает число 12, а не 11.
3. Функция `mb_strlen()` учитывает кодировку, которую использует интерпретатор PHP (UTF-8), и отображает правильное количество символов в строке как 11.
4. Функция `strpos()` находит позицию первого вхождения подстроки 444. Позиция того места, в котором найдена подстрока, вычисляется с использованием количества байтов, а не символов. Поэтому возвращается значение 9, а не 8.
5. Функция `mb_strpos()` находит позицию первого вхождения подстроки 444, возвращая корректное число 8.

Упражнение. В шаге 1 измените символ £ в строке на символ €.

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Такие наборы символов, как номера кредитных карт, почтовые индексы или телефонные номера, строятся по определенным шаблонам. Регулярные выражения описывают такие шаблоны. В PHP есть встроенные функции, которые проверяют наличие этих шаблонов в строке.

Регулярные выражения пишутся между парой одинаковых символов-ограничителей. Традиционно это прямой слеш. Шаблон в приведенном ниже выражении описывает, что:

`[A-z]` в шаблон входят буквы от a до z (в нижнем и верхнем регистре);
`{3,9}` которые встречаются от 3 до 9 раз;
`/[A-z]{3,9}/.`

Если использовать это выражение для проверки строки «Рисунок», то функция PHP для поиска совпадений с регулярным выражением найдет первое совпадение с шаблоном в приведенной строке, а именно последовательность букв A-z (строчные и прописные) длиной от 3 до 9 символов. Эта последовательность выделена ниже:

`Thomas was 1st!`

Синтаксис регулярных выражений может быть довольно сложным, и существуют целые книги о том, как их составлять. Но на этих страницах вы изучите только самые основы. В таблице ниже показаны примеры регулярных выражений, которые ищут первое совпадение с конкретной подстрокой, различными диапазонами символов, а также с символами, расположенными в начале или конце строки.

ВЫРАЖЕНИЕ	ОПИСАНИЕ	ПРИМЕР
`/1st/`	Буквальное соответствие символам **1st**	Thomas was 1st!
`/[abcde]/`	Если символы заключены в квадратные скобки, реализуется поиск совпадения с любой из букв. В данном случае требуется совпадение с a, b, c, d или e	Thomas was 1st!
`/[K-Z]/`	Дефис в квадратных скобках создает диапазон символов для сопоставления. Поиск соответствия с любым символом в верхнем регистре между K и Z	Thomas was 1st!
`/[a-e]/`	Поиск соответствия с любым символом в нижнем регистре между a и e	Thomas was 1st!
`/[0-9]/`	Поиск соответствия с любой цифрой между 0 и 9	Thomas was 1st!
`/[A-z0-9]/`	Поиск соответствия с любой заглавной или строчной буквой A-z или цифрой от 0 до 9	Thomas was 1st!
`/^[A-Z]/`	Символ каретки ^ в начале шаблона указывает, что строка должна начинаться с этих символов. В данном случае первый символ должен входить в диапазон A–Z	Thomas was 1st!
`/1st\!$/`	Знак доллара $ в конце шаблона указывает, что строка должна заканчиваться указанными символами. В данном случае совпадение будет обнаружено, если символы **1st!** находятся в самом конце строки	Thomas was 1st!
`/\s/`	Любой пробельный символ	Thomas was 1st!

Следующие символы имеют особое значение в регулярных выражениях: `\ / . | $ ( ) ^ ? { } + *`.

Чтобы добавить в шаблон один из этих символов, добавьте перед ним **обратный слеш**.

ВЫРАЖЕНИЕ	ОПИСАНИЕ	ПРИМЕР
`/[\!\?\(\)]/`	Поиск соответствия восклицательному знаку, вопросительному знаку или скобкам	Thomas was 1st!

Вы можете добавить **квантор**, чтобы указать, сколько раз шаблон должен появляться в строке.

В приведенных ниже примерах производится поиск символов, которые встречаются указанное количество раз.

ВЫРАЖЕНИЕ	ОПИСАНИЕ	ПРИМЕР
`/[a-z]+/`	Знак плюс «+» означает любое количество больше нуля	Thomas was 1st!
`/[a-z]{3}/`	Число в фигурных скобках {} указывает точное количество раз, когда совпадение с шаблоном должно быть обнаружено	Thomas was 1st!
`/[A-z]{3,5}/`	Два числа, разделенные запятой в фигурных скобках {}, указывают минимальное и максимальное количество символов, которые должны совпасть с шаблоном	Thomas was 1st!
`/[a-z]{3,}/`	Число, затем запятая (без второго числа) в фигурных скобках — это минимальное количество раз, когда совпадение должно быть обнаружено	Thomas was 1st!

Чтобы найти последовательность шаблонов, укажите их один за другим. Ниже за совпадением для первого шаблона должно следовать совпадение для второго шаблона.

ВЫРАЖЕНИЕ	ОПИСАНИЕ	ПРИМЕР
`/[0-9][a-z]/`	Соответствует числу 0-9, за которым следует строчная буква a–z	Thomas was 1st!

Размещение круглых скобок вокруг части выражения создает **группу**. Можно добавить квантор после группы, чтобы указать, сколько раз она должна повторяться.

Если необходимо найти совпадение с одной из нескольких строк, вы можете указать их внутри группы, разделив вертикальной чертой.

ВЫРАЖЕНИЕ	ОПИСАНИЕ	ПРИМЕР
`/[0-9]([a-z]{2})/`	[0-9] соответствует любому числу 0-9; за ним следует ([a–z]{2}), которое соответствует двум строчным буквам	Thomas was 1st!
`/[1-31](st|nd|rd|th)/`	[1-31] соответствует любому числу 1-31, за которым следует (st|nd|rd|th), что соответствует st, nd, rd или th	Thomas was 1st!

ФУНКЦИИ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Эти функции проверяют, содержит ли строка набор символов, описываемых регулярным выражением. Каждая функция выполняет собственное действие, если совпадение будет найдено.

Все приведенные ниже функции используют регулярные выражения для поиска заданного шаблона в строке[27].

Они выполняют такие задачи:

- проверка, было ли обнаружено совпадение с шаблоном;
- подсчет того, сколько раз был найден шаблон;
- поиск подстроки, совпадающей с шаблоном, и замена ее другой (похоже на функцию поиска и замены в текстовом процессоре).

В каждом случае у функции есть параметры:

- регулярное выражение, описывающее набор искомых символов (оно заключено в кавычки, потому что это строка);
- строка, в которой оно ищет этот шаблон.

Функция, заменяющая совпавшие с шаблоном символы новыми, также должна знать, чем их заменить.

ФУНКЦИЯ	ОПИСАНИЕ
`preg_match($regex, $string)`	Ищет соответствующий шаблон в строке. Возвращает 1, если совпадение было найдено; 0, если совпадение не обнаружено; false, если произошла ошибка
`preg_match_all($regex, $string)`	Ищет соответствующий шаблон в строке. Возвращает количество найденных совпадений (0, если не найдено ни одного) или false, если произошла ошибка
`preg_split($regex, $string)`	Разбивает строку на части, используя совпадающую с шаблоном подстроку как разделитель, а затем возвращает все части в виде индексированного массива
`preg_replace($regex, $replace, $string)`	Заменяет найденную по шаблону подстроку другой строкой. Действует аналогично инструменту поиска и замены в текстовом редакторе: возвращает строку с замененными символами, если нашлось совпадение, или исходную строку — если нет. Чтобы удалить символы, замените их пустой строкой

Примечание. Имена этих функций начинаются с префикса `preg` (что означает регулярные выражения Perl — **P**erl **reg**ular expressions), потому что они следуют синтаксису регулярных выражений в другом языке программирования, называемом Perl.

ПРИМЕНЕНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

PHP section_b/c05/regular-expression-functions.php

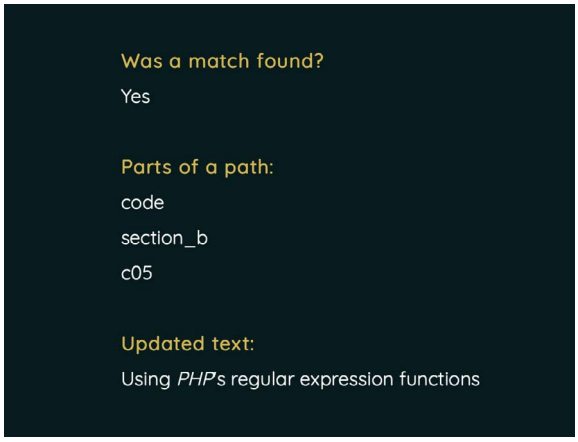
```
<?php
(1) $text = 'Using PHP\'s regular expression functions';
(2) $path = 'code/section_b/c05/';

(3) $match = preg_match('/PHP/', $text);
(4) $path  = preg_split('/\//', $path);
(5) $text  = preg_replace('/PHP/', '<em>PHP</em>', $text);
?> ...
<b>Was a match found?</b><br>
(6) <?= ($match === 1) ? 'Yes' : 'No' ?><br><br>

<b>Parts of a path:</b><br>
<?php foreach($path as $part) { ?>
(7)   <?= $part ?><br>
<?php } ?>

<b>Updated text:</b><br>
(8) <?= $text ?>
```

РЕЗУЛЬТАТ



1. Текст присваивается переменной с именем `$text`.
2. В переменную с именем `$path` записывается путь к файлу.
3. Функция `preg_match()` проверяет, найдена ли строка «PHP» в тексте, сохраненном в переменной с именем `$text` в шаге 1. Если совпадение найдено, переменная `$match` получает значение 1.
4. Функция `preg_split()` разбивает на части путь, который был сохранен в переменной `$path` в шаге 2, используя символ слеша как разделитель. Все части строки, которые встречаются между слешами, помещаются в итоговый массив с именем `$path`, который перезаписывает исходную переменную `$path`.
5. Функция `preg_replace()` ищет буквы «PHP» в тексте, который был сохранен в переменной `$text` на шаге 1. Если совпадение с шаблоном найдено, то оно заменяется теми же буквами внутри HTML-элемента `<em>`.
6. С помощью тернарного оператора проверяется, содержит ли переменная `$match` значение 1. Если содержит, то на странице будет написано слово `Yes`, если нет, отображается слово `No`.
7. С помощью цикла каждый элемент массива `$path` отображается в новой строке.
8. Отображается измененный текст переменной `$text`, с символами PHP в тегах `<em>`.

РАБОТА С ЧИСЛАМИ

В дополнение к математическим операторам, с которыми вы познакомились в главе 1, существуют функции для типовых действий, которые программистам приходится выполнять с числами.

ФУНКЦИЯ	ОПИСАНИЕ
`round($number, $places, $round)`	Округляет числа с плавающей точкой в большую или меньшую сторону: *$number* — это число, которое можно округлить в большую или меньшую сторону. *$places* — это количество знаков после запятой, до которых округляется число. *$round* указывает, как округлять числа в большую или меньшую сторону со следующими параметрами:
	ПАРАМЕТР — **НАЗНАЧЕНИЕ** `PHP_ROUND_HALF_UP` — Округляет половинные значения в большую сторону (например, 3,5 становится 4) `PHP_ROUND_HALF_DOWN` — Округляет половинные значения в меньшую сторону (например, 3,5 становится 3) `PHP_ROUND_HALF_EVEN` — Округляет половинные значения до ближайшего четного числа `PHP_ROUND_HALF_ODD` — Округляет половинные значения до ближайшего нечетного числа
`ceil($number)`	Округляет число до ближайшего большего целого числа
`floor($number)`	Округляет число до ближайшего меньшего целого числа
`mt_rand($min, $max)`	Создает случайное число в диапазоне от *$min* до *$max*
`rand($min, $max)`	Начиная с PHP 7.1, rand() совпадает с mt_rand(). До этого функция rand()использовала алгоритм, который был менее случайным и медленным
`pow($base, $exponent)`	Возвращает основание $base, возведенное в степень $exponent (например, 3^4 вернет 81)
`sqrt($number)`	Возвращает квадратный корень из числа
`is_numeric($number)`	Проверяет, является ли значение числом (целым или с плавающей точкой). Возвращает значение true, если это число, и значение false, если нет
`number_format($number [, $decimals] [, $decimal_point] [, $thousand_separator])`	Указывает, как должно быть отформатировано число. Если задано только $number, оно форматируется без десятичных знаков, а для разделения тысяч используется запятая. $decimals задает количество знаков после запятой. $decimal_point и $thousand_separator позволяют указать символы, используемые для разделения десятичных знаков и тысяч. По умолчанию используется точка для разделения десятичных дробей и запятая для разделения тысяч. Если требуется задать $thousand_separator, то придется также указать и $decimal_point

ФУНКЦИИ ДЛЯ РАБОТЫ С ЧИСЛАМИ

1. Округление чисел разными способами.
2. Генерируется случайное число от 0 до 10.
3. Показано значение 4 в степени 5.
4. Показан квадратный корень из 16.

5. Проверка, является ли значение числом (`int` или `float`). Возвращается `true`, если условие соблюдено (при выводе на страницу отображается как 1), и `false` — если нет (ничего не показывает).

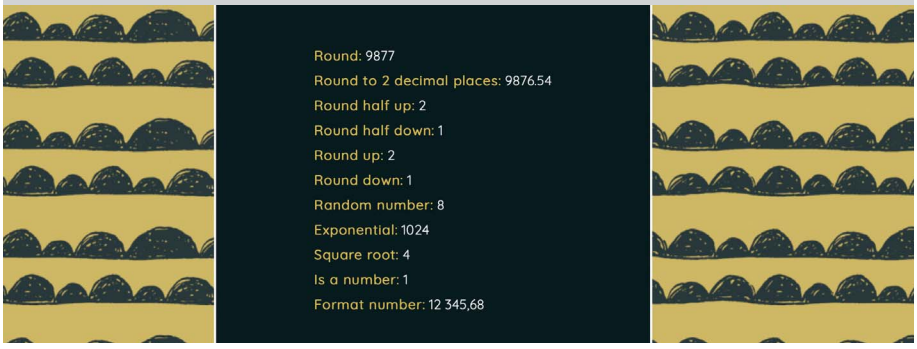
6. Число форматируется с точностью до 2 знаков после запятой. Пробел разделяет тысячи, а запятая — десятичные дроби.

Упражнение. В шаге 2 создайте случайное число от 50 до 100.

PHP section_b/c05/numeric-functions.php

```
(1) <b>Round:</b>                    <?= round(9876.54321) ?><br>
    <b>Round to 2 decimal places:</b> <?= round(9876.54321, 2) ?><br>
    <b>Round half up:</b>             <?= round(1.5, 0, PHP_ROUND_HALF_UP) ?><br>
    <b>Round half down:</b>           <?= round(1.5, 0, PHP_ROUND_HALF_DOWN) ?><br>
    <b>Round up:</b>                  <?= ceil(1.23) ?><br>
    <b>Round down:</b>                <?= floor(1.23) ?><br>
(2) <b>Random number:</b>             <?= mt_rand(0, 10) ?><br>
(3) <b>Exponential:</b>               <?= pow(4, 5) ?><br>
(4) <b>Square root:</b>               <?= sqrt(16) ?><br>
(5) <b>Is a number:</b>               <?= is_numeric(123) ?><br>
(6) <b>Format number:</b>             <?= number_format(12345.6789, 2, ',', ' ') ?><br>
```

РЕЗУЛЬТАТ



РАБОТА С МАССИВАМИ

Эти функции могут выполнять поиск по массиву, подсчитывать количество содержащихся элементов и выбирать случайные ключи. Они также могут преобразовывать массивы в строку или строку в массив.

Как вы уже видели, массивы содержат набор пар «ключ — значение» в одной переменной. В индексированном массиве ключ — это номер индекса. Он указывает положение элемента в массиве. Ассоциативный массив больше похож на набор связанных переменных: каждый ключ представляет собой строку.

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О МАССИВЕ

ФУНКЦИЯ	ОПИСАНИЕ
`array_key_exists($key, $array)`	Проверяет наличие ключа в массиве. Возвращает true, если ключ существует, в противном случае возвращает значение false
`array_search($value, $array[, $strict])`	Выполняет поиск $value среди значений массива $array и возвращает ключ для первого совпадения. Если параметр $strict будет в значении true, то проверка на совпадение будет производиться с учетом типа
`in_array($value, $array)`	Проверяет, присутствует ли значение $value в массиве $array. Возвращает значение true, если это так, и false, если нет. Параметр $strict действует так же, как и в предыдущей функции
`count($array)`	Возвращает количество элементов в массиве
`array_rand($array[, $number])`	Выбирает случайный элемент из массива и возвращает его ключ. Если в качестве второго параметра указано число, функция возвращает массив, содержащий указанное количество случайных ключей

ПРЕВРАЩЕНИЕ МАССИВА В СТРОКУ И ОБРАТНО

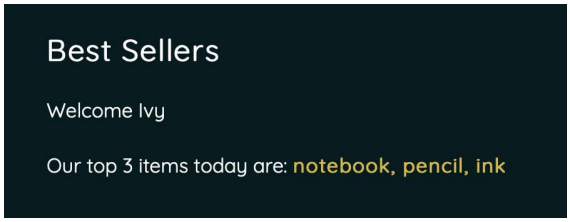
ФУНКЦИЯ	ОПИСАНИЕ
`implode([$separator, ]$array)`	Соединяет все элементы массива $array в строку (только значения, ключи в нее не включаются), вставляя между ними значение, указанное в *$separator*
`explode($separator, $string[, $limit])`	Разбивает строку $string на части, используя значение $separator в качестве разделителя, и возвращает индексированный массив. Дополнительное ограничение (*$limit*) можно использовать для установки максимального количества элементов в массиве

ФУНКЦИИ ДЛЯ РАБОТЫ С МАССИВАМИ

PHP section_b/c05/array-functions.php

```php
<?php
// Создание массива приветствий, а затем получение
// случайного значения
$greetings    = ['Hi ', 'Howdy ', 'Hello ', 'Hola ',
                 'Welcome ', 'Ciao ',];                    (1)
$greeting_key = array_rand($greetings);                    (2)
$greeting     = $greetings[$greeting_key];                 (3)
// Массив бестселлеров, подсчет товаров, вывод списком
$bestsellers      = ['notebook', 'pencil', 'ink',];        (4)
$bestseller_count = count($bestsellers);                   (5)
$bestseller_text  = implode(', ', $bestsellers);           (6)
// Массив, содержащий сведения о клиентах
$customer     = ['forename' => 'Ivy',                      (7)
                 'surname'  => 'Stone',
                 'email'    => 'ivy@eg.link',];
// Если у нас есть имя клиента, добавить его
// в приветствие
if (array_key_exists('forename', $customer)) {             (8)
    $greeting .= $customer['forename'];
}
?> ...
<h1>Best Sellers</h1>
<p><?= $greeting ?></p>                                    (9)
<p>Our top <?= $bestseller_count ?> items today are:       (10)
   <b><?= $bestseller_text ?></b></p>
```

РЕЗУЛЬТАТ



1. В массив с именем `$greetings` добавляется несколько приветствий.
2. Из массива выбирается случайный ключ и сохраняется в переменной с именем `$greeting_key`.
3. Случайный ключ используется для выбора приветствия из массива и сохранения его в `$greeting`.
4. Массив самых популярных товаров хранится в `$bestsellers`.
5. Функция `count()` используется для подсчета количества элементов в массиве. Результат сохраняется в `$bestseller_count`.
6. Массив преобразуется в строку с помощью `implode()`, каждый элемент отделяется запятой и пробелом. Результат присваивается переменной `$bestseller_text`.
7. Создается ассоциативный массив, содержащий сведения о клиенте.
8. `array_key_exists()` проверяет, существует ли элемент с именем пользователя. Если да, то его значение добавляется к `$greeting`.
9. Выводится приветствие.
10. Отображается количество бестселлеров и их названия.

Упражнение. В шаге 4 добавьте еще один товар в массив бестселлеров.

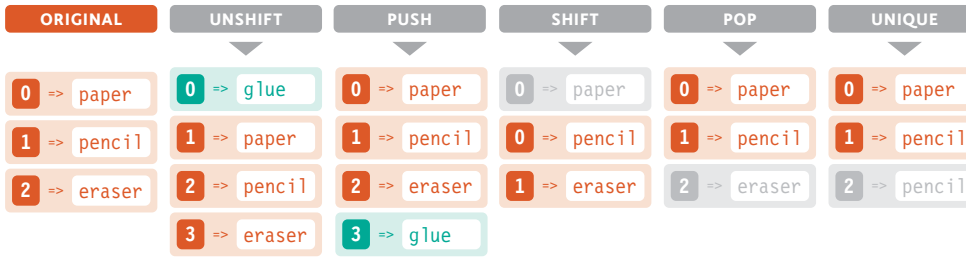
ДОБАВЛЕНИЕ И УДАЛЕНИЕ ЭЛЕМЕНТОВ МАССИВА

Эти функции добавляют и удаляют элементы массива. Вы можете указать, следует ли добавлять новые элементы в начало или в конец массива.

Чтобы добавить элемент в массив, необходимо указать добавляемое значение. Для удаления элемента из массива требуется указать его ключ. На диаграмме внизу показаны позиции добавляемых или удаляемых элементов.

ФУНКЦИЯ	ОПИСАНИЕ
`array_unshift($array, $items)`	Добавляет один или несколько элементов в начало индексированного массива. Возвращает количество элементов в массиве. Для добавления элементов в начало ассоциативного массива используйте функцию array_merge(), упомянутую ниже
`array_push($array, $items)`	Добавляет один или несколько элементов в конец индексированного массива. Возвращает новое количество элементов в массиве. Для добавления нескольких элементов в конец ассоциативного массива используйте array_merge(). Для добавления одного элемента в конец массива используйте стандартный синтаксис с квадратными скобками
`array_shift($array)`	Удаляет первый элемент из массива. Возвращает значение удаленного элемента
`array_pop($array)`	Удаляет последний элемент из массива. Возвращает значение удаленного элемента
`array_unique($array)`	Удаляет повторяющиеся записи из массива. Возвращает новый массив
`array_merge($array1, $array2)`	Объединяет два или более массивов и возвращает новый массив. Если массивы имеют одинаковые строковые ключи, следующее значение с таким же ключом будет перезаписывать предыдущее. Однако элементы с совпадающими числовыми ключами будут не перезаписаны, а добавлены в конец массива. Так же можно объединить два массива с помощью оператора +, $array1 + $array2, но в этом случае элементы с совпадающими числовыми ключами будут перезаписаны

ORIGINAL — Исходный массив
Остальные названия столбцов повторяют слова названий функций из таблицы выше



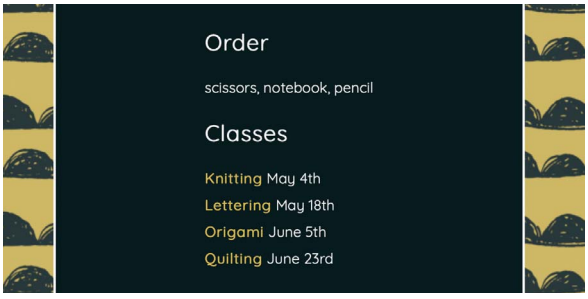
ФУНКЦИИ ДЛЯ ДОБАВЛЕНИЯ И УДАЛЕНИЯ ЭЛЕМЕНТОВ МАССИВА

PHP section_b/c05/array-updating-functions.php

```php
<?php
// Массив заказываемых товаров
$order = ['notebook', 'pencil', 'eraser',];
array_unshift($order, 'scissors'); // Добавление
                                   // в начало
array_pop($order);       // Удаление последнего элемента
$items = implode(', ', $order);    // Преобразование
                                   // в строку

// Массив занятий
$classes = ['Patchwork' => 'April 12th',
            'Knitting'  => 'May 4th',
            'Lettering' => 'May 18th',];
array_shift($classes);         // Удаление 1-го элемента
$new   = ['Origami'  => 'June 5th',
          'Quilting' => 'June 23rd',];  // Новые товары
$classes = array_merge($classes, $new); // Добавление
                                        // в конец
?>
<h1>Order</h1>
<?= $items ?>
<h1>Classes</h1>
<?php foreach($classes as $description => $date) { ?>
  <b><?= $description ?></b> <?= $date ?><br>
<?php } ?>
```

РЕЗУЛЬТАТ



1. Создается индексированный массив и сохраняется в переменной `$order`.
2. Функция `array_unshift()` добавляет элемент в начало массива. Первый параметр — это массив, второй — добавляемый элемент (это работает только с индексированными массивами).
3. Функция `array_pop()` удаляет последний элемент.
4. Массив преобразуется в строку с помощью `implode()` и сохраняется в `$items`. Каждый элемент отделяется запятой и пробелом.
5. Создается ассоциативный массив и сохраняется в переменной `$classes`.
6. `array_shift()` удаляет первый элемент из массива.
7. Создается еще один ассоциативный массив с новыми элементами.
8. `array_merge()` используется для добавления в конец массива `$classes` элементов, созданных в шаге 7, и результат сохраняется в переменной `$classes`.
9. Выводится значение `$items`.
10. В цикле `foreach` выводятся ключи и значения ассоциативного массива.

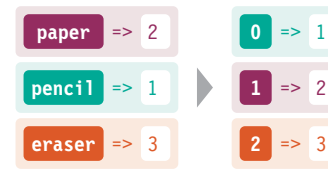
Упражнение. В шаге 4 разделить элементы в строке точкой с запятой.

СОРТИРОВКА МАССИВОВ (ИЗМЕНЕНИЕ ПОРЯДКА ЭЛЕМЕНТОВ)

Функции сортировки изменяют порядок элементов в массиве. Сортировка по возрастанию упорядочивает элементы от меньшего значения к большему (например, A–Z или 0–9). Сортировка по убыванию упорядочивает элементы от большего значения к меньшему (например, Z–A или 9–0).

СОРТИРОВКА ПО ЗНАЧЕНИЮ С ИЗМЕНЕНИЕМ КЛЮЧЕЙ

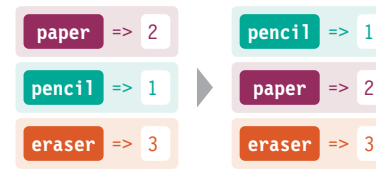
Когда вы сортируете массив с помощью приведенных ниже функций, ключи становятся индексными номерами, начинающимися с 0 (независимо от того, был ли массив индексированным или ассоциативным).
Буква r в функции `rsort()` означает обратный порядок.



ФУНКЦИЯ	ОПИСАНИЕ
`sort($array)`	Сортировка значений по возрастанию
`rsort($array)`	Сортировка значений по убыванию

СОРТИРОВКА ПО ЗНАЧЕНИЮ С СОХРАНЕНИЕМ КЛЮЧЕЙ

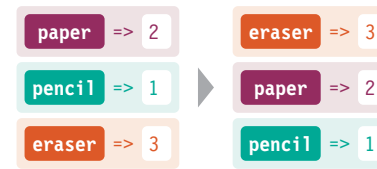
Когда вы сортируете массив с помощью приведенных ниже функций, ключи перемещаются вместе со своими значениями.



ФУНКЦИЯ	ОПИСАНИЕ
`asort($array)`	Сортировка значений по возрастанию
`arsort($array)`	Сортировка значений по убыванию

СОРТИРОВКА ПО КЛЮЧУ С СОХРАНЕНИЕМ ЗНАЧЕНИЙ

Когда вы сортируете массив с помощью приведенных ниже функций, значения перемещаются вместе со своими ключами.



ФУНКЦИЯ	ОПИСАНИЕ
`ksort($array)`	Сортировка ключей по возрастанию
`krsort($array)`	Сортировка ключей по убыванию

ФУНКЦИИ СОРТИРОВКИ МАССИВОВ

PHP section_b/c05/array-sorting-functions.php

```php
<?php
// Массив, содержащий сведения о заказах
$order = ['notebook', 'pencil', 'scissors',
          'eraser', 'ink', 'washi tape',];
sort($order);                //Сортировка по возрастанию
$items = implode(', ', $order); //Преобразование в текст

// Создание массива, содержащего список занятий
$classes = ['Patchwork' => 'April 12th',
            'Knitting'  => 'May 4th',
            'Origami'   => 'June 8th',];
ksort($classes);                // Сортировка по ключу
?>

<h1>Order</h1>
<?= $items ?>
<h1>Classes</h1>
<?php foreach($classes as $description => $date) { ?>
  <b><?= $description ?></b> <?= $date ?><br>
<?php } ?>
```

РЕЗУЛЬТАТ



1. Создается индексированный массив и присваивается переменной `$order`.
2. Значения в массиве сортируются в алфавитном порядке по возрастанию с помощью `sort()`. Это дает каждому элементу в массиве новый индексный номер, начинающийся с 0.
3. Массив преобразуется в строку с помощью `implode()`. Каждый элемент разделяется запятой, за которой следует пробел. Результат сохраняется в переменной `$text`.
4. Создается ассоциативный массив и сохраняется в переменной `$classes`.
5. Ключи в массиве сортируются в алфавитном порядке с помощью функции `ksort()` (значения перемещаются вместе с ключами).
6. Выводится строка, содержащаяся в `$items`.
7. Цикл `foreach` используется для вывода ключей и значений массива `$classes` на страницу.

Упражнение. В шаге 5 измените порядок массива, содержащего список занятий, на обратный.

КОНСТАНТЫ

Константа представляет собой пару «имя — значение» и действует как переменная. Однако после того, как значение присвоено, изменить его больше нельзя.

Константа — это пара «имя — значение», как и переменная, но:

- она создается с помощью функции `define()`;
- ее значение невозможно изменить после того, как оно было задано;
- к ней можно получить доступ в любой области видимости (включая код внутри функций).

Название константы должно описывать суть лежащих в ней данных и начинаться с буквы или символа подчеркивания (не символа доллара). Ее значением может быть скалярный тип данных или массив.

Параметры функции `define()`:

- имя константы, которое обычно пишется в верхнем регистре. Это строка, поэтому она пишется в кавычках;
- еее значение. Строки должны быть заключены в кавычки, числа и логические значения — нет;
- необязательное логическое значение, указывающее, чувствительно ли имя к регистру (`true`, если чувствительно, `false`, если нет). Если третий параметр не указан, регистр будет учитываться. Этот параметр объявлен устаревшим и не рекомендуется к использованию.

```
define('SITE_NAME', 'Mountain Art Supplies');
```

ИМЯ — 'SITE_NAME'; ЗНАЧЕНИЕ — 'Mountain Art Supplies'

Константы часто используются для хранения настроек, необходимых сайту для работы. Такие значения меняются только при создании сайта или при перемещении его на другой сервер, а также в случае, когда тот же самый код используется для создания нового сайта.

Константу также можно создавать с помощью ключевого слова `const`. За ним следует имя константы, оператор присваивания и значение, которое она должна содержать. Этот способ также может быть использован для определения константы внутри класса (функция `define()` не дает такой возможности).

```
const SITE_NAME = 'Mountain Art Supplies';
```

ИМЯ — SITE_NAME; ЗНАЧЕНИЕ — 'Mountain Art Supplies'

ИСПОЛЬЗОВАНИЕ КОНСТАНТ

PHP section_b/c05/includes/settings.php

```php
<?php
define('SITE_NAME', 'Mountain Art Supplies');
const ADMIN_EMAIL = 'admin@eg.link';
```

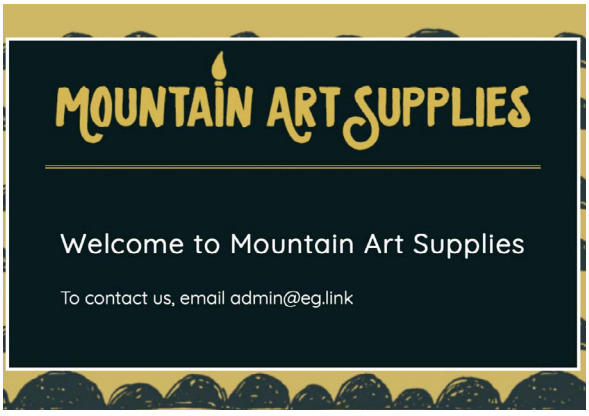
PHP section_b/c05/includes/constants.php

```php
<?php
include 'includes/settings.php';
include 'includes/header.php';
?>

<h1>Welcome to <?= SITE_NAME ?></h1>
<p>To contact us, email <?= ADMIN_EMAIL ?></p>

<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



В этом примере в подключаемом файле с названием `settings.php` создается две константы, содержащие информацию о сайте.

1. Функция `define()` используется для создания константы с именем `SITE_NAME`. Ее значение — это название сайта.
2. Ключевое слово `const` используется для создания константы, вызываемой `ADMIN_EMAIL`. Ее значением будет адрес электронной почты владельца сайта.

Второй файл в этом примере называется `constants.php` и в нем выводятся значения этих двух констант.

3. Файл `settings.php` подключается к странице `constants.php`, чтобы ее код мог получить доступ к константам.
4. Сокращенная команда `echo` используется для вывода содержимого константы с именем сайта.
5. Отображается адрес электронной почты владельца сайта.

ДОБАВЛЕНИЕ ИЛИ ИЗМЕНЕНИЕ HTTP-ЗАГОЛОВКОВ

Функция `header()` отправляет HTTP-заголовок в браузер, в дополнение к тем, которые веб-сервер и интерпретатор PHP отправляют автоматически. Если такой заголовок уже был отправлен, то эта функция изменяет его. Ее единственный аргумент — строка, которая содержит имя и значение устанавливаемого заголовка, разделенные двоеточием.

Иногда пользователи запрашивают одну страницу, но вам нужно отправить их на другую. Например, если текущая:

- больше недоступна;
- перемещена на новый URL;
- требует дополнительных данных.

В этом случае в `header()` передается аргумент, состоящий из следующих частей:

- имя заголовка (`Location`);
- двоеточие;
- новый URL.

Когда браузер получает заголовок `Location`, он запрашивает указанный в нем URL. После отправки этого заголовка всегда должна идти команда `exit`, чтобы предотвратить дальнейшее выполнение PHP-кода (см. справа).

```
header('Location: http://www.example.com/');
```

ЗАГОЛОВОК — НОВЫЙ URL-АДРЕС

Чаще всего PHP используется для генерации HTML, выводимого в браузер, но PHP можно использовать для создания файлов и других типов, например JSON, XML или CSS.

Для этого функции `header()` потребуются:

- имя заголовка (Content-type);
- двоеточие;
- MIME-тип следующего после заголовков содержимого.

Это создает HTTP-заголовок, сообщающий браузеру MIME-тип файла. Дополнительные сведения о MIME-типах можно получить здесь http://notes.re/media-types.

```
header('Content-type: application/json');
```

ЗАГОЛОВОК — MIME-ТИП

Браузеры могут кэшировать (временно сохранять) просмотренные пользователями страницы. Если пользователь запрашивает ее снова, браузер может показать страницу, которую он сохранил, вместо того чтобы повторно запрашивать ее с сервера, и это ускоряет ее отображение. Чтобы указать браузеру, на какой период он может кэшировать страницу, используйте:

- заголовок `Cache-Control`;
- двоеточие;
- строку `max-age=` с последующим указанием количества секунд, в течение которых страница может быть кэширована.

Интернет-провайдеры могут использовать **прокси-серверы** (**proxy**) для кэширования веб-страниц. Если страницы содержат личные данные, после миллисекунд ставьте запятую, пробел и слово `private`. В таком случае прокси-сервер не будет кэшировать их. Если личных данных нет, установите для них значение `public`.

```
header('Cache-Control: max-age=3600, public');
```

ЗАГОЛОВОК — ДЕЙСТВИТЕЛЕН НА — ВРЕМЯ (С) — ПРИВАТНОСТЬ

ПЕРЕНАПРАВЛЕНИЕ ПОЛЬЗОВАТЕЛЕЙ С ИСПОЛЬЗОВАНИЕМ HTTP-ЗАГОЛОВКОВ

PHP section_b/c05/redirect.php

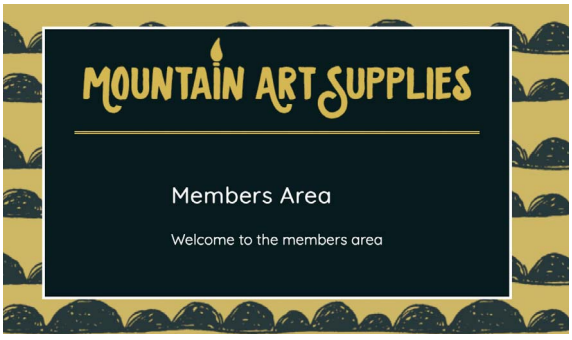
```
<?php
① $logged_in = true;

② if ($logged_in == false) {
③     header('Location: login.php');
④     exit;
}
?>
<?php include 'includes/header.php'; ?>
<h1>Members Area</h1>
<p>Welcome to the members area</p>
<?php include 'includes/footer.php'; ?>
```

PHP section_b/c05/login.php

```
<h1>Login</h1>
<b>You need to log in to view this page.</b>
<p>(You create a full login system in Chapter 16.)</p>
```

RESULT



Упражнение. Измените в шаге 1 значение переменной `$logged_in` на `false`, после чего вы будете перенаправлены на страницу входа в систему.

В этом примере показано, как перенаправить пользователя на другую страницу с помощью функции `header()`. Нельзя отправлять в браузер какую-либо разметку или текст до использования функции `header()`, даже пробел или возврат каретки, иначе попытка выполнить эту функцию приведет к ошибке.

1. Переменная с именем `$logged_in` содержит логическое значение, указывающее, вошел ли пользователь в систему.
2. Условие в конструкции `if` проверяет значение переменной `$logged_in`.
3. Если оно равно `false`, пользователь перенаправляется на страницу `login.php` с использованием функции `header()`.

О том, как создать личный кабинет с работающей страницей входа в систему, вы узнаете в главе 16.

4. Команда `exit` предотвращает дальнейшее выполнение PHP-кода в файле после отправки заголовка `Location`.

Если значение в `$logged_in` равно `true`, блок кода в конструкции `if` будет пропущен и будет показана остальная часть страницы.

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О ФАЙЛАХ И ИХ УДАЛЕНИЕ

Функции для работы с файлами принимают путь к файлу в качестве параметра, а затем либо возвращают информацию о файле, либо выполняют с ним какое-то действие, например удаление.

В таблице ниже показаны некоторые функции для работы с файлами. Часть этих функций возвращают разные фрагменты пути к файлу; эти фрагменты описаны на схеме справа.
У PHP также есть встроенные константы:
`__FILE__` содержит путь к текущему файлу;
`__DIR__` содержит директорию текущего файла.



ФУНКЦИЯ	ОПИСАНИЕ
`file_exists($path)`	Проверяет существование файла. Возвращает значение true, если он существует, и false, если нет
`filesize($path)`	Возвращает размер файла в байтах
`mime_content_type($path)`	Возвращает MIME-тип файла (см. http://notes.re/media-types)
`unlink($path)`	Удаляет файл. Возвращает значение true, если выполнение успешно, и false, если нет
`pathinfo($path[, $part])`	Возвращает информацию о пути к файлу, переданном в аргументе $path. С помощью второго параметра вы можете указать конкретный фрагмент для извлечения. Если вы этого не сделаете, функция вернет массив со следующими четырьмя ключами:

ФРАГМЕНТ	ОПИСАНИЕ
`PATHINFO_DIRNAME`	Путь к директории, в которой находится файл
`PATHINFO_BASENAME`	Имя файла
`PATHINFO_FILENAME`	Имя файла (без расширения)
`PATHINFO_EXTENSION`	Расширение имени файла

`basename($path)`	Возвращает имя файла из пути, содержащегося в $path
`dirname($path[, $levels])`	Возвращает имя каталога для указанного в $path пути. Дополнительный параметр $levels указывает, сколько элементов надо убрать из возвращаемого результата
`realpath($path)`	Возвращает абсолютный путь к файлу

Разница между абсолютным и относительным путями описана здесь **http://notes.re/paths**[28].

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О ФАЙЛЕ

PHP section_b/c05/files.php

```
<?php
① $path = 'img/logo.png';
?>
<?php include 'includes/header.php'; ?>
② <?php if (file_exists($path)) { ?>
③   <b>Name:</b>      <?= pathinfo($path, PATHINFO_BASENAME) ?><br>
④   <b>Size:</b>      <?= filesize($path) ?> bytes<br>
⑤   <b>Mime type:</b> <?= mime_content_type($path) ?><br>
⑥   <b>Folder:</b>    <?= pathinfo($path, PATHINFO_DIRNAME) ?><br>
<?php } else { ?>
⑦   <p>There is no such file.</p>
<?php } ?>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



Упражнение. В шаге 1 изменить `$path` на `img/pattern.png`. Вы увидите новое имя и размер (тип `mime` и папка останутся прежними).

Упражнение. В шаге 1 изменить `$path` на `img/nologo.png`. Поскольку этот файл не существует, вы должны увидеть сообщение об ошибке в шаге 7.

1. Переменная `$path` содержит путь к файлу.

2. В конструкции `if` используется функция `file_exists()`, чтобы проверить, существует ли файл. Если существует, функция выводит информацию о файле.

3. Функция `pathinfo()` показывает имя файла, включая его расширение (известное как полное имя).

4. Функция `filesize()` показывает размер файла в байтах.

5. Функция `mime_content_type()` показывает MIME-тип файла.

6. Функция `pathinfo()` показывает путь к папке, в которой находится файл.

7. Если файл не существует, пользователю сообщается, что такого файла нет.

ЗАКЛЮЧЕНИЕ
ВСТРОЕННЫЕ ФУНКЦИИ

- Программисты могут использовать встроенные функции PHP для выполнения множества стандартных операций, которые требуются при создании сайтов.
- Встроенные функции вызываются точно так же, как и любые другие, но не требуют добавления определения функции на страницу.
- Строковые функции ищут, подсчитывают и заменяют символы в тексте, а также изменяют регистр букв.
- Числовые функции округляют числа, выполняют математические операции и генерируют случайные числа.
- Функции для работы с массивами добавляют и удаляют элементы, сортируют содержимое массива, проверяют наличие ключей или значений и превращают массивы в строки и обратно.
- Константы подобны переменным, но без возможности изменить значение после его установки.
- Функция `header()` отправляет HTTP-заголовки в браузер. В частности, может перенаправить пользователя на другую страницу.

6

ПОЛУЧЕНИЕ ДАННЫХ ИЗ БРАУЗЕРА

В этой главе вы узнаете, как получить доступ к данным, которые браузер отправляет на сервер, а также как проверить, что они готовы к использованию и безопасны для отображения на веб-странице.

Во введении к этому разделу вы узнали, что чаще всего для отправки данных на сервер используются два механизма: с помощью добавления информации к ссылкам или через заполнение формы. Вы также видели, как эти данные отправляются через HTTP GET (в строке запроса) или HTTP POST (в теле запроса HTTP).

В этой главе вы узнаете, как получить доступ к этим данным в PHP, чтобы их можно было использовать на странице.

Этот процесс включает в себя четыре ключевых шага.

- **Получение** каждого фрагмента данных из строки запроса или тела HTTP-запроса.
- **Валидация** каждого фрагмента данных. Это позволяет проверить, что значение передано и что оно находится в правильном формате (например, если на странице нужно число, вы проверяете, что было указано число, а не текст).
- **Принятие решения** о том, можно ли продолжить обработку данных, предоставленных пользователем. И если нет, то потребуется ли показать ему сообщения об ошибках.
- **Экранирование**, или **очистка** данных, для обеспечения их безопасного использования на странице, поскольку определенные символы могут помешать правильному отображению страницы или даже нанести вред сайту.

Не существует единого стандартного способа выполнения каждого из этих четырех шагов — разные разработчики используют разные подходы. В этой главе представлена подборка из множества различных способов сбора данных и обеспечения их безопасного использования.


Chapter 6 Examples
phpandmysql.com/code/section_b/c06/collecting-form-data.php
MOUNTAIN ART SUPPLIES
Name: Ivy
Age: 26
Email: ivy@eg.link
Password:
Bio:
Contact preference: Email
Rating: 1 2 3
I agree to the terms and conditions.
SAVE
Type here to search
10:27
06/02/2021
Book of Seaweed
A. Major
EVERYONES BOOK OF BRITISH NATURAL HISTORY

ЧЕТЫРЕ ШАГА ДЛЯ ПОЛУЧЕНИЯ И ИСПОЛЬЗОВАНИЯ ДАННЫХ

Чтобы получить данные от пользователя и обеспечить безопасность их использования, требуется выполнить четыре определенных шага.

1. ПОЛУЧЕНИЕ ДАННЫХ

Сначала вам нужно получить данные, которые браузер отправил на сервер. Для этого можно использовать:

- два суперглобальных массива, которые интерпретатор PHP создает каждый раз, когда запрашивается файл PHP;
- две встроенные функции, называемые **функциями фильтрации**.

Как вы увидите в дальнейшем, PHP-скрипт не всегда будет получать значения, необходимые ему для выполнения своей задачи, и это будет приводить к ошибке.

Если часть данных необязательна, вы можете указать значение по умолчанию, которое должно использоваться на странице, если значение не указано.

Если же значение требуется обязательно, но отсутствует, вам будет неоходимо сообщить посетителю, что он предоставил недостаточно информации (см. следующий шаг).

2. ВАЛИДАЦИЯ ДАННЫХ

После того как на странице PHP получены данные из браузера, обычно осуществляется их **валидация** (т. е. проверка, соответствуют ли они ожиданиям) для каждого отдельного полученного фрагмента данных. Это делается чтобы убедиться, что данные не вызовут ошибок при использовании в коде страницы. Виды проверки:

- получены ли все данные, необходимые для выполнения задачи. Они называются **обязательными данными**;
- относятся ли данные к **правильному формату**. Например, если коду на вашей странице требуется число для выполнения вычислений, вы можете проверить, что полученное значение действительно является числом. Или, если вы ожидаете получить адрес электронной почты, вы можете проверить, что введенное значение соответствует формату адреса электронной почты.

PHP предоставляет два способа валидации данных:

- вы можете написать собственные функции валидации;
- или использовать одну из встроенных функций с фильтрами для проверки данных. Каждый фильтр проводит валидацию своего типа данных.

Существуют различные способы выполнения каждого из этих шагов, и вы познакомитесь с некоторыми из них в текущей главе.

3. ПРИНЯТИЕ РЕШЕНИЯ

После получения и проверки переданных значений PHP-скрипт должен определить, есть ли у него все необходимые для дальнейшей работы данные. Затем принимается решение.

- Если все данные проверены и соответствуют ожиданиям, их можно обрабатывать.
- Если данные не прошли валидацию или отсутствуют, их не следует использовать. Вместо этого страница может выдать пользователю сообщение об ошибке.

Отображение ошибок отличается при использовании при передаче данных через форму и через строку запроса.

- Если данные формы не прошли проверку, можно отобразить ее снова и вывести сообщения об ошибках, рядом с соответствующими элементами формы. В сообщениях должна быть подсказка, какие именно данные ожидаются.
- Если неверные данные содержит строка запроса, вам не стоит ожидать, что посетители будут ее редактировать. Вместо этого можно вывести либо сообщение об ошибке, либо сообщение, которое объясняет, как запросить нужные ему данные.

4. ЭКРАНИРОВАНИЕ, ИЛИ ОЧИСТКА ДАННЫХ

Всякий раз, когда вы отображаете на HTML-странице данные, предоставленные посетителем[29], их необходимо **экранировать**, чтобы убедиться, что они безопасны для отображения. Процесс включает в себя замену некоторых символов, которые браузеры обрабатывают как код (например, символы < и >), на **HTML-сущности**. Браузер отображает их как есть, вместо того чтобы обрабатывать в виде HTML-кода.

Если вы не выполните этот шаг перед отображением данных на странице, хакер может попытаться заставить страницу запустить вредоносный код JavaScript.

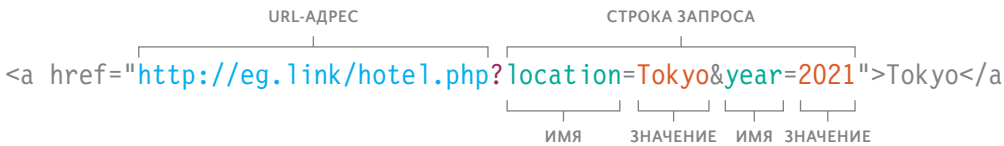
При передаче данных через URL в строке запроса вам также необходимо экранировать любые символы, имеющие особое значение (например, амперсанды и вопросительные знаки). Веб-сервер может быть не в состоянии обработать URL, если не экранировать эти символы.

ПОЛУЧЕНИЕ ДАННЫХ, ОТПРАВЛЕННЫХ ЧЕРЕЗ HTTP GET

Когда данные передаются через в строку запроса в URL, интерпретатор PHP добавляет эти данные в суперглобальный массив с именем `$_GET`, чтобы PHP-код на странице мог получить к ним доступ.

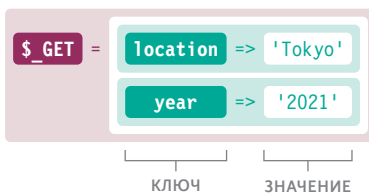
Ниже представлена HTML-ссылка. В ее атрибуте `href` отображается URL-адрес страницы, на которую она ведет.

В конце него есть строка запроса, содержащая две пары «имя — значение». Они отправляются на сервер, когда посетитель нажимает на ссылку.



Когда интерпретатор PHP получает этот запрос, он добавляет данные из строки запроса в суперглобальный массив с именем `$_GET`. Как и все суперглобальные массивы в PHP, `$_GET` — ассоциативный массив. В него добавляется элемент для каждой пары «имя — значение», находящейся в строке запроса. При этом:

- **ключом** массива становится имя элемента;
- **значением** — значение, отправленное под этим именем.



В коде PHP можно получить доступ к значениям суперглобального массива `$_GET` таким же образом, как и к значениям любого ассоциативного массива:



Часто один PHP-файл используется для отображения нескольких страниц веб-сайта, а данные в строке запроса — для определения того, какие данные он будет отображать.

На странице справа массив состоит из трех элементов. Каждый элемент содержит название города и адрес магазина в нем. Значение в строке запроса указывает, данные какого магазина должны отображаться, поэтому с помощью одного этого PHP-файла можно отобразить три разные страницы сайта — каждая из них посвящена отдельному магазину. Данные в массиве также используются для создания ссылок, запрашивающих эти три страницы.

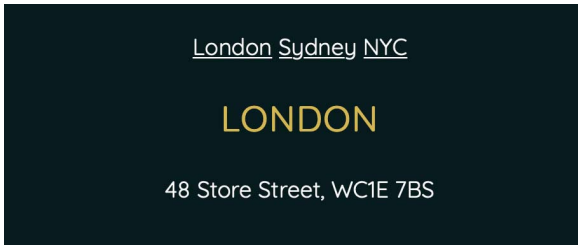
ИСПОЛЬЗОВАНИЕ СТРОКИ ЗАПРОСА ДЛЯ ВЫБОРА, КАКОЙ КОНТЕНТ ПОКАЗЫВАТЬ

PHP section_b/c06/get-1.php?city=London

```
<?php
$cities  = [
    'London' => '48 Store Street, WC1E 7BS',
    'Sydney' => '151 Oxford Street, 2021',
    'NYC'    => '1242 7th Street, 10492',
];
$city    = $_GET['city'];
$address = $cities[$city];
?>
...
<?php foreach ($cities as $key => $value) { ?>
  <a href="get-1.php?city=<?= $key ?>"><?= $key ?></a>
<?php } ?>

<h1><?= $city ?></h1>
<p><?= $address ?></p>
```

РЕЗУЛЬТАТ



В этом примере код PHP получает название города из строки запроса и показывает адрес магазина в этом городе.

1. Переменная `$cities` содержит ассоциативный массив. Каждый ключ — это названия разных городов, а каждое значение — это адрес филиала магазина в этом городе.
2. Название города берется из суперглобального массива `$_GET` и присваивается переменной с именем `$city`.
3. Название города используется для выбора адреса филиала в этом городе из массива, созданного в шаге 1. Результат присваивается переменной `$address`. Обратите внимание, что эта операция чувствительна к регистру.
4. Цикл `foreach` перебирает каждый элемент в массиве `$cities`.
5. Внутри цикла создается ссылка для каждого города. Название города добавляется в строку запроса, а затем еще раз выводится в виде текста ссылки. Это показывает, как PHP может создавать ссылки и как эти ссылки могут указывать на один файл, способный отображать разные данные.
6. Значения, присвоенные переменным `$city` и `$address` на шагах 2 и 3, отображаются на странице.

Упражнение. Удалите строку запроса из URL в адресной строке браузера (знак вопроса и все что идет после него) и перезагрузите страницу. Она должна отобразить две ошибки.

Это связано с тем, что названия города не было в строке запроса и, следовательно, оно не было добавлено в суперглобальный массив `$_GET`.

ОБРАБОТКА ОТСУТСТВИЯ ДАННЫХ В СУПЕРГЛОБАЛЬНЫХ МАССИВАХ

Если вы попытаетесь получить доступ к ключу, который отсутствует в массиве, интерпретатор PHP выдаст сообщение об ошибке. Чтобы предотвратить это, можно проверить, существует ли ключ в массиве, прежде чем обращаться к нему.

Когда кто-то делится ссылкой на страницу, он может случайно пропустить часть или всю строку запроса.

В упражнении для предыдущего примера вы видели, что если в строке запроса отсутствуют данные, то они и не попадают в суперглобальный массив `$_GET`. Если после этого попытаться получить доступ к элементам этого массива, то интерпретатор PHP выдаст ошибку `Undefined array key` или `Undefined index`, поскольку ключ (или индекс) отсутствует в массиве.

Чтобы предотвратить эту ошибку, в коде необходимо проверить наличие значения в суперглобальном массиве `$_GET`, прежде чем пытаться получить к нему доступ.

В PHP есть встроенная конструкция под названием `isset()`. Она принимает имя переменной либо обращение к ключу массива или свойству объекта в качестве аргумента. Эта функция возвращает значение `true`, если данная переменная, ключ или свойство существуют и их значение не равно `null`. В противном случае будет возвращено значение `false`. Важно отметить, что, если указанная переменная, ключ или параметр не существуют, это не приведет к ошибке.

Ниже объявляется переменная с именем `$city`. Затем с помощью тернарного оператора (сокращенный вариант конструкций `if…else`) проверяется, есть ли в суперглобальном массиве `$_GET` ключ с именем `city` (и его значение не равно `null`). Если значение есть, оно будет присвоено переменной `$city`. В противном случае будет присвоена пустая строка.

```
$city = isset($_GET['city']) ? $_GET['city'] : '';
```

ПЕРЕМЕННАЯ — СУЩЕСТВУЕТ ЛИ КЛЮЧ? — ДА: СОХРАНИТЬ ЕГО ЗНАЧЕНИЕ — НЕТ: СОХРАНИТЬ

В PHP 7 введен оператор объединения с `null` (null-coalescing operator), который пишется как два вопросительных знака подряд — `??`. Он действует как сокращенная форма `isset()` с тернарным оператором. Если значение слева от оператора `??` не существует, то будет использовано альтернативное значение, которое указано справа.

```
$city = $_GET['city'] ?? '';
```

ПЕРЕМЕННАЯ — ПОПЫТКА ИСПОЛЬЗОВАТЬ ЭТО ЗНАЧЕНИЕ — ЕСЛИ ОНО НЕ СУЩЕСТВУЕТ: ВЗЯТЬ ПУСТУЮ СТРОКУ

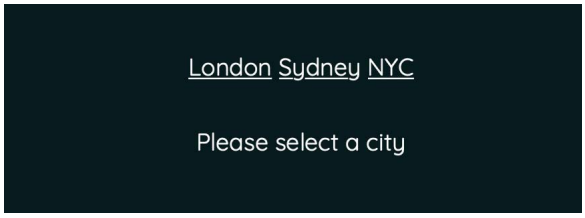
ПРИМЕР ИСПОЛЬЗОВАНИЯ СТРОКИ ЗАПРОСА ДЛЯ ВЫБОРА КОНТЕНТА

PHP section_b/c06/get-2.php

```
<?php
$cities  = [
    'London' => '48 Store Street, WC1E 7BS',
    'Sydney' => '151 Oxford Street, 2021',
    'NYC'    => '1242 7th Street, 10492',
];
(1) $city = $_GET['city'] ?? '';
(2) if ($city) {
(3)     $address = $cities[$city];
(4) } else {
(5)     $address = 'Please select a city';
}
?>
...
<?php foreach ($cities as $key => $value) { ?>
  <a href="get-2.php?city=<?= $key ?>"><?= $key ?></a>
<?php } ?>

<h1><?= $city ?></h1>
<p><?= $address ?></p>
```

РЕЗУЛЬТАТ



Упражнение. Использовать Tokyo в строке запроса в качестве города. Страница покажет сообщение об ошибке, поскольку она не может найти этот ключ в массиве `$cities`.

Добавьте новый элемент в массив (шаг 1) с ключом `Tokyo` и добавьте для него адрес, затем попробуйте снова использовать его в строке запроса.

Этот пример основан на предыдущем. Различия выделены цветом.

1. Значение переменной `$city` присваивается с помощью оператора объединения с `null`. Если в суперглобальном массиве `$_GET`:

- есть ключ с именем `city` и его значение не равно `null`, то значение этого ключа будет сохранено в переменной `$city`;
- ключа с именем `city` в массиве нет или его значение равно `null`, то переменной `$city` будет присвоена пустая строка.

2. Переменная `$city` используется в качестве условия в конструкции `if`. Если значение представляет собой не пустую строку, интерпретатор PHP обрабатывает это значение как `true` и выполняет последующий блок кода.

3. Переменной `$address` присваивается адрес филиала в городе, который был передан в строке запроса.

4. В противном случае, если значение в переменной `$city` — пустая строка, она приравнивается к `false` и выполняется блок кода в конструкции `else`.

5. Переменной `$address` присваивается сообщение, в котором посетителю предлагается выбрать город.

ВАЛИДАЦИЯ ДАННЫХ

Прежде чем использовать в PHP-скрипте полученные данные, их необходимо проверить, чтобы избежать ошибок при выполнении кода.

Валидация данных, которые получает PHP скрипт, включает в себя проверку того, были ли переданы все необходимые для выполнения задачи значения (их называют **обязательными данными**) и что эти данные получены в **правильном формате**. Например, если программе требуется число для выполнения вычислений, необходимо проверить, что она получила число, а не строку.

В примере на предыдущей странице строка запроса должна соответствовать следующим требованиям:

- содержать имя и значение, чтобы указать филиал для отображения;
- при этом значение должно соответствовать одному из ключей в массиве городов.

Если значение, указанное в строке запроса, отсутствует в массиве городов, интерпретатор PHP выдает ошибку. Поэтому, прежде чем выводить город на странице, необходимо проверить, присутствует ли значение из строки запроса в массиве городов.

В оставшейся части этой главы вы узнаете о нескольких способах валидации различных типов данных. В примере на странице справа для проверки, соответствует ли значение в строке запроса ключу в массиве городов, используется встроенная в PHP функция `array_key_exists`. Функция возвращает значение `true`, если ключ найден, и значение `false`, если его нет. Возвращаемое функцией значение сохраняется в переменной под названием `$valid`.

Как только данные будут проверены, код на странице должен определить, может ли он продолжать выполнение или необходимо вывести сообщение об ошибке.



Использовавшийся ранее в этой главе пример расширен на странице справа. Переменная `$valid` используется в качестве условия в конструкции `if`, чтобы определить, подходят ли переданные данные для дальнейшей работы страницы.

- Если данные подходят, то можно выполнять код, который получает адрес магазина из массива и присваивает его переменной `$address`, которая будет отображена далее на странице.
- Если данные не подходят, в переменную `$address` записывается сообщение, в котором пользователю предлагается выбрать город. Это обеспечивает полезную обратную связь для посетителя, рассказывая ему, как использовать страницу и получить необходимую ему информацию.

Далее в этой главе вы узнаете, как получать не одно, а несколько значений из браузера и как проверить, все ли они соответствуют требованиям.

ВАЛИДАЦИЯ ДАННЫХ ИЗ СТРОКИ ЗАПРОСА

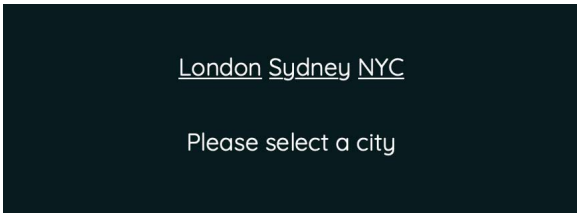
PHP section_b/c06/get-3.php

```php
<?php
$cities  = [
    'London' => '48 Store Street, WC1E 7BS',
    'Sydney' => '151 Oxford Street, 2021',
    'NYC'    => '1242 7th Street, 10492',
];
$city  = $_GET['city'] ?? '';
$valid = array_key_exists($city, $cities);

if ($valid) {
    $address = $cities[$city];
} else {
    $address = 'Please select a city';
}
?>
...
<?php foreach ($cities as $key => $value) { ?>
  <a href="get-3.php?city=<?= $key ?>"><?= $key ?></a>
<?php } ?>

<h1><?= $city ?></h1>
<p><?= $address ?></p>
```

РЕЗУЛЬТАТ



Этот пример основан на предыдущих и использует валидацию, чтобы проверить, содержит ли строка запроса допустимое значение.

1. Если строка запроса содержит название города, она будет сохранена в переменной с именем `$city`. Если нет, то `$city` будет содержать пустую строку.
2. Функция `array_key_exists()` проверяет, является ли значение в `$city` ключом массива `$cities`. Если это так, то переменная `$valid` будет содержать значение `true`. Если нет, то `$valid` будет содержать значение `false`.
3. Переменная `$valid` используется в условии оператора `if`. Если сохраненное значение равно `true`, будет выполнен первый блок кода.
4. Адрес магазина для этого города будет получен из массива `$cities` и сохранится в переменной `$address`.
5. Если значение в `$valid` равно `false`, выполняется второй блок кода.
6. Переменной `$address` будет присвоено сообщение, в котором посетителю предлагается выбрать город.

Упражнение. Ввести в адресной строке браузера название города Shanghai в строку запроса, `get-3.php?city=Shanghai`.

ОТОБРАЖЕНИЕ СТРАНИЦЫ С ОШИБКОЙ, ЕСЛИ ДАННЫЕ ОТСУТСТВУЮТ

Если на странице необходимо получить данные из строки запроса, но эти данные отсутствуют или не проходят проверку, интерпретатор PHP может указать браузеру отобразить сообщение об ошибке.

Важно всегда проверять данные, полученные в строке запроса: когда люди ссылаются на ваш сайт, им легко случайно пропустить необходимые данные.

Вам не стоит ожидать от посетителей, что они отредактируют данные в строке запроса вручную. Поэтому, если данные неверны, вы можете помочь им:

- показав сообщение об ошибке. Оно может сообщить посетителям, что запрошенная страница не найдена, или предложить им выбрать из списка параметров (как в примере на предыдущей странице);
- перенаправив пользователя на другую страницу, содержащую сообщение об ошибке[30].

Обратите внимание, как условие в приведенном ниже коде проверяет, что значение, сохраненное в переменной `$valid`, *не* является истинным.

Мы показываем, что встроенную PHP-функцию `header()` можно использовать, чтобы задать отправляемый в браузер интерпретатором PHP заголовок `Location`. Это сообщает браузеру, что нужно запросить другую страницу.

Если страница не может быть отображена из-за неверных данных, рекомендуется отправить в браузер специальный код ответа. Это помогает предотвратить добавление поисковыми системами неправильных URL-адресов в результаты поиска. Встроенная в PHP функция `http_response_code()` используется для установки кода ответа HTTP. Ее единственный аргумент — код ответа, который следует использовать. Отправка кода ответа 404 указывает, что запрошенная страница не может быть найдена.

Как только код ответа и заголовок заданы, команда `exit` останавливает выполнение любого дальнейшего кода на странице, поскольку это может привести к ошибке.

ЕСЛИ ВАЛИДАЦИЯ НЕ ПРОЙДЕНА
ЗАДАЕТ КОД ОТВЕТА
ПЕРЕНАПРАВЛЯЕТ НА СТРАНИЦУ ОШИБКИ
ОСТАНАВЛИВАЕТ ВЫПОЛНЕНИЕ КОДА

```
if (!$valid) {
    http_response_code(404);
    header('Location: page-not-found.php');
    exit;
}
```

ПРИМЕР ОТОБРАЖЕНИЯ СТРАНИЦЫ С ОШИБКОЙ

PHP section_b/c06/get-4.php

```
<?php
$cities  = [
    'London' => '48 Store Street, WC1E 7BS',
    'Sydney' => '151 Oxford Street, 2021',
    'NYC'    => '1242 7th Street, 10492',
];
$city  = $_GET['city'] ?? '';
(1) $valid = array_key_exists($city, $cities);

(2) if (!$valid) {
(3)     http_response_code(404);
(4)     header('Location: page-not-found.php');
(5)     exit;
}
$address = $cities[$city];
?>
...
<?php foreach ($cities as $key => $value) { ?>
  <a href="get-4.php?city=<?= $key ?>"><?= $key ?></a>
<?php } ?>

<h1><?= $city ?></h1>
<p><?= $address ?></p>
```

РЕЗУЛЬТАТ



Если значение в `$valid` равно `true`, шаги 3–5 игнорируются и отображается страница.

Примечание: в браузере отображается содержимое файла `page-not-found.php`, потому что в строке запроса нет города.

Код в этом примере отправляет посетителей на страницу с ошибкой, если в строке запроса будет передано недопустимое название города.

1. Функция PHP `array_key_exists()` проверяет, является ли название города, полученное из строки запроса, одним из ключей в массиве городов. Она возвращает значение `true`, если ключ существует, и `false`, если его нет. Это значение сохраняется в переменной с именем `$valid`.
2. Условие в операторе `if` проверяет, что значение, сохраненное в `$valid`, *не* является `true`. Оператор ! указывает, что логическое значение после него должно быть инвертировано. Соответственно, следующий блок кода выполнится, если `$valid` будет содержать значение, которое *не* является истинным (т. е. `false`).
3. Функция PHP `http_response_code()` указывает интерпретатору PHP отправить код ответа 404 в браузер, сообщая, что страница не может быть найдена.
4. Функция PHP `header()` указывает интерпретатору PHP добавить заголовок `Location:`, чтобы дать указание браузеру вместо текущей страницы запросить файл с именем `page-not-found.php`.
5. Команда PHP `exit` указывает интерпретатору PHP не выполнять дальнейший код в файле.

ЭКРАНИРОВАНИЕ ВЫВОДА

Когда отправленные на сервер данные отображаются на странице, они должны быть **экранированы**, чтобы хакеры не могли использовать их для запуска вредоносных скриптов[31].

Экранирование данных — это процесс замены определенных символов на их безопасные эквиваленты. Например, HTML содержит пять **зарезервированных символов**, которые браузеры рассматривают как код:

< и > используются в тегах;
" и ' содержат значения атрибутов;
& используется для создания сущностей.

Чтобы отобразить эти пять символов на странице, их необходимо заменить на соответствующие **HTML-сущности**, как именованные, так и использующие код символа. В результате браузер отобразит соответствующие символы, а не будет рассматривать их как код.



<	>	&	"	'
<	>	&	"	'

При выводе данных на HTML-страницу PHP код должен проверить наличие этих пяти зарезервированных символов и заменить их соответствующими сущностями. Это можно сделать с помощью встроенной в PHP функции `htmlspecialchars()`.

Если вы не замените зарезервированные символы HTML-сущностями, хакеры могут отправить значения, загружающие файл JavaScript с вредоносным кодом внутри. Эта атака называется **межсайтовым скриптингом** (Cross-site scripting, XSS).

Например, если посетитель введет следующее имя пользователя, а затем оно будет выведено без всякой обработки, это может привести к запуску скрипта по ссылке.

```
Luke<script src="http://eg.link/bad.js">
</script>
```

Когда зарезервированные символы будут заменены сущностями, скрипт не будет запущен, а посетители увидят приведенный выше текст как есть. Если же открыть исходный HTML для этой страницы, то имя пользователя будет выглядеть следующим образом:

```
Luke&lt;script src=&quot;http://eg.link/
bad.js&quot;&gt;&lt;/script&gt;
```

Введенные пользователями данные должны отображаться только в HTML-разметке, которая видна на веб-странице (или в элементах `<title>` или `<meta>`). **Не** отображайте данные, предоставленные пользователем:

- в комментариях вашего кода;
- правилах CSS, поскольку они могут включать в себя скрипт на странице;
- элементах `<script>`;
- именах тегов;
- названиях атрибутов;
- в качестве значения атрибутов событий HTML, например `onclick` и `onload`;
- как значения атрибута HTML, который загружает файлы (например, атрибут `src`).

Значения, используемые в URL-адресе или строке запроса, также должны быть экранированы.

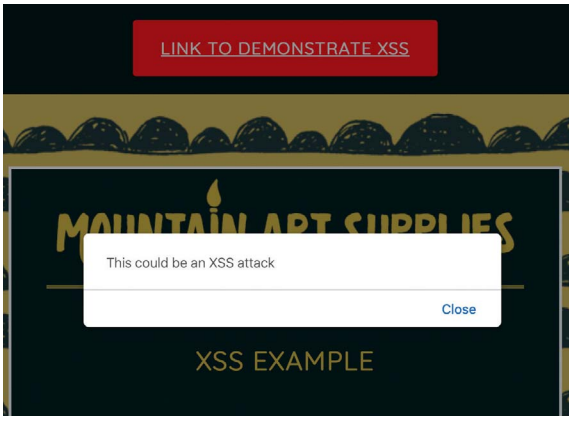
РИСК НЕЭКРАНИРОВАННОГО ВЫВОДА

PHP section_b/c06/xss-1.php

```
① <a class="badlink" href="xss-1.php?msg=<script
  src=js/bad.js></script>">LINK TO DEMONSTRATE XSS</a>

  <?php
② $message = $_GET['msg'] ?? 'Click link at top of page';
  ?>
...
  <h1>XSS Example</h1>
③ <p><?= $message ?></p>
```

РЕЗУЛЬТАТ



Примечание: на следующей странице показано, как экранирование выводимых данных приводит к тому, что браузер выводит теги скрипта как текст и не рассматривает их как код.

Этот пример показывает, что произойдет, если данные не будут экранированы.

1. В целях демонстрации, ссылка ведет на эту же страницу. Ссылка содержит строку запроса, содержащую теги `<script>`. В реальной XSS-атаке ссылка на эту страницу может появиться на другом веб-сайте, в электронном письме или другом типе сообщения.

2. Страница PHP проверяет суперглобальный массив `$_GET`, чтобы увидеть, содержит ли строка запроса элемент с именем `msg`:

- если это так, то соответствующее значение присваивается переменной с именем `$message`;
- если этого не происходит, в `$message` пишется сообщение, говорящее пользователю перейти по ссылке.

3. Значение из `$message` отображается на странице.

Когда вы нажмете на ссылку в верхней части страницы, скрипт выполнится, поскольку значение из строки запроса не было экранировано.

ЭКРАНИРОВАНИЕ ЗАРЕЗЕРВИРОВАННЫХ СИМВОЛОВ HTML

Встроенная в PHP функция `htmlspecialchars()` заменяет зарезервированные символы HTML соответствующими сущностями, так что эти символы просто выводятся на экран и не могут быть выполнены как код.

У функции `htmlspecialchars()` есть четыре параметра. Первый обязателен, остальные нет:

- *`$text`* — это текст, который вы хотите экранировать;
- *`$flag`* — это параметр для управления тем, какие символы кодируются (некоторые возможные значения приведены в таблице ниже);
- *`$encoding`* — это кодировка символов в параметре `$text` (если не указано, по умолчанию используется UTF-8);
- *`$double_encode`*. Поскольку HTML-сущности начинаются с амперсанда, то в случае, если кодируемая строка уже содержит сущность, амперсанд в ней будет закодирован, таким образом сущность будет закодирована дважды и отобразится на странице в виде сущности, а не в виде символа. Использование значения `false` для этого параметра указывает интерпретатору PHP не кодировать сущности в строке.

Если экранируемая строка состоит из символов, допустимых для используемой кодировки, то функция вернет строку, в которой зарезервированные символы заменены HTML-сущностями.

Если строка содержит недопустимые для выбранной кодировки символы, она возвращает пустую строку (если не используется флаг `ENT_SUBSTITUTE`, как описано в таблице ниже).

Поскольку `htmlspecialchars()` — довольно длинное имя функции, которая к тому же поддерживает четыре параметра, некоторые программисты создают свои функции с более коротким именем, чтобы экранировать значения и возвращать закодированную версию (как показано на странице справа).

```
htmlspecialchars($text[, $flag][, $encoding][, $double_encode]);
```

ФЛАГ	ОПИСАНИЕ
`ENT_COMPAT`	Преобразует двойные кавычки, не трогает одинарные кавычки (это значение по умолчанию, если флаг не указан)
`ENT_QUOTES`	Преобразует двойные и одинарные кавычки
`ENT_NOQUOTES`	Не преобразует двойные или одинарные кавычки
`ENT_SUBSTITUTE`	Чтобы не возвращать пустую строку, недопустимые символы будут заменены знаком вопроса в черном ромбе  (в UTF-8 это U+FFFD, в любой другой кодировке это &#xFFFD)
`ENT_HTML401`	Обрабатывает код как HTML 4.01
`ENT_HTML5`	Обрабатывает код как HTML 5
`ENT_XHTML`	Обрабатывает код как XHTML

Чтобы указать несколько флагов, разделите каждый из них вертикальной чертой, например `ENT_QUOTES|ENT_HTML5`.

ЭКРАНИРОВАНИЕ КОНТЕНТА

PHP section_b/c06/xss-2.php

```
<a class="badlink" href="xss-2.php?msg=<script
src=js/bad.js></script>">ESCAPING MARKUP</a>

<?php
$message = $_GET['msg'] ?? 'Click the link above';
?> ...
<h1>XSS Example</h1>
(1) <p><?= htmlspecialchars($message) ?></p>
```

PHP section_b/c06/xss-3.php

```
<a class="badlink" href="xss-3.php?msg=<script
src=js/bad.js></script>">ESCAPING MARKUP</a>

<?php
function html_escape(string $string): string
{
(2)     return htmlspecialchars($string,
            ENT_QUOTES|ENT_HTML5, 'UTF-8', true);
}
$message = $_GET['msg'] ?? 'Click the link above';
?> ...
<h1>XSS Example</h1>
(3) <p><?= html_escape($message) ?></p>
```

РЕЗУЛЬТАТ



1. В первом примере есть только одно изменение по сравнению с предыдущим. При выводе переменной `$message` используется функция PHP `htmlspecialchars()` для замены зарезервированных символов HTML соответствующими сущностями. Поэтому при нажатии на ссылку HTML-код для тегов `<script>` будет отображаться на экране, а не исполняться браузером.
2. Вторая версия того же примера добавляет пользовательскую функцию `html_escape()`. Она принимает строку в качестве аргумента и возвращает эту строку со всеми зарезервированными символами, замененными сущностями. При вызове `htmlspecialchars()` подставляются значения для всех четырех параметров.
3. Функция `html_escape()` вызывается для вывода сообщения, полученного из строки запроса.

Результат обоих примеров выглядит совершенно одинаково.

Примечание. В загружаемом для этой главы коде определение функции `html_escape()` также находится во включаемом файле `functions.php`.

Упражнение. В шаге 2 заменить определение функции конструкцией `include`, которая подключает файл `functions.php`.

КАК ДАННЫЕ ФОРМЫ ОТПРАВЛЯЮТСЯ НА СЕРВЕР

Формы позволяют посетителям вводить текст и выбирать параметры. С помощью каждого элемента формы браузер может отправить на сервер пару имя=значение при запросе страницы.

HTML-тегу `<form>` требуются два атрибута:

- значение атрибута `action` — это URL, на который должны быть отправлены данные формы;
- значение атрибута `method` указывает, какой метод использовать.

Атрибут `method` должен содержать одно из двух значений:

- `GET` отправляет данные формы с помощью HTTP GET в строке запроса, добавленной в конец URL-адреса;
- `POST` отправляет данные с использованием HTTP POST в теле запроса, отправляемого из браузера на сервер.

АДРЕС ДЛЯ ОТПРАВКИ ДАННЫХ — МЕТОД HTTP ДЛЯ ОТПРАВКИ ДАННЫХ

```
<form action="join.php" method="POST">
   <p>Email: <input type="email" name="email"></p>
   <p>Age:   <input type="number" name="age"></p>
   <p><input type="checkbox" name="terms" value="true">
      I agree to the terms and conditions.</p>
   <input type="submit" value="Save">
</form>
```

Когда посетитель отправляет форму, браузер запрашивает страницу, указанную в атрибуте `action`.

Значением атрибута `action` может быть относительный путь от страницы, создающей форму, до страницы, обрабатывающей форму, или это может быть полный URL. Обычно форма отправляется на ту же страницу, которая использовалась для отображения формы. В этом случае значение атрибута `action` можно оставить пустым.

Приведенная выше форма отправляется через `HTTP POST`, поэтому браузер добавит имена и значения элементов управления формой в тело `HTTP-запроса после заголовков`. Данные формы отправляются вместе с запросом к файлу `join.php`. Для каждого элемента формы отправляется:

- **имя** — это значение атрибута `name` элемента управления формой;
- **значение** — это текст, введенный пользователем, или значение выбранного им элемента.

Приведенные ниже элементы управления HTML-формой относятся к одной из двух категорий: поля ввода, позволяющие посетителям вводить текст, и элементы выбора, позволяющие посетителям выбирать один или несколько из предложенных вариантов.

Если посетитель вводит текст, то имя фрагмента данных берется из атрибута `name`, а значение — из введенного им текста. Если пользователь не вводит никакого текста для этого элемента управления формой, имя все равно отправляется на сервер, а значение представляет собой пустую строку.

Если посетитель отмечает один из вариантов выбора, то имя берется из атрибута `name`, а значение — из атрибута `value` для выбранного параметра. Если пользователь не выбрал какой-либо параметр, браузер не отправляет на сервер никаких данных для этого элемента управления формой (или элемент по умолчанию, если он есть).

ТЕКСТОВЫЙ ВВОД	ПРИМЕР	НАЗНАЧЕНИЕ
Ввод текста	`<input type="text" name="username">`	Ввести одну строку текста
Ввод числа	`<input type="number" name="age">`	Ввести число
Ввод e-mail	`<input type="email" name="email">`	Ввести e-mail
Пароль	`<input type="password" name="password">`	Ввести пароль
Текстовая область	`<textarea name="bio"></textarea>`	Ввести многострочный текст

ВЫБОР ВАРИАНТА	ПРИМЕР	НАЗНАЧЕНИЕ
Переключатель	`<input type="radio" name="rating" value="good">` `<input type="radio" name="rating" value="bad">`	Выбрать один из нескольких вариантов

Выпадающий список	`<select name="preferences">`
	`<option value="email">Email</option>`
	`<option value="phone">Phone</option>`
	`</select>`
		Выбрать один из нескольких вариантов

Флажок	`<input type="checkbox" name="terms" value="true">`	Выбрать один вариант

Чтобы продемонстрировать валидацию на стороне сервера, в этой книге данные проверяются только на сервере. Однако реальные сайты часто используют валидацию перед отправкой на сервер, прямо в браузере, используя JavaScript и атрибуты полей ввода, например для чисел или адресов электронной почты. Но данные все равно необходимо проверить на сервере еще раз, поскольку проверку в браузере можно обойти.

Примечание. Когда интерпретатор PHP добавляет данные из браузера в суперглобальный массив, они всегда имеют строковый тип, даже если элемент является числом или логическим значением.

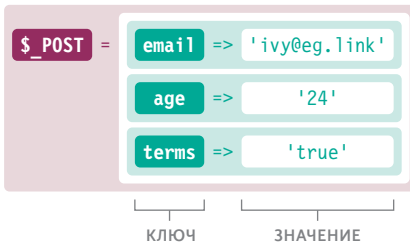
В следующей главе вы познакомитесь с элементом управления загрузкой файлов, используемым для отправки файлов на сервер.

ПОЛУЧЕНИЕ ДАННЫХ ИЗ ФОРМЫ

Когда интерпретатор PHP получает данные, отправленные через HTTP POST, они добавляются в суперглобальный массив $_POST.

При отправке посетителем формы через HTTP POST интерпретатор PHP добавляет данные формы (отправленные в теле запроса HTTP) в суперглобальный массив $_POST, где:

- **ключ** — это имя элемента управления формой;
- **значение** — это значение, введенное или выбранное пользователем.



Если форма была отправлена с использованием HTTP GET, интерпретатор PHP получает данные формы из строки запроса и добавляет их в суперглобальный массив $_GET[32].

Код в файле PHP может получить доступ к значениям в суперглобальном массиве $_POST таким же образом, как и к значениям из любого ассоциативного массива. Если элемент управления формой представляет собой поле для ввода текста, для него всегда будет указано значение (если только этому полю не был поставлен атрибут `disabled`):

```
$email = $_POST['email'];
```

ПЕРЕМЕННАЯ — КЛЮЧ

Если элемент управления формой представляет собой выбор значения с помощью флажка или переключателя, то имя и значение отправляются на сервер только в том случае, если посетитель выбирает опцию. Поэтому для таких элементов при получении значений из суперглобального массива $_POST используется оператор объединения с `null` (таким же образом, как он использовался ранее для получения значений из строки запроса).

```
$age = $_POST['age'] ?? false;
```

ПЕРЕМЕННАЯ — КЛЮЧ — ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ

В следующем примере можно увидеть содержимое суперглобальных массивов, когда на страницах применяются формы. Функция `var_dump()` используется для отображения содержимого суперглобального массива. С ее помощью можно увидеть, какие элементы добавлены в массив, а также что все данные этих суперглобальных массивов имеют строковый тип данных, даже если это число или логическое значение.

Важно попробовать этот пример самостоятельно и посмотреть, как изменяются данные в суперглобальном массиве, когда:

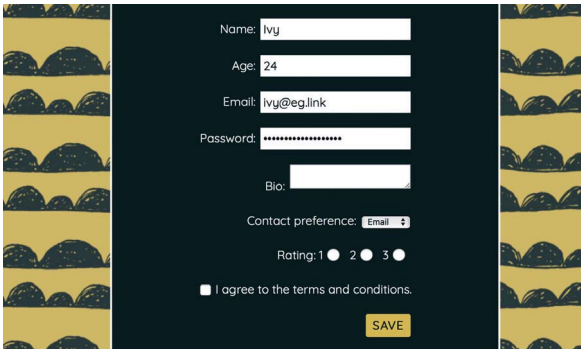
- страница загружается впервые, перед отправкой формы;
- форма отправляется без заполнения каких-либо данных;
- все поля формы заполнены.

ПРИМЕР ПОЛУЧЕНИЯ ДАННЫХ ИЗ ФОРМЫ

PHP section_b/c06/collecting-form-data.php

```
<form action="collecting-form-data.php" method="POST">
  <p>Name:     <input type="text" name="name"></p>
  <p>Age:      <input type="text" name="age"></p>
(1)<p>Email:    <input type="text" name="email"></p>
  <p>Password: <input type="password" name="pwd"></p>
  <p>Bio:      <textarea name="bio"></textarea></p>
  <p>Contact preference:
    <select name="preferences">
      <option value="email">Email</option>
      <option value="phone">Phone</option>
    </select></p>
(2)<p>Rating:
   1 <input type="radio" name="rating" value="1"> 
   2 <input type="radio" name="rating" value="2"> 
   3 <input type="radio" name="rating" value="3"></p>
  <p><input type="checkbox" name="terms" value="true">
    I agree to the terms and conditions.</p>
  <p><input type="submit" value="Save"></p>
</form>
(3)<pre><?php var_dump($_POST); ?></pre>
```

РЕЗУЛЬТАТ



Упражнение. Измените значение в атрибуте `method` тега `<form>` на GET, и данные будут отправлены через HTTP GET. Затем, в шаге 3, отобразите содержимое суперглобального массива `$_GET`.

1. Пять текстовых элементов управления запрашивают у пользователя имя, возраст, адрес электронной почты, пароль и биографию.
2. Три различных элемента выбора предлагают посетителю отметить нужный пункт.
3. Содержимое суперглобального массива `$_POST` выводится с помощью функции `var_dump()`.

Когда страница открывается первый раз и форма еще не отправлена, суперглобальный массив `$_POST` будет пустым.

Если форма отправлена без ввода каких-либо данных, суперглобальный массив `$_POST` содержит элементы для каждого из вводимых текстовых полей. Значением каждого элемента будет пустая строка. Элемент выпадающего списка отправляется на сервер со значением по умолчанию, показанным при загрузке страницы.

Но имена и значения переключателей и флажков не отправляются на сервер.

Если все элементы управления формы заполнены, суперглобальный массив `$_POST` будет содержать данные для каждого элемента управления формы. Каждое значение, отправляемое на сервер, представляет собой строку.

КАК УЗНАТЬ, ЧТО ФОРМА БЫЛА ОТПРАВЛЕНА

Прежде чем обрабатывать данные, переданные через форму, необходимо убедиться в том, что она была отправлена. Существуют несколько способов проверить это. Они зависят от метода отправки формы — через HTTP POST или HTTP GET.

HTTP POST

В суперглобальном массиве `$_SERVER` находится ключ с именем `REQUEST_METHOD`. Его значением является метод HTTP, который был использован для запроса страницы. Когда форма отправляется с использованием HTTP POST, ключ содержит значение `POST`.

Чтобы выяснить, была ли форма отправлена с использованием HTTP POST, в конструкции `if` пишется условие, которое проверяет, содержит ли ключ `REQUEST_METHOD` значение `POST`. Соответственно, код для обработки формы пишется внутри фигурных скобок.

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Код для получения и обработки данных формы
}
```

HTTP GET

Когда пользователь нажимает на ссылку или вводит URL-адрес в адресную строку браузера, запрос всегда отправляется через HTTP GET. Следовательно, вы не можете отличить такие запросы от отправки формы методом GET, используя массив `$_SERVER`. Вместо этого вы можете использовать:

- либо скрытое поле ввода (`<input type=»hidden»>`);
- либо имя и значение кнопки отправки формы.

Когда форма будет отправлена, имя и значение из скрытого поля ввода или кнопки будут добавлены в строку запроса и попадут в суперглобальный массив `$_GET`.

С помощью условия в конструкции `if` можно после этого проверить, содержит ли суперглобальный массив `$_GET` искомый элемент. Если да, то можно выполнять код для получения и обработки данных формы.

```
$submitted = $_GET['submitted'] ?? '';
if ($submitted === 'true') {
    // Код для получения и обработки данных формы
}
```

ПРОВЕРКА ОТПРАВКИ ФОРМЫ

PHP section_b/c06/check-for-http-post.php

```php
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $term = $_POST['term'];
    echo 'You searched for ' . htmlspecialchars($term);
} else { ?>
    <form action="check-for-http-post.php" method="POST">
      Search for: <input type="text" name="term">
      <input type="submit" value="search">
    </form>
<?php } ?>
```

PHP section_b/c06/check-for-http-get.php

```php
<?php
$submitted = $_GET['sent'] ?? '';
if ($submitted === 'search') {
    $term = $_GET['term'] ?? '';
    echo 'You searched for ' . htmlspecialchars($term);
} else { ?>
    <form action="check-for-http-get.php" method="GET">
      Search for: <input type="search" name="term">
      <input type="submit" name="sent" value="search">
    </form>
<?php } ?>
```

РЕЗУЛЬТАТ



1. С помощью условия в конструкции `if` проверяется, содержит ли элемент `REQUEST_METHOD` суперглобального массива `$_SERVER` значение `POST`.
2. Если да, значит, форма поиска была отправлена через HTTP POST и будет отображен поисковый запрос.
3. В противном случае вывод пропускается и будет отображена форма.

В следующем примере имя кнопки отправки — `sent`, а ее значение — `search`. Если форма была отправлена, они добавляются в суперглобальный массив `$_GET`.

4. С помощью оператора объединения с `null` проверяется, содержит ли суперглобальный массив `$_GET` элемент с ключом `sent`. Если да, то его значение присваивается переменной `$submitted`. Если нет, то будет присвоена пустая строка.
5. В конструкции `if` проверяется, содержит ли переменная `$submitted` значение `search`. Если да, то форма была отправлена через HTTP GET и отображается поисковый запрос.
6. В противном случае выводится форма.

Упражнение. Использовать скрытое поле формы, чтобы указать, что форма была отправлена.

ВАЛИДАЦИЯ ЦИФР

При получении данных из формы их необходимо проверить, чтобы убедиться, что все требуемые значения были введены и соответствуют требуемому формату. Это предотвращает появление ошибок при дальнейшем выполнении кода из-за неправильных данных.

Чтобы проверить, является ли значение числом, используйте встроенную в PHP функцию `is_numeric()`. Или, если вам нужно проверить, что число находится в пределах указанного диапазона разрешенных значений, можно создать пользовательскую функцию для выполнения этой задачи. Ниже представлена функция, использующая операторы сравнения для проверки, находится ли число в пределах минимального и максимального диапазона допустимых значений. Функция содержит три параметра:

- `$number` — это значение, которое необходимо проверить;
- `$min` — это минимально допустимое число;
- `$max` — это максимально допустимое число.

В коде функции условие содержит два выражения, проверяющие, что значение:

- больше или равно минимальному числу;
- меньше или равно максимальному числу.

Если оба выражения принимают значение `true`, функция возвращает значение `true`. Если любое из выражений приводит к значению `false`, функция возвращает значение `false`. При получении числа из формы с помощью этой функции код может проверить, допустимо ли введенное значение[33].

```
function is_number($number, int $min = 0, int $max = 100): bool
{
    return ($number >= $min and $number <= $max);
}
```

ЗНАЧЕНИЕ >= МИНИМУМА? ЗНАЧЕНИЕ <= МАКСИМУМА?

Если введенные данные не прошли проверку, то посетителю обычно снова показывают форму для повторной попытки ввода. В таких случаях введенное ранее значение можно отобразить в поле ввода, записав его в атрибуте `value` тега `<input>`. Функция `htmlspecialchars()` используется при отображении значения для предотвращения XSS-атаки.

Поскольку введенное пользователем значение присваивается переменной только в том случае, если форма была отправлена, переменную `$age` необходимо объявить в начале страницы и присвоить ей начальное значение пустой строки. Если же этого не сделать, то попытка отобразить ее в атрибуте `value` при первом показе формы приведет к появлению ошибки `Undefined variable` в поле ввода.

```
<input type="text" name="age" value="<?= htmlspecialchars($age) ?>">
```

ПРОВЕРКА, ЯВЛЯЕТСЯ ЛИ ВВЕДЕННОЕ ЗНАЧЕНИЕ ЧИСЛОМ

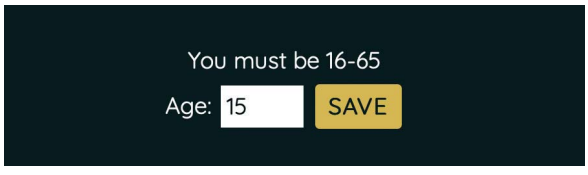
PHP section_b/c06/validate-number-range.php

```php
<?php
declare(strict_types = 1);
$age     = '';
$message = '';

function is_number($number, int $min = 0, int $max = 100): bool
{
    return ($number >= $min and $number <= $max);
}

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $age   = $_POST['age'];
    $valid = is_number($age, 16, 65);
    if ($valid) {
        $message = 'Age is valid';
    } else {
        $message = 'You must be 16-65';
    }
}
?> ...
<?= $message ?>
<form action="validate-number-range.php" method="POST">
  Age: <input type="text" name="age" size="4"
         value="<?= htmlspecialchars($age) ?>">
  <input type="submit" value="Save">
</form>
```

РЕЗУЛЬТАТ



1. Две переменные, `$age` и `$message`, инициализируются значениями, представляющими собой пустые строки.
2. Определение функции `is_number()` (см. страницу слева).
3. Проверка, была ли форма отправлена. Если была, то:
4. Возраст берется из суперглобального массива `$_POST`. Данные поступают из текстового поля, поэтому при отправке формы в него всегда будет передаваться значение.
5. Вызывается функция `is_number()`. Значение, введенное пользователем, — это первый аргумент, а числа 16 и 65 — минимально и максимально допустимые числа. Возвращаемое функцией логическое значение пишется в переменную `$valid`.
6. С помощью конструкции `if` проверяется, содержит ли переменная `$valid` значение `true`. Если да, то в переменную `$message` будет записано сообщение, что возраст подходящий.
7. В противном случае в `$message` запишется сообщение об ошибке.
8. Вывод сообщения.
9. Число, введенное пользователем (или начальное значение из шага 1), отображается в поле ввода с помощью функции `htmlspecialchars()`.

ВАЛИДАЦИЯ ДЛИНЫ ТЕКСТА

Сайты часто ограничивают количество символов в таких элементах, как имена пользователей, сообщения, названия статей и имена файлов. Для проверки длины любой строки, получаемой сайтом, вам понадобится только одна функция.

Чтобы проверить, содержит ли текст, предоставляемый пользователями, минимально и максимально допустимое количество символов:

- примените встроенную в PHP функцию `mb_strlen()` для подсчета количества символов в строке и сохраните результат в переменной.
- затем используйте условие с двумя выражениями, которое проверяет, находится ли количество символов в допустимом диапазоне — таким же образом, как на предыдущей странице проверялось, входит ли число в допустимый диапазон.

Если количество символов входит в него, функция возвращает значение `true`; если нет, она возвращает значение `false`.

Если код для валидации данных поместить в функцию, ее можно будет использовать для валидации сразу нескольких значений. Это избавляет от необходимости писать повторяющийся код для выполнения одной и той же задачи.

Поскольку приведенная ниже функция (как и предыдущая) использует параметры, то при каждом вызове этой функции можно указывать разные минимальные и максимальные значения.

Когда на нескольких разных страницах выполняются одни и те же задачи валидации, необходимо поместить определения функций в подключаемый файл. Затем вы можете просто подключить этот файл, вместо того чтобы дублировать одни и те же определения функций на каждой странице. Загружаемый для этой главы код содержит подключаемый файл с именем `validate.php`. Он содержит три определения функций из этой главы.

```
function is_text($text, int $min = 0, int $max = 100): bool
{
    $length = mb_strlen($text);
    return ($length >= $min and $length <= $max);
}
```

ЗНАЧЕНИЕ >= МИНИМУМА?

ЗНАЧЕНИЕ <= МАКСИМУМА?

ПРОВЕРКА ДЛИНЫ ТЕКСТА

PHP section_b/c06/validate-text-length.php

```php
<?php
declare(strict_types = 1);
$username = '';
$message  = '';

function is_text($text, int $min = 0, int $max = 1000): bool
{
    $length = mb_strlen($text);
    return ($length >= $min and $length <= $max);
}
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $valid    = is_text($username, 3, 18);
    if ($valid) {
        $message = 'Username is valid';
    } else {
        $message = 'Username must be 3-18 characters';
    }
}
?> ...
<?= $message ?>
<form action="validate-text-length.php" method="POST">
  Username: <input type="text" name="username"
    value="<?= htmlspecialchars($username) ?>">
  <input type="submit" value="Save">
</form>
```

РЕЗУЛЬТАТ



1. Инициализация переменных `$username` и `$message`.
2. Объявление пользовательской функций с именем `is_text()` (показана на странице слева).
3. Проверка, была ли форма отправлена. Если была, то:

- получение введенного текста из суперглобального массива `$_POST`;
- функция `is_text()` вызывается для проверки длины текста, которая должна составлять от 3 до 18 символов. Значение, которое возвращает функция, пишется в переменную `$valid`;
- условие в конструкции `if` проверяет, истинно ли значение в `$valid`. Если это так, то в `$message` записывается сообщение о том, что имя пользователя прошло проверку.

4. В противном случае в `$message` пишется сообщение, что имя должно быть длиной от 3 до 18 символов.
5. Отображается значение переменной `$message`.
6. Значение в `$username` отображается в поле ввода. Это либо значение, отправленное пользователем, либо пустая строка, присвоенная при инициализации переменной в шаге 1.

ВАЛИДАЦИЯ ДАННЫХ С ПОМОЩЬЮ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Регулярные выражения можно использовать для проверки введенного пользователем значения на соответствие определенному шаблону.

Регулярные выражения могут описывать определенный шаблон, к примеру используемый в номерах кредитных карт, почтовых индексах или телефонных номерах. Приведенная ниже функция применяет регулярные выражения для проверки надежности введенного пароля.

Она принимает пароль в качестве параметра, а затем проверяет, чтобы он состоял из 8 или более символов. Затем функция использует регулярные выражения, чтобы проверить, содержит ли пароль:

- символы верхнего регистра;
- символы нижнего регистра;
- цифры.

Каждая проверка разделяется с помощью оператора `and`. Если все условия возвращают `true`, функция возвращает `true`; в противном случае она возвращает `false`. Теоретически синтаксис регулярных выражений позволяет написать один шаблон для всех проверок разом, но его было бы гораздо сложнее прочитать (и написать).

Функция содержит условие с четырьмя выражениями.

Сначала функция `mb_strlen()` проверяет, состоит ли значение из 8 или более символов. Затем функция PHP `preg_match()` применяется трижды, чтобы проверить, найден ли в пароле символ, соответствующий шаблону, описанному в регулярном выражении.

Если все выражения возвращают `true`, то выполняется следующий блок кода, в котором функция возвращает значение `true`, поскольку пароль соответствует требованиям. Если же этого не произошло, то функция продолжит выполнение и вернет значение `false`.

```
function is_password(string $password):
bool
{
    if (
        mb_strlen($password) >= 8
        and preg_match('/[A-Z]/', $password)
        and preg_match('/[a-z]/', $password)
        and preg_match('/[0-9]/', $password)
    ) {
        return true; // Все тесты пройдены
    }
    return false;   // Валидация не пройдена
}
```

Примечание. Хотя браузеры скрывают пароль при его вводе, но на сервер вводимые пользователем данные отправляются в виде обычного текста в теле запроса HTTP. Поэтому все персональные данные должны отправляться по протоколу HTTPS.

ПРОВЕРКА НАДЕЖНОСТИ ПАРОЛЯ

PHP section_b/c06/validate-password.php

```php
<?php
declare(strict_types = 1);
$password = '';
$message  = '';
function is_password(string $password): bool
{
    if (
        mb_strlen($password) >= 8
        and preg_match('/[A-Z]/', $password)
        and preg_match('/[a-z]/', $password)
        and preg_match('/[0-9]/', $password)
    ) {
        return true;  // Passed all tests
    }
    return false;     // Invalid
}
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $password = $_POST['password'];
    $valid    = is_password($password);
    $message  = $valid ? 'Password is valid' :
        'Password not strong enough';
}
?> ...
<?= $message ?>
<form action="validate-password.php" method="POST">
  Password: <input type="password" name="password">
  <input type="submit" value="Save">
</form>
```

РЕЗУЛЬТАТ



1. Инициализация переменных `$password` и `$message`.
2. Функция `is_password()` определяется с одним параметром: паролем для проверки.
3. В конструкции `if` используется четыре выражения; каждое из них дает в результате `true` или `false`. Они разделены операторами `and`, поэтому все выражение вернет `true`, если *все* выражения дают результат `true`.
4. Блок кода возвращает значение `true`, и функция прекращает выполнение.
5. В противном случае, если какое-либо условие не выполнено, функция возвращает значение `false`.
6. Если форма была отправлена, выполняется следующий блок кода.
7. Пароль читается из суперглобального файла `$_POST`.
8. Функция `is_password()` вызывается для проверки пароля пользователя. Результат сохраняется в переменной с именем `$valid`.
9. С помощью тернарного оператора проверяется, содержит ли переменная `$valid` значение `true`. Если проверка проходит успешно, то в `$message` пишется сообщение об этом. В противном случае пишется сообщение об ошибке.
10. Отображается значение переменной `$message`.

ВЫПАДАЮЩИЕ СПИСКИ И ПЕРЕКЛЮЧАТЕЛИ

Выпадающие списки и переключатели позволяют пользователям выбирать один вариант из нескольких. Браузер отправляет на сервер имя и значение переключателя только в том случае, если было выбрано какое-то значение. Валидация значений производится путем проверки введенного значения на вхождение в заранее подготовленный список.

Когда в форме необходимо вывести выпадающий список или переключатель, вы можете использовать для этого индексированный массив, содержащий все доступные пользователю варианты выбора.

Приведенный ниже массив содержит варианты для выставления рейтинга от 1 до 5 звезд.

Этот массив можно использовать:

- как для вывода вариантов выбора в списках или переключателях,
- так и для проверки, что пользователь выбрал один из этих вариантов.

```
$star_ratings = [1, 2, 3, 4, 5,];
```

Чтобы проверить правильность варианта выбора пользователя, используется встроенная в PHP функция `in_array()`. Если введенное значение найдено в массиве параметров, функция `in_array()` возвращает значение `true`. Если нет, то она возвращает значение `false`.

```
$valid = in_array($stars, $star_ratings);
```

ВВЕДЕННОЕ ЗНАЧЕНИЕ — ДОПУСТИМЫЕ ВАРИАНТЫ

Для создания выпадающего списка или переключателя вы можете перебрать в цикле все доступные варианты и добавить в форму элемент выбора для каждого из них. Если форма снова будет показана пользователю, выбранный параметр может быть выделен с помощью тернарного оператора. Условие в тернарном операторе проверяет, соответствует ли значение в переменной `$stars` текущему значению в цикле. Если соответствует, то добавляется атрибут `checked`. Если нет, то вместо этого выводится пустая строка.

```
<?php foreach ($option as $star_ratings) { ?>
  <?= $option ?>
  <input type="radio" name="stars" value="<?= $option ?>"
     <?= ($stars == $option) ? 'checked' : '' ?>>
<?php } ?>
```

Примечание. Этот пример предполагает, что переменная `$stars` была инициализирована (см. шаг 1 на правой странице).

ВАЛИДАЦИЯ ПЕРЕКЛЮЧАТЕЛЕЙ

PHP section_b/c06/validate-options.php

```
<?php
(1) $stars   = '';
(1) $message = '';
(2) $star_ratings = [1, 2, 3, 4, 5,];

(3) if ($_SERVER['REQUEST_METHOD'] == 'POST') {
(4)     $stars   = $_POST['stars'] ?? '';
(5)     $valid   = in_array($stars, $star_ratings);
(6)     $message = $valid ? 'Thank you' : 'Select an option';
}
?> ...
(7) <?= $message ?>
<form action="validate-options.php" method="POST">
  Star rating:
(8)  <?php foreach ($star_ratings as $option) { ?>
(9)    <?= $option ?> <input type="radio" name="stars"
           value="<?= $option ?>"
(10)       <?= ($stars == $option) ? 'checked' : '' ?>>
  <?php } ?>
  <input type="submit" value="Save">
</form>
```

РЕЗУЛЬТАТ



1. Инициализация переменных `$stars` и `$message`.
2. Переменная `$star_ratings` содержит индексированный массив значений, который будет использоваться для создания набора переключателей.
3. Проверка, была ли форма отправлена.
4. Если да, то значение выбранного варианта берется из суперглобального массива `$_POST`.
5. Функция PHP `in_array()` проверяет, соответствует ли переданное браузером значение одному из допустимых вариантов.
6. С помощью тернарного оператора создается сообщение о том, прошло введенное значение проверку или нет.
7. Отображается значение в `$message`.
8. Цикл `foreach` создает варианты выбора в HTML-форме, перебирая элементы массива `$star_ratings`.
9. Каждый элемент массива выводится на страницу, а за ним следует тег-переключатель, в атрибут `value` которого снова выводится тот же элемент массива.
10. Тернарный оператор проверяет, выбран ли какой-либо вариант. Если выбран, то добавляется атрибут `checked`.

КАК ОПРЕДЕЛИТЬ, УСТАНОВЛЕН ЛИ ФЛАЖОК

Флажок имеет только два варианта выбора, он либо установлен, либо нет. Но при этом имя и значение флажка отправляются на сервер только тогда, когда флажок установлен.

Поэтому определить состояние флажка можно в два шага.

- Сначала используйте функцию PHP `isset()`, чтобы проверить, есть ли значение для флажка в суперглобальном массиве.
- Если да, то проверьте, было ли предоставленное значение именно тем, которое ожидалась.

Обе эти две проверки могут быть выполнены в условии тернарного оператора. Если обе проверки приводят к значениям `true`, это означает, что пользователь установил флажок и вы можете присвоить логическое значение `true`.

В коде ниже, если обе проверки приведут к результату `true`, `$terms` сохранит значение `true`, в противном случае она сохранит значение `false`[34].

```
$terms = (isset($_POST['terms']) and $_POST['terms'] == true) ? true : false;
```

ЕСЛИ ЗНАЧЕНИЕ ЕСТЬ В СУПЕРГЛОБАЛЬНОМ МАССИВЕ

И УКАЗАННОЕ ЗНАЧЕНИЕ ПРАВИЛЬНОЕ

Если форма снова будет показана пользователю, то, чтобы поставить флажок, вам просто нужно сравнить значение переменной с `true`[35]. Если результат положительный, то к тегу добавляется атрибут `checked`. В противном случае добавляется пустая строка.

```
<input type="checkbox" name="terms" value="true"
  <?= $terms ? 'checked' : '' ?>>
```

ЕСЛИ ФЛАЖОК БЫЛ УСТАНОВЛЕН

ДОБАВИТЬ АТРИБУТ CHECKED

ИНАЧЕ ДОБАВИТЬ ПУСТУЮ СТРОКУ

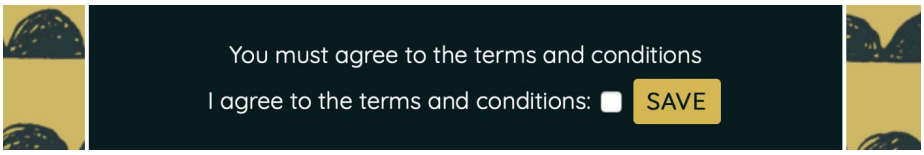
ВАЛИДАЦИЯ ФЛАЖКОВ

PHP section_b/c06/validate-checkbox.php

```php
<?php
$terms   = '';
$message = '';

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $terms   = (isset($_POST['terms']) and $_POST['terms'] == true) ? true : false;
    $message = $terms ? 'Thank you' : 'You must agree to the terms and conditions';
}
?> ...
<?= $message ?>
<form action="validate-checkbox.php" method="POST">
  I agree to the terms and conditions: <input type="checkbox" name="terms" value="true"
    <?= $terms ? 'checked' : '' ?>>
  <input type="submit" value="Save">
</form>
```

РЕЗУЛЬТАТ



1. Инициализация переменных `$terms` и `$message`.
2. Если форма была отправлена…
3. Два выражения используются в условии тернарного оператора, чтобы определить, был ли установлен флажок. Конструкция `isset()` проверяет, был ли отправлен флажок. Если да, то второе выражение проверяет, истинно ли его значение. Если оба выражения возвращают `true`, переменной `$terms` присваивается значение `true`; если нет, ей присваивается значение `false`.
4. Если `$terms` равно `true`, то в `$message` пишется текст `Thank you`. В противном случае записывается сообщение, в котором пользователю предлагается согласиться с условиями использования.
5. Сообщение отображается на странице.
6. Тернарный оператор используется для проверки, содержит ли переменная `$terms` значение `true` (указывающее, что флажок был отмечен). Если это так, то к флажку добавляется атрибут `checked`. Если нет, то вместо этого добавляется пустая строка.

ВАЛИДАЦИЯ НЕСКОЛЬКИХ ЗНАЧЕНИЙ

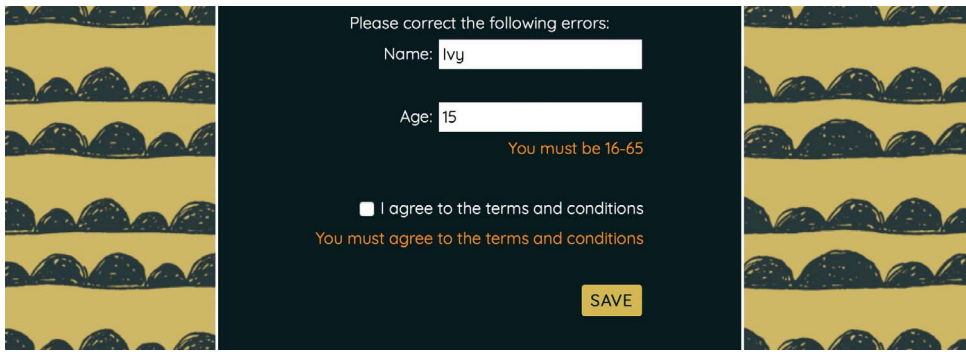
Часто программе требуется проверить не один фрагмент данных, а несколько. Например, в большинстве форм пользователи вводят сразу несколько значений.

Взгляните на представленную ниже форму. В ней посетителям предлагается ввести следующие значения (используя для этого три разных типа данных):

- **Name** (имя) — строка длиной от 2 до 10 символов;
- **Age** (возраст) — целое число от 16 до 65;
- **I agree to the terms and conditions** (согласие с условиями использования) — логическое значение `true` или `false`.

Если при отправке пользователем формы какие-либо данные неверны, код на странице должен:

- не обрабатывать данные;
- создать сообщения об ошибках, помогающие посетителю исправить каждую из проблем;
- снова отобразить в форме значения, введенные пользователем.



Помимо самой формы, на этой странице могут выводиться сообщения об ошибках, а также значения, которые были введены пользователем.

Чтобы это реализовать, нам понадобится два массива:

- один для введенных пользователем значений;
- второй для сообщений об ошибках.

Все элементы этих массивов должны быть инициализированы заранее и содержать значения для каждого элемента формы, чтобы они могли быть отображены при первой загрузке страницы (до отправки формы). Если этого не сделать, то при попытке обратиться к несуществующему элементу массива интерпретатор PHP выдаст ошибку.

Исходно массив для сообщений об ошибках содержит пустые строки, поскольку при первой загрузке страницы, когда форма еще не отправлена, проверка данных не производится.

1. Если форма была отправлена, введенные пользователем данные добавляются в созданный ранее массив и перезаписывают исходно пустые значения.

МАССИВ ДЛЯ ХРАНЕНИЯ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ — ПОЛУЧЕНИЕ ЗНАЧЕНИЙ

```
$user['name']  = $_POST['name'];
$user['age']   = $_POST['age'];
$user['terms'] = (isset($_POST['terms']) and $_POST['terms'] == true) ? true : false;
```

2. Затем проверяется каждое введенное значение. Если оно не проходит проверку, то в соответствующем элементе массива `$errors` сохраняется сообщение об ошибке. Данные проверяются с помощью функций валидации, представленных в этой главе. Они возвращают результат `true`, если значение, указанное пользователем, допустимое, и `false`, если это не так.

Это означает, что функции валидации могут использоваться в качестве условия в тернарном операторе.

- Если введенное значение проходит проверку, то соответствующему элементу массива присваивается пустая строка;
- Если же не проходит, то в массиве сохраняется сообщение об ошибке, информирующее пользователя о причинах недействительности данных.

МАССИВ ДЛЯ СООБЩЕНИЙ ОБ ОШИБКАХ — ВАЛИДАЦИЯ ЗНАЧЕНИЙ ФОРМЫ — ПУСТАЯ СТРОКА — СООБЩЕНИЕ ОБ ОШИБКЕ

```
$errors['name']  = is_text($user['name'], 2, 20)   ? '' : 'Name must be 2-20 characters';
$errors['age']   = is_number($user['age'], 16, 65) ? '' : 'You must be 16-65';
$errors['terms'] = $user['terms']                  ? '' : 'You must agree to the terms';
```

3. Чтобы проверить наличие ошибок, встроенная в PHP функция `implode()` используется для объединения всех значений массива `$errors` в одну строку. Результат сохраняется в переменной с именем `$invalid`.

Если `$invalid` содержит пустую строку, все данные прошли проверку. Если нет, то произошла по крайней мере одна ошибка.

```
$invalid = implode($errors);
```

СТРОКА, СОДЕРЖАЩАЯ ВСЕ ОШИБКИ — МАССИВ С ОШИБКАМИ

4. Конструкция `if` используется для проверки наличия текста в `$invalid`. Если текст есть, значение будет рассматриваться как `true` и сообщения об ошибках будут отображаться вместе с формой.

Если ошибок не было, `$invalid` будет содержать пустую строку (обрабатываемую как `false`) и страница сможет обработать полученные данные.

```
if ($invalid) {
    // Показать сообщения об ошибках и не обрабатывать данные
} else {
    // Данные действительны, страница может обрабатывать их
}
```

ВАЛИДАЦИЯ ФОРМ

В этом примере показано, как проверить несколько элементов управления формой. Результат был показан на предыдущей странице.

1. Подключение файла `validate.php`. Он содержит определения для трех функций валидации, описанных в этой главе. Размещение их в подключаемом файле позволяет использовать эти функции на любой странице.

2. Переменная `$user` содержит массив записей для каждого элемента управления формой. Каждому элементу присваивается начальное значение для использования в форме при первой загрузке страницы.

3. Переменная `$errors` содержит массив с элементами для каждого проверяемого фрагмента данных.

4. Переменной `$message` присваивается пустая строка. Когда данные пройдут валидацию, она будет содержать сообщение об успешном выполнении или ошибке.

5. Проверка, была ли отправлена форма.

6. Если да, то введенные пользователем данные добавляются в массив `$user`, перезаписывая те значения, что были там изначально.

7. Введенное пользователем имя проверяется с помощью функции `is_text()`. Она возвращает значение `true`, если имя соответствует требованиям, и `false`, если нет. Если имя прошло проверку, в соответствующий элемент в массиве `$errors` (см. шаг 3) пишется пустая строка. Если нет, то пишется сообщение, помогающее пользователю это исправить.

8. Возраст пользователя проверяется с помощью функции `is_number()`. Она возвращает значение `true`, если введенный возраст соответствует требованиям, и `false`, если нет. Если возраст прошел проверку, то в соответствующий элемент массива `$errors` пишется пустая строка. Если нет, то пишется сообщение об ошибке.

9. Если пользователь установил флажок `terms`, то в соответствующий элемент массива `$errors` запишется пустая строка. Если нет, то запишется сообщение об ошибке, в котором говорится, что пользователю необходимо принять условия использования.

10. Все значения в массиве `$errors` объединяются в одну строку с помощью PHP- функции `implode()`. Результат сохраняется в переменной `$invalid`.

11. Условие в конструкции `if` проверяет, имеется ли непустое значение в переменной `$invalid`. Результат будет принимать значение `true` при наличии любого текста в переменной. Пустая строка обрабатывается с результатом `false`.

12. Если были ошибки, то в переменную `$message` пишется дополнительное сообщение, в котором пользователю предлагается исправить ошибки формы.

13. В противном случае в `$message` пишется сообщение о том, что данные прошли проверку. В этом случае код на странице может их обработать.

14. Обычно, когда все данные валидны, форма не нуждается в повторном отображении.

15. Вывод переменной `$message`.

16. Если пользователь отправил форму, значение, которое он ввел для своего имени, выводится в атрибуте `value` соответствующего поля ввода. При этом оно экранируется с помощью PHP-функции `htmlspecialchars()`.

17. Если форма не была отправлена, отобразится пустая строка, сохраненная в соответствующем ключе массива `$user` при его инициализации в шаге 2.

18. Отображается значение элемента из массива `$errors`, соответствующего этому элементу управления формой.

19. Если пользователь указал свой возраст, он отображается в атрибуте `value` соответствующего поля ввода. За этим следует соответствующее значение из массива `$errors`.

20. Если посетитель принял условия использования, к флажку добавляется атрибут `checked`. Затем выводится соответствующий элемент массива `$errors`.

PHP section_b/c06/validate-form.php

```php
<?php
declare(strict_types = 1);                              // Включение строгой типизации
require 'includes/validate.php';                        // Подключение функций валидации

$user = [
    'name'  => '',
    'age'   => '',
    'terms' => '',
];                                                      // Инициализация массива $user
$errors = [
    'name'  => '',
    'age'   => '',
    'terms' => '',
];                                                      // Инициализация массива ошибок
$message = '';                                          // Инициализация $message

if ($_SERVER['REQUEST_METHOD'] == 'POST') {             // Если форма отправлена
    $user['name']  = $_POST['name'];                    // Получение имени
    $user['age']   = $_POST['age'];                     // Получение возраста
                                                        // А затем правил и условий
    $user['terms'] = (isset($_POST['terms']) and $_POST['terms'] == true) ? true : false;

    $errors['name']  = is_text($user['name'], 2, 20)   ? '' : 'Must be 2-20 characters';
    $errors['age']   = is_number($user['age'], 16, 65) ? '' : 'You must be 16-65';
    $errors['terms'] = $user['terms']                  ? '' : 'You must agree to the
        terms and conditions';                          // Валидация данных

    $invalid = implode($errors);                        // Слияние ошибок в одну строку
    if ($invalid) {                                     // Если есть ошибки
        $message = 'Please correct the following errors:'; // Не обрабатывать
    } else {                                            // В противном случае
        $message = 'Your data was valid';               // Можно обрабатывать данные
    }
}
?> ...
<?= $message ?>
<form action="validate-form.php" method="POST">
  Name: <input type="text" name="name" value="<?= htmlspecialchars($user['name']) ?>">
  <span class="error"><?= $errors['name'] ?></span><br>
  Age:  <input type="text" name="age" value="<?= htmlspecialchars($user['age']) ?>">
  <span class="error"><?= $errors['age'] ?></span><br>
  <input type="checkbox" name="terms" value="true" <?= $user['terms'] ? 'checked' : '' ?>>
    I agree to the terms and conditions
  <span class="error"><?= $errors['terms'] ?></span><br>
  <input type="submit" value="Save">
</form>
```

ПОЛУЧЕНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ ФИЛЬТРАЦИИ

В PHP имеется две встроенные функции, позволяющие получить отправленные из браузера данные и сохранить их в переменные. Они называются функциями фильтрации, потому что могут применять различные фильтры к данным, отправленным браузером.

Функция `filter_input()` получает одно отправленное на сервер значение. Ей требуется два аргумента. Первый аргумент — это константа, обозначающая **источник данных** (input source). Константы пишутся без кавычек. Используются:

- `INPUT_GET` для получения данных, отправленных через HTTP GET;
- `INPUT_POST` для получения данных, отправленных через HTTP POST;
- `INPUT_SERVER` для получения данных, которые доступны в суперглобальном массиве `$_SERVER`.

Второй аргумент — это имя элемента формы, которое было отправлено на сервер в составе одной из пар «имя — значение». Оно должно быть заключено в кавычки. При использовании в таком варианте функция `filter_input()` возвращает:

- переданное значение, если оно было отправлено на сервер;
- `null`, если данные не были отправлены на сервер.

Помимо такого базового варианта использования вы также узнаете, как применять эту функцию с указанием фильтра в качестве третьего параметра.

```
$data = filter_input(INPUT_SOURCE, 'name');
```

ИСТОЧНИК ДАННЫХ — ИМЯ

Функция `filter_input_array()` собирает все отправленные на сервер через HTTP GET или POST значения и сохраняет каждое из них как элемент массива.

Поскольку она получает сразу все значения, ей требуется только один аргумент — источник данных. Варианты источников данных те же, что и для `filter_input()`.

```
$data = filter_input_array(INPUT_SOURCE);
```

ИСТОЧНИК ДАННЫХ

Все полученные данные имеют строковый тип. Некоторые фильтры, применяемые с этими функциями фильтрации, преобразуют тип данных.

На следующих страницах используется PHP функция `var_dump()` для отображения значений, полученных с помощью этих функций, поскольку важно видеть тип данных каждого значения.

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ ФИЛЬТРАЦИИ ДЛЯ ПОЛУЧЕНИЯ ДАННЫХ

PHP section_b/c06/filter_input.php

```
① <?php $location = filter_input(INPUT_GET, 'city'); ?> ...
② <a href="filter_input.php?city=London">London</a> |
② <a href="filter_input.php?city=Sydney">Sydney</a>
③ <pre><?php var_dump($location); ?></pre>
```

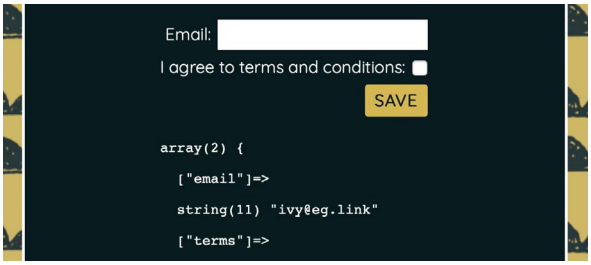
РЕЗУЛЬТАТ



PHP section_b/c06/filter_input_array.php

```
④ <?php $form = filter_input_array(INPUT_POST); ?> ...
⑤ <form action="filter_input_array.php" method="POST">
⑤   Email: <input type="text" name="email" value=""><br>
⑤   I agree to terms and conditions:
⑤   <input type="checkbox" name="terms" value="true"><br>
⑤   <input type="submit" value="Save">
⑤ </form>
⑥ <pre><?php var_dump($form); ?></pre>
```

РЕЗУЛЬТАТ



При первой загрузке этих двух примеров будет отображаться значение `null`, поскольку строка запроса пуста.

1. Функция `filter_input()` получает одно значение, отправленное через HTTP GET в строке запроса с именем `city`. Получаемое значение присваивается переменной `$location`.
2. Две ссылки используют строку запроса для отправки значений с именем `city`. Они содержат разные названия городов.
3. Функция `var_dump()` используется для отображения значения переменной `$location` и его типа данных (строковый либо null).
4. Функция `filter_input_array()` используется для получения всех значений из формы при ее отправке с использованием HTTP POST. Создаваемый функцией массив сохраняется в переменной с именем `$form`.
5. Форма содержит поле ввода и флажок, данные отправляются через HTTP POST.
6. Функция `var_dump()` отображает имена и значения, лежащие в переменной `$form`, вместе с типом данных каждого значения.

Упражнение. Отправить форму, не заполняя ее. Массив будет содержать пустую строку для ввода текста, а для флажка элемент будет отсутствовать, поскольку он не будет передан на сервер.

ВАЛИДАЦИЯ С ПОМОЩЬЮ ФИЛЬТРОВ

Когда функции фильтрации получают отправленные из браузера данные, эти данные сохраняются в виде строк. Ниже представлены три фильтра валидации. Они проверяют, является ли значение логическим, целым числом или числом с плавающей точкой. У каждого фильтра есть собственный идентификатор.

Если код на странице ожидает получить логическое значение, целое число или число с плавающей точкой, функции фильтрации могут использовать три нижеприведенных фильтра, чтобы проверить, соответствует ли предоставленное значение правильному типу данных.

Помимо проверки на соответствие типу, эти функции также преобразуют значение из строкового типа в тот тип данных, который указан в фильтре.

ИДЕНТИФИКАТОР ФИЛЬТРА	ОПИСАНИЕ
`FILTER_VALIDATE_BOOLEAN`	По умолчанию проверяет, является ли значение истинным. Возвращает **`true`**, если передано одно из следующих значений: 1, "on", "yes" и "true" в любом регистре. В остальных случаях возвращает false. Если нужна строгая проверка, что значение соответствует либо true, либо false, то см. флаг FILTER_NULL_ON_FAILURE далее
`FILTER_VALIDATE_INT`	Проверяет, является ли значение целым числом. Если да, то возвращается число в виде типа данных int. Если нет, то возвращается значение false. Важно при этом не забывать, что при нестрогом сравнении 0 приравнивается к false
`FILTER_VALIDATE_FLOAT`	Проверяет, является ли значение числом с плавающей точкой (десятичным). Если да, то возвращается значение с типом данных float. Если нет, возвращается значение false. Целые числа также проходят этот фильтр

Каждый фильтр также имеет два типа настроек. Их можно использовать для управления поведением фильтра:

- **флаги** (Flags) — это настройки, которые вы можете включать либо выключать;
- **параметры** (Options) — это настройки, в которых вы должны задать значение.

Например, фильтры для целых чисел и чисел с плавающей точкой содержат параметры, позволяющие задать минимальное и максимальное число, которое разрешено указывать пользователю. Таким образом, если посетителя попросили сообщить свой возраст и ему должно было быть от 16 до 65 лет, фильтр может проверить, находится ли введенное пользователем число в этом диапазоне.

Если число не было указано, оказалось слишком маленьким или слишком большим, оно не проходит проверку.

Все фильтры проверки также содержат параметр, позволяющий указать значение по умолчанию. Такое значение будет использовано, если полученное значение не прошло проверку.

Флаги — это параметры, у которых может быть только два значения, включен или выключен. Например, у фильтра целых чисел есть флаг, который вы можете включить, чтобы разрешить ввод чисел в шестнадцатеричной системе счисления в дополнение к стандартным цифрам 0–9. Шестнадцатеричная система счисления использует цифры 0–9 и буквы A–F для представления чисел от 0 до 15. Вы могли встречать ее при указании цвета в HTML и CSS.

Ниже вы можете увидеть фильтры для проверки различных текстовых значений. Также можно создавать пользовательские фильтры с использованием регулярных выражений.

Чаще всего данные проверяются по следующим критериям:

- количество символов, которые они могут содержать;
- разрешенные символы, которые они могут использовать;
- порядок, которому эти символы должны следовать.

Например, существуют правила, проверяющие, соответствует ли введенное значение формату адреса электронной почты, URL-адреса, имени домена или IP-адреса. Приведенные ниже четыре фильтра проверяют, соответствует ли значение этим правилам.

ИДЕНТИФИКАТОР ФИЛЬТРА	ОПИСАНИЕ
`FILTER_VALIDATE_EMAIL`	Проверяет, совпадает ли структура строки со структурой адреса электронной почты
`FILTER_VALIDATE_URL`	Проверяет, совпадает ли структура строки со структурой URL-адреса
`FILTER_VALIDATE_DOMAIN`	Проверяет, соответствует ли структура строки структуре допустимого имени домена
`FILTER_VALIDATE_IP`	Проверяет, соответствует ли структура строки структуре действительного IP-адреса

Регулярные выражения можно использовать для написания собственных фильтров, которые проверяют, соответствует ли значение определенному шаблону. Регулярное выражение указывается в качестве параметра для фильтра `FILTER_VALIDATE_REGEXP`.

ИДЕНТИФИКАТОР ФИЛЬТРА	ОПИСАНИЕ
`FILTER_VALIDATE_REGEXP`	Проверяет, содержит ли строка набор символов, описанный с помощью регулярного выражения

ИСПОЛЬЗОВАНИЕ ФИЛЬТРОВ ДЛЯ ВАЛИДАЦИИ ОТДЕЛЬНЫХ ЗНАЧЕНИЙ

Когда функции фильтрации используются для проверки данных, им необходимо указать идентификатор используемого фильтра, а также любые флаги или параметры, которые должен использовать фильтр.

При использовании `filter_input()` для получения одного фрагмента данных, помимо первых двух параметров можно также указать третий — это идентификатор используемого фильтра, а также четвертый, содержащий настройки, которые может использовать фильтр.

Функция возвращает:

- переданное значение, если оно соответствует требованиям;
- `false`, если проверка не пройдена;
- `null`, если элемент с таким именем отсутствует в источнике.

```
$data = filter_input(INPUT_SOURCE, 'name', FILTER_ID[, $settings]);
```

ИСТОЧНИК ДАННЫХ — ИМЯ — ИДЕНТИФИКАТОР — ФЛАГИ/ПАРАМЕТРЫ

Когда фильтр использует флаги и параметры, они передаются в ассоциативном массиве с двумя ключами:

- `flags` содержит настройки, которые можно включить;
- `options` содержит настройки, для которых требуется значение.

Ниже массив флагов и параметров сохраняется в переменной с именем `$settings`.

Значение для ключа `flags` — это имя требующего включения флага (оно не записывается в кавычки). Чтобы использовать несколько флагов, разделите имя каждого флага символом вертикальной черты `|`.

Значение для ключа `options` представляет собой еще один ассоциативный массив. Ключ каждого элемента — это имя заданного параметра, а значение — это используемое значение.

ФЛАГИ

```
$settings['flags'] = FLAG_NAME1 | FLAG_NAME2;
$settings['options']['option1'] = value1;
$settings['options']['option2'] = value2;
```

ПАРАМЕТР — ЗНАЧЕНИЕ

Когда необходимо задать значения для вложенных массивов, то приведенный выше синтаксис может быть проще для чтения.

Примечание. Когда функция `filter_input()` получает недопустимые данные, она возвращает значение `false`. Это означает, что указанное пользователем значение не может быть использовано.

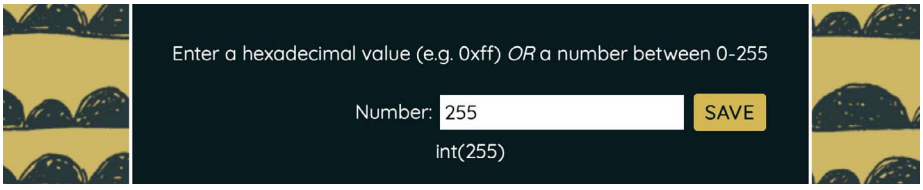
ПОЛУЧЕНИЕ ДАННЫХ С ПОМОЩЬЮ ФИЛЬТРОВ

PHP section_b/c06/validate-input.php

```php
<?php
$settings['flags']  = FILTER_FLAG_ALLOW_HEX;   // Разрешить шестнадцатеричный формат
$settings['options']['min_range'] = 0;         // Параметр для минимального числа
$settings['options']['max_range'] = 255;       // Параметр для максимального числа

$number = filter_input(INPUT_POST, 'number', FILTER_VALIDATE_INT, $settings);
?> ...
<form action="validate-input.php" method="POST">
  Number: <input type="text" name="number" value="<?= htmlspecialchars($number) ?>">
  <input type="submit" value="Save">
</form>
<?php var_dump($number); ?>
```

РЕЗУЛЬТАТ



1. Переменная `$settings` содержит массив флагов и параметров, используемых для проверки числа. Флаг указывает, что число можно вводить в шестнадцатеричной системе счисления. Параметры указывают, что минимальное допустимое число равно 0, а максимальное допустимое число равно 255.

2. Функция `filter_input()` получает значение поля ввода, отправленное из формы через HTTP POST. Имя этого поля — `number`. Третий параметр — это идентификатор фильтра. Четвертый — имя переменной, содержащей массив параметров и флагов для использования с фильтром.

3. Полученное значение присваивается переменной `$number` и отображается в поле ввода в форме. Если в переменной `$number` содержится значение `null` (если форма не была отправлена) или `false` (если данные не прошли проверку), вы его не увидите, поскольку PHP преобразует эти значения в пустую строку при выводе.

4. Функция `var_dump()` используется чтобы показать значение, содержащееся в `$number`, ведь `false` или `null` выводятся как пустые строки. Кроме того, она также отображает тип данных, чтобы показать, что прошедшее проверку значение преобразуется из строки в целое число.

ФИЛЬТРЫ ДЛЯ ВАЛИДАЦИИ НЕСКОЛЬКИХ ВХОДНЫХ ЗНАЧЕНИЙ

Чтобы получить и проверить набор из нескольких значений одновременно, вы можете использовать `filter_input_array()` и указать фильтр, который будет использоваться для каждого элемента собираемых данных.

Если код на странице ожидает получения нескольких значений, создайте ассоциативный массив с элементом для каждого проверяемого значения. Ключи для каждого элемента массива — это имена элементов управления формой или имена элементов строки запроса.

Значение для каждого элемента должно содержать:

- либо имя фильтра, который будет использоваться при сборе данных (если у него нет флагов или параметров);
- либо массив, содержащий имя фильтра и любые используемые флаги или параметры.

```
$filters['name1'] = FILTER_ID;
$filters['name2']['filter'] = FILTER_ID;
$filters['name2']['options']['option1'] = value1;
$filters['name2']['options']['option2'] = value2;
```

Затем вызывается функция `filter_input_array()` с двумя параметрами:

- источник данных (`INPUT_GET` или `INPUT_POST`);
- массив используемых фильтров со значениями, которые страница ожидает получить для каждого ввода.

В качестве результата возвращается новый ассоциативный массив. Ключами этого массива являются ключи массива `$filters`, а значение каждого элемента равно:

- полученному значению, если оно прошло проверку;
- `false`, если значение было передано, но не прошло проверку;
- `null`, если элемент с таким именем отсутствует в источнике.

```
$data = filter_input_array(INPUT_SOURCE, $filters);
```

ИСТОЧНИК ВВОДА — INPUT_SOURCE; МАССИВ ФИЛЬТРОВ — $filters

Если в скрипт передаются данные, которые не были указаны в массиве фильтров, эти данные **не** добавляются в возвращаемый функцией `filter_input_array()` массив.

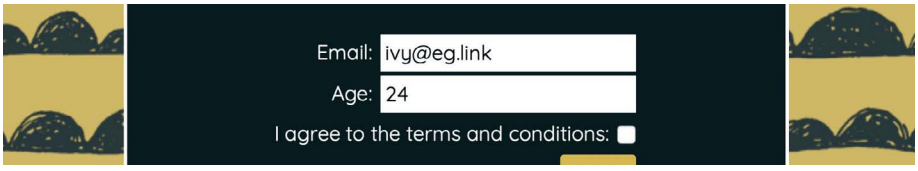
Если какие-то данные отсутствуют в источнике, то соответствующим элементам результирующего массива присваивается значение `null`. Чтобы такие значения совсем не добавлялись в массив, укажите `false` в качестве третьего аргумента.

ВАЛИДАЦИЯ НЕСКОЛЬКИХ ВХОДНЫХ ЗНАЧЕНИЙ С ПОМОЩЬЮ ФИЛЬТРОВ

PHP section_b/c06/validate-multiple-inputs.php

```php
<?php
$form['email'] = '';                                    // Инициализация email
$form['age']   = '';                                    // Инициализация возраста
if ($_SERVER['REQUEST_METHOD'] == 'POST') {             // Если отправлено
    $filters['email'] = FILTER_VALIDATE_EMAIL;          // Фильтр email
    $filters['age']['filter'] = FILTER_VALIDATE_INT;    // Фильтр целых чисел
    $filters['age']['options']['min_range'] = 16;       // Минимальное значение 16
    $form = filter_input_array(INPUT_POST, $filters); // Валидация данных
}
?> ...
<form action="validate-multiple-inputs.php" method="POST">
  Email: <input type="text" name="email" value="<?= htmlspecialchars($form['email']) ?>">
  Age: <input type="text" name="age" value="<?= htmlspecialchars($form['age']) ?>"><br>
  I agree to the terms and conditions: <input type="checkbox" name="terms" value="1"><br>
  <input type="submit" value="Save">
</form>
<pre><?php var_dump($form); ?></pre>
```

РЕЗУЛЬТАТ



1. Массив `$form` инициализируется значениями для двух полей ввода — электронной почты и возраста.
2. Если форма была отправлена, в переменной `$filters` формируется массив. Ключом для каждого элемента будет имя поля ввода формы, а значением — используемые фильтры/параметры:

- `email` должен соответствовать формату адреса электронной почты;
- `age` должно быть целым числом, равным 16 или более.

3. Функция `filter_input_array()` получает и проверяет данные и перезаписывает содержащиеся в `$form` значения.
4. Функция `var_dump()` отображает данные.

Примечание. Даже если флажок принятия условий использования был установлен, его значение не добавляется в массив `$form`, поскольку его имя было указано в массиве `$filters`. Кроме того, значения, не прошедшие проверку, не отображаются в форме, поскольку `false` выводится как пустая строка.

ФУНКЦИИ ФИЛЬТРАЦИИ ДЛЯ РАБОТЫ С ПЕРЕМЕННЫМИ

В PHP есть две встроенные функции для фильтрации значений переменных. Функция `filter_var()` применяет фильтр к одной переменной, а `filter_var_array()` применяет фильтры к набору значений в массиве.

Функции `filter_var()` требуется:

- имя переменной, значение которой будет проверяться;
- ID фильтра.

Значения для параметров или флагов устанавливаются таким же образом, как и для `filter_input()`. Возвращаемые значения тоже одинаковы: возвращается значение, если оно прошло проверку, или `false` в противном случае. Значение `null` не возвращается, поскольку проверяемая переменная задается явно при вызове функции.

```
filter_var($variable, FILTER_ID[, $settings]);
```

СОДЕРЖАЩАЯ ДАННЫЕ ПЕРЕМЕННАЯ — ФИЛЬТР — ФЛАГИ/ПАРАМЕТРЫ

Функция `filter_var_array()` также поддерживает два параметра:

- имя переменной, содержащей массив данных для проверки;
- массив фильтров и их параметров/флагов.

Значения для параметров или флагов устанавливаются таким же образом, как и для `filter_input_array()`; возвращаемые значения также совпадают.

Если указан только один фильтр, он применяется ко всем значениям в массиве.

```
filter_var_array($array, $filters);
```

СОДЕРЖАЩИЙ ДАННЫЕ МАССИВ — ИСПОЛЬЗУЕМЫЕ ФИЛЬТРЫ

Когда `filter_input()` или `filter_input_array()` используются для проверки данных, они возвращают значение `false` для любых не прошедших проверку значений (вместо введенных пользователем данных). Это означает, что, если из формы пришли некорректные данные, пользователь не сможет увидеть введенное им недопустимое значение.

Чтобы отобразить введенные значения, их можно собрать в отдельный массив. При этом значения, прошедшие валидацию с помощью `filter_var()` или `filter_var_array()`, сохраняются в другом массиве. И далее:

- если данные прошли проверку, то далее используются отфильтрованные данные;
- если же нет, то в форме данные отображаются в исходном виде, из массива, в котором они были первоначально собраны до проверки.

ВАЛИДАЦИЯ ДАННЫХ В ПЕРЕМЕННЫХ

PHP section_b/c06/validate-variables.php

```php
<?php
$form['email'] = '';                                    // Инициализация
$form['age']   = '';
$form['terms'] = 0;
$data          = [];
if ($_SERVER['REQUEST_METHOD'] == 'POST') {             // Если отправлено
    $filters['email'] = FILTER_VALIDATE_EMAIL;          // Фильтр email
    $filters['age']['filter'] = FILTER_VALIDATE_INT;    // Фильтр целых чисел
    $filters['age']['options']['min_range'] = 16;       // Минимальный возраст
    $filters['terms'] = FILTER_VALIDATE_BOOLEAN;        // Фильтр логических переменных
    $form = filter_input_array(INPUT_POST);             // Получение всех значений
    $data = filter_var_array($form, $filters);          // Применение фильтров
}
?> ...
<form action="validate-variables.php" method="POST">
  Email: <input type="text" name="email" value="<?= htmlspecialchars($form['email']) ?>">
  Age: <input type="text" name="age" value="<?= htmlspecialchars($form['age']) ?>"><br>
  I agree to the terms and conditions: <input type="checkbox" name="terms" value="1"><br>
  <input type="submit" value="Save">
</form>
<pre><?php var_dump($data); ?></pre>
```

Этот пример выглядит так же, как и предыдущий.

1. Массивы `$form` и `$data` инициализируются значениями, которые будут выведены, когда форма еще не отправлена.
2. Если форма была отправлена, в массиве `$filters` будут определены фильтры и параметры для проверки данных.
3. Функция `filter_input_array()` получает данные из формы, перезаписывая сохраненные в `$form` в шаге 1 значения. Поскольку в ней не используются фильтры, все данные записываются как есть.
4. Функция `filter_var_array()` проверяет данные формы, используя фильтры, указанные в массиве `$filters`. Она сохраняет массив результатов в переменной с именем `$data`.
5. В полях ввода отображаются предоставленные пользователем значения, которые были сохранены в массиве `$form` до валидации данных.
6. Функция `var_dump()` показывает проверенные данные, которые были сохранены в массиве `$data` (или `null`, если форма не была отправлена).

Упражнение. Удалить из формы поле ввода для возраста, обновить страницу и затем повторно отправить форму. Элемент `age` будет присутствовать в выдаче функции `var_dump()` в шаге 6, несмотря на то что его нет в источнике данных. Это происходит потому, что этот элемент задан в массиве фильтров.

ФИЛЬТРЫ ВАЛИДАЦИИ, ИХ ФЛАГИ И ПАРАМЕТРЫ

В таблицах ниже показаны фильтры, флаги и параметры для работы с логическими значениями и числами. В таблицах справа показаны фильтры, флаги и параметры для работы со строками.

Все фильтры валидации также содержат параметр под названием `default`.

Этот параметр позволяет указать значение по умолчанию, если данные неверны.

`FILTER_VALIDATE_BOOLEAN`
Проверяет, равно ли значение `true` (`1`, `on`, `yes` также рассматриваются как `true`). Возвращает логическое значение `true`, если значение равно `true`, и `false`, если это не так, и `null`, если данные отсутствуют. Нечувствителен к регистру.

ФЛАГ	ОПИСАНИЕ
`FILTER_NULL_ON_FAILURE`	Возвращает `null` (не `false`), если значение не может быть интерпретировано ни как `true`, ни как `false`

`FILTER_VALIDATE_INT`
Проверяет, является ли значение целым числом. Если да, возвращает число в виде типа данных `int`.

ФЛАГ	ОПИСАНИЕ
`FILTER_FLAG_ALLOW_HEX`	Разрешение на использование шестнадцатеричных чисел
`FILTER_FLAG_ALLOW_OCTAL`	Разрешение на использование восьмеричных чисел

ПАРАМЕТР	ОПИСАНИЕ
`min_range`	Минимально допустимое число
`max_range`	Максимально допустимое число

`FILTER_VALIDATE_FLOAT`
Проверяет, является ли значение числом с плавающей точкой (десятичным). Допустимы целые числа. Если значение проходит проверку, оно возвращается в виде типа данных `float`.

ФЛАГ	ОПИСАНИЕ
`FILTER_FLAG_ALLOW_THOUSAND`	Позволяет числу с плавающей точкой использовать запятую в качестве разделителя тысяч. Возвращает `null` (не `false`), если значение недопустимо

FILTER_VALIDATE_REGEXP
Проверяет, содержит ли строка шаблон символов, описанный в регулярном выражении.

ПАРАМЕТР	ОПИСАНИЕ
regexp	Регулярное выражение для использования

FILTER_VALIDATE_EMAIL
Проверяет, совпадает ли структура строки со структурой адреса электронной почты.

ФЛАГ	ОПИСАНИЕ
FILTER_FLAG_EMAIL_UNICODE	Разрешает использование символов юникода в части имени адреса (часть перед символом @)

FILTER_VALIDATE_URL
Проверяет, соответствует ли структура строки структуре допустимого URL-адреса.

ФЛАГ	ОПИСАНИЕ
FILTER_FLAG_SCHEME_REQUIRED	Должен содержать схему например, `http://` или `ftp://`
FILTER_FLAG_HOST_REQUIRED	Должно содержать имя хоста
FILTER_FLAG_PATH_REQUIRED	Должен содержать путь к файлу или директории
FILTER_FLAG_QUERY_REQUIRED	Должен содержать строку запроса

FILTER_VALIDATE_DOMAIN
Проверяет, соответствует ли структура строки структуре имени домена.

ФЛАГ	ОПИСАНИЕ
FILTER_FLAG_HOSTNAME	Валидация имени хоста

FILTER_VALIDATE_IP
Проверяет, соответствует ли структура строки структуре IP-адреса.

ФЛАГ	ОПИСАНИЕ
FILTER_FLAG_IPV4	Проверяет, является ли значение допустимым IPV4 IP-адресом
FILTER_FLAG_IPV6	Проверяет, является ли значение допустимым IPV6 IP-адресом
FILTER_FLAG_NO_RES_RANGE	Не разрешает IP-адреса из зарезервированного диапазона (адреса, используемые в локальных сетях и не отправляемые через интернет)
FILTER_FLAG_NO_PRIV_RANGE	Не разрешает IP-адреса из закрытого диапазона (подмножество зарезервированных IP-адресов)

ФИЛЬТРЫ ДЛЯ ОЧИСТКИ

Очистка данных включает в себя удаление символов, которых не должно быть в значении, и при необходимости — их замену. Все четыре функции фильтрации PHP могут использовать набор встроенных фильтров для очистки данных.

В дополнение к фильтрам валидации встроенные функции фильтрации PHP могут также использовать набор встроенных **фильтров очистки**. Они удаляют (либо заменяют) любые неразрешенные символы.

Первый фильтр в таблице ниже выполняет ту же задачу, что и функция `htmlspecialchars()`, — он заменяет на сущности пять зарезервированных символов, которые HTML рассматривает как код.

Второй фильтр кодирует значение для строки запроса в URL-адресе, заменяя символы, появление которых запрещено в URL, закодированными версиями этих символов.

Остальные фильтры удаляют символы, которым запрещено появляться в тексте, цифрах, адресах электронной почты и URL-адресах (но не заменяют эти символы).

Вы должны экранировать данные перед использованием, а не при получении, потому что этот процесс изменит данные. Например, представьте, что посетитель ввел текст `Fish & Chips`.

Чтобы отобразить этот текст на странице, амперсанд нужно экранировать: `Fish & Chips`. Но если текст был экранирован при получении, функция поиска может не найти текст `Fish & Chips`, поскольку амперсанд уже экранирован.

Кроме того, символы экранируются по-разному в зависимости от способа использования данных, это известно как **контекст** вывода данных. Чтобы отобразить тот же текст в строке запроса, пробел заменяется на `%20`, а амперсанд — на `%26`, так что ссылка становится такой: `http://eg.link/search.php?Fish%20%26%20Chips`. То есть невозможно экранировать значение на все случаи жизни при получении. И это еще одна причина экранировать перед использованием.

ИДЕНТИФИКАТОР ФИЛЬТРА	ОПИСАНИЕ
`FILTER_SANITIZE_FULL_SPECIAL_CHARS`	Эквивалент функции `htmlspecialchars()` с активным флагом `ENT_QUOTES`
`FILTER_SANITIZE_ENCODED`	Кодирует значение для строки запроса в URL
`FILTER_SANITIZE_STRING`	Удаляет теги из строки[36]
`FILTER_SANITIZE_NUMBER_INT`	Удаляет символы, отличные от `0-9` и `+` или `-`
`FILTER_SANITIZE_NUMBER_FLOAT`	Удаляет символы, отличные от `0-9` и `+` или `-`. Поддерживает флаги, позволяющие использовать разделители десятичные и тысяч, а также «e» или «E» для экспоненциального формата
`FILTER_SANITIZE_EMAIL`	Удаляет символы, запрещенные в адресах электронной почты. Допускаются: `A–z 0-9 ! # $ % & ' * + – = ? ^ _ ` {|} ~ @ . []`
`FILTER_SANITIZE_URL`	Удаляет символы, недопустимые в URL-адресах. Допускаются `A–z 0-9 $ – _ . + ! * ' () , {} | \ \ ^ ~ [] ` <> # % " ; / ?: @ & =`

ПРИМЕНЕНИЕ ФИЛЬТРОВ ОЧИСТКИ К ПЕРЕМЕННЫМ

PHP section_b/c06/sanitization-filters.php

```php
<?php
$user['name']  = 'Ivy<script src="js/bad.js"></script>';      // Имя пользователя
$user['age']   = 23.75;                                       // Возраст пользователя
$user['email'] = '£ivy@eg.link/';                             // Email пользователя

$sanitize_user['name']  = FILTER_SANITIZE_FULL_SPECIAL_CHARS; // Фильтр HTML Escape
$sanitize_user['age']   = FILTER_SANITIZE_NUMBER_INT;         // Фильтр целых чисел
$sanitize_user['email'] = FILTER_SANITIZE_EMAIL;              // Фильтр email

$user = filter_var_array($user, $sanitize_user);              // Очистка данных
?> ...
<p>Name:  <?= $user['name'] ?></p>
<p>Age:   <?= $user['age'] ?></p>
<p>Email: <?= $user['email'] ?></p>
<pre><?php var_dump($user); ?></pre>
```

1. В переменной $user содержится массив данных о пользователе.
2. Массив $sanitize_user инициализируется с тремя ключами, имена которых совпадают с ключами в массиве $user. Значения — это имена фильтров очистки, используемых с этими значениями.
3. Функция filter_var_array() вызывается для применения фильтров очистки к хранящимся в массиве $user значениям. Имя экранируется, а нежелательные символы удаляются из возраста и адреса электронной почты.
4. Отображаются очищенные данные.

РЕЗУЛЬТАТ

```
Name: Ivy<script src="js/bad.js"></script>

Age: 2375

Email: ivy@eg.link
```

5. Функция PHP var_dump() показывает очищенный массив $user (не показан в приведенном выше результате).

Упражнение. В шаге 2 удалить элемент age из массива $user. Поскольку элемент присутствует в массиве фильтров, ему будет присвоено значение null в массиве $user после отправки формы.

Примечание: десятичный разделитель был удален из возраста (age), что делает его значение равным 2375. Следует проявить осторожность, чтобы процесс очистки не изменил полученные вами значения. Чтобы разрешить применение десятичных разделителей, разделителей тысяч или специфичную нотацию, добавьте флаги для фильтра чисел **http://notes.re/php/sanitize**.

ВАЛИДАЦИЯ ФОРМ С ПОМОЩЬЮ ФИЛЬТРОВ

В этом примере используются фильтры валидации для проверки данных из нескольких элементов управления формой, а также фильтры очистки, чтобы гарантировать, что любые данные, предоставленные пользователями, безопасны для отображения на странице.

1. Инициализируются массивы `$user` и `$error` и переменная `$message`. Это позволяет использовать их в форме внизу страницы при ее первой загрузке (перед отправкой).
2. В конструкции `if` проверяется, была ли отправлена форма.
3. Переменная `$validation_filters` содержит массив фильтров, используемых для проверки данных формы.
4. Функция `filter_input_array()` получает значения из формы и применяет фильтры проверки к собранным значениям. Результаты перезаписывают значения, которые были сохранены в `$user` в шаге 1:

- если значение прошло проверку, оно сохраняется в массиве;
- если нет, то сохраняется значение `false`;
- если значение отсутствует, в массив запишется значение `null`.

5. Каждое значение в массиве `$errors` задается с помощью тернарного оператора. Проверяется каждый фрагмент данных:

- если значение приравнивается к `true`, то в массив запишется пустая строка;
- в противном случае запишется сообщение об ошибке, информирующее пользователя, как исправить это значение.

6. Функция PHP `implode()` используется для объединения всех значений в массиве `$errors` в одну строку и сохранения их в переменной с именем `$invalid`.
7. Условие в конструкции `if` проверяет, содержит ли переменная `$invalid` любой текст, что приравнивается к значению `true` (а пустая строка интерпретируется как `false`).
8. Если данные не прошли проверку, то в переменную `$message` запишется сообщение, в котором посетителю предлагается исправить ошибки формы.
9. В противном случае переменная `$message` сообщает пользователю, что проблем во введенных данных не обнаружено. На этом этапе страница могла бы обработать полученные данные (и не было бы необходимости снова отображать форму).
10. Сохраненные в массиве `$user` имя и возраст очищаются, чтобы убедиться, что их можно безопасно отобразить на странице. Это делается с помощью функции PHP `filter_var()`:

- имя очищается, чтобы заменить все зарезервированные символы HTML сущностями;
- число очищается таким образом, чтобы оно содержало только разрешенные в целых числах символы.

11. Отображается сообщение из переменной `$message`.
12. Отображение формы. Если предоставленные пользователем данные:

- прошли проверку, то такие значения отображаются в элементах управления формой;
- если не прошли, то эти элементы управления формой будут пустыми.

Если пользователь не отправил форму, элементы управления формой используют начальные значения, присвоенные каждому элементу массива `$user` в шаге 1.

Если какие-либо данные неверны, рядом с соответствующим элементом управления формой отображается сообщение об ошибке.

PHP section_b/c06/validate-form-using-filters.php

```php
<?php
$user    = ['name' => '', 'age' => '', 'terms' => '', ];        // Инициализация
$errors  = ['name' => '', 'age' => '', 'terms' => false, ];     // (1)
$message = '';

if ($_SERVER['REQUEST_METHOD'] == 'POST') {                     // (2) Если форма отправлена
    // Фильтры валидации
    $validation_filters['name']['filter']              = FILTER_VALIDATE_REGEXP;
    $validation_filters['name']['options']['regexp']   = '/^[A-z]{2,10}$/';
    $validation_filters['age']['filter']               = FILTER_VALIDATE_INT;       // (3)
    $validation_filters['age']['options']['min_range'] = 16;
    $validation_filters['age']['options']['max_range'] = 65;
    $validation_filters['terms']                       = FILTER_VALIDATE_BOOLEAN;

    $user = filter_input_array(INPUT_POST, $validation_filters); // (4) Валидация данных

    // Создание сообщений об ошибках
    $errors['name']  = $user['name']  ? '' : 'Name must be 2-10 letters using A-z';
    $errors['age']   = $user['age']   ? '' : 'You must be 16-65';                  // (5)
    $errors['terms'] = $user['terms'] ? '' : 'You must agree to the terms & conditions';
    $invalid = implode($errors);         // (6) Объединение сообщений об ошибках в строку

    if ($invalid) {                                             // (7) Если есть ошибки
        $message = 'Please correct the following errors:';      // (8) Не обрабатывать
    } else {                                                    // В противном случае
        $message = 'Thank you, your data was valid.';           // (9) Можно обрабатывать
    }

    // Очистка данных
    $user['name'] = filter_var($user['name'], FILTER_SANITIZE_FULL_SPECIAL_CHARS);
    $user['age']  = filter_var($user['age'],  FILTER_SANITIZE_NUMBER_INT);      // (10)
}
?> ...
<?= $message ?>                                                 // (11)
<form action="validate-form-using-filters.php" method="POST">
  Name: <input type="text" name="name" value="<?= $user['name'] ?>">
  <span class="error"><?= $errors['name'] ?></span><br>
  Age: <input type="text" name="age" value="<?= $user['age'] ?>">
  <span class="error"><?= $errors['age'] ?></span><br>          <!-- (12) -->
  <input type="checkbox" name="terms" value="true"
        <?= $user['terms'] ? 'checked' : '' ?>> I agree to the terms and conditions
  <span class="error"><?= $errors['terms'] ?></span><br>
  <input type="submit" value="Save">
</form>
```

ЗАКЛЮЧЕНИЕ

ПОЛУЧЕНИЕ ДАННЫХ ИЗ БРАУЗЕРА

- Данные, отправляемые на сервер через строку запроса и форму, добавляются в суперглобальные массивы `$_GET` и `$_POST` в виде строк.
- Если значение может отсутствовать в суперглобальном массиве, используйте конструкцию `isset()` или укажите значение по умолчанию с помощью оператора объединения с `null`.
- Данные также могут быть получены с помощью функций `filter_input()` или `filter_input_array()`.
- Перед тем как использовать данные, их необходимо проверить. Выясните, были ли предоставлены все необходимые данные и находятся ли они в правильном формате.
- Прежде чем отображать любые данные в браузере, их необходимо экранировать, чтобы предотвратить XSS-атаки и ошибки. Замените зарезервированные символы HTML-сущностями.
- Фильтры валидации используются в функциях фильтрации для проверки значений и преобразования их в правильный тип данных.
- Фильтры очистки используются в функциях фильтрации для замены или удаления нежелательных символов.

7

ИЗОБРАЖЕНИЯ И ФАЙЛЫ

В этой главе показано, как разрешить посетителям загружать изображения на сервер и как безопасно отображать их на ваших PHP-страницах. Эти методы работают и для других типов файлов.

Сначала вы узнаете, как пользователи загружают изображения и как сервер их получает. Вы увидите, как:

- поле ввода для загрузки файлов используется в HTML-форме, чтобы пользователи могли загружать файлы;
- интерпретатор PHP добавляет данные о файле в суперглобальный массив с именем `$_FILES`;
- файл помещается во временную папку на сервере;
- файл перемещается в папку, в которой будут храниться загруженные файлы.

Далее вы узнаете, как провести валидацию загруженных файлов и проверить, что:

- имя файла содержит только разрешенные символы;
- файла с таким именем еще нет;
- у файла разрешенные MIME-тип и расширение;
- размер файла не слишком велик.

Наконец, вы узнаете, как управлять изображениями для создания:

- миниатюр изображений;
- обрезанных версий изображений.

На протяжении главы вы познакомитесь с новыми встроенными функциями, помогающими в решении этих задач. Хотя в этой главе загрузка файлов демонстрируется на примере изображений, те же самые приемы можно применить для загрузки посетителями аудио, видео, PDF и других типов файлов.


NATURAL
of COLOURS
RED
orange-red
red-orange
Orange
yellow-orange
orange-yellow
YEL
yellow-green
Safari File Edit View History Bookmarks Develop Window Help
Thu 16:16 Jon Duckett
localhost
MOUNTAIN ART SUPPLIES
Upload file: Choose File no file selected
UPLOAD

ЗАГРУЗКА ФАЙЛОВ ИЗ БРАУЗЕРА

HTML-формы могут содержать поле ввода для загрузки файлов. Посетители используют его для отправки файлов на сервер.

При создании HTML-формы, которая позволяет посетителям загружать файлы, открывающий тег `<form>` должен содержать следующие атрибуты:

- `method` со значением `POST` указывает, что форма должна быть отправлена через HTTP POST (поскольку файлы не должны отправляться с использованием HTTP GET);
- `enctype` со значением `multipart/form-data` указывает тип кодировки, который требуется для отправки файлов;
- `action`. Его значение — PHP-файл, предназначенный для получения данных формы.

Элемент управления загрузкой файлов создается с помощью HTML-тега `<input>`. Его атрибут `type` должен содержать значение `file`. В браузере это создает кнопку, открывающую новое окно, которое позволяет пользователю выбрать файл для загрузки:

```
<input type="file" name="image">
```

При загрузке файлов на сервер структура отправляемых данных является более сложной, но для простоты можно принять, что отправляется пара «имя — значение»:

- имя — это значение атрибута `name` для элемента управления файлом (выше он называется `image`);
- значение — это отправляемый пользователем файл.

Тег поля загрузки файла может содержать атрибут `accept`. Его применяют, чтобы ограничить перечень типов файлов, доступных для загрузки пользователем. Его значением должен быть разделенный запятыми список MIME-типов, которые принимает сайт. Вы можете узнать об этих типах здесь **`http://notes.re/media-types`**.

```
<input type="file" name="image"
 accept="image/jpeg, image/png">
```

Когда посетитель нажимает на кнопку для загрузки файла, современные браузеры не отображают файлы, отсутствующие в списке принятых типов, чтобы их нельзя было выбрать. Это возможно благодаря атрибуту `accept`.

Этот способ можно применять для удобства пользователей, но нельзя полагаться на атрибут `accept` для ограничения загружаемых посетителями типа файлов, поскольку они могут переопределить этот параметр, а старые браузеры его не поддерживают. Chrome 10, Internet Explorer 10, Firefox 10 и Safari 6 были первыми версиями популярных браузеров, поддерживавших эту функцию. Поэтому всегда необходимо проверять MIME-тип файла на сервере с помощью PHP.

Чтобы разрешить все подтипы MIME-типа, можно добавить символ звездочки вместо подтипа.

Следующее выражение позволяет использовать все форматы изображений (включая BMP, GIF, JPEG, PNG, TIFF и WebP):

```
<input type="file" accept="image/*">
```

1. Приведенная ниже форма позволяет посетителям загружать изображения. Она используется во всех примерах в этой главе. Для открывающего тега `<form>` требуется:

- атрибут `method` со значением `POST`;
- установленное для атрибута `enctype` значение `multipart/form-data`;
- атрибут `action`, указывающий файл для отправки данных формы (это значение меняется в каждом примере).

2. Для создания элемента управления загрузкой файла тег `<input>` содержит атрибут `type`, значение которого равно `file`.

Поскольку примеры в этой главе посвящены загрузке изображений, значение атрибута `name` равно `image` (изображение).

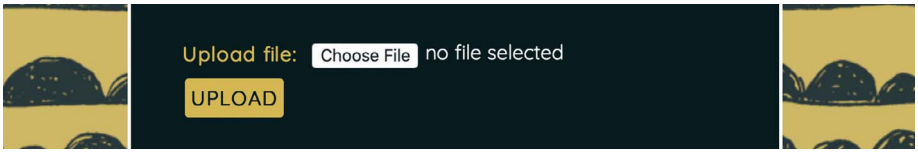
3. Кнопка отправки (`submit`) используется для отправки формы.

HTML

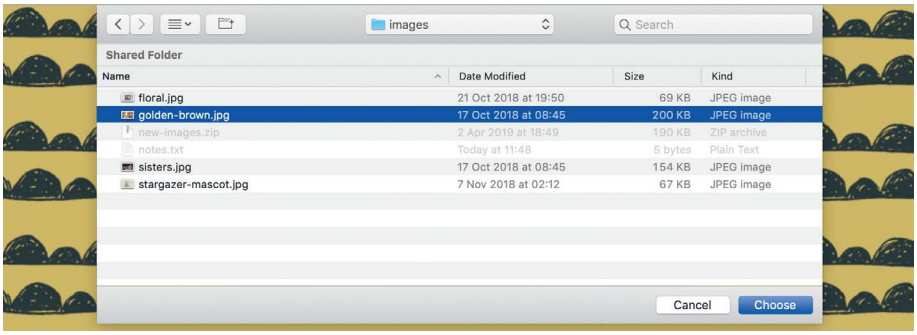
```
① <form method="post" action="filename.php" enctype="multipart/form-data">
    <label for="image"><b>Upload file:</b></label>
②   <input type="file" name="image" accept="image/*" id="image"><br>
③   <input type="submit" value="Upload">
  </form>
```

На первой иллюстрации показана форма с полем для загрузки файла, которую создает вышеприведенный код. После того как файл выбран, текст рядом с кнопкой заменяется именем файла. На второй иллюстрации показано окно, открывающееся, когда пользователь нажимает на кнопку **Choose File** (Выбрать файл). Текстовый и zip-файл не активны, поскольку это не изображения.

РЕЗУЛЬТАТ



РЕЗУЛЬТАТ



Внешний вид окна, появляющегося для выбора файлов, зависит от браузера и операционной системы (вы не можете управлять его внешним видом с помощью CSS).

ПОЛУЧЕНИЕ ФАЙЛОВ НА СЕРВЕРЕ

Когда файл загружается через форму на странице, веб-сервер сохраняет его во временной папке, а интерпретатор PHP сохраняет сведения о файле в суперглобальном массиве с именем `$_FILES`.

Форма может содержать несколько элементов управления загрузкой файлов, поэтому интерпретатор PHP создаст элемент в суперглобальном массиве `$_FILES` для каждого такого элемента, отправляемого формой.

Имя элемента совпадает с именем поля для загрузки файла, а его значение представляет собой массив данных о файле, который был загружен с помощью этого элемента.

В таблице ниже приведена информация, содержащаяся в суперглобальном массиве `$_FILES` для каждого загруженного файла.

Изображения в этой главе загружаются с помощью поля для загрузки файлов с именем `image`, поэтому массив `$_FILES` будет содержать элемент с ключом `image`, а его значением будет массив, содержащий информацию об этом изображении.

КЛЮЧ	ЗНАЧЕНИЕ	КАК ПОЛУЧИТЬ ДОСТУП К ЗНАЧЕНИЮ
`name`	Имя файла	`$_FILES['image']['name']`
`tmp_name`	Временное расположение файла (задается интерпретатором PHP)	`$_FILES['image']['tmp_name']`
`size`	Размер в байтах	`$_FILES['image']['size']`
`type`	MIME-тип (установленный браузером)	`$_FILES['image']['type']`
`error`	0 при успешной загрузке файла; код ошибки, если возникла проблема	`$_FILES['image']['error']`

После загрузки файла необходимо проверить, что не возникло никаких ошибок.

Если ключ `error` в созданном для этого файла массиве содержит значение 0, это означает, что ошибок не было.

```
if ($_FILES['image']['errors'] === 0) {
    //Обработка изображения
} else {
    // Вывод сообщения об ошибке
}
```

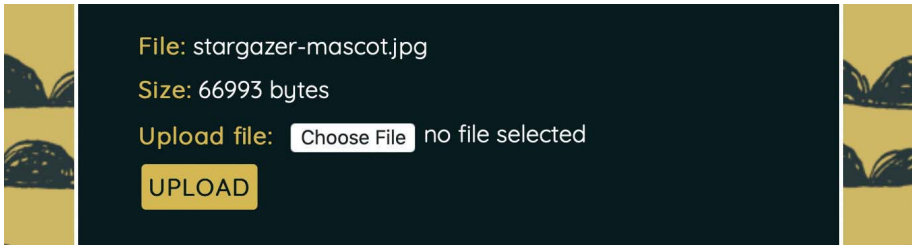
ПРОВЕРКА ЗАГРУЗКИ ФАЙЛА

1. Переменная `$message` инициализируется пустой строкой. В нее запишется новое значение, когда форма будет отправлена.
2. Проверка, была ли форма отправлена с использованием HTTP POST.
3. Конструкция `if` проверяет отсутствие ошибок.
4. Если ошибок нет, имя и размер файла сохраняются в `$message`.
5. В противном случае а `$message` сохраняется сообщение об ошибке.
6. Отображается значение переменной `$message`.

PHP section_b/c07/upload-file.php

```
<?php
(1) $message = '';                                          // Инициализация
(2) if ($_SERVER['REQUEST_METHOD'] == 'POST') {             // Если форма отправлена
(3)     if ($_FILES['image']['error'] === 0) {              // Если ошибок нет
(4)         $message  = '<b>File:</b> ' . $_FILES['image']['name'] . '<br>';   // Имя файла
            $message .= '<b>Size:</b> ' . $_FILES['image']['size'] . ' bytes'; // Размер файла
        } else {                                            // В противном случае
(5)         $message  = 'The file could not be uploaded.';  // Сообщение об ошибке
        }
}
?> ...
(6) <?= $message ?>
<form method="POST" action="upload-file.php" enctype="multipart/form-data">
  <label for="image"><b>Upload file:</b></label>
  <input type="file" name="image" accept="image/*" id="image"><br>
  <input type="submit" value="Upload">
</form>
```

РЕЗУЛЬТАТ



ПЕРЕМЕЩЕНИЕ ФАЙЛА В ПУНКТ НАЗНАЧЕНИЯ

Функция PHP `move_uploaded_file()` перемещает файл из его временного местоположения туда, где он должен храниться на сервере.

Когда файл загружается на сервер, ему присваивается временное имя и он помещается во временную папку. Временное имя файла создается интерпретатором PHP.

Интерпретатор PHP удалит временный файл из этой папки, когда скрипт завершит выполнение. Поэтому, чтобы сохранить загруженный файл на сервере, необходимо вызвать функцию `move_uploaded_file()` и переместить его в другую папку. Эта функция вызывается с двумя параметрами:

- временное местоположение файла;
- место назначения, в котором должен быть сохранен файл.

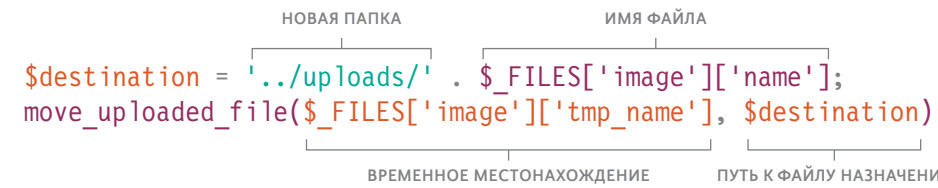
Она возвращает значение `true`, если удалось переместить файл в новое местоположение, и `false`, если нет.

Компоненты пункта назначения (местоположения, в котором файл должен быть сохранен):

- путь к папке, в которой будет храниться загруженный файл (эта папка должна быть создана до того, как вы попытаетесь переместить в нее файл);
- имя файла (его исходное или новое имя).

Если вы хотите использовать исходное имя загруженного файла, получить к нему доступ можно через массив, созданный интерпретатором PHP для этого файла, по ключу `name`.

Ниже путь к файлу назначения присваивается переменной `$destination`. Он создается путем указания папки `uploads` (лежащей на один уровень выше от текущей), за которой следует исходное имя файла, использованное при загрузке изображения.



ПРАВА ДОСТУПА К ФАЙЛАМ И ПАПКАМ

Права доступа (permissions) для целевой директории должны:

- разрешить веб-серверу читать/записывать файлы — это позволит ему сохранять и показывать изображения;
- отключить разрешение на выполнение — это предотвращает выполнение вредоносных скриптов.

ПРИМЕР ПРОВЕРКИ УСПЕШНОСТИ ЗАГРУЗКИ ФАЙЛА

Функция PHP `move_uploaded_file()` проверяет, что файл был загружен через HTTP POST, прежде чем перемещать его. Если вам нужно использовать файл перед его перемещением, примените функцию PHP `is_uploaded_file()`, чтобы выполнить эту проверку. Это помогает убедиться, что мы обращаемся именно к тому файлу, который был загружен.

ПРИМЕР ПЕРЕМЕЩЕНИЯ ЗАГРУЖЕННОГО ФАЙЛА

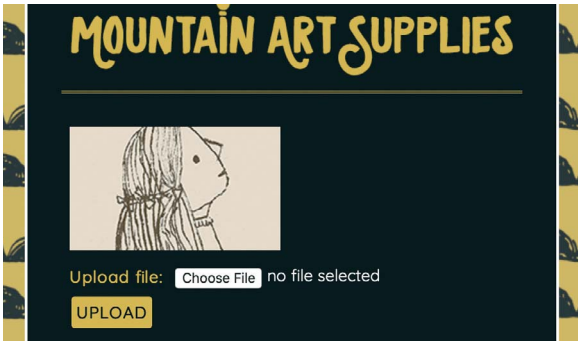
PHP section_b/c07/move-file.php

```php
<?php
$message = '';                              // Инициализация
$moved   = false;                           // Инициализация

if ($_SERVER['REQUEST_METHOD'] == 'POST') { // Если
                                            // отправлен +
    if ($_FILES['image']['error'] === 0) {// Ошибок нет
  // Присвоение переменным временного пути и нового места
  // назначения
        $temp = $_FILES['image']['tmp_name'];
        $path = 'uploads/' . $_FILES['image']['name'];
// Перемещение файла и сохранение результата в $moved
        $moved = move_uploaded_file($temp, $path);
    }

    if ($moved === true) { // Если перемещение
                    //сработало, показать изображение
        $message = '<img src="' . $path . '">';
    } else {                // Иначе сообщение об ошибке
        $message = 'The file could not be saved.';
    }
}
?> ...
<?= $message ?>
```

РЕЗУЛЬТАТ



1. Переменная с именем `$moved` инициализируется значением `false`. Это значение изменится на `true`, если изображение будет успешно перемещено.

2. Если форма была отправлена и в ней не было ошибок…

3. В `$temp` пишется путь к временному файлу.

4. В `$path` пишется путь, по которому файл необходимо сохранить (с тем же именем, что и при загрузке).

5. `move_uploaded_file()` перемещает файл из временного местоположения (в `$temp`) в новое место (в `$path`). Функция возвращает значение `true`, если перемещение было успешным, или `false`, если оно завершилось неудачей. Это значение заменяет сохраненное в шаге 1 значение переменной `$moved`.

6. С помощью условной конструкции проверяется значение `$moved`, оно должно быть равно `true`, если перемещение сработало.

7. В случае успеха в `$message` записывается HTML-тег `<img>`, показывающий загруженное изображение.

8. В противном случае в `$message` сохраняется сообщение об ошибке.

9. Значение `$message` отображается пользователю.

ОЧИСТКА ИМЕНИ ФАЙЛА И ПРОВЕРКА НА ДУБЛИКАТЫ

Прежде чем переместить файл из его временного расположения, вы должны:

а) удалить из имени файла символы, которые могут вызвать проблемы;

б) убедиться, что не будет перезаписан другой файл с тем же именем.

Такие символы, как амперсанды, двоеточия, точки и пробелы, должны быть удалены из имен файлов, поскольку они могут вызвать проблемы. Для этого вы можете заменить символы, отличные от A–Z, a–z и 0–9, тире.

1. Используйте функцию PHP `pathinfo()`, чтобы получить базовое имя (часть имени до точки)[37] и расширение файла.

2. Примените функцию PHP `preg_replace()`, чтобы заменить на дефис любые символы, кроме A–Z, a–z и 0–9, в базовом имени файла.

3. Создайте путь к файлу назначения, присоединив директорию загрузки, базовое имя, точку и расширение файла. Это значение необходимо сохранить в переменной.

```
① $basename  = pathinfo($filename, PATHINFO_FILENAME);
  $extension = pathinfo($filename, PATHINFO_EXTENSION);
② $basename  = preg_replace('/[^A-z0-9]/', '-', $basename);
③ $filepath  = 'uploads/' . $basename . '.' . $extension;
```

Если при перемещении файла с помощью функции `move_uploaded_file()` файл с таким именем уже существует, то старый файл заменяется новым. Чтобы предотвратить это, каждому файлу необходимо присвоить уникальное имя.

4. Создайте переменную-счетчик `$i` и присвойте ей начальное значение 1.

5. В условии цикла `while` используйте функцию PHP `file_exists()`, чтобы проверить существование файла с таким же именем.

6. Если файл существует, добавьте 1 к значению, сохраненному в счетчике.

7. Обновите имя файла, добавив значение счетчика после базового имени перед расширением. Например, если `upload.jpg` существует, назовите файл `upload1.jpg`.

Затем условие цикла выполняется снова, чтобы проверить, существует ли новое имя файла. Цикл повторяет шаги 5–7 до тех пор, пока не получит уникальное имя файла.

```
④ $i = 1;
⑤ while (file_exists('uploads/' . $filename)) {
⑥     $i = $i + 1;
⑦     $filename = $basename . $i . '.' .  $extension;
  }
```

ВАЛИДАЦИЯ РАЗМЕРА И ТИПА ФАЙЛА

Чтобы убедиться, что сайт может работать с загруженным файлом, перед его перемещением проверьте, что:

а) файл не слишком большой (загрузка/обработка больших файлов занимает больше времени);

б) сайт может работать с MIME-типом и расширением файла.

Вы можете установить максимальный размер загружаемого файла в `php.ini` или `.htaccess` либо добавить код проверки размера файла на странице, принимающей загруженный файл.

Чтобы узнать, что размер файла не превышает максимально допустимое значение, заданный в `php.ini` или `.htaccess`, вы можете использовать элемент с ключом `error` для этого файла в массиве `$_FILES`. Если превышает, то значение этого элемента будет равно 1.

Вы также можете проверить размер файла самостоятельно. Элемент массива `$_FILES` для этого файла с ключом `size` содержит размер файла в байтах.

Два приведенных ниже тернарных оператора используются для выполнения обеих этих проверок. Условие в первом тернарном операторе проверяет, равен ли код ошибки 1, условие во втором — превышает ли размер файла 5 Мб.

```
$error = ($_FILES['image']['error'] === 1)     ? 'Too large' : '';
$error = ($_FILES['image']['size'] <= 5242880) ? '' : 'Too large';
```

Проверка MIME-типа и расширения помогает гарантировать, что сайт может безопасно обрабатывать файл.

1. Переменная `$allowed_types` — это массив разрешенных MIME-типов.

2. Функция PHP `mime_content_type()` пытается определить тип файла и сохраняет его в `$type`.

3. Функция PHP `in_array()` проверяет, находится ли MIME-тип этого файла в массиве разрешенных.

4. Переменная `$allowed_exts` содержит массив разрешенных расширений.

5. Имя файла преобразуется в нижний регистр и сохраняется в переменной `$filename`.

6. Из имени файла выделяется расширение, которое сохраняется в переменной `$ext`.

7. Функция PHP `in_array()` проверяет, находится ли это расширение файла в массиве разрешенных расширений.

```
① $allowed_types = ['image/jpeg', 'image/png', 'image/gif',];
② $type  = mime_content_type($_FILES['image']['tmp_name']);
③ $error = in_array($type, $allowed_types) ? '' : 'Wrong file type ';
④ $allowed_exts = ['jpeg', 'jpg', 'png', 'gif',];
⑤ $filename = strtolower($_FILES['image']['name']);
⑥ $ext      = pathinfo($filename, PATHINFO_EXTENSION));
⑦ $error   .= in_array($ext, $allowed_exts) ? '' : 'Wrong extension ';
```

ВАЛИДАЦИЯ ЗАГРУЗКИ ФАЙЛОВ

В этом примере собран код для загрузки, проверки и сохранения файла.

1. Шесть переменных создаются для записи:

- результата попытки загрузки файла;
- сообщения об успехе/сбое, отображаемого пользователю;
- текста ошибок при возникновении проблем с изображением;
- пути к папке, хранящей загруженные файлы;
- максимального размера файла в байтах;
- допустимых MIME-типов;
- допустимых расширений файлов.

2. Создается функция `create_filename()`. Она использует код для очистки имени файла и обеспечения его уникальности, а затем возвращает новое имя. Она использует два параметра:

- имя файла;
- относительный путь к папке, в которой он будет храниться.

3. Конструкция `if` проверяет, была ли отправлена форма.
4. Тернарный оператор проверяет, не было ли ошибки при загрузке изображения, если оно больше предельного размера, заданного в файлах `php.ini` или `.htaccess`. Если ошибка произошла, то сообщение о ней сохраняется в переменной `$error`.
5. Другая конструкция `if` проверяет, был ли файл загружен без каких-либо ошибок.
6. Валидация размера файла. Если он не превышает максимальный размер, сохраненный в переменной `$max_size` в шаге 1, то в переменную `$error` ничего не добавляется. Если размер превышает допустимый, то в `$error` добавляется сообщение `too big`.
7. Встроенная PHP функция `mime_content_type()` получает MIME-тип файла и сохраняет его в переменной `$type`.
8. Функция PHP `in_array()` сопоставляет содержимое переменной `$type` с типами в массиве `$allowed_types`. Если совпадение обнаружено, то к значению переменной `$error` добавляется пустая строка. Если нет, то добавляется сообщение об ошибке.
9. Функция PHP `pathinfo()` получает расширение файла загруженного изображения. Оно также обрабатывается функцией PHP `strtolower()`, которая переводит его в нижний регистр. Затем расширение сохраняется в переменной `$ext`.
10. Функция PHP `in_array()` используется для проверки допустимости использования расширения файла. Если расширение допустимо, то к значению переменной `$error` добавляется пустая строка. Если нет, добавляется сообщение, указывающее, что это неправильное расширение.
11. Условие в конструкции `if` проверяет, содержит ли переменная `$error` значение, которое не обрабатывается как `true`. Пустая строка обрабатывается как `false` (ошибок нет).
12. Если ошибок нет, вызывается функция `create_filename()` из шага 2. Она помогает убедиться, что имя файла очищено и является уникальным.
13. В переменную `$destination` пишется путь для сохранения нового файла.
14. Функция PHP `move_uploaded_file()` вызывается для перемещения файла из его временного местоположения в папку `uploads`. Она возвращает значение `true`, если операция успешна; `false`, если нет. Результат сохраняется в переменной с именем `$moved`.
15. Если переменная `$moved` содержит значение `true`, то изображение было загружено, успешно прошло проверку и было сохранено. И в результате в переменную `$message` записывается HTML-тег `<img>`, выводящий изображение.
16. В противном случае в эту переменную сохраняется сообщение об ошибке.
17. Значение, сохраненное в переменной `$message`, отображается перед формой загрузки.

PHP section_b/c07/validate-file.php

```php
<?php
$moved         = false;                          // Инициализация
$message       = '';                             // Инициализация
$error         = '';                             // Инициализация
$upload_path   = 'uploads/';                     // Путь загрузки
$max_size      = 5242880;                        // Максимальный размер файла (в байтах)
$allowed_types = ['image/jpeg', 'image/png', 'image/gif',];  // Допустимые типы файлов
$allowed_exts  = ['jpeg', 'jpg', 'png', 'gif',];        // Допустимые расширения файлов

function create_filename($filename, $upload_path)   // Функция для создания имени файла
{
    $basename   = pathinfo($filename, PATHINFO_FILENAME);    // Получение полного имени
    $extension  = pathinfo($filename, PATHINFO_EXTENSION);    // Получение расширения
    $basename   = preg_replace('/[^A-z0-9]/', '-', $basename); // Очистка полного имени
    $i          = 0;                                          // Счетчик
    while (file_exists($upload_path . $filename)) {           // Если файл существует
        $i        = $i + 1;                                   // Обновить счетчик
        $filename = $basename . $i . '.' . $extension;        // Новый путь к файлу
    }
    return $filename;                                         // Возврат имени файла
}
if ($_SERVER['REQUEST_METHOD'] == 'POST') {                   // Если форма отправлена
    $error = ($_FILES['image']['error'] === 1) ? 'too big ' : ''; // Ошибка проверки
                                                                  // размера
    if ($_FILES['image']['error'] == 0) {                // Если ошибок при загрузке нет
        $error .= ($_FILES['image']['size'] <= $max_size) ? '' : 'too big '; // Проверка размера
        // Проверка, что медиатип находится в массиве $allowed_types
        $type   = mime_content_type($_FILES['image']['tmp_name']);
        $error .= in_array($type, $allowed_types) ? '' : 'wrong type ';
        // Проверка, что расширение файла находится в массиве $allowed_exts
        $ext    = strtolower(pathinfo($_FILES['image']['name'], PATHINFO_EXTENSION));
        $error .= in_array($ext, $allowed_exts) ? '' : 'wrong file extension ';
        // Если ошибок нет, создается новый путь к файлу и попытка переместить файл
        if (!$error) {
          $filename    = create_filename($_FILES['image']['name'], $upload_path);
          $destination = $upload_path . $filename;
          $moved       = move_uploaded_file($_FILES['image']['tmp_name'], $destination);
        }
    }
    if ($moved === true) {                                    // Если он переместился
        $message = 'Uploaded:<br><img src="' . $destination . '">'; // Показать изображение
    } else {                                                  // В противном случае
        $message = '<b>Could not upload file:</b> ' . $error;   // Показать ошибку
    }
}
?> ... <?= $message ?> <!-- Show form -->
```

ИЗМЕНЕНИЕ РАЗМЕРА ИЗОБРАЖЕНИЙ

Сайты часто изменяют загруженные пользователями изображения, чтобы все они были одинакового размера. Это делает страницу более аккуратной и ускоряет ее загрузку. Чтобы изменить размер изображения, вам нужно знать его **соотношение сторон**: ширина, поделенная на высоту.

Размер загруженных изображений обычно меняют по двум причинам:

- когда все изображения на сайте одинакового размера, они выглядят аккуратнее, чем когда они все разных размеров;
- если искусственно изменить отображаемый размер изображения с помощью CSS, загружаться в браузер будет все равно файл большего размера, что замедлит загрузку страницы.

При изменении размера изображений вы должны сохранять одинаковое соотношение сторон (ширина, деленная на высоту), в противном случае измененные изображения будут выглядеть искаженными (см. страницу справа).
Если вы хотите, чтобы все изображения были строго одинакового размера, их можно обрезать (выделить часть), а затем изменить размер выделенного фрагмента, сохранив соотношение его сторон.

АЛЬБОМНОЕ ИЗОБРАЖЕНИЕ

У альбомных изображений ширина больше высоты, поэтому соотношение больше 1. В приведенном ниже примере, если ширина 2000, а высота — 1600, соотношение будет 2000 ÷ 1600 = 1,25.



КВАДРАТНОЕ ИЗОБРАЖЕНИЕ

У квадратных изображений высота и ширина одинаковы, поэтому соотношение равно 1. В приведенном ниже примере, если ширина 2000, а высота 2000, соотношение будет 2000 ÷ 2000 = 1.



КНИЖНОЕ ИЗОБРАЖЕНИЕ

У книжных (портретных) изображений ширина всегда меньше высоты, поэтому соотношение меньше 1. В приведенном ниже примере, если ширина 1600, а высота 2000, соотношение будет 1600 ÷ 2000 = 0,8.



Ниже показано, как определить новую ширину и высоту изображения при изменении его размера. При сохранении того же соотношения, что и в исходном изображении, измененное изображение не будет выглядеть искаженным.

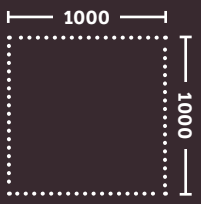
Чтобы изображения с измененными размерами выглядели более однородно, их размер изменяется так, чтобы они помещались внутри **квадратного контейнера** (или **ограничивающей рамки**).

Он задает максимальную ширину и высоту изображения. При изменении размера изображения более длинная сторона изображения (ширина *или* высота) заполнит ограничитель, а более короткая сторона будет изменена пропорционально.

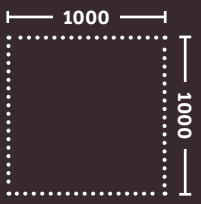
Ширина и высота контейнера должны быть определены как максимальная ширина и высота, которые может принять измененное изображение.

В этом примере максимальные ширина и высота установлены равными 1000.

АЛЬБОМНОЕ ИЗОБРАЖЕНИЕ



КНИЖНОЕ ИЗОБРАЖЕНИЕ



1
Получите ширину и высоту исходного загруженного изображения.
Используйте их для вычисления соотношения сторон (ширина ÷ высота).



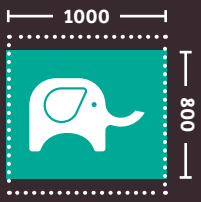


2
Если ширина больше высоты, изображение будет альбомным.
В противном случае изображение будет книжным.
Установите более длинную сторону изображения в соответствии с размером контейнера.

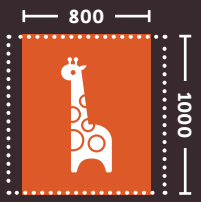




3
Вычислите длину более короткой стороны измененного изображения.
Альбомное: *разделите* высоту контейнера на соотношение.
Книжное: *умножьте* ширину контейнера на соотношение.



Альбомное изображение не будет заполнять всю высоту контейнера.



Книжное изображение не будет заполнять всю ширину контейнера.

ОБРЕЗКА ИЗОБРАЖЕНИЙ

Обрезка изображений позволяет создавать изображения одинакового размера, а также позволяет новым изображениям полностью занять ограничивающую рамку. При обрезке изображений часть исходного изображения удаляется.

Чтобы обрезать изображение, вам нужно выбрать часть исходного изображения, которую вы хотите сохранить.

Чтобы придать набору изображений одинаковую форму, обрезанная часть каждого изображения должна быть с одинаковым соотношением сторон.

После того как вы выбрали область для обрезки, можно изменить ее размер, чтобы убедиться, что у всех изображений он одинаковый.

Для выбора области изображения, которую вы хотите обрезать, вам понадобятся четыре фрагмента данных:

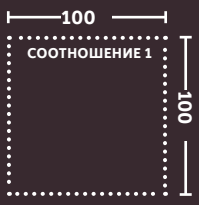
- ширина выделенной области: ширина области на изображении, которую вы хотите сохранить от смещения по оси X;
- высота выделенной области: высота области на изображении, которую вы хотите сохранить от смещения по оси Y;
- смещение по оси X: расстояние от левой части изображения до того места, где должно начинаться выделение;
- смещение по оси Y: расстояние от верхней части изображения до того места, где должно начинаться выделение.

Существуют инструменты JavaScript, позволяющие пользователям обрезать изображения в браузере перед загрузкой изображения. Некоторые из доступных вариантов см. **http://notes.re/php/images/crop-javascript**.

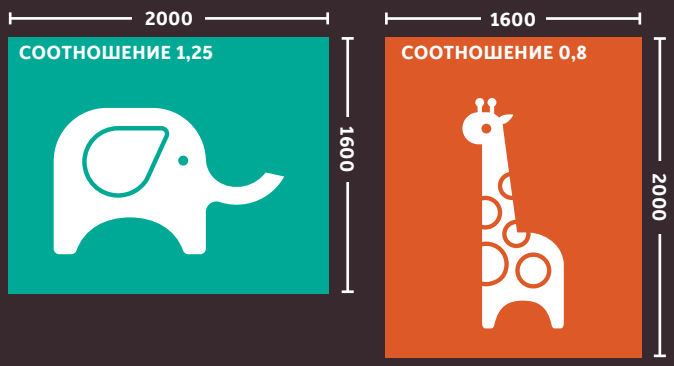




Чтобы обеспечить всем сохраняемым изображениям одинаковые размеры, вам необходимо указать требуемую ширину и высоту. Эти значения будут использоваться для вычисления соотношения сторон нового изображения (ширина ÷ высота).



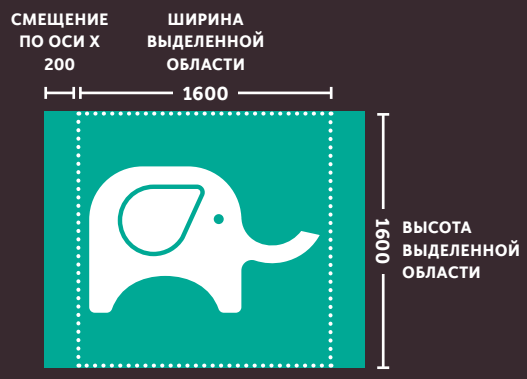
1
Получите ширину и высоту загруженного изображения и рассчитайте соотношение сторон загруженного изображения (ширина ÷ высота).

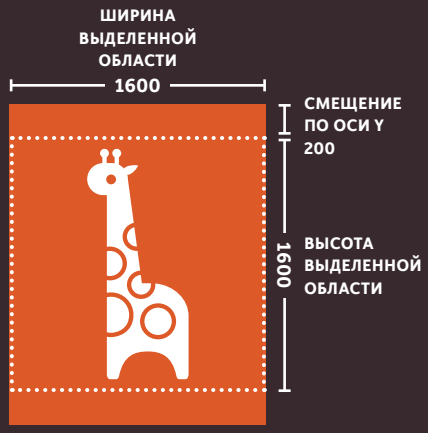


2
Выберите соответствующую часть изображения для сохранения.

Выделенная область должна иметь то же соотношение сторон, что и новое изображение.

Расчеты для выделенных областей и смещения показаны ниже.





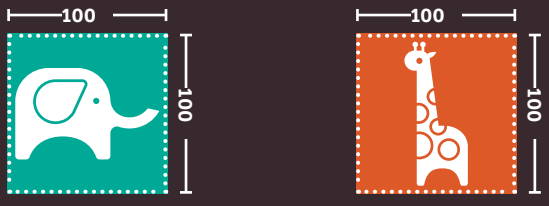
Если соотношение сторон нового изображения меньше, чем соотношение загруженного изображения:

- ширина выделенной области = исходная высота × новое соотношение;
- высота выделенной области = исходная высота;
- смещение по оси X = (исходная ширина — ширина выделения) / 2;
- смещение по оси Y = 0.

В противном случае используйте следующие вычисления:

- ширина выделенной области = исходная высота;
- высота выделенной области = исходная высота × новое соотношение;
- смещение по оси X = 0;
- смещение по оси Y = (исходная ширина — ширина выделения) / 2.

3
Размер обрезанной области изменяется таким образом, чтобы она соответствовала размеру нового изображения (как определено на странице слева).



РЕДАКТИРОВАНИЕ ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ РАСШИРЕНИЙ

Расширения (или модули) добавляют функциональность интерпретатору PHP, позволяя ему выполнять дополнительные задачи. GD и Imagick — это два популярных расширения, позволяющих интерпретатору PHP изменять размер и обрезать изображения.

Когда расширение активируется на веб-сервере, оно обычно предоставляет дополнительные функции или классы. Их можно использовать в ваших PHP-скриптах точно так же, как ваш код использует встроенные функции и классы PHP.

Расширения GD и Imagick выполняют задачи, аналогичные базовым функциям Photoshop, но вместо того чтобы управлять изображениями с помощью графического пользовательского интерфейса, они позволяют редактировать изображение с помощью PHP-кода.

В остальной части этой главы объясняется, как изменять размер изображений и обрезать их с помощью GD, а затем с Imagick. Пример обрезки изображений с помощью GD доступен онлайн **http://notes.re/php/gd-crop**.

Расширение GD немного сложнее в использовании, чем Imagick, но оно было добавлено по умолчанию в интерпретатор PHP начиная с версии 4.3. В то время как для расширения Imagick требуется, чтобы на веб-сервере была установлена программа ImageMagick.

РАСШИРЕНИЕ GD

Если вы используете MAMP на Mac, расширение GD должно быть доступно по умолчанию.

Если вы используете XAMPP на ПК, вам, вероятно, потребуется включить расширение GD, прежде чем вы сможете его использовать, см. **http://notes.re/php/enable-gd**.

Чтобы изменить размер и обрезать изображение с помощью GD, вам понадобится пять функций GD (показаны ниже).

В GD имеется набор функций для открытия файлов различных графических форматов (GIF, JPEG, PNG, WEBP и т. д.) и соответствующие функции для их сохранения (выделенный курсивом *mediatype* ниже необходимо заменить типом файла — см. правую страницу).

ФУНКЦИЯ	ОПИСАНИЕ
`getimagesize()`	Получает размеры и MIME-тип изображения
`imagecreatefrom`*`mediatype`*`()`	Открывает изображение (замените *`mediatype`* на тип изображения)
`imagecreatetruecolor()`	Создает новое пустое изображение, с размерами, которые должно иметь измененное или обрезанное изображение
`imagecopyresampled()`	Получает выделенную часть исходного изображения, изменяет ее размер и вставляет в новое изображение, созданное на предыдущем шаге
`image`*`mediatype`*`()`	Сохраняет изображение (замените *`mediatype`* на тип изображения)

ОПРЕДЕЛЕНИЕ MIME-ТИПА

Чтобы выбрать правильную функцию для открытия или сохранения изображения, вам необходимо знать тип изображения.

Функция GD `getimagesize()` требует в качестве аргумента путь к изображению. Она возвращает массив, содержащий данные об изображении, включая его тип.

В таблице справа показаны данные, хранящиеся в этом массиве (ключи представляют собой смесь цифр и слов).

КЛЮЧ	ОПИСАНИЕ
`0`	Ширина изображения (в пикселях)
`1`	Высота изображения (в пикселях)
`2`	Константа, описывающая тип изображения
`3`	Строка с размерами для использования в теге `<img>: height="yyy" width="xxx"`
`mime`	MIME-тип изображения
`channels`	3, если изображение в формате RGB; 4, если в формате CMYK
`bits`	Количество битов, используемых для каждого цвета

ОТКРЫТИЕ И СОХРАНЕНИЕ ИЗОБРАЖЕНИЙ

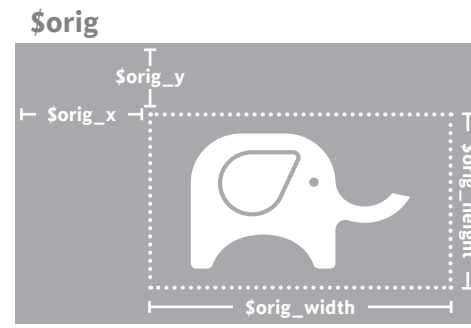
MIME-тип изображения можно использовать в конструкции *switch* (как показано на следующей странице), чтобы вызвать нужную функцию для открытия или сохранения изображения.
В таблице справа показаны некоторые функции, предлагаемые GD для открытия и сохранения форматов изображений.

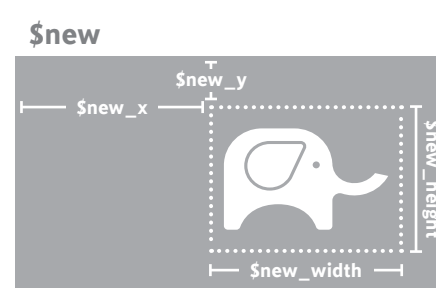
ФОРМАТ	ОТКРЫТЬ	СОХРАНИТЬ
GIF	`imagecreatefromgif()`	`imagegif()`
JPEG	`imagecreatefromjpeg()`	`imagejpeg()`
PNG	`imagecreatefrompng()`	`imagepng()`
WEBP	`imagecreatefromwebp()`	`imagewebp()`

ИЗМЕНЕНИЕ РАЗМЕРА И ОБРЕЗКА ИЗОБРАЖЕНИЙ

Функция `imagecopyresampled()` копирует изображение (или его часть) в новое пустое изображение.
Для этого у функции есть 10 параметров, но проще представить их как 5 пар параметров.

- `$new`, `$orig`
  Новое и исходное изображение (сохраняются в переменных перед вызовом функции).
- `$new_x`, `$new_y`
  Смещение по осям X и Y, определяющее, где должна располагаться скопированная область в новом изображении.
- `$orig_x`, `$orig_y`
  Смещение по осям X и Y, где функция должна получить выделенную из исходного изображения область.
- `$new_width`, `$new_height`
  Ширина и высота выделенной области в новом изображении.
- `$orig_width`, `$orig_height`
  Ширина и высота выделенной области из исходного изображения.






```
imagecopyresampled($new, $orig, $new_x, $new_y, $orig_x, $orig_y,
                   $new_width, $new_height, $orig_width, $orig_height);
```

ИЗМЕНЕНИЕ РАЗМЕРА ИЗОБРАЖЕНИЙ С ПОМОЩЬЮ GD

Функция на странице справа использует GD для создания миниатюр. Миниатюра сохранит то же соотношение сторон, что и исходное изображение. Новый размер основан на максимальной ширине и высоте, заданных в качестве аргументов.

Код этого примера в основном соответствует нашему прошлому коду из примера. Различия заключаются в том, что здесь создается путь для уменьшенного изображения миниатюры, а затем, после перемещения загруженного изображения, вызывается функция для создания миниатюры (шаги 14–15).

1. Функция `resize_image_gd()` имеет четыре параметра:

- путь к загруженному изображению;
- путь для сохранения измененного изображения;
- максимальная ширина нового изображения;
- максимальная высота нового изображения.

2. Функция GD `getimagesize()` возвращает массив, содержащий данные об изображении, включая его размеры и MIME-тип (см. предыдущую страницу).
3. Ширина, высота и тип изображения берутся из массива и сохраняются в переменных.
4. В переменные `$new_width` и `$new_height` записываются максимально возможные ширина и высота для миниатюры.
5. Переменная `$orig_ratio` содержит соотношение сторон загруженного изображения.
6. Если ширина больше высоты, изображение будет альбомным.
7. Для альбомного изображения ширина измененного изображения будет максимальной шириной. Это значение было задано при инициализации переменной в шаге 4. Но новую высоту изображения нужно рассчитать, для этого разделите ширину изображения на соотношение его сторон.
8. В противном случае изображение будет книжным или квадратным. Его высота остается максимальной высотой, заданной в шаге 4. Новая ширина вычисляется путем умножения новой высоты изображения на соотношение его сторон.
9. Конструкция `switch` используется при выборе нужной функции для открытия изображения. Как показано на предыдущей странице, GD использует отдельные функции для открытия изображений, относящихся к разным типам носителей. MIME-тип изображения (сохраненный в `$media_type` в шаге 3) используется в качестве условия в конструкции `switch`. Изображение, открытое с помощью выбранной функции, сохраняется в переменной `$origin`.
10. Функция GD `imagecreatetruecolor()` создает пустое изображение, сохраняемое в переменной `$new`. Два аргумента при вызове этой функции — это ширина и высота, которые должны быть у нового изображения.
11. Функция GD `imagecopyresampled()` копирует исходное изображение, изменяет его размер и вставляет в новое изображение, созданное в шаге 10. Ему необходимо передать значения для всех 10 параметров, описанных на предыдущей странице.
12. Конструкция `switch` используется при выборе нужной функции для сохранения измененного изображения. На этот раз применяется сокращенная форма конструкции `switch`, чтобы пример поместился на странице. Функции, сохраняющие изображения, возвращают значение `true`, если изображение сохранено, и `false`, если нет. Итоговое значение сохраняется в `$result`.
13. Функция возвращает значение, сохраненное в `$result`.
14. После того как изображение было загружено и перемещено, путь, по которому будет сохранено новое уменьшенное изображение, создается путем объединения пути к папке загрузки, текста `thumb_` и имени файла.
15. Вызов `resize_image_gd()`.

PHP section_b/c07/resize-gd.php

```php
<?php
function resize_image_gd($orig_path, $new_path, $max_width, $max_height)
{
    $image_data  = getimagesize($orig_path);    // Получение данных изображения
    $orig_width  = $image_data[0];              // Ширина изображения
    $orig_height = $image_data[1];              // Высота изображения
    $media_type  = $image_data['mime'];         // MIME-тип
    $new_width   = $max_width;                  // Максимальное значение новой ширины
    $new_height  = $max_height;                 // Максимальное значение новой высоты
    $orig_ratio  = $orig_width / $orig_height;  // Соотношение сторон исходного файла

    // Вычисление нового размера
    if ($orig_width > $orig_height) {            // Если изображение альбомное
        $new_height = $new_width / $orig_ratio;  // Установка высоты, с соотношением сторон
    } else {                                     // В противном случае
        $new_width  = $new_height * $orig_ratio; // Установка ширины, с соотношением сторон
    }

    switch($media_type) {                            // Проверка медиатипа
        case 'image/gif' :                           // Если это GIF-файл
            $orig = imagecreatefromgif($orig_path);  // Эта функция открывает изображение
            break;                                // останавливает дальнейшее выполнение switch
        case 'image/jpeg' :                          // Если это JPG
            $orig = imagecreatefromjpeg($orig_path); // Эта функция открывает изображение
            break;                               // останавливает дальнейшее выполнение switch
        case 'image/png' :                           // Если это PNG
            $orig = imagecreatefrompng($orig_path);  // Эта функция открывает изображение
            break;                               // останавливает дальнейшее выполнение switch
    }

    $new = imagecreatetruecolor($new_width, $new_height); // Создание пустого изображения

    imagecopyresampled($new, $orig, 0, 0, 0, 0, $new_width, $new_height,
        $orig_width, $orig_height);      // Копирование исходного изображения в новый файл

    // Сохранение изображения. Папка thumbs должна быть создана + правильные разрешения
    switch($media_type) {
        case 'image/gif' : $result = imagegif($new, $new_path);  break;
        case 'image/jpeg': $result = imagejpeg($new, $new_path); break;
        case 'image/png' : $result = imagepng($new, $new_path);  break;
    }
    return $result;
}
$moved     = move_uploaded_file($_FILES['image']['tmp_name'], $destination); // Перемещение файла
$thumbpath = $upload_path . 'thumb_' . $filename;                // Создание пути к миниатюре
$resized   = resize_image_gd($destination, $thumbpath, 200, 200); // Создание миниатюры
```

ИЗМЕНЕНИЕ РАЗМЕРА И ОБРЕЗКА С ПОМОЩЬЮ IMAGICK

Расширение Imagick PHP требует для работы установленную программу для редактирования изображений с открытым исходным кодом под названием ImageMagick, позволяя обращаться к этой программе через функции PHP. Расширение Imagick:

- требует гораздо меньше кода, чем GD;
- само вычисляет соотношения и параметры для изменения размера изображений;
- использует одни и те же методы для всех форматов изображений;
- поддерживает больше форматов изображений, чем GD.

Но для этого требуется, чтобы на веб-сервере были установлены как расширение Imagick, так и программа ImageMagick. По умолчанию они не устанавливаются. Поэтому вам понадобится:

- включить Imagick для MAMP на компьютере с MacOS;
- установить Imagick + ImageMagick для XAMPP на ПК, см. **http://notes.re/php/install-imagick**;
- проверить, поддерживает ли его ваш хостинг-провайдер.

Чтобы использовать Imagick, вы создаете объект, который будет представлять изображение, используя класс `Imagick`, и передаете в конструктор путь к изображению, с которым он будет работать.

ПЕРЕМЕННАЯ ДЛЯ ОБЪЕКТА — ИМЯ КЛАССА — ПУТЬ К ИЗОБРАЖЕНИЮ

```
$image = new Imagick($filepath);
```

Создаваемый объект `Imagick` содержит набор методов, управляющих изображением и сохраняющих его. В частности:

МЕТОД	ОПИСАНИЕ
`thumbnailImage()`	Изменение размера изображения
`cropThumbnailImage()`	Обрезка и изменение размера изображения
`writeImage()`	Сохранение изображения

На ПК путь, который Imagick использует для сохранения файла, должен быть абсолютным[38], а абсолютные пути на ПК отличаются на Mac и Unix:

- на ПК пути начинаются с буквы диска, например, `c:/`;
- на Mac и Unix они начинаются с обратного слеша \.

Разделитель директорий также отличается: на ПК это слеш, на Mac и Unix — обратный слеш.

Чтобы создать правильный путь к директории загрузки и сохранить его в переменной, используется приведенный ниже код.
В инструкции ниже используется:

- Функция PHP `dirname()`[39]. Возвращает путь к содержащей файл директории, указанной в качестве аргумента.
- Константа `__FILE__`. Содержит путь к файлу, выполняемому в данный момент.
- Константа `DIRECTORY_SEPARATOR`. Содержит правильный разделитель директорий для операционной системы, используемой для запуска файлов PHP.

ТЕКУЩИЙ ФАЙЛ — РАЗДЕЛИТЕЛЬ КАТАЛОГОВ

```
$upload_path = dirname(__FILE__) . DIRECTORY_SEPARATOR . 'uploads' . DIRECTORY_SEPARATOR;
```

ПУТЬ К РОДИТЕЛЬСКОМУ КАТАЛОГУ

В двух приведенных ниже примерах показаны две пользовательские функции, которые используют Imagick для изменения размера и обрезки изображений.

Инструкции, вызывающие эти функции, вызываются сразу после кода, перемещающего загруженный файл в пункт назначения.

1. Функция `create_thumbnail()` создает миниатюру изображения с помощью Imagick. У нее два параметра:

- путь к только что загруженному изображению;
- путь для новой миниатюры, создаваемой Imagick.

2. С помощью класса `Imagick` создается новый объект `$image`. Ему нужен путь к загруженному изображению.

3. Метод `thumbnailImage()` объекта `$image` изменяет размер изображения. Для этого он использует три аргумента:

- новая ширина изображения;
- новая высота изображения;
- логическое значение `true`, чтобы сообщить Imagick, что ширина и высота — это максимальные значения и что миниатюра должна быть в том же соотношении, что и оригинал.

4. Метод Imagick `writeImage()` сохраняет изображение в местоположение, указанное в параметре `$destination`.

5. Функция возвращает значение `true`, подтверждая, что выполнена.

6. После перемещения файла переменная `$thumbpath` сохраняет путь, по которому будет расположена новая миниатюра.

7. Вызывается функция `create_thumbnail()`, ей передается путь к загруженному изображению и путь к миниатюре.

PHP section_b/c07/resize-im.php

```
① function create_thumbnail($temporary, $destination)
  {
②     $image = new Imagick($temporary);        // Объект для представления изображения
③     $image->thumbnailImage(200, 200, true);  // Создание миниатюры
④     $image->writeImage($destination);        // Сохранение файла
⑤     return true;        // Возврат значения true, чтобы показать успешное выполнение
  } ... // Как только файл проверен и перемещен, создается путь, затем сама миниатюра
  $moved = move_uploaded_file($_FILES['image']['tmp_name'], $destination); // Перемещение
⑥ $thumbpath = $upload_path . 'thumb_' . $filename;          // Путь к миниатюре
⑦ $thumb = create_thumbnail($destination, $thumbpath);   // Создание миниатюры
```

8. Функция `create_cropped_thumbnail()` создает квадратную обрезку загруженного изображения. Это гарантирует, что все миниатюры будут одинакового размера.

9. Единственное отличие от приведенного выше примера заключается в том, что в этом варианте используется метод `cropThumbnailImage()` объекта `Imagick` для создания обрезанной миниатюры.

PHP section_b/c07/crop-im.php

```
⑧ function create_cropped_thumbnail($temporary, $destination)
  {
      $image = new Imagick($temporary);           // Объект для представления изображения
⑨     $image->cropThumbnailImage(200, 200, true); // Создание миниатюры
      $image->writeImage($destination);           // Сохранение файла
      return true;          // Возврат значения true, чтобы показать успешное выполнение
  }
```

ЗАКЛЮЧЕНИЕ
ИЗОБРАЖЕНИЯ И ФАЙЛЫ

- Через HTML-форму можно загружать на сервер файлы с помощью поля для загрузки файлов.
- Когда файл загружен, суперглобальный массив `$_FILES` содержит данные о нем.
- При загрузке файлы помещаются во временное хранилище, затем они должны быть перемещены в другую папку для сохранения.
- Прежде чем начать работу с файлом, убедитесь, что он был загружен по протоколу HTTP и не было ошибок.
- Убедитесь, что в имени файла используются только разрешенные символы.
- Проведите валидацию размера, MIME-типа и расширения загруженных файлов перед их сохранением.
- При изменении размера изображения сохраняйте существующее соотношение сторон, в противном случае оно будет выглядеть растянутым и искаженным.
- GD и Imagick — это расширения, позволяющие изменять размер и обрезать изображения на сервере с помощью PHP.

8

ДАТА И ВРЕМЯ

Обратите внимание, что работа с датами и временем происходит на английском языке. Для вывода даты с учетом локали нужна функция `strftime`, ее в книге нет. Поэтому перевод дней недели и месяцев стоит давать в качестве справочного материала, а не замены текста.

Дату и время можно записать самыми разными способами.
PHP предоставляет встроенные функции и классы, помогающие обрабатывать и отображать дату и время в различных форматах.

В этой главе вы узнаете о различных способах, с помощью которых интерпретатор PHP может принимать дату и время в качестве входных данных, а также как он может форматировать их в качестве выходных данных при отображении посетителям.

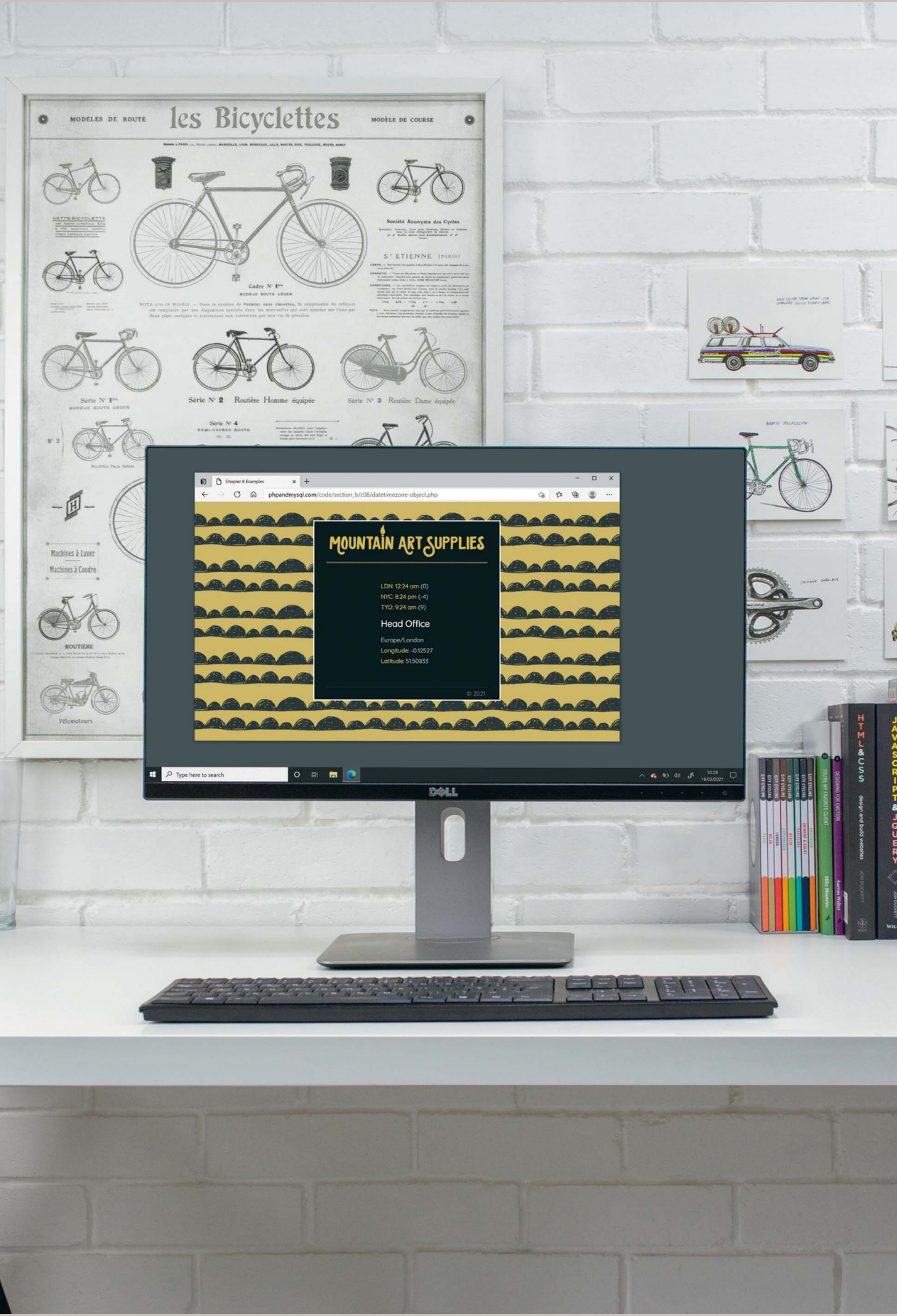
В PHP можно задать определенные дату и время тремя способами:

- указав отдельно год, месяц, день, час, минуту и секунду;
- используя словесный формат, например, «1 июня 2001 года», «1/6/2001» или «следующий вторник» (правда, на английском языке);
- с помощью метки времени Unix, которая представляет собой количество секунд, прошедших с 1 января 1970 года (это может показаться странным способом представления даты/времени, но многие языки программирования используют его).

После изучения того, как PHP обрабатывает форматы даты и времени, вы познакомитесь с набором встроенных функций, генерирующих метки времени Unix и преобразующих их обратно в удобочитаемый формат.

Затем вы узнаете, как дата и время могут быть представлены в виде объектов, созданных с использованием четырех встроенных классов:

- `DateTime` создает объекты, представляющие определенную дату и время;
- `DateInterval` создает объекты, представляющие интервал времени (например, час или неделю);
- `DatePeriod` создает объекты, представляющие повторяющиеся события, которые происходят через регулярные промежутки времени (например, каждый день, месяц или год);
- `DateTimeZone` создает объекты, представляющие часовой пояс.


MODÈLES DE ROUTE
les Bicyclettes
MODÈLE DE COURSE
Société Anonyme des Cycles
ST ETIENNE (PARIS)
Série N° 2 Routière Homme équipée
Série N° 3 Routière Dame équipée
Machines à Laver
Machines à Coudre
ROUTIÈRE
Vélomoteurs
Chapter 8 Examples
phpandmysql.com/code/section_b/c08/datetimezone-object.php
MOUNTAIN ART SUPPLIES
LDN: 12:24 am (0)
NYC: 8:24 pm (-4)
TYO: 9:24 am (9)
Head Office
Europe/London
Longitude: -0.12527
Latitude: 51.50833
© 2021
Type here to search
DELL
HTML&CSS
design and build websites
JAVASCRIPT
DESIGNING FOR EMOTION
YOU'RE MY FAVORITE CLIENT

ФОРМАТЫ ДАТЫ

Дата может отображаться различными способами. PHP использует набор символов в определенном формате для описания того, как записывается дата.

Дата может состоять из следующих компонентов:

- день недели;
- день месяца;
- месяц;
- год.

PHP использует символы определенного формата для представления каждого из этих компонентов. Например, **m-d-Y** отформатирует дату в виде 04-26-2022. Символы формата сообщают интерпретатору PHP, как именно дату необходимо обрабатывать при чтении и форматировать при отображении.

Вы можете добавить пробелы, косую черту, тире и точки между символами формата, чтобы визуально разделить каждый компонент.

Ниже представлено, как символы формата описывают разные способы записи одной и той же даты:

ФОРМАТ СИМВОЛОВ	ФОРМАТЫ ДАТЫ
`l m j Y`	Saturday April 26 2022 (Суббота Апрель 6 2022)
`D jS F Y`	Sat 6th April 2022 (Суб 26-е Апрель 2022)
`n/j/Y`	4/26/2022
`m/d/y`	04/26/22
`m-d-Y`	04-26-2022

ДЕНЬ НЕДЕЛИ

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
`D`	Первые три буквы	Sat (Суб)
`l`	Полное имя	Saturday (Суббота)

ДЕНЬ МЕСЯЦА

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
`d`	Цифры, начинающиеся с нуля	09
`j`	Цифры без нуля в начале	9
`S`	Буквенное наращение	th (–е)

МЕСЯЦ

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
`m`	Цифры, начинающиеся с нуля	04
`n`	Цифры без нуля в начале	4
`M`	Первые три буквы	Apr (Апр)
`F`	Полное имя	April (Апрель)

ГОД

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
`Y`	Четыре цифры	2022
`y`	Две цифры	22

ФОРМАТЫ ВРЕМЕНИ

Эти символы формата могут использоваться для представления различных способов отображения времени.

ЧАС

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
h	12-часовой с начальным нулем	08
g	12-часовой без начального нуля	8
H	24-часовой с начальным нулем	08
G	24-часовой без начального нуля	8

МИНУТА

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
i	Цифры, начинающиеся с нуля	09

СЕКУНДА

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
s	Цифры, начинающиеся с нуля	04

AM/PM

СИМВОЛ	ОПИСАНИЕ	ПРИМЕР
a	Строчные	am
A	Заглавные	AM

Время может состоять из следующих компонентов:

- часы;
- минуты;
- секунды;
- am/pm (если не используется 24-часовой формат).

Каждый из них может быть представлен с помощью символов формата. Например, **g: i a** представляет время в формате «8:09 am». Эти символы формата используются для указания интерпретатору PHP, как показания времени будут обрабатываться при получении и форматироваться при отображении.

Можно добавить пробелы, двоеточия, точки и круглые скобки между символами формата, чтобы визуально разделить каждый компонент.

Ниже представлено, как символы формата могут описывать разные способы записи одного и того же времени:

ФОРМАТ СИМВОЛОВ	ФОРМАТЫ ДАТЫ
g: i a	8:09 am
h: i(A)	08:09(AM)
H: i	08:09

УКАЗАНИЕ ДАТЫ И ВРЕМЕНИ С ПОМОЩЬЮ СТРОК

Некоторые функции и методы позволяют указывать дату и время с помощью строки. Строка должна быть в одном из принятых форматов (они описаны ниже).

Интерпретатор PHP может принимать даты, используя следующие строковые форматы. Ниже в таблице представлено, как можно написать дату (справа), и как ее воспримет PHP (слева). Если используются косые черты, интерпретатор PHP ожидает, что на первом месте будет стоять месяц, а на втором — день. Если используются тире или точки, то наоборот — сначала день, потом месяц.

ФОРМАТЫ ДАТЫ	ПРИМЕР
`d F Y`	04 September 2022 (04 сентября 2022)
`jS F Y`	4th September 2022 (4-е сентября 2022)
`F j Y`	September 4 2022 (Сентябрь 4 2022)
`M d Y`	Sep 04 2022 (Сен 04 2022)
`m/d/Y`	09/04/2022
`Y/m/d`	2022/09/04
`d-m-Y`	04-09-2022
`n-j-Y`	9-4-2022
`d.m.y`	04.09.22

Вы также можете использовать относительное время справа. Например:
`+ 1 day (1 день);`
`+ 3 years 2 days 1 month (3 года 2 дня 1 месяц);`
`- 4 hours 20 mins (4 часа 20 минут);`
`next Tuesday (следующий вторник);`
`first Sat of Jan (первая суббота января);`
`First/Last (первый/последний; работает только с днями месяца).`
Если точное время не указано, оно будет установлено на полночь.

Интерпретатор PHP может принимать значения времени, используя значения в следующих строковых форматах (указаны справа). Допускается:
использовать заглавные или строчные буквы для am и pm;
использовать букву **t**, чтобы отделить дату от времени;
добавлять часовой пояс в конце.

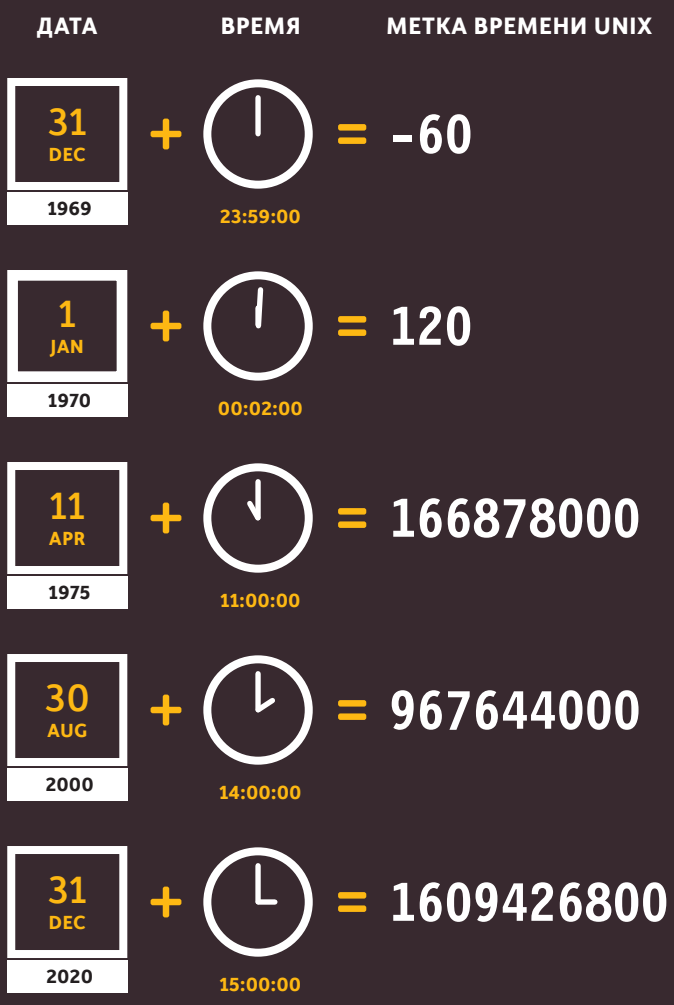
12-ЧАСОВОЙ ФОРМАТ ВРЕМЕНИ	ПРИМЕР
`Ga`	4am
`g: i a`	4:08 am
`g: i: s a`	4:08:37 am
`g.i.s a`	4.08.37 am

24-ЧАСОВОЙ ФОРМАТ ВРЕМЕНИ	ПРИМЕР
`H: i`	04:08
`H: i: s`	04:08:37
`His`	040837
`H.i.s`	04.08.37

ТИП	ОТНОСИТЕЛЬНОЕ ВРЕМЯ
Прибавить/отнять	`+ –`
Значение	`0-9`
Единицы измерения времени (может быть множественным числом)	`day, fortnight, month, year, hour, min, minute, sec, second (день, две недели, месяц, год, час, минута, минута, секунда, секунда)`
Названия дней	`Monday – Sunday and Mon – Sun (Понедельник – Воскресенье)`
Относительные термины	`next, last, previous, this (следующий, последний, предыдущий, текущий)`
Порядковые термины	`first – twelfth (первый – двенадцатый)`

МЕТКА ВРЕМЕНИ UNIX

Метка времени Unix позволяет представить дату и время, используя количество секунд, прошедших с полуночи 1 января 1970 года.



Интерпретатор PHP позволяет указывать и извлекать дату и время, используя метки времени Unix.

Слева вы можете увидеть конкретные примеры дат и времени, за которыми следует соответствующая им метка времени Unix.

Даты до 1 января 1970 года записываются с использованием отрицательных чисел.

Ниже вы узнаете, как встроенные функции и классы PHP позволяют работать с метками времени Unix. Они используют те же самые символы форматирования, с которыми вы только что познакомились, чтобы преобразовывать метку времени Unix во что-то понятное для человека.

Максимальная дата для метки времени Unix — 19 января 2038 года.

Unix — это операционная система, которая была разработана в 1970-е годы.

ВСТРОЕННЫЕ ФУНКЦИИ ДАТЫ И ВРЕМЕНИ

PHP содержит встроенные функции, которые могут создавать метки времени Unix для представления даты и времени. В нем также есть встроенные функции для преобразования этих меток времени Unix в легко читаемый формат.

Все три функции, приведенные ниже, используются для создания метки времени Unix.

При невозможности создать метку времени возвращается значение `false`.

Если вы не укажете время для `strtotime()` или `mktime()`, время устанавливается равным полуночи.

ФУНКЦИЯ	ОПИСАНИЕ
`time()`	Возвращает текущие дату и время в виде метки времени Unix

`strtotime($string)`	Преобразует строку в метку времени Unix
	ПРИМЕР
	`strtotime('December 1 2020');`
	`strtotime('1/12/2020');`

`mktime(H, i, s, n, j, Y)`	Преобразует указанные в аргументах компоненты даты/времени в метку времени Unix

ПРИМЕР	ВЫВОДИТСЯ В ВИДЕ
`mktime(17, 01, 05, 2, 1, 2001);`	February 1 2001 17:01:05
`mktime(01, 30, 45, 4, 29, 2020);`	April 29 2020 01:30:45

Функция `date()` преобразует метку времени Unix в удобочитаемый формат. Если метка времени не указана, отображаются текущие дата и время.

ФУНКЦИЯ	ОПИСАНИЕ
`date($format[, $timestamp])`	Возвращает метку времени Unix, отформатированную в удобочитаемом виде. Первый параметр определяет, как должна быть отформатирована дата. Второй параметр — это метка времени Unix для форматирования

ПРИМЕР	ВЫВОД РЕЗУЛЬТАТА
`date('Y');`	Текущий год
`date('d-m-y h:i a', 1609459199);`	31-12-20 11:59 pm
`date('D j M Y H:i:a', 1609459199);`	Thu 31 Dec 2020 23:59:59

ФУНКЦИИ ДАТЫ

PHP section_b/c08/date-functions.php

```php
<?php
① $start      = strtotime('January 1 2021');
② $end        = mktime(0, 0, 0, 2, 1, 2021);
③ $start_date = date('l, d M Y', $start);
  $end_date   = date('l, d M Y', $end);
?>
<?php include 'includes/header.php'; ?>

④ <p><b>Sale starts:</b> <?= $start_date ?></p>
  <p><b>Sale ends:</b> <?= $end_date ?></p>

<?php include 'includes/footer.php'; ?>
```

PHP section_b/c08/includes/footer.php

```php
⑤ <footer>&copy; <?php echo date('Y')?></footer> ...
```

РЕЗУЛЬТАТ



1. Функция `strtotime()` создает метку времени Unix, которая представляет дату в прошлом. Она присваивается переменной с именем `$start`.
2. Функция `mktime()` создает метку времени Unix для представления даты на месяц вперед. Она сохраняется в переменной с именем `$end`.
3. Функция `date()` преобразует эти метки времени Unix в удобочитаемый формат, используя:

- название дня недели;
- день (с начальным нулем);
- месяц (первые три буквы);
- год (четыре цифры).

Они сохраняются в переменных `$start_date` и `$end_date`.
4. Отображается удобочитаемая версия каждой даты.
5. Подключаемый файл для футера добавляет уведомление об авторских правах. Год выводится с помощью функции `date()`. Поскольку метка времени не указана, используется текущая дата.

Упражнение. В шаге 2 изменить дату и время на полдень следующей недели. В шаге 3 измените формат даты на Mon 1st February 2021 (Пн, 1 февраля 2021 года).

Примечание. Если PHP отображает текущее время с разницей в несколько часов, проверьте настройку часового пояса по умолчанию в файле `php.ini`.

КЛАССЫ ДЛЯ ПРЕДСТАВЛЕНИЯ ДАТЫ И ВРЕМЕНИ

Встроенный в PHP класс `DateTime` создает объект, представляющий дату и время. Его методы могут возвращать содержащиеся в объекте дату и время либо в удобочитаемом формате, либо в виде метки времени Unix.

Чтобы создать объект `DateTime`, напишите:

- переменную для хранения объекта;
- оператор присвоения;
- ключевое слово `new`;
- имя класса `DateTime`;
- пару круглых скобок.

В круглых скобках укажите дату/время, которые должен представлять объект.

Вы можете использовать любой из форматов даты и времени, которые были показаны ранее. Значение должно быть заключено в кавычки.

Если вы не укажете дату и время, объект будет использовать текущие.

Если вы укажете дату, но не время, объект будет использовать полночь в указанный день.

```
$date = new DateTime('2001-02-01 15:01:05');
```

ПЕРЕМЕННАЯ — ИМЯ КЛАССА — ДАТА И ВРЕМЯ

Вы также можете использовать функцию `date_create_from_format()` для создания объекта `DateTime`.

Первый аргумент — это формат, в котором будут указаны дата и время.

Второй аргумент — это дата и время в указанном формате. Оба аргумента заключены в кавычки.

```
$date = date_create_from_format('j-M-Y', '15-Jan-2020');
```

ПЕРЕМЕННАЯ — ФУНКЦИЯ — ФОРМАТ — ДАТА/ВРЕМЯ

Методы класса `DateTime`, приведенные ниже, возвращают дату и время, представленные в объекте.

Для получения даты/времени в удобочитаемом формате используйте метод `format()`.

Чтобы получить дату/время в качестве метки времени Unix, используйте метод `getTimestamp()`.

МЕТОД	ОПИСАНИЕ
`format(`*`$format[, $DateTimeZone]`*`)`	Возвращает дату и время в указанном формате Необязательный второй параметр задает часовой пояс
`getTimestamp()`	Возвращает метку времени Unix для даты и времени, которые представляет объект

КЛАСС DateTime

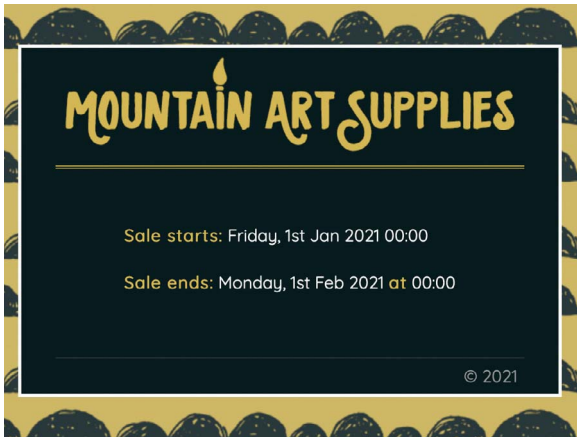
PHP section_b/c08/datetime-object.php

```
<?php
① $start = new DateTime('2021-01-01 00:00');
② $end   = date_create_from_format('Y-m-d H:i',
       '2021-02-01 00:00');
?>
<?php include 'includes/header.php'; ?>

<p><b>Sale starts:</b>
③   <?= $start->format('l, jS M Y H:i') ?></p>
<p><b>Sale ends:</b>
④   <?= $end->format('l, jS M Y') ?> <b>at</b>
⑤   <?= $end->format('H:i') ?></p>

<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



1. Этот пример начинается с создания объекта класса `DateTime`. Он сохраняется в переменной с именем `$start`.

2. Второй объект `DateTime` создается с помощью функции `date_create_from_format()`. Первый параметр определяет формат представления даты. Второй параметр задает дату и время. Этот объект сохраняется в переменной с именем `$end`.

3. Дата и время начала выводятся на страницу с помощью метода `format()` объекта `DateTime`. Аргумент указывает формат, в котором должны быть представлены дата и время.

4. Конечная дата (не время) выводится на страницу с помощью метода `format()` объекта `DateTime`. Его параметр определяет формат, в котором должна быть выведена дата.

5. Время окончания выводится отдельно и также использует метод `format()`. Это показывает, как вы можете отдельно вывести дату или время, содержащиеся в объекте.

Упражнение. В шаге 1 установите дату так, чтобы она содержала вчерашнюю дату. В шаге 2 изменить дату на 7 дней после начала действия предложения.

ИЗМЕНЕНИЕ ДАТЫ И ВРЕМЕНИ В ОБЪЕКТАХ DateTime

После создания объекта с помощью класса `DateTime` можно использовать приведенные ниже методы для установки или обновления представляемых им даты/времени.

Методы, устанавливающие дату/время, перезаписывают любую дату/время, находящиеся в данный момент в объекте.

Методы `add()` и `sub()` используют объект `DateInterval`.

МЕТОД	ОПИСАНИЕ
`setDate(`*`$year`*`, `*`$month`*`, `*`$day`*`)`	Устанавливает дату для объекта
`setTime(`*`$hour`*`, `*`$minute`*` [, `*`$seconds`*`][, `*`$microseconds`*`])`	Устанавливает время для объекта
`setTimestamp(`*`$timestamp`*`)`	Устанавливает дату/время, используя метку времени Unix
`modify(`*`$DateFormat`*`)`	Обновляет дату/время с помощью строки
`add(`*`$DateInterval`*`)`	Добавляет интервал времени с помощью объекта `DateInterval`
`sub(`*`$DateInterval`*`)`	Отнимает интервал времени с помощью объекта `DateInterval`

Когда интерпретатор PHP сохраняет в переменной скалярное значение или массив, то они сохраняются *в самой* переменной. И если присвоить ее значение другой переменной, то вы получите два независимых друг от друга значения. Когда же интерпретатор PHP создает объект, то переменная содержит не сам объект, а ссылку на область памяти, где он хранится.

Это означает, что если у вас есть переменная, содержащая объект, и вы присвоите ее значение другой переменной, то они обе будут указывать на одну и ту же область памяти, в которой содержится сам объект.

И в результате, если вы обновите объект в одной переменной, он также будет обновлен и в другой:

```
$start = new DateTime('2020/12/1');
$end = $start;
// время поменялось в обеих переменных
$end->modify('+1 day');
```

Чтобы обойти это, вы можете использовать ключевое слово `clone` для создания копии объекта:

```
$start = new DateTime('2020/12/1');
$end = clone $start;
// Изменяется только объект в $end
$end->modify('+1 day');
```

КАК УСТАНОВИТЬ ДАТУ И ВРЕМЯ В ОБЪЕКТЕ КЛАССА DateTime

PHP section_b/c08/datetime-object-set-date-and-time.php

```
<?php
① $start = new DateTime();
② $start->setDate(2021, 12, 01);
③ $start->setTime(17, 30);
④ $end = clone $start;
⑤ $end->modify('+2 hours 15 min');
?>
<?php include 'includes/header.php'; ?>

<p><b>Event starts:</b>
⑥   <?= $start->format('g:i a - D, M j Y') ?></p>

<p><b>Event ends:</b>
⑥   <?= $end->format('g:i a - D, M j Y') ?></p>

<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



1. Новый объект создается с использованием класса `DateTime`. Объект сохраняется в переменной с именем `$start` и содержит текущую дату и время.
2. Дата устанавливается с помощью метода `setDate()` класса `DateTime`.
3. Время обновляется с помощью метода `setTime()` класса `DateTime`.
4. Объект, содержащийся в переменной `$start`, клонируется с помощью ключевого слова `clone`, а сам клон сохраняется в переменной с именем `$end`.
5. Метод `modify()` класса `DateTime` используется для изменения времени, которое содержит объект в переменной `$end`. Таким образом, он теперь представляет дату и время на 2 часа 15 минут позже, чем объект в переменной `$start`.
6. Дата и время, которые содержатся в обоих объектах, выводятся с помощью метода `format()`.

Упражнение. В шаге 5 изменить окончание мероприятия на два дня после даты начала.

ПРЕДСТАВЛЕНИЕ ИНТЕРВАЛА С ПОМОЩЬЮ DateInterval

Класс `DateInterval` используется для создания объекта, представляющего интервал времени. Он измеряется в годах, месяцах, неделях, днях, часах, минутах и секундах.

Методы `add()` и `sub()` класса `DateTime` используют объект класса `DateInterval` для указания интервала времени при добавлении или удаления от текущей даты/времени. Вы указываете продолжительность интервала, используя формат из таблицы справа.
Буква **P** предшествует каждому интервалу. Буква **T** предшествует периоду времени.

ИНТЕРВАЛ	ПРЕДСТАВЛЕНИЕ
1 год	`P1Y`
2 месяца	`P2M`
3 дня	`P3D`
1 год, 2 месяца, 3 дня	`P1Y2M3D`
1 час	`PT1H`
30 минут	`PT30M`
15 секунд	`PT15S`
1 час, 30 минут, 15 секунд	`PT1H30M15S`
1 год, 1 день, 1 час и 30 минут	`P1Y1DT1H30M`

```
$interval = new DateInterval('P1M');
```

ПЕРЕМЕННАЯ — ИМЯ КЛАССА — ИНТЕРВАЛ

Метод `diff()` (сокращение от difference — «разница») класса `DateTime` сравнивает два объекта `DateTime` и возвращает представляющий разницу между ними объект `DateInterval`.

Чтобы отобразить интервал, хранящийся в объекте `DateInterval`, используйте его метод `format()`. Его аргументом будет строка.
В те места, где вам требуется отображение интервала, добавляются символы из таблицы справа.

ИНТЕРВАЛ	ОПИСАНИЕ
`%y`	Годы
`%m`	Месяцы
`%d`	Дни
`%h`	Часы
`%i`	Минуты
`%s`	Секунды
`%f`	Микросекунды

СТРОКА ДЛЯ ОТОБРАЖЕНИЯ

```
$interval->format('%h hours %i minutes');
```

ИНТЕРВАЛ — ИНТЕРВАЛ

КЛАСС DateInterval

PHP section_b/c08/dateinterval-object.php

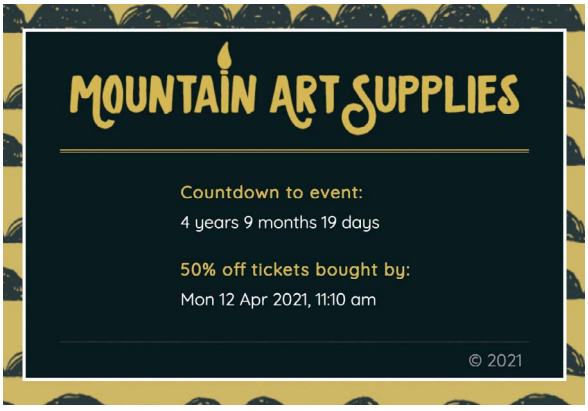
```php
<?php
$today     = new DateTime();
$event     = new DateTime('2025-12-31 20:30');
$countdown = $today->diff($event);

$earlybird = new DateTime();
$interval  = new DateInterval('P1M');
$earlybird->add($interval);
?>
<?php include 'includes/header.php'; ?>

<p><b>Countdown to event:</b><br>
  <?= $countdown->format('%y years %m months %d days') ?>
</p>
<p><b>50% off tickets bought by:</b><br>
  <?= $earlybird->format('D d M Y, g:i a') ?>
</p>

<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



1. Текущие дата и время представлены с помощью объекта `$today` класса `DateTime`.
2. Дата события представлена с помощью объекта `$event` класса `DateTime`.
3. Метод `diff()` объекта `DateTime` получает интервал времени между настоящим моментом и датой события. Возвращаемый объект `DateInterval` сохраняется в `$countdown`.
4. Создается переменная `$earlybird`, которая содержит текущие дату и время.
5. Интервал в один месяц представлен объектом `$interval` класса `DateInterval`.
6. Метод `add()` объекта `$earlybird` добавляет к нему интервал из переменной `$interval`.
7. Вывод интервала из переменной `$countdown`. Обратите внимание, как символ «%» ставится перед символами формата, представляющими интервалы.
8. Выводится сохраненная в `$earlybird` дата.

Упражнение. В шаге 2 измените дату события на 3 месяца в будущем. В шаге 5 сделайте интервал 12 часов.

ПОВТОРЯЮЩИЕСЯ СОБЫТИЯ С ИСПОЛЬЗОВАНИЕМ DatePeriod

Класс `DatePeriod` может создавать объект, содержащий набор объектов класса `DateTime`, которые встречаются через регулярные промежутки времени между начальной и конечной датой. Затем можно выполнить цикл, чтобы получить каждый из созданных объектов по очереди.

Чтобы создать объект `DatePeriod`, вам нужны три вещи:

- начальная дата (объект `DateTime`);
- частота события (объект `DateInterval`);
- дата окончания периода.

Дата окончания периода может быть:

- объектом `DateTime`;
- целым числом, указывающим, сколько раз должно произойти событие (после даты начала).

Когда объект `DatePeriod` создан, он содержит ряд объектов `DateTime`. Каждый из них представляет определенный момент между начальной и конечной датой в интервале, заданном в объекте `DateInterval`.

```
$period = new DatePeriod($start, $interval, $end);
```

ПЕРЕМЕННАЯ — ИМЯ КЛАССА — ДАТА/ВРЕМЯ НАЧАЛА — ИНТЕРВАЛ — ДАТА/ВРЕМЯ ОКОНЧАНИЯ

Вы можете получить доступ к каждому объекту `DateTime` в объекте `DatePeriod` с помощью цикла `foreach`.

Как и во всех циклах, используйте переменную для доступа к каждому объекту `DateTime` по мере прохождения цикла.

Внутри фигурных скобок вы можете использовать методы объекта `DateTime` для работы с этой датой/временем.

ОБЪЕКТ **DatePeriod** СОДЕРЖИТ ОБЪЕКТЫ **DateTime** — ИМЯ ПЕРЕМЕННОЙ ДЛЯ ПРЕДСТАВЛЕНИЯ КАЖДОГО ОБЪЕКТА **DateTime**

```
foreach($period as $occurrence) {
    echo $occurrence->format('Y jS F');
}
```

КЛАСС DatePeriod

PHP section_b/c08/dateperiod-object.php

```php
<?php
$start    = new DateTime('2025-1-1');
$end      = new DateTime('2026-1-1');
$interval = new DateInterval('P1M');
$period   = new DatePeriod($start, $interval, $end);
?>
<?php include 'includes/header.php'; ?>

<p>
  <?php foreach ($period as $event) { ?>
    <b><?= $event->format('l') ?></b>,
    <?= $event->format('M j Y') ?></b><br>
  <?php } ?>
</p>

<?php include 'includes/footer.php'; ?>
```

RESULT



1. Переменная `$start` содержит объект `DateTime`, представляющий 1 января 2025 года.
2. Переменная `$end` содержит объект `DateTime`, представляющий 1 января 2026 года.
3. Переменная `$interval` содержит объект `DateInterval`, представляющий один месяц.
4. Переменная `$period` будет содержать экземпляр класса `DatePeriod`. Для создания ему потребуется три параметра (значения которых были определены в шагах 1–3):

- дата начала;
- интервал;
- дата окончания.

Полученный в итоге объект будет содержать двенадцать экземпляров класса `DateTime` (по одному на каждый месяц 2025 года).

5. Цикл `foreach` перебирает все эти объекты, по очереди присваивая их переменной `$event` внутри цикла.
6. Метод `format()` используется для вывода дня недели, а затем месяца, дня и года.

Упражнение. В шаге 3 измените интервал на три месяца (`P3M`).

УПРАВЛЕНИЕ ЧАСОВЫМИ ПОЯСАМИ С ПОМОЩЬЮ DateTimeZone

При создании объекта `DateTime` можно указать часовой пояс. Такую возможность дает объект, созданный с использованием класса `DateTimeZone`.

Класс `DateTimeZone` создает объект, представляющий часовой пояс. В нем будет храниться информация об этом часовом поясе.

В круглых скобках укажите часовой пояс в формате IANA (полный список см. http:// notes.re/ timezones).

Вы можете использовать этот объект при создании экземпляра класса `DateTime`, чтобы указать его часовой пояс; также этот объект управляет переходом на летнее время.



МЕТОД	ОПИСАНИЕ
`getName()`	Возвращает название текущего часового пояса
`getLocation()`	Возвращает индексированный массив, содержащий следующую информацию:

КЛЮЧ	ЗНАЧЕНИЕ
`country_code`	Короткий код страны
`latitude`	Широта для указанной зоны
`longitude`	Долгота для указанной зоны
`comments`	Любые комментарии по поводу этого места

`getOffset()`	Возвращает смещение от UTC для часового пояса в секундах (UTC — это то же время, что и GMT, но это стандарт времени, не привязанный к какому-либо часовому поясу)
`getTransitions()`	Возвращает массив, указывающий, когда в данном часовом поясе вступает в силу переход на летнее время

КЛАСС DateTimeZone

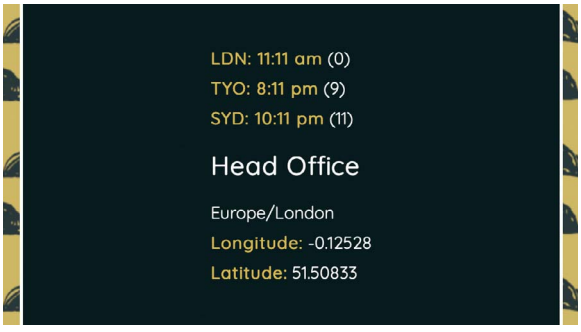
PHP section_b/c08/datetimezone-object.php

```php
<?php
$tz_LDN   = new DateTimeZone('Europe/London');
$tz_TYO   = new DateTimeZone('Asia/Tokyo');
$location = $tz_LDN->getLocation();

$LDN      = new DateTime('now', $tz_LDN);
$TYO      = new DateTime('now', $tz_TYO);
$SYD      = new DateTime('now',
                new DateTimeZone('Australia/Sydney'));
?> ...
<p><b>LDN: <?= $LDN->format('g:i a') ?></b>
  (<?= ($LDN->getOffset() / (60 * 60)) ?>)<br>
  <b>TYO: <?= $TYO->format('g:i a') ?></b>
  (<?= ($TYO->getOffset() / (60 * 60)) ?>)<br>
  <b>SYD: <?= $SYD->format('g:i a') ?></b>
  (<?= ($SYD->getOffset() / (60 * 60)) ?>)<br></p>

<h1>Head Office</h1>
<p><?= $tz_LDN->getName() ?><br>
  <b>Longitude:</b> <?= $location['longitude'] ?><br>
  <b>Latitude:</b>  <?= $location['latitude'] ?></p>
```

РЕЗУЛЬТАТ



1. Два объекта `DateTimeZone` создаются для представления часовых поясов Лондона и Токио.
2. Функция `getLocation()` возвращает данные о местоположении для лондонского часового пояса в виде массива. Массив сохраняется в переменной `$location`.
3. Два объекта `DateTime` создаются с использованием объектов `DateTimeZone` из шага 1. Они представляют текущее время в двух различных часовых поясах.
4. Третий объект `DateTime` создается, чтобы показать, как можно создать объект `DateTimeZone` одновременно с объектом `DateTime`.
5. Для каждого объекта `DateTime`:

- функция `format()` показывает текущее время в этом местоположении;
- функция `getOffset()` показывает разницу во времени между текущим местоположением и UTC. Она возвращает разницу во времени в секундах, поэтому значение делится на 60 × 60, чтобы показать смещение в часах.

6. Имя первого часового пояса выводится с помощью `getName()`.
7. Выводятся долгота и широта данного часового пояса.

Упражнение. Создать объект для офиса в Лос-Анджелесе и отобразить местное время.

ЗАКЛЮЧЕНИЕ

ДАТА И ВРЕМЯ

- Символы форматирования позволяют указать формат, в котором могут быть выведены дата и время.
- Метка времени Unix представляет дату и время, используя секунды, прошедшие с 1 января 1970 года.
- Функции `time()`, `strtotime()` и `mktime()` создают метки времени Unix. Функция `date()` преобразует метку времени Unix в удобочитаемый формат.
- Класс `DateTime` позволяет создавать объекты, представляющие дату и время. В нем есть методы, которые позволяют изменить значение даты и времени, а также отобразить его в различных форматах.
- С помощью класса `DateInterval` можно создавать объекты, представляющие интервал времени, например месяц или год.
- С помощью класса `DatePeriod` можно создать объект, содержащий набор экземпляров класса `DateTime`, представляющих повторяющиеся события.
- Класс `DateTimeZone` позволяет создавать объекты для представления часовых поясов и хранения информации о них.

9

COOKIE И СЕССИИ

Чтобы создавать веб-страницы, выводящие персонализированные данные, например имя пользователя, изображение профиля или список недавно просмотренных страниц, сайту необходимо знать, кто запрашивает страницу в данный момент.

Протокол HTTP описывает правила, определяющие, как браузер должен запрашивать веб-страницу и как сервер должен отвечать, но эти запросы являются полностью независимыми друг от друга. HTTP не предоставляет веб-сайту механизма, чтобы определить, какой посетитель запрашивает страницу.

Если сайту необходимо узнать, кто запрашивает веб-страницу, или показать какую-либо персонализированную информацию, он может отслеживать каждого посетителя и хранить информацию об их предпочтениях, используя cookie и сессии.

- **Cookie** (куки, «печеньки») — это текстовые файлы, которые хранятся в браузере пользователя[40]. Сайт может указать браузеру, какие данные сохранить в cookie, и затем браузер будет отправлять эти данные обратно на веб-сайт при запросе каждой страницы с этого сайта.
- **Сессии** (session) позволяют сайту временно хранить данные о пользователе на сервере. Когда посетитель запрашивает другую страницу с сайта, интерпретатор PHP может получить доступ к данным из сессии этого отдельного пользователя.

Cookie и сессии временно хранят небольшие объемы данных. Они не гарантируют сохранение данных в течение длительного времени, поскольку пользователи могут удалять cookie (или заходить на сайт из другого браузера, в котором нет этих cookie), а сессии рассчитаны только на продолжительность одного посещения сайта (они не хранят данные между посещениями).

Если данные о пользователях должны храниться в течение более длительного периода, их сохраняют в базе данных. Вы узнаете, как это сделать, в главе 13. Однако, чтобы сопоставить пользователя и его запись в базе данных, потребуется понимание принципов работы cookie и сессий.


MOUNTAIN ART SUPPLIES
HOME PRODUCTS MY ACCOUNT LOG IN
Login
Email:
Password:
LOGIN
MacBook Pro

ЧТО ПРЕДСТАВЛЯЮТ СОБОЙ COOKIE?

Веб-сайт может указать браузеру хранить данные о пользователе в текстовом файле, называемом **cookie**. Затем каждый раз, когда браузер запрашивает другую страницу с этого сайта, браузер отправляет cookie обратно на сервер.

ЧТО ТАКОЕ COOKIE

Cookie — это небольшой именованный фрагмент информации, который веб-сайт может отправить браузеру и который он может попросить «запомнить», например, путем сохранения в файле на компьютере пользователя, где он и будет храниться.
У каждого cookie есть имя. Оно должно описывать, какую именно информацию содержит cookie. Это имя будет одинаковым для всех посетителей. А вот значение будет разным. Таким образом, cookie можно сравнить с переменной, которая хранится в браузере пользователя.

СОЗДАНИЕ COOKIE

Когда браузер запрашивает веб-страницу, веб-сайт может отправить вместе с ответом дополнительный HTTP-заголовок. Этот HTTP-заголовок сообщает браузеру имя создаваемого cookie и значение, которое должно быть сохранено. Оно представляет собой текст, который не должен быть длиннее 4096 символов. Каждый сайт может создать несколько cookie.

ПОЛУЧЕНИЕ ДАННЫХ COOKIE

Если после создания cookie браузер запросит любую страницу сайта, то он добавит cookie в HTTP-заголовки, отправляемые на сервер с запросом страницы. Затем интерпретатор PHP добавляет данные cookie в суперглобальный массив с именем `$_COOKIE`, чтобы PHP-код на этой странице мог их использовать. Имя cookie — это ключ, а его значение — это значение, сохраненное в браузере.

КТО МОЖЕТ ПОЛУЧИТЬ ДОСТУП К COOKIE

Браузеры отправляют cookie на сервер только тогда, когда они запрашивают страницу с создавшего cookie домена. Например, если google.com создает файл cookie, то браузер будет отправлять его на сервер, запрашивая страницы только из google.com.
Но никогда не отправит его на vk.com или куда-либо еще.

COOKIE ПРИВЯЗАНЫ К ОДНОМУ БРАУЗЕРУ

Потому что именно браузер создает и хранит файлы cookie.

- Если на устройстве установлено более одного браузера, cookie отправляется только из того браузера, в котором он был сохранен, а не из каких-либо других браузеров, установленных на этом устройстве.
- Если пользователь зайдет с нового устройства на сайт, у этого устройства не будет cookie для отправки на сервер.

КАК ДОЛГО ХРАНЯТСЯ ФАЙЛЫ COOKIE

Сервер может указать дату и время, когда истечет срок действия cookie. Это момент, когда браузер должен прекратить отправку данных cookie на сервер. Если дата истечения срока действия не указана, браузер хранит такие cookie до закрытия.
Пользователи также могут отказаться принимать cookie или удалить их из браузера, поэтому сайт должен поддерживать возможность работать без них.

ПЕРВЫЙ ЗАПРОС СТРАНИЦЫ

- Браузер запрашивает страницу с помощью HTTP-запроса.



ЗАПРОС: `page.php`

- Сервер отправляет обратно запрошенную страницу.
- Добавляет HTTP-заголовок, сообщающий браузеру имя cookie и значение, которое нужно сохранить.

ОТВЕТ: `page.php`

ЗАГОЛОВОК: `counter = 1`

- Браузер отображает страницу.
- Сохраняет cookie, используя данные из заголовка HTTP.



COOKIE: `counter = 1`

Cookie не следует использовать для хранения конфиденциальных данных (например, адресов электронной почты или номеров кредитных карт), поскольку содержимое HTTP- заголовков можно просмотреть в инструментах разработчика браузера, а передача между браузером и сервером ведется в виде обычного текста.

ПОСЛЕДУЮЩИЕ ЗАПРОСЫ СТРАНИЦ

- Браузер запрашивает страницу с помощью HTTP-запроса.
- Отправляет HTTP-заголовок с именем и значением cookie.



ЗАПРОС: `page.php`

ЗАГОЛОВОК: `counter = 1`

- Сервер добавляет данные cookie в `$_COOKIE`.
- Создает страницу, используя данные из `$_COOKIE`.
- Отправляет запрошенную страницу в браузер.
- При этом он может отправить в браузер обновленное значение cookie.

ОТВЕТ: `page.php`

ЗАГОЛОВОК: `counter = 2`

- Браузер отображает страницу, созданную с использованием данных cookie.
- Обновляет сохраненный cookie, используя данные в заголовке HTTP.



COOKIE: `counter = 2`

Чтобы никто не мог прочитать HTTP-заголовки, когда они передаются между браузером и сервером, запустите свой сайт, используя HTTPS, а не HTTP. Этот протокол шифрует информацию в заголовках.

КАК СОЗДАВАТЬ COOKIE И ПОЛУЧАТЬ К НИМ ДОСТУП

Встроенная в PHP функция `setcookie()` используется для отправки cookie в браузер. Для доступа к полученным из браузера значениям cookie вы можете использовать суперглобальный массив `$_COOKIE` или функции `filter_input()` и `filter_input_array()`.

Функция PHP `setcookie()` создает HTTP-заголовок, который отправляется вместе с веб-страницей в браузер и указывает ему создать cookie. Эта функция позволяет вам задать имя и значение для cookie.

Поскольку эта функция создает HTTP-заголовок, ее необходимо вызывать **перед** отправкой содержимого в браузер, как и в случае с функцией `header()`. Даже пробел перед открывающим тегом `<?php` рассматривается как содержимое.

Если вы не установите дату истечения срока действия cookie, браузер будет отправлять его на сервер до своего закрытия.

```
setcookie($name, $value);
```

Как только браузер получает cookie, он начинает отправлять его обратно на сервер вместе с каждым запросом. Когда интерпретатор PHP получает запрос, он добавляет данные cookie в суперглобальный массив с именем `$_COOKIE`.

Для каждого cookie в массив добавляется свой элемент, в котором:

- **ключ** — это имя cookie;
- **значение** — это значение, присланное браузером (всегда строковый тип данных).

Значения элементов этого массива можно присвоить отдельным переменным.

Если код пытается получить доступ к несуществующему в `$_COOKIE` ключу, генерируется ошибка. Чтобы предотвратить это, можно использовать оператор объединения с `null`. Если ключ есть в массиве, возвращается значение, сохраненное в cookie. Если нет, то возвращается `null`.

```
$preference = $_COOKIE['name'] ?? null;
```

Функции PHP `filter_input()` и `filter_input_array()` также могут получать значения cookie. Источник данных должен быть установлен как `INPUT_COOKIE`.

Второй параметр — это имя cookie. Третий и четвертый параметры необязательны — они определяют идентификатор используемого фильтра и любые параметры для этого фильтра.

Если cookie с таким именем отсутствует, функция не выдает ошибку. Кроме того, если используются фильтры целых чисел, логических значений или чисел с плавающей точкой, они преобразуют значение в этот тип данных.

```
$preference = filter_input(INPUT_COOKIE, $name[, $filter[, $options]]);
```

УСТАНОВКА ЗНАЧЕНИЙ COOKIE И ДОСТУП К НИМ

PHP section_b/c09/cookies.php

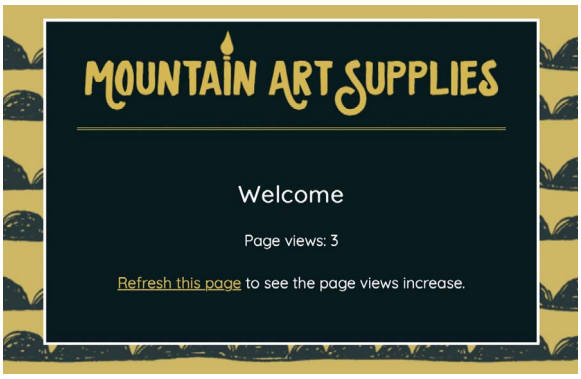
```
<?php
① $counter = $_COOKIE['counter'] ?? 0; // Получение данных
② $counter = $counter + 1;             // +1 к счетчику
③ setcookie('counter', $counter);      // Обновление cookie

④ $message = 'Page views: ' . $counter; // Сообщение
?>
<?php include 'includes/header.php'; ?>

<h1>Welcome</h1>
⑤ <p><?= $message ?></p>
<p><a href="sessions.php">Refresh this page</a> to see
the page views increase.</p>

<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



В этом примере cookie используется для подсчета количества просмотров страниц.

1. Переменной `$counter` присваивается количество просмотров страницы. Если браузер отправил cookie с названием `counter` на сервер, переменная `$counter` получит это значение. Если нет, то вместо него оператор объединения с `null` присвоит значение 0.
2. К значению в переменной `$counter` добавляется 1, поскольку посетитель только что просмотрел страницу.
3. Функция `setcookie()` указывает браузеру создать или обновить cookie с именем `counter` со значением из переменной `$counter`.
4. Переменной `$message` присваивается сообщение, в котором используется количество просмотров страницы.
5. Вывод этого сообщения.

Упражнение. После того как вы просмотрели страницу один раз, обновите ее и посмотрите, как растет счетчик.

Упражнение. Сохраните свое имя в cookie под названием `name`, а затем отобразите его после просмотра страницы.

БЕЗОПАСНОСТЬ ПРИ РАБОТЕ С COOKIE

У функции `setcookie()` есть параметры, управляющие тем, как браузер будет хранить и возвращать значение cookie на сервер. Вы также должны проверять данные, полученные из cookie, и использовать `htmlspecialchars()`, если их содержимое отображается на странице.

Чтобы обновить значение cookie, вызовите функцию `setcookie()` с новым значением. Для того чтобы браузер перестал отправлять cookie на сервер (т. е. чтобы удалить cookie), также вызовите `setcookie()`, но при этом установите в качестве значения пустую строку, а срок жизни — в прошлом. Если вы обновляете значение или время жизни cookie, то в последних четырех аргументах **обязательно** должны использоваться ровно те же значения, которые использовались при создании cookie.

Поскольку существует возможность подделать HTTP-заголовки, в том числе и содержащие cookie, сервер должен проверять данные cookie перед их использованием (используя методы из главы 6). Если на странице отображается значение cookie, необходимо использовать функцию `htmlspecialchars()` для предотвращения атаки XSS.

```
setcookie($name[, $value, $expire, $path, $domain, $secure, $httponly])
```

ПАРАМЕТР	ОПИСАНИЕ
$name	Имя cookie
$value	Значение, которое необходимо присвоить cookie (оно всегда отправляется как строка, поскольку cookie не сохраняет информацию о типе)
$expire	Дата и время, когда браузер должен прекратить отправку cookie на сервер (как метка времени Unix). Чтобы установить метку времени, используйте функцию PHP `time()` и добавьте период, в течение которого вы хотите сохранить файл cookie

ПЕРИОД	СЕЙЧАС	СЕК.	МИН.	ЧАСЫ	ДНИ
1 день	`time() +`	`60 *`	`60 *`	`24`	
30 дней	`time() +`	`60 *`	`60 *`	`24 *`	`30`

$path	Если файл cookie необходим только для части сайта, укажите каталоги, для которых он должен использоваться. По умолчанию путь — это корневая папка «/», что означает все папки. Замена этого параметра на «`/members`» означает, что он отправляется только на страницы в папке `members` сайта
$domain	Если файл cookie необходим только для поддомена, напишите имя этого домена. По умолчанию cookie отправляется на все поддомены сайта. Если он установлен на поддомен members.example.org, значение cookie отправляется только на файлы в поддомене members.example.org
$secure	Если этому параметру присвоено значение `true`, файл cookie будет создан в браузере, но браузер отправит его обратно на сервер, только если страница запрашивается с использованием защищенного HTTPS-соединения
$httponly	Если задано значение `true`, то браузер возвращает значение cookie только серверу, но при этом оно будет недоступно из JavaScript

УПРАВЛЕНИЕ ПАРАМЕТРАМИ COOKIE

PHP section_b/c09/cookie-preferences.php

```php
<?php
$color   = $_COOKIE['color'] ?? null; // Получение данных
$options = ['light', 'dark',];        // Параметры

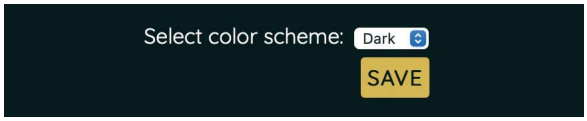
if ($_SERVER['REQUEST_METHOD'] == 'POST') { // Если форма
// отправлена
    $color = $_POST['color'];      // Получение цвета
    setcookie('color', $color, time() + 60 * 60,
        '/', '', false, true);     // Установка cookie
}

//Если цвет является допустимым параметром, использовать
//его — в противном случае использовать темную тему
$scheme = (in_array($color, $options)) ? $color : 'dark';
?>
<?php include 'includes/header-style-switcher.php'; ?>
  <form method="POST" action="cookie-preferences.php">
    Select color scheme:
    <select name="color">
      <option value="dark">Dark</option>
      <option value="light">Light</option>
    </select><br>
    <input type="submit" value="Save">
  </form>
<?php include 'includes/footer.php'; ?>
```

PHP section_b/c09/includes/header-style-switcher.php

```php
<body class="<?= htmlspecialchars($scheme) ?>">
```

РЕЗУЛЬТАТ



1. Переменная `$color` получает значение, полученное из cookie под названием `color`, или `null`, если cookie с таким именем не существует.
2. Массив с разрешенными параметрами для цветовой схемы.
3. Проверка, была ли форма отправлена.
4. Если да, то переменной `$color` присваивается значение, полученное из формы, из выпадающего списка с именем `color`. Оно перезаписывает значение, присвоенное в шаге 1.
5. Функция `setcookie()` вызывается для отправки в браузер cookie с именем `color`. Ему присваивается значение, выбранное пользователем в выпадающем списке. Кроме того, выставляются следующие параметры:

- срок жизни cookie истекает через один час;
- действует на всех страницах сайта;
- отправляется через HTTP или HTTPS;
- скрыт от JavaScript.

6. Условие в тернарном операторе проверяет, присутствует ли значение из переменной `$color` в массиве `$options`. Если да, то оно присваивается переменной `$scheme`. Если нет, то переменная `$scheme` получает значение `dark`.
7. Подключается новый файл с заголовком сайта. Он выводит значение из переменной `$color` в атрибут `class` тега `<body>`, чтобы на странице использовалась выбранная цветовая схема.

ЧТО ТАКОЕ СЕССИИ?

Сессии (или сеансы) позволяют сохранять информацию о пользователе и его предпочтениях на сервере. Они называются сессиями, потому что хранят данные только в течение одного посещения сайта.

ЧТО ТАКОЕ СЕССИЯ?

Когда начинается сессия, интерпретатор PHP создает три вещи/

- **Идентификатор сессии** (**session ID**) — строка, используемая для идентификации конкретного посетителя.
- **Файл сессии** (**session file**) представляет собой текстовый файл, хранящийся на сервере. Он используется для хранения данных об этом пользователе. Имя файла будет содержать идентификатор сессии.
- **Сессионный cookie** (**session cookie**), хранящийся в браузере. Его имя — `PHPSESSID`, а значение — идентификатор сессии этого пользователя.

КАК ДОЛГО ДЛЯТСЯ СЕССИИ

Для работы сессии необходим как cookie-файл сессии в браузере, так и файл сессии на сервере. Срок действия сессионных cookie истекает, когда пользователь закрывает браузер. Файлы сессии могут быть удалены сервером, если в них не вносились изменения в течение определенного периода времени (по умолчанию — 24 минуты).

ПОЛУЧЕНИЕ ДАННЫХ СЕССИИ

Если в браузере для сайта сохранен cookie с именем `PHPSESSID`, его значение отправляется на сервер каждый раз, когда пользователь запрашивает страницу с этого сайта. Идентификатор сессии используется для идентификации пользователя. При этом PHP:

- находит файл сессии, имя которого содержит идентификатор сессии, отправленный в cookie;
- читает данные из файла сессии и помещает их в суперглобальный массив `$_SESSION`, чтобы код мог получить к ним доступ.

КАК СТАРТУЮТ СЕССИИ

Когда сайт использует сессии, каждая страница должна вызывать встроенную в PHP функцию `session_start()`.
При вызове этой функции интерпретатор PHP автоматически запускает новый сеанс для текущего пользователя, если браузер, запрашивающий страницу, не отправил сессионный файл cookie или если соответствующий файл сессии невозможно найти.

СОХРАНЕНИЕ ДАННЫХ СЕССИИ

После старта сессии новые данные можно сохранить в сессии этого пользователя, добавив их в суперглобальный массив `$_SESSION`. Когда страница завершает работу, интерпретатор PHP берет все данные из суперглобального массива `$_SESSION` и сохраняет их в файле сессии этого пользователя. Сохранение данных в файл сессии обновляет время последнего изменения этого файла, и интерпретатор PHP может проверить это время, чтобы определить, использовался ли сеанс недавно.

ДРУГИЕ ВАРИАНТЫ ОРГАНИЗАЦИИ РАБОТЫ СЕССИЙ

Вместо использования сессионных cookie можно передавать идентификатор сеанса в URL-адресе, но это менее безопасно.
Также возможно хранить данные сессии не в файле, а в базе данных, но эта тема выходит за рамки данной книги. Обычно этот вариант используется на сайтах с большим объемом трафика, которые требуют нескольких серверов для обработки нагрузки.

ПЕРВЫЙ ЗАПРОС СТРАНИЦЫ

- Браузер запрашивает страницу с помощью HTTP-запроса.



ЗАПРОС: `page.php`

На сервере код PHP вызывает функцию `session_start()`. Браузер не отправил сессионный cookie, поэтому PHP:

- генерирует новый **идентификатор сессии** для этого пользователя;
- создает **файл сессии** для хранения данных этого пользователя, используя в имени файла идентификатор сессии.

Далее в коде добавляются значения в суперглобальный массив `$_SESSION`. По завершении работы скрипта значения из этого массива добавляются в файл сессии, созданный для этого пользователя.

- Сервер отправляет запрошенную страницу в браузер.
- И кроме того, отправляет HTTP-заголовок, который создает **сессионный cookie**, содержащий идентификатор сеанса.

ОТВЕТ: `page.php`

ЗАГОЛОВОК: `PHPSESSID = 1234567`

- Браузер отображает страницу.
- Создает сессионный cookie, содержащий идентификатор сеанса.



COOKIE: `PHPSESSID = 1234567`

ПОСЛЕДУЮЩИЕ ЗАПРОСЫ СТРАНИЦ

- Браузер запрашивает страницу с помощью HTTP-запроса.
- Отправляет в HTTP-заголовках cookie с идентификатором сеанса.



ЗАПРОС: `page.php`

ЗАГОЛОВОК: `PHPSESSID = 1234567`

На сервере в коде PHP вызывается функция `session_start()`. Интерпретатор PHP находит файл сессии по ее идентификатору, полученному из cookie. И далее:

- добавляет данные из файла сессии в суперглобальный массив `$_SESSION`, чтобы код мог их использовать;
- код PHP формирует страницу, используя данные из этого массива;
- а также может обновлять данные в нем.

Когда страница завершит выполнение, значения из суперглобального массива `$_SESSION` сохраняются в файле сессии. Это обновит время последнего изменения файла сессии.

- Сервер отправляет запрошенную страницу в браузер.

ОТВЕТ: `page.php`

- Браузер отображает страницу.
- Отправляет сессионный cookie с каждым запросом к этому сайту до тех пор, пока пользователь не закроет окно браузера.



COOKIE: `PHPSESSID = 1234567`

КАК СОЗДАВАТЬ СЕССИИ И ПОЛУЧАТЬ К НИМ ДОСТУП

Каждая страница сайта, использующая сеансы, должна вызывать `session_start()`.
Если у пользователя нет сессии, эта функция создаст сессию для него. Если сессия уже есть, PHP прочитает данные сессии и поместит их в суперглобальный массив `$_SESSION`.

Когда посетитель впервые запрашивает страницу, на которой вызывается `session_start()`, то создается новый идентификатор сессии, сессионный cookie и файл сессии.

Эта функция должна быть вызвана перед отправкой любого содержимого в браузер, поскольку она отправляет HTTP-заголовок для создания сессионного cookie.

Она также должна быть вызвана до того, как в коде понадобится доступ к данным сессии, поскольку она заполняет суперглобальный массив `$_SESSION` данными из файла сессии.

```
session_start();
```

Если вы изменяете данные в суперглобальном массиве `$_SESSION`, то после завершения работы страницы интерпретатор PHP автоматически запишет все эти изменения в файл сессии для пользователя, который запрашивал эту страницу.

Синтаксис для добавления данных в массив такой же, как и для любого ассоциативного массива. Ключ должен описывать данные, которые будут храниться в этом элементе.

Значение для каждого элемента может быть скалярным (строка, число или логическое значение), массивом или объектом, поскольку в сессии можно хранить данные любых типов (в отличие от cookie, которые хранят только строки).

```
$_SESSION['name'] = 'Ivy';
$_SESSION['age']  = 27;
```

При обращении к суперглобальному массиву `$_SESSION` используйте оператор объединения с `null` в случае, если значение может отсутствовать, или же используйте функции фильтрации PHP с источником данных `INPUT_SESSION`.

```
$name = $_SESSION['username'] ?? null;
$age  = $_SESSION['age']      ?? null;
```

ФУНКЦИЯ	ОПИСАНИЕ
`session_start()`	Создает новую сессию или получает данные из существующей
`session_set_cookie_params()`	Настройки, используемые для создания сессионного cookie
`session_get_cookie_params()`	Возвращает массив, содержащий параметры, использованные для установки файла cookie
`session_regenerate_id()`	Создает новый идентификатор сессии и обновляет файл сессии и cookie
`session_destroy()`	Удаляет файл сессии с сервера

СОХРАНЕНИЕ ДАННЫХ В СЕССИИ И ДОСТУП К НИМ

PHP section_b/c09/sessions.php

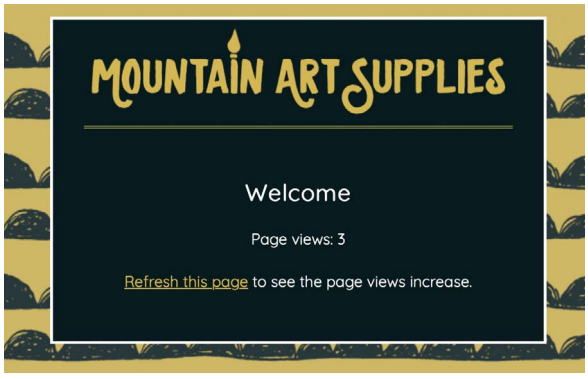
```php
<?php
session_start();      // Создание/возобновление сессии
$counter = $_SESSION['counter'] ?? 0; // Получение
                                      // значения
$counter = $counter + 1;          // Увеличение счетчика
$_SESSION['counter'] = $counter; // Обновление значения
                                 // в сессии

$message  = 'Page views: ' . $counter;  // Сообщение
?>
<?php include 'includes/header.php'; ?>

<h1>Welcome</h1>
<p><?= $message ?></p>
<p><a href="sessions.php">Refresh this page</a> to see
the page views increase.</p>

<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



Упражнение. После того как вы просмотрели страницу один раз, обновите ее и посмотрите, как растет счетчик.

Упражнение. Сохраните свое имя в суперглобальном массиве `$_SESSION` и отобразите его на странице.

В этом примере выполняется та же задача, что и в прошлом, но счетчик сохраняется в сессии.

1. Когда вызывается функция PHP `session_start()`, интерпретатор PHP пытается получить данные из файла сессии и сохранить их в суперглобальном массиве `$_SESSION`. Если это невозможно, для этого посетителя будет создана новая сессия.
2. Если суперглобальный массив `$_SESSION` содержит значение с ключом `counter`, то оно присваивается переменной с именем `$counter`. В противном случае в нее записывается 0.
3. Посетитель только что просмотрел страницу, поэтому к счетчику добавляется 1.
4. Обновляется значение для ключа `counter` в суперглобальном массиве `$_SESSION`.
5. Переменной `$message` присваивается сообщение с указанием количества просмотров страницы.
6. Это сообщение выводится на страницу.

По окончании обработки страницы PHP сохраняет данные из суперглобального массива `$_SESSION` в файле сессии.

Также по окончании работы страницы PHP обновляет время последнего изменения файла сессии на сервере, поэтому данные сессии будут храниться дольше.

ЖИЗНЕННЫЙ ЦИКЛ СЕССИИ

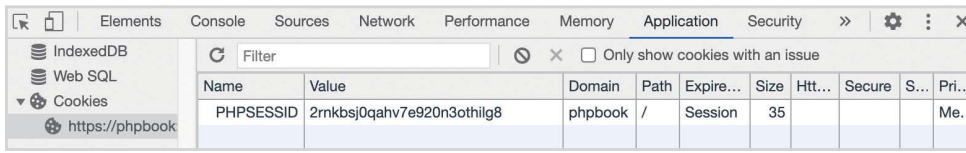
Браузер удаляет сессионные cookie, когда окно браузера закрывается. Сервер удаляет файлы сессии во время процесса, называемого **сборкой мусора**, который запускается нерегулярно. Это означает, что сессии могут длиться дольше, чем ожидалось.

Если вы еще этого не делали, откройте предыдущий пример в своем браузере. Затем откройте:

- инструменты разработчика в браузере, чтобы они отображали cookie;
- папку, в которой веб-сервер хранит файлы сессий.

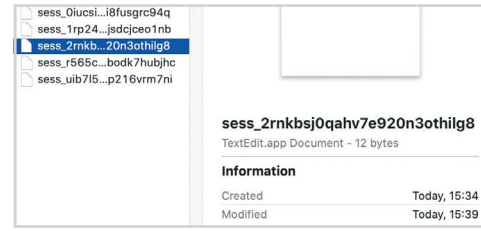
Для получения помощи в поиске см. http://notes.re/php/session-locations.

В браузере вы должны увидеть cookie с именем PHPSESSID. Значение этого cookie — это идентификатор сессии.



В папке, где веб-сервер хранит файлы сессии, вы должны увидеть файл с именем, содержащим идентификатор сессии. Обратите внимание на дату и время изменения файла сессии, а затем обновите браузер, показывающий предыдущий пример, и снова проверьте файл. Время его изменения должно обновиться.

Если вы откроете пример в другом браузере (например, попробуйте Chrome и Firefox), это создаст новый сеанс, поскольку сеанс привязан к браузеру.



Если на странице вызывается функция `session_start()`, а интерпретатор PHP не получает сессионный cookie или не может найти файл сессии, соответствующий полученному cookie, то он создаст новую сессию.

Когда вызвавший `session_start()`скрипт завершает выполнение, он сохраняет данные из суперглобального массива `$_SESSION` в файл сессии и обновляет время изменения этого файла.

Интерпретатор PHP проверяет дату и время обновления файла сессии, чтобы определить, можно ли его удалять (что приведет к завершению сессии).

Поэтому, когда сайт использует сессии, важно вызывать `session_start()` на каждой странице. В противном случае (если пользователь просматривал страницы сайта, на которых не стартует сессия и, следовательно, не менялось время обновления файла) сессия может быть завершена, даже если пользователь все еще просматривает сайт.

При работе с сессиями PHP запускает процесс, называемый **сборкой мусора**. Он удаляет файлы сессии, дата последнего изменения которых превышает заданный период (по умолчанию — 24 минуты).

Как только файл сессии удаляется, завершается и сама сессия, поскольку даже если браузер отправит сессионный cookie, то соответствующий ему файл не будет найден. Проверка времени обновления каждого файла сессии и удаление старых файлов отнимают ресурсы сервера, поэтому PHP старается делать это не слишком часто.

Частота запуска сборки мусора зависит от количества обращений к сессиям. Поэтому на сайте с небольшим количеством посещений сборка мусора может не выполняться в течение нескольких часов или даже дней.

Как показано в следующем примере, сессии часто используются для проверки того, авторизован ли пользователь на сайте. В таких случаях у пользователей должна быть возможность выйти из системы.

Если пользователь не закрывает окно своего браузера (и браузер все еще отправляет сессионный cookie), а посещаемость сайта низкая (и поэтому он долго не запускает сборку мусора), сессия может длиться дольше, чем было задумано.

Это большая проблема, в случае если пользователь заходит на сайт с компьютера общего пользования, поскольку если он не выйдет из системы, другой человек сможет зайти на сайт, используя его учетную запись.

Завершение сессии включает в себя следующие четыре шага.

---

1. Удалите все данные из файла сессии, присвоив суперглобальному массиву `$_SESSION` значение пустого массива. Это также предотвратит доступ к этим значениям любого последующего кода на той же странице.

```
$_SESSION = [];
```

2. В шаге 3 функция `setcookie()` будет использоваться для обновления сессионного cookie. При этом аргументы для параметров `path`, `domain`, `secure` и `httponly` должны использовать ровно те же значения, которые использовались при создании cookie.

Функция PHP `session_get_cookie_params()` вернет значения для этих параметров для сессионного cookie.

Массив, который она возвращает, сохраняется в переменной `$params`.

```
$params = session_get_cookie_params();
```

3. Для обновления сессионного cookie используется функция PHP `setcookie()`. Параметру `value` присваивается значение пустой строки; таким образом из сессионного cookie удаляется идентификатор сессии. В параметр `expires` передается дата в прошлом, чтобы браузер не отправлял файл cookie на сервер. Все остальные параметры задаются с использованием значений, собранных на шаге 2 и сохраненных в виде массива в переменной `$params`.

```
setcookie('PHPSESSID', '', time() - 3600, $params['path'],
  $params['domain'], $params['secure'], $params['httponly']);
```

4. Вызовите функцию PHP `session_destroy()`, чтобы интерпретатор PHP удалил файл сессии. Интерпретатор PHP удалит файл немедленно, вместо того чтобы ждать, пока он будет удален при сборке мусора.

```
session_destroy();
```

ПРОСТАЯ СИСТЕМА АВТОРИЗАЦИИ

Часто на сайтах пользователям требуется войти в систему для просмотра определенных страниц.

В этом примере пользователь должен авторизоваться, чтобы просмотреть страницу My Account (Моя учетная запись). После того как пользователь войдет в систему:

- сессия, привязанная к его браузеру, будет помнить, что он вошел в систему;
- он сможет просматривать страницу своей учетной записи;
- текст последней ссылки на панели навигации меняется с Log In (Войти) на Log Out (Выйти).

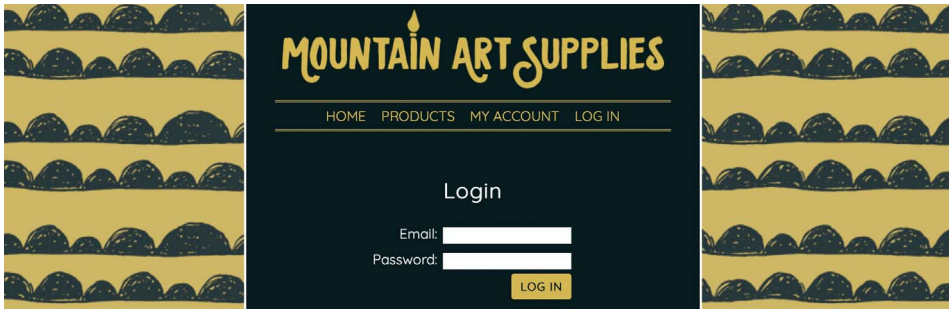
Примечание. В этом примере показано только, как сессии используются для определения того, вошел ли пользователь в систему. В главе 16 вы узнаете, как создать полноценную систему авторизации (входа в систему), позволяющую каждому пользователю использовать собственные регистрационные данные (хранящиеся в базе данных).

Когда сайт использует сессии, каждая страница должна вызывать функцию `session_start()` перед отправкой любого содержимого в браузер. Это гарантирует, что у каждого пользователя есть активная сессия и что время изменения файла сессии обновляется каждый раз при просмотре новой страницы.

В этом примере каждая страница включает файл `sessions.php` (см. страницу справа). Она вызывает `session_start()` и содержит весь код, связанный с сессией.

1. Функция `session_start()` указывает интерпретатору PHP взять данные из файла сессии посетителя и поместить их в суперглобальный массив `$_SESSION` или создать новую сессию, если файл отсутствует.
2. Если суперглобальный массив `$_SESSION` содержит запись об авторизации пользователя, переменная `$logged_in` получает значение `true`; в противном случае оператор объединения с `null` присваивает ей значение `false`.
3. Переменные `$email` и `$password` содержат сведения, которые пользователь должен будет ввести для входа в систему.
4. Также файл содержит три определения функций.
5. Страница входа в систему вызовет функцию `login()`, если пользователь введет правильный адрес электронной почты и пароль.
6. Когда пользователь входит в систему, рекомендуется сбросить идентификатор его сессии. Функция PHP `session_regenerate_id()` создает новый идентификатор сессии и обновляет файл сессии и cookie, чтобы использовать новый идентификатор сессии. Аргумент `true` указывает интерпретатору PHP удалить все данные, которые содержатся в сессии.
7. В массив `$_SESSION` добавляется элемент с ключом `logged_in`. Его значение равно `true`. Оно указывает, что посетитель вошел в систему.
8. Функция `logout()` используется для завершения сессии.
9. Суперглобальному массиву `$_SESSION` присваивается значение пустого массива. Это приведет к удалению данных из файла сессии, а также к тому, что дальнейший код на странице не увидит данных сессии.
10. Сессионный cookie обновляется; идентификатор сессии заменяется пустой строкой, а дата истечения срока действия устанавливается на прошедшую (поэтому браузер перестает его отправлять).
11. Удаляется файл сессии на сервере.
12. Функция `require_login()` может быть вызвана на любой странице, требующей, чтобы посетитель вошел в систему.
13. Конструкция `if` проверяет, содержит ли переменная `$logged_in` значение `false`. Если это так, то либо пользователь не вошел в систему, либо сессия завершилась.
14. Пользователь перенаправляется на страницу входа в систему.
15. Команда `exit` останавливает выполнение любого другого кода.

РЕЗУЛЬТАТ



PHP section_b/c09/includes/sessions.php

```
<?php
session_start();                                    // Начало/возобновление сеанса
$logged_in = $_SESSION['logged_in'] ?? false;       // Пользователь авторизовался?

$email    = 'ivy@eg.link';                          // Email для входа в систему
$password = 'password';                             // Пароль для входа в систему

function login()                                    // Запоминание авторизации пользователя
{
    session_regenerate_id(true);                    // Обновление ID сессии
    $_SESSION['logged_in'] = true;          // Присвоение элементу logged_in значения true
}

function logout()                                   // Завершение сессии
{
    $_SESSION = [];                                 // Очистка содержимого массива

    $params = session_get_cookie_params();      // Получение параметров сессионных cookie
    setcookie('PHPSESSID', '', time() - 3600, $params['path'], $params['domain'],
        $params['secure'], $params['httponly']);    // Удаление сессионного cookie

    session_destroy();                              // Удаление файла сессии
}

function require_login($logged_in)                  // Проверка авторизации пользователя
{
    if ($logged_in == false) {                      // Если не авторизован
        header('Location: login.php');              // Отправить на страницу входа
        exit;                                       // Остановить выполнение кода
    }
}
```

КАК УБЕДИТЬСЯ В ТОМ, ЧТО ПОЛЬЗОВАТЕЛЬ МОЖЕТ ПРОСМАТРИВАТЬ СТРАНИЦУ

Функция `require_login()` должна вызываться в начале любой страницы, которая требует от посетителей входа в систему.

В этом примере посетители должны войти в систему, чтобы просмотреть страницу `account.php`.

1. Подключение файла `sessions.php`.
2. Функция `require_login()`, определенная в файле `sessions.php`, проверяет, вошел ли пользователь в систему. Если:

- он вошел в систему, отображается остальная часть страницы;
- он не вошел в систему, то он отправляется на страницу `login.php`.

Единственный ее аргумент — переменная `$logged_in`, которая объявляется в шаге 2 в файле `sessions.php`.

3. Подключается новый файл с заголовком сайта (см. третий блок кода).
4. Далее показана страница `login.php`. Она начинается с подключения файла `sessions.php`.
5. В конструкции `if` проверяется значение переменной `$logged_in` (созданной в файле `sessions.php`), чтобы узнать, авторизовался ли уже пользователь.
6. Если авторизация пройдена, он перенаправляется на страницу `account.php`, поскольку вход в систему уже не требуется (возможно, пользователь попал на эту страницу, потому что нажал на ссылку или воспользовался кнопкой «Возврат» в браузере).
7. Команда `exit` останавливает выполнение остального кода на странице.
8. Если же код все еще продолжает выполнение, то дальше идет проверка, отправил ли пользователь форму (см. внизу страницы).
9. Если форма была отправлена, то значения, введенные пользователем в поля ввода для электронной почты и пароля, присваиваются переменным `$user_email` и `$user_password`.
10. В конструкции `if` проверяется, совпадают ли введенные пользователем адрес электронной почты и пароль с сохраненными в переменных `$email` и `$password` файла `sessions.php` (см. шаг 3 на предыдущей странице).
11. Если они совпадают, то вызывается функция `login()` (определенная в файле `sessions.php`). Она генерирует новый ID сессии и добавляет элемент `logged_in` в суперглобальный массив `$_SESSION` со значением `true`, указывающий, что пользователь вошел в систему.
12. Затем пользователь перенаправляется на страницу `account.php`. Команда `exit` останавливает дальнейшее выполнение кода.
13. Если форма не была отправлена или данные для входа были неверными, код продолжает выполнение и подключает файл с заголовком сайта.
14. Форма авторизации содержит два поля ввода: адрес электронной почты и пароль.
15. В файле, который содержит новый заголовок сайта, в панели навигации происходит проверка, вошел ли пользователь в систему. Если вошел, то отображается ссылка на страницу выхода из системы. Если нет, то будет показана ссылка на страницу входа в систему.

Примечание: идентификатор сессии отправляется в заголовках HTTP с каждым запросом. Если кто-то завладеет идентификатором сессии, он может подделать HTTP-запрос, добавив к нему этот идентификатор и выдав себя за пользователя, создавшего сессию. Это называется **перехватом сессии**.

Чтобы предотвратить перехват сессии, доступ к любым страницам, использующим сессии, должен осуществляться только через HTTPS-соединение, поскольку оно шифрует все данные (включая заголовок, содержащий идентификатор сессии). Данный пример не требует SSL-сертификата, но на любом действующем сайте сертификат должен быть установлен.

PHP section_b/c09/account.php

```php
<?php
① include 'includes/sessions.php';   // Подключение файла sessions.php
② require_login($logged_in);         // Перенаправление, если пользователь не авторизовался
?>
③ <?php include 'includes/header-member.php'; ?> ...
```

PHP section_b/c09/login.php

```php
<?php
④ include 'includes/sessions.php';

⑤ if ($logged_in) {                          // Если уже авторизовался
⑥     header('Location: account.php');       // Перенаправить на страницу учетной записи
⑦     exit;                                  // Остановить дальнейшее выполнение
}

⑧ if($_SERVER['REQUEST_METHOD'] == 'POST') {   // Если форма отправлена
⑨     $user_email    =  $_POST['email'];       // Введенный пользователем email
      $user_password =  $_POST['password'];    // Введенный пользователем пароль

⑩     if ($user_email == $email and $user_password == $password) { // Если данные верны
⑪         login();                           // Вызов функции авторизации (login)
⑫         header('Location: account.php');   // Перенаправление на страницу учетной записи
          exit;                              // Остановить дальнейшее выполнение
      }
}
?>
⑬ <?php include 'includes/header-member.php'; ?>
<h1>Login</h1>
<form method="POST" action="login.php">
⑭   Email: <input type="email" name="email"><br>
    Password: <input type="password" name="password"><br>
    <input type="submit" value="Log In">
</form>
<?php include 'includes/footer.php'; ?>
```

PHP section_b/c09/includes/header-member.php

```php
<a href="home.php">Home</a>
<a href="products.php">Products</a>
<a href="account.php">My Account</a>
⑮ <?= $logged_in ? '<a href="login.php">Log In</a>' : '<a href="logout.php">Log Out</a>' ?>
```

ЗАКЛЮЧЕНИЕ
COOKIE И СЕССИИ

- Cookie хранят данные в браузере посетителя.
- Данные cookie, полученные из браузера, доступны для кода PHP через суперглобальный массив `$_COOKIE`.
- Вы можете установить срок жизни cookie (но пользователи могут удалить их раньше).
- Сессии хранят данные на сервере.
- Доступ к данным сессии и их обновление осуществляются с помощью суперглобального массива `$_SESSION`.
- Каждая страница сайта, использующего сессии, должна начинаться с вызова функции `session_start()`.
- Файлы сессий сохраняют данные в течение одного посещения сайта и удаляются после периода бездействия.
- Если вы хотите хранить данные в течение более длительного времени или сохранять личную информацию, используйте для хранения базу данных, как показано в главе 16.

10

ОБРАБОТКА ОШИБОК

Если у интерпретатора PHP возникают сложности с выполнением вашего кода, он может выдать сообщение, которое поможет определить, в чем заключается проблема.

Когда вы создаете новый скрипт на PHP, не стоит ожидать, что он с первой попытки заработает идеально. Даже опытные программисты регулярно получают сообщения об ошибках при тестировании нового кода.

Видеть ошибки, конечно, неприятно, но при этом важно понимать, что информация, которую предоставляет PHP в сообщении об ошибке, помогает выявить причину проблемы и предоставляет информацию, помогающую ее устранить. Интерпретатор PHP поддерживает два способа информирования о проблемах, с которыми он сталкивается: ошибки и исключения.

- **Ошибки** (**Errors**) — это выдаваемые интерпретатором PHP сообщения. Они появляются в случае возникновения проблем с выполнением кода. Это подобно тому, как если бы интерпретатор PHP поднял руку и сказал вам: «Здесь что-то не так». Некоторые ошибки останавливают выполнение кода на странице, а некоторые — нет.
- **Исключения** (**Exceptions**) — это объекты, создаваемые либо интерпретатором PHP, либо программистом. Эти объекты создаются, когда код, который обычно выполняется без сбоев, не смог выполниться из-за какой-либо исключительной ситуации. Когда создается объект исключения, интерпретатор PHP прекращает выполнение кода и ищет альтернативный блок кода, написанный для обработки такой ситуации. Это дает возможность исправить проблему и восстановить нормальный ход выполнения после нее.

  Исключения подобны тому, что интерпретатор PHP или программист как бы говорят: «Здесь что-то не так. Есть ли инструкции по устранению проблемы?» Если нет альтернативного блока кода для обработки такой ситуации, то будет выдана ошибка, останавливающая выполнение скрипта.

Ошибки и исключения содержат информацию, которая помогает вам понять, в чем заключается проблема и где она возникла. Эти сообщения могут отображаться прямо на выводимой в браузер веб-странице или сохраняться в текстовом файле на сервере, известном как лог (журнал) ошибок.

В дополнение к сообщениям об ошибках, создаваемым интерпретатором PHP, веб-сервер также может выдавать собственные ошибки, которые он отправляет в браузер. Сервер делает это, когда не может найти запрошенный браузером файл или по какой-либо другой причине не может обработать запрос.


Chapter 10 Examples
phpandmysql.com
MOUNTAIN ART SUPPLIES
Sorry, a problem occurred
The site's owners have been informed. Please try again later.
Type here to search
Lenovo
JAVASCRIPT & JQUERY
30-SECOND THEORIES
Art Map
MASS MoCA
Canon
EOS 1300D
Canon

УПРАВЛЕНИЕ ОТОБРАЖЕНИЕМ ОШИБОК PHP

Если интерпретатор PHP сталкивается с проблемой при выполнении кода, он создает сообщение об ошибке, которое описывает возникшую проблему. Это сообщение отображается в браузере или сохраняется в файле на сервере.

Когда сайт находится в стадии разработки, сообщения об ошибках, создаваемые интерпретатором PHP, могут выводиться прямо на экран.
Это позволяет программисту сразу узнавать обо всех проблемах, как только он обновит страницу в браузере, и тут же исправить их.

Но если сайт «заступает на боевое дежурство» и возникают ошибки, которые не были обнаружены во время разработки, то сообщения о них **не должны** отображаться на веб-странице, поскольку они плохо понятны посетителям, а кроме того, могут подсказать хакерам, как устроен сайт.

Вместо этого PHP должен показать понятное пользователю сообщение. Само же исходное сообщение об ошибке должно быть добавлено в текстовый файл на сервере, называемый **файлом логов** (**log file**), с тем чтобы разработчик мог заглянуть в него и узнать, не было ли каких-либо ошибок после запуска сайта в эксплуатацию.

Интерпретатор PHP поддерживает три настройки поведения при ошибках:

- должны ли ошибки отображаться на экране;
- следует ли записывать их в файл логов;
- какого типа ошибки следует порождать (при изучении PHP или во время разработки сайта вы должны включать ошибки *всех* уровней).

Этими настройками можно управлять с помощью файлов `php.in` или `.htaccess`.

- Файл `php.in` содержит настройки по умолчанию для всех файлов на веб-сервере. После его редактирования необходимо перезапустить сервер, чтобы изменения вступили в силу.
- Файл `.htaccess` может быть размещен в любой папке на веб-сервере. Он управляет всеми файлами в этой папке и любыми вложенными папками. Сервер не нуждается в перезапуске при изменении файлов `.htaccess`.

ФАЙЛ php.ini

В файле `php.ini` должны быть установлены значения перечисленных ниже настроек. Они указывают интерпретатору PHP показывать ошибки на экране, записывать их в лог-файл и сообщать об ошибках всех уровней:

```
display_errors = On
log_errors = On
error_reporting = E_ALL
```

Когда сайт запускается в рабочем режиме, для параметра `display_errors` должно быть установлено значение `Off`, чтобы предотвратить отображение ошибок на экране.

```
display_errors = Off
```

ФАЙЛ .htaccess

Если в файл `.htaccess` добавить следующие настройки, то интерпретатор PHP будет сообщать обо всех ошибках, отображать их на экране и записывать в файл логов:

```
php_flag display_errors On
php_flag log_errors On
php_value error_reporting -1
```

Когда сайт запускается в рабочем режиме, для параметра `display_errors` должно быть установлено значение `Off`, чтобы предотвратить отображение ошибок на экране.

```
php_flag display_errors Off
```

Загружаемый код для книги использует файлы `.htaccess` для управления настройками интерпретатора PHP. Некоторые папки содержат собственные файлы `.htaccess` для управления настройками конкретного набора примеров. Примеры кода в этой главе находятся в двух папках: одна для сайтов на этапе разработки, а другая — для действующих.

PHP section_b/c10/development/.htaccess

```
① php_flag   display_errors  On
  php_flag   log_errors      On
  php_value  error_reporting -1
```

PHP section_b/c10/live/.htaccess

```
② php_flag   display_errors  Off
  php_flag   log_errors      On
  php_value  error_reporting -1
```

PHP section_b/c10/development/find-error-log.php

```
  Your error log is stored here:
③ <?= ini_get('error_log') ?>
```

PHP section_b/c10/development/sample-error.php

```
  <?php
④ echo 'Finding an error";
  ?>
```

РЕЗУЛЬТАТ

Parse error: syntax error, unexpected string content "Finding an error";" in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/sample-error.php** on line **2**

РЕЗУЛЬТАТ

```
[27-Jan-2021 14:41:13 UTC] PHP Parse error:  syntax error,
unexpected string content "Finding an error";" in
/Users/Jon/Sites/localhost/phpbook/section_b/c10/
development/sample-error.php on line 2
```

Примечание: файлы логов должны храниться выше корневого каталога сайта, чтобы хакеры не могли угадать их путь и запросить их по URL-адресу.

1. Эти настройки следует использовать во время разработки сайта. Они указывают интерпретатору PHP отображать все ошибки на экране, а также записывать их в лог-файл.
2. Когда сайт начнет принимать посетителей, сообщения об ошибках не должны отображаться в браузере. Вместо этого их следует вносить в лог-файл, чтобы разработчики могли видеть, возникают ли у пользователей какие-либо проблемы.
3. Веб-сервер может размещать файлы логов в разных папках. Чтобы узнать, где находится файл логов ошибок на вашем сервере, используйте встроенную в PHP функцию `ini_get()`.
4. Эта простая страница на PHP выдает ошибку, поскольку кавычки не совпадают. Сообщение об ошибке, создаваемое интерпретатором PHP при запуске этого файла, показано ниже.

На первой иллюстрации сообщение об описанной выше ошибке отображается прямо в браузере. Как видите, это сообщение не ориентировано на восприятие пользователем. Следующие восемь страниц помогут вам понять, что означают эти сообщения об ошибках.

На второй иллюстрации отображается содержимое лог-файла. Файл логов можно открыть в текстовом редакторе или редакторе кода. Сообщение об ошибке такое же, как и показанное на экране, но ему предшествуют дата и время появления сообщения.

ПОНИМАНИЕ СООБЩЕНИЙ ОБ ОШИБКАХ

Сообщения об ошибках, создаваемые интерпретатором PHP, на первый взгляд могут показаться запутанными, но на самом деле они имеют четкую структуру и содержат четыре блока информации, которые помогают разработчику найти источник ошибки.

Программисты говорят, что интерпретатор PHP **выдает** ошибки, когда он их находит. Сообщения об ошибках используют показанную ниже структуру и содержат четыре фрагмента данных:

- первые два (уровень и описание) описывают возникшую ошибку;
- следующие два (путь к файлу и номер строки) указывают вам, где начать поиск проблемы.

Уровень ошибки (**error level**) описывает общий тип (или уровень) проблемы, с которым столкнулся интерпретатор PHP.

Описание (**description**) — это подробное объяснение ошибки.

Путь к файлу (**filepath**) — это имя файла, в котором была обнаружена ошибка.

Номер строки (**line number**) — это строка в этом файле, на которой произошла ошибка.

Основные **типы** ошибок, с которыми вы, вероятно, столкнетесь, описаны на странице справа. Затем на следующих шести страницах показаны примеры файлов PHP, содержащих ошибки каждого типа, а также пояснения, рассказывающие, как найти и исправить ошибку.

Некоторые ошибки возникают до того, как интерпретатор PHP сообщит об ошибке, но имя файла и номер строки подсказывают вам, с какого места надо начать поиск.



УРОВНИ/ТИПЫ ОШИБОК

Основные типы ошибок PHP представлены ниже. Некоторые ошибки останавливают работу интерпретатора PHP и должны быть исправлены для того, чтобы страница могла продолжить работу. Другие могут показаться больше похожими на рекомендации, но они все равно должны быть исправлены.

СИНТАКСИЧЕСКИЕ ОШИБКИ

Ошибка парсинга указывает на наличие ошибки в синтаксисе PHP-кода. Это не позволяет интерпретатору PHP прочитать или провести парсинг (синтаксический анализ) файла, поэтому он даже не будет пытаться выполнить этот код.

Если в настройках интерпретатора PHP указано, что ошибки должны отображаться на экране, то будет выведена только эта ошибка. Если ошибки не отображаются на экране, то посетитель увидит пустую страницу.

Ошибки парсинга должны быть исправлены, чтобы код на странице мог был выполнен и отображал что-либо, кроме сообщения об ошибке.

ФАТАЛЬНЫЕ ОШИБКИ

Фатальная ошибка говорит о том, что интерпретатор PHP посчитал синтаксис кода PHP допустимым и попытался его выполнить, но что-то помешало ему отработать правильно.

Интерпретатор PHP останавливается на строке кода, где обнаружена фатальная ошибка. Это означает, что на экране можно увидеть созданную до обнаружения интерпретатором PHP часть страницы.

Начиная с PHP 7 большинство фатальных ошибок создают объект исключения. Это дает программистам возможность обработать такую ошибку.

НЕФАТАЛЬНЫЕ ОШИБКИ

Нефатальная ошибка приведет к появлению сообщения о ней. Сообщение укажет на возможную проблему, но код будет продолжать выполняться.

- **Предупреждение (warning)** — это нефатальная ошибка, сообщающая, что интерпретатор PHP столкнулся с ошибкой, но может продолжать работу.
- **Уведомление (notice)** — это нефатальная ошибка, сообщающая о возможных проблемах в коде.

В PHP 8 многие уровень многих уведомлений был изменен на предупреждения.

Примечание. Уровни ошибок и их описания меняются в зависимости от разных версий PHP. В этой главе показаны сообщения об ошибках из PHP 8.

Если вы видите ошибку, начинающуюся со слова `Deprecated`, это означает, что такой функционал в скором времени будет удален из PHP.

Если вы увидите сообщение об ошибке, начинающееся со слова `Strict`, оно будет содержать рекомендации по улучшению вашего PHP-кода.

СИНТАКСИЧЕСКИЕ ОШИБКИ

Ошибки парсинга (parse errors) вызваны проблемами с синтаксисом вашего кода. Это препятствует отображению страницы, потому что интерпретатор PHP не может понять код. Для запуска кода необходимо исправить ошибки парсинга.

Синтаксические ошибки часто вызываются опечаткой (несоответствующая кавычка или пропущенная точка с запятой, круглая или квадратная скобка). Эти простые ошибки могут помешать интерпретатору PHP прочитать код. Чтобы исправить ошибки парсинга, найдите строку, на которую указывает сообщение об ошибке. Прочитайте строку слева направо и проверьте каждую инструкцию в ней. Если вы не видите проблемы, посмотрите на предыдущую строку.

Когда для параметра `display_errors` установлено значение `Off`, ошибка парсинга приводит к тому, что пользователь видит пустой экран. При возникновении такой ошибки вы должны найти ее и исправить, иначе страница не сможет что-либо показать.

В строке 2 этого примера строка, присваиваемая переменной, использует несовпадающие кавычки: одна одинарная, другая двойная.

В сообщении об ошибке говорится, что проблема возникла в строке 3, потому что интерпретатор PHP не понимал, что произошла ошибка, до тех пор, пока он не нашел еще одну одинарную кавычку.

Когда он наткнулся на другую одинарную кавычку в строке 3, он обработал эту кавычку так, как если бы она закрывала переменную из строки 2.

В сообщении говорится, что в строке 3 есть `unexpected identifier 'pencil'`, потому что текст после второй одинарной кавычки — это слово `pencil`.

section_b/c10/development/parse-error-1.php **PHP**

```
1 <?php
2 $username = 'Ivy";
3 $order    = ['pencil', 'pen', 'notebook',];
4 ?>
5 <h1>Basket</h1>
6 <?= $username ?>
7 <?php foreach ($order as $item) { ?>
8     <?= $item ?><br>
9 <?php } ?>
```

РЕЗУЛЬТАТ

Parse error: syntax error, unexpected identifier "pencil" in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-error-1.php** on line **3**

Примечание. Если вы используете редактор кода с подсветкой синтаксиса, цвет кода обычно дает подсказки о местонахождении синтаксической ошибки.

PHP section_b/c10/development/parse-error-2.php

```
1 <?php
2 $username = 'Ivy'
3 $order    = ['pencil', 'pen', 'notebook',];
4 ?> ...
```

РЕЗУЛЬТАТ

Parse error: syntax error, unexpected variable "$order" in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-error-2.php** on line **3**

PHP section_b/c10/development/parse-error-3.php

```
1 <?php
2 $username = 'Ivy';
3 $order    = ['pencil', 'pen', 'notebook',);
4 ?> ...
```

РЕЗУЛЬТАТ

Parse error: Unclosed '[' does not match ')' in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-error-3.php** on line **3**

PHP section_b/c10/development/parse-error-4.php

```
1 <?php
2 $username = 'Ivy';
3 order     = ['pencil', 'pen', 'notebook',];
4 ?> ...
```

РЕЗУЛЬТАТ

Parse error: syntax error, unexpected identifier "order" in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-error-4.php** on line **3**

Чтобы найти ошибку парсинга, закомментируйте нижнюю половину страницы. Если вы все еще видите ту же ошибку, значит, проблема находится в первой половине страницы. Если нет, значит, она во второй. Повторите этот процесс, чтобы еще больше сузить круг поиска ошибки.

Упражнение. Чтобы убедиться, что вы поняли проблемы в этом разделе, попробуйте исправить ошибки, описанные в файле, а затем снова запустите примеры.

В конце строки 2 должна быть точка с запятой. В сообщении говорится `unexpected variable '$order' on line 3` (неожиданная переменная `$order` в строке 3), потому что в конце предыдущей инструкции не стоит точка с запятой.

Как показывают первые два примера, ошибки парсинга часто возникают в строке, предшествующей строке, указанной в качестве ошибочной. Если вы не видите проблемы в строке из сообщения, посмотрите на предыдущие строки выполняемого кода.

В строке 3 создается массив. Он начинается с квадратной открывающей скобки, но заканчивается круглой закрывающей скобкой.

Здесь сообщение об ошибке показывает правильную строку и точную проблему. В нем говорится, что `unclosed '[' does not match ')' on line 3` (незакрытая скобка '[' не совпадает с ')' в строке 3).

В строке 3 имя переменной не начинается с символа **$**.

В сообщении об ошибке говорится `unexpected identifier 'order' on line 3` (неожиданный идентификатор 'order' в строке 3), потому что `order` не является ключевым словом или командой, которую может выполнить интерпретатор PHP (он не знает, что это должно быть имя переменной, потому что перед словом нет символа **$**).

ФАТАЛЬНЫЕ ОШИБКИ

Фатальная ошибка возникает, когда интерпретатор PHP обнаруживает проблему, не позволяющую ему выполнять код дальше. Это означает, что пользователи могут видеть страницу только частично, до того момента, когда возникла фатальная ошибка.

Если вы видите фатальную ошибку, это значит, что интерпретатор смог прочитать PHP код и приступил к его выполнению, но в какой-то момент возникла ошибка.

Если до возникновения ошибки был начат вывод данных в браузер, то пользователь может увидеть эту часть страницы. Если нет, то он увидит пустую страницу.

При фатальных ошибках вы должны разобраться, почему интерпретатор PHP не может обработать код, а затем устранить проблему, чтобы страница могла отобразиться целиком.

В этом примере код в 4 строке пытается умножить целое число на строку (число присвоено переменной в `$price` в строке 2, а строковое значение — переменной `$quantity` в строке 3).
В сообщении об ошибке говорится `Unsupported operand types int * string` (неподдерживаемые типы операндов int * string), чтобы показать, что целое число не может быть умножено на строку. Поскольку проблема обнаруживается до того, как начинает выводиться какой-либо контент, то заголовок с надписью `Basket` и текст со словом `Total:` не отображаются посетителю.

Чтобы предотвратить появление этой ошибки, в код можно добавить проверку, что оба значения являются числами, прежде чем пытаться их умножить.

section_b/c10/development/fatal-error-1.php **PHP**

```
1 <?php
2 $price    = 7;
3 $quantity = 'five';
4 $total    = $price * $quantity;
5 ?>
6 <h1>Basket</h1>
7 Total: $<?= $total ?>
```

РЕЗУЛЬТАТ

Fatal error: Uncaught TypeError: Unsupported operand types: int * string in /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-1.php:4 Stack trace: #0 {mai thrown in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-1.php** on line

До версии PHP 7.4 этот пример выдавал предупреждение, а не фатальную ошибку.

Примечание. До версии PHP 7 код не мог восстановить работу после фатальной ошибки. В PHP 7 многие фатальные ошибки начали создавать объект исключения, что дает возможность обработать проблемную ситуацию. Если исключение не обработать (не перехватить), то оно становится фатальной ошибкой и сообщение об ошибке в этом случае начинается со слов `Uncaught error` (неперехваченная ошибка).

PHP section_b/c10/development/fatal-error-2.php

```
1 <?php
2 function total(int $price, int $quantity) {...}
6 ?>
7 <h2>Basket</h2>
8 <?= totals(3, 5) ?>
```

РЕЗУЛЬТАТ

Basket

Fatal error: Uncaught Error: Call to undefined function totals() in /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-2.php:8 Stack trace: #0 {main} thrown in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-2.php** on line **8**

PHP section_b/c10/development/fatal-error-3.php

```
1 <?php
2 function total(int $price, int $quantity) {...}
6 ?>
7 <h2>Basket</h2>
8 <?= total(3) ?>
```

РЕЗУЛЬТАТ

Basket

Fatal error: Uncaught ArgumentCountError: Too few arguments to function total(), 1 passed in /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-3.php on line 8 and exactly 2 expected in /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-3.php:2 Stack trace: #0 /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-3.php(8): total(3) #1 {main} thrown in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/fatal-error-3.php** on line **2**

PHP section_b/c10/development/fatal-error-4.php

```
1 <?php $basket = new Basket(); ?><h2>Basket</h2>
```

РЕЗУЛЬТАТ

Fatal error: Uncaught Error: Class 'Basket' not found in /Applications/MAMP/htdocs/phpbook/section_b/c10/development/fatal-error-4.php:1 Stack trace: #0 { thrown in **/Applications/MAMP/htdocs/phpbook/section_b/c10/development/fatal-error-4.php** on

Примечание. Когда объект исключения создается (или выбрасывается) внутри функции или метода, **трассировка стека вызовов функций** (показанная в сообщении об ошибке) показывает нам цепочку вызовов, которые привели к данному участку кода, и в частности — имя файла и номер строки, на которой была вызвана эта функция или метод.

В строке 2 объявляется функция с именем `total()` (полное определение функции находится в загружаемом коде). В строке 8 код вызывает функцию с именем `totals()`.

В сообщении об ошибке говорится `Call to undefined function totals()`(вызов неопределенной функции totals()), потому что функции `totals()` нет, а нужно вызывать функцию `total()`. Заголовок `Basket` отображается потому, что интерпретатор PHP прекращает выполнение кода только после обнаружения ошибки.

В строке 8 вызывается функция `total()`, но она вызывается только с одним аргументом, а не с двумя.

В сообщении об ошибке говорится `Too few arguments to function total()`(слишком мало аргументов для функции total()), поскольку функции требуются два аргумента.
В сообщении также указано, что был передан один аргумент, а ожидаются именно два. Чтобы исправить это, функция должна быть вызвана с правильным количеством аргументов.

В строке 1 создается объект класса `Basket`, но определение класса отсутствует. В сообщении говорится `Class 'Basket' not found` (класс 'Basket' не найден). Интерпретатор не может продолжить отображение остальной части страницы, потому что не может создать объект. Чтобы исправить это, сначала нужно добавить определение класса.

НЕФАТАЛЬНЫЕ ОШИБКИ (ПРЕДУПРЕЖДЕНИЯ И УВЕДОМЛЕНИЯ)

Нефатальная ошибка возникает, когда интерпретатор PHP сталкивается с проблемой, но может продолжить выполнение кода.

Предупреждение (warning) говорит о том, что интерпретатор PHP столкнулся с явной ошибкой, а уведомление (notice) указывает на возможную ошибку. Оба этих типа ошибок являются нефатальными. Они возникают, когда интерпретатор PHP сталкивается с проблемой, но может продолжить выполнение кода. Но это не значит, что на такие ошибки не нужно обращать внимание. Все ошибки должны быть исправлены, так как они могут серьезно повлиять на правильность выполнения кода (это показано в примере ниже).

В этом примере объявлены три переменные.

- В строке 2 объявляется переменная `$price`. Ее значение равно числу 7.
- В строке 3 объявляется переменная `$quantity`. Ее значение представляет собой строку, содержащую '0a'.
- В строке 4 объявляется переменная `$total`. Ее значение должно быть значением `$price`, умноженным на значение `$quantity`.

section_b/c10/development/warning-1.php **PHP**

```
1 <?php
2 $price    = 7;
3 $quantity = '0a';
4 $total    = $price * $quantity;
5 ?>
6 <h1>Basket</h1>
7 Total: $<?= $total ?>
```

РЕЗУЛЬТАТ

Warning: A non-numeric value encountered in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/warning-1.php** on line **4**

Basket

Total: $0

Строка 4 создает предупреждение об обнаружении нечислового значения (`A non-numeric value encountered`), поскольку `$quantity` содержит недопустимый для числа символ[41]. Поскольку первый символ в строке — число 0, интерпретатор PHP пытается привести значение к числовому типу, используя первый символ 0 и игнорируя остальную часть строки, на при этом выдает предупреждение.
Затем страница продолжает работать, и на ней отображается сохраненное в `$total` значение.

Это может стать большой проблемой для сайта, потому что общая сумма, как показано, равна нулю[42].
Вы можете использовать функцию PHP `var_dump()` для проверки значений в переменных и их типа данных.

PHP section_b/c10/development/warning-2.php

```
1 <?php $list = false; ?>
2 <h1>Basket</h1>
3 <?php foreach ($list as $item) { ?>
4     Item: <?= $item ?><br>
5 <?php } ?>
```

РЕЗУЛЬТАТ

Basket

Warning: foreach() argument must be of type arraylobject, bool given in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/warning-2.php** on line **3**

В строке 1 переменная `$list` должна была содержать массив, но ей было присвоено логическое значение `false`. Затем, в строке 3, цикл `foreach` пытается перебрать элементы массива `$list`.

Сообщение об ошибке `foreach() argument must be of type array|object` говорит о том, что интерпретатор не может выполнять цикл по логическому значению.

Примечание. Ошибка не возникает, если цикл `foreach` используется с пустым массивом или не имеющим свойств объектом.

PHP section_b/c10/development/warning-3.php

```
1 <?php include 'header.php'; ?>
2 <h1>Basket</h1>
```

РЕЗУЛЬТАТ

Warning: include(header.php): Failed to open stream: No such file or directory in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/warning-3.php** on line **1**

Warning: include(): Failed opening 'header.php' for inclusion (include_path='.:/Applications/MAMP/bin/php/php8.0.0/lib/php') in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/warning-3.php** on line **1**

Basket

В строке 1 должен быть подключен файл, но он не может быть найден. В результате отображаются два сообщения об ошибках:

- `Failed to open stream: No such file or directory…`, которая говорит о том, что файл не найден;
- `Failed opening … for inclusion`, которая означает, что файл не может быть подключен в код.

При возникновении ошибки в конструкции `include` интерпретатор PHP продолжает выполнение кода, выводя оставшийся контент в браузер.

PHP section_b/c10/development/warning-4.php

```
1 <?php $list = ['pencil', 'pen', 'notebook',]; ?>
2 <?= $list ?>
```

РЕЗУЛЬТАТ

Warning: Array to string conversion in **/Users/Jon/Sites/localhost/phpbook/section_b/c10/development/warning-4.php** on line **2**
Array

В строке 1 переменная `$list` содержит массив. В строке 2 сокращенная команда `echo` используется для вывода переменной `$list`.

Это вызывает ошибку `Array to string conversion`, которая говорит о том, что поскольку `echo` работает только со строками, то PHP пытается преобразовать массив в строку, но эта операция не удается, поэтому содержимое не может быть отображено. До PHP 8 это было уведомление, а не предупреждение.

ОТЛАДКА: ПОИСК ОШИБОК В КОДЕ

Иногда бывает непонятно, что является причиной ошибки. Либо код вообще не выдает ошибок, но при этом работает неправильно, возвращая неверный результат. Существует несколько методов, которые помогут вам разобраться с такой проблемой и исправить ее.

1. Вывод на экран отладочных сообщений позволяет узнать, как далеко продвинулся интерпретатор, прежде чем возникла ошибка. Здесь команда `echo` используется для отображения таких отладочных сообщений в строках 2, 9, 17, 24.

Сообщение в строке 9 будет показано только в том случае, если функция `total()` будет вызвана. То же самое относится и к строке 18.

2. Комментирование отдельных участков скрипта уменьшает объем потенциально проблемного кода. Закомментировав в строках 20 и 23 команды подключения хидера и футера, можно убедиться в том, что ошибка не в этих файлах. Вы также можете закомментировать вызовы функций, чтобы проверить, не внутри ли функции возникает ошибка (если функция возвращает какое-то значение, то вместо вызова функции нужно будет вписать временное значение вручную).

3. Функция PHP `var_dump()` выводит значения переменных и их тип, чтобы вы могли проверить, содержит ли переменная ожидаемое значение. Она используется в строке 26 для проверки значения переменной `$basket`. И вот как раз здесь можно увидеть, что значение третьего элемента в массиве `$basket` является строкой `two`, а не числом, что и приводит к ошибке.

РЕЗУЛЬТАТ

1: Start of page

2: Before function called

3: Inside total() function

Warning: A non-numeric value encountered in
/Applications/MAMP/htdocs/phpbook/section_b/c10/development/tracking-down-errors.php on line **12**

Basket

Total: $2.00

4: End of page

$basket: array(3) { ["pen"]=> float(1.2) ["pencil"]=> float(0.8) ["paper"]=> string(3) "two" }

Test total() function:

3: Inside total() function

4

PHP section_b/c10/development/tracking-down-errors.php

```php
<?php
echo '<p><i>1: Start of page</i></p>';
$basket['pen']    = 1.20;
$basket['pencil'] = 0.80;
$basket['paper']  = 'two';

function total(array $basket): int
{
echo '<p><i>3: Inside total() function</i></p>';
    $total = 0;
    foreach ($basket as $item => $price) {
        $total = $total + $price;
    }
    return $total;
}

echo '<p><i>2: Before function called</i></p>';
$total = total($basket);
?>
<?php // include 'header.php' ?>
<h3>Basket</h3>
<p><b>Total: $<?= number_format($total, 2) ?></b></p>
<?php // include 'footer.php' ?>
<?php echo '<p><i>4: End of page</i></p>'; ?>
<hr><!-- Весь оставшийся тестовый код -->
<p><b>$basket:</b> <?= var_dump($basket) ?></p>
<b>Test total() function:</b>
<?php
$testbasket['pen']    = 1.20;
$testbasket['pencil'] = 0.80;
$testbasket['paper']  = 2;
?>
<?= total($testbasket) ?>
```

Вывод отладочных сообщений, помеченный цифрой 1, оформлен без отступов. Это облегчает их поиск и удаление, после того как проблема найдена и исправлена.

Профессиональные программисты используют отладчики, встроенные в некоторые редакторы PHP-кода. Он позволяют выполнять код построчно и сразу видеть всю описанную выше информацию без необходимости добавлять (и потом удалять) отладочный вывод вручную. Это называется **пошаговой отладкой** (**stepping through**) кода. На каждом шаге редактор позволяет проверить значения всех переменных, чтобы выяснить место, где код начал работать неправильно. Чтобы не выполнять пошагово все приложение с самого начала, вы можете задать в коде контрольную точку, на которой редактор остановит выполнение кода, и начать пошаговую отладку с этого места.

4. Вы также можете создавать тесты для функций и методов, чтобы проверить, выдают ли они ожидаемый результат.

Когда в коде используются функции или методы, тестирование каждого из них в отдельности — хороший способ проверить, что они работают правильно (вместо того чтобы проверять их в контексте всей страницы).

Посмотрите, как выполняется тест для функции `total()` в конце страницы. В массив `$testbasket` заносятся тестовые значения, затем функция вызывается и выводится полученное значение. Теперь программист должен самостоятельно посчитать ожидаемый результат и сравнить его с полученным. Если они совпали, то функция работает корректно.

Обратите внимание, что сообщение `echo` в строке 9 отображается снова, когда функция `total()` вызывается во второй раз в строке 33.

Часто программисты пишут специальные скрипты для тестирования каждой функции. Если вы знаете, что протестированная отдельно функция работает корректно, это означает, что сам код функции не должен вызывать проблем. Следовательно, необходимо проверить те значения, которые передаются в функцию.

Если переданные в функцию параметры оказываются неверными, то нужно отследить выше по коду, откуда они берутся, чтобы найти источник ошибки.

Примечание. Правильное форматирование кода, когда инструкции внутри фигурных скобок отступают на 4 пробела, не только придает коду логическую стройность, но и облегчает поиск множества проблем, например таких, как отсутствие закрывающей скобки.

ЗАПУСК САЙТА В ЭКСПЛУАТАЦИЮ

Когда сайт готов к приему посетителей, настройки PHP необходимо изменить, чтобы интерпретатор не отображал сообщения об ошибках в браузере. Вместо этого они должны сохраняться в файле логов, который необходимо регулярно проверять.

Даже если сайт был тщательно протестирован, некоторые ошибки все равно могут быть пропущены. Или же могут возникнуть проблемы с сервером. Поэтому на рабочем сервере (программисты часто говорят «боевом») используйте настройки в файлах `php.ini` или `.htaccess`, чтобы:

- не показывать сообщения об ошибках на экране;
- сохранять сообщения об ошибках в логе ошибок.

Файл логов можно открыть в текстовом редакторе или редакторе кода. Каждое сообщение в нем начинается с даты и времени возникновения ошибки, за которыми следуют сообщения, идентичные тем, которые вы видели на экране. Если вы уже выполнили примеры, приведенные в этой главе, файл логов будет содержать приведенные ниже ошибки.

Ошибки должны исправляться на **тестовой** копии сайта (на тестовом сервере или на вашем локальном компьютере), а не на реальном сайте. Для каждой ошибки в файле логов:

1) попробуйте воссоздать ошибку, чтобы узнать, что вызвало появление сообщения;
2) используйте методы, показанные на предыдущих страницах, чтобы найти код, вызывающий ошибку;
3) исправьте код, вызывающий ошибку.

Как только проблема будет устранена, код следует протестировать еще раз, прежде чем новая версия кода будет загружена на действующий сайт, поскольку исправление могло привести к возникновению новых проблем.

```
[27-Jan-2021 14:56:44 UTC] PHP Parse error:  syntax error, unexpected string
content "Finding an error";" in /Users/Jon/Sites/localhost/phpbook/section_b/c10/
development/sample-error.php on line 2
[27-Jan-2021 14:56:51 UTC] PHP Parse error:  syntax error, unexpected identifier
"pencil" in /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-
error-1.php on line 3
[27-Jan-2021 14:57:02 UTC] PHP Parse error:  syntax error, unexpected variable
"$order" in /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-
error-2.php on line 3
[27-Jan-2021 14:57:04 UTC] PHP Parse error:  Unclosed '[' does not match ')' in /
Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-error-3.php on
line 3
[27-Jan-2021 14:57:06 UTC] PHP Parse error:  syntax error, unexpected identifier
"order" in /Users/Jon/Sites/localhost/phpbook/section_b/c10/development/parse-
error-4.php on line 3
```

Интерпретатор PHP также может сохранять ошибки в специализированных базах данных, но это не рассматривается в книге, поскольку это требуется для очень больших сайтов.

Файлы логов могут занимать много места на диске веб-сервера, поэтому его администратору необходимо следить за тем, чтобы логи регулярно архивировались или удалялись.

ОБРАБОТКА ОШИБОК

Когда возникает фатальная или нефатальная ошибка, интерпретатор PHP может вызвать заданную программистом функцию, называемую **обработчиком ошибок**. На работающем сайте такая функция поможет избежать того, что пользователи в случае ошибки увидят пустую или внезапно прерванную страницу.

ОБРАБОТКА НЕФАТАЛЬНЫХ ОШИБОК

Большинство нефатальных ошибок не прекращает выполнение кода. Это может вызвать непредсказуемое поведение кода или даже серьезные проблемы. Например, ранее ошибка привела к тому, что стоимость заказа составила 0 долларов.

Вы можете использовать **функцию обработки ошибок**, чтобы предотвратить дальнейшее выполнение кода до тех пор, пока ошибка не будет исправлена.

Встроенная в PHP функция `set_error_handler()` сообщает интерпретатору PHP имя функции, которая должна вызываться при возникновении ошибки. Имя функции передается в как параметр функции `set_error_handler()`в виде строки, причем оно пишется без скобок.

Эта функция может, к примеру, отображать понятное для пользователя сообщение, а затем останавливать выполнение кода.

Существует возможность указать во втором параметре, для ошибок каких уровней будет запускаться обработчик, но на этапе обучения лучше использовать его для всех нефатальных ошибок.

```
set_error_handler('name')
```

НЕФАТАЛЬНАЯ ОШИБКА
ФУНКЦИЯ ОБРАБОТКИ ОШИБОК

ОБРАБОТКА ФАТАЛЬНЫХ ОШИБОК

Фатальные ошибки останавливают выполнение кода, и функция, указанная в `set_error_handler()`, не выполняется.

Начиная с PHP 7 интерпретатор PHP вместо большинства фатальных ошибок выбрасывает исключения. Вы узнаете об исключениях и о том, как с ними обращаться, после следующего примера. Но если исключение не обрабатывается, оно тоже становится фатальной ошибкой.

Однако можно определить функцию, которая будет вызываться при завершении работы скрипта, ее называют **функцией завершения (shutdown function)**. Она вызывается всякий раз, когда выполнение кода прекращается либо естественным путем, либо после вызова команды `exit`, либо из-за фатальной ошибки.

Функция завершения проверяет, возникла ли ошибка во время работы страницы, и если да, то в этом случае можно показать понятное для пользователя сообщение об ошибке, а также записать ее в лог-файл.

Функция PHP `register_shutdown_function()` сообщает интерпретатору PHP имя функции, которую нужно вызвать, когда страница перестает выполняться. Обратите внимание, что при указании имени функции завершения круглые скобки писать не нужно.

```
register_shutdown_function('name')
```

ФАТАЛЬНАЯ ОШИБКА
ФУНКЦИЯ ОБРАБОТКИ ОШИБОК

ФУНКЦИЯ ОБРАБОТКИ НЕФАТАЛЬНЫХ ОШИБОК

Любые нефатальные ошибки сигнализируют о проблемах, которые могут привести к тому, что на сайте будет отображаться неверная информация. Поэтому при возникновении такой ошибки желательно все равно прекращать выполнение кода. Функция обработки фатальных ошибок обеспечивает запись сообщения об ошибке в лог-файл, информирование пользователей о проблемах на сайте в понятной форме и остановку дальнейшего выполнения кода.

В данном примере функция PHP `set_error_handler()` указывает интерпретатору PHP вызвать функцию с именем `handle_error()`, если он обнаружит нефатальную ошибку.
Если сайт использует свою функцию для обработки ошибок, то интерпретатор PHP не будет запускать собственный обработчик ошибок (если только эта функция не вернет значение `false`). Но если встроенный обработчик ошибок PHP не выполнится, то сообщение об ошибке не запишется в лог-файл и, как следствие, программист не узнает о том, что возникла проблема.
Данная функция не может вернуть `false`, поскольку она останавливает выполнение кода (после того как отобразит понятное для пользователя сообщение об ошибке). Таким образом, функция должна будет сама сохранить сообщение об ошибке в лог, прежде чем завершит работу.
Когда интерпретатор PHP вызывает функцию обработки ошибок, он передает ей четыре аргумента, содержащие информацию об ошибке (это те же значения, которые выводятся в сообщении об ошибке):

- уровень ошибки (в виде целого числа);
- сообщение об ошибке (в виде строки);
- путь к файлу, в котором произошла ошибка;
- строка кода, в которой была обнаружена ошибка.

Определение функции `handle_error()` должно включать четыре параметра, чтобы использовать эти значения.

В данном примере информация об ошибке используется для создания сообщения, добавляемого в лог. Формат сообщения об ошибке будет аналогичным тому, что создает интерпретатор PHP.
В PHP есть встроенная функция под названием `error_log()`. Ее можно использовать для отправки сообщений в лог ошибок. Ее единственным параметром будет сообщение об ошибке.
Поскольку произошла ошибка, функция также должна установить статус ответа HTTP равным 500, чтобы оповестить HTTP-клиент об ошибке на сервере. Это можно сделать с помощью функции PHP `http_response_code()` (но это сработает, только если функция вызывается до того, как что-либо будет выведено на страницу)[43]. Ее единственным параметром будет код состояния ответа (response status code).
Перед тем как отобразить посетителю сообщение об ошибке, страница использует команду `require_once`, чтобы включить файл заголовка. Это гарантирует, что страница с ошибкой сохранит тот же дизайн, что и остальная часть сайта. Команда `require_once` используется вместо `include`, чтобы гарантировать, что заголовок будет выведен, только если он еще не был добавлен на страницу.
После отображения понятного для пользователя сообщения об ошибке команда `require_once` используется для включения футера страницы.
Наконец, команда `exit` останавливает дальнейшее выполнение кода.

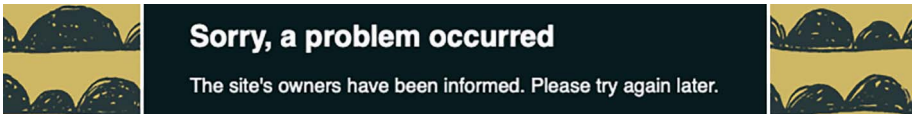
PHP c10/live/error-handling-function.php

```php
<?php
set_error_handler('handle_error');

function handle_error($level, $message, $file = '', $line = 0)
{
    $message = $level . ' ' . $message . ' in ' . $file . ' on line ' . $line;
    error_log($message);
    http_response_code(500);

    require_once 'includes/header.php';
    echo "<h1>Sorry, a problem occurred</h1>
        The site's owners have been informed. Please try again later.";
    require_once 'includes/footer.php';
    exit;
}
$username = $_GET['username'];
?>
<?php include 'includes/header.php'; ?>
<h1>Welcome, <?= $username ?></h1>
<?php include 'includes/header.php'; ?>
```

РЕЗУЛЬТАТ



1. Функция PHP `set_error_handler()` указывает интерпретатору PHP вызвать функцию `handle_error()`, если он обнаруживает нефатальную ошибку. При этом имя функции указывается без круглых скобок.
2. Определяется функция `handle_error()`. Она использует четыре параметра для получения данных, которые передает интерпретатор PHP: уровень ошибки, сообщение об ошибке, имя файла, в котором произошла ошибка, и номер строки.
3. Сообщение об ошибке создается с использованием информации, содержащейся в четырех параметрах, указанных в шаге 2.
4. Функция PHP `error_log()` используется для записи сообщения об ошибке в файл логов ошибок PHP.
5. Функция PHP `http_response_code()` устанавливает код ответа HTTP равным 500, который сообщит браузеру, что на сервере возникла ошибка.
6. Подключается файл с заголовком сайта, если он не был включен ранее, чтобы обеспечить отображение стандартного дизайна и подключение стилей CSS.
7. Понятное для пользователя сообщение об ошибке выводится на страницу с помощью команды `echo`.
8. Подключение файла с футером сайта.
9. Команда `exit` останавливает дальнейшее выполнение кода.
10. Когда код пытается получить доступ к отсутствующему в суперглобальном массиве `$_GET` ключу, возникает нефатальная ошибка.

ИСКЛЮЧЕНИЯ

Когда код сам по себе рабочий, но во время его выполнения возникает исключительная ситуация, то PHP создает **объект исключения**. Это дает программе возможность обработать ошибку и восстановить нормальную работу[44].

Когда возникает исключительная ситуация, интерпретатор PHP останавливает выполнение скрипта и ищет код, написанный для ее обработки. Это дает программе возможность справиться с проблемой.

- Если интерпретатор находит код для обработки исключительной ситуации, он запускает этот код, после которого выполнение скрипта может быть продолжено.
- Если код для обработки ситуации отсутствует, интерпретатор выдает фатальную ошибку. Она начинается со слов `Uncaught exception` (непойманное исключение), после которых идет сообщение об ошибке. Как и любая другая фатальная ошибка, непойманное исключение останавливает работу скрипта.

Программисты говорят, что код **выбрасывает** исключение, а код для обработки **ловит** его. Свойства объекта исключения будут содержать имя файла и строку, в которой возникла проблема, точно так же, как это делает сообщение об ошибке.
Если проблема возникает в функции или методе, объект исключения также содержит информацию о том, какие строки кода вызвали эту функцию или метод. Она называется **трассировкой стека**.
Трассировка стека может быть очень полезна для поиска источника проблемы, поскольку одна и та же функция или метод могут вызываться во множестве разных мест. И когда возникает ошибка, важно знать, где именно была вызвана эта функция или метод.

Объект исключения может быть создан двумя способами.

- Начиная с PHP 7 большинство фатальных ошибок приводят к тому, что интерпретатор PHP создает **объект исключения ошибки** (**error exception object**), используя встроенный класс `Error`. Это позволяет программам восстанавливать работу после ошибки или показывать понятное для пользователя сообщение (вместо того чтобы страница внезапно обрывалась).
- Программисты также могут и сами выбрасывать исключения в коде, используя либо готовые классы исключений, либо создавая собственные — **пользовательские исключения**, основанные на одном из встроенных классов исключений.

Исключения следует использовать, когда программист уверен, что код должен работать, но **исключительная ситуация** помешала ему это сделать[45].

Исключительная ситуация — это когда вы предвидите вероятность возникновения проблемы, но не можете использовать код, чтобы избежать ее появления. Два примера.

- Работоспособность сайта на основе базы данных зависит от ее доступности. Когда сайт не может подключиться к БД, это **является** исключительной ситуацией.
- Пользователи часто вводят недопустимые значения в HTML-форму. Это **не является** исключительной ситуацией и должно обрабатываться с помощью кода проверки.

Код, используемый для обработки исключения, может либо позволить скрипту восстановиться после ошибки и продолжить работу, либо показать пользователю полезное сообщение. Когда программист может предвидеть ситуацию, которая помешает коду выполнять свою работу (но он не может предотвратить возникновение такой ситуации с помощью кода проверки), он может создать собственный пользовательский объект исключения для такой ситуации. Это позволит программе либо восстановиться и продолжить работу, либо

вывести конкретное описание проблемы (вместе с трассировкой стека).

Чтобы создать пользовательский объект исключения, вы должны создать пользовательский класс исключения. Это можно сделать в одной строке кода, потому что все пользовательские исключения **расширяют** встроенный класс, называемый `Exception`.

Когда один класс расширяет другой, он **наследует** свойства и методы расширяемого класса. По этой причине пользовательские классы исключений содержат все свойства и методы, определенные во встроенном классе `Exception`.

Как класс `Exception`, так и встроенный класс `Error` реализуют **интерфейс**, называемый `Throwable`.

Интерфейс описывает имена свойств и/или методов, реализуемых объектом, и данные, которые они должны возвращать. В таблице ниже показаны методы в интерфейсе `Throwable`. Все объекты исключений будут содержать в себе эти методы.

МЕТОД	ВОЗВРАЩАЕТ
`getMessage()`	Текст сообщения об исключении. Для встроенных исключений это сообщение об ошибке, генерируемое интерпретатором PHP. Для пользовательских исключений это сообщение, созданное программистом
`getCode()`	Код исключения, используемый для определения типа исключения. Для встроенных исключений этот код будет сгенерирован интерпретатором PHP. Для пользовательских исключений этот код будет определен программистом
`getFile()`	Имя файла, в котором было создано исключение
`getLine()`	Номер строки, на которой было создано исключение
`getTraceAsString()`	Трассировка стека в виде строки
`getTrace()`	Трассировка стека в виде массива

Чтобы создать пользовательский класс исключений, напишите:

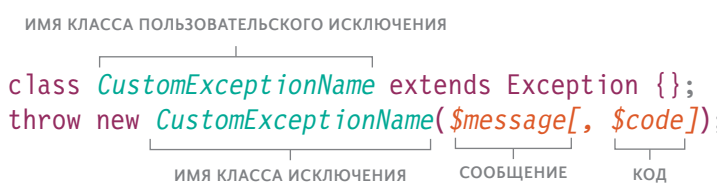
- ключевое слово `class`;
- имя класса. Это имя должно описывать суть выбрасываемого исключения;
- ключевое слово `extends`, чтобы показать, что класс является расширением существующего класса;
- имя класса, которому он будет наследовать (в данном случае встроенный класс `Exception`). Это означает, что он унаследует свойства и методы расширяемого класса;
- пару фигурных скобок.

Чтобы создать (выбросить) исключение с помощью написанного вами пользовательского класса, используйте:

- ключевое слово `throw`. Эта команда порождает исключение, которое прерывает текущую работу кода и указывает интерпретатору PHP искать код для его перехвата;
- ключевое слово `new` для создания нового объекта;
- имя класса пользовательского исключения.

Затем в скобках добавьте:

- сообщение об ошибке, описывающее проблему;
- дополнительный код для определения проблемы.



ОБРАБОТКА ИСКЛЮЧЕНИЙ С ПОМОЩЬЮ БЛОКА try catch

Если вы знаете, что какой-либо код может привести к исключению, и вы можете устранить проблему, то эта ситуация может быть обработана с помощью инструкции try… catch.

В инструкции try…catch за ключевым словом try следует блок кода. В нем содержатся инструкции, которые интерпретатор PHP должен попробовать (try) выполнить, но они могут вызвать исключение. Если в блоке try возникает исключение, интерпретатор PHP:

- останавливает выполнение кода;
- выполняет поиск ближайшего блока catch, который подходит для выброшенного исключения;
- подходящим считается блок catch, в котором указано либо само выброшенное исключение, либо любой из его предков;
- если такой блок найден, то запускаются находящиеся в нем инструкции.

В круглых скобках после ключевого слова catch укажите:

- имя класса исключения (или реализуемый им интерфейс), которое надо поймать в этом блоке (ранее было показано, как указать более одного блока catch, каждый из которых способен обрабатывать исключения, созданные с использованием разных классов);
- имя переменной, которая будет содержать объект исключения в блоке catch (ее часто называют $e).

Если исключение не генерируется в блоке try, интерпретатор PHP пропускает блок catch.

За блоком catch также может следовать необязательный блок finally. Инструкции блока finally будут выполняться независимо от того, было вызвано исключение или нет.

После того как исключение обработано, интерпретатор PHP продолжает выполнение со строки кода, идущей после блока catch. Интерпретатор делает это, не добавляя подробностей о том, что вызвало исключение, в файл журнала ошибок PHP. Это означает, что программисты не будут знать, как часто возникала исключительная ситуация.

Если хотите записать сведения об обработанном исключении в журнал ошибок, используйте встроенную в PHP функцию error_log( ). Ей нужен только один аргумент: выброшенный объект исключения. Интерпретатор PHP возьмет данные о проблеме, хранящиеся в объекте исключения, преобразует их в строку, а затем добавит в файл логов ошибок.

```
try {
    //Попытка сделать что-то, что может вызвать исключение
} catch (ExceptionClassName $e) {
    //Выполнить код, чтобы обработать это исключение, если оно возникло
} finally () {
    //Выполнить код независимо от того, произошло исключение или нет
}
```

ФУНКЦИЯ ОБРАБОТКИ ИСКЛЮЧЕНИЙ ПО УМОЛЧАНИЮ

Интерпретатор PHP может запустить пользовательскую функцию, чтобы обработать исключение, не пойманное блоком `catch` (если оно было выброшено вне блока `try` или не относится к классу, определенному в блоке `catch`).

Можно задать имя пользовательской функции, называемой **обработчиком исключений по умолчанию**, которую интерпретатор PHP должен вызывать, если исключение не обрабатывается блоком `catch`.

Встроенная в PHP функция `set_exception_handler()` задает имя вызываемой пользовательской функции. Ее единственный параметр — имя функции. За именем не должны следовать круглые скобки.

```
set_exception_handler('name')
```

ИМЯ ФУНКЦИИ

Приведенная ниже простая функция обработки исключений:

- добавляет сведения о проблеме в файл логов ошибок;
- задает правильный код ответа HTTP (500);
- отображает сообщение для пользователя;
- прекращает выполнение любого дальнейшего кода на странице.

Посмотрите пример использования функции обработки исключений по умолчанию.

Более сложная функция обработки исключений может проверять класс, использованный при создании исключения, и реагировать на каждое исключение по-разному.

```
function handle_exception($e)
{
    error_log($e);
    http_response_code(500);
    echo '<h1>Sorry, an error occurred, please try again later.</h1>';
    exit;
}
```

ИСПОЛЬЗОВАНИЕ БЛОКА try catch ДЛЯ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

1. В этом примере блок `try` содержит код, потенциально взывающий исключение.
2. Представьте, что подключаемый файл содержит код для отображения рекламы и что он обычно работает, но проблема в этом коде иногда приводит к исключению.
3. Если в блоке `try` генерируется исключение, интерпретатор PHP ищет блок `catch`, способный его обработать. За ключевым словом `catch` следуют круглые скобки. В скобках:

- имя класса указывает, что этот блок `catch` будет выполняться для любого объекта исключения, созданного с использованием класса `Exception` или его потомков;
- `$e` — это имя переменной, в которую записывается объект пойманного исключения, чтобы его данные можно было использовать в блоке `catch`.

4. Будет отображено объявление-заглушка. Такой результат лучше, чем остановка загрузки страницы при возникновении исключения.
5. Функция PHP `error_log()` добавляет ошибку в файл логов ошибок PHP. Ее единственным аргументом будет объект исключения, которое было поймано блоком `catch`.

section_b/c10/live/try-catch.php **PHP**

```
<?php include 'includes/header.php'; ?>

<?php
1 try {
2     include 'includes/ad-server.php';
3 } catch (Exception $e) {
4     echo '<img src="img/advert.png" alt="Newsletter">';
5     error_log($e);
}
?>
<h1>Latest Products</h1>
...
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



Примечание. Для целей этого примера подключаемый файл `ad-server.php` в загружаемом коде вызывает исключение, чтобы вызвать срабатывание блока `catch`.

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ИСКЛЮЧЕНИЙ

Приведенный ниже класс `ImageHandler` обрабатывает изображения с помощью GD. Он выбрасывает пользовательское исключение, если файл изображения не соответствует требованиям (на следующей странице показано, как эти выброшенные исключения могут быть обработаны).

1. Создается пользовательский класс исключения с именем `ImageHandlerException`. Он унаследует свойства и методы встроенного в PHP класса `Exception`.

2. Когда создается объект класса `ImageHandler`, метод `__construct()` проверяет, относится ли изображение к одному из разрешенных MIME-типов. Если нет, генерируется исключение с использованием класса `ImageHandlerException`.

Сообщение об ошибке показывает, что формат изображения не подходит, а код ошибки задается равным 1.

3. При вызове метода `resizeImage()`, если пользователь загрузил изображение, размеры которого стандартны, будет выброшено исключение, поскольку это ухудшит качество изображения.

В сообщении об ошибке говорится, что исходное изображение слишком маленькое, а код ошибки задается равным 2.

PHP section_b/c10/live/classes/ImageHandler.php

```php
<?php
① class ImageHandlerException extends Exception {};

class ImageHandler
{
    public    $fileTypes  = ['image/jpeg', 'image/png',];     // Допустимые MIME-типы
    ...
    public function __construct(string $filepath, string $filename)
    {
        ...
② if (!in_array($this->mediaType, $this->fileTypes)) {  // Если тип недопустим
            throw new ImageHandlerException('File not an accepted image format', 1);
        }
    ...
    }
    public function resizeImage(int $newWidth, int $newHeight, string $uploadPath)
    {
        if (($this->origWidth < $newWidth)
③ or ($this->origHeight < $newHeight)) {            // Если оригинал слишком мал
            throw new ImageHandlerException('Original image too small', 2);
        }
        // Далее идет код для изменения размера и сохранения изображения…
    }
}
```

ОТЛОВ РАЗЛИЧНЫХ ТИПОВ ИСКЛЮЧЕНИЙ

Эта страница позволяет пользователям зарегистрироваться на сайте, указав свой адрес электронной почты и загрузив фотографию профиля.

Обычно этот процесс происходит нормально, но в исключительных ситуациях изображение может не сохраниться. В таких случаях исключение обрабатывается таким образом, чтобы код мог выполняться дальше и сохранять адрес электронной почты посетителя (даже если не получается сохранить изображение). В примере используются два блока `catch` для обработки различных типов исключений, возникающих при загрузке изображения пользователем.

1. Подключение файла `imagehandler.php` с предыдущей страницы. Он содержит определение класса для `ImageHandlerException`, а также класс `ImageHandler`, используемый для изменения размера и сохранения загруженных изображений.
2. Внутри блока `try` содержатся инструкции, которые создают объект `ImageHandler` для обработки изображения, загруженного пользователем, и с его помощью изменяют размер и сохраняют изображение, а затем отображают его. Затем следуют два блока `catch`.
3. Внутри круглых скобок первого блока `catch` указано имя класса `ImageHandlerException`. Это означает, что интерпретатор PHP выполнит этот блок `catch`, только если код в блоке `try` выбросил исключение с использованием класса `ImageHandlerException`. Внутри блока `catch` объект исключения будет сохранен в переменной с именем `$e`.
4. Создаваемые классом `ImageHandler` исключения являются понятными пользователю, поэтому в переменную `$message` записывается сообщение об ошибке, которое возвращает метод `getMessage()`, наследуемый от встроенного класса `Exception`. Сообщение информирует пользователя, что было выбрано изображение с неправильным MIME-типом либо что оно слишком маленькое.

Если первый блок `catch` поймает и обработает исключение, то интерпретатор PHP продолжит выполнение остального кода, пропуская все оставшиеся блоки `catch`.

5. Если исключение было выброшено, но первый блок `catch` его не поймал (то есть оно не относится к классу `ImageHandler`), то будет запущен второй блок `catch`.

Внутри круглых скобок второго блока `catch` прописан интерфейс `Throwable`, указывающий, что блок должен перехватывать любые исключения (поскольку интерфейс `Throwable` является самым базовым типом исключений). Следовательно, интерпретатор будет обрабатывать любое ранее необработанное исключение из блока `try`. Это включает в себя и фатальные ошибки PHP (например, если изображение не удалось сохранить из-за переполнения диска).

Обычно блоки `catch` должны перехватывать определенные классы исключений, а не пытаться перехватывать все исключения, как здесь, но это может быть оправданно, если:

- исключение не даст выполнить критически важный код, например не даст пользователю зарегистрироваться (лучше получить хотя бы адрес электронной почты пользователя, чем вообще ничего);
- это была временная мера, пока не установлена точная причина проблемы.

6. Этот блок `catch` обрабатывает ситуацию по-другому. В переменную `$message` записывается сообщение, информирующее пользователя о том, что изображение не было сохранено. В нем не отображается сообщение об ошибке из объекта исключения, поскольку оно может быть непонятно пользователю или предоставлять хакерам конфиденциальную информацию.
7. Функция PHP `error_log()` вызывается для добавления информации об исключении в лог ошибок PHP.

После того как код в этом блоке `catch` отработает, PHP продолжит выполнение остального кода на странице. Который может, к примеру, сохранить адрес электронной почты посетителя в базе данных. Если же исключение не обработать, то никакие введенные пользователем данные не сохранятся.

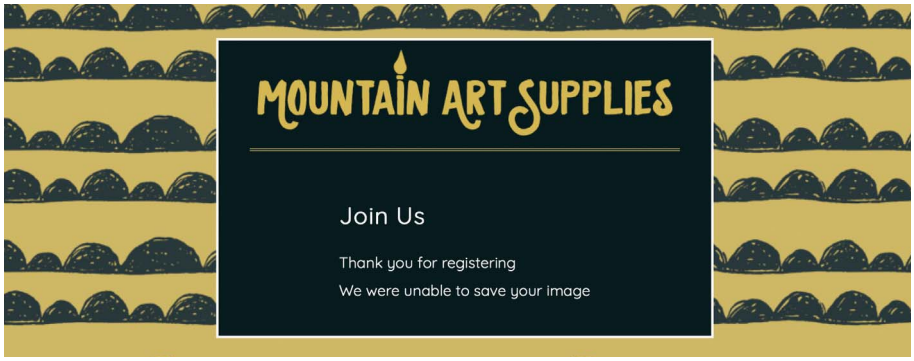
PHP section_b/c10/live/throwing-exceptions.php

```php
<?php
include 'classes/ImageHandler.php';        // Подключение файла с определением класса
$message = '';                             // Инициализация переменных
$thumb   = '';
$email   = '';

if ($_SERVER['REQUEST_METHOD'] == 'POST') { // Если форма отправлена
    $email = $_POST['email'] ?? '';         // Получение email пользователя
    if ($_FILES['image']['error'] == 0) {   // Если ошибок при загрузке нет
        $file = $_FILES['image']['name'];   // Получение имени файла
        $temp = $_FILES['image']['tmp_name']; // Получение пути к временному файлу

        try {                                  // Попытка изменить размер изображения
            $image = new ImageHandler($temp, $file);  // Создание объекта
            $thumb = $image->resizeImage(300, 300, 'uploads/'); // Изменение размера
            $message = '<img src="uploads/' . $thumb . '">'; // HTML для вывода картинки
        } catch (ImageHandlerException $e) {   // Если выброшено ImageHandlerException
            $message = $e->getMessage();              // Получение сообщения об ошибке
        } catch (Throwable $e) {                      // Если другая причина
            $message = 'We were unable to save your image'; // Общее сообщение
            error_log($e);                                  // Логирование ошибки
        }
    }
    // Здесь код может сохранить адрес электронной почты
}
?>
<?php include 'includes/header.php' ?>
<h1>Join Us</h1>
<?= $message ?>
...
```

РЕЗУЛЬТАТ



ОБРАБОТКА ОШИБОК И ИСКЛЮЧЕНИЙ ПО УМОЛЧАНИЮ

В этом примере показано, как можно упорядочить обработку любых ошибок и неотловленных исключений.

Сначала функция PHP `set_exception_handler()` регистрирует пользовательскую функцию `handle_exception()` в качестве обработчика исключений по умолчанию, то есть она будет вызываться, если было выброшено исключение, которое не было поймано ни одним блоком `catch`. Эта функция:

- записывает ошибку в лог;
- задает код состояния ответа HTTP;
- отображает понятное сообщение об ошибке;
- останавливает выполнение кода.

Далее функция PHP `set_error_handler()` регистрирует функцию `error_handler()` в качестве обработчика ошибок, она будет вызываться при возникновении любой нефатальной ошибки.

Функция `error_handler()`преобразует нефатальные ошибки в исключения, что позволяет использовать один и тот же код для обработки как ошибок, так и исключений.

Наконец, функция PHP `register_shutdown_function()` регистрирует пользовательскую функцию `handle_shutdown()`, которая должна вызываться при завершении работы скрипта. Функция `handle_shutdown()` проверяет, возникала ли ошибка при запуске страницы, используя встроенную в PHP функцию `error_get_last()`. Если это так, то эта ошибка преобразуется в исключение и для ее обработки вызывается функция — обработчик исключений.

НЕОТЛОВЛЕННЫЕ ИСКЛЮЧЕНИЯ

1. Функция PHP `set_exception_handler()` регистрирует пользовательскую функцию, которая вызывается, если исключение не было поймано ни одним блоком `catch`.
2. Определение функции `handle_exception()`. Ее единственным параметром является объект исключения.
3. Исключение записывается в лог ошибок PHP.
4. Скрипт возвращает код ответа HTTP, равный 500.9
5. Подключается заголовок сайта, если этого еще не произошло.
6. Отображается понятное пользователю сообщение об ошибке.
7. Подключается футер сайта, если это не произошло ранее.
8. После этого интерпретатор прекращает дальнейшее выполнение кода на странице.

НЕФАТАЛЬНЫЕ ОШИБКИ

9. Функция `set_error_handler()`регистрирует пользовательскую функцию, которая вызывается, если возникает нефатальная ошибка.
10. В эту функцию передается информация об ошибке (текст ошибки, ее тип, а также файл и строка, в которых она произошла).
11. Данные об ошибке используются для выброса нового исключения. Это позволяет сайту обрабатывать все нефатальные ошибки таким же образом, как и фатальные ошибки и исключения, используя одну и ту же функцию `handle_exception()`.

Объект исключения генерируется с использованием встроенного в PHP класса `ErrorException`, который обычно используется для таких целей. Второй параметр — это необязательный код ошибки (целое число) для предоставления дополнительной информации об исключении (здесь он равен 0).

PHP c10/live/includes/functions.php

```php
<?php ...
set_exception_handler('handle_exception');     // Установка обработчика исключений
function handle_exception($e)
{
    error_log($e);                             // Логирование ошибки
    http_response_code(500);                   // Задать код ответа
    require_once 'header.php';                 // Подключить заголовок
    echo "<h1>Sorry, a problem occurred</h1>
          <p>The site's owners have been informed. Please try again later.</p>";
    require_once 'footer.php';                 // Добавить футер
    exit;                                      // Остановить выполнение кода
}

set_error_handler('handle_error');             // Задать обработчик ошибок
function handle_error($type, $message, $file = '', $line = 0)
{
    throw new ErrorException($message, 0, $type, $file, $line); // Выбросить ErrorException
}

register_shutdown_function('handle_shutdown');  // Задать обработчик завершения работы
function handle_shutdown()
{
    $error = error_get_last();                 // Была ли ошибка в скрипте?
    if ($error) {                              // Если да — создать объект исключения
        $e = new ErrorException($error['message'], 0, $error['type'],
                                $error['file'], $error['line']);
        handle_exception($e);                  // Вызов обработчика исключений
    }
}
```

ФАТАЛЬНЫЕ ОШИБКИ

12. Встроенная в PHP функция `register_shutdown_function()` указывает интерпретатору PHP вызывать пользовательскую функцию `handle_shutdown()`при завершении работы скрипта. Это требуется потому, что некоторые фатальные ошибки не преобразуются в исключения и не попадают в обработчик ошибок по умолчанию.

13. Определение функции `handle_shutdown()`.

14. Встроенная в PHP функция `error_get_last()` проверяет, возникали ли ошибки во время выполнения кода. Если да, то сведения о последней вызванной ошибке будут возвращены в массиве; если нет, то будет возвращено значение `null`. Возвращаемое значение сохраняется в `$error`.

15. Конструкция `if` проверяет, содержит ли переменная `$error` не пустое значение, указывающее на то, что произошла ошибка.

16. Если да, ошибка преобразуется в объект исключения.

Примечание. Ключевое слово `throw` здесь не используется, поскольку обработчик исключений по умолчанию не будет улавливать исключения, генерируемые в функции завершения. Вместо этого объект создается так же, как и любой другой объект.

17. Вызывается функция `handle_exception()`. Объект исключения, созданный в шаге 16, передается в нее в качестве аргумента.

Упражнение. В файле `example.php` загружаемого кода есть закомментированные строки для проверки обработчиков ошибок и исключений. Раскомментируйте их по одной.

КАК ОТОБРАЖАЮТСЯ ОШИБКИ ВЕБ-СЕРВЕРА

Если веб-сервер не может найти запрошенный файл или ошибка на сервере не позволяет ему обработать запрос браузера, веб-сервер возвращает в браузер код ошибки и веб-страницу с ее описанием.

Кроме того, сервер отправляет **код состояния ответа**, указывающий, был запрос успешным или нет.

- Когда сервер успешно обрабатывает запрос, он возвращает код 200 и результат запроса.
- Если файл не найден, он возвращает код 404, при этом сервер может также вернуть страницу с описанием ошибки.
- Если ошибка на сервере помешала отображению страницы, он возвращает код 500, при этом сервер может также вернуть страницу с описанием ошибки. В ней говорится, что произошла внутренняя ошибка сервера.

Страницы ошибок, которые веб-сервер создает при невозможности ответить на запрос, не очень понятны для пользователя. Но вы можете создать собственные **пользовательские страницы ошибок** для отправки веб-сервером вместо встроенных.

Пользовательские страницы ошибок позволяют предоставить посетителям более четкое описание проблемы. Кроме того, они могут быть оформлены так же, как и остальной сайт.

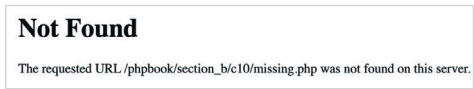
Чтобы настроить пользовательскую страницу с ошибкой, добавьте директиву `ErrorDocument` в файл `.htaccess` и укажите:

- код состояния, для которого этот файл следует использовать;
- путь к файлу, который должен быть отображен для этого кода.

```
ErrorDocument code replacement-page.php
```

ДИРЕКТИВА — КОД СОСТОЯНИЯ — СТРАНИЦА ДЛЯ ОТОБРАЖЕНИЯ ВМЕСТО СТАНДАРТНОЙ

Страница ошибки по умолчанию, отправляемая Apache, когда файл не может быть найден:



Страница ошибок по умолчанию, отправляемая Apache, когда внутренняя ошибка препятствует отображению страницы:



PHP section_b/c10/live/.htaccess

```
ErrorDocument 404 /code/section_b/c10/live/page-not-found.php
ErrorDocument 500 /code/section_b/c10/live/error.php
```

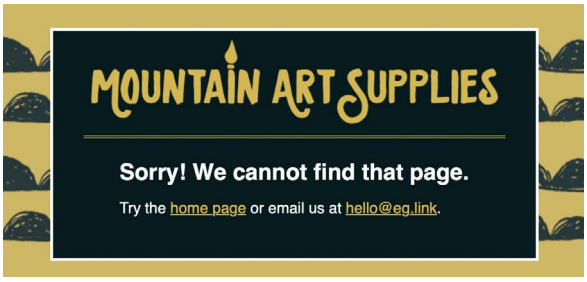
PHP section_b/c10/live/page-not-found.php

```
<?php require_once 'includes/header.php'; ?>
<h1>Sorry! We cannot find that page.</h1>
<p>Try the <a href="index.php">home page</a> or email us at
  <a href="mailto:hello@eg.link">hello@eg.link</a>.</p>
<?php require_once 'includes/footer.php'; ?>
```

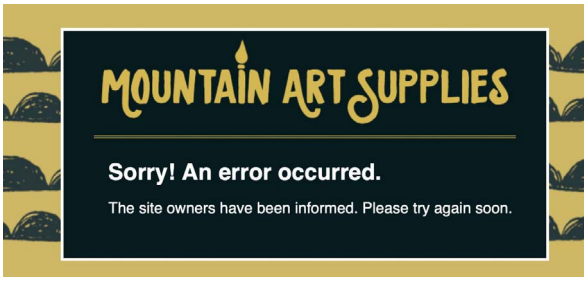
PHP section_b/c10/live/error.php

```
<?php include 'includes/header.php'; ?>
<h1>Sorry! An error occurred.</h1>
<p>The site owners have been informed. Please try again soon.</p>
<?php include 'includes/footer.php'; ?>
```

РЕЗУЛЬТАТ



РЕЗУЛЬТАТ



1. Файл `.htaccess` использует директиву `ErrorDocument`, чтобы задать пути к пользовательским страницам ошибок, отображаемым, если:
   - невозможно найти файл;
   - произошла ошибка сервера.

2. Файл `page-not-found.php` удобным для пользователя способом поясняет, что не удалось найти запрошенный файл.
3. Файл `error.php` сообщает посетителям, что произошла ошибка.

Упражнение. Запросить несуществующую страницу в действующей папке для этой главы, например `missing.php`.

Примечание. Страницы с ошибками не должны содержать код, подключающийся к базе данных, потому что, если произошла ошибка с базой данных, страница с ошибкой не будет отображаться.

ЗАКЛЮЧЕНИЕ
ОБРАБОТКА ОШИБОК

- Сообщения об ошибках помогают определить, в чем заключается проблема.
- При разработке сайта выводите все ошибки на экран.
- Когда сайт запускается в рабочем режиме, отключите отображение ошибок на экране. Вместо этого записывайте их в лог-файл.
- Можно написать функцию, которая запустится при возникновении нефатальной ошибки для ее обработки.
- Исключения — это объекты, создаваемые в исключительных обстоятельствах, препятствующих нормальной работе страницы.
- Когда объект исключения создается (или выбрасывается), его можно перехватить и выполнить код, написанный специально для такого случая.
- Перехватить исключение можно с помощью конструкции `try…catch` или функции обработки исключений.
- Ошибки могут быть преобразованы в исключения, чтобы можно было обрабатывать все возникающие проблемы единообразно.

В

САЙТЫ НА ОСНОВЕ БАЗ ДАННЫХ

Веб-сайт на основе базы данных (БД) использует ее для хранения большей части отображаемого контента, а также другой информации, например учетных записей пользователей.

Поскольку PHP может читать и изменять информацию в БД, то веб-сайт под управлением базы данных позволяет:

- создавать или обновлять содержимое сайта с помощью форм на веб-странице пользователям без технических навыков (им не нужно знать, как писать код или использовать FTP для обновления файлов на сервере);
- адаптировать страницы сайта для каждого пользователя, используя данные из БД.

В этой книге для обеспечения работы базы данных используется программа под названием **MySQL**. БД хранит данные в виде набора таблиц (каждая таблица похожа на электронную таблицу). Поскольку данные в одной таблице часто связаны с данными в другой таблице, она называется **реляционной базой данных**. Всякий раз, когда мы говорим о MySQL, эта информация также относится и к MariaDB.

PHP поставляется с набором встроенных классов, называемых **PHP Data Objects** (**PDO**), которые можно использовать для доступа к БД.

Следовательно, чтобы создавать веб-сайты на основе баз данных, вам нужно будет изучить:

- как использовать язык, называемый **Structured Query Language,** или **SQL** (произносится как *sequel* или *ess-queue-elle*), для запроса данных из БД и их изменения;
- как использовать PDO для выполнения команд SQL, запрашивающих данные из БД, и делать эти данные доступными для вашего PHP-кода;
- как использовать PDO для выполнения SQL-запросов, которые обновляют данные, хранящиеся в БД.

MySQL не предоставляет графический пользовательский интерфейс[46], но существует бесплатный инструмент под названием **phpMyAdmin**. Он дает возможность управлять базой данных и просматривать ее содержимое. Вы узнаете, как его использовать, во введении к этому разделу.

Прежде чем читать главы этого раздела, вам необходимо понять, как базы данных хранят информацию, которая будет использоваться на веб-сайте, и как управлять базой данных с помощью phpMyAdmin.

УЧЕБНЫЙ ВЕБ-САЙТ

На протяжении второй половины этой книги мы будем разрабатывать учебное веб-приложение. На его примере продемонстрируем, как создавать веб-сайты на основе баз данных, а также другие изучаемые концепции. Мы начнем с описания этого веб-сайта, что поможет вам понять структуру его базы данных, с которой вы будете работать.

КАК БАЗЫ ДАННЫХ ХРАНЯТ ИНФОРМАЦИЮ

Далее вы узнаете, как БД хранят данные, которые организованы в виде набора таблиц. Вы также познакомитесь с типами данных, используемыми в MySQL (они отличаются от типов данных в PHP).

КАК ИСПОЛЬЗОВАТЬ PHPMYADMIN

phpMyAdmin — это инструмент, работающий на веб-сервере. Он похож на веб-сайт, предоставляющий возможность управлять базой данных MySQL. Вы узнаете, как он позволяет:

- создавать новые БД;
- просматривать данные в БД;
- создавать резервные копии БД.

КАК СОЗДАТЬ БАЗУ ДАННЫХ

Далее вы узнаете, как настроить базу данных, которая будет использоваться для учебного приложения, разрабатываемого на протяжении всей остальной части книги. В базе данных хранится содержимое сайта и данные о его пользователях.

Сначала вы узнаете, как создать пустую базу данных. Затем запустите SQL-код (предоставленный в загружаемом коде), который:

- создаст таблицы, используемые в учебной базе данных;
- добавит данные в каждую из этих таблиц.

Примечание. Все остальные примеры используют эту базу данных, поэтому для продолжения работы необходимо создать ее и заполнить данными.

СОЗДАНИЕ УЧЕТНОЙ ЗАПИСИ ПОЛЬЗОВАТЕЛЯ БАЗЫ ДАННЫХ

Наконец, вы узнаете, как создать учетную запись пользователя БД. PHP-коду требуется учетная запись пользователя для подключения к базе данных (точно так же, как почтовой программе требуется учетная запись электронной почты для отправки и получения электронных писем).

ЗНАКОМСТВО С УЧЕБНЫМ ВЕБ-САЙТОМ

Учебный веб-сайт, создаваемый далее в этой книге, представляет собой простую **систему управления контентом** (CMS, Content management system). Это инструмент, позволяющий пользователям обновлять контент сайта без написания какого-либо кода.

Система управления контентом, разрабатываемая далее в этой книге, представляет собой витрину работ коллектива дизайнеров, но ее можно использовать в качестве CMS и для многих других типов сайтов.

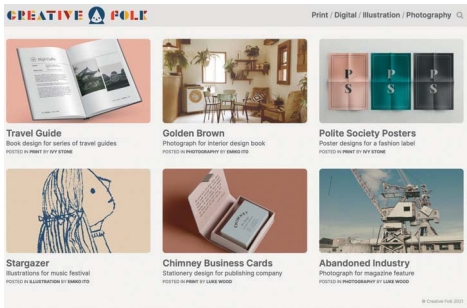
В верхней части каждой страницы, под названием сайта (Creative Folk), на панели навигации отображаются разделы или **категории** (categories) сайта.

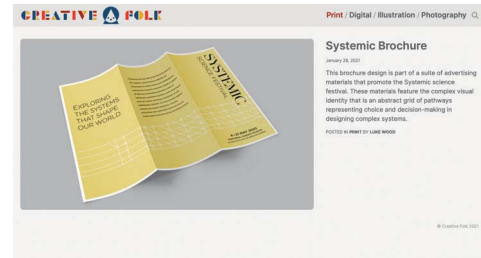
Главная страница (показанная ниже) отображается скриптом index.php. Под списком категорий в нем отображается информация о шести самых последних публикациях. Когда загружается новая публикация, она появляется на домашней странице, а самая старая публикация пропадает с нее.

Каждая статья (article), или скорее публикация, представляет одно творческое произведение. Все работы отображаются с использованием одного и того же файла `article.php`. Его задача состоит в том, чтобы получать из базы данных по одной публикации за раз и отобразить ее на странице.

На странице публикации отображается само произведение, его название, дата добавления на сайт, описание, категория, к которой она относится, и имя автора.

У каждой публикации также есть параметр под названием «опубликовано» (published) — статья не отображается для всеобщего обозрения, пока не будет установлен этот признак.



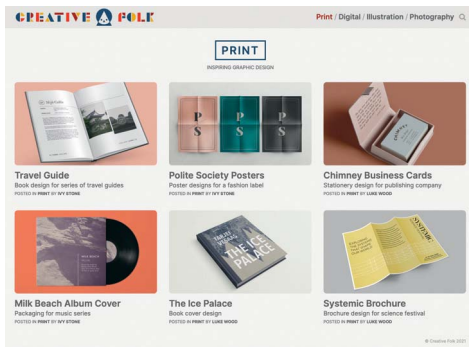


В последующих главах сайт развивается, позволяя посетителям добавлять собственные публикации, оставлять комментарии и отмечать понравившиеся работы.

Публикации сгруппированы по категориям (они подобны разделам сайта). Все категории отображаются с использованием одной и той же страницы `category.php`. На ней отображаются название и описание категории, за которыми следуют карточки с краткими описаниями работ, относящихся к этой категории.

При этом для каждой публикации показывается изображение, название, краткое описание, категория, к которой относится работа, и ее автор (в том же формате, в каком отображаются последние шесть работ на домашней странице).

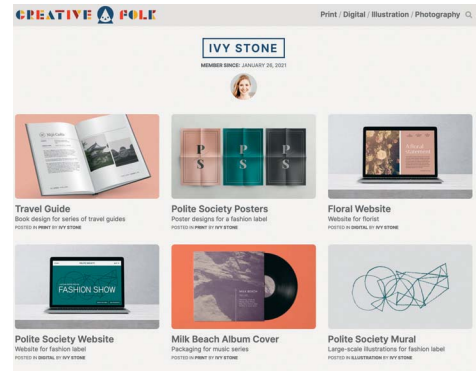
Для каждой категории также есть возможность указать, должно ли ее название отображаться на главной панели навигации или быть скрытым.



Публикации размещаются **пользователями** сайта, и таким образом пользователь, разместивший публикацию, является ее **автором**. Персональные страницы всех пользователей выводятся с использованием одного и того же файла `member.php`.

На них отображается имя автора, дата его регистрации на сайте и фотография, за которой следует список его публикаций. Информация о каждой работе выводится так же, как на странице категории и домашней странице (изображение, заголовок, краткое описание, название категории и автор).

К концу этой книги у посетителей сайта появится возможность добавлять свои работы, отмечать понравившиеся публикации и комментировать их.



КАК РЕЛЯЦИОННАЯ БД ХРАНИТ ИНФОРМАЦИЮ

Реляционные базы данных хранят данные в таблицах. Одна база данных может состоять из множества таблиц. Ниже показаны таблицы базы данных для учебного веб-сайта, разрабатываемого далее в этой книге.

Система управления реляционными базами данных (**СУБД**), или **Relational database management system** (**RDBMS**), такая как MySQL или MariaDB, представляет собой программу, которая может обслуживать сразу несколько баз данных (точно так же, как на одном веб-сервере может размещаться сразу несколько веб-сайтов).

Сервер баз данных может быть размещен:

- либо на том же компьютере, что и веб-сервер (MAMP и XAMPP устанавливают MySQL или MariaDB на ваш компьютер);
- на отдельном компьютере, к которому у веб-сервера есть доступ (как, например, почтовая программа подключается к почтовому серверу).

ТАБЛИЦЫ

В базе данных учебного веб-сайта используется четыре **таблицы**. Каждая таблица представляет одну из сущностей, с которыми работает приложение:

- `article` представляет каждую отдельную публикацию и данные о ней (например, название или дату создания);
- `category` представляет разделы сайта, объединяющие связанные публикации в темы;
- `image` содержит данные о размещаемых в публикациях изображениях;
- `member` содержит информацию о каждом пользователе сайта.

article

id	title	summary	content	created	category_id	member_id	image_id	published
1	Systemic	Brochure...	This...	2021-01-26	1	2	1	1
2	Forecast	Handbag...	This...	2021-01-28	3	2	2	1
3	Swimming	Photos...	This...	2021-02-02	4	1	3	1

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual...	1

Реляционные базы данных получили свое название потому, что данные в разных таблицах могут быть связаны между собой.

СТРОКИ

Каждая строка в таблице (также называемая записью или кортежем) содержит один из представляемых таблицей элементов. Каждая строка в таблице `article` представляет одну публикацию, каждая строка в таблице `member` представляет одного пользователя.

СТОЛБЦЫ

Каждый столбец (колонка)[47] в таблице (также называемый атрибутом) хранит одну характеристику представляемых таблицей элементов. Например, в столбцах таблицы `member` хранятся имя, фамилия, адрес электронной почты пользователя, пароль, дата регистрации и имя файла с его фотографией профиля.

ПОЛЯ

Поле — это отдельный фрагмент информации в строке, пересечение строки и колонки.

ПЕРВИЧНЫЙ КЛЮЧ

Если посмотреть на схемы таблиц, то в каждой из них есть столбец под названием id. Он так называется потому, что является **уникальным идентификатором** для своей строки (то есть он позволяет однозначно идентифицировать каждую публикацию, категорию, пользователя или изображение). Чтобы это работало, в столбце id каждой строки таблицы должно быть свое уникальное значение. В наших таблицах это значение создается с помощью специального механизма под названием **автоинкремент**. Благодаря ему каждая новая строка создается со значением в столбце id, увеличенным на единицу, независимо от того, сколько всего строк в таблице или какое значение имеет id в предыдущей строке. Этот столбец также является **первичным ключом (primary key)** таблицы.

image

id	file	alt
1	systemic-brochure.jpg	Brochure for Systemic Science Festival
2	forecast.jpg	Illustration of a handbag
3	swimming-pool.jpg	Photography of swimming pool

СТРОКА

member

id	forename	surname	email	password	joined	picture
1	Ivy	Stone	ivy@eg.link	c63j-82ve-...	2021-01-26 12:04:23	ivy.jpg
2	Luke	Wood	luke@eg.link	saq8-2f2k-...	2021-01-26 12:15:18	*NULL*
3	Emiko	Ito	emi@eg.link	sk3r-vd92-...	2021-02-12 10:53:47	emi.jpg

СТОЛБЕЦ

ТИПЫ ДАННЫХ В БД

Вам необходимо указать тип данных для каждого столбца каждой таблицы и максимальное количество символов, допустимое к хранению в каждом поле этого столбца.

В MySQL больше типов данных, чем в PHP, но в нашей учебной базе данных используются только пять. Они показаны ниже.
Примечание. MySQL не поддерживает логический тип данных.
Логические значения представляются с использованием типа данных `tinyint` со значением 0 для `false` и 1 для `true`.

ТИП ДАННЫХ	ОПИСАНИЕ
`int`	Целое число
`tinyint`	Целое число от 0 до 255 или от –127 до 127 (в учебной БД используется для логических значений)
`varchar`	Буквенно-цифровые символы до 65 535
`text`	Буквенно-цифровые символы до 65 535
`timestamp`	Дата и время[48]

Для некоторых типов данных размер не указывается, как, например, для типов text или timestamp, а для некоторых указывается:

- для типа varchar размер задает максимальное количество символов, которые могут храниться в ячейке;
- для числовых типов размер может задавать формат хранимых значений или игнорироваться.



В приведенных ниже таблицах типы данных указаны под именами столбцов, за ними в круглых скобках указано максимальное количество цифр или символов, которые может содержать столбец.

article

id int(11)	title varchar(254)	summary varchar(1000)	content text	created timestamp	category_id int(11)	member_id int(11)	image_id int(11)	published tinyint(1)
1	Systemic	Brochure...	This...	2021-01-26	1	2	1	1
2	Forecast	Handbag...	This...	2021-01-28	3	2	2	1
3	Swimming	Photos...	This...	2021-02-02	4	1	3	1

category

id int(11)	name varchar(24)	description varchar(254)	navigation tinyint(1)
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual...	1

ИЗБЕГАЙТЕ ДУБЛИРОВАНИЯ ДАННЫХ В БД

Избегайте дублирования одних и тех же данных в БД. Вместо этого связывайте данные в разных таблицах с помощью **первичных** и **внешних ключей**.

Первый столбец в каждой таблице ниже — это **первичный ключ.** Он содержит значение, идентифицирующее каждую строку таблицы.

Если вы посмотрите на таблицу articles, показанную слева внизу, то увидите.

- Столбец `category_id` показывает, к какой категории относится публикацию. Его значение соответствует первичному ключу в таблице категорий.
- Столбец `member_id` показывает, кто разместил публикацию. Его значение соответствует первичному ключу в таблице пользователей.
- Столбец `image_id` показывает, какое изображение должно быть показано вместе с публикацией. Его значение соответствует первичному ключу в таблице изображений.

Столбцы `category_id`, `member_id` и `image_id` таблицы `article` являются **внешними ключами (foreign key)**. Они показывают, что:

- первая публикация относится к категории `Print`;
- первая и вторая публикации размещены автором по имени `Luke Wood`;
- третья публикация использует изображение `swimming-pool.jpg` из таблицы `image`.

Эти значения описывают взаимосвязь между данными в разных таблицах и позволяют избежать повторения одних и тех же названий категорий и имен авторов в нескольких строках таблицы `article`. Предотвращение дублирования делает базу данных меньше, а работу с ней — быстрее, плюс снижает риск ошибок, поскольку при необходимости изменить какие-либо данные это делается только в одном месте.

image

id int(11)	file varchar(254)	alt varchar(1000)
1	systemic-brochure.jpg	Brochure for Systemic Science Festival
2	polite-society-posters.jpg	Posters for Polite Society
3	swimming-pool.jpg	Photography of swimming pool

member

id int(11)	forename varchar(254)	surname varchar(254)	email varchar(254)	password varchar(254)	joined timestamp	picture varchar(254)
1	Ivy	Stone	ivy@eg.link	c63j-82ve-...	2021-01-26 12:04:23	ivy.jpg
2	Luke	Wood	luke@eg.link	saq8-2f2k-...	2021-01-26 12:15:18	*NULL*
3	Emiko	Ito	emi@eg.link	sk3r-vd92-...	2021-02-12 10:53:47	emi.jpg

ИСПОЛЬЗОВАНИЕ PHPMYADMIN ДЛЯ РАБОТЫ С MYSQL

Обычно на веб-сайтах информация в БД обновляется с помощью PHP-скриптов, принимающих информацию из HTML-форм. Однако бывают случаи, когда необходимо обратиться к БД напрямую. Это можно сделать с помощью phpMyAdmin.

ИСПОЛЬЗОВАНИЕ PHPMYADMIN

Существуют задачи, которые иногда требуется выполнять на сайтах, использующих базы данных. Такие задачи обычно называют **администрированием.** Они включают в себя:

- создание новой БД;
- резервное копирование существующей БД;
- добавление новых таблиц и столбцов в БД;
- проверка правильности работы с БД при добавлении новых функций на сайт;
- создание учетных записей пользователей для контроля того, кто из пользователей может получать доступ/редактировать содержимое базы данных.

MySQL не поставляется с собственным визуальным интерфейсом, который вы могли бы использовать для выполнения этих задач администратора. Но существует бесплатный инструмент с открытым исходным кодом под названием **phpMyAdmin**, и он поможет вам их выполнить.

phpMyAdmin написан на PHP и работает на вашем веб-сервере. Этот инструмент похож на веб-сайт, и с его помощью можно управлять базой данных. На следующих страницах показано, как он используется для выполнения некоторых задач администрирования.

КАК НАЙТИ PHPMYADMIN

При установке MAMP или XAMPP на вашем компьютере будет установлен phpMyAdmin.
Для получения доступа к нему введите в своем браузере URL-адрес http://localhost/phpMyAdmin/.

Если вы добавили номер порта к URL-адресам для запуска примеров кода, вам, вероятно, придется добавить тот же номер порта для доступа к phpMyAdmin.

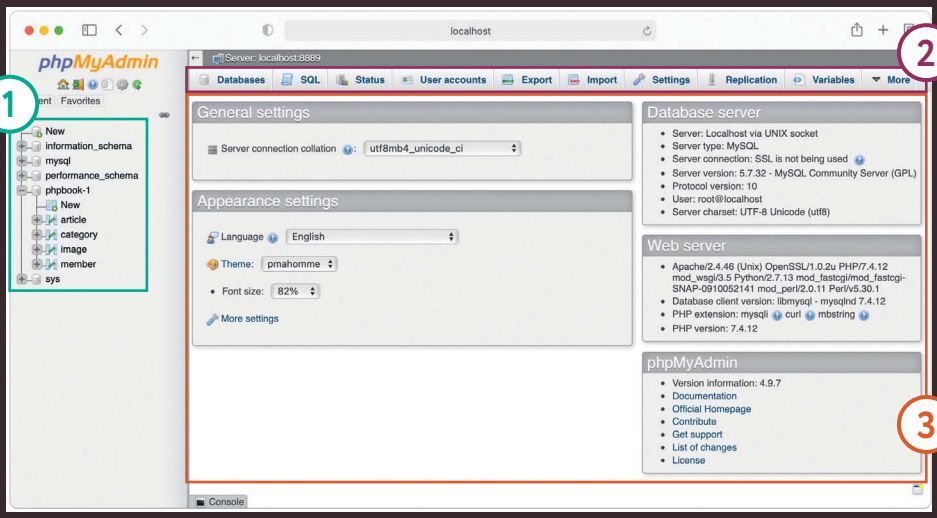
Пример: http://localhost:8888/phpMyAdmin/.

Хостинговые компании, предлагающие БД MySQL, могут использовать собственные инструменты для выполнения задач администрирования. Поэтому вам необходимо узнать, как конкретно ваша хостинговая компания позволяет вам их выполнять. Вместо того чтобы предоставить вам доступ к полной версии phpMyAdmin, они часто используют:

- собственные инструменты для создания баз данных/учетных записей пользователей;
- ограниченную версию phpMyAdmin для резервного копирования, проверки и обновления содержимого базы данных;
- собственные URL-адреса для доступа к phpMyAdmin.

ИСПОЛЬЗОВАНИЕ PHPMYADMIN ДЛЯ АДМИНИСТРИРОВАНИЯ БАЗЫ ДАННЫХ

Экран phpMyAdmin состоит из трех основных областей. Разные версии phpMyAdmin могут выглядеть немного по-разному, но функциональность и расположение элементов на странице должны быть одинаковыми.



1. Базы данных и таблицы
Один сервер MySQL может содержать несколько баз данных (так же, как на веб-сервере может размещаться много сайтов).

Названия баз данных отображаются в меню слева. Когда вы нажимаете на имя базы данных, символы + в списке раскрываются, чтобы показать имена таблиц в этой базе данных.

2. Вкладки
Они представляют собой то, что вы можете делать с помощью интерфейса. Когда вы:

- открываете phpMyAdmin, на вкладках отображаются задачи, связанные со всем сервером MySQL;
- нажимаете на базу данных в меню слева, вкладки изменяются на задачи, которые вы можете выполнять с этой базой данных.

3. Главное окно
Здесь вы выполняете задачи администрирования (например, оно показывает содержимое базы данных и позволяет редактировать данные или добавлять строки).

MySQL часто создает несколько собственных баз данных, например information_schema, mysql, performance_schema и sys. Новичкам не следует их редактировать.

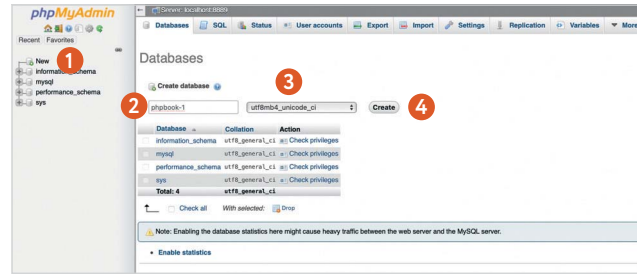
НАСТРОЙКА УЧЕБНОЙ БАЗЫ ДАННЫХ

Чтобы работать с остальными примерами в этой книге, мы начнем с создания новой базы данных и импорта в нее некоторых данных.

СОЗДАЙТЕ ПУСТУЮ БАЗУ ДАННЫХ

Сначала создайте пустую базу данных с помощью phpMyAdmin.

1. Нажмите пункт `New` в верхней части списка баз данных.
2. Введите имя базы данных `phpbook-1`.
3. В раскрывающемся списке выберите пункт **`utf8mb4_unicode_ci`**[49] для указания кодировки.
4. Нажмите кнопку создания **`Create`**.

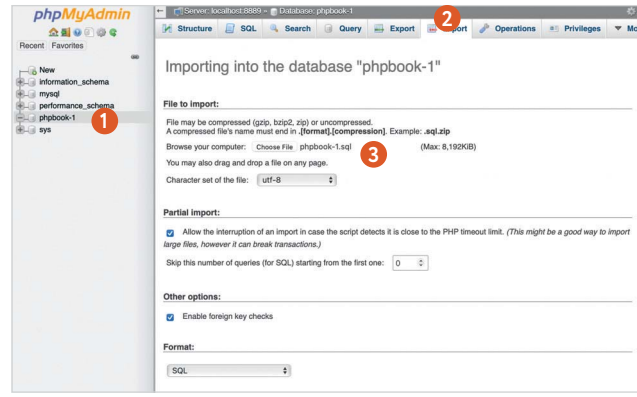


Если вы пользуетесь услугами хостинговой компании, вам может потребоваться использовать их инструменты для создания пустой базы данных. Если это так, пропустите этот шаг.

ДОБАВЛЕНИЕ ДАННЫХ В БД

После того как вы создали БД, вы можете добавить в нее данные.

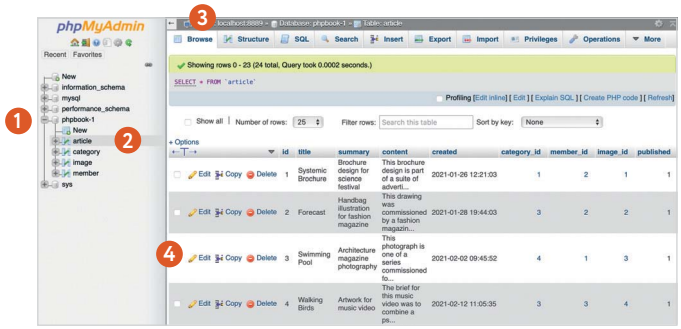
1. Щелкните по имени базы данных из списка баз данных.
2. Перейдите на вкладку импортирования **`Import`**.
3. В разделе `File to import` нажмите кнопку **`Choose file`** и выберите файл phpbook-1.sql из папки `section_c/intro` в загружаемом коде.

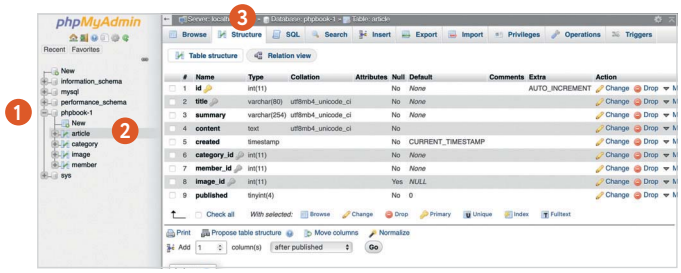


4. Нажмите кнопку **`Go`** или **Import** (внизу страницы, не показано), чтобы импортировать образцы данных в БД.

ОБЗОР УЧЕБНОЙ БАЗЫ ДАННЫХ

После того как вы импортировали данные, посмотрите на созданные таблицы и изучите их содержимое и структуру.





ИЗУЧЕНИЕ СОДЕРЖИМОГО БАЗЫ ДАННЫХ

Вы можете ознакомиться с содержимым базы данных, щелкнув по имени базы данных.

1. Нажмите на имя базы данных на панели списка баз данных.
2. Выберите таблицу `article`.
3. Перейдите на вкладку **`Browse`**.
4. Каждая строка таблицы представляет собой одну публикацию.

СТРУКТУРА ТАБЛИЦЫ

Отображаются имена столбцов, типы данных и размеры полей каждой таблицы.

1. Нажмите на имя базы данных.
2. Выберите таблицу `article`.
3. Перейдите на вкладку **`Structure`**.

Каждый столбец отображается в виде строки в показанной таблице.

Вы можете увидеть имя каждого столбца, за которым следует тип данных с указанием его максимального размера в круглых скобках. Столбец `Collation` содержит кодировку символов. Столбец `Null` указывает, может ли ячейка в этом столбце принимать значение `NULL`. Столбец `Default` указывает значение, которое столбец должен использовать по умолчанию, если актуальное значение не было указано.

Чтобы узнать, как вручную добавлять таблицы и столбцы, см. **http://notes.re/mysql/create-manually.**

СОЗДАНИЕ УЧЕТНЫХ ЗАПИСЕЙ ПОЛЬЗОВАТЕЛЕЙ БАЗЫ ДАННЫХ

Сервер MySQL позволяет вам создавать различные **учетные записи пользователей**[50]. Каждая из них использует имя пользователя и пароль для входа в базу данных. Вы можете контролировать, какие данные может пользователь запрашивать и какие может обновлять каждая учетная запись.

Каждая учетная запись пользователя MySQL позволяет вам указать:

- к каким базам данных этот пользователь может получить доступ;
- таблицы, к которым он может получить доступ;
- другие задачи, которые может выполнять пользователь.

Сразу после установки сервер MySQL содержит только одну учетную запись, известную как **root** — основная учетная запись, с помощью которой можно совершать любые действия, в том числе создавать и удалять учетные записи пользователей и базы данных.
По соображениям безопасности не используйте **учетную запись root** в своем PHP-коде. Вместо этого создайте учетную запись пользователя, которая:

- имеет доступ только к базе данных, используемой для этого конкретного сайта (а не ко всем базам данных на этом сервере);
- может выполнять только те задачи, которые необходимы для работы приложения. Не позволяйте ему выполнять расширенные команды (такие как создание или удаление таблиц), если сайт этого не требует.

На странице справа показано, как учетная запись root (настроенная при установке сервера) позволяет просматривать и создавать учетные записи пользователей в phpMyAdmin.

Если вы пользуетесь услугами веб-хостинг-компании и не видите эти параметры, вам следует обратиться к их файлам справки, поскольку каждый хост работает по-разному и у них могут быть созданные для вас имя пользователя и пароль, а также их собственные инструменты для создания и обновления пользователей.

1. Выберите имя базы данных на левой панели, для которой вы хотите создать пользователя.

2. Перейдите на вкладку **`Privileges`**. При этом отобразится таблица пользователей, имеющих доступ к этой базе данных.

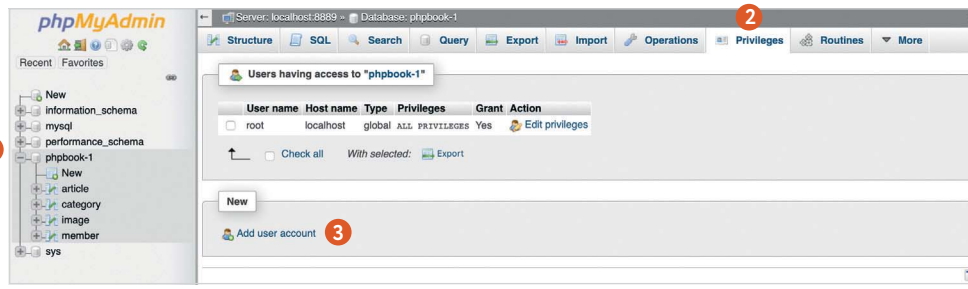
3. Нажмите пункт **`Add user`** или **`Add user account`**.

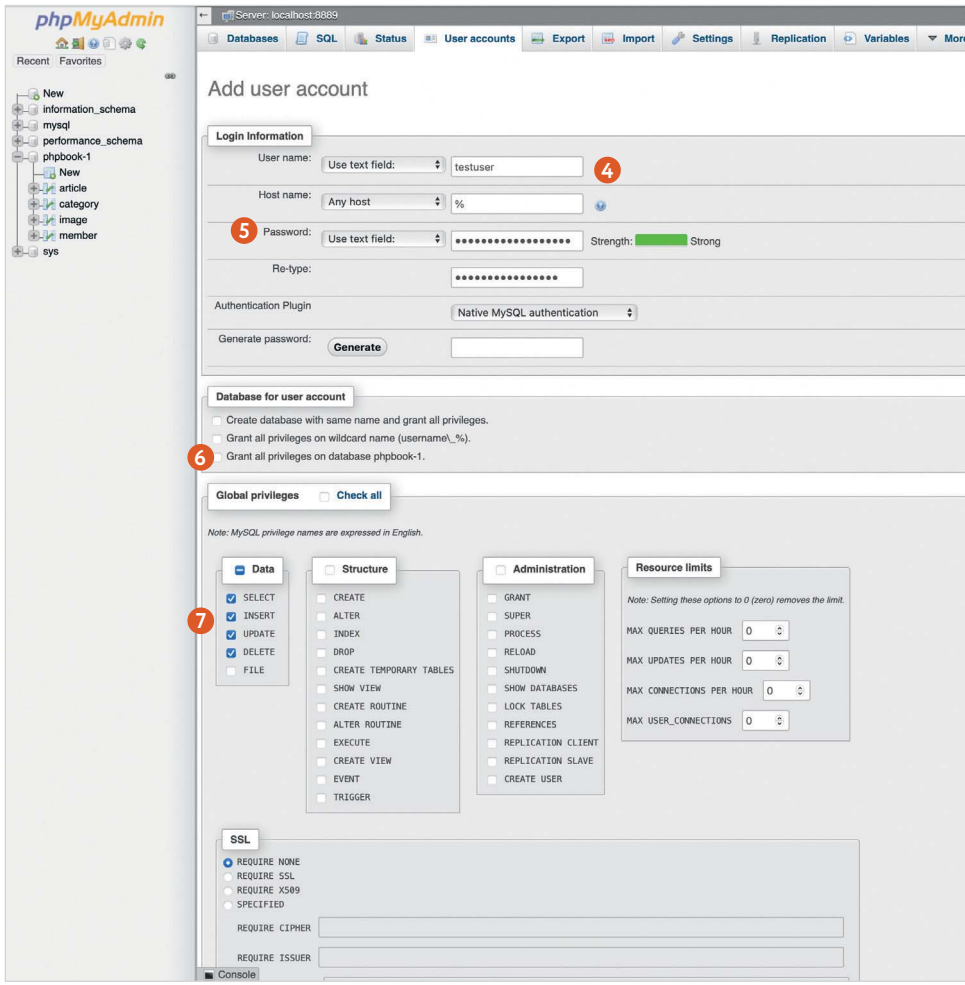
4. Введите имя пользователя.

5. Введите пароль или используйте функцию **Generate**.

6. В разделе **`Database for user account`** проверьте состояние флажка **`Grant all privileges on database phpbook-1`**. Если он установлен, снимите этот флажок.

7. Параметры **`Global privileges`** определяют, что пользователю разрешено делать с базой данных в целом. Для учебного веб-сайта нужны только четыре параметра, которые и надо отметить в столбце Data: Select, Insert, Update и Delete (выборка, вставка, обновление и удаление). Другие функции в учебном приложении не используются, поэтому их включать не нужно.





Под показанными на скриншоте настройками могут быть и другие. Просто оставьте их как есть.

8. Сохраните данные пользователя с помощью кнопки **Go** в нижней части страницы (на скриншотах не показана).

Вы также можете выбрать каждую таблицу на левой панели и установить привилегии пользователя для этой таблицы.

В ЭТОМ РАЗДЕЛЕ
САЙТЫ НА ОСНОВЕ БАЗ ДАННЫХ

ВАЖНОЕ ПРИМЕЧАНИЕ: для остальных глав книги вам необходимо загрузить код с http://addons.eksmo.ru/it/phpbook.zip и запускать его на локальном веб-сервере. Кроме того, очень удобно открывать полные примеры кода в редакторе по мере чтения каждой главы.

11

ЯЗЫК СТРУКТУРИРОВАННЫХ ЗАПРОСОВ (SQL)

SQL — это язык, позволяющий вам указать, какие данные вы хотите извлечь из базы данных и какие данные в ней вы хотите поменять. В этой главе будет показано, как работает язык SQL, с помощью ввода SQL-команд в интерфейс phpMyAdmin.

12

ПОЛУЧЕНИЕ ИНФОРМАЦИИ ИЗ БД

После изучения SQL вы ознакомитесь с тем, как с помощью PDO можно отправлять SQL- запросы в БД из PHP и как получить возвращаемые данные в виде массива или объекта.

13

ОБНОВЛЕНИЕ ИНФОРМАЦИИ В БД

В этой главе вы узнаете, как получить данные от посетителя сайта, проверить их и затем использовать для обновления БД. Вы также узнаете, как устранять проблемы, которые могут возникнуть при выполнении запросов.

11

ЯЗЫК СТРУКТУРИРОВАННЫХ ЗАПРОСОВ (SQL)

Язык структурированных запросов (SQL — Structured Query Language) — это язык, используемый для взаимодействия с базами данных. Он используется для получения, удаления, добавления новых и редактирования существующих данных.

В этой главе вы узнаете, как использовать SQL для выполнения следующих задач:

- **выборка** данных из БД;
- **создание** новых строк в таблице БД;
- **обновление** сохраненных в БД данных;
- **удаление** строк из таблиц БД[51].

Команду для получения или изменения информации, хранящейся в базе данных, называют **командой SQL (SQL statement)**. Команда SQL, только запрашивающая информацию, также может называться **SQL-запросом (query)**, поскольку вы запрашиваете данные у БД[52]. Сначала вы изучите, как запрашивать информацию из БД, а затем — как создавать, обновлять или удалять данные из базы данных.

Чтобы практиковаться в языке SQL, вы будете использовать программу phpMyAdmin. После изучения принципов работы SQL в следующих двух главах будет показано, как выполнять SQL-запросы в PHP-коде, используя PDO (PHP Data Objects).

Некоторые примеры в этой главе обновляют данные в учебной БД, составляющей основу примера веб-сайта в этой книге, поэтому их следует запускать только один раз и в приведенном в главе порядке. Если они выполняются не в строгом порядке, то последующие примеры могут не работать. В этом случае (или при желании запустить их снова) удалите базу данных и установите ее заново, используя инструкции из введения к этому разделу.


phpMyAdmin
Recent Favorites
New
information_schema
mysql
performance_schema
phpbook-1
New
article
category
image
member
sys
Server: localhost:8889 » Database: phpbook-1 » Table: article
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
More
Run SQL query/queries on table phpbook-1.article:
SELECT a.title, a.summary, a.content, a.created, a.category_id, a.member_id,
c.name AS category,
CONCAT(m.forename, ' ', m.surname) AS author,
i.file AS image_file,
i.alt AS image_alt
FROM article AS a
JOIN category AS c ON a.category_id = c.id
JOIN member AS m ON a.member_id = m.id
LEFT JOIN image AS i ON a.image_id = i.id
WHERE a.id
AND a.published = 1;
Columns
id
title
summary
content
created
SELECT *
SELECT
INSERT
UPDATE
DELETE
Clear
Format
Get auto-saved query
Bind parameters
[Delimiter ;]
Show this query here again
Retain query box
Rollback when finished
Enable foreign key checks
Go
Console
MacBook Pro

ПОЛУЧЕНИЕ ИНФОРМАЦИИ ИЗ БД

Чтобы запросить данные из БД, используйте команду SQL SELECT, а затем укажите, какие данные вы хотите получить. Затем база данных создаст **результирующий набор** (**result set**), содержащий запрошенные вами данные.

Команда SELECT указывает, что вы хотите получить данные из БД. За ней следуют имена столбцов, содержащих нужные вам данные. Имя каждого столбца должно быть отделено запятой. Затем следует оператор FROM, после которого пишется имя таблицы, из которой вы хотите получить данные.

Инструкция SQL должна заканчиваться точкой с запятой, хотя некоторые программы позволяют выполнить одиночный запрос без нее, например PHPMyAdmin. Также точка с запятой обычно не пишется в запросах, которые выполняются из кода PHP.

СТОЛБЦЫ ДЛЯ ВЫБОРА → `SELECT column1, column2`
ТАБЛИЦА, В КОТОРОЙ НАХОДЯТСЯ ЭТИ СТОЛБЦЫ → `FROM table;`

Приведенный ниже запрос SQL выбирает сведения из колонок forename и surname таблицы member. Он возвращает данные из каждой строки таблицы.

Запросы SQL напоминают предложения на английском языке. Например, вы можете прочитать следующий запрос как английский текст. В нем говорится, что вы хотите:

- выбрать (SELECT) столбцы forename (имя) и surname (фамилия)
- из (FROM) таблицы member (пользователь).

СТОЛБЕЦ 1 СТОЛБЕЦ 2

```
SELECT forename, surname
  FROM member;
```

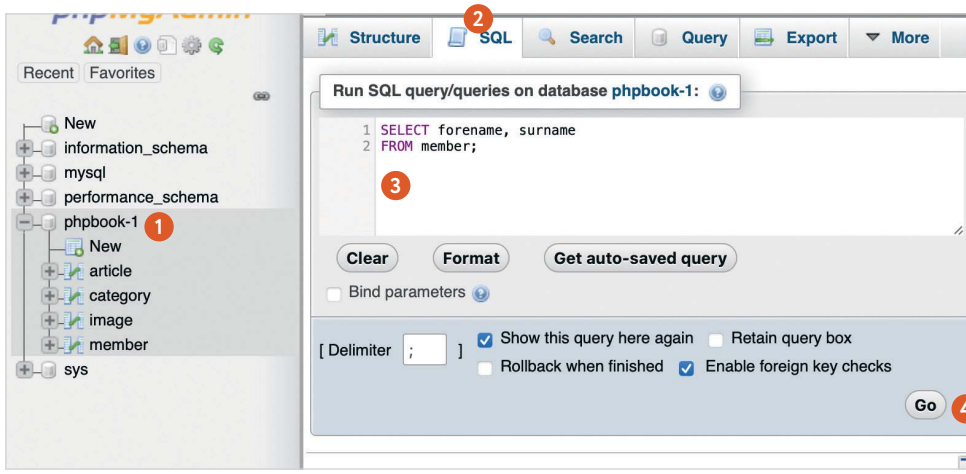
ТАБЛИЦА

Когда SQL-запрос **запускается** или **выполняется**, база данных получает запрошенные данные и помещает их в результирующий набор. Столбцы добавляются в результирующий набор в том же порядке, в котором они указаны в запросе. Чтобы управлять порядком добавления строк в результирующий набор, используйте оператор ORDER BY.

result set	
forename	surname
Ivy	Stone
Luke	Wood
Emiko	Ito

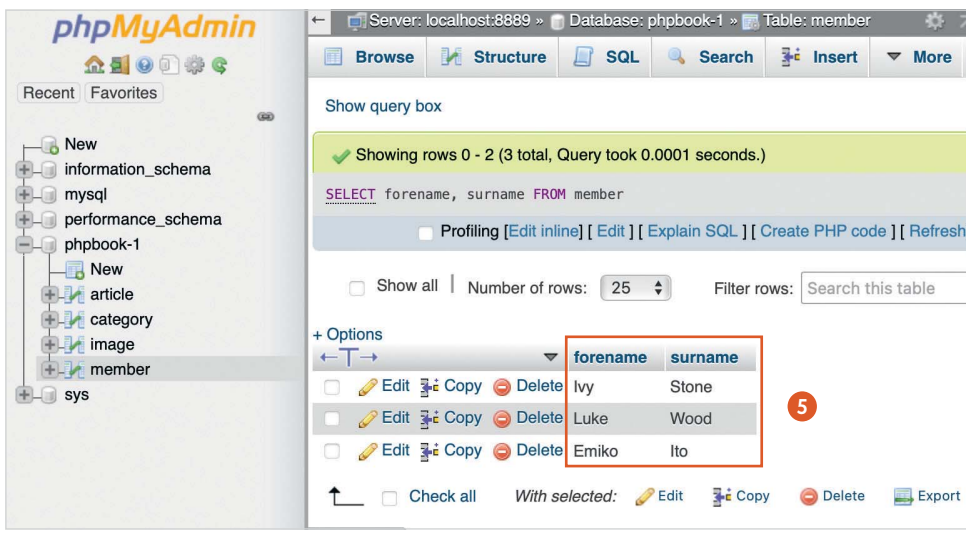
Команды SQL можно писать в верхнем или нижнем регистре. В этой книге они пишутся в верхнем регистре, чтобы отличать их от имен таблиц и столбцов. В именах таблиц и столбцов должен использоваться тот же регистр, что и в базе данных. На странице справа показано, как ввести SQL-запрос в phpMyAdmin и как отображается результирующий набор, генерируемый запросом.

1. Откройте **phpMyAdmin** и выберите базу данных phpbook-1.
2. Выберите вкладку **SQL**.
3. Введите SQL-запрос со страницы слева в область ввода.
4. Нажмите кнопку **Go**.



5. Когда вы нажимаете кнопку **Go**, выполняется SQL-запрос. Затем MySQL возвращает результирующий набор в phpMyAdmin и он отображает результирующий набор в таблице.

В остальной части этой главы вместо показа скриншотов из phpMyAdmin будут показаны SQL-запросы и их результирующие наборы, как на странице слева.



ПОЛУЧЕНИЕ ОПРЕДЕЛЕННЫХ СТРОК ИЗ ТАБЛИЦЫ

Чтобы получить данные из определенных строк таблицы (а не из всех), добавьте оператор `WHERE`, за которым следует **условие поиска**.

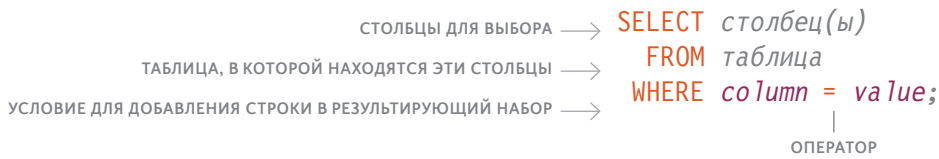
После того как вы указали, какие столбцы и из какой таблицы вы хотите получить, можно добавить условие поиска. Это даст возможность определить, какие строки этой таблицы следует добавлять в результирующий набор. В условии поиска укажите имя столбца из таблицы и задайте условие, которому должно соответствовать содержимое ячейки в этом столбце, чтобы строка попала в результирующий набор. Например, должно ли значение в ячейке быть равно `=`, больше `>` или меньше `<` заданного вами значения.

База данных перебирает каждую строку в таблице и проверяет, возвращает ли условие значение `true`. Если да, то значения из столбцов, указанных после команды `SELECT`, добавляются в результирующий набор.

Если указанное в условии значение является текстовым, то заключите его в одинарные кавычки.

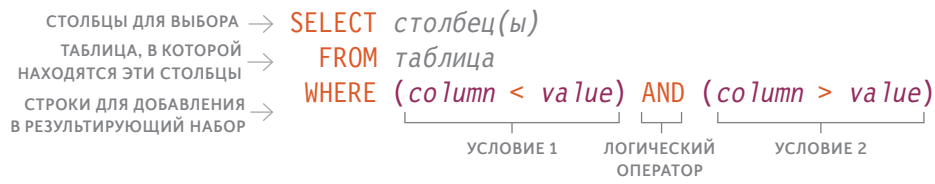
Если значение, указанное в условии, — число, не заключайте его в кавычки.

MySQL не поддерживает тип данных `boolean`, но вы можете использовать тип данных `tinyint`, чтобы представлять логические значения: 1 для `true` и 0 для `false`. Поскольку эти значения — числа, их не следует заключать в кавычки.



Вы можете комбинировать несколько условий поиска, используя три логических оператора, показанных в таблице справа. Они работают так же, как логические операторы PHP. Каждое условие поиска заключается в круглые скобки, чтобы визуально выделить его.

ОПЕРАТОР	ОПИСАНИЕ
`AND`	Все условия должны возвращать значение `true`
`OR`	Любое одно условие может возвращать значение `true`
`NOT`	Отменяет условие; проверяет, что оно неверно



ИСПОЛЬЗОВАНИЕ ОПЕРАТОРОВ СРАВНЕНИЯ В SQL

SQL section_c/c11/comparison-operator-1.sql

```
SELECT email
  FROM member
 WHERE forename = 'Ivy';
```

result set

email
ivy@eg.link

В этом примере происходит выборка адреса электронной почты всех пользователей, у которых в столбце `forename` таблицы `member` стоит значение `Ivy`. Выполните этот SQL-запрос в phpMyAdmin.

Упражнение. Найти адреса электронной почты всех пользователей, которых зовут Luke.

SQL section_c/c11/comparison-operator-2.sql

```
SELECT email
  FROM member
 WHERE id < 3;
```

result set

email
ivy@eg.link
luke@eg.link

В этом примере происходит поиск адресов электронной почты тех пользователей, у которых в столбце `id` таблицы `member` стоит значение меньше 3.

Упражнение. Найти адреса электронной почты пользователей, чей идентификатор меньше или равен 3.

SQL section_c/c11/logical-operator.sql

```
SELECT email
  FROM member
 WHERE (email > 'E') AND (email < 'L');
```

result set

email
emi@eg.link
ivy@eg.link

В этом примере оператор `AND` используется для поиска строк, в которых адрес электронной почты пользователя должен соответствовать двум условиям, а именно начинаться с:

- буквы больше, чем E;
- буквы меньше L.

Упражнение. Найти адреса электронной почты всех пользователей, чей адрес электронной почты начинается с букв G–L.

ПОИСК СТРОК С ПОМОЩЬЮ ОПЕРАТОРА LIKE И ПОДСТАНОВОК

Оператор LIKE может использоваться в условии для поиска подстроки, то есть для нахождения таких строк в таблице, где значение в указанном столбце содержит определенные символы.

Оператор LIKE находит строки данных, в которых ячейка в соответствующем столбце содержит символы, соответствующие указанному вами **шаблону**. Например, шаблон можно использовать для поиска строк, в которых значение в указанном столбце:

- начинается с указанной буквы;
- заканчивается указанным числом;
- содержит указанное слово или набор символов (это часто используется для реализации функции поиска).

Шаблон может использовать **подстановочные символы**, чтобы указать, где могут находиться другие символы (как показано в таблице справа):

- `%` — соответствует любому количеству символов, включая ноль;
- `_` — соответствует одному символу.

ЗНАЧЕНИЕ	СООТВЕТСТВУЕТ СТОЛБЦАМ, ЗНАЧЕНИЕ КОТОРЫХ
`To%`	Начинается с *To*
`%day`	Заканчивается на *day*
`%to%`	Содержит *to* в любой части строки
`h_ll`	Состоит из 4 букв, где вместо подчеркивания может быть любая буква (например, hall, hell, hill, hull)
`%h_ll%`	Содержит подстроку, в которой сначала идет *h*, затем любой символ, затем *ll* (например, hall, hell, chill, hilly, shellac, chilled, hallmark, hullabaloo
`1%`	Начинается с *1*
`%!`	Заканчивается на !

По умолчанию сравнение строк в MySQL нечувствительно к регистру[53]. Это означает, что при поиске имени `Ivy` также будут найдены значения `IVY` и `ivy`.

СТОЛБЦЫ ДЛЯ ВЫБОРА → `SELECT столбец(ы)`
ТАБЛИЦА, В КОТОРОЙ НАХОДЯТСЯ ЭТИ СТОЛБЦЫ → `FROM таблица`
УСЛОВИЕ ДЛЯ ДОБАВЛЕНИЯ СТРОКИ В РЕЗУЛЬТИРУЮЩИЙ НАБОР → `WHERE column LIKE '%value%';`

ОПЕРАТОР LIKE
ПОДСТАНОВОЧНЫЕ СИМВОЛЫ

ПОИСК ЗНАЧЕНИЙ

SQL section_c/c11/like-1.sql

```
SELECT email
  FROM member
 WHERE forename LIKE 'I%';
```

result set

email
ivy@eg.link

В этом примере выполняется поиск адресов электронной почты всех пользователей, чье имя начинается с буквы *I* (без учета регистра).

Упражнение. Найти пользователей, имя которых начинается на букву *E*.

SQL section_c/c11/like-2.sql

```
SELECT email
  FROM member
 WHERE forename LIKE 'E_I%';
```

result set

email
emi@eg.link

В этом примере будут получены адреса электронной почты всех пользователей, чье имя:

- начинается с буквы *E*;
- за ней следует другой символ;
- затем буква *I*;
- затем любые другие символы.

Должны быть возвращены такие имена, как Eli, Elias, Elijah, Elisha, Emi, Emiko, Emil, Emilio, Emily, Eoin и Eric.

Упражнение. Найти пользователей, имя которых совпадает с 'L_K%'.

SQL section_c/c11/like-3.sql

```
SELECT email
  FROM member
 WHERE forename LIKE 'Luke';
```

result set

email
luke@eg.link

В этом примере выполняется поиск адреса электронной почты любого пользователя с именем *Luke*. Поскольку в нем нет подстановочных символов, код возвращает только точные совпадения.

Упражнение. Найти пользователей по имени *Ivy*.

УПРАВЛЕНИЕ ПОРЯДКОМ СТРОК В РЕЗУЛЬТИРУЮЩЕМ НАБОРЕ

Чтобы управлять порядком, в котором строки добавляются в результирующий набор, используйте оператор ORDER BY (упорядочить по), за которым укажите имя столбца, по значениям которого будут сортироваться результаты. Далее укажите ASC для сортировки по возрастанию или DESC — по убыванию.

ORDER BY добавляется в конец запроса, чтобы управлять порядком добавления строк в результирующий набор. Значения в указанном столбце используются для управления порядком представления результатов.

За именем столбца может следовать одно из двух ключевых слов: ASC для возрастания или DESC для убывания. Если вы не укажете его, то результат будет отсортирован в порядке возрастания, но ваш код SQL будет понятнее, если вы напишете ASC явно.



Вы можете отсортировать порядок добавления строк в результирующий набор, используя значения из нескольких столбцов. Имя каждого столбца отделяется запятой. Если первый столбец, используемый для сортировки значений, содержит идентичные значения, то такие строки будут отсортированы по значениям второго указанного столбца.

Например, если вы сортировали пользователей по их имени и у нескольких пользователей будет одинаковое имя (в столбце forename), можно затем упорядочить их по фамилии (сохраненной в столбце surname).



СОРТИРОВКА РЕЗУЛЬТАТОВ

SQL section_c/c11/order-by-1.sql

```
SELECT email
  FROM member
ORDER BY email DESC;
```

result set

email
luke@eg.link
ivy@eg.link
emi@eg.link

Этот пример получает все адреса электронной почты и сортирует их в порядке убывания.

Упражнение. Получить обратный порядок результатов, изменив DESC на ASC.

SQL section_c/c11/order-by-2.sql

```
SELECT title, category_id
  FROM article
ORDER BY category_id ASC, title ASC;
```

result set (показаны первые 10 строк из 24)

title	category_id
Chimney Business Cards	1
Milk Beach Album Cover	1
Polite Society Posters	1
Systemic Brochure	1
The Ice Palace	1
Travel Guide	1
Chimney Press Website	2
Floral Website	2
Milk Beach Website	2
Polite Society Website	2

Этот пример возвращает значения из столбцов `title` и `category_id` таблицы `article` и упорядочивает их сначала по `category_id` в порядке возрастания, а затем по `title` в алфавитном порядке.

Весь результирующий набор содержит больше строк, чем показано слева (поскольку это все строки таблицы `article`), но здесь недостаточно места, чтобы отобразить их все.

Упражнение. Выбрать столбцы `member_id` и `title` из таблицы `article` и упорядочить результаты сначала по `member_id`, а затем по `title` в алфавитном порядке.

ПОДСЧЕТ И ГРУППИРОВКА РЕЗУЛЬТАТОВ

Если после команды SELECT вместо списка полей использовать функцию SQL COUNT( ), то она вернет общее количество строк, соответствующих запросу. Группировка результатов позволяет подсчитать, сколько строк содержат одно и то же значение.

Чтобы подсчитать количество строк в таблице, вызовите функцию COUNT( ) после команды SELECT и укажите название таблицы. Используйте символ звездочки * (которая используется как символ подстановки) в качестве аргумента для функции COUNT( ).

ФУНКЦИЯ

```
SELECT COUNT(*)
  FROM table;
```

Если вы добавляете условие поиска в запрос, функция COUNT( ) возвращает количество строк, соответствующих запросу с условием поиска.

```
SELECT COUNT(*)
  FROM table
 WHERE column LIKE '%value%';
```

Если вы укажете имя столбца в качестве аргумента для функции COUNT( ), она подсчитает количество строк, в которых значение в указанном столбце не равно NULL.

```
SELECT COUNT(column)
  FROM table;
```

Оператор GROUP BY можно использовать с функцией SQL COUNT( ), чтобы определить, сколько строк содержат одинаковое значение в столбце. Например, можно подсчитать, сколько публикаций разместил каждый пользователь или сколько публикаций входит в каждую категорию. После команды SELECT:

1) выберите имя столбца, который может содержать одинаковые значения (например, member_id или category_id);
2) используйте COUNT(*) для подсчета количества строк;
3) укажите имя таблицы;
4) напишите оператор GROUP BY и после него то же самое имя столбца. Тогда MySQL сгруппирует строки с одинаковым значением в этом столбце и подсчитает их.

СТОЛБЕЦ, КОТОРЫЙ МОЖЕТ СОДЕРЖАТЬ ОДИНАКОВЫЕ ЗНАЧЕНИЯ

```
SELECT column, COUNT(*)
  FROM table
GROUP BY column;
```

СТОЛБЕЦ, КОТОРЫЙ МОЖЕТ СОДЕРЖАТЬ ОДИНАКОВЫЕ ЗНАЧЕНИЯ

ПОДСЧЕТ КОЛИЧЕСТВА СТРОК, ПОДХОДЯЩИХ ПОД УСЛОВИЕ

SQL section_c/c11/count-1.sql

```
SELECT COUNT(picture)
  FROM member;
```

result set

COUNT(picture)
2

В этом примере функция SQL `COUNT()` используется для подсчета количества пользователей, загрузивших фотографию профиля. Значения в столбце picture, содержащие `NULL`, учитываться не будут.

Упражнение. Подсчитать количество пользователей с адресом электронной почты.

SQL section_c/c11/count-2.sql

```
SELECT COUNT(*)
  FROM article
 WHERE title LIKE '%design%' OR content LIKE '%design%';
```

result set

COUNT(*)
9

В этом примере функция SQL `COUNT()` используется для того, чтобы вернуть количество публикаций, у которых в столбцах `title` или `content` встречается слово design.

Упражнение. Найти количество публикаций, содержащих слово photo.

SQL section_c/c11/count-3.sql

```
SELECT member_id, COUNT(*)
  FROM article
 GROUP BY member_id;
```

result set

member_id	COUNT(*)
1	10
2	8
3	6

Этот запрос подсчитывает количество публикаций каждого пользователя. Оператор `SELECT` выбирает столбец `member_id`, а функция `COUNT()` подсчитывает количество совпадающих строк. Оператор `FROM` указывает, что просматривается таблица `article`. Оператор `GROUP BY` группирует значения в столбце `member_id`, чтобы вы смогли увидеть количество размещенных каждым пользователем публикаций.

Упражнение. Подсчитать количество публикаций в каждой категории.

ОГРАНИЧЕНИЕ КОЛИЧЕСТВА ВОЗВРАЩАЕМЫХ РЕЗУЛЬТАТОВ

Команда LIMIT ограничивает количество строк, добавляемых в результирующий набор, а OFFSET указывает базе данных, сколько найденных записей надо пропустить, перед тем как начать добавлять их в результирующий набор.

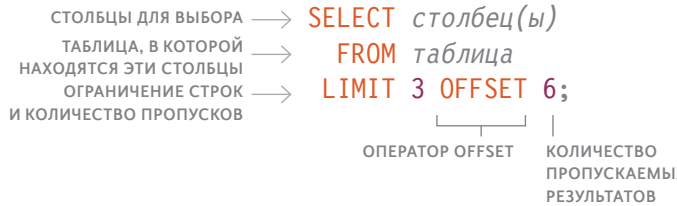
Чтобы ограничить общее количество строк, добавляемых в результирующий набор, используйте оператор LIMIT.

Следующий запрос добавит в результирующий набор только первые пять результатов, соответствующих запросу.



Оператор OFFSET можно использовать после оператора LIMIT, чтобы пропустить некоторое количество строк, перед тем как остальные будут добавлены в результирующий набор.

В приведенном ниже примере будут пропущены первые шесть результатов, соответствующих запросу, а затем добавлены следующие три в результирующий набор.



LIMIT и OFFSET часто используются, когда запрос генерирует много результатов. Результаты разбиваются на отдельные страницы с использованием метода, называемого **разбивкой на страницы** (**пагинацией**). Один из хорошо известных примеров — результаты поиска Google.

Внизу первой страницы расположены ссылки на дополнительные, отображающие больше результатов, соответствующих тому же запросу. В следующей главе вы узнаете, как использовать эти команды для добавления разбивки на страницы.

ОГРАНИЧЕНИЕ КОЛИЧЕСТВА РЕЗУЛЬТАТОВ

SQL section_c/c11/limit.sql

```
SELECT title
  FROM article
 ORDER BY id
 LIMIT 1;
```

result set
title
Systemic Brochure

В этом примере запрашиваются названия публикаций, упорядоченные по значению в столбце `id`. Здесь используется оператор `LIMIT`, чтобы добавить только первое совпадение в результирующий набор.

Упражнение. Получить первые пять публикаций из категории `print`.

SQL section_c/11/offset.sql

```
SELECT title
  FROM article
 ORDER BY id
 LIMIT 3 OFFSET 9;
```

result set
title
Polite Society Mural
Stargazer Website and App
The Ice Palace

В этом примере запрашиваются названия публикаций, упорядоченные по значению в столбце `id`. Здесь используется оператор `OFFSET`, чтобы пропустить первые девять результатов, соответствующих запросу, а также применяется оператор `LIMIT`, чтобы добавить следующие три совпадения в результирующий набор.

Упражнение. Пропустить первые шесть совпадений и вернуть следующие шесть значений.

ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА JOIN ДЛЯ ПОЛУЧЕНИЯ ДАННЫХ ИЗ ДВУХ ТАБЛИЦ

Команда `JOIN` позволяет запрашивать данные из нескольких таблиц. Данные из каждой таблицы добавляются в одну строку результирующего набора.

При проектировании базы данных необходимо создавать отдельную таблицу для каждой использующейся на сайте сущности и избегать дублирования данных в таблицах.

На нашем учебном веб-сайте данные о публикациях, категориях, пользователях и изображениях хранятся в разных таблицах. Первый столбец в этих таблицах содержит значение, используемое для идентификации каждой строки таблицы. Например, значение в столбце `id` таблицы `category` позволяет однозначно идентифицировать каждую категорию. Это значение является **первичным ключом**.

В таблице `article` необходимо хранить сведения о том, к какой категории относится каждая публикация. Вместо того чтобы дублировать название категории для каждой публикации, в таблице используется столбец с именем `category_id`. Значение в этом столбце известно как **внешний ключ**, и оно будет соответствовать первичному ключу категории, к которой относится публикация.

Первичный и внешний ключи описывают, как данные в одной строке таблицы могут быть **связаны** со строками другой таблицы. На иллюстрации ниже показана взаимосвязь между публикацией и категорией, к которой она относится.

Когда вы пишете SQL-запрос для получения информации о публикации и хотите включить информацию из другой таблицы (например, название категории, к которой она относится), публикация представляет собой основной объект запроса. Поэтому таблица `article` известна как **левая таблица**.

Когда вы получаете дополнительные данные о публикации из второй таблицы (например, таблицы категорий), она становится **правой таблицей**.

Оператор `JOIN` описывает, как соотносятся строки в таблицах.

article

id	title	summary	content	created	category_id	member_id	image_id	published
1	Systemic Bro…	Brochure…	This bro…	2021-01-26	1	2	1	1
2	Forecast	Handbag…	This dra…	2021-01-29	3	2	2	1
3	Swimming Pool	Architec…	This pho…	2021-02-02	4	1	3	1

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1

До сих пор в этой главе запросы получали информацию только из одной таблицы. Когда используется оператор JOIN для получения данных из более чем одной таблицы, к именам столбцов добавляются имена таблиц, к которым они относятся. Для этого:

- напишите имя таблицы, в которой находится нужный столбец;
- за ним символ точки;
- и название столбца.

Приведенный ниже запрос выбирает название и аннотацию (summary) для всех строк из таблицы `article`, а также получает из таблицы `category` название категории, к которой относится публикация.

1. За командой SELECT следуют имена столбцов, из которых должны быть возвращены данные.

2. За командой FROM следует имя левой таблицы (основной объект запроса). В данном случае это таблица `article`.

3. За командой JOIN следует имя правой таблицы (той, что содержит дополнительную информацию). В данном случае это таблица `category`.

Затем необходимо сообщить базе данных, какие именно строки из правой таблицы должны соответствовать строкам левой при объединении.
Для этого используется ключевое слово ON, за которым следует условие объядинения:

- столбец в левой таблице, содержащий внешний ключ;
- знак равенства;
- столбец в правой таблице, содержащий первичный ключ.

```
SELECT article.title, article.summary, category.name
  FROM article
  JOIN category ON article.category_id = category.id;
                   ВНЕШНИЙ КЛЮЧ        ПЕРВИЧНЫЙ КЛЮЧ
```

В результирующем наборе ниже показаны первые три добавленных в него строки данных (полный результирующий набор будет содержать все элементы таблицы `article`).

Примечание. Имена столбцов в результирующем наборе не используют имена таблиц, поскольку данные, взятые из двух таблиц, были объединены в один результирующий набор[54].

result set (показано 3 строки из 24)

title	summary	name
Milk Beach Website	Website for music series	Digital
Wellness App	App for health facility	Digital
Stargazer Website and App	Website and app for music festival	Digital

Чтобы выбрать отдельную строку или подмножество строк, после JOIN может быть добавлено условие поиска. Например, приведенный ниже запрос вернет сведения только о публикациях, входящих в категорию `print`.

За условием поиска также могут идти любые другие команды SQL, рассмотренные в этой главе, например упорядочивающие или ограничивающие количество строк в результирующем наборе.

```
SELECT article.title, article.summary, category.name
  FROM article
  JOIN category ON article.category_id = category.id
 WHERE category.id = 1;
```

ПОВЕДЕНИЕ ОПЕРАТОРА JOIN ПРИ ОТСУТСТВИИ ДАННЫХ

Когда база данных выполняет оператор JOIN и в правой таблице не оказывается строки, соответствующей строке в левой таблице, вы можете указать, следует ли добавлять данные хотя бы из левой таблицы в строку результирующего набора или пропустить ее целиком.

Представьте, что помимо самой публикации вы хотите получить информацию о соответствующем ей изображении. Вы можете использовать JOIN для присоединения соответствующей строки из таблицы image (так же, как получали название категории в предыдущем примере).

Столбец image_id таблицы article является внешним ключом, поскольку его значение — первичный ключ из таблицы image, содержащей изображение для публикации.

Однако есть одно ключевое отличие. Хотя каждая публикация *должна* принадлежать к определенной категории (база данных обеспечивает это с помощью так называемого ограничения (constraint)), публикация может и не содержать изображение.

Если для публикации не загружено изображение, то в столбце image_id таблицы article будет записано значение NULL.

В приведенной ниже таблице article столбец image_id одной из строк содержит значение NULL, поскольку для публикации не загружено изображение. Это означает, что оператор JOIN не найдет соответствующую строку в таблице image.

На странице справа показаны два типа оператора JOIN. С их помощью указывается, должен ли запрос все равно добавлять в результирующий набор данные из левой таблицы или же следует пропустить всю строку, поскольку не удалось найти соответствующую строку в правой таблице (то есть не удалось найти соответствующее публикации изображение).

article

id	title	summary	content	created	category_id	member_id	image_id	published
4	Walking Birds	Artwork …	The brie…	2021-02-12	3	3	4	1
5	Sisters	Editoria…	The arti…	2021-02-27	3	3	*NULL*	1
6	Micro-Dunes	Photogra…	This pho…	2021-03-03	4	1	6	1

image

id	file	alt
4	birds.jpg	Collage of two birds
6	micro-dunes.jpg	Photograph of tiny sand dunes

ОПЕРАТОР INNER JOIN

Оператор внутреннего объединения (**inner join**) добавляет данные в результирующий набор, только если БД находит соответствующие друг другу строки в обеих таблицах. Чтобы создать внутреннее объединение, используйте оператор `INNER JOIN` (который сокращается до просто `JOIN`, так что оба этих написания эквивалентны).

Если такой запрос выполнить для таблиц, показанных на странице слева, то публикация с идентификатором 5 не будет добавлена в результирующий набор, поскольку ее столбец `image_id` содержит значение `NULL` и, следовательно, объединение для этой строки создать невозможно.

```
SELECT article.id, article.title, image.file
  FROM article
  JOIN image ON article.image_id = image.id;
```

result set (показаны первые 5 строк из 23)

id	title	file
1	Systemic Brochure	systemic-brochure.jpg
2	Forecast	forecast.jpg
3	Swimming Pool	swimming-pool.jpg
4	Walking Birds	birds.jpg
6	Micro-Dunes	micro-dunes.jpg

ОПЕРАТОР LEFT OUTER JOIN

Оператор левого внешнего объединения (**left outer join**) добавляет в результирующий набор все запрошенные строки из левой таблицы, при этом используя `NULL` для всех значений, которые невозможно получить из правой таблицы. Чтобы создать левое внешнее объединение, используйте либо оператор `LEFT JOIN`, либо `LEFT OUTER JOIN`.

Если такой запрос выполнить для таблиц, показанных на странице слева, то публикация с идентификатором 5 добавляется в результирующий набор, но значение в столбце `file` будет равно `NULL`, поскольку соответствующее изображение не найдено.

```
   SELECT article.id, article.title, image.file
     FROM article
LEFT JOIN image ON article.image_id = image.id;
```

result set (показаны первые 5 строк из 24)

id	title	file
1	Systemic Brochure	systemic-brochure.jpg
2	Forecast	forecast.jpg
3	Swimming Pool	swimming-pool.jpg
4	Walking Birds	birds.jpg
5	Sisters	*NULL*

ПОЛУЧЕНИЕ ДАННЫХ ИЗ НЕСКОЛЬКИХ ТАБЛИЦ

В запросе SELECT может использоваться несколько операторов JOIN для получения данных из более чем двух таблиц.

Чтобы получить данные из нескольких таблиц, можно добавить несколько операторов JOIN.

- После инструкции SELECT укажите имена всех столбцов, из которых вы хотите получить данные (используя имя таблицы, точку, затем имя столбца).
- Используйте оператор объединения, чтобы указать связь между таблицами.

Приведенный ниже запрос собирает данные о публикации из трех таблиц: `article`, `category` и `image`.

Таблица `article` — это левая таблица. Из нее берется название и описание публикации.

В таблице `category` указано название категории, к которой относится публикация. Поскольку каждая публикация обязательно относится к какой-либо категории, используется оператор JOIN (который реализует объединение inner join).

Таблица `image` содержит имя файла и сопроводительный текст для изображения (alt text), относящегося к публикации. Поскольку в каждой публикации необязательно присутствует изображение, используется оператор LEFT JOIN, чтобы обеспечить добавление в результирующий набор всех строк из левой таблицы.

```
SELECT article.title, article.summary,
       category.name,
       image.file, image.alt
  FROM article
  JOIN category   ON article.category_id = category.id
  LEFT JOIN image ON article.image_id    = image.id
 ORDER BY article.id ASC;
```

result set (показаны 3 строки из 24)

title	summary	name	file	alt
Systemic Brochure	Brochure design for…	Print	systemic-brochure.jpg	Brochure…
Forecast	Handbag illustration…	Illustration	forecast.jpg	Illustrati…
Swimming Pool	Architecture magazine…	Photography	swimming-pool.jpg	Photograph…

ПРИМЕНЕНИЕ НЕСКОЛЬКИХ ОПЕРАТОРОВ JOIN

SQL section_c/c11/joins.sql

```
(1) SELECT article.id, article.title,
           category.name,
           image.file, image.alt

(2)   FROM article
(3)   JOIN category   ON article.category_id = category.id
(4)   LEFT JOIN image ON article.image_id    = image.id

(5) WHERE article.category_id = 3
      AND article.published   = 1
(6) ORDER BY article.id DESC;
```

result set

id	title	name	file	alt
21	Stargazer	Illustration	stargazer-masc…	Illustrat…
17	Snow Search	Illustration	snow-search.jpg	Illustrat…
10	Polite Society…	Illustration	polite-society…	Mural for…
5	Sisters	Illustration	*NULL*	*NULL*
4	Walking Birds	Illustration	birds.jpg	Collage…
2	Forecast	Illustration	forecast.jpg	Illustrat…

Упражнение. Получить тот же набор данных, но для публикаций, у которых в таблице `category` в столбце `id` стоит значение 2.

Упражнение. Добавить имя и фамилию (`forename` и `surname`) автора из таблицы `member`.

Этот запрос получает данные о нескольких публикациях, которые входят в указанную категорию.

1. За инструкцией `SELECT` следуют имена столбцов, которые должны быть добавлены в результирующий набор. Данные выбираются из таблиц `article`, `category` и `image`.
2. Оператор `FROM` указывает, что левой таблицей является таблица `article`.
3. Первый оператор объединения указывает, что из таблицы `category` следует присоединить строку, в столбце `id` которой содержится то же значение, что и в столбце `category_id` таблицы `article`.
4. Второй оператор объединения указывает, что данные из таблицы `image` должны браться из строки, столбец `id` которой содержит то же значение, что и столбец `image_id` таблицы `article`. Это `LEFT JOIN`, поэтому все отсутствующие данные заменяются значением `NULL`.
5. Оператор `WHERE` ограничивает выборку, чтобы в нее попали только те строки, в которых содержится значение 3 в столбце `category_id` и значение 1 в столбце `published` в таблице `article`.
6. Оператор `ORDER BY` управляет порядком добавления результатов в результирующий набор, используя идентификаторы публикаций в порядке убывания.

ПСЕВДОНИМЫ

Псевдоним (alias) для имени таблицы облегчает чтение запросов, использующих объединения. Псевдонимы для имен столбцов задают то, какие имена будут у колонок в результирующем наборе.

В сложных SQL-запросах, где операторы объединения выбирают данные из нескольких таблиц, удобнее присвоить каждой таблице псевдоним.

Псевдоним таблицы подобен сокращению имени таблицы, что уменьшает объем текста в запросе.

Псевдоним таблицы создается в командах FROM или JOIN. После имени таблицы используйте ключевое слово AS, после которого укажите псевдоним для этой таблицы. Затем в запросе используйте псевдоним вместо полного имени таблицы.

```
SELECT t1.column1, t1.column2, t2.column3
  FROM table1 AS t1
  JOIN table2 AS t2 ON t1.column4 = t2.column1;
```

СОЗДАНИЕ ПСЕВДОНИМА — ПСЕВДОНИМ ДЛЯ ТАБЛИЦЫ 1 — ПСЕВДОНИМ ДЛЯ ТАБЛИЦЫ 2

Имена столбцов в результирующем наборе обычно берутся из имен столбцов в таблицах, предоставивших данные для сбора.

Псевдонимы столбцов позволяют изменять имя столбца в результирующем наборе. Они также могут присвоить имя столбцу, содержащему результат функции, например COUNT().

СТОЛБЕЦ БАЗЫ ДАННЫХ — ПСЕВДОНИМ ДЛЯ РЕЗУЛЬТИРУЮЩЕГО НАБОРА

```
SELECT column1 AS newname1
  FROM table;
```

Чтобы создать псевдоним столбца, после указания имени столбца, из которого вы хотите получить данные, используйте ключевое слово AS и имя столбца, которое должно использоваться в результирующем наборе.

Точно так же после функции COUNT() напишите ключевое слово AS и имя, которое будет у столбца с результатами подсчета в результирующем наборе.

ФУНКЦИЯ COUNT — ПСЕВДОНИМ ДЛЯ РЕЗУЛЬТИРУЮЩЕГО НАБОРА

```
SELECT COUNT(*) AS members
  FROM members;
```

ИСПОЛЬЗОВАНИЕ ПСЕВДОНИМОВ ДЛЯ ИМЕН ТАБЛИЦ И СТОЛБЦОВ

SQL section_c/c11/table-alias.sql

```
SELECT a.id, a.title,
       c.name,
       i.file, i.alt

  FROM article     AS a
  JOIN category    AS c  ON a.category_id = c.id
  LEFT JOIN image  AS i  ON a.image_id    = i.id

 WHERE a.category_id = 3
   AND a.published   = 1
 ORDER BY a.id DESC;
```

Этот запрос получает те же данные, что и в предыдущем примере, но он задает псевдонимы для имен таблиц после команд FROM и JOIN:

- a для таблицы article;
- c для таблицы category;
- i для таблицы image.

После этого псевдонимы будут использоваться вместо полного названия таблицы в команде SELECT, операторах WHERE, AND и ORDER BY.

result set

id	title	name	file	alt
21	Stargazer	Illustration	stargazer-masc…	Illustrat…
17	Snow Search	Illustration	snow-search.jpg	Illustrat…
10	Polite Society…	Illustration	polite-society…	Mural for…
5	Sisters	Illustration	*NULL*	*NULL*
4	Walking Birds	Illustration	birds.jpg	Collage…
2	Forecast	Illustration	forecast.jpg	Illustrat…

Упражнение. Изменить псевдонимы на:

- art для таблицы article;
- cat для таблицы category;
- img для таблицы image.

SQL section_c/c11/column-alias.sql

```
SELECT forename AS firstname, surname AS lastname
  FROM member;
```

В этом примере выбираются значения из столбцов forename и surname таблицы member и используются псевдонимы, чтобы дать им новые имена столбцов в результирующем наборе.

result set

firstname	lastname
Ivy	Stone
Luke	Wood
Emiko	Ito

Упражнение. Подсчитать количество публикаций в таблице article и использовать псевдоним, чтобы результат выводился в колонке с названием articles.

ОБЪЕДИНЕНИЕ СТОЛБЦОВ И АЛЬТЕРНАТИВА ДЛЯ NULL

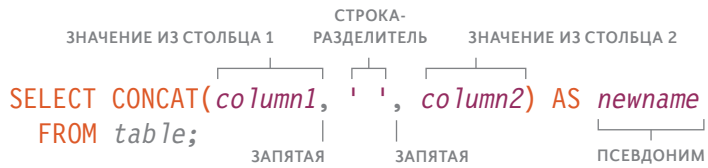
Функция CONCAT() объединяет несколько значений в один столбец результирующего набора. Функция COALESCE() возвращает первое из переданных значений, которое не равно NULL.

Функция SQL CONCAT() используется для объединения значений из двух или более столбцов таблицы в один столбец в результирующем наборе.

Обычно между значениями объединяемых столбцов добавляется строка или символ, разделяющий их. Результату функции обычно присваивается псевдоним для указания имени этого столбца в результирующем наборе.

Как и в случае с любой функцией, аргументы, передаваемые в функцию CONCAT() для создания нового значения, разделяются запятой. Ниже значения из двух столбцов объединяются и между ними вставляется пробел.

Если значение одного из столбцов равно NULL, то функция вернет NULL.

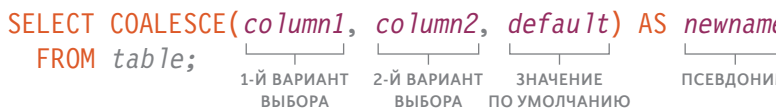


Если заранее известно, что значение одного из столбцов может быть равно NULL, примените функцию SQL COALESCE(), чтобы в этом случае она использовала:

- имя другого столбца. Функция вернет его значение, если оно не равно NULL;
- значение по умолчанию. Функция вернет его, если все остальные варианты содержат NULL.

Если значение первого аргумента равно NULL, то проверяется значение второго, и так далее. Если значения всех остальных аргументов также равны NULL, то будет использоваться самое последнее.

При применении функции COALESCE() можно указать псевдоним для имени столбца в результирующем наборе, задав свое собственное имя получаемому значению.



ФУНКЦИИ CONCAT И COALESCE

SQL section_c/c11/concat.sql

```
SELECT CONCAT(forename, ' ', surname) AS author
  FROM member;
```

result set

author
Ivy Stone
Luke Wood
Emiko Ito

В этом примере используется функция `concat()`для объединения столбцов `forename` и `surname` в столбец с названием `author`.

Между значениями добавляется пробел, разделяющий имя и фамилию пользователя.

Упражнение. Добавьте адрес электронной почты к выбранным значениям и назовите псевдоним `author_details`.

SQL section_c/c11/coalesce.sql

```
SELECT COALESCE(picture, forename, 'friend') AS profile
  FROM member;
```

result set

profile
ivy.jpg
Luke
emi.jpg

В этом примере используется функция SQL `COALESCE()` для указания альтернативных вариантов, если значение в столбце с фотографией профиля таблицы `member` равно `NULL` (поскольку пользователь не загрузил фотографию). Здесь оператор `SELECT` ищет значение в столбце `picture` таблицы `member`. Далее:

- если значение равно `NULL`, будет использоваться значение из `forename`;
- если и это значение равно `NULL`, вместо него будет показан текст `friend`.

Псевдоним указывает имя, которое этот столбец будет иметь в результирующем наборе. В данном случае это имя — `profile`.

Упражнение. Сделать значением по умолчанию `placeholder.png`, если пользователь не загрузил изображение.

ЗАПРОСЫ К ТАБЛИЦЕ ARTICLE В УЧЕБНОЙ CMS

CMS, которая будет создаваться в следующих главах, использует SQL-запросы для отображения информации об отдельной публикации, а также списка карточек публикаций. В этих запросах используются приемы, показанные на протяжении всей текущей главы.

1. Приведенный ниже SQL-запрос получает данные об отдельной публикации из всех четырех таблиц базы данных. За инструкцией `SELECT` следуют имена столбцов, откуда будут собираться данные.

2. Псевдонимы столбцов используются для присвоения новых имен колонкам с именем файла и альтернативным текстом изображения, а также названию категории.

3. Функция `CONCAT()` объединяет имя и фамилию пользователя, разместившего публикацию, а псевдоним столбца указывает, что результирующий набор должен хранить имя в столбце с названием `author`.

4. Левая таблица — это таблица `article`. Три оператора объединения показывают связи с данными в других таблицах.

После команд `FROM` и `JOIN` каждому имени таблицы присваивается псевдоним. При каждом обращении к столбцу данных в запросе эти псевдонимы используются вместо полных имен таблиц.

5. Оператор `WHERE` указывает идентификатор публикации для отображения. Запрос возвращает результат только в том случае, если столбец `published` содержит значение 1.

section_c/c11/article.sql — SQL

```
(1) SELECT a.title, a.summary, a.content, a.created, a.category_id, a.member_id,
(2)        c.name      AS category,
(3)        CONCAT(m.forename, ' ', m.surname) AS author,
(2)        i.file      AS image_file,
(2)        i.alt       AS image_alt

     FROM article     AS a
     JOIN category    AS c   ON a.category_id = c.id
(4)  JOIN member      AS m   ON a.member_id   = m.id
     LEFT JOIN image  AS i   ON a.image_id    = i.id
(5) WHERE a.id        = 22
      AND a.published = 1;
```

result set

title	summary	content	created	category_id	member_id	category	author	image_file	image_alt
Polite…	Poster…	These…	2021-0…	1	1	Print	Ivy St…	polite-so…	Photogra…

1. Приведенный ниже SQL-запрос получает сводную информацию обо всех публикациях в указанной категории, используя данные из всех четырех таблиц базы данных.

За инструкцией SELECT следуют имена столбцов, откуда будут собираться данные.

2. Как и в предыдущем примере, псевдонимы используются для названия категории, а также имени файла и альтернативного текста изображения.

Функция CONCAT( ) также вызывается, чтобы соединить имя и фамилию автора.

3. Объединения таблиц идентичны предыдущему примеру.

4. В операторе WHERE указывается идентификатор категории, в которую должны входить публикации, и что публикация должна быть разрешена к показу. Затем результаты упорядочиваются по id публикаций в порядке убывания.

SQL section_c/c11/article-list.sql

```
1 SELECT a.id, a.title, a.summary, a.category_id, a.member_id,
2        c.name      AS category,
         CONCAT(m.forename, ' ', m.surname) AS author,
         i.file      AS image_file,
         i.alt       AS image_alt

3   FROM article     AS a
    JOIN category    AS c   ON a.category_id = c.id
    JOIN member      AS m   ON a.member_id   = m.id
    LEFT JOIN image  AS i   ON a.image_id    = i.id

4 WHERE a.category_id = 1
    AND a.published   = 1
ORDER BY a.id DESC;
```

result set

id	title	summary	category_id	member_id	category	author	image_file	image_alt
24	Travel Guide	Book de…	1	1	Print	Ivy Stone	feathervi…	Two page…
22	Polite Societ…	Poster…	1	1	Print	Ivy Stone	polite-so…	Photogra…
20	Chimney Busin…	Station…	1	2	Print	Luke Wood	chimney-c…	Business…
14	Milk Beach Al…	Packagi…	1	1	Print	Ivy Stone	milk-beac…	Vinyl LP…
12	The Ice Palace	Book co…	1	2	Print	Luke Wood	the-ice-p…	The Ice…
1	Systemic Broc…	Brochure…	1	2	Print	Luke Wood	systemic-…	Brochure…

Упражнение. Выберите те же данные о каждой публикации, но используйте другое условие в операторе WHERE, чтобы выбрать публикации, размещенные определенным пользователем сайта.

Упражнение. Выберите те же данные о каждой публикации, но используйте оператор LIMIT, чтобы получить шесть самых последних опубликованных публикаций из базы данных (в любой категории).

Упражнение. Выберите те же данные о каждой публикации, но используйте оператор SQL LIKE для извлечения данных о публикациях, название которых содержит термин design.

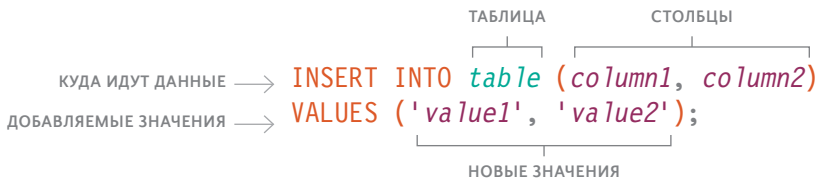
ДОБАВЛЕНИЕ ИНФОРМАЦИИ В БД

Команда SQL `INSERT INTO` добавляет новую строку данных в таблицу. Она может добавлять данные только в одну таблицу за раз.

Команда `INSERT INTO` используется для добавления данных в одну таблицу. За ней следует:

- имя таблицы, в которую добавляются данные;
- круглые скобки, содержащие имена столбцов, в которые добавляются данные.

Затем следует команда `VALUES`, после которой в круглых скобках идут значения, которые вы хотите добавить в эти столбцы. Значения должны отображаться в порядке указания столбцов. Строковые значения должны быть заключены в кавычки; числа — нет.



В учебной базе данных столбец `id` любой таблицы является первичным ключом, поэтому его значение в каждой строке должно быть уникальным.

Чтобы гарантировать уникальность значений для этого столбца, он создается с помощью специального механизма MySQL под названием **автоинкремент**, который генерирует значения для этого столбца автоматически и гарантирует, что они уникальны, увеличивая значение на 1 каждый раз, когда в таблицу добавляется новая строка.

Поскольку это значение создается базой данных автоматически, то при добавлении новой строки в таблицу не следует указывать в запросе ни имя столбца с идентификатором, ни значение для него.

Также в различных таблицах учебной БД есть четыре столбца, для которых указаны **значения по умолчанию**. Это означает, что задавать их значение не обязательно. Значения по умолчанию:

- у столбца `created` таблицы `article` это дата и время добавления строки в БД;
- у столбца `image_id` таблицы `article` это `NULL` (если изображение не загружено);
- у столбца `published` таблицы `article` это 0 (если не задано значение 1, публикация является скрытой);
- у столбца `joined` таблицы `member` это дата и время добавления строки в БД.

SQL section_c/c11/insert-1.sql

```
INSERT INTO category (name, description, navigation)
VALUES ('News', 'Latest news from Creative Folk', 0);
```

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1
4	Photography	Capturing the moment	1
5	News	Latest news from Creative Folk	0

SQL-код слева добавляет категорию под названием News в таблицу category.

В запросе должны быть заданы значения для столбцов name, description и navigation. В нем не должно быть значения для столбца id, поскольку база данных добавляет его с помощью автоинкремента.

Добавленная строка выделена цветом в таблице category слева.

SQL section_c/c11/insert-2.sql

```
INSERT INTO image (file, alt)
VALUES ('bicycle.jpg', 'Photo of bicycle'),
       ('ghost.png',   'Illustration of ghost'),
       ('stamp.jpg',   'Polite Society stamp');
```

image

id	file	alt
22	polite-society-posters.jpg	Photograph of three posters…
23	golden-brown.jpg	Photograph of the interior…
24	featherview.jpg	Two pages from a travel boo…
25	bicycle.jpg	Photo of bicycle
26	ghost.png	Illustration of ghost
27	stamp.jpg	Polite Society stamp

В этом примере в таблицу image добавляются три строки, каждая из них содержит сведения об отдельном изображении.

Для каждого изображения в запросе должны быть заданы имя файла и альтернативный текст. Значение для столбца id не должно указываться, поскольку база данных добавляет его с помощью механизма автоинкремента автоматически.

Значения для каждой строки заключаются в круглые скобки, как и при добавлении одной строки.

Каждая пара круглых скобок отделяется запятой. После последней строки данных стоит точка с запятой (не запятая).

Добавленные строки выделены цветом в таблице image.

Если phpMyAdmin отображает только 25 строк, то можно выбрать режим отображения всех данных в таблице, эта кнопка располагается над полем отображения результата.

Дополнительные данные, добавляемые в БД этими примерами, будут удалены в последующих примерах этой главы.

Вам нужно выполнять все примеры в том же порядке, в котором они приведены в книге. Если вы этого не сделаете, то столкнетесь с ошибками.

ОБНОВЛЕНИЕ ИНФОРМАЦИИ В БД

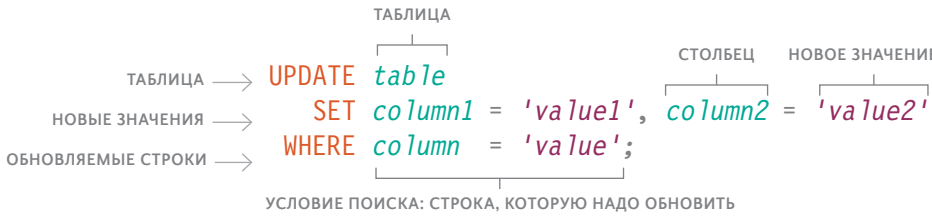
Команда SQL UPDATE позволяет обновлять данные в БД. В операторе SET перечисляются столбцы для обновления и их новые значения. Оператор WHERE определяет, какая строка (или строки) таблицы должны быть обновлены.

Команда UPDATE сообщает БД, что вы хотите обновить в ней данные. После нее пишется название обновляемой таблицы (или таблиц).

После этого оператор SET используется для указания столбцов, которые вы хотите обновить, и новых значений для них. Необходимо указывать имена и значения только для обновляемых столбцов (другие столбцы сохранят уже имеющиеся значения).

Оператор WHERE используется для указания того, какие строки необходимо обновить, точно так же, как он применяется при в запросе SELECT. Если он не используется, обновляются все строки таблицы.

Если условие поиска соответствует нескольким строкам, каждая их них обновляется с одинаковыми значениями. Чтобы обновить данные в нескольких таблицах одновременно, необходимо использовать оператор JOIN.



Часто требуется обновлять только одну строку за раз. В таких случаях оператор WHERE будет использовать первичный ключ таблицы, чтобы указать, какую строку следует обновить. В учебной БД первичный ключ для каждой таблицы — значение в столбце id.

Чтобы обновить несколько строк в таблице, используйте условие поиска, при котором выбирается более одной строки. Например, чтобы скрыть все публикации автора, значение в столбце published необходимо установить в 0, а в условии поиска указать идентификатор автора.

SQL section_c/c11/update-1.sql

```
UPDATE category
   SET name = 'Blog', navigation = 1
 WHERE id = 5;
```

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1
4	Photography	Capturing the moment	1
5	Blog	Latest news from Creative Folk	1

SQL section_c/c11/update-2.sql

```
UPDATE category
   SET navigation = 0
 WHERE navigation = 1;
```

category

id	name	description	navigation
1	Print	Inspiring graphic design	0
2	Digital	Powerful pixels	0
3	Illustration	Hand-drawn visual storytelling	0
4	Photography	Capturing the moment	0
5	Blog	Latest news from Creative Folk	0

SQL-код слева обновляет строку, добавленную в таблицу `category` в предыдущем примере. Он обновляет информацию только в этой строке, поскольку в операторе `WHERE` задано, что должна быть обновлена только та строка таблицы `category`, у которой значение в столбце `id` равно 5.

Этот запрос изменяет значение в столбце `name` на `Blog`, а значение в столбце `navigation` — на 1.

На иллюстрации слева обновленная строка выделена в таблице `category`.
SQL-код слева обновляет те строки в таблице `category`, где столбец `navigation` содержит значение 1, поскольку в операторе `WHERE` задано, что запрос действует на любую строку, где значение в `navigation` равно 1.

В результате во всех строках таблицы `category` в столбце `navigation` будет стоять значение 0. Таким образом, категории перестанут отображаться на панели навигации. Бывают случаи, когда необходимо одним запросом обновить сразу несколько строк в таблице. Но этот пример также подчеркивает, как важно убедиться в том, что ваш SQL затрагивает только те строки, которые надо обновить.

Упражнение. Написать в phpMyAdmin запрос, который снова отобразит все категории.

Примечание. Вы **должны** снова включить категории, чтобы видеть их при выполнении примеров в следующих главах.

Резервное копирование базы данных: поскольку неосторожный SQL-запрос может обновить сразу множество строк в базе данных, рекомендуется создать резервную копию БД перед выполнением новых запросов.

Если SQL-запрос случайно затрагивает больше данных, чем предполагалось, можно использовать резервную копию для восстановления БД до того состояния, которое было до выполнения запроса.

Действия, чтобы создать резервную копию вашей базы данных в phpMyAdmin.

1. Выберите базу данных.

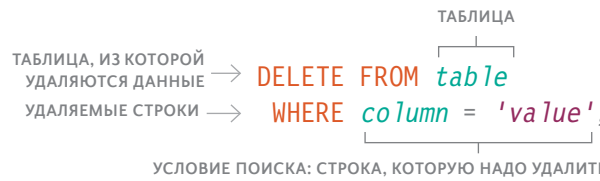
2. Перейдите на вкладку **Export** (Экспорт).

3. Задайте желаемые параметры и нажмите кнопку **Go**. Будет сгенерирован SQL-код. Его необходимо сохранить в текстовом файле. Это похоже на файл, который вы использовали для создания базы данных.

УДАЛЕНИЕ ДАННЫХ ИЗ БД

Команда SQL DELETE удаляет одну или несколько строк из таблицы. Оператор FROM указывает таблицу, откуда удаляются данные. Оператор WHERE определяет, какие строки таблицы должны быть удалены.

Вы можете удалить одну или несколько строк из таблицы одновременно. Сначала используйте команду DELETE FROM, за ней напишите имя таблицы, из которой вы хотите удалить данные, и далее укажите условие поиска, чтобы указать, какие строки следует удалить (если вы этого не сделаете, из таблицы будут удалены все строки данных). Чтобы быть уверенным, что удаляется только одна строка, в условии следует указать столбец с первичным ключом.



Если условию в операторе WHERE соответствует более одной строки, то каждая из них удаляется из таблицы.

С помощью команды DELETE нельзя удалить значения в отдельных столбцах базы данных. Вместо этого применяется запрос UPDATE и в столбец записывается значение NULL.

SQL section_c/c11/delete.sql

```
DELETE FROM category
 WHERE id = 5;
```

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1
4	Photography	Capturing the moment	1
5	Blog	Latest news from Creative Folk	1

SQL SQL НЕ ЗАПУСКАЙТЕ этот пример

```
DELETE FROM category
 WHERE navigation = 1;
```

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1
4	Photography	Capturing the moment	1

SQL-код слева удаляет строку из таблицы `category`, у которой в столбце `id` содержится значение 5. Она содержит категорию `Blog`, добавленную в предыдущих примерах.

Выделенная на иллюстрации строка удаляется из таблицы.

Если под условие в операторе `WHERE` попадает более одной строки, из таблицы удаляются все соответствующие строки.

НЕ ЗАПУСКАЙТЕ ЭТОТ ПРИМЕР

Важно быть осторожным, чтобы с помощью команды `DELETE` не удалить больше данных, чем следует. Например, SQL-запрос слева удаляет все категории, использующиеся при навигации по сайту.

Создание резервной копии базы данных перед выполнением новой инструкции SQL, удаляющей данные из базы данных, гарантирует, что у вас будет копия данных, если запрос удалит больше данных, чем ожидалось.

ОБЕСПЕЧЕНИЕ УНИКАЛЬНОСТИ

Значения в некоторых столбцах должны быть уникальными. Например, у категорий не может быть одинаковых названий или у разных пользователей не может быть одинаковый адрес электронной почты.

Если значение в столбце должно быть уникальным, но в разных строках в нем все же содержатся одинаковые значения, это называется **дубликатом**.

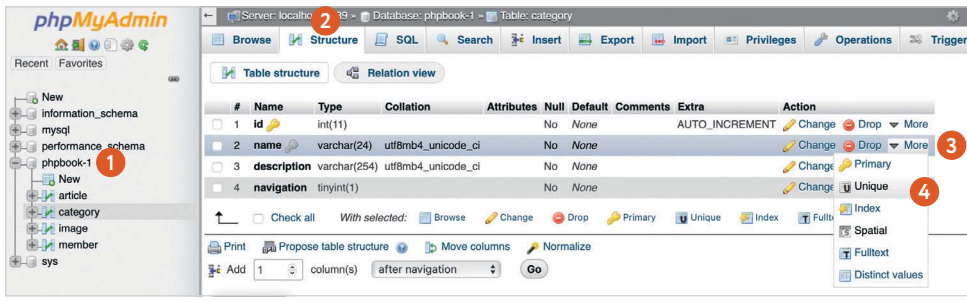
Чтобы столбец не содержал одинаковых значений, необходимо дать указание MySQL применить **ограничение уникальности (unique constraint).** Это название говорит о том, что такое свойство накладывает ограничение на значения, размещенные в этом столбце. Данное ограничение позволяет гарантировать, что значения будут уникальными.

Ограничение уникальности следует использовать только в случае необходимости, потому что при каждом добавлении новых или обновлении существующих данных БД должна проверять все остальные строки в этом столбце, чтобы убедиться, что значение еще не существует. Это может замедлить работу базы данных.

Ниже показано, как добавить ограничение уникальности для названий категорий в phpMyAdmin, чтобы гарантировать, что эти названия не повторяются.

1. Выберите базу данных `phpbook-1`, а затем таблицу `category` в меню слева.
2. Перейдите на вкладку структуры — **Structure**.
3. Каждая строка в таблице ниже представляет собой столбец в базе данных. В строке, представляющей столбец `name`, нажмите на раскрывающееся меню **More**.
4. Нажмите на ссылку с надписью **Unique**.

Теперь, если попытаться добавить или обновить категорию, используя существующее имя, база данных выдаст сообщение об ошибке.



В учебной базе данных есть три таблицы. Каждая из них содержит один столбец, требующий ограничения уникальности, чтобы гарантировать, что все значения в этой колонке разные.

К ним относятся: столбец `title` таблицы `article`, столбец `name` таблицы `category` и столбец `email` таблицы `member`.

ОГРАНИЧЕНИЕ ВНЕШНЕГО КЛЮЧА

Когда существует связь между двумя таблицами, **ограничение внешнего ключа (foreign key constraint)** проверяет, что значение внешнего ключа — это допустимый первичный ключ в другой таблице.

В трех столбцах таблицы `article` используются внешние ключи:

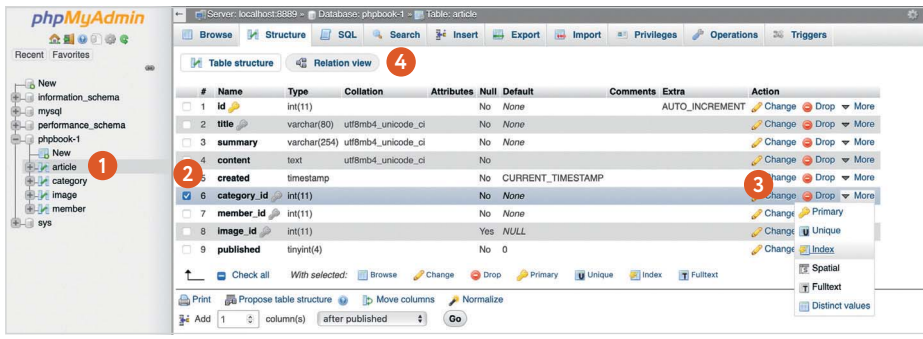
- `category_id` — идентификатор категории, в которой находится публикация;
- `member_id` — идентификатор написавшего ее пользователя;
- `image_id` — идентификатор изображения для этой публикации.

Таблица `article` использует ограничения внешнего ключа, чтобы:

- убедиться, что любое добавляемое в эти столбцы значение существует в колонке id соответствующей таблицы (а если это не так, база данных выдаст ошибку);
- предотвратить удаление категории, пользователя или изображения, если их первичный ключ используется в качестве внешнего ключа в таблице `article`.

Чтобы добавить ограничение внешнего ключа к столбцу таблицы:

1) выберите таблицу, содержащую внешний ключ;
2) установите флажок для столбца с внешним ключом;
3) нажмите на меню **More**, затем пункт **Index**, чтобы добавить индекс к столбцу. Index — это копия выбранных столбцов в таблице, ускоряющая поиск данных в таблице. Но такие копии следует использовать осторожно, так как они могут занимать дополнительное место и замедлять работу базы данных.
4) выберите относительное представление с помощью кнопки **Relation view**;
5) добавьте имя для ограничения;
6) выберите столбец с внешним ключом;
7) выберите таблицу и столбец, содержащие первичный ключ. Затем нажмите кнопку **Save** (не показана).





ЗАКЛЮЧЕНИЕ

ЯЗЫК СТРУКТУРИРОВАННЫХ ЗАПРОСОВ (SQL)

- Язык SQL используется для взаимодействия с базами данных.
- Команда `SELECT` задает, какие столбцы необходимо выбрать из БД и добавить в результирующий набор.
- Команды `INSERT`, `UPDATE` и `DELETE` применяются для создания, обновления и удаления строк данных.
- `FROM` указывает таблицу, с которой будет работать запрос.
- `WHERE` указывает, с какими строками данных нужно работать.
- `JOIN` описывает связи между несколькими таблицами.
- Первичный ключ — это столбец, который содержит уникальное значение для идентификации каждой строки. Значение может создаваться автоматически, с помощью механизма автоинкремента MySQL.
- Внешний ключ — это столбец, в котором хранится первичный ключ другой таблицы и описывается их взаимосвязь.
- Ограничения предотвращают дублирование записей и гарантируют, что внешний ключ соответствует первичному ключу в другой таблице.

12

ПОЛУЧЕНИЕ И ВЫВОД ИНФОРМАЦИИ ИЗ БАЗЫ ДАННЫХ

В этой главе показано, как в PHP можно получать информацию из базы данных и отображать ее на странице, а также как один PHP-файл можно использовать для отображения нескольких страниц сайта.

В предыдущей главе мы познакомились с SQL-запросами, которые получают данные из БД, и с тем, как база данных создает результирующий набор, содержащий запрошенные данные. В этой главе мы рассмотрим, как выполнить SQL из PHP и сохранить полученные данные в переменной, чтобы их можно было использовать на странице. Для этого в PHP есть набор встроенных классов, называемых **PHP Data Objects** (`PDO`).

- Сначала создается объект с использованием класса `PDO` для управления подключением к базе данных. Этот объект должен подключиться к базе данных, прежде чем он сможет запросить у нее данные, точно так же как FTP-программа подключается к FTP-серверу, прежде чем появляется возможность получить файлы, или программа электронной почты подключается к почтовому серверу для получения электронных писем.
- Затем создается объект с использованием класса `PDOStatement`. Каждый такой объект работает с одним SQL-запросом, который вам необходимо выполнить в базе данных. С помощью методов объекта `PDOStatement` можно выполнить запрос и получить данные из результирующего набора, генерируемого базой данных.

На протяжении большей части этой главы каждая строка данных в результирующем наборе будет представлена с помощью массива и сохранена в переменной для использования на странице PHP.

При выводе данных на HTML-страницу в них необходимо экранировать управляющие символы HTML, чтобы предотвратить риск межсайтовой скриптовой атаки.

В конце главы показано, как PDO может сделать полученные данные доступными в виде объектов, а не массивов.


CREATIVE FOLK
Print / Digital / Illustration / Photography
Travel Guide
Book design for series of travel guides
Stargazer
Illustrations for music festival
Golden Brown
Photograph for interior design book
Abandoned Industry
Photograph for magazine feature
Chimney Business Cards
Stationery design for publishing company
Forecast
Illustration for fashion magazine article
CREATIVE FOLK
Polite Society Posters
These posters were designed to increase brand awareness in fashion districts as part of an international campaign ahead of the upcoming autumn-winter season. The client required something aesthetically modern that introduced the vibrant palette of the new collection.

ПОДКЛЮЧЕНИЕ К БАЗЕ ДАННЫХ

Data source name, или DSN, — это строка, которая содержит информацию, необходимую для подключения к БД. При соединении с MySQL она содержит пять фрагментов данных, необходимых PDO для поиска базы данных и подключения к ней. DSN используется для создания объекта PDO с использованием встроенного класса PDO.

Для создания DSN мы будем использовать пять переменных. Данные из этих пяти переменных затем объединяются, и результат сохраняется в шестой переменной (ниже эта переменная называется $dsn).

Переменная $type содержит тип базы данных. Это необходимо, так как PDO может работать со многими типами баз данных. Для MySQL и MariaDB используется значение mysql.

Переменная $server содержит имя хоста сервера, где размещена база данных. Для этого значения используйте:

- localhost, если БД находится на том же сервере, что и веб-сервер (например, если вы используете MAMP или XAMPP);
- IP-адрес или доменное имя сервера, на котором находится база данных, если это не тот же сервер.

Переменная $db содержит имя базы данных, к которой необходимо подключиться. База данных, используемая в этом разделе, называется phpbook-1.

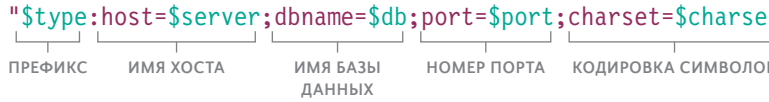
Переменная $port содержит номер порта для базы данных. По умолчанию обычно используется порт 3306, например XAMPP обычно использует именно его. Однако MAMP обычно использует порт 8889.

Переменная $charset — это кодировка символов. В этой кодировке данные будут отправляться в базу данных, и в ней же данные будут возвращаться из БД. Это значение равно utf8mb4.

Затем эти пять значений используются для создания DSN и сохранения его в переменной $dsn. Здесь используются двойные кавычки, чтобы имена переменных заменялись их значениями. Синтаксис DSN очень строг: в нем не должно быть лишних пробелов или других символов.

- Префикс указывает на тип используемой базы данных. За ним следует двоеточие.
- Затем записаны четыре пары имя параметра/значение. Каждая пара разделяется точкой с запятой. За именем следует символ =, затем используемое для этого параметра значение (каждое из значений хранится в одной из пяти только что созданных переменных).

```
$type    = 'mysql';      // Тип базы данных
$server  = 'localhost';  // Имя хоста
$db      = 'phpbook-1';  // Имя базы данных
$port    = '8889';       // Для XAMPP использовать порт 3006
$charset = 'utf8mb4';    // Кодировка UTF-8 с 4 байтами данных
$dsn     = "$type:host=$server;dbname=$db;port=$port;charset=$charset";
```



После сохранения DSN в переменной вы можете создать объект PDO для соединения с базой данных.

Объекты PDO создаются с помощью встроенного в PHP класса PDO. Объекту PDO необходимо:

- DSN (показано на левой странице), чтобы найти базу данных, к которой он должен подключиться;
- имя и пароль учетной записи, позволяющие осуществлять вход в базу данных.

В приведенном ниже коде:
- переменная `$username` содержит имя пользователя учетной записи;
- переменная `$password` содержит пароль учетной записи.

При создании объекта PDO вы также можете задать параметры, определяющие, как он будет работать с базой данных. Ниже эти параметры хранятся в массиве с именем `$options`.

1. Параметр `PDO::ATTR_ERRMODE` определяет, как объект PDO поступает с возникающими ошибками. Если задать ему значение `PDO::ERRMODE_EXCEPTION`, то при возникновении ошибки базы данных PDO будет выбрасывать исключение с использованием встроенного класса `PDOException`. Этот параметр необходимо задавать для всех версий, предшествующих PHP 8 (в противном случае PDO не будет сообщать об ошибках), но в PHP 8 это стало режимом ошибок по умолчанию, и этот параметр можно не настраивать.
2. Параметр `PDO::ATTR_DEFAULT_FETCH_MODE` указывает PDO, в каком виде представлять каждую строку результирующего набора в коде PHP. Параметр `PDO::FETCH_ASSOC` указывает, что каждая строка результирующего набора будет возвращаться в виде ассоциативного массива.
3. Параметр `PDO::ATTR_EMULATE_PREPARES` включает/выключает то, что называется режимом эмуляции. В этой книге для него установлено значение `false`, чтобы числовые типы данных возвращались как есть, то есть `int` и `float`. Если же для этого параметра установить значение `true`, то любые значения из БД будут возвращаться как строки.

УЧЕТНАЯ ЗАПИСЬ ПОЛЬЗОВАТЕЛЯ БАЗЫ ДАННЫХ

```
$username = 'enter-your-username';
$password = 'enter-your-password';
$options  = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,    (1)
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,          (2)
    PDO::ATTR_EMULATE_PREPARES   => false,                     (3)
];
```

Как только DSN, данные учетной записи и параметры будут сохранены в переменных, вы можете создать объект PDO с помощью конструктора, как и любой другой объект. Объект PDO сохраняется в переменной с именем `$pdo`. Поскольку PDO — встроенный класс, определение класса в коде не требуется.

```
$pdo = new PDO($dsn, $username, $password, $options);
```

ПЕРЕМЕННАЯ · ИМЯ КЛАССА · DSN · ИМЯ ПОЛЬЗОВАТЕЛЯ · ПАРОЛЬ · ПАРАМЕТРЫ

Примечание: те пять значений, необходимых DSN, чтобы объект PDO подключился к базе данных, могут быть вписаны в DSN и напрямую, вместо того чтобы сначала быть присвоенными переменным. Но редактировать эти значения легче, если они содержатся в отдельных переменных, и меньше вероятность возникновения ошибок при изменении этих значений, поскольку у DSN очень строгий синтаксис.

КОД СОЕДИНЕНИЯ С БД МОЖЕТ НАХОДИТЬСЯ В ПОДКЛЮЧАЕМОМ ФАЙЛЕ

На веб-сайтах, использующих базу данных, подключение к ней требуется почти на каждой странице. Поэтому код для создания объекта PDO, который соединяется с базой данных, обычно помещают в подключаемый файл.

Подключаемый файл, показанный справа, создает объект PDO и присваивает его переменной с именем `$pdo`. Это позволяет любой странице, которая работает с БД, подключить этот файл и использовать переменную `$pdo`. Преимущества размещения этого кода в подключаемом файле:

- вам не нужно повторять один и тот же код на каждой странице, которая работает с базой данных;
- при необходимости изменить параметры подключения к базе данных их нужно будет поменять только в одном файле, а не на каждой странице, которая работает с БД;
- если у вас есть тестовая и рабочая версии сайта, то поменять параметры подключения к БД также нужно будет только в одном файле[55].

В коде, прилагаемом к этой книге, в разделе, посвященном этой главе, в папке `CMS/includes` находится файл `database-connection.php`. Он используется как в коде учебной CMS, так и в примерах для этой главы. Следовательно, вам необходимо отредактировать этот файл, чтобы использовать базу данных (с которой вы работали во введении к этому разделу и в предыдущей главе) в последующих примерах.
Чтобы проверить подключение к учебной БД, отредактируйте переменные в шагах 1 и 2, чтобы использовать значения для вашей базы данных. Затем попробуйте загрузить домашнюю страницу учебного сайта (это можно сделать как через навигацию с главной страницы загружаемого кода, так и набрав адрес прямо в браузере, http://localhost/phpbook/section_c/c12/cms/index.php). Если PDO сможет подключиться к базе данных, вы увидите сайт. Потратьте несколько минут на его изучение.
Если PDO не удалось подключиться к БД, функция обработки исключений может выдать сообщение об ошибке. Если у вас возникли проблемы с подключением к базе данных, см. примечание по устранению неполадок на странице справа.

1. Переменные с параметрами для DSN, показанные на предыдущей странице.
2. Переменные, которые содержат имя пользователя и пароль учетной записи, настроенной для доступа к базе данных. Это должны быть **именно те** значения, которые вы создали для своей базы данных.
3. Настройки PDO, предназначенные для того, чтобы:
- любые ошибки, с которыми сталкивается PDO, вызывали исключение;
- каждая строка данных возвращалась в виде ассоциативного массива;
- целые числа возвращались как тип данных integer (а не string).

4. Создание DSN с использованием данных, полученных в шаге 1.
5. Блок `try`, чтобы предотвратить отображение сведений об учетной записи пользователя (см. шаг 8).
6. При создании объекта PDO он подключается к базе данных. В случае успеха объект PDO присваивается переменной `$pdo`.
7. Если PDO не может подключиться к базе данных, он выбрасывает исключение, используя встроенный класс `PDOException`. Затем интерпретатор PHP выполняет код в блоке `catch`. Если выполняется блок `catch`, объект исключения сохраняется в переменной `$e`.
8. Исключение повторно генерируется в блоке `catch`. Это важно, поскольку

в случае ошибки при отсутствии обработчика исключений в сообщении об ошибке будут указаны имя пользователя и пароль базы данных. Этот метод предотвращает отображение имени пользователя и пароля.

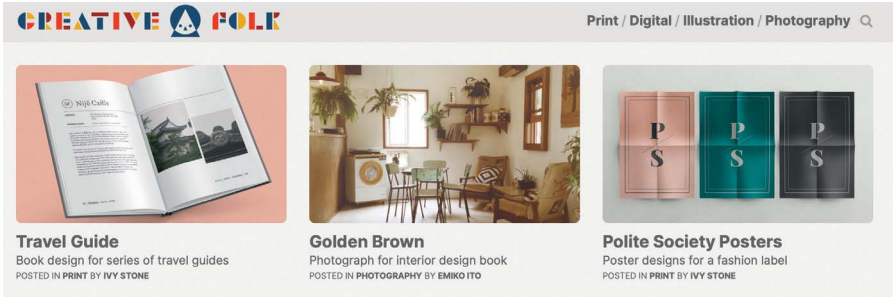
PHP section_c/c12/cms/includes/database-connection.php

```php
<?php
$type     = 'mysql';                // Тип базы данных
$server   = 'localhost';            // Сервер, на котором находится база данных
$db       = 'phpbook-1';            // Имя базы данных
$port     = '8889';                 // Обычно 8889 для MAMP и 3306 для XAMPP
$charset  = 'utf8mb4';              // Кодировка UTF-8 с 4 байтами данных на символ

$username = 'enter-your-username';  // Имя пользователя БД
$password = 'enter-your-password';  // Пароль пользователя БД

$options = [                        // Настройки PDO
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];
// НЕ МЕНЯЙТЕ НИЧЕГО ПОД ЭТОЙ СТРОКОЙ
$dsn = "$type:host=$server;dbname=$db;port=$port;charset=$charset"; // Создание DSN
try {                   // Использование конструкции Try, чтобы отловить исключение
    $pdo = new PDO($dsn, $username, $password, $options);   // Создание объекта PDO
} catch (PDOException $e) {                                 // Если выброшено исключение
    throw new PDOException($e->getMessage(), $e->getCode()); // Повторный выброс
                                                             // исключения
}
```

РЕЗУЛЬТАТ



Подключение к базе данных

В коде, прилагаемом к этой книге, для каждой оставшейся главы есть собственный файл с настройками подключения к базе данных. Чтобы выполнять примеры кода к остальным главам, вам необходимо сначала отредактировать такой файл в соответствующей папке прилагаемого кода.

Устранение неисправностей

Если у вас не получается подключиться к базе данных, замените шаг 8 следующей строкой (она загружает файл устранения неполадок):

```
include 'database-troubleshooting.php'
```

и прочитайте рекомендации по исправлению. После того как код заработает, замените строку в шаге 8 на исходную.

КАК ОДИН PHP-ФАЙЛ МОЖЕТ ОТОБРАЖАТЬ РАЗНЫЕ ДАННЫЕ

Веб-сайты, использующие базу данных, выполняют SQL-запросы для получения информации из БД, которую затем используют для отображения веб-страниц.

В предыдущей главе вы видели, что SQL-запросы могут запрашивать данные:

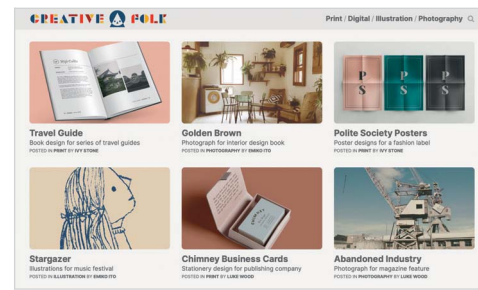
- об отдельном элементе данных (одна публикация или один пользователь);
- о наборе элементов (краткое описание последних публикаций или все публикации одного пользователя).

В коде учебного сайта файл `article.php` отвечает за отображение каждой публикации, используя данные, хранящиеся в БД. Информация о том, какую публикацию SQL-запрос должен получить из базы данных, передается через строку запроса в URL. Файл `article.php` использует PDO, чтобы выполнить запрос, и база данных создает результирующий набор из одной строки данных, который содержит информацию о публикации. Эти данные записываются в массив, а затем отображаются на странице.

Домашняя страница (`index.php`), показанная ниже, использует SQL-запрос для получения информации о последних шести добавленных на сайт работах. Когда на сайт добавляется новая публикация, она автоматически появляется на домашней странице и отображается на ней как самая новая.

Когда SQL запрашивает набор элементов, таких как последние шесть добавленных на сайт публикаций (или все публикации, написанные одним пользователем сайта), результирующий набор, создаваемый базой данных, может содержать несколько строк данных. Каждая строка данных в результирующем наборе будет представлять одну публикацию и может быть представлена в виде массива. Затем эти данные могут быть отображены на странице.





Один PHP-файл может использовать несколько SQL-запросов для получения необходимой информации из базы данных для создания страницы.

Страница категорий (`category.php`) может отображать подробную информацию о любой отдельной категории сайта. Сначала отображаются название и описание категории; за этим следуют карточки всех публикаций, которые находятся в указанной категории. При этом используется два SQL-запроса.

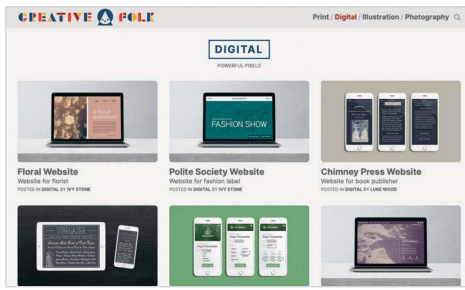
1. Первый SQL-запрос получает название и описание категории (этот запрос создаст результирующий набор с одной строкой данных).

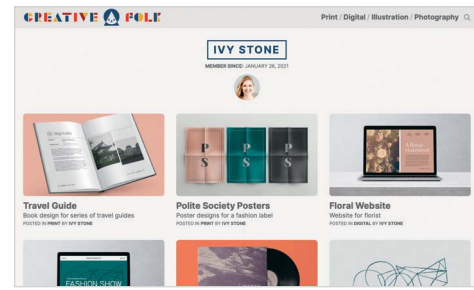
2. Второй получает все публикации в этой категории (этот запрос создаст результирующий набор, содержащий несколько строк данных — каждая из них представляет отдельную публикацию). Если в категорию будет добавлена новая публикация, она автоматически отобразится на странице.

Страница пользователя (`member.php`) используется для отображения персональной страницы каждого пользователя. Сначала отображается информация о нем самом (имя, изображение и дата регистрации), а затем карточки всех размещенных им публикаций. Опять же, на странице потребуется выполнить два запроса.

3. Первый получает имя пользователя, фотографию профиля и дату регистрации (это создаст результирующий набор с одной строкой данных).

4. Второй получает все размещенные им публикации (этот запрос создает результирующий набор, который может содержать несколько строк данных — каждая из них представляет отдельную публикацию). Если пользователь добавляет новую публикацию на сайт, она автоматически отображается на странице.





ПОЛУЧЕНИЕ ДАННЫХ С ПОМОЩЬЮ SQL-ЗАПРОСА

Для запросов SELECT база данных создает результирующий набор с запрошенными данными. Каждая строка в результирующем наборе может быть представлена в виде ассоциативного массива.

Приведенный ниже SQL-запрос возвращает имя и фамилию одного пользователя сайта. Он использует предложение WHERE для запроса данных о пользователе, идентификатор которого равен 1:

```
SELECT forename, surname
  FROM member
 WHERE id = 1;
```

Поскольку здесь запрашиваются данные об одном пользователе сайта, результирующий набор никогда не будет содержать более одной строки данных.

result set	
forename	surname
Ivy	Stone

Ниже показано, как строка данных из БД может быть представлена в виде ассоциативного массива. Имена столбцов в результирующем наборе используются в качестве ключей в массиве, а значения — это значения из соответствующих столбцов.

```
$member = [
    'forename' => 'Ivy',
    'surname'  => 'Stone',
];
```

Приведенный ниже SQL-запрос возвращает имя и фамилию каждого зарегистрированного пользователя сайта.

```
SELECT forename, surname
  FROM member;
```

Этот запрос создаст результирующий набор, содержащий несколько строк данных.

result set	
forename	surname
Ivy	Stone
Luke	Wood
Emiko	Ito

Если из PHP выполнить SQL-запрос, который создает результирующий набор с несколькими строками данных, то создается индексированный массив. Значение каждого элемента в этом индексированном массиве — ассоциативный массив, и каждый из этих ассоциативных массивов представляет одну строку данных из результирующего набора.

```
$members = [
    0 => ['forename' => 'Ivy',
          'surname'  => 'Stone',],
    1 => ['forename' => 'Luke',
          'surname'  => 'Wood',],
    2 => ['forename' => 'Emiko',
          'surname'  => 'Ito',],
];
```

Метод query( ) класса PDO выполняет SQL-запрос и создает объект PDOStatement для представления результирующего набора, созданного базой данных. Методы объекта PDOStatement позволяют получить данные из результирующего набора.

Метод query( ) объекта PDO принимает один параметр — SQL-запрос, который должна выполнить база данных.

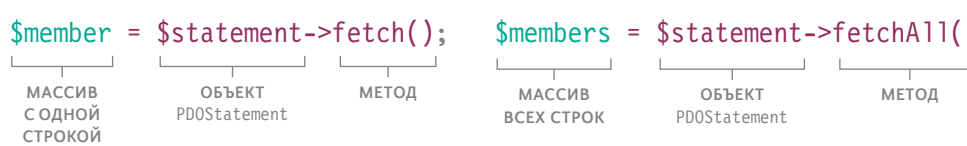
При вызове метода query( ) выполняется SQL-запрос и возвращается объект PDOStatement. Объект PDOStatement представляет результирующий набор, созданный базой данных для запроса.

Приведенный ниже SQL-запрос возвращает имя и фамилию каждого пользователя сайта. Когда выполняется приведенный ниже код PHP, объект PDO возвращает объект PDOStatement, который содержит ссылку на результирующий набор, содержащий имя и фамилию каждого пользователя сайта. Этот объект присваивается переменной с именем $statement.



Метод fetch( ) объекта PDOStatement используется для получения одной строки данных из результирующего набора. Согласно настройкам, которые мы использовали при создании объекта PDO, эта строка данных возвращается в виде ассоциативного массива, который записывается в переменную, чтобы остальной кода PHP на странице мог ее использовать.

Если запрос генерирует несколько строк данных, следует использовать метод fetchAll( ) объекта PDOStatement, который возвращает все данные из результирующего набора в виде индексированного массива. Каждый элемент этого массива содержит ассоциативный массив, представляющий одну строку данных в результирующем наборе.



Если запрос возвращает пустой набор записей, метод fetch( ) вернет значение false.

Если запрос возвращает пустой набор записей, метод fetchAll( ) вернет пустой массив.

ПОЛУЧЕНИЕ ОДНОЙ СТРОКИ ДАННЫХ ИЗ БД

В этом примере показано, как один PHP-файл может отображать данные о любом пользователе сайта.

1. Подключение файла `database-connection.php`. В нем создается объект PDO, обеспечивающий подключение к базе данных, и присваивается переменной с именем `$pdo`.
2. Подключение файла `functions.php`. В нем содержится определение функции `html_escape()`, которая использует `htmlspecialchars()` для замены зарезервированных символов в HTML сущностями, чтобы предотвратить XSS-атаки.
3. SQL-запрос записывается в переменную с именем `$sql`. Он запрашивает имя и фамилию пользователя, id которого равен 1.
4. Вызывается метод `query()` объекта PDO. Его единственный аргумент — переменная, содержащая SQL-запрос, который он должен выполнить. Метод `query()` выполняет SQL-запрос и возвращает объект `PDOStatement`, содержащий результирующий набор. Возвращаемый объект `PDOStatement` сохраняется в переменной `$statement`.
5. Метод `fetch()` объекта `PDOStatement` получает запрошенную строку данных и сохраняет ее в виде ассоциативного массива в переменной с именем `$member`.

section_c/c12/examples/query-one-row.php **PHP**

```
<?php
1 require '../cms/includes/database-connection.php';
2 require '../cms/includes/functions.php';
3 $sql       = "SELECT forename, surname
                  FROM member
                 WHERE id = 1;";
4 $statement = $pdo->query($sql);
5 $member    = $statement->fetch();
?>
<!DOCTYPE html>
<html> ...
  <body>
    <p>
6     <?= html_escape($member['forename']) ?>
7     <?= html_escape($member['surname']) ?>
    </p>
  </body>
</html>
```

РЕЗУЛЬТАТ

```
Ivy Stone
```

6. Имя пользователя выводится с помощью функции `html_escape()` для замены любых зарезервированных символов HTML на соответствующие им сущности.
7. Фамилия пользователя также выводится с помощью функции `html_escape()`.

Упражнение. В шаге 3 изменить условие в предложении SQL `WHERE`, чтобы получить строку таблицы, id которой равен 2. При этом будет отображен другой пользователь.

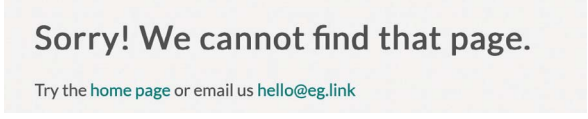
Затем измените код, чтобы запросить пользователя с id 4. Появится сообщение об ошибке, поскольку в базе данных нет такого пользователя.

ПРОВЕРКА, ВЕРНУЛ ЛИ ЗАПРОС ДАННЫЕ

PHP section_c/c12/examples/checking-for-data.php

```
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
$sql       = "SELECT forename, surname
                FROM member
(1)            WHERE id = 4;";
$statement = $pdo->query($sql);
(2) $member    = $statement->fetch();
(3) if (!$member) {
(4)     include 'page-not-found.php';
}
?>
<!DOCTYPE html>
<html> ...
  <body>
    <p>
      <?= html_escape($member['forename']) ?>
      <?= html_escape($member['surname']) ?>
    </p>
  </body>
</html>
```

РЕЗУЛЬТАТ



Примечание: файл `page-not-found.php` выполняет два дополнительных действия.

- Сначала он устанавливает код ответа HTTP равным 404.
- Затем, после сообщения о том, что страница не найдена, команда `exit` останавливает выполнение всего скрипта (не только этого файла, но и того, который его вызвал).

Если SQL-запрос не находит никаких данных, метод `fetch()` объекта `PDOStatement` вернет значение `false`.

Если после этого файл попытается отобразить полученные данные, интерпретатор PHP выдаст ошибку `Undefined index` («Индекс не определен»), поскольку переменная `$member` содержит значение `false`, а не массив.

Чтобы предотвратить эти ошибки, можно написать код, который проверит, были ли найдены какие-либо данные. Если данных нет, код может сообщить пользователю, что страница не найдена.

1. SQL-запрос ищет пользователя с `id`, равным 4. В таблице пользователей нет строки с идентификатором 4.
2. Вызывается метод `fetch()`. Он возвращает значение `false`, которое сохраняется в переменной `$member`.
3. Перед тем как отобразить данные, код с помощью конструкции `if` проверяет, содержит ли переменная `$member` значение `false`. Если да, то пользователь не найден.
4. Файл `page-not-found.php` подключается к коду страницы, чтобы сообщить посетителям, что страница не найдена.

Упражнение. В шаге 1 изменить значение `id` на 2.

ПОЛУЧЕНИЕ НЕСКОЛЬКИХ СТРОК ДАННЫХ ИЗ БД

В этом примере показано, как PHP-файл может получить и отобразить данные о каждом пользователе сайта. Если в БД будет добавлен новый пользователь, на странице автоматически отобразятся его данные.

1. Подключение двух файлов: `database-connection.php`, поскольку в нем создается объект PDO и выполняется подключение к базе данных; а также файла `functions.php`, поскольку в нем содержится определение функции `html_escape()`.
2. SQL-запрос записывается в переменную с именем `$sql`. Он запрашивает из БД имя и фамилию каждого пользователя.
3. Вызывается метод `query()` объекта PDO. Его параметр — SQL-запрос, который надо выполнить. Метод `query()` выполняет SQL-запрос и возвращает объект `PDOStatement`, содержащий результирующий набор, который записывается в переменную `$statement`.
4. Метод `fetchAll()` объекта `PDOStatement` возвращает все строки данных из результирующего набора в виде индексированного массива, который сохраняется в переменной `$members`. Каждый элемент этого массива представляет одну строку из результирующего набора в виде ассоциативного массива.

section_c/c12/examples/query-multiple-rows.php **PHP**

```php
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
$sql       = "SELECT forename, surname
                FROM member;";
$statement = $pdo->query($sql);
$members   = $statement->fetchAll();
?>
<!DOCTYPE html>
<html> ...
  <body>
    <?php foreach ($members as $member) { ?>
      <p>
        <?= html_escape($member['forename']) ?>
        <?= html_escape($member['surname']) ?>
      </p>
    <?php } ?>
  </body>
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

Luke Wood

Emiko Ito

5. Цикл `foreach` перебирает с элементы индексированного массива, лежащего в переменной `$members`. При каждой итерации цикла в переменную `$member` записывается ассоциативный массив, представляющий одну строку результирующего набора.
6. Вывод имени и фамилии пользователя.

Примечание. Если запрос не находит ни одной строки, метод `fetchAll()` возвращает пустой массив, поэтому код, расположенный внутри цикла, не выполняется (если бы он выполнялся, это вызвало бы ошибку несуществующего индекса — Undefined index).

ЦИКЛ ДЛЯ ИЗВЛЕЧЕНИЯ ОДНОЙ СТРОКИ ДАННЫХ ЗА ИТЕРАЦИЮ

PHP section_c/c12/examples/query-multiple-rows-while-loop.php

```php
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
$sql       = "SELECT forename, surname
              FROM member;";
$statement = $pdo->query($sql);
?>
<!DOCTYPE html>
<html> ...
  <body>
    <?php while ($row = $statement->fetch()) { ?>
      <p>
        <?= html_escape($row['forename']) ?>
        <?= html_escape($row['surname']) ?>
      </p>
    <?php } ?>
  </body>
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

Luke Wood

Emiko Ito

Примечание. Обычно веб-сайт не показывает слишком много информации на одной странице (он должен разбивать результаты на несколько страниц). Но если код должен работать с большими объемами данных (больше, чем может быть показано на обычной веб-странице), следует использовать именно этот подход, чтобы предотвратить слишком большой расход памяти, который будет вызван тем, что функция `fetchAll()` возвращает в виде массива сразу все строки, которые вернул запрос, и этот массив может потребовать очень много памяти интерпретатора PHP. В то время как вызов метода `fetch()` в цикле `while` возвращает только одну строку данных из результирующего набора за итерацию[56].

Также для получения нескольких строк можно использовать цикл `while`, с тем чтобы объект `PDOStatement` возвращал по одной строке данных из результирующего набора за итерацию, как показано в этом примере.

1. Подключение файлов `database-connection.php` и `functions.php`.
2. SQL-запрос записывается в переменную с именем `$sql`.
3. Метод `query()` объекта `PDO` вызывается для выполнения SQL-запроса и создания объекта `PDOStatement`, представляющего результирующий набор.
4. В условии цикла `while` вызывается метод `fetch()`. Он будет возвращать по одной строке данных из результирующего набора за одну итерацию. Массив, представляющий эту строку данных, сохраняется в переменной с именем `$row`, поэтому он может использоваться операторами внутри цикла.
Как только цикл выполняется один раз, он снова вызывает метод `fetch()`, который автоматически извлекает следующую строку данных из результирующего набора и сохраняет ее в переменной `$row`. Когда в результирующем наборе больше нет строк, метод `fetch()` возвращает значение `false` и цикл прекращается.
5. Внутри цикла выводится элемент массива, содержащий имя пользователя.

ИСПОЛЬЗОВАНИЕ В SQL-ЗАПРОСЕ ИЗМЕНЯЮЩИХСЯ ДАННЫХ

Каждый раз, когда запрашивается страница, она может использовать разные значения в SQL-запросе для получения разных данных из БД. В таких случаях SQL-запрос должен быть сначала **подготовлен (prepare)**, а затем **выполнен (execute)**.

ШАГ 1: ПОДГОТОВКА ЗАПРОСА

В SQL используются специальные маркеры, называемые **плейсхолдерами** (или их еще называют **параметрами** запроса), для представления значений, которые могут изменяться при каждом выполнении запроса.

Плейсхолдер в PDO работает как переменная, но его имя начинается с двоеточия, а не с символа $. Также в PDO можно использовать позиционные плейсхолдеры, они выглядят просто как знаки вопроса.

Если в SQL-запросе есть плейсхолдеры, то вместо вызова метода `query()` объекта `PDO` необходимо вызвать метод `prepare()`. Метод `prepare()` также возвращает объект `PDOStatement`, но не выполняет запрос (и, соответственно, объект еще не содержит результирующий набор, который появится после выполнения запроса на следующем шаге).

ШАГ 2: ВЫПОЛНЕНИЕ

Далее запрос выполняется путем вызова метода `execute()` объекта `PDOStatement`. Для него требуется один аргумент: массив, содержащий значения, используемые для замены плейсхолдеров.

Передаваемые значения могут быть предоставлены в виде ассоциативного массива, где:

- **ключ** — это имя плейсхолдера в SQL-запросе (перед ним может стоять двоеточие, но это не обязательно);
- **значение** — это значение, которое будет использовано в SQL-запросе на месте соответствующего плейсхолдера (часто оно представлено переменной).

ПЛЕЙСХОЛДЕР

```
$sql       = "SELECT forename, surname FROM member WHERE id = :id;";
$statement = $pdo->prepare($sql);
$statement->execute(['id' => $id]);
```

ИМЯ ПЛЕЙСХОЛДЕРА — ИСПОЛЬЗУЕМОЕ ЗНАЧЕНИЕ

Приведенный выше SQL-запрос содержит плейсхолдер с именем: `id`.
Метод `execute()` заменяет: `id` значением, хранящимся в переменной `$id`. Программисты называют это **подготовленным выражением (**или **параметризованным запросом)**.

Никогда не добавляйте переменные напрямую в SQL-запрос (например, как показано ниже). Это делает ваш код подверженным атаке, известной, как **внедрение SQL-кода (SQL-инъекция)**. Использование плейсхолдеров делает SQL неуязвимым для этого типа атак.

```
⊗ $sql = 'SELECT * FROM member WHERE id=' . $id;
⊗ $sql = 'SELECT * FROM member WHERE id=' . $_GET['id'];
```

ОТОБРАЖЕНИЕ РАЗНЫХ ДАННЫХ НА ОДНОЙ И ТОЙ ЖЕ СТРАНИЦЕ

PHP section_c/c12/examples/prepared-statement.php

```
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
$id        = 1;
$sql       = "SELECT forename, surname
                   FROM member
                  WHERE id = :id;";
$statement = $pdo->prepare($sql);
$statement->execute(['id' => $id]);
$member    = $statement->fetch();
if (!$member) {
    include 'page-not-found.php';
}
?>
<!DOCTYPE html>
<html> ...
  <body>
    <p>
      <?= html_escape($member['forename']) ?>
      <?= html_escape($member['surname']) ?>
    </p>
  </body>
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

Упражнение. В шаге 2 измените значение переменной `$id` на число 2. Затем сохраните и обновите страницу.

На странице будут отображаться данные другого пользователя. Если значение `$id` сделать равным 4, то будет выполнен файл `page-not-found.php`.

1. Подключение файлов `database-connection.php` и `functions.php`.
2. Переменной с именем `$id` присваивается целое число 1, это идентификатор строки, которую необходимо получить из базы данных.
3. SQL запрос записывается в переменную с именем `$sql`. Плейсхолдер: `id` стоит в том месте, на которое необходимо подставить значение переменной.
4. Вызывается метод `prepare()` объекта PDO, который возвращает объект `PDOStatement`, содержащий подготовленный к выполнению запрос. Этот объект сохраняется в переменной с именем `$statement`.
5. Метод `execute()` объекта `PDOStatement` вызывается для выполнения запроса и генерации результирующего набора. Его аргументом является массив, содержащий имя плейсхолдера и значение, на которое его следует заменить.
6. Метод `fetch()` объекта `PDOStatement` используется для получения строки данных из результирующего набора и сохранения ее в виде ассоциативного массива в переменной, называемой `$member`.
7. Если запрос не вернул данных, то посетитель получает сообщение, что страница не найдена.

ПРИВЯЗКА ЗНАЧЕНИЙ К SQL-ЗАПРОСУ

Методы `bindValue()` и `bindParam()` объекта `PDOStatement` предлагают еще один способ заменить плейсхолдеры в SQL-запросе.

Можно также использовать методы `bindValue()` и `bindParam()`, чтобы заменить плейсхолдеры в SQL-запросе. Они должны быть вызваны из объекта `PDOStatement`, полученного с помощью `prepare()`, и перед вызовом `execute()`. Оба они поддерживают три параметра:

- имя плейсхолдера;
- переменная, значение которой заменяет плейсхолдер;
- тип данных значения (это не требуется, если тип данных строковый).

В таблице ниже показаны значения, используемые в третьем параметре метода `bindValue()`. Они применяются для указания типа данных значения, подставляемого с помощью плейсхолдера в SQL-запрос.

ТИП ДАННЫХ	ЗНАЧЕНИЕ
String	`PDO::PARAM_STR`
Integer	`PDO::PARAM_INT`
Boolean	`PDO::PARAM_BOOL`

Разница между методами `bindValue()` и `bindParam()` заключается в том, когда именно интерпретатор PHP получает значение из переменной и использует его для замены плейсхолдера в SQL-запросе.

- При использовании метода `bindValue()` интерпретатор получает значение из переменной сразу при вызове `bindValue()`.
- При использовании метода `bindParam()` интерпретатор получает значение из переменной в момент вызова `execute()`. Следовательно, если значение, сохраненное в переменной, изменяется между вызовами методов `bindParam()` и `execute()`, используется обновленное значение.

Ниже `placeholder: id` в SQL-запросе будет заменен значением, содержащимся в переменной с именем `$id`.

В остальной части книги для привязки данных используется метод, показанный на предыдущей странице, поскольку вам не нужно указывать тип данных для каждого из значений и при этом требуется меньше кода.

ПЛЕЙСХОЛДЕР

```
$sql       = "SELECT * FROM member WHERE id = :id;";
$statement = connection->prepare($sql);
$statement->bindValue('id', $id, PDO::PARAM_INT);
```

ПЛЕЙСХОЛДЕР ПЕРЕМЕННАЯ ТИП ДАННЫХ

ПРИВЯЗКА ЦЕЛОГО ЧИСЛА К SQL-ЗАПРОСУ

PHP section_c/c12/examples/bind-value.php

```
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
$id        = 1;
$sql       = "SELECT forename, surname
                   FROM member
                  WHERE id = :id;";
$statement = $pdo->prepare($sql);
(1) $statement->bindValue('id', $id, PDO::PARAM_INT);
(2) $statement->execute();
$member    = $statement->fetch();
if (!$member) {
    include 'page-not-found.php';
}
?>
<!DOCTYPE html>
<html> ...
  <body>
    <p>
      <?= html_escape($member['forename']) ?>
      <?= html_ escape($member['surname']) ?>
    </p>
  </body>
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

Этот пример выглядит так же, как и предыдущий, но для замены плейсхолдера в SQL-запросе в нем используется метод `bindValue()` объекта `PDOStatement`.

1. С помощью метода `prepare()` объекта PDO создается объект `PDOStatement`, содержащий подготовленный запрос. Затем вызывается метод `bindValue()` этого объекта для замены плейсхолдера в SQL-запросе. Он использует три аргумента:

- имя плейсхолдера в SQL-запросе;
- имя переменной, значение которой необходимо использовать в запросе;
- тип `PDO::PARAM_INT`, чтобы указать, что значение — целое число.

2. Метод `execute()` объекта `PDOStatement` вызывается для выполнения запроса. Ему не нужны никакие аргументы, поскольку плейсхолдер в SQL-запросе уже заменен нужным значением.

Упражнение. В шаге 2 изменить значение, сохраненное в переменной `$id`, на число 2. Затем сохраните и обновите страницу. На странице должны отобразиться данные другого пользователя. Если значение переменной `$id` будет равно 4, то будет выполнен код из файла `page-not-found.php`.

ИСПОЛЬЗОВАНИЕ ОДНОГО ФАЙЛА ДЛЯ ОТОБРАЖЕНИЯ МНОЖЕСТВА СТРАНИЦ

Когда один PHP-файл используется для отображения многих страниц сайта, в конец URL-адреса можно добавить строку запроса[57], чтобы передать в PHP-скрипт информацию о том, какие данные ему необходимо получить из БД.

Код в файле PHP может прочитать значение из строки запроса, а затем использовать его в SQL-запросе, чтобы указать, какие данные следует получить из базы данных. Ниже приведена ссылка на файл `member.php`.

Она содержит строку запроса, содержащую элемент с именем `id` и значением 1. Это значение столбца `id` таблицы `member` в той строке, которую мы хотим получить.

```
<a href="member.php?id=1">Ivy Stone</a>
```

В коде страницы `member.php` используется функция PHP `filter_input()`, чтобы получить значение из строки запроса. Поскольку все значения столбцов-идентификаторов в базе данных — целые числа, функция использует фильтр для целых чисел. Возвращаемое значение сохраняется в переменной с именем `$id`:

- если значение будет целым числом, в `$id` будет целое число;
- если значение не будет целым числом, в `$id` будет значение `false`;
- если значение `id` отсутствует в строке запроса, в `$id` будет значение `null`.

Затем конструкция `if` проверяет, содержит ли переменная `$id` значение `false` (то есть было передано не целочисленное значение), `null` (значение не было передано вообще) или 0 (который не является подходящим значением для id в БД). Если это так, то страница заведомо не сможет отобразить данные пользователя и подключается файл `page-not-found.php`. Этот файл:

- отправляет код ответа HTTP со значением 404;
- сообщает посетителю, что страница не найдена;
- использует команду `exit`, чтобы остановить дальнейшее выполнение кода.

```
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
if (!$id) {
    include 'page-not-found.php';
}
```

Если же в строке запроса было передано допустимое значение, код на странице продолжает работу дальше, получая из БД данные и сохраняя их в переменной с именем `$member`. Затем вторая конструкция `if` проверяет, вернул ли запрос какие-либо данные. Если нет, снова будет подключен файл `page-not-found.php`, который остановит выполнение кода. Остальная часть страницы будет показана только в том случае, если данные пользователя были успешно получены из БД.

Чтобы отобразить данные другого пользователя, в строке запроса необходимо передать идентификатор его строки из таблицы `members` базы данных.

ИСПОЛЬЗОВАНИЕ СТРОКИ ЗАПРОСА ДЛЯ ОТОБРАЖЕНИЯ НУЖНОЙ СТРАНИЦЫ

PHP section_c/c12/examples/query-strings.php?id=1

```php
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';

$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
if (!$id) {                          // Если id отсутствует
    include 'page-not-found.php'; // Показать страницу 404
}

$sql       = "SELECT forename, surname
                FROM member
               WHERE id = :id;";        // SQL-запрос
$statement = $pdo->prepare($sql);       // Подготовка
$statement->execute([':id' => $id]);    // Выполнение
$member    = $statement->fetch();  // Получение данных

if (!$member) {               // Если данные отсутствуют
    include 'page-not-found.php'; // Показать страницу 404
}
?>
<!DOCTYPE html>
<html> ...
  <body>
    <p>
      <?= html_escape($member['forename']) ?>
      <?= html_escape($member['surname']) ?>
    </p>
  </body>
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

Упражнение. Изменить число в строке запроса на 4: будет показано, что страница не найдена.

Этот пример основан на предыдущих и использует строку запроса для передачи на страницу идентификатора отображаемого элемента.

1. Функция PHP `filter_input()` получает идентификатор пользователя из строки запроса и записывает результат в переменную `$id`. Если было передано целое число, то оно и будет записано, а если нет — то значение `false`. Если данные отсутствуют, переменная получит значение `null`.
2. Конструкция `if` проверяет, содержит ли `$id` пустое значение (0, `false` или `null`).
3. Если да, будет подключен файл `page-not-found.php`, поскольку в строке запроса нет подходящего идентификатора пользователя, и выполнение кода будет остановлено.
4. Если же код продолжает выполнение, это значит, что в строке запроса было передано допустимое значение и теперь можно выполнить SQL-запрос, чтобы получить из БД строку с таким идентификатором.
5. Следующая конструкция `if` проверяет, содержит ли переменная `$member` значение `false`.
6. Если да, то пользователь не был найден, и подключается файл `page_not_found.php`.
7. В противном случае данные пользователя используются для создания HTML-страницы.

ОТОБРАЖЕНИЕ ИНФОРМАЦИИ ИЗ БАЗЫ ДАННЫХ В HTML

Сначала получите данные из базы и сохраните их в переменных. Затем используйте эти переменные для создания HTML-страницы.

Чтобы сделать ваш код более удобным для чтения и сопровождения, четко разделяйте в своих файлах PHP:

- код, получающий данные из БД;
- код, генерирующий HTML-страницу.

На странице справа это разделение показано пунктирной линией. Часть файла, предназначенная для вывода HTML, должна содержать как можно меньше PHP-кода. Ниже представлены три типа операций, которые требуются чаще всего.

ФУНКЦИИ

Функции часто используются для форматирования данных. Функция `html_escape()` уже использовалась на многих страницах для предотвращения XSS-атаки путем замены любых зарезервированных символов HTML-сущностями.

Справа вы можете увидеть еще одну функцию, находящуюся в файле `functions.php` для этой главы. Он используется вывода даты в едином удобочитаемом формате.

Сначала дата и время, хранящиеся в базе данных, преобразуются в метку времени Unix с помощью функции PHP `strtotime()`. Затем данные преобразуются в удобочитаемый формат[58] с помощью функции PHP `date()`.

```
function format_date(string $string): string
{
    $date = strtotime($string);
    return date('F d, Y', $date);
}
```

УСЛОВНЫЕ КОНСТРУКЦИИ

Конструкция `if` или ее варианты могут использоваться для того, чтобы решить, как именно отображать HTML-код. Например, если пользователь не загрузил изображение профиля, база данных вернет `NULL` вместо имени файла, и тогда надо будет показать заглушку.

Изображение профиля может быть показано с помощью оператора объединения с `null`. Если изображение было загружено, оно будет выведено, а если нет, то вместо него будет показана заглушка.

```
html_escape($member['picture'] ?? 'blank.png');
```

ЦИКЛЫ

Как уже было показано в нескольких примерах, цикл используется для обработки каждой строки результирующего набора, возвращенного базой данных. Справа вы можете видеть, как цикл `foreach` используется для повторения одних и тех же инструкций для каждого пользователя сайта, полученного из базы данных.

ОФОРМЛЕНИЕ ВЫВОДА ДАННЫХ С ИСПОЛЬЗОВАНИЕМ HTML

PHP section_c/c12/examples/formatting-data-in-html.php

```php
<?php
require '../cms/includes/database-connection.php';              // Создание объекта PDO
require '../cms/includes/functions.php';                        // Функции
$sql       = "SELECT id, forename, surname, joined, picture FROM member;"; // SQL
$statement = $pdo->query($sql);                                 // Выполнение запроса
$members   = $statement->fetchAll();                            // Получение данных
?>
<!DOCTYPE html> ...
<body> ...
<?php foreach ($members as $member) { ?>
  <div class="member-summary">
    <img src="../cms/uploads/<?= html_escape($member['picture'] ?? 'blank.png') ?>"
         alt="<?= html_escape($member['forename']) ?>" class="profile">
    <h2><?= html_escape($member['forename'] . ' ' . $member['surname']) ?></h2>
    <p>Member since:<br><?= format_date($member['joined']) ?></p>
  </div>
<?php } ?> ...
</body>
```

РЕЗУЛЬТАТ



1. Код получает данные всех пользователей сайта.

2. Цикл `foreach` повторяет один и тот же набор инструкций для каждого пользователя.

3. Оператор объединения с `null` проверяет, была ли загружена фотография профиля. Если была, то в теге `<img>` выводится имя это файла. Если нет, то вместо него выводится изображение-заглушка с именем `blank.png`.

4. Дата регистрации пользователя форматируется с помощью функции `format_date()`.

Цикл повторяет одни и те же инструкции для каждого пользователя сайта.

ФУНКЦИЯ ДЛЯ ВЫПОЛНЕНИЯ SQL-ЗАПРОСОВ

Чтобы с помощью PDO выполнить SQL-запрос и вернуть результирующий набор, требуется выполнить две или три инструкции. Написание пользовательской функции позволяет сделать это с помощью одной.

Если в SQL-запросе не используются плейсхолдеры, метод `query()` объекта PDO выполнит запрос и вернет объект `PDOStatement`, представляющий результирующий набор:

```
$statement = $pdo->query($sql);
```

Если в SQL-запросе используются плейсхолдеры, необходимо сначала вызвать метод `prepare()` объекта PDO для создания объекта `PDOStatement`, содержащего подготовленный запрос, а затем выполнить его, вызвав метод `execute()` этого объекта:

```
$statement = $pdo->prepare($sql);
$statement->execute($sql);
```

После этих шагов один из методов объекта `PDOStatement` может быть использован для получения данных из результирующего набора.

Приведенная ниже функция содержит три параметра:

- `$pdo` для объекта `PDO`, который обеспечивает подключение к базе данных;
- `$sql` для SQL-запроса, который должен быть выполнен;
- `$arguments` — это массив с данными для замены плейсхолдеров в запросе (обратите внимание: этот аргумент является необязательным, и если он не указан, то по умолчанию его значение будет равно `null`).

Функция проверяет, передан ли массив с плейсхолдерами:

- Если его нет, функция выполнит метод `query()` и вернет полученный в результате объект `PDOStatement`.
- Если массив передан, то сначала будет вызван метод `prepare()` для создания объекта `PDOStatement`, затем метод `execute()` этого объекта и после этого функция вернет этот же объект.

```
function pdo(PDO $pdo, string $sql, array $arguments = null)
{
    if (!$arguments) {
        return $pdo->query($sql);
    }
    $statement = $pdo->prepare($sql);
    $statement->execute($arguments);
    return $statement;
}
```

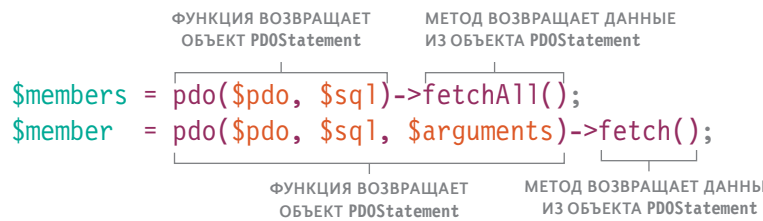
При вызове пользовательской функции pdo( ) выполнить запрос и получить данные можно в одной строчке благодаря **вызову методов по цепочке (method chaining)**.

После создания объекта PDOStatement и выполнения SQL-запроса, можно вызвать один из следующих трех методов для получения данных из результирующего набора:

- fetch( ) получает одну строку данных;
- fetchAll( ) получает все строки набора;
- fetchColumn( ) получает значение из одной ячейки (одного столбца одной строки).

Если функция или метод возвращает объект, то можно использовать **вызов методов по цепочке,** то есть сразу вызвать метод объекта, который вернула функция.

Ниже, когда вызывается функция pdo( ), она возвращает объект PDOStatement. За вызовом функции сразу же следует объектный оператор и вызов метода возвращаемого функцией объекта PDOStatement.



Определение функции pdo( )добавлено в подключаемый файл functions.php для этой главы. Он будет использоваться как в этой главе, так и в следующей.

В главе 14 вы познакомитесь с другим подходом, когда для работы с БД вместо функции будут использоваться пользовательские классы.

Передавая массив в метод execute( ) объекта PDOStatement, вы можете использовать либо ассоциативный массив, в котором ключи соответствуют именам плейсхолдеров в SQL-запросе, как было показано ранее; либо индексированный массив, но в этом случае значения в массиве должны идти ровно в том порядке, в каком идут плейсхолдеры в SQL- запросе.

ПРИМЕР ИСПОЛЬЗОВАНИЯ ФУНКЦИИ PDO() БЕЗ ПАРАМЕТРОВ

В этом примере показано, как функция `pdo()`, определенная на предыдущей странице, получает данные из БД, когда SQL-запрос не содержит плейсхолдеров.

1. Подключение файла `database-connection.php`. В нем создается объект `PDO`, который присваивается переменной `$pdo`.
2. Подключение файла `functions.php`. Он содержит определение функции `pdo()`, которое показано на предыдущей странице.
3. SQL-запрос для получения имени и фамилии каждого пользователя записывается в переменную с именем `$sql`.
4. Функция `pdo()` вызывается с двумя аргументами:

- созданный в шаге 1 объект `PDO`;
- сохраненный в переменной `$sql` в шаге 3 SQL-запрос.

Функция возвращает объект `PDOStatement`, и его метод `fetchAll()` вызывается в той же инструкции, чтобы сразу получить все данные из результирующего набора. Эти данные будут сохранены в виде массива в переменной `$members`.

5. С помощью цикла `foreach` выводятся данные, лежащие в массиве `$members`.

section_c/c12/examples/pdo-function-no-parameters.php **PHP**

```php
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
$sql = "SELECT forename, surname
          FROM member;";
$members = pdo($pdo, $sql)->fetchAll();
?>
<!DOCTYPE html>
<html> ...
  <body>
    <?php foreach ($members as $member) { ?>
      <p>
        <?= html_escape($member['forename']) ?>
        <?= html_escape($member['surname']) ?>
      </p>
    <?php } ?>
  </body>
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

Luke Wood

Emiko Ito

Упражнение. В шаге 3 отредактируйте SQL-запрос, чтобы получить адреса электронной почты пользователей и их имена. В шаге 5 выведите адрес электронной почты после имени.

ПРИМЕР ИСПОЛЬЗОВАНИЯ ФУНКЦИИ PDO() С ПАРАМЕТРАМИ В ЗАПРОСЕ

PHP section_c/c12/examples/pdo-function-with-parameters.php?id=1

```
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
(1) $id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
if (!$id) {
    include 'page-not-found.php';
}

$sql = "SELECT forename, surname
          FROM member
         WHERE id = :id;";
(2)
(3) $member = pdo($pdo, $sql, ['id' => $id])->fetch();

if (!$member) {
    include 'page-not-found.php';
}
?> ...
    <p>
(4)     <?= html_escape($member['forename']) ?>
        <?= html_escape($member['surname']) ?>
    </p>
```

RESULT

```
Ivy Stone
```

Упражнение. В шаге 3 заменить ассоциативный массив, используемый в качестве третьего параметра, на индексированный с одной переменной `$id` в квадратных скобках:

```
$member = pdo($pdo, $sql, [$id]);
```

В этом коде третий параметр является индексированным массивом, а не ассоциативным. Если используется этот метод, значения в массиве должны располагаться в том же порядке, в котором плейсхолдеры располагаются в SQL-запросе[59].

В этом примере показано, как функция `pdo()` применяется для выполнения запроса, когда в нем используются плейсхолдеры.

1. Из строки запроса получаем идентификатор пользователя, информацию о котором необходимо запросить из БД.
2. SQL запрашивает из БД имя и фамилию пользователя, идентификатор которого представлен плейсхолдером с именем: `id`.
3. Функция `pdo()` вызывается с тремя аргументами:

- созданный в файле `database-connection.php` объект `PDO`;
- сохраненный в переменной `$sql` в шаге 2 SQL-запрос;
- ассоциативный массив, содержащий имя плейсхолдера и значение, которое он должен использовать.

Примечание: массив создается *прямо в* аргументе функции, а не сохраняется заранее в переменной: `['id' => $id]`.
Функция возвращает объект `PDOStatement`. Его метод `fetch()` затем используется в той же строке для получения одной строки данных, возвращаемой SQL-запросом. Эти данные записываются в переменную `$member`.

4. Данные из переменной `$member` выводятся на страницу.

КАК НЕСКОЛЬКО ФАЙЛОВ PHP МОГУТ ОБЕСПЕЧИТЬ РАБОТУ ЦЕЛОГО САЙТА

На следующих двенадцати страницах показано, как четыре PHP-файла используются для отображения более чем пятидесяти страниц учебного веб-сайта.

Каждый из этих четырех PHP-файлов разделен на две части:

- сначала расположен код, получающий данные из БД и записывающий их в переменные;
- затем данные из этих переменных используются для создания HTML-страниц, отправляемых в браузер.

Эти файлы можно представить в качестве шаблонов.

Каждый раз, когда запрашивается один из файлов, он может получать различные данные из БД, вставлять их в соответствующую часть HTML-страницы, а затем отправлять HTML-код в браузер посетителя.

Каждая страница использует четыре подключаемых файла:

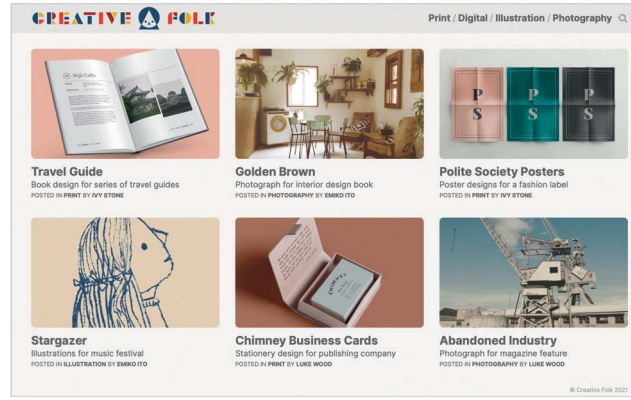
- `database-connection.php` создает объект PDO, который обеспечивает подключение к базе данных;
- `functions.php` содержит функции для получения и форматирования данных из БД;
- `header.php` содержит заголовок сайта, выводимый на каждой странице, и обеспечивает навигацию;
- `footer.php` содержит футер сайта. Также выводится на каждой странице.

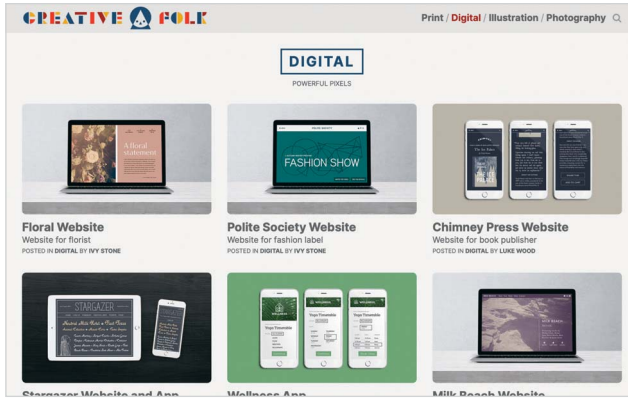
ФАЙЛ INDEX.PHP

На главной странице выводятся карточки последних шести размещенных на сайте публикаций.

Как только новая публикация сохраняется в базе данных, она автоматически появляется на этой странице (а самая старая пропадает с нее).

Структура HTML-страницы неизменна, но содержимое, отображаемое на ней, может меняться.

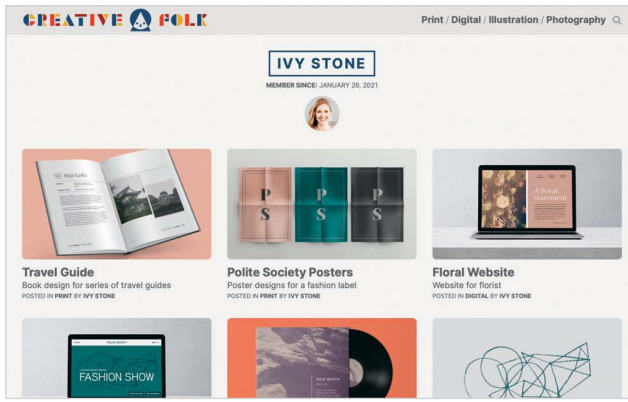




ФАЙЛ CATEGORY.PHP

Файл, отвечающий за вывод категорий, отображает название и описание какой-либо категории, за которыми следуют карточки публикаций, относящихся к этой категории. Структура HTML-страницы всегда одна и та же, но отображаемые на ней данные могут изменяться.

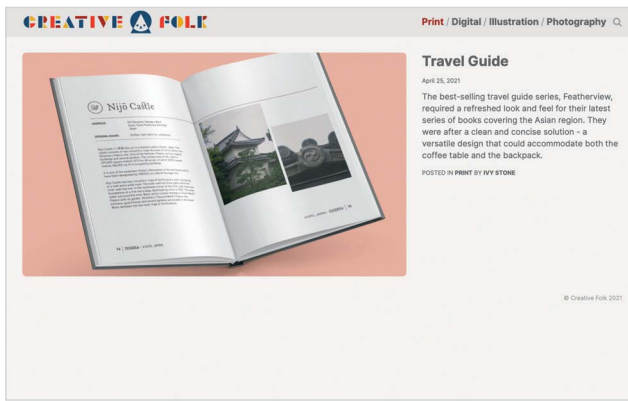
Строка запроса в URL содержит идентификатор категории, информация о которой должна быть получена из базы данных и показана на странице. Например, `category.php?id=1`.



ФАЙЛ MEMBER.PHP

Файл пользователей служит для отображения пользовательских профилей. За профилем следует список публикаций этого пользователя.

Строка запроса в URL содержит идентификатор пользователя, информация о котором должна быть получена из базы данных и показана на странице. Например, `member.php?id=1`.



ФАЙЛ ARTICLE.PHP

Файл публикаций отображает подробную информацию о выбранной публикации. Изображение, категория, название, дата, содержание и имя автора находятся в одном и том же месте для каждой публикации, но отображаемые данные могут измениться.

Строка запроса в URL содержит идентификатор публикации, который должен быть получен из базы данных и показан на странице. Например, `article.php?id=1`.

ФАЙЛЫ, ВЫВОДЯЩИЕ ВЕРХНЮЮ И НИЖНЮЮ ЧАСТЬ ОФОРМЛЕНИЯ САЙТА

Заголовок (хидер) для каждой страницы на сайте всегда один и тот же. Поэтому, вместо того чтобы повторять этот код в каждом скрипте, его помещают в подключаемый файл `header.php`.

Прежде чем этот файл будет подключен в код страницы, необходимо присвоить значения четырем переменным, которые выводятся в этом файле. Поэтому давайте познакомимся с ним поближе.

Первые две переменные выводятся в тегах `<title>` и `<meta>`.

Вторые две используются для создания панели навигации. Одна из них содержит массив выводимых категорий, а другая используется для выделения категории, которую посетитель просматривает в данный момент.

ПЕРЕМЕННАЯ	ЗНАЧЕНИЕ
`$title`	Значение для отображения в HTML-теге `<title>` страницы
`$description`	Значение для отображения в HTML-теге `<meta>` с атрибутом description
`$navigation`	Массив, содержащий имена и идентификаторы категорий, отображаемых в панели навигации
`$section`	Если файл подключается на странице категории, то на ней есть идентификатор просматриваемой категории. Если файл подключается на странице публикации, то на ней также содержится идентификатор категории, к которой относится эта публикация. Эти значения позволяют выделить активную категорию на панели навигации. Для других страниц эта переменная содержит пустую строку

1. Содержимое переменной `$title` выводится внутри элемента `<title>`.

2. Содержимое переменной `$description` выводится внутри тега `<meta>` с атрибутом `description`.

3. Цикл `foreach` перебирает каждую категорию в массиве `$navigation`.

При каждой итерации цикла идентификатор и название одной категории сохраняются в виде ассоциативного массива в переменной с именем `$link`.

4. Создается ссылка на файл `category.php`. В строке запроса передается идентификатор категории, которую он должен отобразить.

5. С помощью тернарного оператора определяется, следует ли выделить эту категорию.

В условии проверяется, совпадает ли значение в переменной `$section` с идентификатором категории, выводимой в цикле в данный момент. Если значения совпадают, к ссылке добавляются атрибуты `class="on"` и `aria-current="page"`, чтобы выделить текущую категорию.

6. В ссылке выводится название категории.

7. После ссылок на страницы категорий выводится ссылка на страницу поиска.

8. Файл `footer.php` содержит только одну инструкцию PHP, чтобы показать текущий год после уведомления об авторских правах.

9. `site.js` содержит код JavaScript для универсального отображения навигации сайта на любых устройствах.

PHP section_c/c12/cms/includes/header.php

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
(1) <title><?= html_escape($title) ?></title>
(2) <meta name="description" content="<?= html_escape($description) ?>">
    <link rel="stylesheet" type="text/css" href="css/styles.css">
    <link rel="preconnect" href="https://fonts.gstatic.com">
    <link rel="stylesheet"
      href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap">
    <link rel="shortcut icon" type="image/png" href="img/favicon.ico">
  </head>
  <body>
    <header>
      <div class="container">
        <a class="skip-link" href="#content">Skip to content</a>
        <div class="logo">
          <a href="index.php"><img src="img/logo.png" alt="Creative Folk"></a>
        </div>
        <nav role="navigation">
          <button id="toggle-navigation" aria-expanded="false">
            <span class="icon-menu"></span><span class="hidden">Menu</span>
          </button>
          <ul id="menu">
(3)         <?php foreach ($navigation as $link) { ?>
(4)         <li><a href="category.php?id=<?= $link['id'] ?>"
(5)           <?= ($section == $link['id'] ) ? 'class="on" aria-current="page"' : '' ?>>
(6)           <?= html_escape($link['name']) ?>
            </a></li>
            <?php } ?>
            <li><a href="search.php">
(7)           <span class="icon-search"></span><span class="search-text">Search</span>
            </a></li>
          </ul>
        </nav>
      </div><!-- /.container -->
    </header>
```

PHP section_c/c12/cms/includes/footer.php

```
(8)   <footer><div class="container">&copy; Creative Folk <?= date('Y');?></div></footer>
  </body>
(9) <script src="js/site.js"></script>
</html>
```

ГЛАВНАЯ СТРАНИЦА САЙТА

Главная (домашняя) страница (index.php) показывает карточки с кратким описанием шести последних публикаций, размещенных на сайте. Код на странице начинается с получения необходимых данных, которые присваиваются переменным.

1. Включение строгой типизации, чтобы функции принимали аргументы только разрешенного типа.
2. Подключение файла database-connection.php, в котором создается объект класса PDO, обеспечивающий подключение к базе данных. Он присваивается переменной под названием $pdo.
3. Подключение файла functions.php, в котором содержатся определения для функции pdo() и функций для форматирования выводимых данных.
4. Переменной $sql присваивается строка с SQL-запросом для получения информации о самых новых публикациях.
5. Выполнение запроса с помощью функции pdo(), которая возвращает объект PDOStatement, представляющий результирующий набор. Затем метод fetchAll() этого объекта возвращает все данные о полученных публикациях в виде массива, который записывается в переменную $articles.
Следующие пять шагов позволяют получить данные, использующиеся в header.php, и записать их в переменные.
6. SQL для получения идентификаторов и названий категорий, отображаемых в панели навигации, пишется в переменную $sql.
7. Выполнение запроса и запись полученных данных в переменную $navigation.
8. Если пользователь находится на странице категории или публикации, переменная $section содержит идентификатор текущего раздела. Домашняя страница не относится ни к какой категории, поэтому на ней переменная содержит пустую строку.
9. Переменной $title присваивается строка с текстом для элемента <title>.
10. Переменной $description присваивается строка с текстом, выводимым в теге <meta> с атрибутом description.

Остальная часть файла, под пунктирной линией, использует PHP-код только для отображения содержащихся в переменных данных. Это отделяет PHP-код, получающий данные, от HTML-кода, который выводится в браузер.

11. Подключение файла header.php, показанного на предыдущей странице. Он использует данные, которые были получены и сохранены в переменных на этапах 6–10.
12. Цикл foreach перебирает элементы массива $articles, созданного в шаге 5, и отображает карточки публикаций. При каждой итерации цикла данные очередной публикации присваиваются переменной $article.
13. Создание ссылки на страницу article.php, которая выводит подробную информацию о публикации. В строку запроса передается идентификатор публикации, которую необходимо отобразить.
14. Вывод изображения для публикации. Если оно не было загружено, то отображается изображение-заглушка.
15. Если есть альтернативный текст для изображения, то он выводится в атрибуте alt. Если нет, значение атрибута будет пустым.
16. Заголовок публикации отображается в элементе <h2>.
17. Вывод аннотации.
18. Создание ссылки на файл category.php. Идентификатор категории добавляется в строку запроса. Название категории отображается внутри ссылки.
19. Создание ссылки на файл member.php. Идентификатор автора публикации добавляется в строку запроса. В качестве текста ссылки используется имя пользователя, разместившего публикацию.
20. Подключение файла footer.php (показан на предыдущей странице).

PHP section_c/c12/cms/index.php

```php
<?php
declare(strict_types = 1);                                  // Включение строгой типизации
require 'includes/database-connection.php';                 // Создание объекта PDO
require 'includes/functions.php';                           // Подключение функций

$sql = "SELECT a.id, a.title, a.summary, a.category_id, a.member_id,
               c.name AS category,
               CONCAT(m.forename, ' ', m.surname) AS author,
               i.file     AS image_file,
               i.alt      AS image_alt
          FROM article    AS a
          JOIN category   AS c ON a.category_id = c.id
          JOIN member     AS m ON a.member_id   = m.id
          LEFT JOIN image AS i ON a.image_id    = i.id
         WHERE a.published = 1
      ORDER BY a.id DESC
         LIMIT 6;";                          // SQL для получения 6 последних публикаций
$articles = pdo($pdo, $sql)->fetchAll();                    // Получение данных в массив

$sql = "SELECT id, name FROM category WHERE navigation = 1;"; // Запрос категорий
$navigation  = pdo($pdo, $sql)->fetchAll();                 // Навигация по категориям

$section     = '';                                          // Текущая категория
$title       = 'Creative Folk';                             // Для HTML-тега <title>
$description = 'A collective of creatives for hire';        // Для тега с описанием
?>
<?php include 'includes/header.php'; ?>
  <main class="container grid" id="content">
    <?php foreach ($articles as $article) { ?>
      <article class="summary">
        <a href="article.php?id=<?= $article['id'] ?>">
          <img src="uploads/<?= html_escape($article['image_file'] ?? 'blank.png') ?>"
               alt="<?= html_escape($article['image_alt']) ?>">
          <h2><?= html_escape($article['title']) ?></h2>
          <p><?= html_escape($article['summary']) ?></p>
        </a>
        <p class="credit">
          Posted in <a href="category.php?id=<?= $article['category_id'] ?>">
          <?= html_escape($article['category']) ?></a>
          by <a href="member.php?id=<?= $article['member_id'] ?>">
          <?= html_escape($article['author']) ?></a>
        </p>
      </article>
    <?php } ?>
  </main>
<?php include 'includes/footer.php'; ?>
```

СТРАНИЦА КАТЕГОРИИ

Файл `category.php` отображает название и описание выбранной категории, а также выводит карточки публикаций, принадлежащих этой категории.

1. Включение строгой типизации, чтобы функции принимали аргументы только разрешенного типа.
2. Подключение файлов `database-connection.php` и `functions.php`.
3. Функция `filter_input()` ищет в строке запроса параметр с именем `id` и проверяет, является ли его значение целым числом. Если это так, в переменную `$id` записывается это значение, а если нет, то `false`. Если значения не было в строке запроса, то в переменную записывается значение `null`.
4. Если в строке запроса не было допустимого целого числа, подключается файл `page-not-found.php`. Он заканчивается командой `exit`, останавливающей выполнение любого другого кода.
5. SQL-запрос для получения идентификатора, имени и описания выбранной категории записывается в переменную `$sql`.
6. Функция `pdo()` используется для выполнения SQL-запроса, а затем метод `fetch()` объекта `PDOStatement` получает данные и сохраняет их в переменной `$category`.
7. Если категория не была найдена в базе данных, подключается файл `page-not-found.php`, и код останавливается.
8. Если код все еще выполняется, в переменную `$sql` записывается SQL-запрос для получения публикаций в выбранной категории.
9. Выполняется запрос, затем метод `fetchAll()` объекта `PDOStatement` получает данные в виде массива, который сохраняется в переменной `$articles`.
10. SQL для получения идентификаторов и названий категорий, отображаемых в навигации, записывается в переменную `$sql`.
11. Запрос выполняется, метод `fetchAll()` получает возвращаемые данные, и они сохраняются в переменной `$navigation`.
12. Идентификатор категории присваивается переменной `$section`. Она используется для выделения этой категории в панели навигации.
13. Название категории записывается в переменную `$title`. Она будет выводиться в элементе `<title>`.
14. Описание категории записывается в переменную `$description` для использования в теге `<meta>`.

Поскольку все необходимые для отображения страницы данные сохранены в переменных, остальная часть файла под пунктирной линией занимается генерацией HTML-кода, выводимого в браузер.

15. Подключение файла `header.php`.
16. Название категории выводится в элементе `<h1>`.
17. Описание категории выводится под ее названием в элементе `<p>`.
18. Цикл `foreach` перебирает массив, содержащий все публикации в категории. Он был сохранен в переменной `$articles` в шаге 9. При каждой итерации цикла информация об очередной публикации сохраняется в переменной с именем `$article`.
19. Код для вывода карточек публикации тот же самый, который использовался на домашней странице (см. предыдущую страницу).

Если в категории отсутствуют публикации, то в переменной `$articles` будет пустой массив, поскольку метод `fetchAll()` объекта `PDOStatement` возвращает пустой массив, когда нет данных, соответствующих SQL-запросу.

Когда в цикл `foreach` передается пустой массив, код внутри цикла выполняться не будет. Это означает, что ошибка `Undefined index` возникать *не будет*, если запрос не вернул никаких данных.

20. Подключение файла `footer.php`.

PHP section_c/c12/cms/category.php?id=1

```php
<?php
declare(strict_types = 1);                                  // Включение строгой типизации
require 'includes/database-connection.php';                 // Создание объекта PDO
require 'includes/functions.php';                           // Подключение функций
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Валидация идентификатора
if (!$id) {                                                 // Если нет валидного ID
    include 'page-not-found.php';                           // Показать страницу 404
}
$sql = "SELECT id, name, description FROM category WHERE id=:id;"; // SQL
$category = pdo($pdo, $sql, [$id])->fetch();                // Получить данные о категории
if (!$category) {                                           // Если категория не найдена
    include 'page-not-found.php';                           // Показать страницу 404
}

$sql = "SELECT a.id, a.title, a.summary, a.category_id, a.member_id,
               c.name AS category,
               CONCAT(m.forename, ' ', m.surname) AS author,
               i.file AS image_file,
               i.alt  AS image_alt
          FROM article    AS a
          JOIN category   AS c   ON a.category_id  = c.id
          JOIN member     AS m   ON a.member_id    = m.id
          LEFT JOIN image AS i   ON a.image_id     = i.id
         WHERE a.category_id = :id AND a.published = 1
         ORDER BY a.id DESC;";                                  // SQL
$articles = pdo($pdo, $sql, [$id])->fetchAll();                 // Получение публикаций

$sql = "SELECT id, name FROM category WHERE navigation = 1;"; // SQL для категорий
$navigation  = pdo($pdo, $sql)->fetchAll();                   // Навигация по категориям
$section     = $category['id'];                               // Текущая категория
$title       = $category['name'];                             // Для HTML-тега <title>
$description = $category['description'];                      // Для тега <meta>
?>
<?php include 'includes/header.php'; ?>
<main class="container" id="content">
  <section class="header">
    <h1><?= html_escape($category['name']) ?></h1>
    <p><?= html_escape($category['description']) ?></p>
  </section>
  <section class="grid">
  <?php foreach ($articles as $article) { ?>
  <!-- Код для вывода публикаций такой же, как было показано ранее -->
  <?php } ?>
  </section>
</main>
<?php include 'includes/footer.php'; ?>
```

СТРАНИЦА ПУБЛИКАЦИИ

Файл `article.php` используется для отображения полной информации о каждой отдельной публикации на сайте. В строке запроса передается идентификатор публикации, которая должна быть отображена.

Как и в случае с другими страницами, этот файл начинается с получения данных из БД и сохранения их в переменных.

1. Включение строгой типизации подключения необходимых файлов, `database-connection.php` и `functions.php`.
2. Функция `filter_input()` ищет в строке запроса параметр с именем `id` и проверяет, является ли его значение целым числом. Если это так, то переменная `$id` содержит число; если нет, то она получает значение `false`. Если значения не было в строке запроса, переменная получает значение `null`.
3. Если в строке запроса не было допустимого целого числа, подключается файл `page-not-found.php`. Он заканчивается командой `exit`, останавливающей выполнение любого другого кода.
4. SQL для получения данных публикации пишется в переменную `$sql`.
5. Функция `pdo()` выполняет запрос, метод `fetch()` получает информацию о публикации, которая записывается в переменную `$article`.
6. Если публикация не была найдена в базе данных, подключается файл `page-not-found.php` и код останавливает выполнение.
7. Если код все еще выполняется, SQL для получения категорий в навигации сохраняется в переменной `$sql`.
8. Выполняется запрос, затем метод `fetchAll()` получает данные, и они сохраняются в переменной `$navigation`.
9. Идентификатор категории, к которой относится публикация, записывается в переменную `$section`, чтобы ее можно было выделить на панели навигации.
10. Название публикации пишется в переменную `$title`.
11. Краткое описание публикации пишется в `$description`.

Поскольку все необходимые для страницы данные получены и записаны в переменные, остальная часть файла под пунктирной линией создает HTML-код, выводимый в браузер.

12. Подключение файла `header.php`.
13. Если для публикации было загружено изображение, имя файла изображения выводится в атрибуте `src` тега `<img>`.

Если изображение не было загружено, вместо него отображается заглушка.

14. Если для изображения был добавлен пояснительный текст, он отображается в атрибуте `alt` тега `<img>`. Если нет, то атрибут будет пустым.
15. Название публикации отображается в элементе `<h1>`.
16. Отображается дата создания публикации с помощью функции `format_date()`. Она обеспечивает единое форматирование дат.
17. Вывод текста публикации.
18. Создание ссылки на файл `category.php`. Идентификатор категории, к которой принадлежит публикация, добавляется в строку запроса, чтобы можно было отобразить страницу этой категории.
19. Название категории используется в качестве текста ссылки.
20. Создание ссылки на файл `member.php`. Идентификатор пользователя, разместившего публикацию, добавляется в строку запроса, чтобы можно было отобразить персональную страницу этого пользователя.
21. В качестве текста ссылки используется имя пользователя, разместившего публикацию.
22. Подключение файла `footer.php`.

PHP section_c/c12/cms/article.php?id=1

```php
<?php
declare(strict_types = 1);                                  // Включение строгой типизации
require 'includes/database-connection.php';                 // Создание объекта PDO
require 'includes/functions.php';                           // Подключение функций
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Валидация идентификатора
if (!$id) {                                                 // Если идентификатор неподходящий
    include 'page-not-found.php';                           // Вывести 404
}
$sql = "SELECT a.title, a.summary, a.content, a.created, a.category_id, a.member_id,
               c.name      AS category,
               CONCAT(m.forename, ' ', m.surname) AS author,
               i.file AS image_file, i.alt  AS image_alt
          FROM article     AS a
          JOIN category    AS c  ON a.category_id = c.id
          JOIN member      AS m  ON a.member_id   = m.id
          LEFT JOIN image  AS i  ON a.image_id    = i.id
         WHERE a.id = :id  AND a.published = 1;";           // SQL
$article = $article = pdo($pdo, $sql, [$id])->fetch();      // Получение публикации
if (!$article) {                                            // Если не найдена
    include 'page-not-found.php';                           // Вывести 404
}
$sql = "SELECT id, name FROM category WHERE navigation = 1;"; // SQL для категорий
$navigation  = pdo($pdo, $sql)->fetchAll();                 // Навигация по категориям
$section     = $article['category_id'];                     // Текущая категория
$title       = $article['title'];                           // Для HTML-тега <title>
$description = $article['summary'];                         // Для тега <meta>
?>
<?php include 'includes/header.php'; ?>
  <main class="article container">
    <section class="image">
      <img src="uploads/<?= html_escape($article['image_file'] ?? 'blank.png') ?>"
           alt="<?= html_escape($article['image_alt']) ?>">
    </section>
    <section class="text">
      <h1><?= html_escape($article['title']) ?></h1>
      <div class="date"><?= format_date($article['created']) ?></div>
      <div class="content"><?= html_escape($article['content']) ?></div>
      <p class="credit">
        Posted in <a href="category.php?id=<?= $article['category_id'] ?>">
        <?= html_escape($article['category']) ?></a>
        by <a href="member.php?id=<?= $article['member_id'] ?>">
          <?= html_escape($article['author']) ?></a>
      </p>
    </section>
  </main>
<?php include 'includes/footer.php'; ?>
```

СТРАНИЦА ПОЛЬЗОВАТЕЛЯ

Файл member.php отображает информацию о пользователе и выводит список его публикаций. Строка запроса содержит идентификатор пользователя, информация о котором должна быть выведена на странице.

1. Включение строгой типизации и подключение необходимых файлов, database-connection.php и functions.php.
2. Функция filter_input() ищет в строке запроса параметр с именем id и проверяет, является ли его значение целым числом. Если это так, то в переменную $id записывается это число, а если нет, то false. Если такого параметра не было в строке запроса, переменная получает значение null.
3. Если в строке запроса не было подходящего целого числа, подключается файл page-not-found.php. Он заканчивается командой exit, останавливающей выполнение любого другого кода.
4. SQL для получения данных пользователя записывается в переменную $sql.
5. Функция pdo() выполняет запрос, метод fetch() возвращает полученные данные в переменную $member.
6. Если переменная $member пустая, подключается файл page-not-found.php и выполнение скрипта останавливается.
7. Если код все еще выполняется, SQL для получения публикаций присваивается переменной $sql.
8. Запрос выполняется, метод fetchAll() получает возвращаемые данные, и они сохраняются в переменной $articles.
9. SQL для получения категорий для навигации пишется в переменную $sql.
10. Запрос выполняется, метод fetchAll() получает возвращаемые данные, и они сохраняются в переменной $navigation.
11. Поскольку страница пользователя не входит ни в какую категорию, в переменную $section пишется пустое значение.
12. Имя пользователя записывается в переменную $title, чтобы его можно было отобразить в заголовке страницы.
13. Имя пользователя, за которым следуют слова Creative Folk, сохраняется в переменной $description для использования в теге <meta>.

После того как необходимые данные были получены из БД и сохранены в переменных, остальная часть файла под пунктирной линией создает HTML-код для вывода в браузер.

14. Подключение файла header.php.
15. Имя и фамилия пользователя выводятся внутри элемента <h1>.
16. Дата регистрации пользователя отображается с помощью функции format_date() в едином для сайта формате.
17. Если пользователь загрузил фотографию профиля, имя файла с изображением выводится в атрибуте src тега <img>. Если изображение не было загружено, вместо него выводится изображение-заглушка.
18. Если для изображения был добавлен пояснительный текст, он отображается в атрибуте alt тега <img>.
19. Цикл foreach перебирает созданный в шаге 7 массив $articles. В нем содержится подробная информация о размещенных этим пользователем публикациях.
20. Код для вывода карточки публикации идентичен тому, который используется на главной странице.

Если у автора нет публикаций, код внутри цикла выполняться не будет.

21. Подключение файла footer.php.

PHP section_c/c12/cms/member.php?id=1

```php
<?php
declare(strict_types = 1);                                  // Включение строгой тпипзации
require 'includes/database-connection.php';                 // Создание объекта PDO
require 'includes/functions.php';                           // Подключение функций
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Валидация идентификатора
if (!$id) {                                                 // Если нет подходящего
    include 'page-not-found.php';                           // Страница 404
}
$sql = "SELECT forename, surname, joined, picture FROM member WHERE id = :id;"; // SQL
$member = pdo($pdo, $sql, [$id])->fetch();                  // Получение данных пользователя
if (!$member) {                                             // Если массив пуст
    include 'page-not-found.php';                           // Страница 404
}
$sql = "SELECT a.id, a.title, a.summary, a.category_id, a.member_id,
               c.name     AS category,
               CONCAT(m.forename, ' ', m.surname) AS author,
               i.file     AS image_file,
               i.alt      AS image_alt
          FROM article    AS a
          JOIN category   AS c   ON a.category_id = c.id
          JOIN member     AS m   ON a.member_id   = m.id
          LEFT JOIN image AS i   ON a.image_id    = i.id
         WHERE a.member_id = :id AND a.published  = 1
         ORDER BY a.id DESC;";                                  // SQL
$articles = pdo($pdo, $sql, [$id])->fetchAll();                 // Публикации
$sql = "SELECT id, name FROM category WHERE navigation = 1;";   // SQL для категорий
$navigation  = pdo($pdo, $sql)->fetchAll();                     // Получение категорий
$section     = '';                                              // Текущая категория
$title       = $member['forename'] . ' ' . $member['surname']; // HTML-тег <title>
$description = $title . ' on Creative Folk';                    // Мета описание
?>
<?php include 'includes/header.php'; ?>
  <main class="container" id="content">
    <section class="header">
      <h1><?= html_escape($member['forename'] . ' ' . $member['surname']) ?></h1>
      <p class="member"><b>Member since:</b> <?= format_date($member['joined']) ?></p>
      <img src="uploads/<?= html_escape($member['picture'] ?? 'blank.png') ?>"
           alt="<?= html_escape($member['forename']) ?>" class="profile"><br>
    </section>
    <section class="grid">
    <?php foreach ($articles as $article) { ?>
    <!-- Код для вывода карточек публикаций такой же, какой был показан ранее -->
    <?php } ?>
    </section>
  </main>
<?php include 'includes/footer.php'; ?>
```

ДОБАВЛЕНИЕ ФУНКЦИИ ПОИСКА

На примере страницы поиска будет продемонстрировано, как использовать SQL для поиска данных в БД и как реализовать разбивку на страницы, когда запрос к базе данных может возвращать очень много строк.

Когда посетитель вводит поисковый запрос в форму и отправляет его на сервер, база данных ищет веденный текст в столбцах `title`, `summary` и `description` таблицы `article`. Если искомый текст найден в любом из этих столбцов, информация о публикации добавляется в результирующий набор, данные из которого будут отображены пользователю.

На странице поиска выполняются два SQL-запроса:

- один подсчитывает общее количество найденных результатов;
- другой запрашивает сами публикации.

Ниже представлен SQL-запрос, подсчитывающий количество публикаций, соответствующих поисковой фразе.

Искомое слово повторяется в запросе трижды: сначала чтобы проверить столбец `title`, потом для проверки столбца `summary` и наконец столбца `content`. Поскольку при заданных нами настройках PDO нельзя повторно использовать плейсхолдер с одним и тем же именем, то для поиска в каждом из столбцов используются разные имена плейсхолдеров (`:term1`, `:term2` и `:term3`).

```
SELECT COUNT(title)
  FROM article
 WHERE title   LIKE :term1
    OR summary LIKE :term2
    OR content LIKE :term3
   AND published = 1;
```

На странице поиска отображается по три результата. Это сделано, чтобы продемонстрировать, как при нахождении базой данных большого количества строк вы можете:

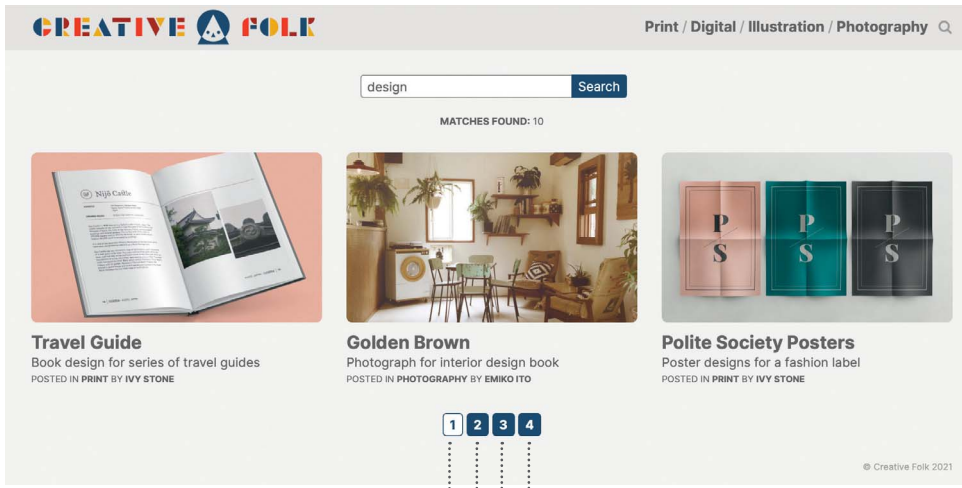
- показать часть этих результатов на странице;
- добавить под результатами ссылки на другие страницы, которые позволят посетителям просматривать остальные найденные результаты.

Это называется **разбивкой на страницы** (пагинацией), потому что результаты разбиваются на несколько страниц. Ссылка на каждую из страниц, отображающих результаты поиска, должна содержать три пары «имя — значение» в строке запроса, на каждой странице в коде имелась информация о том, какие статьи нужно искать, а также сколько результатов выводить:

- `term` содержит поисковую фразу;
- `show` указывает, сколько результатов показывать на странице;
- `from` указывает, сколько результатов уже было показано.

Значение `show` используется с оператором `SQL LIMIT`, чтобы база данных возвращала нужное количество статей, которые будут показаны на странице.
Значение `from` используется с оператором `SQL OFFSET`, чтобы сообщить базе данных, что она должна начинать добавлять результаты в результирующий набор только после того, как будет найдено указанное количество совпадений.

```
 LIMIT :show
OFFSET :from
```



1 2 3 4

```
search.php?term=design
search.php?term=design&show=3&from=3
search.php?term=design&show=3&from=6
search.php?term=design&show=3&from=9
```

FROM	SHOW		СТРАНИЦА
(0	÷ 3)	+ 1	= 1
(3	÷ 3)	+ 1	= 2
(6	÷ 3)	+ 1	= 3
(9	÷ 3)	+ 1	= 4

Для создания ссылок с пагинацией странице поиска требуются три фрагмента данных. Каждый из них содержится в отдельной переменной:

- `$count` — количество результатов, соответствующих запросу;
- `$show` — количество результатов для отображения на странице;
- `$from` — количество результатов, которые нужно пропустить перед добавлением совпадений в результирующий набор.

Чтобы рассчитать количество страниц, необходимых для отображения результатов, разделите значение `$count` (количество совпадений) на значение `$show` (количество результатов, выводимых на одной странице). Приведенный выше запрос находит 10 совпадений, и на каждой странице показывается по 3 результата. 10 ÷ 3 = 3,3333. Это число необходимо округлить с помощью функции PHP `ceil()`, чтобы получить полное количество страниц, необходимое для отображения результатов.

```
$total_pages = ceil($count / $show);
```

Чтобы определить текущую страницу, разделите значение `$from` (количество результатов, которые нужно пропустить) на `$show` (количество результатов на странице), затем добавьте 1. Вычисления для определения текущей страницы показаны вверху справа.

```
$current_page = ceil($from / $show) + 1;
```

Цикл `for` используется для создания ссылок на страницы. Он устанавливает счетчик равным 1 и проверяет, меньше ли значение счетчика общего количества страниц. Если меньше, он добавляет ссылку на страницу, а затем увеличивает счетчик. Цикл выполняется до тех пор, пока счетчик не достигнет общего количества страниц, необходимых для вывода результатов.

```
for ($i = 1; $i <= $total_pages; $i++)
{
    // Отобразить следующую ссылку
}
```

СТРАНИЦА ПОИСКА

Когда пользователь вводит поисковую фразу в форму в верхней части страницы `search.php` и отправляет ее на сервер, запрос приходит на ту же страницу, которая затем ищет соответствующие публикации и отображает найденные результаты.

1. Включение строгой типизации и подключение необходимых файлов: `database-connection.php` и `functions.php`.
2. Получение трех значений из строки запроса с помощью функции `filter_input()` и сохранение их в переменных:

- `$term` содержит поисковую фразу;
- `$show` возвращает количество результатов для отображения на странице (по умолчанию используется число 3, если значение не задано);
- `$from` возвращает количество результатов для пропуска (по умолчанию используется число 0, если значение не задано).

3. Инициализация двух переменных, которые понадобятся в дальнейшем для создания HTML-страницы, но актуальные значения будут присвоены им только при наличии поисковой фразы (`$count` содержит значение 0, а `$articles` — пустой массив).
4. Проверка с помощью конструкции `if`, содержит ли переменная `$term` поисковую фразу. Это нужно потому, что следующая часть кода, которая ищет совпадения в базе данных, должна выполняться, только если форма поиска была отправлена.
5. Создание массива `$arguments`, в который пишутся имена трех плейсхолдеров, используемых в SQL-запросе, и значения, на которые они должны быть заменены. Каждый из них заменяется одним и тем же поисковым термином, поскольку при текущих настройках PDO нельзя использовать одно и то же имя плейсхолдера несколько раз. Знак подстановки `%` добавляется до и после поискового запроса, чтобы оператор LIKE находил совпадение с поисковой фразой в любой части содержимого колонки.
6. Первый SQL-запрос служит для подсчета, сколько публикаций в таблице `article` содержат поисковую фразу в столбцах `title`, `summary` или `content`.
7. Функция `pdo()` выполняет этот запрос, а затем метод `fetchColumn()` объекта `PDOStatement` возвращает число, полученное этим запросом. Оно соответствует количеству публикаций, в которых была найдена поисковая фраза. Как следует из названия, метод `fetchColumn()` получает значение из одного столбца (поля) результирующего набора. Возвращаемое значение сохраняется в `$count`.
8. С помощью конструкции `if` проверяется, были ли найдены соответствующие запросу публикации. Если были, то выполняется запрос, который получает полную информацию об этих публикациях. Если же нет, то нет смысла и пытаться ее запрашивать.
9. В массив `$argument` добавляются еще два элемента, чтобы их можно было использовать во втором SQL-запросе:

- `show` — количество результатов для отображения на странице;
- `from` — количество результатов, которые нужно пропустить (оба были сохранены в переменных в шаге 2).

10. В переменную `$sql` записывается SQL-запрос для получения публикаций, которые будут показаны на этой странице.
11. Условие в операторе `WHERE` ищет статьи, содержащие поисковый запрос в столбцах `title`, `summary` или `description` таблицы `article`.
12. Оператор `ORDER BY` упорядочивает результаты по идентификатору статьи в порядке убывания, поэтому самые новые будут выведены первыми.
13. Оператор `LIMIT` ограничивает количество результатов, добавляемых в набор, количеством результатов поиска, которые должны отображаться на странице.
14. Оператор `OFFSET` определяет, сколько совпадений будет пропущено до добавления данных в результирующий набор.
15. SQL выполняется с использованием значений в обновленном массиве `$arguments`. Затем вызывается метод `fetchAll()` объекта `PDOStatement` для получения всех соответствующих описаний. Они записываются в переменную `$articles`.

PHP section_c/c12/cms/search.php

```php
<?php
declare(strict_types = 1);                                  // Строгая типизация
require 'includes/database-connection.php';                 // Создание объекта PDO
require 'includes/functions.php';                           // Подключение функций

$term  = filter_input(INPUT_GET, 'term');                   // Получение поисковой фразы
$show  = filter_input(INPUT_GET, 'show', FILTER_VALIDATE_INT) ?? 3; // Для LIMIT
$from  = filter_input(INPUT_GET, 'from', FILTER_VALIDATE_INT) ?? 0; // Для OFFSET
$count = 0;                          // инициализация кол-ва найденных результатов
$articles = [];                      // Присвоение переменной article пустого массива

if ($term) {                                        // Если есть поисковая фраза
    $arguments['term1'] = '%$term%';                // Сохранение поисковой фразы в массиве
    $arguments['term2'] = '%$term%';                // три раза, поскольку имя плейсхолдера
    $arguments['term3'] = '%$term%';                // не может быть использовано повторно

    $sql = "SELECT COUNT(title) FROM article
             WHERE title   LIKE :term1
                OR summary LIKE :term2
                OR content LIKE :term3
               AND published = 1;";      // Сколько публикаций соответствует запросу
    $count = pdo($pdo, $sql, $arguments)->fetchColumn();  // Получение количества
    if ($count > 0) {                   // Если публикации по запросу нашлись
        $arguments['show'] = $show;     // Добавляем значения для разбивки на страницы
        $arguments['from'] = $from;     // Добавляем значения для разбивки на страницы
        $sql = "SELECT a.id, a.title, a.summary, a.category_id, a.member_id,
                       c.name      AS category,
                       CONCAT(m.forename, ' ', m.surname) AS author,
                       i.file      AS image_file,
                       i.alt       AS image_alt
                  FROM article     AS a
                  JOIN category    AS c    ON a.category_id = c.id
                  JOIN member      AS m    ON a.member_id   = m.id
                  LEFT JOIN image  AS i    ON a.image_id    = i.id
                 WHERE a.title   LIKE :term1
                    OR a.summary LIKE :term2
                    OR a.content LIKE :term3
                   AND a.published = 1
              ORDER BY a.id DESC
                 LIMIT :show
                OFFSET :from;";                         // SQL для получения публикаций
        $articles = pdo($pdo, $sql, $arguments)->fetchAll(); // Выполнение запроса
                                                             // и получение результатов
    }
}
```

СТРАНИЦА ПОИСКА (ПРОДОЛЖЕНИЕ)

Далее в коде происходит вычисление значений, необходимых для создания ссылок на страницы.

1. Если число в переменной `$count` больше, чем значение переменной `$show`, необходимо вычислить значения для ссылок на страницы.

2. Общее количество страниц, необходимых для отображения результатов, сохраняется в переменной `$total_pages` и рассчитывается по формуле:

- разделить значение переменной `$count` на значение `$show`;
- округлить результат с помощью функции PHP `ceil()`.

3. Текущая страница (сохраненная в переменной `$current_page`) вычисляется так:

- разделить значение переменной `$show` на значение `$from`;
- округлить его с помощью функции PHP `ceil()`;
- добавить 1 к полученному результату.

4. SQL для получения категорий, чтобы они могли отображаться на панели навигации, пишется в переменную `$sql`.

5. Запрос выполняется, метод `fetchAll()` получает данные категорий, и они сохраняются в переменной `$navigation`.

6. Поскольку страница поиска не относится ни к какой категории, в переменную `$section` пишется пустая строка.

7. Заголовок страницы сохраняется в переменной `$title`. Он состоит из слов Search results for («Результаты поиска для») и поискового запроса. Эта информация экранируется при выводе, в файле `header.php`).

8. Текст для тега `<meta>` сохраняется в переменной `$description`.

Остальная часть файла создает выводимую в браузер HTML-страницу.

9. Форма поиска, которая отправляет данные на эту же страницу.

10. Поисковая фраза экранируется и отображается в форме ввода.

11. Если была введена поисковая фраза и, следовательно, в переменной `$term` содержится непустое значение, то выводится количество найденных публикаций.

12. Карточки публикаций выводятся с использованием цикла `foreach`, как и в предыдущих примерах.

13. Конструкция `if` проверяет, больше ли количество найденных публикаций (в переменной `$count`), чем количество, отображаемое на одной странице (в переменной `$show`). Если да, то выводятся ссылки на разбивку по страницам.

14. Элементы `<nav>` и `<ul>` добавляются для форматирования ссылок на страницы. Каждая ссылка будет находиться в элементе `<li>`.

15. Цикл `for` используется для создания ссылок на страницы. В круглых скобках содержатся три выражения:

- `$i = 1` создает счетчик с именем `$i` и устанавливает его равным 1.
- `$i <= $total_pages` — это условие проверяет, должен ли выполняться код в цикле. Если счетчик меньше общего количества страниц, необходимых для отображения результатов поиска, блок кода в цикле будет выполнен.
- `$i++` увеличивает счетчик на 1 при каждом обороте цикла.

16. Внутри цикла для каждой страницы создается ссылка. Атрибут `href` использует строку запроса с тремя элементами, которые передают в файл `search.php`, информацию о том, какие результаты показывать:

- `term` — это поисковая фраза[60];
- `show` — количество результатов, отображаемых на странице;
- `from` — это количество результатов, которые нужно пропустить. Например, на первой странице `$i` равно 1, следовательно, (1-1) * 3 пропускает 0 результатов; на второй странице `$i` равно 2, следовательно, (2-1) * 3 пропускает 3 результата; на третьей странице `$i` равно 3, следовательно, (3-1) * 3 пропускает 6 результатов.

17. Если значение в счетчике совпадает со значением в переменной `$current_page`, ссылка должна быть помечена как текущая. Для этого к атрибуту `class` добавляется значение `active`, а также добавляется атрибут `aria-current` со значением `true`.

18. Внутри ссылки значение счетчика используется в качестве текста ссылки для отображения номера страницы. Цикл выполняется до тех пор, пока значение в счетчике не станет равным значению переменной `$total_pages`.

PHP section_c/c12/cms/search.php

```php
① if ($count > $show) {            // Если результатов больше, чем выводится на странице
②     $total_pages  = ceil($count / $show);         // Подсчет общего количества страниц
③     $current_page = ceil($from / $show) + 1;      // Вычисление текущей страницы
  }
④ $sql = "SELECT id, name FROM category WHERE navigation = 1;"; // SQL для категорий
⑤ $navigation  = pdo($pdo, $sql)->fetchAll();               // Навигация по категориям

⑥ $section     = '';                                        // Текущая категория
⑦ $title       = 'Search results for ' . $term;             // Для тега <title>
⑧ $description = $title . ' on Creative Folk';              // Для тега <meta>
  ?>
  <?php include 'includes/header.php'; ?>
    <main class="container" id="content">
      <section class="header">
⑨       <form action="search.php" method="get" class="form-search">
          <label for="search"><span>Search for: </span></label>
⑩         <input type="text" name="term" value="<?= html_escape($term) ?>"
                 id="search" placeholder="Enter search term"
          /><input type="submit" value="Search" class="btn" />
        </form>
⑪       <?php if ($term) { ?><p><b>Matches found:</b> <?= $count ?></p><?php } ?>
      </section>

      <section class="grid">
        <?php foreach ($articles as $article) { ?>
⑫       <!-- Код для отображения публикаций такой же, как было показано ранее -->
        <?php } ?>
      </section>

⑬     <?php if ($count > $show) { ?>
⑭     <nav class="pagination" role="navigation" aria-label="Pagination navigation">
        <ul>
⑮       <?php for ($i = 1; $i <= $total_pages; $i++) { ?>
          <li>
⑯           <a href="?term=<?= $term ?>&show=<?= $show ?>&from=<?= (($i - 1) * $show) ?>"
⑰             class="btn <?= ($i == $current_page) ? 'active" aria-current="true' : '' ?>">
⑱             <?= $i ?>
            </a>
          </li>
        <?php } ?>
        </ul>
      </nav>
      <?php } ?>
    </main>
  <?php include 'includes/footer.php'; ?>
```

ПОЛУЧЕНИЕ ДАННЫХ В ВИДЕ ОБЪЕКТОВ

PDO может представлять каждую строку данных в результирующем наборе не только как массив, но и в виде объекта (и методы этого объекта могут работать с возвращаемыми данными). Режим выборки PDO определяет, как должны быть представлены данные.

Режимы выборки управляют тем, как объект `PDOStatement` возвращает каждую строку данных в результирующем наборе. Это может быть:

- ассоциативный массив, в котором имя каждого столбца из результирующего набора используется в качестве ключа массива;
- объект, в котором каждое имя столбца из результирующего набора используется как свойство объекта.

В файле `database-connection.php` используется массив `$options`, в котором устанавливается режим выборки по умолчанию, который возвращает строки данных в виде ассоциативных массивов.
Чтобы установить режим выборки по умолчанию, при котором каждая строка данных будет возвращаться в виде объекта, а не массива, используйте значение `PDO::FETCH_OBJ`.

МАССИВ ⟶
```
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC;
```
ОБЪЕКТ ⟶
```
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ;
```

Каждый объект `PDOStatement` также содержит метод под названием `setFetchMode()`. Этот метод может быть использован для установки режима выборки для отдельного объекта `PDOStatement`. Он переопределяет метод выборки, который был задан по умолчанию.

Метод `setFetchMode()` вызывается после метода `execute()`. Его единственным параметром будет используемый режим выборки. Ниже говорится, что каждая строка данных из результирующего набора должна быть возвращена как объект.

```
$statement->setFetchMode(PDO::FETCH_OBJ);
```
ОБЪЕКТ **PDOStatement** — УСТАНОВКА РЕЖИМА ВЫБОРКИ — ИЗВЛЕКАТЬ КАЖДУЮ СТРОКУ КАК ОБЪЕКТ

Режим выборки также может быть указан в качестве аргумента методов `fetch()` и `fetchAll()`. Когда `fetchAll()` используется для извлечения нескольких строк данных из результирующего набора, каждый объект сохраняется в отдельном элементе индексированного массива.

При этом объект создается с использованием стандартного **предопределенного класса** — это пустой класс (без свойств или методов). Имя этого класса — `stdClass`. Далее будет показано, как можно возвращать данные в виде объектов, создаваемых с использованием существующего класса.

```
$statement->fetch(PDO::FETCH_OBJ);
```
ОБЪЕКТ **PDOStatement** — ПОЛУЧЕНИЕ ДАННЫХ — ИЗВЛЕКАТЬ КАЖДУЮ СТРОКУ КАК ОБЪЕКТ

НАСТРОЙКА РЕЖИМА ВЫБОРКИ ДЛЯ ПОЛУЧЕНИЯ ОБЪЕКТА

PHP section_c/c12/examples/fetching-data-as-objects.php

```
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
$sql = "SELECT id, forename, surname
          FROM member;";                    // SQL
$statement = $pdo->query($sql); // Выполнение запроса
$statement->setFetchMode(PDO::FETCH_OBJ); // Режим
                                          // выборки
$members   = $statement->fetchAll(); // Получение данных
?>
<!DOCTYPE html>
<html> ...
  <body>
    <?php foreach ($members as $member) { ?>
      <p>
        <?= html_escape($member->forename) ?>
        <?= html_escape($member->surname) ?>
      </p>
    <?php } ?>
  </body>
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

Luke Wood

Emiko Ito

В этом примере каждая строка результирующего набора представлена объектом, а каждая колонка в строке — свойством этого объекта.

1. Подключение файлов `database-connection.php` и `functions.php`.
2. Запрос пишется в переменную `$sql`.
3. Метод `query()` объекта `PDO` выполняет запрос и возвращает объект `PDOStatement`, представляющий результирующий набор.
4. Вызывается метод `setFetchMode()` объекта `PDOStatement`. Аргумент `PDO::FETCH_OBJ` указывает, что каждая строка результирующего набора должна быть возвращена в виде объекта.
5. Метод `fetchAll()` объекта `PDOStatement` получает все строки из результирующего набора, возвращая индексированный массив. Значение каждого элемента в этом массиве — объект, представляющий одну строку данных.
6. Цикл `foreach` используется для перебора всех элементов массива.
7. Имя пользователя отображается с использованием свойств объекта, содержащих его имя и фамилию.

Упражнение.
В шаге 2 также запросить адрес электронной почты пользователя, а затем отобразить эту информацию в шаге 7.

Упражнение.
Заменить шаг 7 на `<?php var_dump($member)?>`, чтобы увидеть создаваемый для каждой строки данных объект.

ПОЛУЧЕНИЕ ДАННЫХ В ВИДЕ ОБЪЕКТА СУЩЕСТВУЮЩЕГО КЛАССА

PDO может возвращать каждую строку результирующего набора как объект, созданный с применением пользовательского класса. Объекты, созданные с помощью этого класса, автоматически получат любые методы этого класса.

Чтобы добавить данные строки результирующего набора в объект существующего определения класса, используйте метод `setFetchMode()` объекта `PDOStatement` с двумя параметрами:

- режим выборки `PDO::FETCH_CLASS`;
- имя используемого класса.

Имя класса берется в кавычки. Определение класса должно присутствовать на странице перед вызовом функции `setFetchMode()`.

На странице справа показан класс, используемый для создания объектов, представляющих пользователей сайта. Класс находится в файле с именем `Member.php`, который лежит в папке под названием `classes`.

Свойства этого объекта совпадают с именами двух столбцов в таблице `member` базы данных.

- Когда имя столбца в результирующем наборе совпадает с именем свойства в классе, этому свойству присваивается значение из этого столбца.
- Если результирующий набор содержит имя столбца, которое не является свойством класса, это имя столбца добавляется как дополнительное свойство объекта.

Любой объект, создаваемый PDO с помощью этого класса, также будет содержать методы, указанные в определении класса.

Когда `fetchAll()` используется для извлечения нескольких строк данных из результирующего набора, каждый объект сохраняется в отдельный элемент индексированного массива.

```
$statement->setFetchMode(PDO::FETCH_CLASS, 'Member');
```

ОБЪЕКТ PDOStatement	УСТАНОВКА РЕЖИМА ВЫБОРКИ	ИЗВЛЕЧЕНИЕ ДАННЫХ В СУЩЕСТВУЮЩИЙ КЛАСС	ИМЯ КЛАССА

Когда объекты создаются в режиме `PDO::FETCH_CLASS`, значения свойствам объекта присваиваются до вызова метода `__construct()`. Это может привести к непредвиденным результатам. Однако в PDO есть и другие режимы выборки объектов, но их описание выходит за рамки этой книги.

СОЗДАНИЕ ОБЪЕКТА С ИСПОЛЬЗОВАНИЕМ СУЩЕСТВУЮЩЕГО КЛАССА

PHP section_c/c12/examples/classes/Member.php

```php
<?php
class Member
{
  public $forename;
  public $surname;
  public function getFullName(): string
  {
    return $this->forename . ' ' . $this->surname;
  }
}
```

PHP section_c/c12/examples/fetching-data-into-class.php

```php
<?php
require '../cms/includes/database-connection.php';
require '../cms/includes/functions.php';
require 'classes/Member.php';
$sql = "SELECT forename, surname
          FROM member
         WHERE id = 1;";
$statement = $pdo->query($sql);
$statement->setFetchMode(PDO::FETCH_CLASS, 'Member');
$member = $statement->fetch();
?>
<!DOCTYPE html>
<html> ...
  <p><?= html_escape($member->getFullName()) ?></p> ...
</html>
```

РЕЗУЛЬТАТ

Ivy Stone

1. Определение класса `Member`, которое содержит два свойства и один метод.
2. Подключение файлов `database-connection.php`, `functions.php` и `Member.php`.
3. Запрос сохраняется в переменной `$sql`.
4. Метод `query()` объекта `PDO` выполняет запрос и создает объект `PDOStatement` для представления результирующего набора.
5. Вызывается метод `setFetchMode()` объекта `PDOStatement`:

- `PDO::FETCH_CLASS` указывает PDO добавить данные в объект, созданный с использованием существующего класса;
- `Member` — это имя класса, используемого для создания объекта.

6. Метод `fetch()` объекта `PDOStatement` получает строку данных из результирующего набора. Возвращаемый объект сохраняется в переменной `$member`.
7. Метод `getFullName()` объекта вызывается для получения полного имени пользователя.

Упражнение.
Заменить шаг 6 на `<?php var_dump($member)?>`, чтобы увидеть объект, созданный для этого пользователя.

ЗАКЛЮЧЕНИЕ

ПОЛУЧЕНИЕ И ВЫВОД ИНФОРМАЦИИ ИЗ БАЗЫ ДАННЫХ

- Объект `PDO` обеспечивает соединение с базой данных и управляет его настройками.
- Объект `PDOStatement` представляет запрос SQL и результирующий набор, который он генерирует. Он может возвращать каждую строку данных в виде ассоциативного массива или объекта.
- Если результирующий набор содержит более одной строки, каждая строка может быть сохранена в элементе индексированного массива.
- Строку запроса в URL можно использовать для указания, какие данные страница должна получить из базы данных.
- В SQL-запросах необходимо использовать плейсхолдеры для изменяемых значений.
- Убедитесь, что все данные, используемые на странице, экранируются при выводе.

13

ИЗМЕНЕНИЕ ИНФОРМАЦИИ В БАЗЕ ДАННЫХ

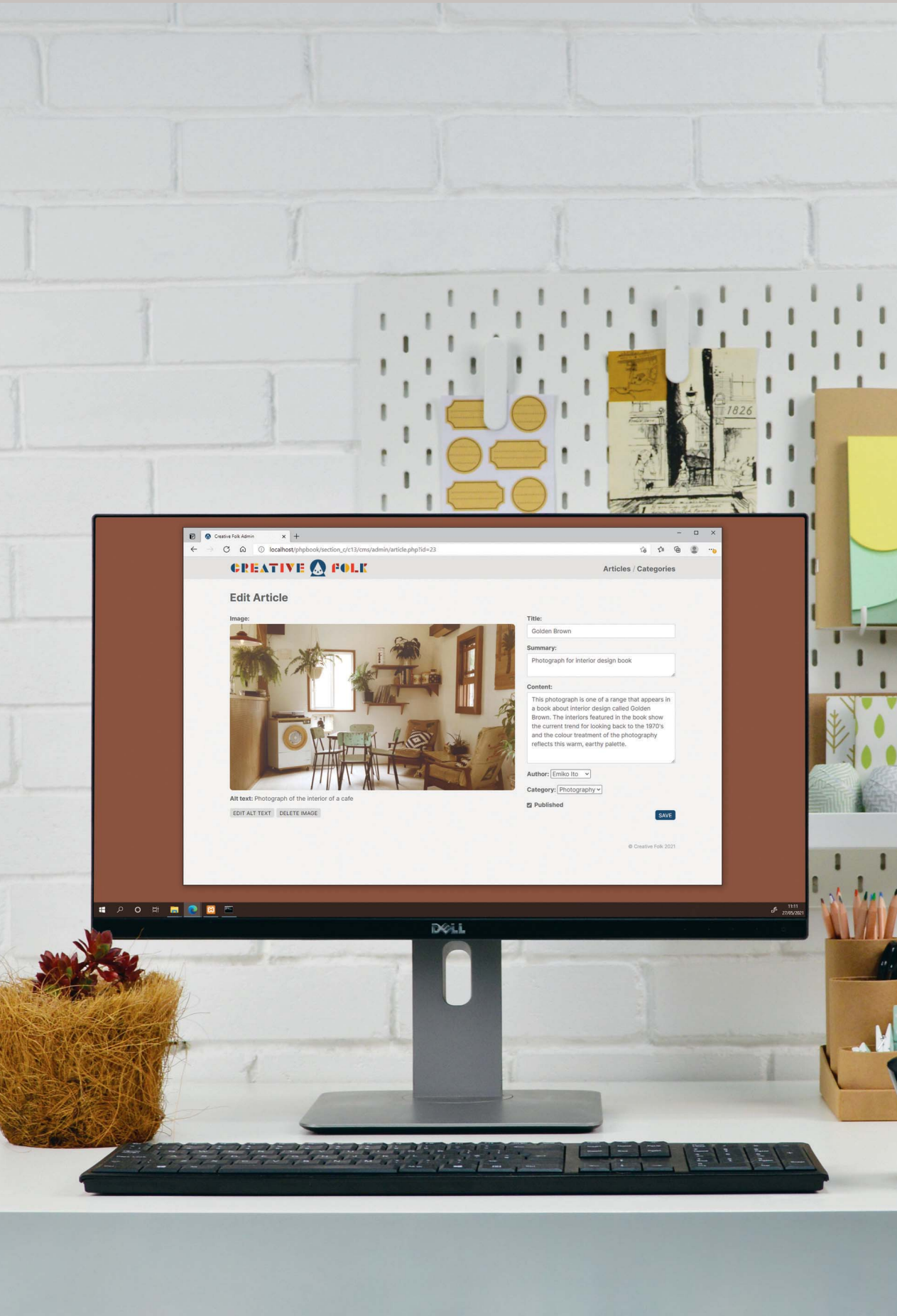
Веб-сайт может предоставлять инструменты, позволяющие пользователям добавлять новые данные в БД, а также обновлять или удалять существующие.

Чтобы сделать это, PHP-страницы должны выполнять следующие задачи.

1. **Получение данных.** В главе 6 показано, как получать данные из форм и URL-адресов.
2. **Валидация данных.** В главе 6 также показано, как проверить, были ли предоставлены все требуемые данные и соответствуют ли они требуемому формату. А если нет, то необходимо сообщить об этих ошибках пользователю.
3. **Обновление базы данных.** В главе 11 показаны запросы, которые добавляют, обновляют или удаляют данные в БД. В главе 12 показано, как SQL-запросы можно выполнять в PHP с помощью `PDO`.
4. **Предоставление обратной связи.** Сообщение, которое проинформирует пользователя об успешности или неуспешности его действий.

Поскольку вы уже научились выполнять большинство из этих задач, в этой главе основное внимание уделяется тому, в каком порядке они должны выполняться. Последовательность, в которой выполняются инструкции, называется **порядком выполнения**. В этой главе с помощью инструкций `if` мы будем сообщать интерпретатору PHP, когда выполнять каждую задачу. Например, если введенные пользователем данные не прошли проверку, нет смысла создавать или выполнять SQL- запросы для обновления базы данных. Аналогично, показывать сообщение об успешном выполнении операции следует только в том случае, если изменения в базе данных действительно были сделаны.

Вы также узнаете, как выполнить серию связанных инструкций SQL, используя механизм **транзакций**, чтобы изменения сохранились только в том случае, если все запросы SQL были выполнены успешно (если хотя бы один из них завершается неудачей, то не будут сохранены результаты ни одного из запросов).


Creative Folk Admin
localhost/phpbook/section_c/c13/cms/admin/article.php?id=23
CREATIVE FOLK
Articles / Categories
Edit Article
Image:
Alt text: Photograph of the interior of a cafe
EDIT ALT TEXT
DELETE IMAGE
Title:
Golden Brown
Summary:
Photograph for interior design book
Content:
This photograph is one of a range that appears in a book about interior design called Golden Brown. The interiors featured in the book show the current trend for looking back to the 1970's and the colour treatment of the photography reflects this warm, earthy palette.
Author: Emiko Ito
Category: Photography
Published
SAVE
© Creative Folk 2021
DELL

ДОБАВЛЕНИЕ ДАННЫХ В ТАБЛИЦУ

Чтобы добавить в новую строку в таблицу, используйте запрос INSERT. Этот запрос может добавить строку только в одну таблицу.

1. Приведенный ниже запрос SQL добавляет новую категорию в таблицу `category`. В нем есть плейсхолдеры для столбцов `name`, `description` и `navigation`. Значение для столбца `id` генерируется базой данных.

2. Используемые в каждом столбце данные представлены в ассоциативном массиве, содержащем по одному элементу для каждого плейсхолдера в SQL-запросе. Массив не должен содержать никаких дополнительных элементов, так как это может привести к ошибке.

3. Метод `prepare()` объекта `PDO` принимает в качестве аргумента SQL-запрос и возвращает объект `PDOStatement`. Метод `execute()` объекта `PDOStatement` использует значения из массива для выполнения запроса.

```
1 $sql = "INSERT INTO category (name, description, navigation)
          VALUES (:name, :description, :navigation);";

2 $category = ['name']        = 'News';
  $category = ['description'] = 'News about Creative Folk';
  $category = ['navigation']  = 1;

3 $statement = $pdo->prepare($sql);
  $statement->execute($category);
```

Добавленная этим запросом строка выделена справа. Механизм автоинкремента присваивает столбцу `id` значение 5. При возникновении проблем с выполнением запроса объект `PDO` выдаст исключение. На учебном веб-сайте оно будет обработано функцией обработки исключений.

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1
4	Photography	Capturing the moment	1
5	News	News about Creative Folk	1

ОБНОВЛЕНИЕ ДАННЫХ В ТАБЛИЦЕ

Чтобы обновить существующую строку в таблице базы данных, используйте SQL-запрос UPDATE. Команда UPDATE может обновлять несколько таблиц с помощью оператора JOIN.

1. Этот SQL-запрос обновляет существующую категорию. Первые три плейсхолдера содержат значения для столбцов `name`, `description` и `navigation`. В операторе `WHERE` плейсхолдер используется, чтобы указать идентификатор строки, которую необходимо обновить.

2. Данные для подстановки на место плейсхолдеров передаются в виде массива, содержащего ровно по одному элементу для каждого плейсхолдера в запросе. Он не должен содержать никаких дополнительных элементов, так как это может привести к ошибке.

3. Запрос выполняется с использованием предоставленных параметров. Ниже показано, как пользовательская функция `pdo()` применяется для выполнения SQL- запроса. Этот подход будет использоваться и в остальной части главы.

```
1 $sql = "UPDATE category
            SET name        = :name,
                description = :description,
                navigation  = :navigation
          WHERE id = :id;";

2 $category = ['id']          = 5;
  $category = ['name']        = 'News';
  $category = ['description'] = 'Updates from Creative Folk';
  $category = ['navigation']  = 0;

3 pdo($pdo, $sql, $category);
```

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1
4	Photography	Capturing the moment	1
5	News	Updates from Creative Folk	0

Слева показано, что была обновлена категория с `id=5`.

Примечание. Если условие поиска, указанное в операторе `WHERE`, соответствует более чем одной строке, SQL-запрос `UPDATE` обновит все соответствующие строки.

УДАЛЕНИЕ ДАННЫХ ИЗ ТАБЛИЦЫ

SQL-запрос DELETE удаляет строки из таблицы. Условие поиска указывает, какие строки должны быть удалены. Оператор JOIN может использоваться для удаления данных из нескольких таблиц.

1. В показанном ниже SQL-запросе после команды DELETE следует оператор FROM и имя таблицы, из которой необходимо удалить строку.

2. Далее в условии поиска задается, какую именно строку необходимо удалить. В примере ниже строка для удаления задается с помощью значения в столбце id.

3. Идентификатор удаляемой строки находится в переменной $id. Затем он передается в функцию pdo() с использованием индексированного массива, создаваемого в самом аргументе.

```
1  $sql = "DELETE FROM category
2            WHERE id = :id;";

3  $id = 5;
   pdo($pdo, $sql, [$id]);
```

Справа показано, что категория с id=5 была удалена из таблицы.

Примечание. Если условие поиска в операторе WHERE соответствует более чем одной строке, SQL-запрос DELETE удалит все совпадающие строки.

category

id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1
4	Photography	Capturing the moment	1

ПОЛУЧЕНИЕ ИДЕНТИФИКАТОРА НОВОЙ СТРОКИ В ТАБЛИЦЕ

Когда в таблицу с автоинкрементным идентификатором добавляется новая строка, получить сгенерированное базой данных для этого столбца значение можно с помощью метода `lastInsertId()` объекта `PDO`.

В базе данных учебного веб-сайта первый столбец в каждой таблице называется `id`. Он содержит первичный ключ, используемый для уникальной идентификации каждой строки таблицы. Когда в таблицу добавляется новая строка, механизм автоинкремента MySQL создает уникальное значение в этом столбце для каждой новой строки.

После выполнения SQL-запроса, добавляющего новую строку данных в таблицу, можно вызвать метод `lastInsertId()` объекта `PDO`, чтобы получить сгенерированное MySQL значение для столбца `id`. Его можно сохранить в переменной и использовать позже в коде.

В дальнейшем вы увидите, как этот метод используется при создании новой публикации и одновременной загрузке изображения для нее.

- Сначала добавляется новая строка в таблицу изображений.
- Затем метод `lastInsertId()` используется для получения ее идентификатора.
- Добавление записи в таблицу `article` выполняется самым последним, чтобы в столбец `image_id` можно было записать идентификатор только что добавленного изображения.

Ниже вы можете увидеть, как метод `lastInsertId()` вызывается после вызова функции `pdo()`, которая добавляет новую строку в базу данных.

```
pdo($pdo, $sql, $arguments);
$new_id = $pdo->lastInsertId();
```

ПОЛУЧЕНИЕ КОЛИЧЕСТВА ЗАТРОНУТЫХ СТРОК

Когда выполняется запрос UPDATE или DELETE, он может изменить более одной строки данных за раз. Метод rowCount() объекта PDOStatement возвращает количество затронутых операцией строк.

При выполнении команд UPDATE или DELETE программа может изменить одну, несколько или же ни одной строки базы данных, в зависимости от того, сколько строк соответствуют условию поиска в операторе WHERE.

Если в таблице category нет категории с id 100, при выполнении приведенного ниже запроса код ничего не удалит из базы данных:

```
DELETE FROM category
 WHERE id = 100;
```

При выполнении приведенного ниже запроса он может обновить одну, несколько или ни одной строки базы данных — в зависимости от того, сколько строк содержало значение 0 в столбце navigation.

```
UPDATE category
   SET navigation = 1
 WHERE navigation = 0;
```

Метод execute() объекта PDOStatement возвращает значение true, если запрос был выполнен успешно (а в случае ошибки будет выброшено исключение). Однако это не позволяет узнать, затронул ли запрос какие-то строки — запрос, который затронул или вернул 0 строк, все равно считается успешным.

Чтобы определить, сколько строк было изменено при выполнении запроса SQL, у объекта PDOStatement есть метод под названием rowCount(), возвращающий количество затронутых строк.

Метод rowCount()должен быть вызван с помощью метода execute(), а возвращаемое им значение может быть сохранено в переменной.

Если вы выполняете SQL-запрос с помощью определенной в предыдущей главе функции pdo(), метод rowCount() можно вызвать в той же инструкции, используя вызов методов по цепочке.

```
$sql = "UPDATE category
           SET navigation = 1
         WHERE navigation = 0;";
$result = $pdo($pdo, $sql)->rowCount();
```

ПРЕДОТВРАЩЕНИЕ ДУБЛИРОВАНИЯ ЗНАЧЕНИЙ

Значения в некоторых столбцах должны быть уникальными. На нашем учебном веб-сайте у двух публикаций или двух категорий не могут быть одинаковые названия, а у двух пользователей не могут быть одинаковые адреса электронной почты.

Ранее ограничение уникальности было добавлено к следующим столбцам базы данных, чтобы гарантировать, что ни в одной из строк не содержится одинаковых значений:

- столбец `title` таблицы `article`;
- столбец `name` таблицы `category`;
- столбец `email` таблицы `member`.

Когда в эти таблицы добавляется новая строка или обновляется существующая, если какая-то строка уже содержит добавляемое значение в одном из этих столбцов, MySQL выдаст ошибку, которую PDO выбросит в виде исключения, поскольку не может сохранить данные — это нарушит ограничение уникальности.

Для создания объекта исключения PDO использует класс `PDOException`. Он похож на классы исключений, но также содержит дополнительные данные, относящиеся к PDO. Ниже показано, как можно обработать ситуацию, когда выполнение SQL-запроса приводит к нарушению ограничения уникальности:

```
(1) try {
        pdo($pdo, $sql, $args);
(2) } catch (PDOException $e) {
(3)     if ($e->errorInfo[1] === 1062) {
            // Сообщить пользователю, что значение уже используется
        } else {
(4)         throw $e;
        }
    }
```

1. Код для создания новой или обновления существующей строки в этих таблицах помещается в блок `try`.

2. Если PDO выбросит исключение при выполнении кода в блоке `try`, выполняется последующий блок `catch`, а объект исключения сохранится в переменной с именем `$e`.

3. У объекта `PDOException` есть свойство, называемое `errorInfo`. Его значение представляет собой индексированный массив с данными об ошибке. Второй элемент в массиве — это код ошибки (http://notes.re/PDO/error-codes содержит полный список кодов ошибок). Если код ошибки равен 1062, это означает, что ограничение уникальности препятствует сохранению данных, и пользователь должен получить сообщение, что такое значение уже используется.

4. Если код ошибки другой, исключение выбрасывается повторно с использованием ключевого слова `throw` и будет обработано функцией обработки исключений по умолчанию.

СТРАНИЦЫ ДЛЯ РЕДАКТИРОВАНИЯ ДАННЫХ В БД

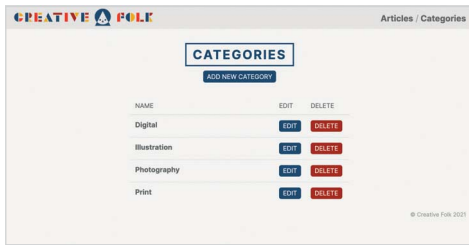
Теперь, после знакомства с тем, как с помощью PDO можно добавлять, обновлять и удалять записи в базе данных, в остальной части этой главы вы узнаете, как эти приемы используются для создания страниц администрирования сайта, позволяющих пользователям редактировать хранящиеся в БД данные с помощью HTML-форм.

Ниже показаны шесть страниц администрирования, позволяющие пользователям создавать, обновлять и удалять категории и статьи.

Все страницы администрирования находятся в папке под названием `admin`. Ознакомьтесь с их работой в браузере, прежде чем просматривать код.

ФАЙЛ CATEGORIES.PHP

На этой странице выводятся все категории, а также ссылки для создания, редактирования и удаления категорий.



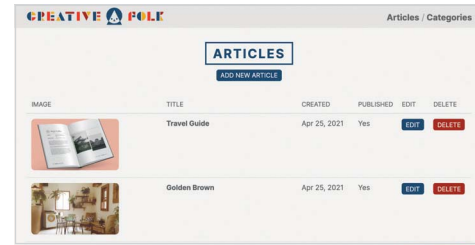
Ссылки для создания или редактирования категорий ведут на страницу `category.php`:

- при создании категории строка запроса в URL отсутствует;
- при редактировании категории строка запроса содержит элемент с именем `id`, значением которого является идентификатор редактируемой категории. Например, `category.php?id=2`.

Ссылки для удаления категории ведут на страницу под названием `category-delete.php`, а строка запроса содержит идентификатор категории для удаления.

ФАЙЛ ARTICLES.PHP

На этой странице выводятся все публикации, а также ссылки, позволяющие администраторам сайта создавать, редактировать и удалять размещенные работы.



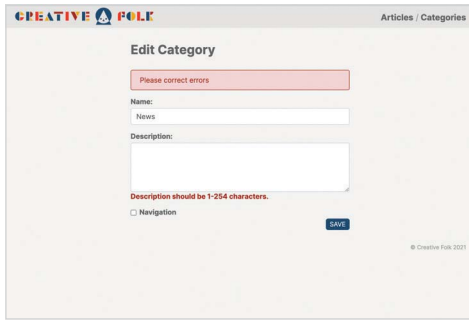
Ссылки для создания или редактирования статьи ведут на `article.php`:

- при создании статьи строка запроса в URL отсутствует;
- при редактировании публикации строка запроса содержит элемент с именем `id`, значением которого является идентификатор редактируемой публикации. Например, `article.php?id=2`.

Ссылки для удаления статьи ведут на страницу под названием `article-delete.php`, а строка запроса содержит идентификатор статьи для удаления.

ФАЙЛ CATEGORY.PHP

На этой странице представлена форма для создания новой категории или редактирования существующей.

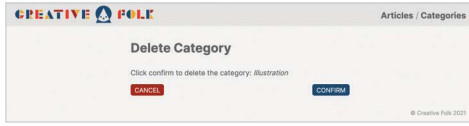


При отправке формы данные проходят валидацию.

- Если она успешна, код обновит базу данных и отправит пользователя обратно на страницу `categories.php`. В строке запроса при этом передается сообщение, чтобы страница категорий могла сообщить пользователю, что данные были сохранены[61].
- Если данные не проходят валидацию, форма отображается снова, с сообщениями под теми полями формы, которые необходимо исправить.

ФАЙЛ CATEGORY-DELETE.PHP

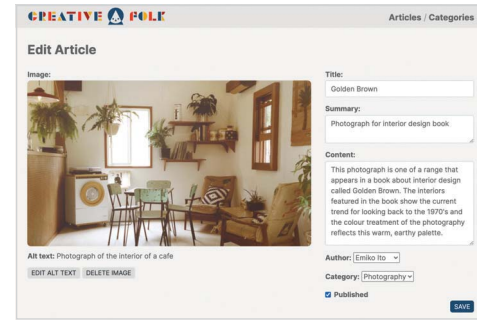
На этой странице пользователю предлагается подтвердить, что он действительно хочет удалить категорию.



Если пользователь нажмет кнопку `Confirm` («Подтвердить»), категория будет удалена и пользователь будет отправлен обратно на страницу `categories.php`. При этом к строке запроса в URL будет добавлено сообщение, которое проинформирует пользователя, что категория была удалена.

ФАЙЛ ARTICLE.PHP

Страница редактирования работ содержит форму для создания новой или редактирования существующей публикации.



При отправке формы данные проходят валидацию.

- Если валидация успешна, код обновит базу данных и отправит пользователя обратно на страницу `articles.php`. В строке запроса при этом передается сообщение, чтобы страница категорий могла сообщить пользователю, что данные были сохранены.
- Если данные не проходят валидацию, форма отображается снова, с сообщениями под теми полями формы, которые необходимо исправить.

ФАЙЛ ARTICLE-DELETE.PHP

На этой странице пользователю предлагается подтвердить, что он действительно хочет удалить публикацию.



При нажатии кнопки `Confirm` публикация будет удалена и пользователь будет отправлен обратно на страницу `articles.php`. При этом к строке запроса в URL будет добавлено сообщение, которое проинформирует пользователя об успешном удалении.

СОЗДАНИЕ, РЕДАКТИРОВАНИЕ И УДАЛЕНИЕ КАТЕГОРИЙ

Страница `categories.php` содержит ссылки на страницы создания, редактирования и удаления категорий.

1. Включение строгой типизации и подключение файлов: `database-connection.php`, создающего объект PDO и `functions.php`, содержащего пользовательские функции, включая функцию `pdo()`, функции для форматирования данных и новую функцию, показанную в шагах 15–19.
2. Если в строке запроса содержится сообщение об успешном выполнении операции, оно сохраняется в переменной с именем `$success`. Если нет, то `$success` будет содержать значение `null`.
3. Если строка запроса содержит сообщение об ошибке, то оно сохраняется в переменной `$failure`. Если нет, то `$failure` получит значение `null`.
4. Переменная `$sql` содержит SQL-запрос для получения данных о каждой из категорий, хранящихся в базе данных.
5. Функция `pdo()` используется для выполнения запроса, а затем метод `fetchAll()` объекта `PDOStatement` получает данные всех категорий и сохраняет их в переменной `$categories`.
6. Подключение файла с верхней частью дизайна сайта для страниц администрирования.
7. Если в строке запроса содержится сообщение о результате выполнения операции, то оно отображается на странице.
8. Вывод ссылки для создания новой категории. Когда в ссылке на файл `category.php` отсутствует строка запроса, это означает что код в `category.php` должен создать новую категорию.
9. Формирование HTML-таблицы, первая строка которой содержит три заголовка столбцов: `Name`, `Edit` и `Delete`.
10. Цикл `foreach` используется для вывода существующих категорий со ссылками для их редактирования или удаления.
11. В первом столбце отображается название категории.
12. Далее создается ссылка на страницу `category.php`. Строка запроса содержит идентификатор категории, поэтому, когда страница `category.php` загружается, пользователь получает возможность редактировать эту категорию. Например, `<a href="category.php?id=1">`.
13. Создается ссылка на страницу `category-delete.php`. Она удаляет категорию из базы данных. Идентификатор категории передается в строке запроса.
14. Подключение файла с нижней частью дизайна страниц администрирования.
15. В файл `functions.php` добавляется новая функция, которая служит для перенаправления пользователя на другую страницу. Она позволяет добавлять сообщения об успехе или ошибке в строку запроса страницы, на которую отправлен пользователь. У нее есть три параметра:

- URL, на который будет отправлен пользователь;
- необязательный массив, используемый для создания строки запроса;
- необязательный код ответа HTTP (по умолчанию 302).

16. В переменную `$qs` пишется значение для строки запроса с помощью тернарного оператора. Если переменная `$parameters` содержит массив, вопросительный знак добавляется к `$qs`, а затем встроенная в функция PHP `http_build_query()` создает строку запроса из значений в массиве. Для каждого элемента в массиве ключ становится именем в строке запроса, а его значение добавляется после символа равенства. Символы, которые не разрешены в URL-адресе, автоматически экранируются.
17. Значение из `$qs` добавляется в конце URL-адреса страницы, на которую отправляется пользователь.
18. Функция PHP `header()` вызывается для перенаправления посетителя. Первый аргумент сообщает браузеру URL для запроса, а второй — это код ответа HTTP.
19. Команда `exit` останавливает дальнейшее выполнение кода.

PHP c13/cms/admin/categories.php

```php
<?php
declare(strict_types = 1);                                  // Строгая типизация
include '../includes/database-connection.php';              // Подключение к БД
include '../includes/functions.php';                        // Подключение функций

$success = $_GET['success'] ?? null;                        // Сообщение об успехе
$failure = $_GET['failure'] ?? null;                        // Сообщение об успехе

$sql = "SELECT id, name, navigation FROM category;";        // SQL для категорий
$categories = pdo($pdo, $sql)->fetchAll();                  // Получение категорий
?>
<?php include '../includes/admin-header.php' ?>
<main class="container" id="content">
  <section class="header">
    <h1>Categories</h1>
    <?php if ($success) { ?><div class="alert alert-success"><?= $success ?></div><?php } ?>
    <?php if ($failure) { ?><div class="alert alert-danger"><?= $failure ?></div><?php } ?>
    <p><a href="category.php" class="btn btn-primary">Add new category</a></p>
  </section>

  <table class="categories">
    <tr><th>Name</th><th class="edit">Edit</th><th class="delete">Delete</th></tr>
    <?php foreach ($categories as $category) { ?>
      <tr>
        <td><?= html_escape($category['name']) ?></td>
        <td><a href="category.php?id=<?= $category['id'] ?>"
               class="btn btn-primary">Edit</a></td>
        <td><a href="category-delete.php?id=<?= $category['id'] ?>"
               class="btn btn-danger">Delete</a></td>
      </tr>
    <?php } ?>
  </table>
</main>
<?php include '../includes/admin-footer.php'; ?>
```

PHP c13/cms/includes/functions.php

```php
function redirect(string $location, array $parameters = [], $response_code = 302)
{
    $qs = $parameters ? '?' . http_build_query($parameters) : '';  // Строка запроса
    $location = $location . $qs;                                   // Новый URL
    header('Location: ' . $location, $response_code);              // Перенаправление
    exit;                                                  // Остановка выполнения кода
}
```

ДОБАВЛЕНИЕ И РЕДАКТИРОВАНИЕ ИНФОРМАЦИИ В БД

Код для создания или редактирования публикаций и категорий разделен на четыре части. Каждая часть содержит набор инструкций `if`, определяющих, какой код должен выполняться.

А: ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

Сначала код на странице подключает нужные файлы, инициализирует переменные, а также определяет, будет ли выполняться создание или обновление данных. Для этого проверяется наличие в строке запроса элемента с названием `id`. Его значение должно представлять собой целое число, отличное от нуля.

- Если нет: требуется добавление новой строки в БД. На этом этапе интерпретатор PHP перейдет к части Б.
- Если да: требуется отредактировать существующую строку данных, и, следовательно, код должен запросить эту строку из БД, чтобы ее можно было редактировать.

Если код не может найти строку, которую пользователь хочет отредактировать, то пользователь получает сообщение, что статья или категория не были найдены.

Б: ПОЛУЧЕНИЕ И ВАЛИДАЦИЯ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ

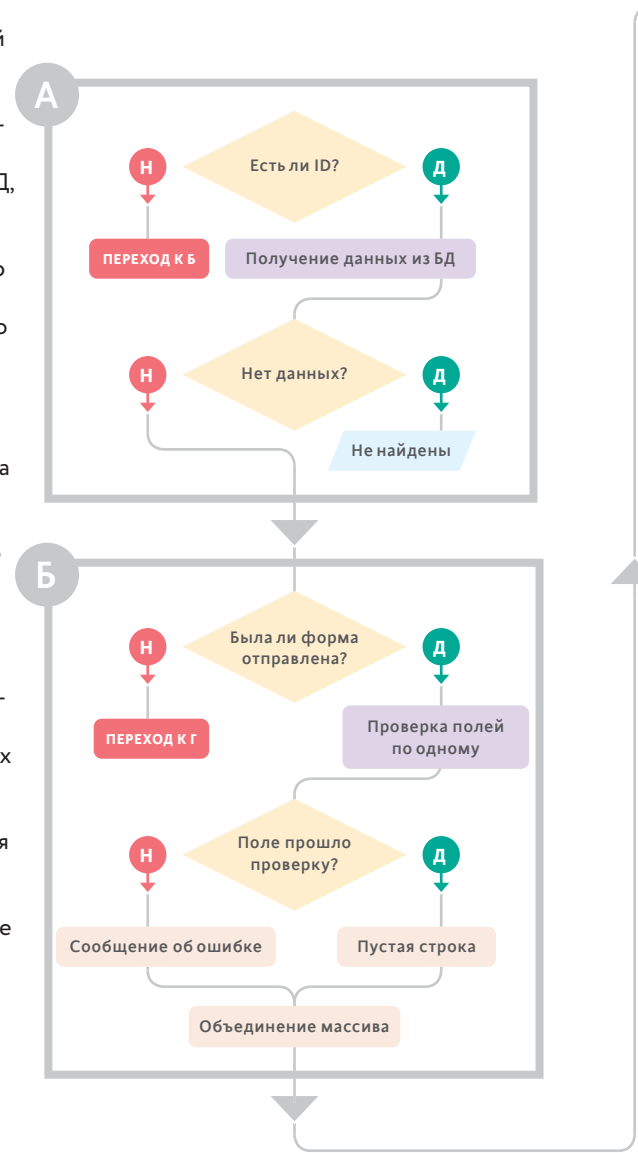
Далее код проверяет, была ли отправлена форма:

- Если нет: переход к части Г.
- Если да: необходимо получить данные и провести валидацию.

Создается массив с результатами проверки для каждого фрагмента данных, переданного на страницу. Значение для каждого элемента присваивается с использованием функций валидации, представленных в главе 6. В соответствующий элемент в массиве ошибок пишется:

- если элемент прошел проверку: пустая строка;
- не прошел проверку: сообщение об ошибке, указывающее, какие данные ожидаются в этом поле ввода.

Затем значения в массиве объединяются в одну строку.

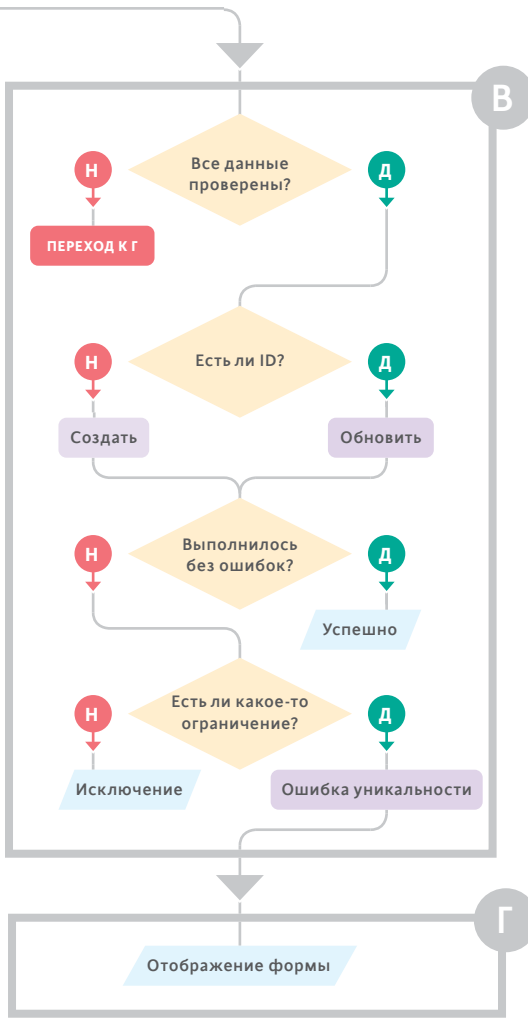


Блок-схемы помогают описать, какой код должен выполняться в различных ситуациях. Вы можете вернуться к этим блок-схемам при работе с кодом.

В: СОХРАНЕНИЕ ВВЕДЕННЫХ ДАННЫХ

В этой части код проверяет, все ли данные прошли проверку.

- Если нет: переход к части Г.
- Если да: продолжение выполнения кода в части В.



Затем он проверяет, был ли передан идентификатор в строке запроса.

- Если нет: выполнить SQL для добавления публикации или категории.
- Если да: выполнить SQL для обновления публикации или категории.

Затем код проверяет, успешно ли выполнен SQL.

- Если нет: проверить тип выброшенного исключения.
- Если да: пользователю будет показано сообщение об успешном завершении.

Если было выброшено исключение, проверить, было ли оно вызвано ограничением уникальности.

- Если нет: повторный выброс исключения.
- Если да: создать сообщение с описанием проблемы.

Г: ОТОБРАЖЕНИЕ ФОРМЫ

Далее будет показана форма.

- Если идентификатор отсутствует и форма не была отправлена, форма будет пустой.
- Если идентификатор есть, но форма не была отправлена, то в форме отображаются существующие данные, подлежащие редактированию.
- Если форма была отправлена, но данные не прошли проверку, то в форме отображаются ранее введенные пользователем данные, а также сообщения об ошибках, указывающие, как их исправить.

ПОЛУЧЕНИЕ И ПРОВЕРКА ДАННЫХ КАТЕГОРИИ

Код для файла `category.php` разделен на следующие шесть страниц. Сначала показан код для частей А и Б (как описано на предыдущей странице). В части А выполняются начальные настройки и определяется, будет ли создаваться новая категория или обновляться уже существующая.

1. Включение строгой типизации и подключение необходимых файлов, `database-connection.php`, `functions.php`, а также `validate.php`, который содержит функции проверки вводимых данных, созданные в главе 6.
2. Если требуется редактирование существующей категории, то URL будет содержать строку запроса с идентификатором редактируемой категории.
Функция PHP `filter_input()` используется для проверки наличия идентификатора и его значения в виде целого числа. Переменная `$id` будет содержать:

- целое число, если оно было передано;
- `false`, если значение не является целым числом;
- `null`, если элемент с именем `id` отсутствует в строке запроса.

3. Объявление массива `$category` с информацией о категории. Он инициализируется значениями, которые будут использоваться в форме в части Г, если нет никаких значений для отображения (т. е. запись не редактируется и не было ошибок при отправке).
4. Инициализация массива `$errors` пустыми строками для каждого элемента, поскольку ошибки еще не обнаружены. Элементы этого массива выводятся в части Г рядом с соответствующими полями ввода в HTML-форме.
5. Проверка с помощью конструкции `if`, было ли введено целое число больше нуля для идентификатора категории в строке запроса.
6. Если было, то в переменную `$sql` пишется SQL для получения из БД информации о категории, которую пользователь хочет отредактировать.
7. Выполнение запроса с помощью функции `pdo()` и получение запрошенных данных с использованием метода `fetch()`. Результат сохраняется в переменной `$category` (перезаписывая значения, созданные в шаге 3).
8. Если в строке запроса был идентификатор, но база данных не нашла соответствующую категорию, переменная `$category` будет содержать значение `false` и запустится последующий блок кода.
9. Функция `redirect()` отправляет пользователя на страницу `categories.php`. Второй аргумент — это массив, используемый для отображения сообщения об ошибке. Это сообщение информирует пользователя, что категория не найдена.

Часть Б получает и проверяет введенные в форму данные.

10. Конструкция `if` проверяет, была ли форма отправлена.
11. Если да, то введенные значения перезаписывают элементы в созданном в шаге 4 массиве `$category`.
Примечание. Параметр `navigation` (который указывает, должна ли категория отображаться на панели навигации) отправляется на сервер только в том случае, если установлен флажок. Поэтому конструкция `isset()` используется для проверки, было ли отправлено значение для этого элемента управления формой, а затем оператор сравнения проверяет, равно ли его значение 1. Если да, то элемент `navigation` будет содержать значение 1; если нет, то он будет содержать 0.
12. Название и описание категории проходят валидацию с помощью функции `is_text()` из подключаемого файла `validate.php`. Если они не проходят проверку, то сообщения об ошибках сохраняются в массиве `$errors`.
13. Значения в массиве `$errors` объединяются вместе, и результат присваивается переменной с именем `$invalid`.

На следующей странице показано, как PHP-код решает, сохранять или не сохранять данные в БД.

PHP c13/cms/admin/category.php

```php
<?php
// Часть А: Настройка
declare(strict_types = 1);                                  // Используем строгую типизацию
include '../includes/database-connection.php';              // Подключение к БД
include '../includes/functions.php';                        // Подключение функций
include '../includes/validate.php';                         // Подключение валидации

// Initialize variables
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Получаем и проверяем id
$category = [
    'id'          => $id,
    'name'        => '',
    'description' => '',
    'navigation'  => false,
];                                                  // Инициализация массива категорий
$errors = [
    'warning'     => '',
    'name'        => '',
    'description' => '',
];                                                  // Инициализация массива ошибок

// Если идентификатор есть, получаем данные для редактирования
if ($id) {                                                  // Если id есть
    $sql = "SELECT id, name, description, navigation
              FROM category
             WHERE id = :id;";                              // SQL
    $category = pdo($pdo, $sql, [$id])->fetch();           // Получаем данные о категории
    if (!$category) {                                       // Если категория не найдена
        redirect('categories.php', ['failure' => 'Category not found']); // Выводим ошибку
    }
}

// Часть Б: Получение и валидация данных формы
if ($_SERVER['REQUEST_METHOD'] == 'POST') {                 // Если форма отправлена
    $category['name']        = $_POST['name'];              // Получение названия
    $category['description'] = $_POST['description'];       // Получение описания
    $category['navigation']  = (isset($_POST['navigation'])
        and ($_POST['navigation'] == 1)) ? 1 : 0;           // Получение навигации

    // Проверка данных и создание сообщений об ошибках
    $errors['name'] = (is_text($category['name'], 1, 24))
        ? '' : 'Name should be 1-24 characters.';           // Проверка названия
    $errors['description'] = (is_text($category['description'], 1, 254))
        ? '' : 'Description should be 1-254 characters.'; // Проверка описания

    $invalid = implode($errors);                            // Объединение ошибок в строку
```

СОХРАНЕНИЕ ДАННЫХ КАТЕГОРИИ

В части B код на странице `category.php` проверяет, следует ли сохранять данные в БД, и если да, следует ли добавить новую категорию или же обновить существующую.

1. Проверка с помощью конструкции `if`, содержит ли переменная `$invalid` какой-либо текст. Если да, то он интерпретируется как значение `true`, указывая на наличие ошибок, которые пользователь должен исправить. В следующей строке кода предупреждение об этом добавляется в массив `$errors`.
2. В противном случае данные прошли проверку и могут быть записаны в БД.
3. Данные из массива `$category` копируются в переменную `$arguments`. Это делается потому, что для вывода в форме и для выполнения запроса нужны немного разные массивы:

- когда PDO выполняет запрос SQL, он использует значения, сохраненные в переменной `$category` для замены плейсхолдеров;
- но для разных запросов нужно разное количество плейсхолдеров, и поэтому для каждого запроса необходимо подготовить свой массив с данными. Если же передать в массиве лишние элементы, то при выполнении запроса будет ошибка (см. шаг 9).

4. Если переменная `$id` содержит отличное от нуля число (которое эквивалентно значению `true`), это означает, что надо обновить существующую категорию.
5. В переменной `$sql` формируется запрос для обновления категории. Она начинается с команды `UPDATE` и имени таблицы для обновления.
6. За оператором `SET` следуют имена обновляемых столбцов и плейсхолдеры, которые будут заменены значениями для этих столбцов.
7. Оператор `WHERE` использует идентификатор строки в таблице категорий, которую надо обновить.
8. Если в переменной `$id` не содержится число, это означает, что новую категорию необходимо добавить в базу данных.
9. Функция PHP `unset()` удаляет элемент, содержащий идентификатор статьи, из массива `$arguments`. Это делается потому, что элементы в массиве с данными для метода `execute()` должны точно совпадать с плейсхолдерами в SQL-запросе.
10. В переменной `$sql` формируется запрос для добавления новой категории. Он начинается с:

- ключевого слова `INSERT`, которое означает добавление новой строки в БД;
- за которым следует команда `INTO` и название таблицы, в которую будут добавлены данные;
- далее в круглых скобках перечисляются имена добавляемых столбцов.

11. После этого идет команда `VALUES` и имена плейсхолдеров, представляющих новые значения, также в круглых скобках.
12. Выполнение запроса заключается внутри блока `try`, поскольку имя категории содержит ограничение уникальности. Если пользователь попытается указать уже используемое имя, будет выброшено исключение.
13. Функция `pdo()` выполняет запрос.
14. Если ошибки не возникло и исключение не было выброшено, то после выполнения запроса вызывается функция `redirect()`. Первый аргумент указывает, что пользователь должен быть отправлен обратно на страницу `categories.php`. Второй аргумент — это массив с сообщением о том, что категория была успешно сохранена: `['success'=> 'Category saved'].` Функция `redirect()` получает все эти данные и указывает браузеру запросить следующую страницу: `category.php?success=Category%20saved.` Перенаправление посетителя на страницу `categories.php` после сохранения категории предотвратит повторную отправку формы, если пользователь решит обновить страницу.

PHP c13/cms/admin/category.php

```
    // Часть B: Проверка данных, в случае успеха — запись в БД
    if ($invalid) {                                          // Если не прошли валидацию
        $errors['warning'] = 'Please correct errors';        // Сообщение об ошибке
    } else {                                                 // В противном случае
        $arguments = $category;                              // Массив аргументов для SQL
        if ($id) {                                           // Если есть идентификатор
            $sql = "UPDATE category
                       SET name = :name, description = :description,
                           navigation = :navigation
                     WHERE id = :id;";                 // Обновление существующей категории
        } else {                                         // If there is no id
            unset($arguments['id']);                     // Remove id from category array
            $sql = "INSERT INTO category (name, description, navigation)
                         VALUES (:name, :description, :navigation);"; // Создание новой
        }                                                             // категории

        // При выполнении SQL-запроса могут произойти три вещи:
        // Категория сохранится | Такое название уже есть | Другая ошибка
        try {                                            // Если предвидим ошибку
            pdo($pdo, $sql, $arguments);                 // Выполнение SQL
            redirect('categories.php', ['success' => 'Category saved']); // Редирект
        } catch (PDOException $e) {                  // Если было поймано исключение PDO
            if ($e->errorInfo[1] === 1062) {         // Если ошибка дублирования
                $errors['warning'] = 'Category name already in use'; // Сообщение о дубле
            } else {                                 // Иначе непредвиденная ошибка
                throw $e;                            // И повторный выброс исключения
            }
        }
    }
}
?>
```

15. Если данные категории не удалось сохранить, будет выброшено исключение и интерпретатор PHP выполнит код в блоке `catch`.

Цель этого кода — проверить, не было ли выброшено исключение из-за того, что имя категории оказалось неуникальным. Внутри блока `catch` объект исключения будет сохранен в переменной с именем `$e`.

16. Условие в конструкции `if` проверяет свойство `errorInfo` объекта исключения, содержащее индексированный массив. Код ошибки находится в элементе, ключ которого равен 1. Если код ошибки 1062, это указывает на то, что ограничение уникальности было нарушено и такое название категории уже есть.

17. Сообщение об ошибке сохраняется в массиве `$errors`, информируя пользователя, что название категории уже используется.

18. Если объект исключения содержит другой код ошибки, исключение выбрасывается повторно, чтобы быть обработанным функцией обработки исключений по умолчанию.

ФОРМА ДЛЯ СОЗДАНИЯ ИЛИ РЕДАКТИРОВАНИЯ ДАННЫХ КАТЕГОРИИ

Часть Г процесса включает в себя показ посетителю формы, которую он может использовать для создания или редактирования информации о категории. При создании и редактировании категории пользователю отображается одна и та же форма.

1. Атрибут `action` открывающего тега `<form>` указывает на ту же страницу (`category.php`). Строка запроса содержит элемент с именем `id`, значение которого выводится из переменной `$id`. Если это редактирование категории, то в этой переменной будет ее идентификатор. Если нет, то в ней будет значение `null` (то есть в этом случае ничего не выведется). Форма отправляется с использованием HTTP-метода POST.
2. Если форма была отправлена, а данные не прошли проверку (или название категории уже использовалось), то в массиве `$errors` будет присутствовать элемент с ключом `warning`, в котором содержится сообщение об ошибке. В конструкции `if` проверяется наличие этого сообщения.
3. Если оно есть, то будет выведено в форме на самом верху.
4. Текстовое поле ввода позволяет пользователю ввести новое или отредактировать существующее название категории. С помощью атрибута `value` выводится соответствующий элемент массива `$category`, то есть если он не пустой, то будет выведен в этом поле.

Функция `html_escape()` следит за тем, чтобы все зарезервированные символы HTML были заменены соответствующими HTML-сущностями. Это предотвращает ошибки и XSS-атаки.

Когда страница впервые загружается для создания новой категории, никакой информации о категории в скрипте еще нет. Вот почему было нужно инициализировать массив `$category` пустыми значениями в части А (задав все нужные ключи массива и присвоив им пустые строки в качестве значений): чтобы при обращении к элементам массива они заведомо существовали и не вызывали ошибки.
5. Массив `$errors` также был инициализирован в части А. В нем есть элементы для каждого введенного значения. Если форма была отправлена, а название категории не прошло проверку, то значение, связанное с ключом `name`, будет содержать сообщение об ошибке, описывающее проблему, и оно будет показано под полем ввода.

Если форма не была отправлена или ошибок не было, элемент с сообщением об ошибке будет содержать пустую строку (поскольку она была присвоена в части А) и, соответственно, под полем ввода ничего не будет выведено. Если бы массив `$errors` не был инициализирован в части А, попытка вывести сообщение об ошибке привела бы к ошибке `Undefined index`.

6. Элемент ввода `<textarea>` позволяет пользователю ввести описание для категории. Если оно не пустое (т. е. это либо редактирование существующей категории, либо повторный вывод формы при ошибке), то отображается между открывающим и закрывающим тегами.
7. Если описание не прошло проверку, под ним отображается сообщение об ошибке.
8. Флажок определяет, должно ли название категории отображаться в навигации.
9. Тернарный оператор проверяет, присвоено ли параметру `navigation` значение 1. Если это так, он добавляет атрибут `checked` к полю ввода флажка, чтобы в форме он отобразился как отмеченный. Если нет, то вместо этого будет выведена пустая строка.
10. В конце формы добавляется кнопка отправки формы.

PHP c13/cms/admin/category.php

```
<?php include 'includes/admin-header.php'; ?>
  <main class="container admin" id="content">
(1)   <form action="category.php?id=<?= $id ?>" method="post" class="narrow">

      <h2>Edit Category</h2>
(2)   <?php if ($errors['warning']) { ?>
(3)     <div class="alert alert-danger"><?= $errors['warning'] ?></div>
      <?php } ?>

      <div class="form-group">
        <label for="name">Name: </label>
        <input type="text" name="name" id="name"
(4)            value="<?= html_escape($category['name']) ?>" class="form-control">
(5)     <span class="errors"><?= $errors['name'] ?></span>
      </div>

      <div class="form-group">
        <label for="description">Description: </label>
        <textarea name="description" id="description" class="form-control">
(6)        <?= html_escape($category['description']) ?></textarea>
(7)     <span class="errors"><?= $errors['description'] ?></span>
      </div>

      <div class="form-check">
(8)     <input type="checkbox" name="navigation" id="navigation"
              value="1" class="form-check-input"
(9)           <?= ($category['navigation'] === 1) ? 'checked' : '' ?>>
        <label class="form-check-label" for="navigation">Navigation</label>
      </div>

(10)  <input type="submit" value="save" class="btn btn-primary btn-save">

    </form>
  </main>
<?php include 'includes/admin-footer.php'; ?>
```

УДАЛЕНИЕ КАТЕГОРИИ

Когда пользователь нажимает на ссылку, чтобы удалить категорию, PHP-код получает название категории из базы данных и отображает его пользователю. Также показывается запрос на подтверждение удаления (это предотвращает удаление при случайном нажатии на удаляющую категорию ссылку).

Если пользователь подтверждает, что он хочет удалить категорию, страница перезагружается и происходит удаление выбранной категории.

Если удаление проходит нормально, пользователь будет отправлен на страницу `categories.php` и получит сообщение о том, что операция удаления прошла успешно.

1. Включение строгой типизации и подключение нужных файлов.
2. Функция `filter_input()` проверяет наличие в строке запроса элемента с именем `id`. Если он присутствует и содержит допустимое целое число, то оно сохраняется в переменной `$id`. Если значение не является допустимым целым числом, в переменной сохраняется значение `false`. Если такой элемент отсутствует, то сохраняется значение `null`.
3. Переменная `$category` будет содержать имя категории, а пока она инициализируется пустой строкой.
4. Конструкция `if` проверяет, что значение в переменной `$id` не эквивалентно `true` (то есть в шаге 2 для него было установлено значение `false`, 0 или `null`). Если это так, пользователь отправляется на страницу `categories.php` с сообщением о том, что категория не найдена, и выполнение кода останавливается.
5. Если код продолжает выполнение, в переменную `$sql` записывается SQL-запрос для получения имени категории.
6. Функция `pdo()` выполняет SQL-запрос, а метод `fetchColumn()` получает единственную ячейку запрошенной строки таблицы в виде скалярного значения. Если категория была найдена, то в переменной `$category` сохраняется ее название. Если нет, то сохраняется значение `false`.
7. Конструкция `if` проверяет, содержится ли в переменной `$category` значение, эквивалентное `false`. Если да, пользователь отправляется на страницу `categories.php` с сообщением о том, что категория не найдена.
8. Если форма была отправлена…
9. Блок `try` создается, чтобы поймать возможную ошибку удаления категории.
10. SQL-код для удаления категории сохраняется в переменной `$sql`.
11. Функция `pdo()` используется для выполнения запроса.
12. Если SQL-запрос выполнился без ошибок, функция `redirect()` используется для отправки пользователя на страницу `categories.php` с сообщением, подтверждающим удаление категории.
13. Если при выполнении запроса было выброшено исключение, выполняется блок `catch`.
14. Если код ошибки равен 1451, это означает, что ограничение целостности не дает удалить категорию (поскольку в ней все еще содержатся публикации).
15. В этом случае функция `redirect()` используется для отправки посетителя на страницу `categories.php` с сообщением об ошибке. Оно информирует пользователя, что категория содержит публикации, которые должны быть помещены в другую категорию или удалены, прежде чем можно будет удалить категорию.
16. В противном случае ошибка будет выброшена повторно и она будет обработана обработчиком исключений по умолчанию.
17. Вывод формы для отображения названия категории и запроса на подтверждение удаления. Атрибут `action` в теге `<form>` отправляет форму на страницу `category-delete.php`. Строка запроса содержит идентификатор категории для удаления.
18. Вывод названия категории для удаления.
19. Кнопка отправки формы для подтверждения удаления.

PHP c13/cms/admin/category-delete.php

```php
<?php
declare(strict_types = 1);                                  // Включение строгой типизации
include '../includes/database-connection.php';              // Подключение к базе данных
include '../includes/functions.php';        // Подключение файла с определениями функций

$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Получение и валидация id
$category = '';                                     // Инициализация названия категории

if (!$id) {                                         // Если нет допустимого идентификатора
    redirect('categories.php', ['failure' => 'Category not found']); // Перенаправление
}                                                                    //  + ошибка

$sql = "SELECT name FROM category WHERE id = :id;";       // SQL для названия
категории
$category = pdo($pdo, $sql, [$id])->fetchColumn();        // Получение названия
категории
if (!$category) {                                         // Если категория не найдена
    redirect('categories.php', ['failure' => 'Category not found']); // Перенаправление
}                                                                    //  + ошибка

if ($_SERVER['REQUEST_METHOD'] == 'POST') {               // Если форма была отправлена
    try {                                                 // Если предвидим ошибку
        $sql = "DELETE FROM category WHERE id = :id;";    // SQL удаления категории
        pdo($pdo, $sql, [$id]);                           // Выполнение удаления
        redirect('categories.php', ['success' => 'Category deleted']); // Перенаправление
    } catch (PDOException $e) {                           // Ловим исключение
        if ($e->errorInfo[1] === 1451) {            // Если в категории есть публикации
            redirect('categories.php', ['failure' => 'Category contains articles that
            must be moved or deleted before you can delete it']); // Перенаправление
        } else {                                          // В противном случае
            throw $e;                                     // Повторный выброс исключения
        }
    }
}?>
<?php include 'includes/admin-header.php'; ?>

  <main class="container admin" id="content">
    <h2>Delete Category</h2>
    <form action="category-delete.php?id=<?= $id ?>" method="POST" class="narrow">
      <p>Click confirm to delete the category <?= html_escape($category) ?></p>
      <input type="submit" name="delete" value="confirm" class="btn btn-primary">
      <a href="categories.php" class="btn btn-danger">cancel</a>
    </form>
  </main>

<?php include 'includes/admin-footer.php'; ?>
```

СОЗДАНИЕ И РЕДАКТИРОВАНИЕ ПУБЛИКАЦИЙ

Служащий для создания и редактирования публикаций файл `article.php` использует тот же порядок выполнения, что и файл `category.php`. Но он должен получать больше данных, а также давать возможность пользователям загружать изображения, что добавляет сложности.

Файл `article.php` позволяет:

- создавать или редактировать описание публикации;
- загружать изображения для публикаций.

Эти операции сложнее, чем у файла `category.php`, поскольку:

- для каждой публикации хранится больше данных, поэтому требуется получить и проверить больше элементов;
- публикация связана с данными в других таблицах (`category` — категория и `member` — пользователь, разместивший публикацию);
- пользователь может дополнительно загрузить изображение для публикации и его описание.

Чтобы сохранить публикацию и ее изображение в базе данных, PHP-код должен работать с таблицами `article` и `image`.

Поскольку SQL может одновременно вставлять новые строки только в одну таблицу, то при выполнении запросов SQL для создания и редактирования статьи, будут использоваться **транзакции**.

Транзакция позволяет базе данных гарантировать, что все запросы, изменяющие данные в БД в рамках одной и той же транзакции, выполнились успешно. База данных может отложить реальное сохранение результатов запросов, и записать их на диск только в том случае, если все запросы выполнились без ошибок. Если же внутри транзакции возникает проблема хотя бы с одним из запросов, то можно отменить запись на диск, и ни одно из изменений не будет сохранено.

Как был показано ранее, порядок выполнения определяет, в какой последовательности должны выполняться отдельные участки кода. Возможность загружать изображения может значительно усложнить порядок выполнения. Представьте, что пользователь уже загрузил публикацию и изображение. Возможно, он захочет отредактировать эти данные и в частности:

- обновить статью, но оставить прежнее изображение. Для этого надо просто обновить таблицу `article`;
- обновить статью и изображение. Эта операция предполагает сначала удаление старого файла с изображением и соответствующих данных из таблиц `image` и `article`, затем загрузку нового файла изображения, а после — обновление таблиц `article` и `image` с измененными данными;
- просто изменить описание изображения, ничего больше. Это означает простое обновление таблицы `image`.

Чем больше вариантов действий у пользователя на одной странице, тем сложнее порядок выполнения и тем сложнее становится код.

Чтобы порядок управления не стал слишком сложным, вы можете ограничить количество доступных пользователю на одной странице действий. Например, требовать удаления изображения перед загрузкой нового или создать отдельную страницу для редактирования описания изображения.

При создании публикации пользователи могут одновременно добавлять изображение и текст.

Примечание. Большинство браузеров не позволяют серверному коду подставить значение в поле для загрузки HTML-файла. Поэтому, если форма отправлена, но не прошла проверку, пользователю необходимо будет снова выбрать изображение.

Когда пользователь загружает новое изображение, он также должен будет повторно вводить описание изображения.

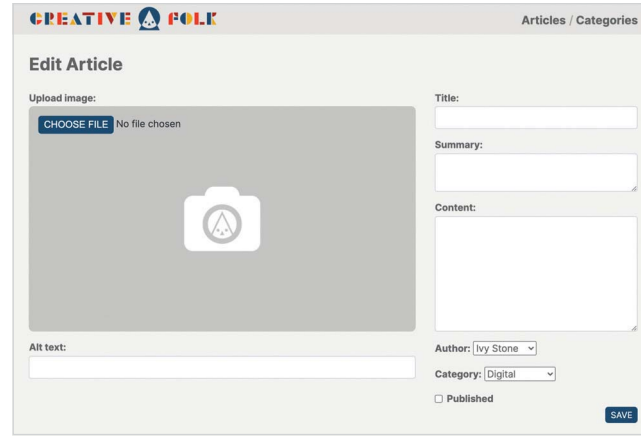
При редактировании публикации:

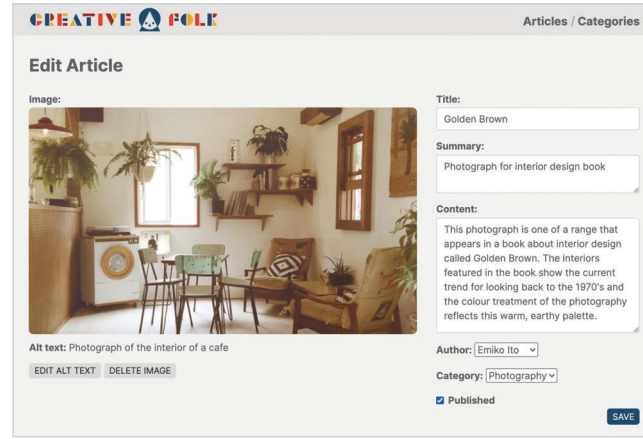
- если пользователь не загрузил изображение, отображаются поля ввода, позволяющие загрузить изображение (как показано на иллюстрации выше);
- если изображение уже загружено, то оно будет отображено в форме вместе с описанием вместо полей ввода.

После загрузки изображения под ним появляются две ссылки, позволяющие пользователям:

- редактировать описание изображения;
- удалять изображение и текст описания.

Страница `articles.php` служит для вывода списка публикаций. Поскольку она работает так же, как страница `categories.php`, ее код здесь не приводится (но его можно посмотреть в загружаемых файлах).





Следующие две страницы познакомят вас с используемыми на странице `article.php` транзакциями.

Позже нам потребуется восемь страниц, чтобы показать, как работает `article.php`, поскольку она содержит более 230 строк кода.

ТРАНЗАКЦИИ: НЕСКОЛЬКО SQL-ЗАПРОСОВ ЗА РАЗ

Транзакция используется для группировки нескольких запросов SQL. Если все запросы выполнены успешно, изменения можно сохранить в базе данных. Если какой-то из них завершится неудачей, то все изменения можно будет откатить, и в базе данных не сохранится результат ни одного запроса.

Для некоторых задач требуется более одного SQL-запроса. Например, когда для публикации загружается изображение, PHP-код должен:

- добавить данные изображения в таблицу `image`;
- получить идентификатор, созданный базой данных для этого изображения (он создается с помощью механизма автоинкремента);
- добавить полученный идентификатор в таблицу `article` в столбец `image_id`.

Кроме того, одним запросом можно вставить данные только в одну таблицу. Следовательно, при добавлении новой статьи понадобится два запроса, добавляющие по новой строке в таблицы `image` и `article`.

Каждый из этих запросов является отдельной **операцией**. Транзакция представляет собой задачу, которая может включать в себя несколько операций с базой данных.

Включив в транзакцию более одной операции, мы можем гарантировать, что все запросы были выполнены успешно:

- если они не вызывали исключения, БД может **зафиксировать (commit)** эти изменения в базе данных;
- если возникнет проблема с выполнением любого из SQL-запросов, PDO выдаст исключение. И в этом случае мы можем дать базе данных команду отменить все изменения, сделанные в ходе транзакции. Это называется **откатом (rollback)** транзакции.

Транзакции удобно выполнять с использованием блоков `try` и `catch`:

- блок `try` содержит код, ошибку в котором вы захотите обработать;
- блок `catch` используется для обработки исключения, если оно возникает в блоке `try`.

Код внутри блока `try` использует PDO для того, чтобы:

- начать транзакцию;
- выполнить все SQL-запросы, входящие в транзакцию;
- зафиксировать изменения в базе данных.

Если выполнение любого из запросов приведет к исключению, интерпретатор PHP немедленно прервет выполнение кода в блоке `try` и передаст выполнение блоку `catch`.

Блок `catch` должен:

- с помощью `PDO` откатить транзакцию, чтобы отменились все сделанные изменения и БД содержала ровно те же данные, что и до запуска транзакции;
- повторно выбросить исключение, чтобы оно могло быть перехвачено функцией обработки исключений по умолчанию.

Это гарантирует, что либо все SQL-запросы будут выполнены, либо, если в блоке `try` возникнет исключение, ни одно из изменений не будет сохранено.

Объект PDO содержит три метода, которые позволяют вам начать транзакцию, зафиксировать изменения в базе данных или откатить их, чтобы база данных содержала те же данные, что и до внесения изменений.

Транзакции используют три метода объекта PDO, показанные в таблице справа.

Первые два используются в блоке `try`, третий используется в блоке `catch`.

МЕТОД	ОПИСАНИЕ
`beginTransaction()`	Начинает транзакцию
`commit()`	Сохраняет изменения в базе данных
`rollBack()`	Отменяет все изменения в рамках транзакции

1. Блок `try` содержит код для выполнения всех SQL-запросов в одной транзакции.

2. Первая инструкция вызывает метод `beginTransaction()` объекта PDO для запуска транзакции.

3. За этим следует код для выполнения запросов SQL, участвующих в транзакции.

4. Последний оператор в блоке `try` вызывает метод `commit()` объекта PDO для сохранения изменений.

5. Если во время выполнения любого из запросов выбрасывается исключение, код в блоке `try` прекращает выполнение, а последующий блок `catch` обрабатывает исключение[62].

6. Метод `rollBack()` объекта PDO вызывается, чтобы отменить все изменения в БД, сделанные в блоке `try`.

7. Объект исключения создается повторно, чтобы его можно было поймать функцией обработчика исключений по умолчанию.

```
1 try {
2     $pdo->beginTransaction();
3     // Здесь выполняются SQL-запросы
4     $pdo->commit();
5 } catch (PDOException $e) {
6     $pdo->rollBack();
7     throw $e;
  }
```

ПУБЛИКАЦИИ: ПОДГОТОВИТЕЛЬНЫЙ ЭТАП (ЧАСТЬ А)

В коде скрипта `article.php` используется такой же порядок управления, как и в файле `category.php`:

- Если в строке запроса есть элемент с названием `id`, который содержит число, то в нем передается идентификатор статьи, которую необходимо отредактировать.
- Если такого элемента нет, это означает, что будет создаваться новая статья.

1. Включение строгой типизации и подключение файлов.

2. В переменную `$uploads` пишется путь к папке загрузок. Разрешенные типы изображений записываются в `$file_types`, а допустимые расширения файлов — в `$file_exts`. Максимальный размер файла изображения пишется в `$max_size`.

3. Функция `filter_input()` проверяет наличие в строке запроса элемента с именем `id`. Если он присутствует и содержит допустимое целое число, то оно сохраняется в переменной `$id`. Если значение не является допустимым целым числом, в переменной сохраняется значение `false`. Если такой элемент отсутствует, то сохраняется значение `null`.

4. Переменная `$temp` будет служить для проверки, был ли загружен файл. Ей присваивается либо временное местоположение файла, либо — если файл не был загружен — пустая строка, благодаря оператору объединения с `null`.

5. Инициализация переменной `$destination`. В дальнейшем, если изображение будет загружено, этой переменной будет присвоен путь файлу, в котором оно будет сохранено.

6. Объявление и инициализация массива `$article` значениями по умолчанию, которые будут использоваться в части Г, когда форма еще не заполнена (то есть в случае, когда добавляется новая статья и форму еще не отправляли на сервер).

Чтобы пример кода поместился на страницу справа, в некоторых массивах на каждой строке кода объявляется сразу по несколько элементов. Этот же файл в загружаемом коде оформлен стандартно — каждый элемент находится на новой строке.

7. Массив `$errors` инициализируется пустыми строками. Элементы этого массива отображаются под каждым полем ввода в форме.

8. Конструкция `if` проверяет, был ли передан валидный идентификатор в строке запроса. Если да, то требуется редактирование существующей статьи, данные которой должны быть получены из базы данных.

9. Переменная `$sql` с SQL-запросом для получения данных статьи.

10. Функция `pdo()` выполняет SQL-запрос, а затем метод `fetch()` объекта `PDOStatement` возвращает запрошенную строку из БД. Значения, которые он возвращает, перезаписывают те, которые были добавлены в массив `$article` на шаге 6. Если публикация не найдена, `$article` будет содержать значение `false`.

11. Если переменная `$article` содержит значение `false`, пользователь перенаправляется на страницу `articles.php` с сообщением об ошибке, в котором говорится, что публикация не может быть найдена.

12. Переменной `$saved_image` присваивается значение в зависимости от того, присутствует ли в существующей публикации изображение. Если да, то в массиве `$article`, полученном из БД, элемент `image_file` будет содержать непустое значение и переменной `$saved_image` будет присвоено значение `true`. Если изображения нет, ей присваивается значение `false`.

13. Из базы данных необходимо получить информацию обо всех категориях и авторах. Эти данные используются для создания выпадающих списков при выборе автора и категории, а также при проверке полученных значений. Сначала переменной `$sql` присваивается SQL-запрос для получения идентификатора, имени и фамилии каждого пользователя.

14. Функция `pdo()` выполняет запрос, а метод `fetchAll()` объекта `PDOStatement` возвращает массив всех строк с результатом запроса, который сохраняется в переменной `$authors`. Возвращается индексированный массив, каждый элемент которого

представляет собой ассоциативный массив сведений об пользователе.

15. Далее переменной `$sql` присваивается SQL-запрос для получения идентификаторов и названий для всех категорий.

16. Функция `pdo()` выполняет запрос, а затем метод `fetchAll()` объекта `PDOStatement` получает данные категорий. Они сохраняются в переменной `$categories`.

PHP c13/cms/admin/article.php

```php
<?php
// Часть A: Настройка
declare(strict_types = 1);                          // Использование строгой типизации
include '../includes/database-connection.php';      // Подключение к БД
include '../includes/functions.php';                // Функции
include '../includes/validate.php';                 // Функции валидации
$uploads = dirname(__DIR__, 1) . DIRECTORY_SEPARATOR . 'uploads' . DIRECTORY_SEPARATOR;
$file_types = ['image/jpeg', 'image/png', 'image/gif',]; // Разрешенные MIME-типы
$file_exts = ['jpg', 'jpeg', 'png', 'gif',];        // Разрешенные расширения
$max_size  = 5242880;                               // Максимальный размер файла
// Инициализация переменных для PHP-кода
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Получение и проверка ID
$temp  = $_FILES['image']['tmp_name'] ?? '';        // Временный файл изображения
$destination = '';                                  // Где сохранить файл
// Инициализация переменных для вывода в HTML-части
$article = [
    'id'          => $id,  'title'       => '',
    'summary'     => '',   'content'     => '',
    'member_id'   => 0,    'category_id' => 0,
    'image_id'    => null, 'published'   => false,
    'image_file'  => '',   'image_alt'   => '',
];                                                  // Информация о публикации
$errors  = [
    'warning' => '', 'title'    => '', 'summary'    => '', 'content'   => '',
    'author'  => '', 'category' => '', 'image_file' => '', 'image_alt' => '',
];                                                  // Сообщения об ошибках
// Если идентификатор существует, это редактирование и надо получить данные
if ($id) {                                          // Если id существует
    $sql     = "SELECT a.id, a.title, a.summary, a.content,
                       a.category_id, a.member_id, a.image_id, a.published,
                       i.file     AS image_file,
                       i.alt      AS image_alt
                  FROM article    AS a
                  LEFT JOIN image AS i ON a.image_id = i.id
                 WHERE a.id = :id;";                // SQL-код для получения статьи
    $article = pdo($pdo, $sql, [$id])->fetch();     // Получение результата
    if (!$article) {                                // Если такой публикации нет
        redirect('articles.php', ['failure' => 'Article not found']); // Перенаправление
    }
}
$saved_image = $article['image_file'] ? true : false;       // Есть ли изображение
// Получение всех пользователей и категорий
$sql = "SELECT id, forename, surname FROM member;";  // SQL для пользователей
$authors = pdo($pdo, $sql)->fetchAll();             // Получение пользователей
$sql  = "SELECT id, name FROM category;";           // SQL для категорий
$categories = pdo($pdo, $sql)->fetchAll();          // Получение категорий
```

ПУБЛИКАЦИИ: ПОЛУЧЕНИЕ И ВАЛИДАЦИЯ ДАННЫХ (ЧАСТЬ Б)

На этих двух страницах показан код для части Б, в которой производится получение и валидация отправленных пользователем данных.

1. Конструкция `if` проверяет, была ли форма отправлена.

2. С помощью тернарного оператора в массив `$error` добавляется элемент `image_file`. Если файл не удалось загрузить (например, потому что он был больше максимального размера, заданного в `php.ini` или `.htaccess`, добавляется сообщение об ошибке, в противном случае — пустая строка.

3. Конструкция `if` проверяет, был ли файл загружен и не было ли ошибок. Если это так, то факт загрузки проходит валидацию, но далее необходимо также проверить и сам загруженный файл.

4. Поскольку файл был загружен, его описание принимается из формы и сохраняется в массиве `$article`.

5. Если MIME-тип изображения входит в число допустимых типов (заданных в шаге 2 на предыдущей странице), к значению в элементе `image_file` массива `$errors` добавляется пустая строка. Если нет, то будет добавлено сообщение об ошибке.

6. Если расширение файла входит в список разрешенных (задан в шаге 2 на предыдущей странице), к значению в элементе `image_file` массива `$errors` добавляется пустая строка. Если нет, то будет добавлено сообщение об ошибке.

7. Если размер файла превышает максимально разрешенный (установленный в шаге 2 на предыдущей странице), к значению в `image_file` добавляется сообщение о том, что файл слишком большой.

8. Элемент с ключом `image_alt` добавляется в массив `$errors`. Если длина текста описания составляет от 1 до 254 символов, то сохраняется пустая строка. Если нет, то сохраняется сообщение об ошибке.

9. Конструкция `if` проверяет, пусты ли оба ключа `image_file` и `image_alt` массива `$errors`. Если это так, то изображение может быть обработано.

10. Функция `create_filename()` (находится в файле `functions.php`) удаляет нежелательные символы из имени файла и гарантирует, что оно уникально. Полученное имя файла сохранится в элементе `image_file` массива `$article`.

11. В переменной `$desination` сохраняется путь к месту загрузки изображения. Он создается путем объединения пути к папке `uploads` (сохраненного в переменной `$uploads` в шаге 2 на предыдущей странице) с именем файла, созданным на предыдущем шаге.

12. Получение данных о публикации из формы. Если это редактирование, то данные из формы заменяют значения, полученные из базы данных в шагах 9–10 на предыдущей странице.

13. Параметр, отвечающий за показ публикации на сайте, — это флажок. Он отправляется на сервер только в том случае, если был отмечен. Ему присваивается значение 1, если флажок отмечен, и 0, если нет. Это связано с тем, что 0 и 1 — это значения, используемые в таблице для хранения логических значений `true` и `false`.

14. Каждое текстовое поле проверяется с помощью функции, созданной в главе 6. Если введенное значение проходит проверку, в соответствующем элементе массива `$errors` сохраняется пустая строка. В противном случае в нем сохраняется сообщение об ошибке.

15. Функции `is_member_id()` и `is_category_id()` были добавлены в файл `validate.php`. Они перебирают соответствующие массивы элементов или категорий, полученные в шагах 13–16 на предыдущей странице, чтобы проверить, входят ли в них введенные значения.

16. Значения в массиве `$errors` объединяются в одну строку с помощью PHP-функции `implode()` и сохраняются в переменной под названием `$invalid`. Она будет использоваться для определения, следует сохранять данные в БД или нет.

PHP c13/cms/admin/article.php

```
// Часть Б: Получение и валидация данных формы
if ($_SERVER['REQUEST_METHOD'] == 'POST') {                  // Если форма отправлена
    // Если размер файла превышает ограничение, указанное в файлах php.ini
    // или .htaccess, в массиве $errors сохраняется сообщение об ошибке
    $errors['image_file'] = ($_FILES['image']['error'] === 1) ? 'File too big ' : '';
    // Если изображение было загружено, то валидировать его
    if ($temp and $_FILES['image']['error'] === 0) {                // Если файл загружен
        $article['image_alt'] = $_POST['image_alt'];                // Получить описание
        // Валидация файла изображения
        $errors['image_file'] .= in_array(mime_content_type($temp), $file_types)
            ? '' : 'Wrong file type. ';                             // Проверка MIME-типа
        $ext = strtolower(pathinfo($_FILES['image']['name'], PATHINFO_EXTENSION));
        $errors['image_file'] .= in_array($ext, $file_extensions)
            ? '' : 'Wrong file extension. ';                        // Проверка расширения
        $errors['image_file'] .= ($_FILES['image']['size'] <= $max_size)
            ? '' : 'File too big. ';                                // Проверка размера
        $errors['image_alt']  = (is_text($article['image_alt'], 1, 254))
            ? '' : 'Alt text must be 1-254 characters.';            // Проверка описания
        // Если файл изображения прошел валидацию, задать место для сохранения
        if ($errors['image_file'] === '' and $errors['image_alt'] === '') { // Если ОК
            $article['image_file'] = create_filename($_FILES['image']['name'], $uploads);
            $destination = $uploads . $article['image_file'];       // Пункт назначения
        }
    }

    // Get article data
    $article['title']       = $_POST['title'];                      // Название
    $article['summary']     = $_POST['summary'];                    // Анонс
    $article['content']     = $_POST['content'];                    // Содержание
    $article['member_id']   = $_POST['member_id'];                  // Автор
    $article['category_id'] = $_POST['category_id'];                // Категория
    $article['published']   = (isset($_POST['published'])
        and ($_POST['published'] == 1)) ? 1 : 0;                    // Надо ли показывать?

    // Валидация публикации и создание сообщений об ошибках
    $errors['title']    = is_text($article['title'], 1, 80)
        ? '' : 'Title must be 1-80 characters';
    $errors['summary']  = is_text($article['summary'], 1, 254)
        ? '' : 'Summary must be 1-254 characters';
    $errors['content']  = is_text($article['content'], 1, 100000)
        ? '' : 'Article must be 1-100,000 characters';
    $errors['member']   = is_member_id($article['member_id'], $authors)
        ? '' : 'Please select an author';
    $errors['category'] = is_category_id($article['category_id'], $categories)
        ? '' : 'Please select a category';
    $invalid = implode($errors);                                    // Слияние сообщений об ошибках
```

ПУБЛИКАЦИИ: СОХРАНЕНИЕ ИЗМЕНЕНИЙ (ЧАСТЬ B)

На этих двух страницах показан код для части B.

1. Конструкция `if` проверяет, содержит ли переменная `$invalid` какие-либо сообщения об ошибках. Если да, то в массив `$errors` будет добавлен элемент с сообщением, предлагающим пользователю исправить ошибки формы.

2. Если нет, это значит, что данные прошли валидацию и могут быть сохранены.

3. Данные из массива `$article` копируются в переменную `$arguments`. Функция `pdo()` будет использовать значения из `$arguments`. В HTML-форме в части Г будут использоваться значения из `$article`.

4. Код для записи в БД заключается в блок `try`.

5. Старт транзакции, поскольку для создания или обновления статьи необходимо выполнить два SQL-запроса. Если один из них вызовет ошибку, то оба не должны быть выполнены.

6. Если переменная `$destination` содержит значение (шаг 11), изображение было загружено. Запрос для сохранения изображения должен выполняться *первым*, поскольку в таблице публикаций необходимо будет сохранить идентификатор, созданный для изображения.

7. Класс `Imagick` используется для изменения размера и сохранения изображения[63]:

- объект `Imagick` создается для загруженного изображения (путь к которому лежит в переменной `$temp` из шага 4);
- размер меняется до 1200 x 700 пикселей;
- изображение сохраняется по пути, записанному в переменной `$destination`.

8. В переменную `$sql` записывается SQL-запрос для добавления имени файла изображения и текста описания в таблицу `image` базы данных.

9. Функция `pdo()` выполняет запрос. Аргументы (путь к файлу и текст описания) передаются функции `pdo()` в виде индексированного массива.

10. Полученный с помощью метода `lastInsertId()` объекта `PDO` идентификатор изображения сохраняется в элементе `image_id` массива `$arguments` (созданного в шаге 3).

11. Элементы с ключами `image_file` и `image_alt` удаляются из массива `$arguments`, поскольку набор ключей в массиве для `execute()` должен в точности соответствовать набору плейсхолдеров в SQL-запросе, а таких плейсхолдеров в следующем запросе нет.

12. Конструкция `if` проверяет, был ли передан идентификатор публикации. Если это так, то требуется редактирование существующей публикации и в переменную `$sql` пишется запрос для обновления строки в базе данных.

13. Если идентификатора нет, это значит, что создается новая публикация, и элемент с ключом `id` необходимо удалить из массива `$arguments`, а в переменную `$sql` пишется запрос для добавления новой строки в БД.

14. Функция `pdo()` выполняет запрос SQL.

15. Метод `commit()` объекта `PDO` записывает в БД изменения, которые были выполнены в рамках транзакции.

16. Если в этом месте продолжается выполнение кода, это значит, что все запросы успешно выполнены, и метод `redirect()` отправляет пользователя на страницу `articles.php`.

17. Если PDO выбросит исключение в блоке `try`, то управление переходит в блок `catch` и объект исключения будет записан в переменную `$e`.

18. Метод `rollBack()` объекта `PDO` не позволяет базе данных сохранять изменения, сделанные в транзакции.

19. Если изображение было сохранено на сервере в шаге 7, для его удаления используется функция PHP `unlink()`.

20. Если код ошибки объекта `PDOException` равен 1062, это значит, что публикация с таким заголовком уже есть, и в переменную `$errors` добавляется сообщение об этом.

21. В противном случае исключение выбрасывается повторно.

22. Если в редактируемой публикации изначально было изображение, то оно остается в массиве `$article`, в элементе с ключом `image_file`. Если же нет, то в этот элемент пишется пустая строка.

PHP c13/cms/admin/article.php

```
    // Часть B: Валидация данных, в случае успеха — запись в БД
    if ($invalid) {                                          // Если есть ошибки
1       $errors['warning'] = 'Please correct the errors below';  // Сообщить об этом
2   } else {                                          // В противном случае
3       $arguments = $article;                        // Сохранить введенные данные
4       try {                                         // Чтобы обработать возможную ошибку
5           $pdo->beginTransaction();                 // Начало транзакции
6           if ($destination) {                  // Если изображение загружено и проверено
7               $imagick = new \Imagick($temp);              // Создание объекта Imagick
                $imagick->cropThumbnailImage(1200, 700);     // Обрезка изображения
                $imagick->writeImage($destination);          // Сохранение файла
8               $sql = "INSERT INTO image (file, alt)
                        VALUES (:file, :alt);";    // SQL для добавления изображения
9               pdo($pdo, $sql, [$arguments['image_file'], $arguments['image_alt'],]);
10              $arguments['image_id'] = $pdo->lastInsertId();   // Получение его ID
            }
11          unset($arguments['image_file'], $arguments['image_alt']); // Убрать данные
                                                                      // изображения
            if ($id) {
                $sql = "UPDATE article
                           SET title = :title, summary = :summary, content = :content,
12                             category_id = :category_id, member_id = :member_id,
                               image_id = :image_id, published = :published
                         WHERE id = :id;";            // SQL для обновления публикации
            } else {
                unset($arguments['id']);                          // Удаление id
                $sql = "INSERT INTO article (title, summary, content, category_id,
13                             member_id, image_id, published)
                        VALUES (:title, :summary, :content, :category_id, :member_id,
                                :image_id, :published);"; // SQL для создания публикации
            }
14          pdo($pdo, $sql, $arguments);      // Запуск SQL-кода для добавления статьи
15          $pdo->commit();                                       // Фиксация изменений
16          redirect('articles.php', ['success' => 'Article saved']); // Перенаправление
17      } catch (PDOException $e) {    // Если выбрасывается исключение PDOException
18          $pdo->rollBack();                    // Откат изменений SQL
19          if (file_exists($destination)) {     // Если файл изображения существует
                unlink($destination);            // Удалить файл изображения
            } // Если это было ограничение целостности
            if ($e->errorInfo[1] === 1062)) {
20              $errors['warning'] = 'Article title already used'; // Ошибка для показа
                                                                   // в форме
            } else {                                              // В противном случае
21              throw $e;                                  // Повторный выброс исключения
            }
        }
    } // Если загружено новое изображение, но данные недействительны удаляется
      // изображение из массива $article
22  $article['image_file'] = $saved_image ? $article['image_file'] : ''; ...
```

ПУБЛИКАЦИИ: ФОРМА И ВЫВОД СООБЩЕНИЙ (ЧАСТЬ Г)

Одна и та же форма отображается пользователю при создании и редактировании статьи.

Примечание. Код формы в загружаемом коде содержит больше тегов и атрибутов HTML — они требуются для маркировки полей ввода и оформления. Но здесь они удалены из примера справа, чтобы он мог поместиться на одной странице и вы могли бы сосредоточиться на самых важных частях кода.

1. Атрибут `action` открывающего тега `<form>` указывает на страницу `article.php`. В строке запроса передается элемент с названием `id`, значение которого берется из переменной `$id`, созданной в верхней части файла. Если на странице редактируется существующая статья, то в переменной будет ее идентификатор. Данные отправляются с помощью HTTP POST.
2. Если элемент массива `$errors` с ключом `warning` содержит непустое значение, то оно будет показано пользователю.
3. Форма для загрузки изображения выводится только в том случае, если в публикации оно отсутствует.
4. Поле для загрузки файлов позволяет посетителю добавить изображение.
5. Если в массиве `$errors` для этого поля содержится сообщение об ошибке, то оно выводится ниже.
Аналогичные действия выполняются для всех полей ввода, кроме флажка `published`.
6. Текстовое поле ввода позволяет ввести описание для картинки.
7. В противном случае — если в публикации уже есть изображение — оно вместе с описанием выводится на странице.
8. Ниже расположены две ссылки: первая позволяет пользователям редактировать текст описания, вторая — удалить изображение.
9. Название публикации вводится с помощью текстового поля ввода. Если оно уже есть, то выводится с помощью атрибута `value`. Все зарезервированные символы заменяются сущностями, чтобы предотвратить XSS-атаку.
10. Для краткого описания публикации используется элемент `<textarea>`. Если оно уже есть, то выводится между тегами `<textarea>` с использованием функции `html_escape()` для замены зарезервированных символов сущностями.
11. Для ввода основного содержания публикации используется еще один элемент `<textarea>`. Если для него есть значение, оно выводится между тегами `<textarea>`.
12. В поле выбора отображается список всех пользователей, из которых можно выбрать автора.
13. Он создается с использованием цикла `foreach`, перебирающего массив всех пользователей (полученных из БД в части А и сохраненных в переменной с именем `$authors`).
14. Для каждого пользователя добавляется элемент `<option>`.
15. Условие в тернарном операторе проверяет, был ли уже указан автор и совпадает ли его идентификатор с идентификатором пользователя, выводимого в данный момент. Если это так, атрибут `selected` добавляется к тегу `<option>` для текущего пользователя, выделяя его в качестве автора статьи.
16. Вывод имени пользователя.
17. Массив категорий, хранящихся в переменной `$categories`, используется для создания поля выбора категорий.
18. Флажок указывает, должна ли публикация выводиться на сайте. Тернарный оператор проверяет, был ли этот пункт отмечен. Если да, то к элементу добавляется атрибут `checked`.

PHP c13/cms/admin/article.php

```
<!—Часть Г — Вывод формы —>
① <form action="article.php?id=<?= $id ?>" method="post" enctype="multipart/form-data">
    <h2>Edit Articles</h2>
    <?php if ($errors['warning']) { ?>
②     <div class="alert alert-danger"><?= $errors['warning'] ?></div>
    <?php } ?>

③   <?php if (!$article['image_file']) { ?>
④     Upload image: <input type="file" name="image" class="form-control-file" id="image">
⑤     <span class="errors"><?= $errors['image_file'] ?></span>
⑥     Alt text: <input type="text" name="image_alt">
      <span class="errors"><?= $errors['image_alt'] ?></span>
    <?php } else { ?>
     <label>Image:</label> <img src="../uploads/<?= html_escape($article['file']) ?>"
⑦                 alt="<?= html_escape($article['image_alt']) ?>">
      <p class="alt"><strong>Alt text:</strong> <?= html_escape($article['image_alt']) ?></p>
      <a href="alt-text-edit.php?id=<?= $article['id'] ?>">Edit alt text</a>
⑧     <a href="image-delete.php?id=<?= $id ?>">Delete image</a><br><br>
    <?php } ?>

⑨   Title: <input type="text" name="title" value="<?= html_escape($article['title']) ?>">
    <span class="errors"><?= $errors['title'] ?></span>
⑩   Summary: <textarea name="summary"><?= html_escape($article['summary']) ?></textarea>
    <span class="errors"><?= $errors['summary'] ?></span>
⑪   Content: <textarea name="content"><?= html_escape($article['content']) ?></textarea>
    <span class="errors"><?= $errors['content'] ?></span>
⑫   Author: <select name="member_id">
⑬     <?php foreach ($authors as $author) { ?>
⑭       <option value="<?= $author['id'] ?>"
⑮         <?= ($article['author_id'] == $author['id']) ? 'selected' : ''; ?>>
⑯         <?= html_escape($author['forename'] . ' ' . $author['surname']) ?>
        </option>
      <?php } ?></select>
    <span class="errors"><?= $errors['author'] ?></span>
    Category: <select name="category_id">
      <?php foreach ($categories as $category) { ?>
        <option value="<?= $category['id'] ?>"
⑰         <?= ($article['category_id'] == $category['id']) ? 'selected' : ''; ?>>
          <?= html_escape($category['name']) ?>
        </option>
      <?php } ?></select>
    <span class="errors"><?= $errors['category'] ?></span>
    <input type="checkbox" name="published" value="1"
⑱     <?= ($article['published'] == 1) ? 'checked' : '' ?>> Published
    <input type="submit" name="create" value="save" class="btn btn-primary">
  </form>
```

УДАЛЕНИЕ ПУБЛИКАЦИИ

Страница для удаления публикации работает так же, как и страница для удаления категории, с использованием формы для подтверждения, что публикация должна быть удалена.

1. Включение строгой типизации и подключение необходимых файлов.
2. Функция PHP `filter_input()` проверяет наличие элемента с именем `id` в строке запроса. Если он содержит допустимое целое число, оно сохраняется в переменной `$id`. Если в строке запроса содержится недопустимое значение, переменная `$id` сохраняет значение `false`. Если данные отсутствуют, переменная `$id` получает значение `null`.
3. Если идентификатор не содержит допустимое значение, пользователь перенаправляется на страницу `articles.php` с сообщением об ошибке.
4. Переменная `$article` инициализируется значением `false`.
5. Переменная `$sql` содержит SQL-код для получения названия публикации, а также имени файла и идентификатора изображения.
6. Функция `pdo()` выполняет SQL-код, данные о публикации сохраняются в переменной `$article`.
7. Если публикация не найдена, пользователь перенаправляется на страницу `articles.php` с сообщением об ошибке.
8. Конструкция `if` проверяет, была ли отправлена форма (чтобы подтвердить, что статья должна быть удалена).
9. Если форма была отправлена, то выполняется код, используемый для удаления статьи. Он заключается в блок `try`.
10. Старт транзакции, поскольку выполняется несколько запросов SQL.
11. Конструкция `if` проверяет, есть ли в публикации изображение.
12. Если изображение есть, в переменную `$sql` записывается SQL, который установит значение `null` в столбце `image_id` таблицы `article` для этой публикации. Функция `pdo()` выполняет запрос.
13. Далее переменной `$sql` присваивается SQL для удаления изображения из таблицы `image`, и функция `pdo()` выполняет этот запрос.
14. В переменную `$path` записывается путь к изображению.
15. В конструкции `if` используется функция PHP `file_exists()`, чтобы проверить, существует ли файл. Если да, функция PHP `unlink()` удаляет его.
16. Переменная `$sql` получает SQL-код для удаления публикации из таблицы `article`, а функция `pdo()` выполняет его.
17. Если исключение не было выброшено, то вызывается метод `commit()` объекта `PDO` для сохранения всех изменений, внесенных выполненными запросами.
18. Пользователь перенаправляется на страницу `articles.php` с сообщением, в котором говорится об успешном удалении статьи.
19. Если при удалении данных было выброшено исключение, выполняется блок `catch`.
20. Метод `rollBack()` объекта `PDO` откатывает все изменения, сделанные запросами.
21. Исключение повторно выбрасывается, чтобы его можно было уловить с помощью функции обработки исключений по умолчанию.
22. При первой загрузке страницы в форме отображается название статьи и кнопка отправки, подтверждающая, что ее следует удалить. Атрибут `action` тега `<form>` использует идентификатор статьи в строке запроса.

Упражнение. Создайте страницы для удаления изображения из статьи, используя показанный в этом файле подход. Затем создайте страницу для редактирования текста описания изображения.
Решения для обеих задач можно посмотреть в загружаемом коде.

PHP c13/cms/admin/article-delete.php

```php
<?php
declare(strict_types = 1);                                          // Строгая типизация
require_once '../includes/database-connection.php';                 // Подключение к БД
require_once '../includes/functions.php';                           // Функции
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);           // Идентификатор
if (!$id) {                                                        // Если не прошел валидацию
    redirect('articles.php', ['failure' => 'Article not found']); // Редирект с ошибкой
}
$article = false;                                                   // Инициализация $article
$sql = "SELECT a.title, a.image_id, i.file AS image_file FROM article AS a
          LEFT JOIN image AS i ON a.image_id = i.id WHERE a.id = :id;";  // SQL
$article = pdo($pdo, $sql, [$id])->fetch();                         // Получение данных публикации
if (!$article) {                                                    // Если массив $article пуст
    redirect('articles.php', ['failure' => 'Article not found']); // Перенаправление
}
if ($_SERVER['REQUEST_METHOD'] == 'POST') {                         // Если форма была отправлена
    try {                                                           // Попытка удаления
        $pdo->beginTransaction();                                   // Начало транзакции
        if ($image_id) {                                            // Если есть изображение
            $sql = "UPDATE article SET image_id = null WHERE id = :article_id;"; // SQL
            pdo($pdo, $sql, [$id]);                                 // Удалить изображение из статьи
            $sql = "DELETE FROM image WHERE id = :id;"; // SQL для удаления изображения
            pdo($pdo, $sql, [$article['image_id']]);    // Удаление из таблицы image
            $path = '../uploads/' . $article['image_file']; // Путь к файлу
            if (file_exists($path)) {                    // Если файл существует
                $unlink = unlink($path);                 // Удалить файл изображения
            }
        }
        $sql = "DELETE FROM article WHERE id = :id;";   // SQL для удаления публикации
        pdo($pdo, $sql, [$id]);                          // Удаление публикации
        $pdo->commit();                                  // Зафиксировать транзакцию
        redirect('articles.php', ['success' => 'Article deleted']); // Перенаправление
    } catch (PDOException $e) {                         // Если выбрасывается исключение
        $pdo->rollBack();                               // Откат изменений SQL
        throw $e;                                       // Повторный выброс исключения
    }
}
?>
<?php include '../includes/admin-header.php' ?> ...
    <h2>Delete Article</h2>
    <form action="article-delete.php?id=<?= $id ?>" method="POST" class="narrow">
      <p>Click confirm to delete: <i><?= html_escape($article['title']) ?></i></p>
      <input type="submit" name="delete" value="Confirm" class="btn btn-primary">
      <a href="articles.php" class="btn btn-danger">Cancel</a>
    </form> ...
<?php include '../includes/admin-footer.php'; ?>
```

ЗАКЛЮЧЕНИЕ

ИЗМЕНЕНИЕ ИНФОРМАЦИИ В БАЗЕ ДАННЫХ

- Пользовательские данные должны быть получены и проверены перед добавлением в БД.
- Метод `execute()` объекта `PDOStatement` позволяет выполнять подготовленные запросы, которые создают, обновляют или удаляют данные.
- SQL может добавлять новые данные только в одну таблицу за раз.
- Метод `getLastInsertId()` объекта `PDO` возвращает идентификатор новой строки при ее добавлении в базу данных.
- Метод `rowCount()` объекта `PDOStatement` возвращает количество строк, затронутых запросами `INSERT`, `UPDATE` или `DELETE`.
- Транзакция выполняет серию инструкций SQL и позволяет сохранить изменения только в том случае, если вся серия выполнилась без ошибок.
- Если запрос SQL нарушает ограничение уникальности, генерируется объект `PDOException`. Он будет содержать код ошибки, описывающий причину исключения.

Г

РАСШИРЕНИЕ ФУНКЦИОНАЛА И МОДЕРНИЗАЦИЯ УЧЕБНОГО ПРИЛОЖЕНИЯ

В этом, заключительном, разделе показано, как реализовать функции, используемые на многих веб-сайтах. В процессе вы узнаете, как добавлять новый функционал на сайт, а также познакомитесь с некоторыми продвинутыми методами разработки.

Если вы попросите пять PHP-программистов создать один и тот же веб-сайт, то получите пять разных решений. Это связано с тем, что не существует единственно правильного способа создания сайта, зато есть множество способов написать код, выполняющий каждую из связанных с этим задач. Однако существуют рекомендуемые подходы, в том числе по проектированию, приведенные в этом разделе. Они помогут вам при создании собственных проектов.

В главе 14 показано, как перестроить структуру учебного сайта, чтобы получить большую отдачу от использования классов. Опытные разработчики PHP широко используют пользовательские классы для группировки кода, выполняющего набор связанных задач.

В главе 15 показано, как находить и использовать классы PHP, которые были написаны другими разработчиками для выполнения определенных задач, а затем выложены в свободный доступ для использования всем сообществом программистов. Использование этих классов избавит вас от необходимости самостоятельно писать код для выполнения тех же задач.

В главе 16 показано, как добавить пользователям возможность регистрироваться на сайте. Зарегистрированный пользователь может входить в систему, создавать собственные публикации, а также позволяет сайту отображать информацию, предназначенную только для этого пользователя. Вы также узнаете, как ограничить доступ к страницам администрирования, чтобы их могли использовать только те, у кого есть разрешение.

В главе 17 показано, как сделать URL-адреса сайта удобочитаемыми и более подходящими для индексации поисковыми системами. Также будет показано, как разрешить пользователям сайта комментировать публикации и ставить им отметки «нравится».

Каждая из оставшихся глав содержит новую версию учебного сайта. Прежде чем читать дальше, вам нужно понять, как организованы эти файлы, и некоторые ключевые понятия.

АБСОЛЮТНЫЕ И ОТНОСИТЕЛЬНЫЕ ПУТИ

Сначала вы узнаете разницу между абсолютным и относительным путями, а также когда следует использовать каждый из них.

КАК ОРГАНИЗОВАНЫ ФАЙЛЫ

Далее вы узнаете, как были реорганизованы файлы учебного сайта. Все доступные для запроса браузером файлы хранятся в папке, называемой корневым каталогом сайта, но все остальные файлы, обеспечивающие работу сайта, например классы, в целях повышения безопасности хранятся в каталоге, находящемся выше корневого каталога сайта в иерархии папок.

НОВЫЕ ФАЙЛЫ НАСТРОЙКИ

В этом разделе представлены два новых файла, которые обычно используются при создании сайтов:

- `config.php` содержит настройки, изменяемые при установке сайта на новый сервер;
- `bootstrap.php` содержит код, необходимый сайту для запуска (он включается на каждой странице сайта).

КАК ПЕРЕМЕННЫЕ ХРАНЯТ ДАННЫЕ

Эта часть учебника поможет понять, как интерпретатор PHP хранит данные в переменных.

КРОМЕ ТОГО

Одна из важных тем, которые будут повторяться на протяжении последних четырех глав, это то, как программисты используют классы для организации своего кода.

Для формирования запрашиваемых пользователями страниц будут использоваться методы объектов, созданных на основе пользовательских классов.

В каждой из оставшихся глав недостаточно места, чтобы добавить пример каждого файла учебного сайта, поэтому будет полезно, если при работе с остальной частью книги у вас на компьютере будет открыт исходный код из папки с загружаемыми файлами. Помимо ознакомления с кодом, это позволит вам сравнить версии файлов в каждой главе с файлами в предыдущих главах, чтобы увидеть, где они изменились.

В главе 16 вам нужно будет создать новую версию БД с дополнительными таблицами и данными, необходимыми для поддержки новых функций, добавляемых на сайт.
SQL для создания обновленной версии базы данных включен в загружаемый код, и вам будет предложено пересоздать БД в начале главы 16.

АБСОЛЮТНЫЕ И ОТНОСИТЕЛЬНЫЕ ПУТИ

Абсолютный путь описывает точное местоположение файла на компьютере.
Относительный путь описывает расположение одного файла по отношению к другому.

Чтобы понять, как организованы файлы в коде учебного сайта, необходимо прояснить некоторые термины.

- **Путь** (path) указывает местоположение файла или каталога (папки).
- **Корневой каталог** (root directory) — это самая верхняя папка на компьютере.
- **Абсолютный путь** (absolute path) описывает местоположение файла или папки, используя путь от корневого каталога к нему.

На диаграмме на правой странице вы можете увидеть часть загружаемого кода, прилагаемого к этой книге, развернутого на компьютере.

На **Mac** или **Linux** корневой каталог представлен слешем. Далее слеш отделяет каждую папку или файл, поэтому абсолютный путь к файлу `files.php` будет выглядеть так: `/Users/Jon/phpbook/section_b/c05/files.php`.

В **Windows** корневой каталог представляет собой букву диска, за которой следует двоеточие и обратный слеш. Далее обратный слеш разделяет каждую папку или файл, поэтому абсолютный путь к `files.php` будет выглядеть так: `C:\phpbook\section_b\c05\files.php`.

Абсолютные пути точны, но они могут:
- получаться довольно длинными, что требует большого количества вводимого текста;
- изменяться при перемещении файлов на другой компьютер.

Папка, содержащая выполняемый в данный момент файл, называется **текущим рабочим каталогом** (current working directory, CWD). На схеме справа, когда файл `files.php` запущен, текущий рабочий каталог `c05`.

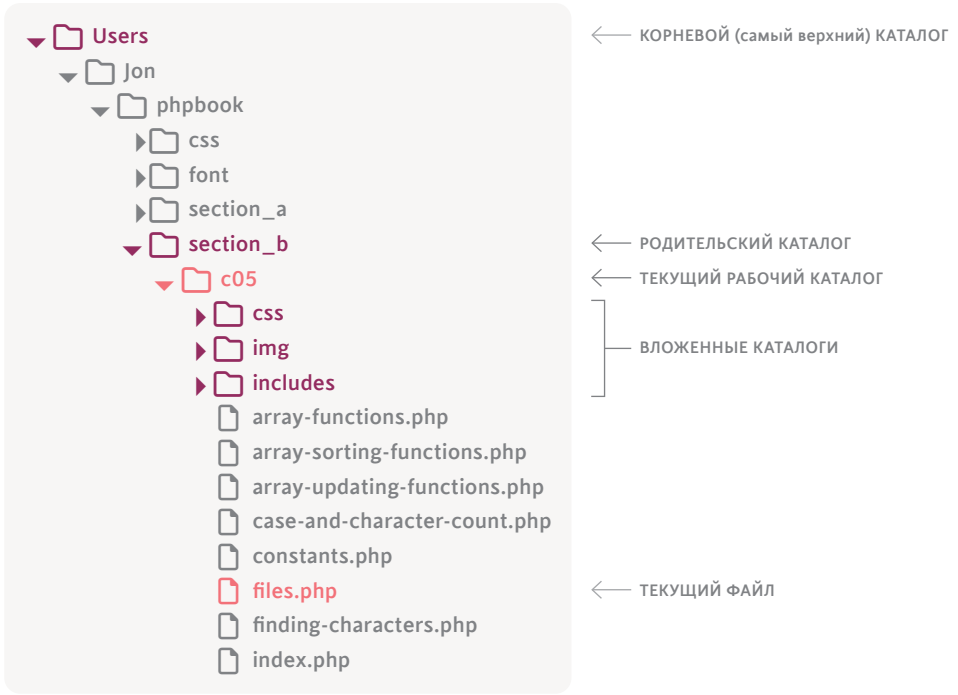
Относительные пути (relative paths) описывают расположение определенного файла по отношению к текущему рабочему каталогу.

Относительный путь к любому файлу в той же папке — это просто имя файла. Например, ниже приводится относительный путь при обращении к файлу `index.php` в папке `c05`: `index.php`.

Чтобы описать путь к файлу в дочернем каталоге, используйте имя дочернего каталога, затем слеш, затем имя файла в этом каталоге. Относительный путь к файлу `header.php` в папке `includes` — это `includes/header.php`.

Чтобы перейти к папке на один уровень выше, используйте «`../`». Относительный путь к файлу с именем `index.php` в каталоге `phpbook` выглядит как `../../index.php`.

Это показывает, что относительные пути требуют меньше ввода текста, чем абсолютные пути. Кроме того, они часто работают, когда сайт перемещается на новый сервер. Например, если пути в загружаемом коде ссылаются только на файлы в папке `phpbook`, относительные пути между файлами и папками не изменяются при запуске кода на разных компьютерах.



Когда в файл PHP подключается другой файл, рекомендуется использовать абсолютные пути. Чтобы понять почему, рассмотрим этот сценарий.

- Выполняется файл PHP в папке `c05`.
- Он подключает PHP-файл из папки `section_a`.
- Этот файл включает третий файл с относительным путем.

Поскольку код из подключаемого файла фактически добавляется в исходный файл, то получается, что в третьем шаге файл из папки `c05` пытается обратиться к файлу, относительный путь к которому был прописан от папки `section_a`. И, соответственно, третий файл не будет найден[64]. Использование абсолютных путей при подключении или чтении файлов позволяет избежать таких ситуаций.

Поскольку абсолютные пути длиннее, требуя вводить больше текста, то на сайте часто первую часть абсолютного пути помещают в константу[65]. Затем эта константа может быть использована для построения абсолютных путей.

Корневой каталог приложения — это самая верхняя папка кода веб-сайта[66]. Абсолютный путь именно к этой папке часто сохраняют в константе, используемой для создания путей к подключаемым файлам. Ниже показано, как путь к папке приложения сохраняется в константе с именем `APP_ROOT`.

- Функция PHP `dirname()` возвращает путь к каталогу, содержащему файл.
- Аргумент — встроенная в PHP константа `__FILE__`, содержащая абсолютный путь к текущему файлу.

Таким образом, этот код сохраняет в константе путь к папке, содержащей текущий файл. Если его использовать в файле внутри папки `c05`, то константа `APP_ROOT` будет содержать абсолютный путь к папке `c05`. Поскольку константа создается интерпретатором PHP динамически при каждом выполнении скрипта, она всегда будет содержать корректный путь, даже при переносе кода на другой сервер (или в другую папку).

```
define('APP_ROOT', dirname(__FILE__));
```

СТРУКТУРА ФАЙЛОВ И КОРНЕВАЯ ПАПКА САЙТА

Все файлы, которые может запросить браузер, должны находиться в папке, называемой **корневой папкой сайта**. Однако интерпретатор PHP имеет доступ и к файлам, расположенным выше корневой папки, поскольку он выполняется на сервере.

У веб-сервера свой корневой каталог, известный как корневая папка сайта (document root) или корень сайта (web-root). Это то место, где пересекаются пути на сайте и пути в файловой системе[67]. Например, если URL-адрес домашней страницы сайта:
`http://example.org/index.php`

Веб-сервер `example.org` будет искать файл `index.php` в корневой папке сайта. На сервере хостинговой компании абсолютный путь к этому файлу может быть, например, таким:

```
/var/www/example.org/htdocs/index.php
                     └──────┘
                КОРНЕВОЙ КАТАЛОГ САЙТА
```

Сама папка, являющаяся корневым каталогом сайта на веб-сервере, может носить разные имена, но обычно это `htdocs`, `public`, `public_html`, `web`, `www` или `wwwroot`.
Каждый запрашиваемый браузером файл должен лежать в корневом каталоге сайта (или в его дочерних папках). Это правило распространяется на запрашиваемые пользователем страницы, изображения или другие медиа, а также файлы CSS и JavaScript. Браузер не может запрашивать файлы выше корневой папки.
Операционные системы Mac и Linux используют слеш для обозначения корневой папки файловой системы (самой верхней папки на компьютере). Точно так же, когда URL-адрес начинается со слеша, это указывает на корень текущего сайта — самая верхняя папка, к которой может получить доступ браузер. Вот почему локальные HTML-ссылки часто начинаются просто со слеша.

Поскольку загружаемый код, прилагаемый к этой книге, содержит отдельную версию сайта для каждой оставшейся главы, то получается, что на одном домене располагается сразу несколько сайтов, каждый со своей корневой папкой. И в итоге папка `public` в примерах кода для каждой из оставшихся глав, хотя и является корневой для своей версии сайта, но по факту — это одна из подпапок настоящего корня сайта (которым является папка `htdocs`). Поэтому в примерах кода используется константа с именем `DOC_ROOT`, в которую пишется путь от настоящего корня сайта, чтобы ее можно было использовать для ссылок на локальные ресурсы в HTML-коде. При реальной эксплуатации (когда на одном домене располагается только один сайт) такая константа обычно не требуется, и в HTML-коде в качестве корневой папки можно использовать просто слеш.

В то время как браузер может запрашивать файлы только из корня сайта, интерпретатор PHP может получить доступ и к файлам, которые располагаются выше в иерархии папок. В следующих главах:

- запрашиваемые браузером по URL-адресу файлы находятся в корне сайта (папка `public`) для каждой главы;
- любой файл, который не требуется запрашивать напрямую (например, файлы с определениями функций или классов), находится в папках выше корня сайта. Это повышает безопасность, поскольку предотвращает доступ пользователей к этим файлам.

В главе 14 папка `c14` будет эквивалентна **корневой папке приложения** (application root). В ней появляются две новые папки, находящиеся внутри корня приложения, но выше корня сайта:

- `config` содержит файл, содержащий любые настройки, которые могут измениться при перемещении сайта на новый сервер;
- `src` содержит файлы с определениями функций и новый файл с именем `bootstrap.php`. У нее также есть дочерняя папка под названием `classes`, содержащая определения классов.



ФАЙЛ КОНФИГУРАЦИИ

Когда сайт устанавливается на новый сервер, то обычно требуется изменить некоторые параметры.

- Могут меняться имена/пути к папкам. Скажем, корневая папка документа может называться htdocs, content или public.
- Сайтам, использующим базы данных, необходимо знать местоположение базы данных, используемое в DSN, а также имя пользователя и пароль учетной записи пользователя базы данных.

Эта информация называется **данными конфигурации**, поскольку они настраивают сайт на работу в определенном окружении. Обычно они все хранятся в переменных или константах в одном файле. Это делает его единственным файлом, который необходимо отредактировать при установке сайта на новый сервер. В коде учебного приложения этот файл называется `config.php`.

Примечание. Вы должны обновлять код в этом файле для каждой из оставшихся глав.

1. Константе `DEV` присваивается значение `true`, если сайт находится в разработке, и `false`, если он запущен в эксплуатацию. Это значение определяет, как обрабатываются ошибки.
2. Константа `DOC_ROOT` содержит путь от реального корня сайта до папки, которая является корнем для текущего сайта.
3. Константа `ROOT_FOLDER` хранит имя корневой папки сайта (например, `public`, `content` или `htdocs`).
4. Параметры подключения к базе данных сохраняются в переменных, а затем создается DSN.
5. Константы, связанные с загрузкой файлов:

- `MEDIA_TYPES` содержит разрешенные типы файлов;
- `FILE_EXTENSIONS` хранит допустимые расширения файлов;
- `MAX_SIZE` — значение максимального размера файла (в килобайтах);
- `UPLOADS` хранит абсолютный путь к папке под названием `uploads`.

section_d/c14/src/config.php **PHP**

```
<?php
define('DEV', true);      // В разработке — true | в эксплуатации — false
define("DOC_ROOT", '/phpbook/section_d/c14/public/');    // Путь от настоящего корня
define("ROOT_FOLDER", 'public');                         // Имя папки с корнем сайта
// Настройки базы данных
$type     = 'mysql';                // Тип базы данных
$server   = 'localhost';            // Сервер, на котором находится база данных
$db       = 'phpbook-1';            // Имя базы данных
$port     = '';             // Значение порта обычно 8889 для MAMP и 3306 для XAMPP
$charset  = 'utf8mb4';      // Кодировка UTF-8 с использованием 4 байт данных на символ
$username = 'ENTER YOUR USERNAME';   // Введите СВОЕ имя пользователя
$password = 'ENTER YOUR PASSWORD';   // Введите СВОЙ пароль
$dsn = "$type:host=$server;dbname=$db;port=$port;charset=$charset"; // НЕ ИЗМЕНЯТЬ
// Настройки загрузки файлов
define('MEDIA_TYPES', ['image/jpeg', 'image/png', 'image/gif',]); // Допустимые типы
define('FILE_EXTENSIONS', ['jpeg', 'jpg', 'png', 'gif',]);    // Допустимые расширения
define('MAX_SIZE', '5242880');                          // Максимальный размер файла
define('UPLOADS', dirname(__DIR__, 1) . DIRECTORY_SEPARATOR . ROOT_FOLDER .
    DIRECTORY_SEPARATOR . 'uploads' . DIRECTORY_SEPARATOR);       // НЕ ИЗМЕНЯТЬ
```

ФАЙЛ НАЧАЛЬНОЙ ЗАГРУЗКИ

В главе 13 каждая страница сайта подключала несколько файлов, содержащих определения функций, создание объекта PDO, и так далее. Чтобы избежать повторения этого кода на каждой странице, теперь каждая страница будет начинаться с подключения единственного файла, который приведен ниже. Он, в свою очередь:

- подключает файлы `config.php` и `functions.php`;
- определяет функции обработки ошибок и исключений, а также новую функцию, которая будет загружать определения классов;
- создает объект `CMS`. Все страницы сайта будут использовать его для работы с базой данных.

Когда один файл загружает другие файлы и создает объекты, необходимые сайту для запуска, ему часто присваивается имя, например `bootstrap.php` или `setup.php`.

После создания объекта CMS переменные с параметрами подключения к БД удаляются, чтобы их нельзя было использовать (или случайно отобразить) где-либо в остальной части сайта.

1. Путь к корневой папке приложения (на два уровня выше этого файла) сохраняется в константе `APP_ROOT`.

2. Подключаются файлы `functions.php` и `config.php`.

3. Функция PHP `spl_autoload_register()`, с которой вы познакомитесь в главе 14, позволяет подключать только те определения классов, которые нужны на этой странице.

4. Конструкция `if` проверяет, находится ли сайт в разработке или работает в режиме эксплуатации. В последнем случае устанавливаются функции обработки исключений и ошибок по умолчанию, а также функция обработки завершения работы.

5. Объект `CMS` создается и сохраняется в переменной с именем `$cms`. Он будет использоваться для работы с базой данных.

6. Функция PHP `unset()` удаляет переменные, содержащие данные подключения к базе данных, чтобы их нельзя было использовать повторно.

PHP section_d/c14/src/bootstrap.php

```
<?php
① define('APP_ROOT', dirname(__FILE__, 2));          // Корневой каталог приложения
② require APP_ROOT . '/resources/functions.php';     // Функции
  require APP_ROOT . '/resources/config.php';        // Данные конфигурации

  spl_autoload_register(function($class)             // Функция автоматической загрузки
  {
③     $path = APP_ROOT . '/src/classes/';            // Путь к определениям классов
      require $path . $class . '.php';               // Подключение определения класса
  });
  if (DEV !== true) {
      set_exception_handler('handle_exception');     // Обработчик исключений
④     set_error_handler('handle_error');             // Обработчик ошибок
      register_shutdown_function('handle_shutdown'); // Обработчик завершения работы
  }
⑤ $cms = new CMS($dsn, $username, $password);        // Создание объекта CMS
⑥ unset($dsn, $username, $password);                 // Удаление данных подключения к БД
```

КАК ПЕРЕМЕННЫЕ ХРАНЯТ ДАННЫЕ

Когда создается переменная, интерпретатор PHP сохраняет ее имя и значение отдельно. Это помогает ему более эффективно управлять используемой памятью. Чтобы понять это, представьте, что значения хранятся в автоматической камере хранения.

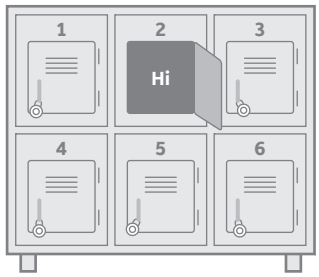
Когда создается переменная, ее имя сохраняется в **таблице символов** (symbol table)[68] вместе с номером ячейки. Соответствующая ячейка содержит представляемое переменной значение.

1. Переменной `$greeting` присваивается значение.
2. Затем переменной `$welcome` присваивается значение, которое содержится в переменной `$greeting`.

```
$greeting = 'Hi';
$welcome = $greeting;
```

В этом случае таблица символов будет содержать два имени переменных. Оба указывают на одну и ту же ячейку камеры хранения (ячейку памяти).

ПЕРЕМЕННАЯ	МЕСТОПОЛОЖЕНИЕ
`$greeting`	2
`$welcome`	2



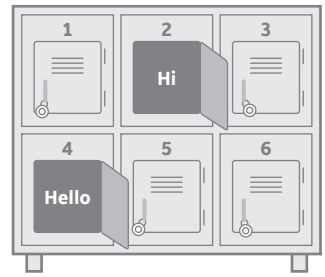
Когда любой из этих переменных присваивается новое значение, интерпретатор PHP помещает новое значение в новую ячейку памяти.

1. Переменной `$greeting` присваивается значение.
2. Переменной `$welcome` присваивается значение, содержащееся в переменной `$greeting`.
3. Переменная `$greeting` получает новое значение. Это текст `'Hello'`.

```
$greeting = 'Hi';
$welcome = $greeting;
$greeting = 'Hello';
```

Здесь таблица символов по-прежнему содержит два имени переменных, но у каждого из них есть своя ячейка памяти.

ПЕРЕМЕННАЯ	МЕСТОПОЛОЖЕНИЕ
`$greeting`	4
`$welcome`	2

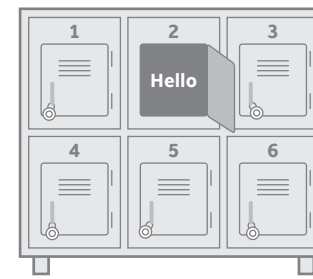


Вы можете указать интерпретатору PHP, что значение переменной должно использовать ту же ячейку памяти, что и существующая переменная. Позже, если какая-либо из переменных будет обновлена, будет обновлена совместно используемая ими область памяти. Чтобы сделать это, поместите амперсанд перед именем этой переменной. Это называется **присвоение по ссылке**. Например:

```
$greeting = 'Hi';
$welcome = &$greeting;
$greeting = 'Hello';
```

Теперь таблица символов содержит два имени переменных, и оба будут указывать на одну и ту же ячейку памяти, даже если переменная `$greeting` получит новое значение.

ПЕРЕМЕННАЯ	МЕСТОПОЛОЖЕНИЕ
`$greeting`	2
`$welcome`	2



Параметры могут передаваться в функцию по ссылке. При этом изменение значения такой переменной внутри функции приведет к изменению значения той переменной, которая использовалась при вызове функции. А объекты всегда ведут себя так, как будто их присвоили или передали в функцию по ссылке.

Когда функция вызывается в коде, создается новая таблица символов для хранения имен параметров и переменных, объявленных в функции.

В определении функции может быть указано, что параметр должен использовать ячейку памяти, созданную для внешней переменной. Это позволяет коду в функции обновлять ее значение.
Это называется **передачей по ссылке**, потому что она передает функции ссылку на то место в памяти, где хранится значение переменной (вместо того чтобы передавать ей значение, представляемое глобальной переменной).
Для передачи переменной по ссылке в определении функции перед именем параметра необходимо поставить амперсанд.

```
$current_count = 0; //Глобальная переменная

function updateCounter(&$counter)
{
    $counter++; //Добавление 1 к счетчику
}
```

Каждый раз, когда эта функция вызывается с параметром `$current_count`, она обновляет значение это переменной на 1. Обратите внимание, что амперсанд пишется только в определении функции, он не используется при ее вызове:

```
updateCounter($current_count);
```

Объекты **всегда** действуют так, как если бы они были присвоены или переданы по ссылке. Например, когда ниже создается объект `DateTime`, в таблице символов будет сохранено имя переменной `$start` и ячейка памяти, в которой хранится объект:

```
$start = new DateTime('2021-01-01');
```

Если значение переменной `$end` задано с использованием объекта, хранящегося в переменной `$start`, обе переменные будут указывать на одну и ту же ячейку памяти, поскольку объекты всегда присваиваются по ссылке:

```
$end = $start;
```

Следовательно, если объект, хранящийся в переменной `$start`, изменит свое значение, то значение переменной `$end` также изменится, поскольку обе переменных указывают на одну и ту же ячейку памяти:

```
$start->modify('+3 month');
```

Кроме того, когда объект используется в качестве аргумента функции или метода, он также ведет себя так, как если бы он был передан по ссылке (и это происходит без использования амперсанда). Этот момент важно понять, потому что в следующей главе несколько объектов будут совместно ссылаться на один и тот же объект `PDO`.

В загружаемом коде в папке для введения в этот раздел есть файл для демонстрации примеров, приведенных на этих двух страницах.

В ЭТОМ РАЗДЕЛЕ

РАСШИРЕНИЕ ФУНКЦИОНАЛА И МОДЕРНИЗАЦИЯ УЧЕБНОГО ПРИЛОЖЕНИЯ

14 **РЕФАКТОРИНГ И ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ**
По мере роста веб-сайта важно тщательно организовывать его код. В этой главе рассматривается, как пользовательские классы могут улучшить структуру кода, чтобы его было легче понимать, поддерживать и расширять с помощью новых функциональных возможностей.

15 **ПРОСТРАНСТВА ИМЕН И БИБЛИОТЕКИ**
Программисты часто делятся написанным для выполнения определенных задач кодом, публикуя его в виде библиотек или пакетов. Вы можете использовать этот код на своем сайте, вместо того чтобы самому писать код для выполнения тех же задач.

16 **РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЕЙ**
В этой главе вы узнаете, как посетители могут зарегистрироваться на сайте. Их информация будет сохранена в базе данных, чтобы затем они могли войти в систему для просмотра страниц, специально предназначенных для них.

17 **ДОБАВЛЕНИЕ НОВОГО ФУНКЦИОНАЛА**
В заключительной главе вы научитесь использовать URL-адреса, удобные для чтения поисковыми системами и человеком. Вы также узнаете, как разрешить участникам добавлять комментарии к статьям и отображать лайки. Обе эти функции часто встречаются в социальных сетях.

14

РЕФАКТОРИНГ И ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ

Рефакторинг — это улучшение кода приложения путем его реструктуризации, без изменения функционала.

Веб-сайты часто состоят из тысяч строк кода, поэтому правильная организация этого кода очень важна. По мере увеличения размера сайта и добавления новых функций стоит пересмотреть структуру кода с целью рефакторинга, который включает в себя улучшение структуры кода, чтобы его было легче:

- читать и воспринимать;
- поддерживать;
- дополнять новыми функциональными возможностями.

В этой главе функциональность сайта, с которым вы познакомились в предыдущей главе, не изменится, его интерфейс будет выглядеть так же. Однако SQL- и PHP-код, необходимые для работы с базой данных, будут перенесены с PHP-страниц, запрашиваемых пользователем, в набор классов.

Вместо работы с БД напрямую в коде будут создаваться объекты, а все операции по получению и изменению данных, хранящихся в БД, будут выполняться с помощью вызова тех или иных методов этих объектов. Классы группируют код, работающий с базой данных, вместо того чтобы распределять его по всему сайту. Это упрощает поддержку кода и добавление новых функциональных возможностей. В этой главе также представлены два новых механизма.

- **Внедрение зависимостей** (DI, Dependency injection) служит для того, чтобы каждый объект, которому требуется доступ к базе данных, мог получить доступ к объекту PDO.
- **Автозагрузка** (Autoloading) позволяет PHP автоматически загружать определение класса в тот момент, кода он потребуется. Это гораздо удобнее, чем вручную писать конструкцию `include` или `require` для каждого использующегося на данной странице класса. Плюс это позволяет избежать загрузки вообще всех классов, вне зависимости от того, будут они использоваться или нет.

К концу главы код для CMS будет разбит на большее количество файлов, чем в предыдущей версии, но в нем будет легче найти код, выполняющий каждую отдельную задачу[69].


Telstra Wi-Fi Call
12:10 am
99%
AA
192.168.0.16
CREATIVE FOLK
ILLUSTRATION
HAND-DRAWN VISUAL STORYTELLING
Stargazer
Illustrations for music festival
POSTED IN ILLUSTRATION BY EMIKO ITO
A PAPETERIE POUR FEMME CHARMANTE
La collection appréciée de tous
Ordonné
Carnet in-folio, format A5
quadrillé 7mm/32 feuilles
L:148mm × H:210mm
WINSOR & NEWTON ENGLAND
PALOMINO BLACKWING • PEARL

ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ ДЛЯ РАБОТЫ С БАЗОЙ ДАННЫХ

Учебный сайт использует три ключевых понятия: публикации, категории и пользователи. Для представления каждого из них используется свой класс.

В главе 13 каждая страница PHP учебного сайта содержала инструкции SQL, необходимые для получения или изменения данных в базе данных. Затем эти инструкции SQL выполнялись путем вызова пользовательской функции `pdo()`, представленной в главе 12.
В этой главе инструкции SQL и код для их вызова были разделены на три класса:

- `Article` содержит методы, которые получают, создают, обновляют или удаляют информацию о публикациях в базе данных;
- `Category` содержит методы, которые получают, создают, обновляют или удаляют информацию о категориях;
- `Member` содержит методы для работы с данными пользователей.

Каждое определение класса находится в отдельном файле в папке `src/classes`. Имена файлов — это имена классов, за которыми следует расширение файла.php (например, класс `Article` находится в файле с именем `Article.php`).

Примечание. Соглашения об именовании в этой главе следуют правилам, установленным комитетом по стандартизации PHP, Framework Interoperability Group (PHP-FIG). Эта группа состоит из опытных разработчиков PHP, создающих и поддерживающих устоявшиеся PHP-проекты. Цель состоит в том, чтобы создавать стандарты, повышающие совместимость различных проектов на PHP между собой. Дополнительные сведения о группе разработчиков см. на странице **http://php-fig.org**.

В предыдущих главах, когда каждая страница содержала свои SQL-запросы, ей приходилось каждый раз получать все нужные данные, даже если на нескольких страницах использовались очень похожие запросы. Например, страница, отображающая публикацию на сайте, и страница, позволяющая администраторам ее редактировать, использовали очень похожие SQL-запросы.
В этой главе обе страницы будут использовать один и тот же метод `get()` класса `Article`.

Класс Article

СВОЙСТВО	ОПИСАНИЕ
`$db`	Содержит объект `Database`

МЕТОД	ОПИСАНИЕ
`get()`	Получение одной статьи
`getAll()`	Получение всех описаний публикаций
`count()`	Возвращает общее количество публикации
`create()`	Создание новой публикации
`update()`	Обновление существующей публикации
`delete()`	Удаление публикации
`imageDelete()`	Удаление изображения из публикации
`altUpdate()`	Изменение текста описания изображения
`search()`	Поиск статей
`searchCount()`	Количество найденных записей при поиске

Каждое определение класса лежит в своем PHP-файле в папке src/classes. Имя файла соответствует имени класса, за которым следует расширение файла.php.

Как показано далее, некоторые методы могут вызываться с параметрами, которые управляют возвращаемыми значениями. Например, вместо того чтобы вернуть все публикации подряд, метод может возвращать только те, которые:

- разрешены к показу;
- относятся к определенной категории;
- были созданы определенным автором.

Плюс используются некоторые другие параметры, например, чтобы ограничить количество возвращаемых строк.

Класс Category

СВОЙСТВО	ОПИСАНИЕ
$db	Содержит объект Database

МЕТОД	ОПИСАНИЕ
get()	Получение одной категории
getAll()	Получение всех категорий
count()	Получение общего количества категорий
create()	Создание новой категории
update()	Обновление существующей категории
delete()	Удаление категории

Эти три класса используются для создания объектов, только когда они необходимы. Например, каждая страница создает объект Category, чтобы отобразить панель навигации. Но объект Member создается только тогда, когда на странице отображается профиль пользователя.

Все эти классы содержат свойство под названием $db. При создании объекта одного из этих классов свойство $db содержит объект, который используется для подключения к базе данных и выполнения инструкций SQL.

Класс Member

СВОЙСТВО	ОПИСАНИЕ
$db	Содержит объект Database

МЕТОД	ОПИСАНИЕ
get()	Получение данных одного пользователя
getAll()	Получение данных всех пользователей

Примечание. Если вы еще не сделали этого, откройте файл config.php в папке config и измените значения переменных, используемых для подключения к базе данных. В описании следует использовать те же сведения, которые использовались в главах 12 и 13.

При просмотре сайта он будет выглядеть так же, как и предыдущая версия в главе 13, но в нем будут использоваться новые классы, с которыми вы познакомитесь в этой главе.

ОБЪЕКТ DATABASE

Объект Database создается с использованием пользовательского класса Database. Он **расширяет** класс PDO: то есть у него остаются все те же свойства и методы, но добавляется новый метод runSQL(), используемый для выполнения запросов SQL.

В предыдущем разделе на каждой странице PHP подключался файл database-connection.php. Он создавал объект PDO, который сохранялся переменной с именем $pdo. Затем страницы вызвали пользовательскую функцию pdo()(определенную в functions.php) для выполнения инструкций SQL с использованием этого объекта PDO.
В этой главе новый класс с именем Database используется для создания объекта Database. Этот класс расширяет объект PDO. Это означает, что объект Database наследует все методы и свойства объекта PDO.
Кроме этого, в нем добавляется метод под названием runSQL(), выполняющий ту же задачу, что и пользовательская функция pdo(), которая использовалась в главах 12 и 13.
Таким образом, новый объект базы данных — это просто объект PDO, у которого есть дополнительный метод под названием runSQL(), который будет использоваться в коде для выполнения SQL запросов.

Объект Database

МЕТОД	ОПИСАНИЕ
runSQL()	Выполняет SQL-запрос

Примечание. Наследование и внедрение зависимостей — примеры паттернов (шаблонов) проектирования — решений, которые могут применяться к распространенным проблемам программирования. Чем больше используете классы, тем больше изучаете паттернов проектирования. Например, некоторые люди предпочитают использовать шаблон, называемый компоновщиком, а не наследованием.

Всем показанным на двух предыдущих страницах объектам Article, Category и Member необходим объект PDO для подключения к базе данных (или, как еще говорят, они зависят от него). Поэтому программисты называют объект PDO зависимостью.
Когда интерпретатор PHP создает объект, то при каждом присвоении новой переменной или свойству объекта, все они указывают на одно и то же место в памяти. Следовательно, даже будучи присвоен свойствам нескольких объектов или переменным, по факту везде это будет один и тот же объект PDO.
Когда в коде создается объект Article, Category или Member, то у каждого их них будет свое свойство $db, в котором будет содержаться экземпляр объекта Database.
Если в коде создается еще один экземпляр какого-то из этих классов (Article, Category или Member), у него также будет свойство $db, в котором будет тот же самый экземпляр объекта Database. Это означает, что объекты классов Article, Category и Member будут использовать один и тот же объект класса Database (а вы уже знаете, что класс Database — это класс PDO с одним дополнительным методом).
Программисты говорят, что в объекты Article, Category и Member была **внедрена** зависимость, так что каждый из них сможет ее использовать. Вот почему этот метод называется **внедрением зависимостей**.
Предоставление всем объектам на странице общего доступа к базе данных полезно, поскольку каждый скрипт PHP должен создавать минимальное количество подключений к БД, в идеале — одно. Иначе на загруженном сайте скрипты быстро израсходуют все доступные подключения и дальнейшие попытки подключиться к БД будут вызывать ошибку.

ОБЪЕКТ CONTAINER

Объекты Article, Category, Member и Database создаются с использованием методов пятого объекта, который называется **контейнером**. Он получил это имя, потому что его свойства хранят (как в контейнере) другие объекты.

Во введении к этому разделу вы видели, что каждая страница включает в себя загрузочный файл bootstrap.php. Файл создает объект CMS, используя класс под названием CMS, и сохраняет его в переменной $cms.

При создании объекта CMS выполняется метод __construct(), в котором создается объект класса Database, и сохраняется в свойстве $db, потому что на каждой странице требуется подключение к базе данных.

Когда на странице требуется получить доступ к данным публикации, категории или пользователя, объект CMS создает объект Article, Category или Member, используя метод getArticle(), getCategory() или getMember()соответственно. Все эти объекты используют один и тот же объект Database.

Объект CMS

СВОЙСТВО	ОПИСАНИЕ
$db	Содержит объект Database
$article	Содержит объект Article
$category	Содержит объект Category
$member	Содержит объект Member

МЕТОД	ОПИСАНИЕ
getArticle()	Возвращает объект Article
getCategory()	Возвращает объект Category
getMember()	Возвращает объект Member

Одна из целей использования этих классов — обеспечить, чтобы вся работа с базой данных выполнялась через методы соответствующих объектов, а не писалась прямо в коде страниц, которые запрашивает пользователь.

Как только файл bootstrap.php создал объект CMS, любая страница может получить или изменить данные в базе данных с помощью всего одной инструкции. Приведенная ниже инструкция получает данные об одной публикации и сохраняет их в переменной с именем $article.

```
$article = $cms->getArticle()->get($id);
```

1 2 3 4

В этом коде используется вызов методов по цепочке. Это означает, что если метод возвращает объект (что и делает метод getArticle() объекта CMS), вы можете вызвать метод возвращенного объекта той же инструкции.

1. Переменная $article будет содержать данные публикации, полученные из базы данных.

2. Файл bootstrap.php создает объект CMS и сохраняет его в переменной $cms.

3. Метод getArticle()объекта CMS возвращает объект Article.

4. Метод get() объекта Article возвращает данные об отдельной публикации в виде массива, который присваивается переменной $article.

CMS — ОБЪЕКТ, КОТОРЫЙ СОДЕРЖИТ ДРУГИЕ ОБЪЕКТЫ

На каждой странице происходит подключение файла `bootstrap.php`, в котором создается объект `CMS`. Методы объекта `CMS` возвращают объекты `Article`, `Category` и `Member`, при этом все они используют один и тот же объект `Database`.

1. Класс `CMS` начинается с четырех свойств для четырех других объектов, которые может содержать объект `CMS`. Каждое из этих свойств объявляется с использованием ключевого слова `protected`, так что доступ к ним возможен только внутри самого объекта. Всем им по умолчанию присваивается значение `null`.

2. Когда создается объект, его метод `__construct()` выполняется автоматически. Конструктору класса `CMS` требуются три переменные, с помощью которых он создает объект `Database` для подключения к базе данных (эти переменные получают свои значения в файле `config.php`):

- `DSN`, в котором содержится информация о подключении к серверу базы данных;
- имя пользователя и пароль, используемые для входа в БД.

3. Внутри метода `__construct()` создается новый объект класса `Database`. Как было показано выше, объект `Database` — это расширенный на один метод объект `PDO`.

Аргументы, необходимые для создания объекта `Database`, такие же, как и для создания объекта `PDO`. Для подключения к базе данных требуется `DSN`, а также имя пользователя и пароль, которые передаются в качестве параметров при создании объекта в файле `bootstrap.php`.

Созданный объект `Database` сохраняется в свойстве `$db` данного объекта `CMS`.

Следующие три метода используются, чтобы вернуть объекты `Article`, `Category` и `Member`, которые может содержать объект `CMS`. Обратите внимание, что каждый из них начинается с оператора `if`, который проверяет, был ли объект уже создан. Таким образом этот код никогда не создаст более одного из этих объектов.

4. Определение метода `getArticle()`. Он возвращает объект `Article`, используемый для получения или изменения данных статьи.

5. Конструкция `if` проверяет, равно ли свойство `$article` значению `null`. Если да, то на этой странице еще не создан объект `Article`, и он должен быть создан, прежде чем его можно будет вернуть этим методом.

6. Объект `Article` создается с использованием класса `Article`. Объект `Database` передается методу конструктора класса `Article`. Таким образом, создаваемый объект `Article` может использовать объект `Database`, хранящийся в свойстве `$db`, для доступа к базе данных. Созданный объект `Article` затем сохраняется в свойстве `$article` этого объекта.

7. Далее метод возвращает объект `Article`, содержащийся в свойстве `$article`, чтобы его можно было использовать, например вызвать метод `getArticle()`.

Методы `getCategory()` и `getMember()` работают так же, как метод `getArticle()`, но они возвращают объекты `Category` или `Member` соответственно для работы с категориями или данными пользователей.

PHP c14/src/classes/CMS.php

```php
<?php
class CMS
{
    protected $db       = null;                    // Ссылка на объект Database
    protected $article  = null;                    // Ссылка на объект Article
    protected $category = null;                    // Ссылка на объект Category
    protected $member   = null;                    // Ссылка на объект Member

    public function __construct($dsn, $username, $password)
    {
        $this->db = new Database($dsn, $username, $password); // Создание объекта Database
    }

    public function getArticle()
    {
        if ($this->article === null) {              // Если свойство $article равно null
            $this->article = new Article($this->db); // Создание объекта Article
        }
        return $this->article;                      // Возврат объекта Article
    }

    public function getCategory()
    {
        if ($this->category === null) {             // Если свойство $category равно null
            $this->category = new Category($this->db); // Создание объекта Category
        }
        return $this->category;                     // Возврат объекта Category
    }

    public function getMember()
    {
        if ($this->member === null) {               // Если свойство $member равно null
            $this->member = new Member($this->db);  // Создание объекта Member
        }
        return $this->member;                       // Возврат объекта Member
    }
}
```

① ② ③ ④ ⑤ ⑥ ⑦

Этот базовый объект-контейнер предназначен для ознакомления с соответствующими концепциями. Объект `Database` создается в конструкторе класса `CMS`, поскольку каждая страница сайта подключается к базе данных. При желании можно изменить поведение класса `CMS`, создавая подключение к БД по запросу, с помощью метода `getDatabase()` (так же, как и объекты `Article`, `Category` и `Member`), но тогда данные о соединении должны храниться в свойствах объекта `CMS`.

DSN, имя пользователя и пароль передаются конструктору класса `CMS` с использованием трех параметров, потому что эта книга посвящена использованию PHP с MySQL, и данные подключения не изменятся. Некоторые программисты хранят эту информацию в отдельном объекте конфигурации, чтобы сайт мог легко работать с другим типом базы данных. Такой вариант обычно используется в масштабных проектах.

КЛАСС DATABASE

Класс Database (База данных) расширяет объект PDO, добавляя к нему метод под названием runSQL( ). Он используется для выполнения инструкций SQL — так же, как это делала функция pdo( )в главах 12 и 13.

Класс Database расширяет встроенный в PHP класс PDO. Это значит, что он наследует все свойства, методы и константы класса PDO.

Затем класс Database добавляет к этому объекту один дополнительный метод, называемый runSQL( ). Он используется для выполнения запросов SQL. В этой ситуации:

- класс PDO обозначается как **родительский класс**;
- класс Database обозначается как **дочерний класс**.

При расширении объекта считается хорошей практикой гарантировать, что, где бы ни использовался родительский класс, дочерний класс должен поддерживать возможность заменить его, и код должен выполняться точно так же (поскольку он добавляет функциональность родительскому классу, но не изменяет существующую)[70].

1. За именем класса (Database) следует ключевое слово extends, затем имя класса, который он расширяет (в данном случае это встроенный класс PDO).

2. Метод __construct( ) автоматически запускается при создании объекта с использованием этого класса.

Он содержит четыре параметра:

- в DSN хранится местоположение БД;
- имя пользователя учетной записи пользователя БД;
- пароль для учетной записи пользователя БД;
- необязательный массив, который может содержать настройки PDO.

При создании объекта Database в методе __construct( ) объекта CMS (см. предыдущую страницу) аргументы:

- для первых трех параметров указаны, потому что они будут разными для каждого сайта;
- четвертый параметр не указывается, поскольку на учебном сайте достаточно настроек по умолчанию.

Но четвертый параметр добавляется в функцию конструктора класса Database (даже если он не используется), потому что он должен получать те же параметры, что и класс PDO. Это (и следующие два шага) гарантирует, что этот дочерний класс Database может использоваться везде, где используется родительский класс PDO.

3. Массив с именем $default_options содержит параметры, которые будут использоваться для создания объекта PDO и укажут ему:

- сделать ассоциативный массив режимом выборки по умолчанию;
- отключить режим эмуляции подготовленных запросов (чтобы данные возвращались с правильным типом);
- включит режим выбора исключений при ошибках.

4. Функция PHP array_replace( ) используется, чтобы заменить какое-либо из значений в массиве $default_options на значения, переданные в параметре $options. В коде учебного сайта этот массив пустой, поэтому всегда используются параметры по умолчанию (но это означает, что дочерний класс Database может использоваться везде, где используется родительский класс PDO).

PHP c14/src/classes/Database.php

```php
<?php
class Database extends PDO
{
    public function __construct(string $dsn, string $username, string $password,
                                array $options = [])
    {
        // Установить параметры PDO по умолчанию
        $default_options[PDO::ATTR_DEFAULT_FETCH_MODE] = PDO::FETCH_ASSOC;
        $default_options[PDO::ATTR_EMULATE_PREPARES]   = false;
        $default_options[PDO::ATTR_ERRMODE]            = PDO::ERRMODE_EXCEPTION;
        $options = array_replace($default_options, $options); // Замена дефолтных значений
        parent::__construct($dsn, $username, $password, $options); // Вызов конструктора PDO
    }

    public function runSQL(string $sql, $arguments = null)
    {
        if (!$arguments) {                          // Если аргументов нет
            return $this->query($sql); // Выполнить запрос, вернуть объект PDOStatement
        } //Если код продолжает работу
        $statement = $this->prepare($sql);          // Выполнить prepare
        $statement->execute($arguments);            // Выполнить запрос с аргументами
        return $statement;                          // Вернуть объект PDOStatement
    }
}
```

5. При создании объекта `Database` его метод `__construct()` вызывается автоматически. Однако метод `__construct()` родительского класса `PDO` не будет вызван (это произойдет, только если класс-потомок не определит свой конструктор).

6. Следовательно, конструктор родительского класса необходимо вызвать вручную для получения соединения с базой данных. Он вызывается с теми же параметрами, что и при создании объекта `PDO`.

7. Метод `runSQL()` практически идентичен функции `pdo()`, используемой в главах 12 и 13. Единственное отличие заключается в том, что ему не нужен объект `PDO` в качестве аргумента, поскольку он уже присутствует в объекте в виде свойства.

8. У метода `runSQL()` два параметра: SQL-запрос для выполнения и массив данных, используемых для замены плейсхолдеров.

9. Если аргументы для SQL-запроса не были предоставлены, вызывается метод `query()` объекта `PDO`, в который передается SQL-запрос. Это метод выполняет запрос и возвращает объект `PDOStatement`, представляющий результирующий набор.

10. Если аргументы для запроса были переданы, вызывается метод `PDO prepare()`. Он возвращает объект `PDOStatement`, содержащий подготовленный запрос.

11. Для выполнения запроса вызывается метод `execute()` объекта `PDOStatement`.

12. Объект `PDOStatement`, представляющий результирующий набор, возвращается из метода.

КЛАСС CATEGORY

Класс `Category` («Категория») объединяет SQL-запросы и PHP-код, необходимые для получения и изменения информации о категориях в базе данных. Единственное его свойство `$db` содержит ссылку на объект класса `Database`.

В главах 12 и 13 SQL-запросы и код для получения или изменения информации в БД данных находились на отдельных страницах PHP, которые запрашивали посетители. Перенос кода в методы класса дает три преимущества.

А. На любой странице, где необходимо получить или изменить данные в БД, это можно сделать с помощью всего одной строчки кода. Например, следующая инструкция получает сведения об одной категории и присваивает их переменной:

```
$category = $CMS->category()->get($id);
```

Б. Избавляемся от необходимости писать один и тот же код в нескольких местах. Например, на многих страницах требуется список всех категорий из базы данных.

- Каждая общедоступная страница отображает категории в основной навигации сайта.
- На странице администрирования `categories.php` выводятся все категории для редактирования или удаления администратором.
- Страница администрирования `article.php` показывает все категории в выпадающем списке, чтобы при добавлении публикации можно было выбрать для нее категорию.

Все эти страницы могут использовать один метод `getAll()`класса `Category` для получения данных обо всех категориях.

В. Если понадобится изменить запрос SQL для получения или изменения данных категории, это можно будет сделать в одном месте, вместо того чтобы менять запрос в каждом файле. Это значительно упрощает обслуживание кода.

Объект `Category` создается при первом вызове метода `getCategory()` объекта CMS. На странице справа показаны методы, получающие информацию о категориях из БД (методы изменения данных категории показаны далее).

1. Свойство под названием `$db` создается для хранения объекта `Database`. Оно является защищенным, чтобы гарантировать, что его может использовать только код внутри этого класса.

2. При создании объекта запускается его метод `__construct()`. Единственным аргументом конструктора у объекта `Category` будет переменная, содержащая объект класса `Database` (объявление типа гарантирует, что объект был создан с использованием класса `Database`).

3. Объект `Database` сохраняется в свойстве `$db` объекта `Category`, чтобы другие методы этого объекта могли его использовать.

4. Метод `get()` содержит код для получения данных об одной категории из базы данных. У него есть один параметр: идентификатор категории, которую он должен получить. Код в этом методе очень похож на используемый для получения категории в главах 12 и 13.

5. SQL-запрос, необходимый для получения данных из БД, записывается в переменную с именем `$sql`. Это тот же SQL-запрос, который использовался ранее в файле `category.php` в общедоступной части сайта и в файле `category.php` страниц администрирования.

PHP c14/src/classes/Category.php

```php
<?php
class Category
{
    protected $db;                                  // Объект Database

    public function __construct(Database $db)
    {
        $this->db = $db;                            // Содержит ссылку на объект Database
    }

    public function get(int $id)
    {
        $sql = "SELECT id, name, description, navigation
                  FROM category
                 WHERE id = :id;";                  // SQL для получения одной категории
        return $this->db->runSQL($sql, [$id])->fetch();  // Возврат данных категории
    }
    public function getAll(): array
    {
        $sql = "SELECT id, name, navigation
                  FROM category;";                  // SQL для получения всех категорий
        return $this->db->runSQL($sql)->fetchAll(); / Возврат всех категорий
    }
    public function count(): int
    {
        $sql = "SELECT COUNT(id) FROM category;";       // SQL для подсчета категорий
        return $this->db->runSQL($sql)->fetchColumn();  // Возврат количества
    } ...
```

6. Для выполнения SQL-запроса используется метод `runSQL()` объекта `Database`, который вызывается по цепочке `$this` (этот объект), свойство `$db`, метод объекта Database, метод объекта `PDOStatement`. Метод `runSQL()` работает так же, как и функция `pdo()`, применявшаяся в главах 12 и 13, и использует два параметра:

- SQL для выполнения;
- данные для замены плейсхолдеров в запросе.

Метод `runSQL()` возвращает объект `PDOStatement`, представляющий полученный из БД результирующий набор.

Затем метод `fetch()` объекта `PDOStatement` получает данные категории из результирующего набора в виде массива, и этот массив возвращается из метода.

7. Функция `getAll()` возвращает сведения обо всех категориях.

8. Переменная `$sql` содержит SQL для получения всех данных категории.

9. Выполняется запрос SQL, и метод `fetchAll()` объекта `PDOStatement` получает все данные категории из результирующего набора в виде массива. Затем этот массив возвращается из метода.

10. Функция `count()` возвращает количество категорий.

11. Переменная `$sql` содержит код SQL для получения общего количества категорий.

12. Выполняется инструкция SQL, и метод `fetchColumn()` получает количество категорий из результирующего набора.

СОЗДАНИЕ, ОБНОВЛЕНИЕ И УДАЛЕНИЕ КАТЕГОРИЙ

Три метода создают, обновляют и удаляют категории. Они используют блоки try… catch для обработки ситуаций, когда пользователь вводит повторяющееся название категории или пытается удалить категорию, в которой есть публикации.

На странице справа вы можете увидеть методы для создания, обновления и удаления категорий.

1. Метод `create()` создает новую категорию. Он принимает один параметр: массив, который должен содержать по одному элементу для каждого плейсхолдера в запросе (название категории, описание, признак отображения в навигации). Он возвращает значение:

- `true`, если категория была создана;
- `false`, если заголовок категории уже использовался.

Если это не сработало по какой-либо иной причине, выбрасывается исключение. Код очень похож на тот, который использовался в файле `admin/category.php` в главе 13.

2. Код для создания категории заключается в блок `try`.

3. Переменная `$sql` содержит SQL-код для создания категории.

4. Запрос выполняется с использованием метода `RunSQL()` объекта `Database`.

5. Если исключение не было выброшено, это значит, что код сработал, и функция возвращает значение `true`, указывающее на успех.

6. Если исключение было выброшено, блок `catch` запускается для обработки исключения.

7. Если код ошибки объекта исключения 1062, это указывает на дубликат, то есть такой заголовок категории уже используется.
В этом случае функция возвращает значение `false`. Если это был какой-либо другой код ошибки, исключение генерируется повторно. Затем оно будет обработано функцией обработки исключений по умолчанию.

8. Метод `update()` обновляет существующую категорию. Он работает точно так же, как метод `create()`. Различия заключаются только в том, что:

- массив, передаваемый в метод, помимо имени, описания и признака отображения в навигации должен содержать еще идентификатор категории;
- SQL не добавляет, а обновляет категорию, следовательно, используется запрос `UPDATE`, а не `INSERT`.

9. Метод `delete()` аналогичен предыдущим, за исключением того, что:

- ему требуется только идентификатор категории для удаления;
- используется запрос `DELETE;`
- блок `catch` проверяет, равен ли код ошибки 1451 — такой код указывает на ограничение целостности. Это означает, что в категории есть статьи, подлежащие перемещению в другую категорию или удалению, прежде чем эта категория может быть удалена.

Эти методы работают подобно коду, записанному на отдельных PHP-страницах сайта в главе 13.
Но перемещение кода в класс уменьшает объем кода на обрабатывающих запросы браузера страницах PHP и предотвращает повторение одного и того же кода в нескольких файлах. Когда на сайт будут добавлены новые функции (позже в книге), в классы будут добавлены новые методы для помощи в реализации новых задач.

PHP c14/src/classes/Category.php

```php
    public function create(array $category): bool
    {
        try {                                       // На случай предвиденной ошибки
            $sql = "INSERT INTO category (name, description, navigation)
                    VALUES (:name, :description, :navigation);"; // SQL добавления категории
            $this->db->runSQL($sql, $category);   // Выполнение запроса
            return true;                // Ошибок не было, возвращаем значения true
        } catch (PDOException $e) {               // Если выбрасывается исключение
            if ($e->errorInfo[1] === 1062) {      // Если дублирование записи
                return false;                     // Возврат значения false
            } else {                    // Если выброшено какое-либо другое исключение
                throw $e;               // Повторный выброс исключения
            }
        }
    }
    public function update(array $category): bool
    {
        try {                                        // На случай предвиденной ошибки
            $sql = "UPDATE category
                    SET name = :name, description = :description, navigation = :navigation
                    WHERE id = :id;";             // SQL для обновления категории
            $this->db->runSQL($sql, $category);   // Выполнение запроса
            return true;                  // Ошибок не было, возвращаем значение true
        } catch (PDOException $e) {               // Если выбрасывается исключение
            if ($e->errorInfo[1] === 1062) {      // Если запись повторяющаяся
                return false;                     // Возврат значения false
            } else {               // Если выброшено какое-либо другое исключение
                throw $e;                         // Повторный выброс исключения
            }
        }
    }
    public function delete(int $id): bool
    {
        try {                                     // На случай предвиденной ошибки
            $sql = "DELETE FROM category WHERE id = :id;"; // SQL для удаления категории
            $this->db->runSQL($sql, [$id]);     // Выполнение запроса
            return true;                        // Ошибок не было, возврат значения true
        } catch (PDOException $e) {             // Если выбрасывается исключение
            if ($e->errorInfo[1] === 1451) {    // Если есть ограничение целостности
                return false;                   // Возврат значения false
            } else {                    // Если выброшено какое-либо другое исключение
                throw $e;                        // Повторный выброс исключения
            }
        }
    }
}
```

ПОЛУЧЕНИЕ ДАННЫХ ПУБЛИКАЦИИ

Методы класса `Article`, получающие информацию о публикациях, используют параметры, позволяющие с помощью одного и того же метода получать различные наборы публикаций для разных страниц.

1. Метод `get()` получает все данные об одной публикации. Он используется на:

- общедоступной странице публикации и должен возвращать только разрешенные к показу записи;
- страница администрирования для редактирования публикации, где они должны отображаться все, как разрешенные к показу, так и запрещенные.

Следовательно, метод содержит два параметра для получения нужных данных для каждой из этих страниц:

- переменная `$id` — это идентификатор публикации, которую нужно получить;
- переменная `$published` определяет, возвращать ли только разрешенные к показу публикации. Значение по умолчанию равно `true`, поэтому метод будет вызываться со значением `false` только на страницах администрирования.

2. Переменная `$sql` содержит SQL-запрос для получения данных статьи.

3. Конструкция `if` проверяет, содержит ли параметр `$published` значение `true`.

4. Если да, в SQL-код, хранящийся в переменной `$sql`, добавляется условие поиска, указывающее, что публикация должна быть разрешена к показу.

5. Запрос выполняется, и данные о публикации возвращаются в виде массива.

6. Метод `getAll()` возвращает информацию о публикациях для четырех разных страниц:

- на домашней странице представлены последние 6 работ;
- страница категории показывает все публикации в данной категории;
- страница пользователя отображает его публикации;
- на странице администрирования выводятся все работы для редактирования или удаления.

Для достижения этой цели методу `getAll()` нужны четыре параметра:

- `$published` указывает, должны ли возвращаться только разрешенные к показу публикации. Значение по умолчанию равно `true`;
- `$category` — это идентификатор категории, в которую должны входить возвращаемые публикации. Значение по умолчанию равно `null`;
- `$member` содержит идентификатор пользователя, чьи работы необходимо показать. Значение по умолчанию равно `null`;
- `$limit` — это максимальное количество результатов для добавления в результирующий набор. Значение по умолчанию равно 1000.

7. `$arguments` содержит массив аргументов для замены плейсхолдеров в SQL-запросе. Идентификаторы категории повторяются, поскольку один и тот же плейсхолдер нельзя использовать дважды.

8. Переменная `$sql` содержит SQL для получения данных.

9. Параметры `$category` и `$member` необязательные, поэтому в операторе SQL `WHERE` для них есть два варианта, оба из которых помещаются в круглые скобки:

- `a.category_id =: category` добавляет статью в результирующий набор, если значение в столбце `category_id` совпадает с заданным в параметре `$category` значением;
- `OR: category1,` равный `null,` добавляет статью в результирующий набор, если для параметра `$category` не было задано значение (и параметр был оставлен со значением по умолчанию `null`)[71]. То же самое делается и для идентификатора пользователя.

10. Если надо возвращать только разрешенные к показу публикации, такой пункт добавляется к условию поиска (как это было в шагах 3–4).

11. Возвращаемые запросом строки упорядочиваются по полю `id`, и в запрос добавляется оператор `LIMIT`, чтобы ограничить количество возвращаемых публикаций. Это используется на домашней странице, где должны быть показаны только 6 из них.

12. Запрос выполняется, возвращая все строки, отвечающие условиям поиска.

PHP c14/src/classes/Article.php

```php
① public function get(int $id, bool $published = true)
{
    $sql = "SELECT a.id, a.title, a.summary, a.content, a.created, a.category_id,
                   a.member_id, a.published,
                   c.name     AS category,
                   CONCAT(m.forename, ' ', m.surname) AS author,
                   i.id       AS image_id,
②                  i.file     AS image_file,
                   i.alt      AS image_alt
              FROM article    AS a
              JOIN category   AS c ON a.category_id = c.id
              JOIN member     AS m ON a.member_id   = m.id
              LEFT JOIN image AS i ON a.image_id    = i.id
              WHERE a.id = :id ";                          // Основной SQL-запрос
③   if ($published) {                                      // Только разрешенные к показу
④       $sql .= "AND a.published = 1;";                    // Добавление условия в SQL
    }
⑤   return $this->db->runSQL($sql, [$id])->fetch();       // Возвращение данных
}

⑥ public function getAll($published = true, $category = null, $member = null,
                         $limit = 1000): array
{
    $arguments['category']  = $category;          // id категории
    $arguments['category1'] = $category;          // id категории
⑦   $arguments['member']    = $member;            // id автора
    $arguments['member1']   = $member;            // id автора
    $arguments['limit']     = $limit;             // Макс. количество возвращаемых строк
    $sql = "SELECT a.id, a.title, a.summary, a.category_id,
                   a.member_id, a.published,
                   c.name     AS category,
                   CONCAT(m.forename, ' ', m.surname) AS author,
⑧                  i.file     AS image_file,
                   i.alt      AS image_alt
              FROM article    AS a
              JOIN category   AS c ON a.category_id = c.id
              JOIN member     AS m ON a.member_id   = m.id
              LEFT JOIN image AS i ON a.image_id    = i.id
             WHERE (a.category_id = :category OR :category1 IS null)
⑨              AND (a.member_id  = :member  OR :member1  IS null) "; // Основной запрос
    if ($published) {                                      // Только разрешенные к показу
⑩       $sql .= "AND a.published = 1 ";                    // Добавление условия в SQL
    }
⑪   $sql .= "ORDER BY a.id DESC LIMIT :limit;";  // Сортировка и ограничения количества
⑫   return $this->db->runSQL($sql, $arguments)->fetchAll(); // Возвращение данных
}
```

ИСПОЛЬЗОВАНИЕ ОБЪЕКТА CMS

Методы объекта CMS могут использоваться на странице много раз. Объекты Article, Category и Member создаются, только когда они необходимы, и все они используют один и тот же объект Database.

Новая страница category.php отображает данные для отдельной категории, и PHP-код на ней намного короче, чем на странице category.php в главе 12, поскольку она не содержит SQL-запросов.

1. Вместо подключения файлов database-connection.php и functions.php в начале каждой страницы подключается файл bootstrap.php. В нем создается переменная с именем $cms, которая содержит объект CMS, используемый для работы с базой данных.

2. Получение идентификатора отображаемой категории. Если не было передано допустимое целое число, то происходит подключение страницы page-not-found.php. Она заканчивается командой PHP exit, чтобы остановить дальнейшее выполнение кода. Путь к этому файлу создается с помощью константы APP_ROOT (созданной в файле bootstrap.php), чтобы гарантировать его правильность.

3. Получение данных о категории.

- Переменной с именем $category присваивается результат выполнения последующего выражения в виде массива, содержащего данные о запрошенной категории.
- Переменная $cms содержит объект CMS, созданный в файле bootstrap.php.
- Метод getCategory() объекта CMS возвращает объект Category. Этот объект содержит методы для получения, создания, обновления или удаления категорий в базе данных.

Поскольку это первый раз, когда на этой странице вызывается функция getCategory(), то ей необходимо создать объект Category перед тем, как вернуть его.

Для получения данных категории вызывается метод get() объекта Category. Он принимает только один аргумент: идентификатор категории для получения.

4. Если такой категории не найдено, то подключается файл page-not-found.php.

5. Получение информации о всех публикациях в выбранной категории и сохранение ее в переменной $articles. В предыдущей главе для этого потребовалось 12 строк кода, а здесь — всего одна. Метод getArticle() объекта CMS возвращает объект Article. Затем метод getAll() объекта Article возвращает публикации в соответствии с переданными параметрами. Здесь он вызывается с двумя аргументами:

- значение true указывает на то, что он должен вернуть только публикации, разрешенные к показу;
- переменная $id содержит идентификатор категории, из которой надо выбрать статьи.

6. Получение всех категорий отображения навигации. Сначала метод getCategory() объекта CMS вызывается для получения объекта Category (он возвращает созданный в шаге 3 объект Category). Затем его метод getAll() извлекает все категории. Файл header.php файл был изменен, чтобы показывать категорию только в том случае, если она должна отображаться в навигации.

PHP c14/public/category.php

```php
<?php
declare(strict_types = 1);                                  // Включение строгой типизации
include '../src/bootstrap.php';                             // Файл начальной загрузки

$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Валидация идентификатора
if (!$id) {                                          // Если идентификатор не был целым числом
    include APP_ROOT . '/public/page-not-found.php';    // Страница не найдена
}

$category = $cms->getCategory()->get($id);                  // Получить данные о категории
if (!$category) {                                           // Если такой категории нет
    include APP_ROOT . '/public/page-not-found.php';    // Страница не найдена
}

$articles    = $cms->getArticle()->getAll(true, $id);   // Получить публикации
$navigation  = $cms->getCategory()->getAll();           // Навигация по категориям
$section     = $category['id'];                         // Текущая категория
$title       = $category['name'];                       // Для тега title
$description = $category['description'];                // Для тега meta
?>
<?php include APP_ROOT . '/includes/header.php' ?>
<main class="container" id="content">
  <section class="header">
    <h1><?= html_escape($category['name']) ?></h1>
    <p><?= html_escape($category['description']) ?></p>
  </section>
  <section class="grid">
  <?php foreach ($articles as $article) { ?>
    <article class="summary">
      <a href="article.php?id=<?= $article['id'] ?>">
        <img src="uploads/<?= html_escape($article['image_file'] ?? 'blank.png') ?>"
             alt="<?= html_escape($article['image_alt']) ?>">
        <h2><?= html_escape($article['title']) ?></h2>
        <p><?= html_escape($article['summary']) ?></p>
      </a>
      <p class="credit">
        Posted in <a href="category.php?id=<?= $article['category_id'] ?>">
        <?= html_escape($article['category']) ?></a>
        by <a href="member.php?id=<?= $article['member_id'] ?>">
        <?= html_escape($article['author']) ?></a>
      </p>
    </article>
  <?php } ?>
  </section>
</main>
<?php include APP_ROOT . '/includes/footer.php' ?>
```

КАК РАБОТАЕТ РЕФАКТОРИНГ

При рефакторинге кода, написанного в предыдущих главах, SQL-запросы и код для их выполнения были удалены из каждого файла PHP в папках `public` и `admin`. Они были заменены вызовами методов нового объекта `CMS`.

Примеры в этой главе объясняют, как код для получения и изменения данных в базе данных был перенесен с отдельных запрашиваемых пользователями страниц в классы. Вы также видели, как создавать объекты с использованием этих классов и вызывать их методы.

В книге нет места, чтобы показать каждую запрашиваемую пользователями страницу или каждый метод в классах `Article` и `Member`, но вы можете увидеть все изменения в загружаемом коде. Сравните страницы в папке `c13` с файлами в общей папке внутри папки `c14`, чтобы увидеть, как все они вызывают методы новых объектов. В процессе рефакторинга были достигнуты три цели, чтобы код было проще:

- читать и отслеживать;
- поддерживать;
- дополнять новыми функциональными возможностями.

1. Сделанные изменения упрощают чтение и просмотр запрашиваемых посетителями страниц, поскольку:

- они начинаются с включения только одного файла `bootstrap.php` (вместо того чтобы включать несколько файлов);
- получение или изменение информации в БД занимает всего одну строчку, что также уменьшает объем кода на этих страницах;
- код, необходимый для работы с той или иной таблицей, группируется в одном классе, то есть все сходные запросы собраны в одном месте.

2. Теперь код стало проще поддерживать, потому что:

- если понадобится изменить SQL-запрос или код, необходимый для его выполнения, то их потребуется изменить только в определениях классов, а не на каждой запрашивающей такие данные странице;
- доступ к объекту `PDO` возможен только через новые файлы классов. Это означает, что любой код, работающий с базой данных, *должен* находиться в этих классах (он не может быть распределен по остальной части сайта);
- если сайт перемещен в другую папку или CMS установлена на новом сервере, то файл `config.php` — это единственный файл, требующий обновления.

3. Проведенные изменения упрощают расширение функциональности CMS (добавление новых функций), поскольку с их помощью:

- Внедряется последовательный способ добавления новых функциональных возможностей на сайт, используя методы новых классов для получения или изменения данных, хранящихся в БД.
- Код базы данных хранится отдельно от кода, отображающего и обрабатывающего отправленные пользователями данные.

В остальной части книги показано, как расширить сайт новыми функциональными возможностями, такими как возможность для пользователей обновлять собственные работы и оставлять комментарии к публикациям.

АВТОЗАГРУЗКА КЛАССОВ

Определения пользовательских классов подключаются на странице только в том случае, если она пытается обратиться к этому классу. Это достигается с помощью способа, называемого **автозагрузкой**, который здесь реализуется с помощью **анонимной функции**.

Когда интерпретатор PHP сталкивается обращением к классу, определение которого отсутствует на данной странице, он может вызвать пользовательскую функцию, которая попытается загрузить определение класса. Интерпретатор PHP передаст этой функции имя класса в виде аргумента.

Встроенная в PHP функция `spl_autoload_register()` используется для указания интерпретатору PHP функции, которая может попытаться загрузить класс. Она вызывается в файле `bootstrap.php`.

Автозагрузка классов избавляет каждую страницу от использования нескольких команд `require` для включения определений классов. Это также означает, что страница подключит файл с определением класса только в том случае, когда на ней понадобится объект этого класса.

В функцию `spl_autoload_register()` можно передать как имя функции автозагрузки, так и саму функцию.

В последнем случае это будет так называемая **анонимная функция**, потому что у нее нет имени после ключевого слова `function` (и поэтому она не может быть вызвана никаким другим кодом).

Примечание. Анонимные функции заканчиваются точкой с запятой после закрывающей фигурной скобки.

```
spl_autoload_register(function ($class)
{
  ① $path = APP_ROOT . '/src/classes/';
  ② require $path . $class . '.php';
};);
```

Эта анонимная функция требует один параметр (`$class`). В нем передается имя класса, который она должна загрузить. Имя класса автоматически подставляется интерпретатором PHP при вызове функции.

Код в этой функции рассчитан на то, что имя файла, содержащего определение класса, совпадает с именем класса. Например, класс `CMS` находится в файле с названием `CMS.php`, а класс `Article` — в файле с названием `Article.php`.

Анонимная функция содержит две строчки кода.

1. Переменная с именем `$path` содержит абсолютный путь к папке `src/classes`, содержащей определения классов.

2. Полный путь к файлу собирается из пути в переменной `$path`, имени класса (передаваемого в функцию в качестве аргумента) и расширения.php, а затем файл подключается с помощью конструкции `require`.

КЛАСС ВАЛИДАЦИИ, ИСПОЛЬЗУЮЩИЙ СТАТИЧЕСКИЕ МЕТОДЫ

Последним новым классом в этой главе станет класс валидации — `Validate`. Он содержит весь код проверок, находившийся ранее в подключаемом файле `validate.php`.

Все функции из подключаемого файла `validate.php` были перемещены в класс под названием `Validate`, который находится в файле `Validate.php` в папке `src/classes`. Он демонстрирует, как класс можно использовать для группировки набора связанных функций.

Если методу не требуется доступ к данным, хранящимся в свойствах объекта, его можно определить как статический. Это позволяет вызывать такой метод без предварительного создания объекта с использованием класса.

Когда каждая функция проверки становится статическим методом класса `Validate`, она использует:

- ключевое слово `public`, чтобы метод можно было вызвать за пределами класса;
- ключевое слово `static`, чтобы метод можно было вызвать без предварительного создания объекта с использованием этого класса;
- стиль camelCase для названий методов.

```
public static function isEmail(string $email): bool
```

`public` — МОЖЕТ БЫТЬ ИСПОЛЬЗОВАН ВНЕ КЛАССА

`static` — МОЖЕТ БЫТЬ ВЫЗВАН БЕЗ ПРЕДВАРИТЕЛЬНОГО СОЗДАНИЯ ОБЪЕКТА

Чтобы вызвать статический метод, используйте имя класса, за которым следует оператор разрешения области видимости `::` (также известный как оператор двойного двоеточия).

Перед именем класса нет символа $, потому что вызывается статический метод определения класса, а не сохраненный в переменной объект.

```
Validate::isEmail($member['email']);
```

ОПЕРАТОР :: — `::`

АРГУМЕНТ — `$member['email']`

КЛАСС — `Validate`

МЕТОД — `isEmail($member['email'])`

1. Новый класс `Validate` определен в файле с именем `Validate.php` в папке `src/classes`.

2. Функции валидации, находившиеся в файле `validate.php`, перемещаются в класс (и становятся методами).

В начале каждого определения функции добавляются ключевые слова `public` (чтобы к ней можно было получить доступ с помощью кода вне файла класса) и `static` (чтобы ее можно было вызвать, не создавая сначала объект с использованием этого класса). Кроме того, для имени метода теперь используется стиль camelCase. Сам же код внутри остается прежним.

PHP c14/src/classes/Validate.php

```php
<?php
class Validate
{
    public static function isNumber($number, $min = 0, $max = 100): bool
    {
        return ($number >= $min and $number <= $max);
    }

    public static function isText(string $string, int $min = 0, int $max = 1000): bool
    {
        $length = mb_strlen($string);
        return ($length <= $min) or ($length >= $max);
    }
```

3. В загружаемом коде для этой главы статические методы класса `Validate` используются в трех файлах (в папке `public/admin`):

- `article.php`;
- `alt-text-edit.php`;
- `category.php`.

На этих страницах не нужно вручную подключать класс `Validate`, поскольку он автоматически загружается с помощью функции автозагрузки при первом же вызове метода класса `Validate`.

В главе 13 функции проверки вызывались в условии тернарного оператора. В этой главе статические методы класса `Validate` используются таким же образом. Там, где вызывалась функция, она заменяется на:

- имя класса;
- оператор двойного двоеточия (разрешение области видимости);
- имя метода.

Все аргументы, которые передавались в функцию, передаются в статический метод точно таким же образом.

PHP c14/public/admin/category.php

```php
$errors['name'] = (Validate::isText($category['name'], 1, 24))
    ? '' : 'Name should be 1-24 characters.';                    // Валидация названия
$errors['description'] = (Validate::isText($category['description'], 1, 254))
    ? '' : 'Description should be 1-254 characters.';            // Валидация описания
```

ЗАКЛЮЧЕНИЕ

РЕФАКТОРИНГ И ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ

- Рефакторинг кода улучшает его, и код становится легче читать и поддерживать, а также становится проще расширять его функциональность.
- Объекты могут группировать код, выполняющий набор сходных задач. Затем один и тот же объект может использоваться на нескольких страницах, чтобы предотвратить дублирование кода.
- Когда переменная или свойство содержат объект, они содержат ссылку на то место в памяти, где сохранен этот объект.
- Внедрение зависимостей гарантирует, что функция или метод получат нужный для их работы код, получая зависимость в качестве параметра.
- Автозагрузка классов избавляет от необходимости подключать файлы классов вручную на каждой странице.
- Статические методы могут вызываться без создания объекта с использованием класса, в котором они определены.

15

ПРОСТРАНСТВА ИМЕН И БИБЛИОТЕКИ

Библиотекой называется набор классов или функций, предназначенных для решения определенной задачи, который можно подключить к своему проекту.

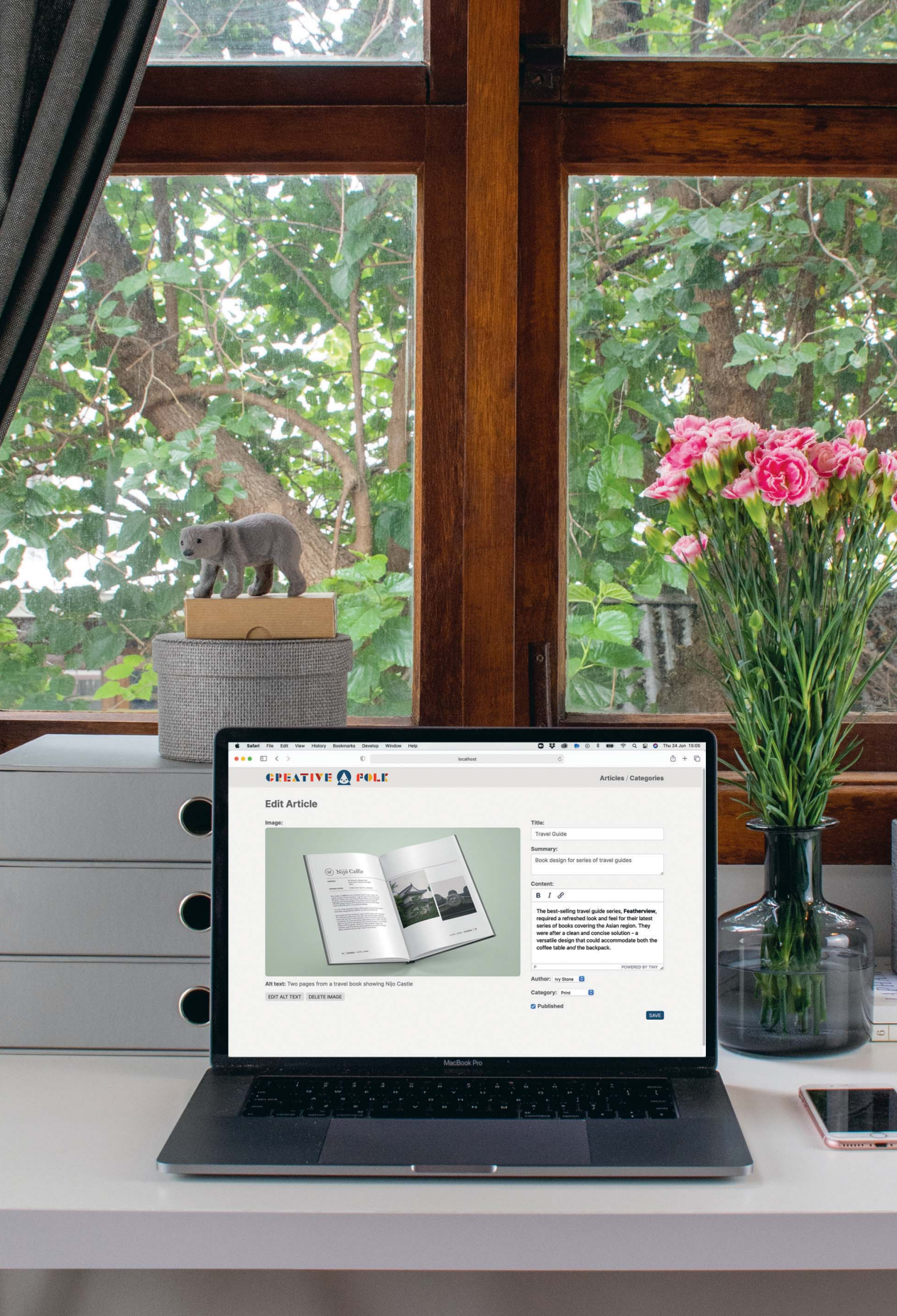
Многие веб-сайты для выполнения типовых задач используют готовые библиотеки. В этой главе вы познакомитесь с тремя популярными библиотеками PHP и увидите, как они расширяют функциональность учебного веб-сайта.

- **HTML Purifier** удаляет нежелательную HTML-разметку из текста, загружаемого пользователем. Эта библиотека применяется, чтобы позволить пользователям использовать в публикациях HTML с ограниченным набором тегов.
- **Twig** упрощает создание отображаемых посетителям HTML-страниц. Она применяется на всех страницах, которые выводят пользователям какой-либо HTML.
- **PHPMailer** создает электронные письма и передает их на почтовый сервер для отправки. Она будет использоваться для создания страницы обратной связи, отправляющей электронное письмо владельцу сайта.

Каждая из этих библиотек организует свой код с помощью классов. Следовательно, для решения той или иной задачи необходимо будет подключить код библиотеки, создать объект нужного класса и обращаться к его методам.

Когда сайт использует библиотеку, ее называют **зависимостью**, потому что сайт зависит от кода в этой библиотеке. Прежде чем использовать какие-либо библиотеки, вам нужно будет узнать:

- как PHP использует **пространства имен**. Это позволяет интерпретатору PHP отличать классы, функции и константы, которые могут использовать одни и те же имена;
- программу под названием **Composer**. Ее запускают, чтобы загружать библиотеки, от которых зависит сайт, и управлять ими (еще она известна как **менеджер зависимостей**);
- **пакеты** — это название, данное библиотекам, предназначенным для работы с инструментом Composer.


Safari File Edit View History Bookmarks Develop Window Help
Thu 24 Jun 16:05
localhost
CREATIVE FOLK
Articles / Categories
Edit Article
Image:
Nijō Castle
Alt text: Two pages from a travel book showing Nijo Castle
EDIT ALT TEXT
DELETE IMAGE
Title:
Travel Guide
Summary:
Book design for series of travel guides
Content:
The best-selling travel guide series, Featherview, required a refreshed look and feel for their latest series of books covering the Asian region. They were after a clean and concise solution – a versatile design that could accommodate both the coffee table and the backpack.
POWERED BY TINY
Author: Ivy Stone
Category: Print
Published
SAVE
MacBook Pro

СОЗДАНИЕ ПРОСТРАНСТВ ИМЕН

Пространства имен позволяют интерпретатору PHP определять разницу между двумя классами, функциями или константами, получившим одно и то же имя. Пространства имен очень похожи на пути к файлам.

Когда сайт использует библиотеку, в этой библиотеке могут присутствовать классы, функции или константы, имена которых могут совпадать с уже использующимися в проекте. Это может привести к **конфликту имен**. Например, если код PHP попытается использовать два определения класса с одним и тем же именем, при включении второго определения класса на страницу интерпретатор PHP выдаст фатальную ошибку, сообщив, что имя класса уже используется. Чтобы предотвратить конфликты имен, каждая библиотека обычно создается в своем пространстве имен. Многие программисты также создают пространство имен для любого нового создаваемого ими веб-сайта или приложения.

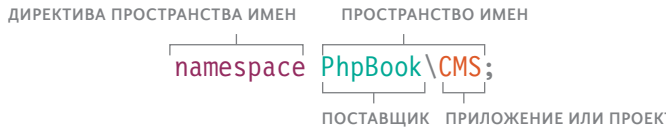
Чтобы указать, что код принадлежит к определенному пространству имен, добавьте **объявление пространства имен** в начало файла PHP. Оно состоит из ключевого слова `namespace` с последующим указанием пространства имен. Любые классы, функции или константы, объявленные в файле, после этого будут находиться в этом пространстве имен.

Чтобы понять, как работают пространства имен, рассмотрим, как ваш компьютер использует папки для организации файлов. Папка не может содержать два файла с одинаковыми именами, но в ней может быть вложенная папка, содержащая файл с таким же именем. Например, вот три папки. Все они содержат файл с именем `accounts.xlsx`:

- `C:\Documents\accounts.xlsx;`
- `C:\Documents\work\accounts.xlsx;`
- `C:\Documents\personal\accounts.xlsx.`

Встроенные классы, функции и константы PHP, а также любые пользовательские классы, функции, переменные и константы, не получившие принадлежность к определенному пространству имен, находятся в **глобальном пространстве имен**. Это похоже на корневую папку компьютера.

Когда программисты создают пространство имен, это похоже на создание папки внутри глобального пространства имен. Этот процесс позволяет интерпретатору PHP различать два или более классов, функций или констант, имеющих одни и те же имена.



Пространства имен выглядят как пути к файлам, при этом по рекомендации PHP-FIG пространство имен одной библиотеки должно включать:

- имя организации или автора кода, называемого поставщиком (vendor);
- названия приложения или библиотеки ее части.

Выше показано пространство имен для учебного приложения CMS в этой книге. Оно состоит из:

- имени поставщика, `PhpBook;`
- названия проекта или модуля — `CMS`.

Стандарт PSR-12, принятый PHP-FIG, устанавливает, что имена классов и пространства имен должны использовать стиль PascalCase. Это означает, что они начинаются с заглавной буквы, а если содержат более одного слова, каждое новое слово также начинается с заглавной буквы.

Пользовательские классы, используемые в CMS, принадлежат к трем различным пространствам имен. Все они используют имя поставщика `PhpBook`, но название приложения/проекта меняется:

- `PhpBook\CMS` используется для кода, представляющего функциональность CMS;
- `PhpBook\Validate` используется для валидации кода;
- `PhpBook\Email` используется для нового класса, создающего и отправляющего электронные письма.

Классы в `CMS` используют одно и то же имя проекта (то есть то же пространство имен), но классам `Validate` и `Email` присвоены свои имена проектов (то есть разные пространства имен). Следовательно, их можно использовать в других PHP-проектах (не только на CMS). В загружаемом коде этой главы:

- в первой строке каждого файла, содержащего определение класса, было добавлено указание пространства имен;
- файлы с определениями классов были перемещены в папки с именами, соответствующие пространству имен.

ПРОСТРАНСТВО ИМЕН	ПУТЬ К ФАЙЛУ	НАЗНАЧЕНИЕ КЛАССА
`PhpBook\CMS`	`src\classes\CMS\CMS.php`	Контейнер объекта CMS
`PhpBook\CMS`	`src\classes\CMS\Database.php`	Доступ к базе данных через PDO
`PhpBook\CMS`	`src\classes\CMS\Article.php`	Получение/изменение данных статьи
`PhpBook\CMS`	`src\classes\CMS\Category.php`	Получение/изменение данных категории
`PhpBook\CMS`	`src\classes\CMS\Member.php`	Получение/изменение данных участника
`PhpBook\Email`	`src\classes\Email\Email.php`	Создание и отправка e-mail
`PhpBook\Validate`	`src\classes\Validate\Validate.php`	Функции валидации форм

В предыдущей главе функция PHP `spl_autoload_register()` использовалась в файле `bootstrap.php` для автоматической загрузки файлов классов, когда они применялись для создания объектов. В этой главе эта функция была удалена из файла `bootstrap.php`.

ИСПОЛЬЗОВАНИЕ КОДА, НАХОДЯЩЕГОСЯ В ПРОСТРАНСТВЕ ИМЕН

Чтобы использовать класс, функцию или константу, находящиеся в пространстве имен, укажите пространство имен перед именем этого класса, функции или константы. Пространство имен действует как префикс.

Когда страница PHP использует код, находящийся в пространстве имен, она должна использовать **абсолютное** имя, состоящее из:

- обратный слеш, обозначающий глобальное пространство имен (точно так же, как слеш в начале пути к файлу указывает на корневую папку)[72];
- пространство имен для нужного класса;
- имя класса, функции или константы.

Строчка ниже взята из файла `bootstrap.php`. Она создает объект `CMS` с помощью класса `CMS` в пространстве имен `PhpBook\CMS`. Если бы пространство имен не было указано, то интерпретатор PHP искал бы класс `CMS` в глобальном пространстве имен и не смог бы его найти.

```
$cms = new \PhpBook\CMS\CMS($dsn, $username, $password);
```

ГЛОБАЛЬНОЕ ПРОСТРАНСТВО ИМЕН — ПРОСТРАНСТВО ИМЕН — ИМЯ КЛАССА

Когда объект `CMS` создает объект `Database`, он может быть создан с использованием абсолютного имени. Оно начинается с символа \, чтобы указать глобальное пространство имен, пространство имен `PhpBook\CMS`, затем имя класса.

```
\PhpBook\CMS\Database($dsn, $username, $password);
```

АБСОЛЮТНОЕ ИМЯ — ИМЯ КЛАССА

Но так делать не нужно, а следует просто использовать имя класса, как показано ниже, потому что класс `Database` находится в том же пространстве имен, что и объект `CMS` (а оба они находятся в пространстве имен `PhpBook\CMS`).

```
Database($dsn, $username, $password);
```

ИМЯ КЛАССА

При обращении же к любым встроенным классам PHP (и любым другим классам к глобальном пространстве имен) внутри класса `Database` (или, говоря шире, внутри любого пространства имен), ситуация обратная — к их именам необходимо всегда добавлять обратный слеш, поскольку иначе PHP будет искать их в текущем пространстве имен, `PhpBook\CMS`, и не найдет их (см. правую страницу).

Соответственно, при обращении к классу `PDO` или при обращении к его константам при настройке параметров соединения с БД, а также при обращении к классу `PDOException` (который используется для перехвата исключений `PDO` в классах `Article` и `Category`) им должен предшествовать символ \, чтобы сообщать интерпретатору PHP, что эти классы относятся к глобальному пространству имен[73].

ИСПОЛЬЗОВАНИЕ ПРОСТРАНСТВ ИМЕН В КЛАССАХ CMS

Здесь вы видите часть файла `bootstrap.php`, создающего объект `CMS`, используемого на каждой странице.

1. При создании объекта перед классом `CMS` добавляется пространство имен.

PHP c15/src/bootstrap.php

```
...
(1) $cms = new \PhpBook\CMS\CMS($dsn, $username, $password);  // Создание объекта CMS
```

Ниже приведено начало файла с определением класса `Database`. Он начинается с объявления пространства имен. Поскольку текущим пространством имен является `PhpBook\CMS`, а класс `PDO` находится в глобальном пространстве имен, то при обращении к нему используется обратный слеш.

1. Объявление пространства имен.

2. Использование обратного слеша перед именем класса PDO.

3. Использование обратного слеша перед константами класса PDO.

PHP c15/src/classes/CMS/Database.php

```
<?php
(1) namespace PhpBook\CMS;                  // Объявление пространства имен

(2) class Database extends \PDO
    {
        protected $pdo = null;              // Объявление переменной класса для объекта PDO

        public function __construct(string $dsn, string $username, string $password,
            array $options = [])
        {
(3)         $default_options[\PDO::ATTR_DEFAULT_FETCH_MODE] = \PDO::FETCH_ASSOC;
            $default_options[\PDO::ATTR_EMULATE_PREPARES]   = false;
            $default_options[\PDO::ATTR_ERRMODE]            = \PDO::ERRMODE_EXCEPTION;
            $options = array_replace($default_options, $options);
            parent::__construct($dsn, $username, $password, $options); // Создание объекта
        }...
```

ИМПОРТ КОДА В ПРОСТРАНСТВО ИМЕН

Чтобы не набирать вручную пространство имен при каждом использовании класса, вы можете импортировать класс в текущее пространство имен (используемое остальной частью страницы).

Класс `Validate` может послужить хорошим примером того, насколько удобным может оказаться такое импортирование. Методы этого класса вызываются при проверке формы по нескольку раз.

Вы можете каждый раз писать полное имя класса, как показано ниже, но это будет неудобно (оператор :: указывает на то, что метод статический).

```
\PhpBook\Validate\Validate::isText();
```

ПРОСТРАНСТВО ИМЕН · КЛАСС · МЕТОД

Или вы можете импортировать класс `Validate` в текущее пространство имен. Для этого добавьте ключевое слово `use`, пространство имен и имя класса в начало файла.

```
use \PhpBook\Validate\Validate;
```

ПРОСТРАНСТВО ИМЕН · КЛАСС

Теперь, когда класс был импортирован в текущее пространство имен, вы можете обращаться к нему без указания пространства имен.

```
Validate::isText();
```

КЛАСС · МЕТОД

Если бы в текущем пространстве имен уже существовал класс `Validate`, это привело бы к конфликту имен. Чтобы обойти это, вы можете добавить **псевдоним**, представляющий собой имя, которое можно использовать для ссылки на импортированный класс.

Затем этот псевдоним можно использовать вместо имени класса при создании объекта или вызове его методов. Чтобы создать псевдоним, добавьте ключевое слово `as` и укажите имя псевдонима, которое будет использоваться при его импорте.

```
use \PhpBook\Validate\Validate as FormValidate;
```

ПРОСТРАНСТВО ИМЕН · КЛАСС · ПСЕВДОНИМ

ИМПОРТ КЛАССА В ТЕКУЩЕЕ ПРОСТРАНСТВО ИМЕН

1. В начале файла с определением класса `Validate` добавляется указание пространства имен, `PhpBook\Validate`.

2. На странице `article.php`, используемой для создания и редактирования статей, инструкция `use` импортирует класс в текущее пространство имен.

3. Для вызова методов класса `Validate` используется имя класса с последующим указанием имени метода, точно так же как это делалось в предыдущей версии кода в главе 14, поскольку имя класса и его методы были импортированы в текущее пространство имен (являющееся глобальным пространством имен).

PHP c15/src/classes/Validate/Validate.php

```
<?php
① namespace PhpBook\Validate;                    // Указание пространства имен

class Validate
{...
```

PHP c15/pubic/admin/article.php

```
<?php
// Part A: Setup
declare(strict_types = 1);                       // Строгая типизация
② use PhpBook\Validate\Validate;                 // Импорт класса

include '../../src/bootstrap.php';               // Подключение настроек
...

    // Проверка, все ли данные верны, и создание сообщений об ошибках
    $errors['title']    = Validate::isText($article['title'], 1, 80)
        ? '' : 'Title should be 1 - 80 characters.';        // Валидация названия
    $errors['summary']  = Validate::isText($article['summary'], 1, 254)
        ? '' : 'Summary should be 1 - 254 characters.';     // Валидация описания
③   $errors['content']  = Validate::isText($article['content'], 1, 100000)
        ? '' : 'Content should be 1 - 100,000 characters.'; // Валидация содержания
    $errors['member']   = Validate::isMemberId($article['member_id'], $authors)
        ? '' : 'Not a valid author';                        // Валидация автора
    $errors['category'] = Validate::isCategoryId($article['category_id'], $categories)
        ? '' : 'Not a valid category';                      // Валидация категории
```

КАК ИСПОЛЬЗОВАТЬ БИБЛИОТЕКИ

Библиотеки позволяют разработчикам использовать код, который они сами или другие программисты уже написали для выполнения той или иной задачи. Инструмент под названием Composer помогает разработчикам управлять библиотеками, которые будут использоваться в коде сайта.

Использование библиотек, написанных другими программистами, избавляет вас от необходимости писать код с нуля для выполнения той же задачи.

Многие библиотеки представляют собой набор классов, реализующих функциональность, предлагаемую библиотекой (подобно тому, как встроенный класс PDO позволяет PHP работать с базой данных или как встроенный класс DateTime представляет дату и время и позволяет выполнять над ними стандартные вычисления).

Когда программисты используют библиотеку, им не нужно изучать, как именно PHP-код в этой библиотеке выполняет свою задачу, поскольку для пользования библиотекой достаточно изучить только:

- для чего служит библиотека;
- как подключить библиотеку на нужных страницах;
- как создавать объекты, реализующие функциональность, предлагаемую библиотекой;
- как обращаться к методам или свойствам этих объектов для выполнения требуемой задачи.

Как и большинство программ, библиотеки могут регулярно обновляться (или даже полностью переписываться). Каждая версия может добавлять новые функциональные возможности или исправлять ошибки, обнаруженные после предоставления общего доступа к библиотеке.

Когда библиотеки обновляются, им присваиваются новые номера версий:

- в обозначении основных версий используются целые числа: v1, v2, v3 и т. д. В крупных обновлениях может потребоваться изменить код на странице PHP, использующей библиотеку;
- незначительные обновления, версии с точкой, обычно содержат небольшие обновления и исправления ошибок. Для их обозначения используют точку и второе или третье число: v2.1, v2.1.1, v2.1.2, v2.3. Они с меньшей вероятностью повлияют на то, как используется библиотека.

Разработчики должны внимательно следить за тем, какие версии библиотек используются на сайте:

- если используется неправильная версия, это может привести к поломке сайта;
- если в библиотеке обнаружены ошибки или угрозы безопасности, ее необходимо обновить (в противном случае на сайте также появятся эти ошибки и уязвимости).

Обеспечение корректности библиотек и их версий раньше было весьма трудоемкой задачей, но инструмент под названием Composer упрощает этот процесс.

Примечание. Некоторые библиотеки используют код из других библиотек, поэтому программисты говорят, что они *зависят* от установки других библиотек.

ИСПОЛЬЗОВАНИЕ COMPOSER И ПАКЕТОВ

Библиотеки, предназначенные для работы с инструментом Composer, называются пакетами. На сайте `Packagist.org` публикуются пакеты, которые может использовать Composer.

Composer — это бесплатная программа, которую разработчики могут использовать для управления библиотеками и их версиями.

Чтобы библиотека работала с Composer, автор библиотеки должен поместить весь код библиотеки в одну папку и добавить в нее файл с именем `composer.json`. Этот файл содержит информацию о библиотеке и ее текущей версии, а также список ее зависимостей.

В совокупности папка, содержащая библиотеку, и файл `composer.json` называются **пакетом**.

Человек, создавший библиотеку, может позже внести пакет в список на веб-сайте http://packagist.org. Этот сайт похож на поисковую систему, помогающую людям находить библиотеки, которые могут быть им полезны. Веб-сайт Packagist известен как **репозиторий (каталог) пакетов**.

Если выпущена новая версия библиотеки, автор обновляет файл `composer.json`, обновляя номер версии. После этого он может либо дождаться, когда Packagist проиндексирует изменения автоматически, либо обновить информацию на этом сайте вручную.

Примечание. Используемые в этой главе библиотеки уже включены в загружаемый код для этой главы. Это нужно, чтобы код учебного веб-сайта работал, без необходимости что-либо догружать.

Когда разработчик использует Composer для загрузки библиотек или их обновленных версий, Composer выполняет следующие действия.

- Загружает содержащий библиотеку пакет. Пакет не хранится в Packagist. Обычно он размещается на сайте, предназначенном для размещения исходного кода, таком как GitHub или Bitbucket.
- Загружает другие библиотеки, от которых зависит пакет, если они еще не установлены (а также библиотеки, от которых они сами зависят, — и так далее по цепочке).
- Добавляет в корневую папку проекта файлы json, которые фиксируют список библиотек и их версии.

В предыдущей главе показано, как функция PHP `spl_autoload_register()` позволяет загружать файлы классов автоматически. Это избавляет программиста от необходимости вручную подключать определения классов на страницу, прежде чем класс можно будет использовать для создания объекта. Далее показано, как с Composer создает файл, автоматически загружающий определения классов для всех пакетов, установленных с помощью Composer.

Composer написан на языке PHP. Он загружает пакеты с библиотеками и разрешает конфликты зависимостей. Вся конфигурация используемых библиотек записывается в файлы json, с помощью которых при переносе кода на новый сервер можно будет скачать пакеты тех же самых версий.

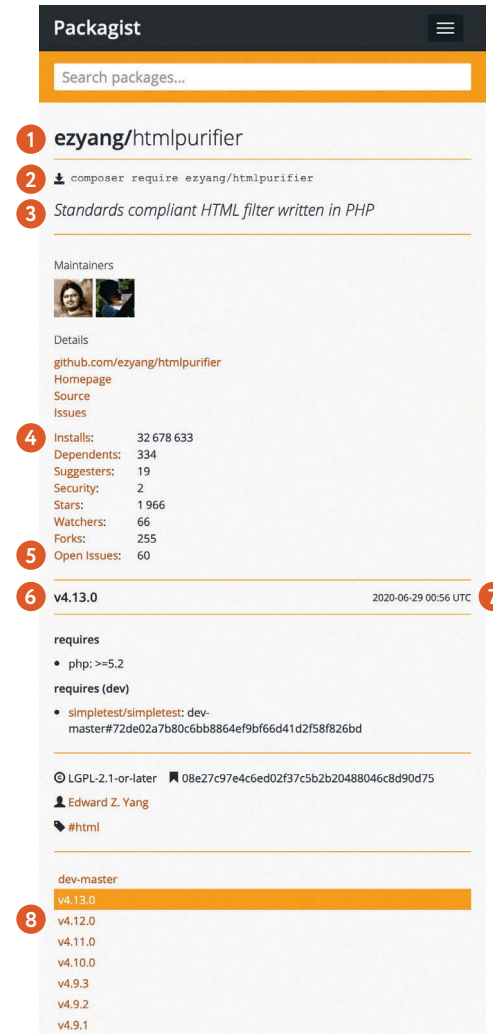
КАТАЛОГ ПАКЕТОВ PACKAGIST

Packagist — это веб-сайт, где зарегистрированы пакеты, с которыми может работать Composer. Он помогает программистам находить пакеты для применения в их собственных проектах.

Сайт Packagist работает как поисковая система. Вы вводите имя пакета или термин, связанный с типом выполняемой задачи (например, валидация), и Packagist отобразит список пакетов, имя или описание которых соответствуют этому термину.

Справа вы можете увидеть страницу библиотеки HTML Purifier. На этой странице показаны:

1) имя пакета;
2) инструкция по установке пакета;
3) информация о том, что делает пакет;
4) сколько раз он был установлен;
5) известные проблемы и ошибки;
6) последняя версия пакета;
7) дата выпуска;
8) предыдущие версии пакета.



Перед выбором пакета желательно убедиться, что он регулярно поддерживается. Чтобы сделать это, просмотрите на сайте Packagist следующую информацию:

- когда библиотека обновлялась в последний раз;
- сколько существует версий библиотеки;
- сколько у нее открытых (нерешенных) проблем.

Если нерешенных проблем много или пакет не обновлялся в последнее время, есть вероятность, что разработчик прекратил работу над библиотекой.
Использование такого пакета может быть рискованным, поскольку в нем могут быть неисправленные ошибки и уязвимости.

УСТАНОВКА COMPOSER И ПАКЕТОВ

Composer должен быть установлен на компьютере, на котором вы занимаетесь разработкой. У него нет графического пользовательского интерфейса, запуск производится из командной строки.

Чтобы установить Composer, перейдите на его веб-сайт **https://getcomposer.org/download/**.

Если вам нужна помощь в его установке, см. http://notes.re/installing_composer. После установки Composer откройте терминал (Mac) или командную строку (Windows) на вашем компьютере и перейдите в корневую папку вашего веб-сайта. Если вы не знакомы с этим, основные инструкции можно найти здесь http://notes.re/command-line.

Когда вы перейдете к корневой папке веб-сайта, введите слово Composer в командной строке и нажмите клавишу **Enter**. Перед вами будет список параметров и команд, принимаемых инструментом Composer[74].

```
Last login: Wed Nov  6 18:11:54 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Jons-MacBook-Pro:~ Jon$ composer
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 1.9.1 2019-11-01 17:20:17

Usage:
  command [options] [arguments]

Options:
  -h, --help                     Display this help message
  -q, --quiet                    Do not output any message
  -V, --version                  Display this application version
      --ansi                     Force ANSI output
      --no-ansi                  Disable ANSI output
  -n, --no-interaction           Do not ask any interactive question
      --profile                  Display timing and memory usage information
      --no-plugins               Whether to disable plugins.
  -d, --working-dir=WORKING-DIR  If specified, use the given directory as working directory.
      --no-cache                 Prevent use of the cache
  -v|vv|vvv, --verbose           Increase the verbosity of messages: 1 for normal output, 2 for more

Available commands:
  about                Shows the short information about Composer.
  archive              Creates an archive of this composer package.
  browse               Opens the package's repository URL or homepage in your browser.
  check-platform-reqs  Check that platform requirements are satisfied.
  clear-cache          Clears composer's internal package cache.
  clearcache           Clears composer's internal package cache.
  config               Sets config options.
  create-project       Creates new project from a package into given directory.
  depends              Shows which packages cause the given package to be installed.
  diagnose             Diagnoses the system to identify common errors.
```

Чтобы использовать пакет в проекте, найдите страницу пакета на веб-сайте Packagist. Имена пакетов состоят из двух частей: автора пакета (vendor) и названия проекта (они могут не отличаться), разделенных слешем. Приведенное ниже имя пакета относится к пакету под названием HTML Purifier.

```
ezyang/htmlpurifier
```

АВТОР ПРОЕКТ

Далее найдите инструкцию по установке пакета. Она показана под именем пакета на странице Packagist (пункт 2 на левой странице). Команда, с помощью которой можно установить HTML Purifier, выглядит так:

```
composer require ezyang/htmlpurifier
```

Здесь:

- `composer` — это имя программы Composer;
- `require` — команда Composer, которая загружает и устанавливает пакеты;
- и последним идет имя пакета, который должен быть установлен.

После того как вы открыли командную строку, перешли в корневую папку вашего веб-сайта и ввели инструкции с сайта Packagist, нажмите клавишу **Enter**. Composer загрузит последнюю версию библиотеки в папку в корневом каталоге сайта (см. следующую страницу).

Если для проекта требуется более одного пакета, повторите описанные выше действия для каждого из них.

УПРАВЛЕНИЕ ПАКЕТАМИ С ПОМОЩЬЮ COMPOSER

Когда Composer используется для установки пакета, от которого зависит сайт, он создает набор файлов и папок внутри корневого каталога. Они помогают инструменту отслеживать версии пакета и управлять ими.

Когда команда `require` используется для загрузки самого первого пакета, Composer добавляет ряд файлов и папок в корневой каталог сайта. Они показаны на скриншоте справа и описаны в таблице ниже. При установке дополнительных пакетов Composer обновляет эти файлы и папки.



ФАЙЛ / ПАПКА	НАЗНАЧЕНИЕ
`composer.json`	Файл с подробной информацией о пакетах, от которых зависит этот PHP-проект
`composer.lock`	Файл с данными о версиях пакетов и о том, откуда они были загружены
`vendor/`	Папка в корневом каталоге, используемая для хранения пакетов
`vendor/autoload.php`	Включаемый на страницы файл. Он автоматически загружает классы для пакетов
`vendor/composer/`	Папка с файлами, используемая инструментом Composer для реализации автозагрузки классов

Чтобы обновить все используемые сайтом пакеты, перейдите в командную строку, перейдите в корневой каталог проекта и введите следующую команду: `composer update`.

Здесь:

- `composer` — это имя программы Composer;
- `update` — команда Composer, которая обновляет пакеты.

Composer проверяет наличие последних версий всех установленных в данный момент пакетов и заменяет существующие пакеты более новыми. Он также обновляет созданные им файлы (включая файл `composer.lock`, в котором записывается версия пакета, используемого сайтом).
После обновления всех пакетов необходимо тщательно протестировать сайт, прежде чем запускать его в эксплуатацию, поскольку обновления иногда могут привести к нарушению работы сайта.

Чтобы обновлять по одному пакету за раз, укажите имя пакета после инструкции `update`. Следующее выражение приведет только к обновлению пакета HTML Purifier: `composer update ezyang/htmlpurifier`.

Если сайту больше не нужно использовать пакет, вы можете использовать команду `remove`, за которой следует имя неиспользуемого пакета. Это приведет к удалению файлов пакета из каталога `vendor` и обновлению других файлов, созданных Composer: `composer remove ezyang/htmlpurifier`.

Если для работы пакета требуется другой пакет, Composer также загрузит и его. Например, для пакета Twig требуются пакеты из папки `symfony`, которые будут загружены одновременно с Twig.

Composer создает файл с именем `autoload.php` для автоматической загрузки классов любого установленного пакета. Его также можно отредактировать для автоматической загрузки пользовательских классов в папке `src/classes`.

Об автозагрузке рассказывалось в главе 14. Composer создает файл `autoload.php` для автозагрузки классов всех установленных пакетов. В загружаемом коде для текущей главы этот файл был включен в файл `bootstrap.php`:

```
require APP_ROOT.'/vendor/autoload.php';
```

Кроме того, для загрузки собственных классов вы также можете вручную добавить информацию об автозагрузке кода в `composer.json`, чтобы указать Composer, как необходимо загружать ваши пользовательские классы. Это заменяет вызов функции `spl_autoload_register()` с анонимной функцией в файле `bootstrap.php`, который использовался в предыдущей главе. Приведенный ниже код, выделенный серым цветом, взят из файла `composer.json`. Composer создал его при добавлении пакетов в этой главе. Код, выделенный зеленым цветом, был добавлен в этот файл вручную. Он описывает механизм, который должен использовать Composer для автозагрузки пользовательских классов.

```
{
    "require": {
        "ezyang/htmlpurifier": "^4.12",
        "twig/twig": "^3.0",
        "phpmailer/phpmailer": "^6.1"
    },
    "autoload": {
        "psr-4": {
            "PhpBook\\": "src/classes/"
        }
    }
}
```

После добавления этих изменений в файл `composer.json`, необходимо перейти в корневой каталог проекта и ввести следующую команду для регенерации файлов автозагрузчика: `composer dump-autoload`.

Файл `composer.json` использует формат JavaScript Object Notation (JSON). Чтобы выполнить автозагрузку ваших собственных классов с помощью Composer, их названия должны следовать набору рекомендаций, называемых PSR-4, созданных группой PHP-FIG. Классы в загружаемом коде для этого раздела уже следуют этим рекомендациям. В загружаемом коде для этой главы все нужные пакеты уже установлены и находятся в папке `vendor`. Это сделано для того, чтобы все примеры работали с самого начала. Кроме того, папки для всех оставшихся глав содержат файлы `composer.json` и `composer.lock`. Чтобы увидеть своими глазами, как инструмент Composer загружает пакеты и создает дополнительные файлы и папки.

1. Создайте новую папку на вашем компьютере.
2. Держите эту папку открытой и на виду.
3. Откройте командную строку.
4. Перейдите к новой папке в командной строке.
5. Введите команду *composer*, чтобы убедиться, что он работает.
6. Найдите желаемый пакет на сайте Packagist.
7. Введите его команду установки в командную строку. Например, эти три команды загружают пакеты для этой главы:

```
composer install ezyang/htmlpurifier
composer install twig/twig
composer install phpmailer/phpmailer
```

При загрузке каждого пакета файлы и папки будут добавляться в папку `vendor` в созданной вами папке.

РАЗРЕШЕНИЕ ВВОДА HTML-КОНТЕНТА С ПОМОЩЬЮ HTML PURIFIER

Библиотека HTML Purifier может удалять код, вызывающий XSS-атаки. Его использование позволяет посетителям создавать контент, содержащий некоторые HTML-теги, но любая потенциально опасная разметка будет удалена.

До сих пор в учебной CMS введенный пользователем текст экранировался при выводе на страницу, чтобы предотвратить XSS-атаки. Это включало в себя замену пяти зарезервированных символов HTML-сущностями и означало, что пользователи не могли создавать контент, содержащий HTML-разметку.

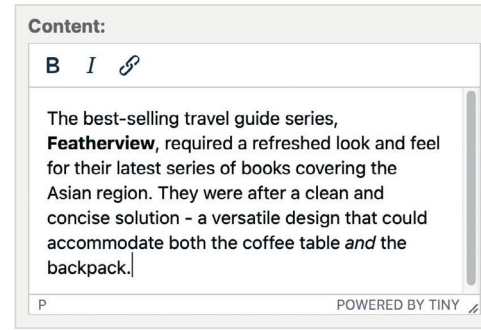
В этом разделе вы узнаете, как разрешить пользователям добавлять в статьи некоторые основные HTML-теги и атрибуты. В учебной CMS будет разрешено добавлять в текст абзацы, жирный и курсивный шрифт, ссылки и изображения.

PHP-код, требуемый для удаления разметки, вызывающей XSS-атаку, очень сложен. Поэтому, вместо того чтобы пытаться написать его самостоятельно, вы можете использовать готовый пакет под названием **HTMLPurifier**. Он позволит вам выполнить эту задачу всего в две строки кода.

Пакет HTML Purifier находится в папке `vendor` в загружаемом коде для этой главы и указан как обязательный пакет в файле `composer.json`. Его название на сайте Packagist `ezyang\htmlpurifier`.

Поскольку задачи, которые должен выполнять HTML Purifier для удаления нежелательной разметки, довольно сложны, он будет использоваться для удаления потенциально опасного HTML-кода при сохранении публикации (а не при каждом ее отображении). Это экономит ресурсы сервера, поскольку статьи просматриваются чаще, чем создаются или редактируются.

Ниже показана страница `article.php` в разделе администрирования, где создаются или редактируются публикации. Текст описания вводится в базовом визуальном редакторе, созданном с использованием библиотеки JavaScript под названием **TinyMCE**. Она заменяет HTML-элемент `<textarea>` редактором, показанным ниже.



Когда форма отправлена, HTML Purifier удаляет разметку, способную вызвать XSS-атаку, из содержимого статьи, прежде чем она будет сохранена в базе данных.

Когда статья отображается на странице, ее нельзя экранировать. Любой содержащийся HTML-код теперь будет безопасен для отображения.

Встроенная в PHP функция `strip_tags()` удаляет HTML-теги из текста, но не всегда удаляет атрибуты, поэтому она не может предотвратить XSS-атаки.

Чтобы удалить разметку, способную вызвать XSS-атаку, сначала создайте объект класса `HTMLPurifier`, а затем вызовите его метод `purify()`.

1. Когда файл PHP принимает текст, который может содержать разметку, создайте объект класса `HTMLPurifier`. У библиотеки HTML Purifier нет собственного пространства имен, то есть она относится к глобальному пространству имен. Ниже экземпляр класса `HTMLPurifier` присваивается переменной с именем `$purifier`.

2. Затем вызывается метод `purify()`. Его единственным аргументом будет подлежащая очистке строка, потенциально содержащая опасную разметку. Этот метод удалит любую разметку, представляющую риск возникновения XSS, а также любые теги или атрибуты, которых нет в XHTML 1.0, а затем вернет текст без этой разметки.

```
1 $purifier = new HTMLPurifier();
2 $text = $purifier->purify($text);
```

Объект `HTMLPurifier` содержит свойство с именем `config`. В нем содержится другой объект, который настраивает параметры, управляющие работой HTML Purifier. Например, вы можете указать, что разрешены только определенные теги и атрибуты, и все остальные тогда будут удалены.

После создания объекта `HTMLPurifier` вы можете изменить настройки объекта конфигурации, используя его метод `set()`, содержащий два аргумента:

- подлежащее обновлению свойство;
- используемые для замены значения.

Например, свойство `HTML.Allowed` указывает, какие HTML-теги и атрибуты разрешены.

Теги, появление которых допускается в тексте, должны быть указаны в виде списка имен тегов, разделенных запятыми (без их угловых скобок). Например, чтобы разрешить эти теги: `<p>`, `<br>`, `<a>` и `<img>`, значение свойства `HTML.Allowed` должно выглядеть так: `p, br, a, img`.

Чтобы разрешить атрибуты для элемента, поместите разрешенные имена атрибутов в квадратные скобки после имен элементов. Чтобы разрешить более одного атрибута для элемента, отделите каждый из них с помощью прямой черты `|`.

Например, чтобы разрешить эти атрибуты для тегов `<a>` и `<img>`: `<a href="">` и `<img alt="" src="">`, значение свойства `HTML.Allowed` будет выглядеть так: `p, br, a[href], img[src|alt]`.

```
$purifier->config->set('HTML.Allowed', 'p,br,a[href],img[src|alt]');
```

ДОБАВЛЕНИЕ БИБЛИОТЕКИ HTML PURIFIER В CMS

Чтобы пользователи могли добавлять базовую разметку к содержимому статьи, страница `article.php` в разделе администрирования должна быть обновлена.

1. Страница `article.php` в разделе администрирования используется для создания или редактирования публикаций. Код, который принимает данные формы, показан в главе 13.
2. После получения данных создается объект класса `HTMLPurifier`.
Ему не требуется указание пространства имен, поскольку он находится в глобальном пространстве имен, которое и используется в скрипте.

Файл автозагрузки, созданный инструментом Composer и подключенный в файле `bootstrap.php`, гарантирует, что необходимые определения классов автоматически включаются на страницу при создании объекта.

3. Параметр `HTML.Allowed` объекта конфигурации `HTMLPurifier` получает значение, в котором указано, какие теги и атрибуты разрешено отображать в разметке.
4. Вызов метода `purify()` объекта `HTMLPurifier` для удаления всех тегов и атрибутов из содержимого, кроме указанных в шаге 3.
5. Поскольку содержимое теперь очищено, его не следует экранировать при отображении на странице, так как больше нет риска XSS-атаки.

Примечание. Как можно заметить, файл `article.php` в загружаемом коде не содержит HTML-кода, поскольку теперь используются шаблоны Twig.

На странице администрирования публикации отображается базовый визуальный редактор для ввода описания, который содержит кнопки, позволяющие пользователям изменять шрифт на жирный или курсивный и добавлять ссылки. Он создается с использованием библиотеки JavaScript под названием `TinyMCE`.

Вам нужно будет зарегистрироваться, чтобы использовать бесплатную версию редактора на сайте разработчика https://tiny.cloud. Если вы не создадите учетную запись для использования редактора, на ваших веб-страницах появится сообщение, что продукт не зарегистрирован, пока вы не создадите учетную запись в разделе templates/admin/layout.html.

6. HTML-тег `<script>` загружает редактор `TinyMCE`. Вам стоит заменить этот тег на указанный при регистрации на `TinyMCE`, потому что он содержит ключ API. Ключ API идентифицирует вашу учетную запись и отображается там, где написано ***no-api-key***.
7. Шаблон `layout.html` используется на всех страницах администрирования. Таким образом, на каждой странице оператор JavaScript `if` проверяет, содержит ли текущая страница элемент, чей атрибут `id` равен `article-content` (это идентификатор элемента `<textarea>`, в который выводится текст описания публикации).
8. Если да, функция `TinyMCE.init()` заменит этот элемент `<textarea>` редактором.

Существует множество настроек, управляющих такими параметрами, как внешний вид редактора или тем, какие функции или кнопки отображаются на панели инструментов. Чтобы узнать больше об этих параметрах, посетите веб-сайт `TinyMCE`.

PHP c15/public/admin/article.php

```
...
if ($_SERVER['REQUEST_METHOD'] == 'POST') {                    // Если форма отправлена
    $article['title']       = $_POST['title'];                 // Название
    $article['summary']     = $_POST['summary'];               // Краткое описание
    $article['content']     = $_POST['content'];               // Содержимое
    $article['member_id']   = $_POST['member_id'];             // Получение member_id
    $article['category_id'] = $_POST['category_id'];           // Получение category_id
    $article['image_id']    = $article['image_id'] ?? null;    // ID изображения
    $article['published']   = (isset($_POST['published'])) ? 1 : 0; // Отображение навигации

    $purifier = new HTMLPurifier();                            // Создание Purifier
    $purifier->config->set('HTML.Allowed', 'p,br,strong,em,a[href],img[src|alt]'); // Теги
    $article['content'] = $purifier->purify($article['content']);   // Содержание Purify
... }
  <!-- ПРИМЕЧАНИЕ. Форма перемещается в новый файл позже в главе; ее нет в этом файле -->
  <div class="form-group">
    <label for="content">Content: </label>
    <textarea name="content" id="article-content" class="form-control">
      <?= $article['content'] ?>
    </textarea>
    <span class="errors"><?= $errors['content'] ?></span>
  </div>
```

TWIG c15/templates/admin/layout.html

```
    ...
    <script src="https://cdn.tiny.cloud/1/no-api-key/tinymce/5/tinymce.min.js"
                referrerpolicy="origin"></script>
    <script>
      if (document.getElementById('article-content')){
        tinymce.init({
          menubar: false,
          selector: '#article-content',
          toolbar: 'bold italic link',
          plugins: 'link',
          target_list: false,
          link_title: false
        });
      }
    </script>
  </body>
</html>
```

Далее вы увидите, как изменился шаблон `article.html` в общедоступной области сайта, чтобы не делать экранирование разметки в содержимом публикации (которое уже было защищено с помощью HTML Purifier).

ШАБЛОНИЗАТОР TWIG

Шаблонизаторы отделяют PHP-код, получающий и обрабатывающий данные, от кода, используемого для создания HTML-страниц, отправляемых в браузеры. В этой книге используется шаблонизатор под названием Twig.

До сих пор в этой книге каждая страница PHP состояла из двух частей.

- Первая использует PHP-код для получения и обработки данных. В этой части хранятся данные, которые должны быть показаны посетителям в переменных, чтобы их можно было отобразить во второй части страницы.
- Вторая часть создает HTML-разметку для отправки посетителю. В ней используются переменные, получившие значения в первой части страницы.

Когда сайт использует шаблонизатор, PHP-код для получения и обработки данных остается в том же файле. Но вторая часть, создающая HTML для отображения посетителям, перемещается в набор файлов, называемых **шаблонами**. Программисты говорят, что так код разделяется на:

- код **приложения** (PHP-код, выполняющий задачи, необходимые для работы сайта);
- код **представления** (код, который создает HTML-разметку, отображаемую для посетителей).

Это разделение особенно популярно на веб-сайтах, где разные разработчики отвечают за:

- **бэкенд** (выполняемый на сервере PHP-код);
- **фронтенд** (код, выполняемый в браузере).

Поскольку код, на которым они работают, физически разделен, существует меньше шансов, что они испортят код друг друга.

Шаблоны Twig не используют PHP-код для отображения данных. Они используют другой синтаксис с меньшим набором команд, который многие люди считают более простым для изучения фронтенд-разработчиком, чем PHP. Например, чтобы вывести содержимое переменной с именем `$title` в PHP, вы можете написать:

```
<p><?= htmlspecialchars($title); ?><p>
```

В то время как в шаблоне Twig используется такая запись:

```
<p>{{ title }}<p>
```

Фигурные скобки сообщают Twig, что он должен вывести значение переменной `title`.

Примечание. Имена переменных в Twig не начинаются со знака $.

Использование Twig в качестве шаблонизатора также повышает безопасность сайта, поскольку:

- Twig может автоматически заменять любые зарезервированные символы HTML на сущности, чтобы исключить риск XSS-атаки. То есть безопасность не зависит от того, помнит ли разработчик об использовании `htmlspecialchars()` при выводе каждой переменной;
- любой PHP-код в шаблоне игнорируется, поэтому нет никакого риска, что фронтенд-разработчик случайно добавит небезопасный PHP-код в шаблон.

Шаблоны обеспечивают повторное использование кода, используя метод, называемый **наследованием,** который является альтернативой подключению файлов. Это гораздо удобнее при работе с основным шаблоном сайта, поскольку он весь помещается в одном файле.

В предыдущих главах заголовок и футер каждой страницы хранились в двух разных подключаемых файлах. Это означало, что открывающие и закрывающие теги находились в разных файлах.

В каждую страницу надо было подключать эти два файла, и их код копировался на то место, где в коде был размещен оператор `include` или `require`.

Include: `header.php`

```
<html>
  <head> ... </head>
  <body>
```

Include: `footer.php`

```
  </body>
</html>
```

Page: `category.php`

```
<?php include 'header.php'; ?>
<h1>Content goes here</h1>
<?php include 'footer.php'; ?>
```

Twig использует один **родительский шаблон** для всего кода, отображаемого на каждой странице. Его легче редактировать, потому что открывающие и закрывающие теги находятся в одном файле. Родительский файл содержит **блоки**, которые **дочерний шаблон** может перезаписать.

Дочерние шаблоны **расширяют** код в родительском шаблоне. Они основаны на коде из родительского шаблона и могут перезаписывать блоки, определенные в родительском шаблоне. Ниже блок содержимого будет перезаписан.

Parent Template: `layout.html`

```
<html>
  <head> ... </head>
  <body>
    {% block content %}
    <h1>Content goes here</h1>
    {% endblock %}
  </body>
</html>
```

Child Template: `category.html`

```
{% extends 'layout.html' %}

{% block content %}
<h1>This replaces anything that was
    in the content block in the
    layout.html file.</h1>
{% endblock %}
```

ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ TWIG ДЛЯ ОТРИСОВКИ ШАБЛОНА

Шаблонизатору передаются данные в виде массива, которые он затем выводит в нужных местах шаблона для получения HTML-кода, который отправляется в браузер. Это называется **рендерингом** (отрисовкой) шаблона.

Существует несколько шаблонизаторов, написанных на PHP. В этой книге используется один из них, под названием Twig. Его название на Packagist: `Twig\Twig`. Его код находится в папке `vendor` в загружаемом коде для этой главы, так как он указан как обязательный в файле `composer.json`.
Каждая страница, использующая Twig, должна создавать два объекта из библиотеки Twig.

1. Класс Twig\Loader\FilesystemLoader создает объект **loader** для загрузки файлов шаблонов. Ему нужен путь к папке с шаблонами.
2. Класс Twig\Environment создает объект **среды** Twig. Это основной класс шаблонизатора. Для загрузки шаблонов ему необходим объект Twig loader.

ПУТЬ К ФАЙЛАМ ШАБЛОНА

```
① $loader = new Twig\Loader\FilesystemLoader(APP_ROOT . '/templates');
② $twig   = new Twig\Environment($loader);
```

ОБЪЕКТ LOADER

Затем метод `render()` объекта среды Twig загружает шаблон и создает HTML-код. HTML-код, возвращаемый методом `render()`, выводится на страницу с помощью команды PHP `echo`.

Метод `render()` использует два параметра:
- шаблон, используемый для отображения страницы;
- данные, которые будут вставлены в шаблон.

```
echo $twig->render('member.html', $data);
```

ОТПРАВКА HTML-КОДА В БРАУЗЕР — ШАБЛОН ДЛЯ ИСПОЛЬЗОВАНИЯ — ДАННЫЕ ДЛЯ ШАБЛОНА

За кулисами Twig загружает файл шаблона и преобразует команды Twig в PHP-код.

Затем интерпретатор PHP выполняет этот код и возвращает полученный в результате HTML, который затем выводится посетителю.

ПАРАМЕТРЫ TWIG

Как и во многих библиотеках, в Twig есть настройки для управления тем, как он работает. Параметры хранятся в массиве, передаваемом в качестве аргумента при создании объекта среды Twig (точно так же, как при работе с объектом PDO).

Ниже переменная с именем `$twig_options` содержит массив. У него есть два ключа, представляющих собой параметры для управления поведением объекта среды Twig. Значение для каждого ключа — настройка для этого параметра. Массив `$twig_options` затем используется в качестве второго аргумента при создании объекта среды Twig.

```
$twig_options['cache'] = APP_ROOT . '/templates/cache';
$twig_options['debug'] = DEV;

$loader = new Twig\Loader\FilesystemLoader(APP_ROOT . '/templates');
$twig   = new Twig\Environment($loader, $twig_options);
```

ОБЪЕКТ LOADER — ПАРАМЕТРЫ СРЕДЫ

КЭШИРОВАНИЕ

За кулисами Twig загружает файл шаблона и преобразует команды Twig в PHP-код. Этот PHP-код может быть **кэширован**, то есть он сохраняется в файл на сервере, а не создается заново каждый раз при запросе страницы. Это ускоряет загрузку шаблонов и экономит ресурсы сервера.

Чтобы включить кэш, параметру `cache` требуется указать абсолютный путь к той папке и месту, где должны храниться готовые файлы. Выше этот путь указан в первом параметре. По умолчанию Twig не включает кэширование.

ОТЛАДКА

Пока сайт находится в разработке, параметр, включающий отладку, позволяет выводить дополнительную отладочную информацию. Заданная в файле `config.php` константа `DEV` используется для установки значения параметра `debug`.

СТРОГИЙ КОНТРОЛЬ ПЕРЕМЕННЫХ

Если шаблон Twig использует переменную, которая не была определена на странице PHP, то по умолчанию Twig создает переменную и присваивает ей значение `null`. Однако это поведение можно изменить, чтобы при попытке обратиться к несуществующей переменной выбрасывалось исключение.

Для этого добавьте ключ с именем `strict_variables` в массив `$twig_options` со значением `true`.

ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ И РАСШИРЕНИЯ

Если в шаблон Twig добавить PHP-код, этот код не будет выполняться. Но нужного результата можно добиться, используя расширения (extensions), которые дополняют его функциональность. Расширение может быть либо уже готовое, либо написано самим программистом. Кроме того, в Twig можно добавить глобальные переменные, которые будут доступны во всех шаблонах.

Глобальные переменные можно использовать, когда большинству шаблонов потребуется доступ к какому-либо значению. Например, созданная в файле `config.php` константа `DOC_ROOT` помогает создавать правильные пути для изображений, таблиц стилей, скриптов и других файлов, запрашиваемых браузером.

Шаблоны Twig не имеют доступа к константам PHP, но нужные значения можно присвоить глобальным переменным Twig, что позволяет использовать их в шаблонах. Чтобы создать глобальную переменную, вызовите метод `addGlobal()` объекта среды Twig с двумя аргументами: именем переменной и значением, которое она должна содержать.

```
$twig->addGlobal('doc_root', DOC_ROOT);
```

ИМЯ ГЛОБАЛЬНОЙ ПЕРЕМЕННОЙ — ЗНАЧЕНИЕ

Поскольку шаблоны Twig не используют PHP, они не могут использовать функцию `var_dump()` для проверки значений переменных и их типа данных. В Twig есть расширение под названием `debug`, позволяющее использовать в шаблонах метод с именем `dump()` для выполнения этой задачи.

Расширение `debug` реализовано в виде объекта. Ниже, если значение константы `DEV` равно `true`, вызывается метод `addExtension()` среды Twig, чтобы добавить расширение, и его аргументом будет создаваемый на лету объект `DebugExtension`.

```
if (DEV){
    $twig->addExtension(new \Twig\Extension\DebugExtension());
}
```

ИМЯ КЛАССА С УКАЗАНИЕМ ПРОСТРАНСТВА ИМЕН

Единственный аргумент для функции Twig `dump()`, похожей на функцию PHP `var_dump()`, — это имя переменной, содержимое которой вы хотите просмотреть.

Функция `dump()` может отображать содержимое переменной только в том случае, если включен параметр `debug` (см. предыдущую страницу).

ПОДКЛЮЧЕНИЕ TWIG В ФАЙЛЕ НАЧАЛЬНОЙ ЗАГРУЗКИ

Поскольку каждая страница сайта использует шаблоны Twig, загрузчик Twig и объекты среды создаются в файле `bootstrap.php`.

1. Подключение файла `autoload.php` из папки `vendor`, который был создан Composer для автоматической загрузки классов установленных пакетов.
2. Параметр `cache` указывает Twig кэшировать PHP-файлы, которые он создает для каждого шаблона.
3. Параметр `debug` используется для включения режима отладки Twig, когда константа `DEV` содержит значение `true`.
4. Создается объект Twig `loader`. Ему нужен путь к папке, содержащей файлы шаблонов.
5. Создается объект среды Twig. Он присваивается переменной с именем `$twig`, которую можно использовать на любой странице сайта (так же, как объект CMS в переменной `$cms`). Методу конструктора нужны два аргумента:

- объект `loader` для загрузки файлов шаблонов;
- массив параметров для объекта среды (он был сохранен в `$twig_options`).

6. Добавление глобальной переменной Twig с именем `doc_root`. Она содержит путь от реального корня сайта к учебному сайту, представляющему код для данной главы.
7. Конструкция `if` проверяет, присвоено ли константе `DEV` значение `true`.
8. Если это так, то загружается расширение `debug`, чтобы шаблоны могли использовать функцию `dump()`.

PHP c15/src/bootstrap.php

```
<?php
define("APP_ROOT", dirname(__FILE__, 2));              // Корневой каталог

require APP_ROOT . '/config/config.php';               // Настройки сайта
require APP_ROOT . '/src/functions.php';               // Функции
(1) require APP_ROOT . '/vendor/autoload.php';         // Автозагрузка классов

...

(2) $twig_options['cache'] = APP_ROOT . '/var/cache';      // Путь к папке Twig cache
(3) $twig_options['debug'] = DEV;  // Если активен режим разработки, включить режим отладки

(4) $loader = new Twig\Loader\FilesystemLoader([APP_ROOT . '/templates']); // Объект Twig loader
(5) $twig   = new Twig\Environment($loader, $twig_options); // Среда Twig
(6) $twig->addGlobal('doc_root', DOC_ROOT);                // Виртуальный корень сайта
(7) if (DEV === true) {                                    // Если в разработке
(8)     $twig->addExtension(new \Twig\Extension\DebugExtension()); // Добавить расширение debug
}
```

ОБНОВЛЕНИЕ PHP-ФАЙЛОВ ДЛЯ ИСПОЛЬЗОВАНИЯ TWIG

Страницы PHP, запрашиваемые посетителями, запрашивают данные из БД. Полученные данные передаются в Twig в виде массива. Затем метод `render()` применяется для создания HTML-кода, отображаемого посетителям.

Страницы PHP, которые запрашивают данные из БД, а затем сохраняют их в переменных, очень похожи на первую часть страниц из предыдущей главы.

Первое отличие заключается в том, что переменные попадают в шаблон не по отдельности, а в виде ассоциативного массива.

Например, на странице категории код получает из БД и сохраняет в массиве следующие данные:

- список всех категорий для создания навигации;
- название и описание выбранной категории;
- краткое описание всех статей в категории;
- идентификатор категории, которую нужно выделить в навигации.

ИНДЕКСИРОВАННЫЙ МАССИВ → `$data['navigation'] = $cms->getCategory()->getAll();`
АССОЦИАТИВНЫЙ МАССИВ → `$data['category'] = $cms->getCategory()->get($id);`
ИНДЕКСИРОВАННЫЙ МАССИВ → `$data['articles'] = $cms->getArticle()->getAll(true, $id);`
ЦЕЛОЕ ЧИСЛО → `$data['section'] = $category['id'];`

Как только все данные, необходимые в шаблоне, будут сохранены в массиве, вызывается метод объекта среды Twig `render()`.

Метод `render()` выводит данные из массива в нужных местах шаблона, а затем полученный HTML-код выводится в браузер с помощью команды PHP `echo`.

```
echo $twig->render('category.html', $data);
```

ОТПРАВКА HTML-КОДА В БРАУЗЕР — ШАБЛОН ДЛЯ ИСПОЛЬЗОВАНИЯ — ДАННЫЕ ДЛЯ ШАБЛОНА

На странице справа вы можете увидеть, как были изменены два файла PHP, сохраняя предназначенные для шаблона данные в массиве `$data`.

Это делает код обеих страниц гораздо проще, чем когда они содержали еще и HTML-разметку.

PHP-ФАЙЛЫ, ПОЛУЧАЮЩИЕ И ОТОБРАЖАЮЩИЕ ДАННЫЕ

Файлы PHP в папке `public` начинаются с тех же инструкций, что в главе 14. Но далее код меняется следующим образом.

1. Данные, получаемые из БД с использованием методов объекта `CMS`, помещаются в массив `$data`.

2. Метод `render()` объекта среды Twig заполняет шаблон данными, хранящимися в массиве `$data`.

Возвращаемый им HTML-код выводится в браузер с помощью команды `echo`.

PHP c15/public/index.php

```php
<?php
declare(strict_types = 1);                                  // Строгая типизация
require_once '../src/bootstrap.php';                        // Настройки

$data['articles']   = $cms->getArticle()->getAll(true, null, null, 6); // Самые новые
                                                                       // публикации
$data['navigation'] = $cms->getCategory()->getAll();        // Все категории

echo $twig->render('index.html', $data);                    // Рендеринг шаблона
```

PHP c15/public/article.php

```php
<?php
declare(strict_types = 1);                                  // Строгая типизация
require_once '../src/bootstrap.php';                        // Настройки

$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT); // Валидация идентификатора
if (!$id) {                                          // Если нет валидного идентификатора
    include APP_ROOT . '/public/page-not-found.php';        // Страница не найдена
}
$article = $cms->getArticle()->get($id);                    // Получение публикации
if (!$article) {                                            // Если такой нет
    include APP_ROOT . '/public/page-not-found.php';        // Получение категорий
}

$data['navigation']  = $cms->getCategory()->getAll();       // Получение категорий
$data['article']     = $article;                            // Данные публикации
$data['section']     = $article['category_id'];             // Текущая категория

echo $twig->render('article.html', $data);                  // Рендеринг шаблона
```

ДОСТУП К ДАННЫМ В ШАБЛОНАХ TWIG

Шаблон обрабатывает каждый элемент массива $data как отдельную переменную, которую может использовать шаблон. Переменные Twig не начинаются с символа $.

Если на странице PHP создан следующий ассоциативный массив, содержащий три элемента:

```
$data['name']    = 'Ivy Stone';
$data['joined']  = '2021-01-26 12:04:23';
$data['picture'] = 'ivy.jpg';
```

то в шаблоне Twig будут доступны три переменные, имена которых совпадают с ключами массива $data (помните, имена переменных в Twig не начинаются с символа "$"):

- `name`;
- `joined`;
- `picture`.

Если значение для одного из элементов — это другой массив (а не скалярное значение), шаблон Twig тоже будет рассматривать эту переменную как массив.

```
$data['category']['name'] = 'Illustration';
$data['category']['description'] =
    'Hand-drawn visual storytelling';
$data['category']['published'] = true;
```

Чтобы обратиться к элементу этого массива, используйте имя переменной (category), точку, затем имя ключа. Например, чтобы получить имя, описание и статус категории:

- `category.name`;
- `category.description`;
- `category.published`.

Если элемент массива $data содержит объект, тот же самый синтаксис через точку используется для доступа к значениям, хранящимся в свойствах этого объекта.

По умолчанию, если шаблон использует переменную, не созданную в массиве $data, Twig обрабатывает ее так, как если бы она была создана и содержала значение `null`, вместо того чтобы генерировать ошибку с сообщением `Undefined variable`.

Это может быть полезно, поскольку шаблону не нужно предоставлять альтернативное значение для переменных, которые, возможно, не были созданы. Но при необходимости это поведение можно изменить.

ВЫВОД ДАННЫХ В ШАБЛОНАХ TWIG

Файлы шаблонов Twig состоят из HTML-тегов и команд Twig. Двойные фигурные скобки {{}} указывают Twig вывести переменную или результат выражения.

Чтобы отобразить содержащееся в переменной значение, имя переменной пишется между двойных фигурных скобок.

Если переменная содержит какой-либо из зарезервированных символов HTML, Twig автоматически экранирует его.

```
<h1>{{ category.name }}</h1>
<p>{{ category.description }}</p>
```

Twig содержит набор **фильтров**, которые могут видоизменять данные или поведение шаблонизатора при выводе переменной. Например, чтобы отключить автоматическое экранирование HTML, используется фильтр `raw`. В частности, нам это потребуется для вывода содержимого публикации, поскольку оно уже содержит безопасный вариант HTML-кода, который был обработан с помощью HTML Purifier. Чтобы использовать фильтр, добавьте символ прямой черты | после имени переменной, а затем имя фильтра.

```
<p>{{ article.content|raw }}</p>
```

Если фильтру требуются данные для выполнения задачи, они указываются в круглых скобках после имени фильтра (как при вызове функций). Например, фильтр даты может форматировать дату. Он:

- работает с теми же форматами дат, что и встроенная в PHP функция `strtotime()`;
- форматирует эти даты, используя те же значения, что и встроенная в PHP функция `date()`.

```
<p>{{ article.created|date('M d Y') }}</p>
```

Поскольку Twig может форматировать даты, функция `format_date()` была удалена из файла `functions.php`.

Фильтр `e` экранирует выводимое значение, чтобы его можно было безопасно отображать на странице. Его параметр указывает фильтру, где будут использоваться данные. Это важно, потому что в HTML, CSS, JavaScript и URL-адресах используются разные зарезервированные символы. По этой причине необходимо экранировать разные символы при использовании данных в каждом из этих разных контекстов.

```
<p>{{ article.summary|e('html_attr') }}</p>
```

ЗНАЧЕНИЕ	КОНТЕКСТ
`html`	Содержимое «тела» HTML
`html_attr`	Значение атрибутов HTML
`css`	CSS
`js`	JavaScript
`url`	Текст, который становится частью URL-адреса

В CMS используются только фильтры `raw`, `date` и `e`, но у Twig есть и другие, форматирующие числа, время и валюта, изменяющие регистр текста в строке и сортирующие, объединяющие или разделяющие значения в массивах.

ИСПОЛЬЗОВАНИЕ УСЛОВИЙ В ШАБЛОНАХ TWIG

Фигурные скобки и символы процента используются для создания открывающих тегов {% и закрывающих тегов %}. Они сообщают Twig, когда ему потребуется выполнить определенное действие, например проверить условие или выполнить операции в цикле.

Конструкция `if` проверяет, приводит ли условие к значению `true`. Если да, он выполняет коды шаблона, расположенные до закрывающего тега `{% endif %}`.

Если условие приводит к значению `false`, шаблонизатор пропускает весь код до закрывающего тега `{% endif %}`. Операторы сравнения используются те же, что и в PHP.

```
{% if published == true %}
  <h1>{{ category.name }}<h1>
{% endif %}
```

Когда условие содержит только имя переменной, Twig проверяет, будет ли значение в переменной рассматриваться как `true` после манипуляций типами.

```
{% if published %}
  <h1>{{ category.name }}</h1>
{% endif %}
```

Несколько условий можно объединить с помощью операторов `and` и `or`.

```
{% if time > 6 and time < 12 %}
  <p>Good morning.</p>
{% endif %}
```

Twig также поддерживает структуры `else` и `elseif`, а также тернарный и оператор объединения с `null`.

```
{% if time > 6 and time < 12 %}
  <p>Good morning.</p>
{% elseif time >= 12 < 5 %}
  <p>Good afternoon.</p>
{% else %}
  <p>Welcome.</p>
{% endif %}
```

ИСПОЛЬЗОВАНИЕ ЦИКЛОВ В ШАБЛОНАХ TWIG

В Twig есть цикл `for`, поддерживающий возможность работы с каждым элементом в массиве или с каждым свойством объекта (так же, как цикл PHP `foreach`).

Цикл `for` в Twig похож на цикл `foreach` в PHP и используется для работы с элементами массива, однако синтаксис при этом отличается. Но принцип при этом остается тем же: содержащиеся между тегами цикла инструкции повторяются каждый раз при каждом проходе цикла. Цикл завершается тегом `{% endfor %}`.

Открывающий тег начинается с ключевого слова `for`, затем идет переменная, которой присваивается текущий элемент массива; за ним следует ключевое слово `in` и переменная, содержащая массив или объект, который цикл должен перебирать.

```
{% for article in articles %}
  <h2>{{ article.title }}</h2>
  <p>{{ article.summary }}</p>
{% endfor %}
```

Для выполнения фиксированного количества проходов цикла используется несколько иной синтаксис. Открывающий тег начинается с ключевого слова `for`, а затем идут:

- имя переменной, которая будет использоваться как счетчик, здесь она называется `i` (ее можно использовать далее внутри цикла);
- ключевое слово `in`;
- стартовое значение счетчика, например 1, затем две точки `..` и затем — конечное значение, содержащее количество итераций, выполняемых циклом.

Начальное и конечное значение могут быть заданы переменными. Если начальное значение равно 1, а конечное задано переменной `count` со значением 5, то цикл будет выполняться пять раз. При первом проходе цикла переменная счетчика `i` будет содержать значение 1. В следующий раз она будет содержать значение 2. Так будет продолжаться до тех пор, пока значение не достигнет цифры 5.

Этот тип цикла используется в шаблоне поиска, где результаты отображаются с разбивкой на страницы.

```
{% for i in 1..count %}
  <a href="?page={{ i }}">{{ i }}</a>
{% endfor %}
```

КАК СТРУКТУРИРОВАТЬ ФАЙЛЫ ШАБЛОНОВ

Один **родительский** шаблон должен использоваться для размещения общего для всего сайта HTML-кода. В этом родительском шаблоне обозначаются **блоки**, которые может перезаписать **дочерний** шаблон.

Когда на сайте используется шаблонизатор Twig, в первую очередь необходимо создать **родительский** шаблон, содержащий код, который используется на каждой странице (тот, который находился в файлах заголовка и футера в предыдущих главах).

Внутри шаблонов можно определять **блоки**. В родительском шаблоне блоки — это разделы макета. Другие страницы могут перезаписывать эти разделы для отображения своих данных.

Блоки начинаются с тега, который дает блоку имя: `{% block имя-блока%}`.

Заканчиваются блоки закрывающим тегом: `{% endblock%}`.

На этой странице вы можете ознакомиться с родительским шаблоном под названием `layout.html`. Он содержит код, использующийся на каждой странице сайта, и состоит из трех блоков:

- `title` отображает текст в теге `<title>` страницы. Если дочерний шаблон не переопределяет его новым значением, используется текст внутри этого блока;
- `content` — это блок, где будет отображаться основная область содержимого каждой страницы. Он не содержит никакой информации по умолчанию, поэтому, если дочерний шаблон не содержит блок `content`, на его месте ничего не будет отображаться;
- `footer` содержит футер сайта. Он отображает заявление об авторских правах и текущий год.

Родительский шаблон: `layout.html`

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      {% block title %}
      Creative Folk
      {% endblock %}
    </title>
  </head>
  <body>
    {% block content %}{% endblock %}
    <footer>
      {% block footer %}
      &copy; Creative Folk
      {{ 'now'|date('Y') }}
      {% endblock %}
    </footer>
  </body>
</html>
```

Дочерние шаблоны могут представлять отдельные страницы сайта. Они наследуют код от родительского шаблона и предоставляют данные для перезаписи содержимого в именованных блоках, содержащихся в родительском шаблоне.

Дочерний шаблон: `category.html`

```
{% extends 'layout.html' %}

{% block title %}
{{ category.name }}
{% endblock %}

{% block content %}
<h1>{{ category.name }}</h1>
<p>{{ category.description }}</p>

{{ include('article-summaries.html') }}
{% endblock %}
```

Для каждого из запрашиваемых посетителями типов страниц (главная страница, категория, страницы публикации, участника и поиска) требуется свой **дочерний** шаблон, **расширяющий** родительский шаблон. Тег `extends` указывает имя родительского шаблона для расширения: `{% extends 'родительский-шаблон.html' %}`.

В дочернем шаблоне все, что находится между тегами блока, перезаписывает содержимое соответствующего блока в родительском шаблоне.

На этой странице дочерний шаблон `category.html` расширяет `layout.html` и состоит из двух блоков:

- `title` заменяет все, что находится в блоке заголовка в родительском шаблоне;
- `content` заменяет блок содержимого в родительском шаблоне.

Дочерний шаблон не содержит в себе блок футера, поэтому на странице будет отображаться сообщение об авторских правах из родительского шаблона.

Внутри этого дочернего шаблона блок `content` использует функцию `include()` внутри пары фигурных скобок для подключения другого файла шаблона. Этот шаблон отображает список кратких описаний публикаций:

```
{{ include('article-summaries.html') }}
```

Родительский, дочерний и подключаемый шаблоны могут получать доступ к переменным, переданным в массиве `$data`.

РОДИТЕЛЬСКИЙ ШАБЛОН И ДОЧЕРНИЙ ШАБЛОН КАТЕГОРИИ

Дочерний шаблон `category.html` ниже показывает подробную информацию о любой категории. Он наследует всю разметку от родительского шаблона `layout.html` справа.

1. Тег `extends` указывает на то, что этот дочерний шаблон наследует код из `layout.html`.

2. Блок названия `title` в этом дочернем шаблоне заменяет блок `title` в родительском шаблоне. В нем отображается название категории, затем слова `On Creative Folk`.

3. Блок для мета-тега `description` в дочернем шаблоне заменяет блок `description` в родительском шаблоне. Он показывает описание категории в атрибуте `value` тега `<meta description>`. Этот блок использует фильтр `e()`, чтобы использовать специальное экранирование для атрибутов в HTML-тегах.

4. Блок содержимого `content` в дочернем шаблоне заменяет блок `content` в родительском шаблоне.

5. Название категории отображается в элементе `<h1>`.

6. За этим следует описание категории, показанное внутри элемента `<p>`.

7. Функция `include` подключает шаблон для отображения кратких описаний статей, относящихся к этой категории.

Шаблон `article-summaries.html` перебирает массив `articles` и отображает информацию о каждой публикации.

8. Закрытие блока `content`.

c15/templates/category.html **TWIG**

```
1 {% extends 'layout.html' %}
2 {% block title %}{{ category.name }} on Creative Folk{% endblock %}
3 {% block description %}{{ category.description|e('html_attr') }}{% endblock %}

4 {% block content %}
  <main class="container" id="content">
    <section class="header">
5     <h1>{{ category.name }}</h1>
6     <p>{{ category.description }}</p>
    </section>
    <section class="grid">
7     {{ include('article-summaries.html') }}
    </section>
  </main>
8 {% endblock %}
```

Шаблоны Twig могут использовать любое расширение файла, но `.html` может быть немного удобнее, поскольку любой редактор кода, открывая этот файл, будет знать, что внутри содержится HTML-код.

Соответственно, редактор будет использовать синтаксическую подсветку HTML и сможет автоматически выделять ошибки.

Родительский шаблон содержит разметку, используемую на каждой странице. Он содержит три блока: `title`, `description` и `content`, а также выводит в цикле категории для создания основной навигации.

TWIG c15/templates/layout.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head> ...
    <title>{% block title %}Creative Folk{% endblock %}</title>
    <meta name="description" value="{% block description %}Hire ceatives{% endblock %}">
    <link rel="stylesheet" type="text/css" href="{{ doc_root }}css/styles.css"> ...
  </head>
  <body>
    <header>
      <div class="container">
        <a class="skip-link" href="#content">Skip to content</a>
        <div class="logo"><a href="{{ doc_root }}index.php">
           <img src="{{ doc_root }}img/logo.png" alt="Creative Folk">
        </a></div>
        <nav>
          <button id="toggle-navigation" aria-expanded="false">
            <span class="icon-menu"></span><span class="hidden">Menu</span>
          </button>
          <ul id="menu">
            {% for link in navigation %}
            {% if (link.navigation == 1) %}
              <li><a href="{{ doc_root }}category.php?id={{ link.id }}"
              {% if (section == link.id) %} class="on"{% endif %}>
                {{ link.name }}</a></li>
            {% endif %}
            {% endfor %}
            <li><a href="{{ doc_root }}search.php">
              <span class="icon-search"></span><span class="search-text">Search</span>
            </a></li>
          </ul>
        </nav>
      </div>
    </header>
    {% block content %}{% endblock %}
    <footer>
      <div class="container">
        <a href="{{ doc_root }}contact.php">Contact Us</a>
        <span class="copyright">&copy; Creative Folk {{ 'now'|date('Y') }}</span>
      </div>
    </footer>
    <script src="{{ doc_root }}js/site.js"></script>
  </body>
</html>
```

ШАБЛОН КАРТОЧКИ ПУБЛИКАЦИИ

Шаблон `article-summaries.html` показывает описания нескольких статей. Он используется в шаблонах домашней страницы, категории, пользователя и страницы поиска.

1. Цикл `for` перебирает массив с данными о публикациях, хранящийся в переменной с именем `articles`. Внутри цикла каждая строка этого массива по очереди присваивается переменной `article`.

2. Выводится ссылка на статью.

3. Тег `if` шаблонизатора Twig проверяет, прилагается ли к публикации изображение.

4. Если да, то изображение и его описание отображаются в теге `<img>`.

5. Если нет, то тег `{% else %}` используется для вывода изображения по умолчанию.

6. Если изображения нет, вместо него отображается заглушка.

7. Тег `{% endif %}` отмечает конец оператора `if`.

8. Название статьи отображается в элементе `<h2>`.

9. Вывод описания работы.

10. Вывод ссылки на категорию, в которой находится статья.

11. Вывод ссылки на страницу автора статьи.

12. Цикл заканчивается тегом `{% endfor %}`.

c15/templates/article-summaries.html **TWIG**

```
1  {% for article in articles %}
   <article class="summary">
2    <a href="{{ doc_root }}article.php?id={{ article.id }}">
3      {% if article.image_file %}
4        <img src="{{ doc_root }}uploads/{{ article.image_file }}"
             alt="{{ article.image_alt }}">
5      {% else %}
6        <img src="{{ doc_root }}uploads/blank.png" alt="">
7      {% endif %}
8      <h2>{{ article.title }}</h2>
9      <p>{{ article.summary }}</p>
     </a>
     <p class="credit">
10     Posted in <a href="{{ doc_root }}category.php?id={{ article.category_id }}">
         {{ article.category }}</a>
11     by <a href="{{ doc_root }}member.php?id={{ article.member_id }}">
         {{ article.author }}</a>
     </p>
   </article>
12 {% endfor %}
```

ШАБЛОН СТРАНИЦЫ ПУБЛИКАЦИИ

1. Этот дочерний шаблон отображает одну публикацию. Тег `extends` указывает ему наследовать код шаблона `layout.html`.

2. В блоке `title` отображается заголовок в элементе `<title>`.

3. В блоке `description` выводится краткое описание публикации. Фильтр `e` использует параметр, который экранирует выводимый текст для использования в атрибутах HTML.

4. Блок `content` отображает полную информацию о публикации.

5. Если в ней есть изображение, оно выводится. Если нет, отображается `blank.png`.

6. Заголовок выводится еще раз.

7. Фильтр `date` форматирует дату написания статьи.

8. Описание работы выводится с использованием фильтра `raw`, чтобы предотвратить его автоматическое экранирование, поскольку оно может содержать HTML-разметку (и она уже была сделана безопасной для отображения с помощью пакета HTML Purifier).

9. Вывод ссылки на категорию, к которой относится публикация, а за ней выводится ссылка на страницу автора.

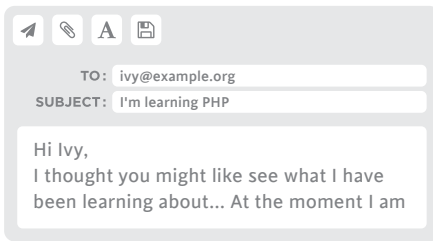
TWIG c15/templates/article.html

```
1 {% extends 'layout.html' %}
2 {% block title %}{{ article.title }}{% endblock %}
3 {% block description %}{{ article.summary|e('html_attr') }}{% endblock %}
4 {% block content %}
<main class="article container" id="content">
  <section class="image">
5   {% if article.image_file %}
      <img src="{{ doc_root }}uploads/{{ article.image_file }}"
        alt="{{ article.image_alt }}">
    {% else %}
      <img src="{{ doc_root }}uploads/blank.png" alt="">
    {% endif %}
  </section>
  <section class="text">
6   <h2>{{ article.title }}</h2>
7   <div class="date">{{ article.created|date('F d, Y') }}</div>
8   <div class="content">{{ article.content|raw }}</div>
    <p class="credit">
9     Posted in <a href="{{ doc_root }}category.php?id={{ article.category_id }}">
      {{ article.category }}</a>
      by <a href="{{ doc_root }}member.php?id={{ article.member_id }}">
      {{ article.author }}</a></p>
  </section>
</main>
{% endblock %}
```

ОТПРАВКА ЭЛЕКТРОННЫХ ПИСЕМ С ПОМОЩЬЮ ПАКЕТА PHPMAILER

Сайты часто отправляют отдельные электронные письма, называемые **транзакционными.** Например, страница сброса пароля может отправлять по электронной почте ссылку для пользователей, чтобы они могли поменять свой пароль, или страница контактов может отправлять сообщение владельцу сайта.

Когда вы отправляете электронное письмо с компьютера или мобильного устройства, вы указываете адрес электронной почты получателя, тему и текст сообщения. Затем почтовая программа отправляет электронное письмо на сервер, называемый SMTP-сервером. Он доставляет электронное письмо получателю.



Когда для работы с электронной почтой используется не браузерный клиент, а почтовая программа, установленная на компьютере пользователя, в ней необходимо настроить подключение к почтовому серверу. Для отправки почты используется SMTP-сервер. Как правило, для подключения к нему требуются следующие сведения:

- **имя хоста**, идентифицирующее SMTP-сервер точно так же, как имя домена идентифицирует веб-сервер;
- **номер порта**, позволяющий различным программам на одном компьютере использовать одно и то же интернет-подключение (см. http://notes.re/php/ports);
- **имя пользователя и пароль** для входа в учетную запись;
- **параметры безопасности** для указания способа безопасной отправки имени пользователя и пароля.

То же самое требуется и при отправке электронных писем с помощью PHP-кода.

Отправка электронного письма происходит в два этапа.

1. Подключение к SMTP-серверу, который может отправлять электронную почту.

2. Создание электронного письма и передача его на SMTP-сервер.

Код для выполнения этих двух задач достаточно сложный. Поэтому, вместо того чтобы писать код для создания и отправки электронных писем с нуля, на нашем учебном сайте мы будем использовать пакет под названием **PHPMailer**. Он используется во многих популярных проектах с открытым исходным кодом, включая WordPress, Joomla и Drupal. Пакет PHPMailer находится в папке `vendor` в загружаемом коде для этой главы и указан как обязательный пакет в файле `composer.json`. Его название на сайте Packagist `phpmailer\phpmailer`.

Хотя сайт может использовать собственный SMTP-сервер, веб-сайты обычно используют специализированный сервис, предоставляющий SMTP-сервер для отправки электронных писем, поскольку:

- отправка слишком большого количества электронных писем с вашего веб-сервера может привести к тому, что службы электронной почты внесут ваше имя домена в черный список и будут рассматривать ваши письма как спам;
- у этих сервисов более высокие показатели успешности доставки.

Список служб, отправляющих транзакционные электронные письма, см. на странице http://notes.re/transactional-emails.

Вам нужно будет зарегистрироваться на одном из этих сервисов, чтобы протестировать отправляющий электронные письма код.

НАСТРОЙКИ ДЛЯ ПОДКЛЮЧЕНИЯ К SMTP-СЕРВЕРУ

Поскольку настройки подключения к SMTP-серверу различны для каждого сайта, они относятся к данным конфигурации, и поэтому хранятся в файле `config.php`.

Каждому сайту, использующему код учебной CMS, требуются разные настройки для подключения к своему SMTP-серверу (точно так же, как каждый сайт использует разные данные для подключения к своей базе данных). Сведения, используемые для подключения к SMTP-серверу, хранятся в виде ассоциативного массива в файле `config.php` вместе с адресом электронной почты владельца сайта.

PHP c15/config/config.php

```
1 $email_config = [
2     'server' => 'smtp.Ваш-Сервер.com',
3     'port' => 'Ваш-Номер-Порта',
4     'username' => 'Ваше-Имя-Пользователя',
      'password' => 'Ваш-Пароль',
5     'security' => 'tls',
6     'admin_email => 'Ваш-email',
7     'debug'       => (DEV) ? 2 : 0,
];
```

Сначала необходимо настроить параметры подключения к SMTP-серверу.

1. `$email_config` — это переменная, содержащая массив с настройками для отправки электронных писем.

2. Параметр `server` содержит имя хоста SMTP-сервера.

3. Параметр `port` содержит номер порта, используемого SMTP-сервером.

4. Параметры `username` и `password` содержат данные для входа в учетную запись SMTP-сервера.

5. Параметр `security` содержит метод, используемый для безопасной отправки данных. Обычно его значение `tls`, что означает Transport Layer Security.

Затем добавьте два других необходимых значения.

6. `admin_email` — это адрес электронной почты владельца сайта, на который отправляются сообщения контактной формы, а также адрес в графе «От» (From) для других электронных писем, отправляемых сайтом в следующей главе.

7. `debug` включает и выключает отладочные сообщения. Значение в константе `DEV` определяет поведение кода:

- значение 2 используется во время разработки сайта, чтобы отображать электронные письма, отправленные веб-сервером, и ответы SMTP-сервера;
- значение 0 устанавливается на действующем сайте, чтобы отключить отладочные сообщения (поскольку в сообщениях отображаются данные об учетной записи SMTP).

СОЗДАНИЕ И ОТПРАВКА ЭЛЕКТРОННОГО ПИСЬМА

Сначала создайте объект класса PHPMailer и укажите ему, как подключиться к SMTP-серверу. Затем создайте и отправьте электронное письмо.

Объект PHPMailer создается точно так же, как и любой другой объект. Его пространство имен PHPMailer\PHPMailer. Это пространство имен должно использоваться при создании объекта.

В конструктор передается только один аргумент, логическое значение true. Оно указывает PHPMailer выбрасывать исключение при возникновении проблем.

```
$phpmailer = new \PHPMailer\PHPMailer\PHPMailer(true);
```

ПЕРЕМЕННАЯ — ПРОСТРАНСТВО ИМЕН — КЛАСС — ВЫБРАСЫВАТЬ ИСКЛЮЧЕНИЯ

После создания объекта PHPMailer мы будем использовать два метода и восемь свойств для его настройки, чтобы сконфигурировать отправку электронных писем.

Они похожи на настройки, используемые почтовой программой для новой учетной записи электронной почты, и будут неизменными каждый раз, когда сайт отправляет транзакционное письмо.

СВОЙСТВО / МЕТОД	ОПИСАНИЕ
isSMTP()	Метод, указывающий, что будет использоваться SMTP-сервер
Host	Свойство задает адреса хоста SMTP-сервера
SMTPAuth	Свойство, которое включает использование SMTP-авторизации. Установлено значение true, поскольку для входа на SMTP-сервер потребуются имя пользователя и пароль
Username	Свойство задает имя пользователя учетной записи SMTP
Password	Свойство задает пароль учетной записи SMTP
Port	Свойство задает номер порта, используемого SMTP-сервером
SMTPSecure	Свойство задает тип шифрования. Обычно устанавливается значение tls
SMTPDebug	Свойство, указывающее библиотеке PHPMailer, показывать отладочную информацию или нет
isHTML()	Метод, который указывает PHPMailer, что электронное письмо может содержать HTML
CharSet	Свойство, задающее кодировку символов, используемую в электронном письме. Если она установлена неправильно, текст может некорректно отображаться в почтовой программе получателя

После создания объекта PHPMailer и настройки параметров можно создавать и отправлять электронное письмо. Процесс создания электронного письма и передачи его на SMTP-сервер включает в себя вызов трех методов и настройку трех свойств объекта PHPMailer.

Свойства и методы в первых пяти строках приведенной ниже таблицы эквивалентны написанию нового электронного письма в почтовой программе. Используемые ими значения могут меняться каждый раз, когда сайт отправляет электронное письмо. Последний метод эквивалентен нажатию кнопки «Отправить» для отправки электронного письма.

СВОЙСТВО / МЕТОД	ОПИСАНИЕ
setFrom()	Метод задает адрес отправителя
addAddress()	Метод задает адрес получателя (можно вызвать несколько раз, чтобы добавить другие адреса)
Subject	Свойство, которое задает заголовок электронного письма
Body	Свойство, которое задает текст электронного письма, в котором содержится HTML
AltBody	Свойство, которое задает текстовую версию электронного письма (без HTML-разметки)
send()	Метод для подключения к SMTP-серверу и передачи ему электронного письма

Создание объекта PHPMailer, указание ему, как подключиться к SMTP-серверу, а также создание и отправка электронного письма занимают не менее 18 строк кода.

Отправка транзакционных электронных писем часто требуется на нескольких страницах. Вместо того чтобы повторять этот код на каждой из них, будет удобнее сохранить его в новом пользовательском классе под названием Email. Класс представлен на следующей странице.

На любой странице, где требуется отправить электронное письмо, это можно будет сделать с помощью всего лишь двух строк, приведенных ниже.

1. Первая строка создает объект с использованием пользовательского класса Email и сохраняет его в переменной с именем $email. Данные конфигурации, необходимые объекту, передаются в метод конструктора.

2. Вторая строка вызывает метод sendEmail() объекта Email для создания и отправки электронного письма.

```
① $email = new Email($email_config);
② $email->sendEmail($from, $to, $subject, $message);
```

Новый пользовательский класс Email показан на следующей странице. Он содержит два описанных ниже метода.

За ним следует пример того, как этот класс используется на следующих страницах.

МЕТОД	ОПИСАНИЕ
__construct(*$email_config*)	Создает объект PHPMailer и сохраняет его в свойстве $phpmailer. Настраивает подключение PHPMailer к SMTP-серверу. Эти инструкции пишутся в методе __construct(), поскольку они всегда будут одни и те же при каждой отправке электронного письма
sendEmail(*$from, $to, $subject, $message*)	Создает электронное письмо и передает его на SMTP-сервер. Этот метод вызывается для отправки электронного письма. При каждом вызове его аргументам могут быть присвоены разные значения

КЛАСС ДЛЯ СОЗДАНИЯ И ОТПРАВКИ ЭЛЕКТРОННЫХ ПИСЕМ

Класс `Email` используется, когда на странице необходимо отправить электронное письмо. Он содержит код, который создает объект `PHPMailer`, а также код, который создает электронное письмо и отправляет его на SMTP-сервер.

1. Класс помещается в пространство имен `PhpBook\Email`.

2. Свойство `$phpmailer` содержит объект `PHPMailer`. Оно объявлено как защищенное, так что его может использовать только в этом классе.

3. Метод `__construct()` принимает только один параметр — данные конфигурации, хранящиеся в переменной `$email_config`. Код внутри этого метода выполняется каждый раз, когда создается экземпляр класса. При этом:

- создается объект `PHPMailer`;
- настраивается конфигурация подключения к SMTP-серверу;
- устанавливается кодировка символов и тип электронной почты.

4. Создание объекта `PHPMailer` и сохранение его в свойстве `$phpmailer` этого объекта `Email`. Аргумент `true` указывает объекту `PHPMailer` выбрасывать исключение, если он сталкивается с проблемой при создании или отправке электронного письма.

5. Метод `isSMTP()` объекта `PHPMailer` задает отправку через SMTP-сервер.

6. Свойству `SMTPAuth` присваивается значение `true`, чтобы для входа на SMTP-сервер использовались имя пользователя и пароль.

7. Данные для подключения к SMTP-серверу задаются с использованием значений из массива `$email_config`, который был передан в конструктор при создании объекта.

8. Задается кодировка символов UTF-8, устанавливается формат HTML для писем.

9. Метод `sendEmail()` создает и отправляет отдельное электронное письмо. У него есть четыре параметра, представляющие данные, которые могут меняться каждый раз, когда объект используется для отправки электронного письма:

- `$from` указывает адрес отправителя;
- `$to` содержит адрес электронной почты получателя;
- `$subject` содержит тему электронного письма;
- `$message` содержит отправляемое сообщение.

Метод возвращает значение `true`, если сообщение было создано и отправлено.

10. Метод `setFrom()` задает адрес отправителя.

11. Метод `addAddress()` задает адрес электронной почты получателя.

12. Свойство `Subject` задает тему электронного письма.

13. Свойство `Body` задает текст электронного письма. Он состоит из нескольких базовых HTML-тегов в начале, после которых идет значение, указанное в параметре `$message`, а затем закрывающих HTML-тегов.

14. Свойство `AltBody` задает текстовую версию электронного письма. Он использует функцию PHP `strip_tags()` для удаления разметки из сообщения (см. примечание справа).

15. Метод `send()` отправляет электронное письмо на SMTP-сервер.

16. Метод возвращает значение `true`, указывающее, что электронное письмо было создано и передано на SMTP-сервер. Он выбросил бы исключение, если бы операция не прошла.

Далее вы увидите, как использовать этот класс для отправки электронного письма. После того как на странице был создан объект класса `Email` и отправлено электронное письмо, можно отправлять электронные письма и дальше, снова вызывая метод `sendEmail()`.

PHP c15/src/classes/Email/Email.php

```php
<?php
namespace PhpBook\Email;                                    // Объявление пространства имен

class Email {

    protected $phpmailer;                                   // Объект PHPMailer

    public function __construct($email_config)
    {
        $this->phpmailer = new \PHPMailer\PHPMailer\PHPMailer(true); // Создание объекта
        $this->phpmailer->isSMTP();                         // Использование SMTP
        $this->phpmailer->SMTPAuth   = true;                // Использование авторизации
        $this->phpmailer->Host       = $email_config['server'];     // Адрес сервера
        $this->phpmailer->SMTPSecure = $email_config['security'];   // Тип безопасности
        $this->phpmailer->Port       = $email_config['port'];       // Порт
        $this->phpmailer->Username   = $email_config['username'];   // Имя пользователя
        $this->phpmailer->Password   = $email_config['password'];   // Пароль
        $this->phpmailer->SMTPDebug  = $email_config['debug'];      // Метод отладки
        $this->phpmailer->CharSet    = 'UTF-8';             // Кодировка символов
        $this->phpmailer->isHTML(true);                     // Установка формата HTML
    }

    public function sendEmail($from, $to, $subject, $message): bool
    {
        $this->phpmailer->setFrom($from);           // Адрес электронной почты отправителя
        $this->phpmailer->addAddress($to);          // Адрес электронной почты получателя
        $this->phpmailer->Subject = $subject;       // Тема письма
        $this->phpmailer->Body    = '<!DOCTYPE html><html lang="en-us"><body>'
          . $message .'</body></html>';             // Текст письма
        $this->phpmailer->AltBody = strip_tags($message);  // Текстовый вариант
        $this->phpmailer->send();                           // Отправка письма
        return true;                                        // Возврат значения true
    }
}
```

Когда отправляется электронное письмо в формате HTML, вместе с HTML-версией отправляется и чисто текстовая версия. Ее важно создавать потому, что спам-фильтры больше доверяют таким письмам, и к тому же некоторые люди используют текстовые программы для работы с электронной почтой.

Встроенная в PHP функция `strip_tags()` предназначена для удаления тегов из разметки. Ее параметр — это строка, содержащая разметку. Функция возвращает строку с удаленными тегами. Вы также можете использовать пакет HTML Purifier для удаления разметки из своих электронных писем.

ИСПОЛЬЗОВАНИЕ КЛАССА Email

Новая страница contact.php (см. правую страницу) содержит форму для отправки электронного письма владельцам сайта. Чтобы отправить электронное письмо, страница контактов создает объект, используя класс Email. Затем она вызывает его метод sendEmail() с четырьмя параметрами: адрес электронной почты отправителя, адрес, на который оно должно быть отправлено, строка темы и сообщение.

1. Команда use импортирует код из класса Validate в текущее пространство имен.

2. Если форма была отправлена, адрес электронной почты и сообщение сохраняются в переменных.

3. Полученные значения проверяются с помощью класса Validate. Массив с сообщениями об ошибках объединяется в строку, которая сохраняется в переменной с именем $invalid.

4. Если в данных были ошибки, то переменная $errors содержит сообщения о них.

5. В противном случае код переходит к отправке электронного письма.

6. Тема письма записывается в переменную $subject.

7. Объект Email создается с помощью класса Email.

8. Для создания и отправки электронного письма вызывается метод sendEmail() объекта Email. Он содержит четыре параметра:

- адрес, с которого отправляется электронное письмо;
- адрес для отправки электронного письма;
- строка темы;
- сообщение.

9. Если при отправке не было выброшено исключение, в переменную $success будет записано сообщение для пользователя об успешной отправке[75].

10. Массив $data содержит данные для страницы, а метод Twig render() генерирует HTML-код.

c15/templates/contact.html **TWIG**

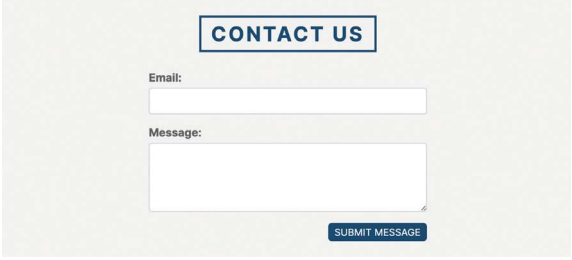
```
{% extends 'layout.html' %}
{% block content %}
<main class="container" id="content">
  <section class="heading"><h1>Contact Us</h1></section>
  <form method="post" action="contact.php" class="form-contact">
    {% if errors.warning %}<div class="alert-danger">{{ errors.warning }}</div>{% endif %}
    {% if success %}<div class="alert-success">{{ success }}</div>{% endif %}
    <label for="email">Email: </label>
    <input type="text" name="email" id="email" value="{{ from }}" class="form-control">
    <span class="errors">{{ errors.email }}</span><br>
    <label for="message">Message: </label><br>
    <textarea id="message" name="message" class="form-control">{{ message }}</textarea>
    <span class="errors">{{ errors.message }}</span><br>
    <input type="submit" value="Submit Message" class="btn">
  </form>
</main>
{% endblock %}
```

PHP c15/public/contact.php

```php
<?php
declare(strict_types = 1);                          // Строгая типизация
use PhpBook\Validate\Validate;                      // Импорт класса валидации
include '../src/bootstrap.php';                     // Файл настроек
$from    = '';                                      // Инициализация: from (от)
$message = '';                                      // Текст письма
$errors  = [];                                      // Массив для ошибок
$success = '';                                      // Сообщение об успешном завершении

if ($_SERVER['REQUEST_METHOD'] == 'POST') {         // Если форма отправлена
    $from              = $_POST['email'];           // Адрес email
    $message           = $_POST['message'];         // Сообщение
    $errors['email']   = Validate::IsEmail($from)            ? '' : 'Email not valid';
    $errors['message'] = Validate::IsText($message, 1, 1000) ? '' : 'Please enter a
        message up to 1000 characters';
    $invalid = implode($errors);                    // "Схлопывание" массива с ошибками
    if ($invalid) {                                 // Если есть ошибки
        $errors['warning'] = 'Please correct the errors';   // Сообщение об этом
    } else {                                        // В противном случае попытка отправки
        $subject = "Contact form message from " . $from;    // Тема письма
        $email   = new \PhpBook\Email\Email($email_config); // Создание объекта email
        $email->sendEmail($email_config['admin_email'], $email_config['admin_email'],
            $subject, $message);                    // Отправка
        $success = 'Your message has been sent';    // Сообщение об успехе
    }
}
$data['navigation'] = $cms->getCategory()->getAll(); // Категории для навигации
// Следующие переменные будут иметь значения, только если форма была отправлена
$data['from']    = $from;                           // Адрес почты отправителя
$data['message'] = $message;                        // Текст письма
$data['errors']  = $errors;                         // Сообщения об ошибках
$data['success'] = $success;                        // Сообщение об успешном завершении
echo $twig->render('contact.html', $data);          // Рендеринг шаблона
```

РЕЗУЛЬТАТ



Примечание. Если константа DEV в файле config.php содержит значение true, PHPMailer создает длинный набор отладочных сообщений, которые отображаются на странице перед заголовком и формой.

Когда константе DEV присвоено значение false, они скрыты.

ЗАКЛЮЧЕНИЕ
ПРОСТРАНСТВА ИМЕН И БИБЛИОТЕКИ

- Пространства имен гарантируют, что, если два или более классов, функций или констант используют одно и то же имя, интерпретатор PHP сможет различать их.
- Библиотеки и пакеты позволяют вам использовать код, написанный другими программистами, для выполнения определенных задач.
- Инструмент Composer помогает управлять пакетами, используемыми в коде.
- На сайте Packagist.org публикуются пакеты, которые может использовать Composer.
- Composer создает автозагрузчик, подключающий файлы классов для пакетов, когда код на странице их использует.
- Библиотеку HTML Purifier можно использовать для удаления разметки, способной вызвать атаку XSS.
- Twig — это шаблонизатор, отделяющий PHP-код, занимающийся получением и обработкой данных, от шаблонов, которые управляют отображением этих данных.
- PHPMailer — это пакет, используемый для создания электронных писем с применением PHP и отправки их на SMTP-сервер.

16

РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЕЙ

В этой главе показано, как посетители могут зарегистрироваться на сайте. После этого они смогут входить в систему, чтобы просматривать страницы, доступные только зарегистрированным пользователям, и видеть персонализированную информацию.

Как правило, когда пользователь регистрируется на сайте, ему необходимо ввести:

- **идентификатор**. Используется для идентификации. Например, это может быть адрес электронной почты или имя пользователя. Каждому участнику сайта нужен свой уникальный идентификатор;
- **пароль**. Используется как подтверждение, что пользователь именно тот, за кого он себя выдает. Пользователь — единственный, кто должен знать свой пароль.

Эти данные сохраняются в базе данных. На нашем учебном веб-сайте, когда участник вошел в систему, он может:

- просматривать страницы, доступ к которым имеют только пользователи;
- создавать и редактировать свой профиль;
- загружайте собственные работы.

Эта глава разделена на три раздела.

- **Регистрация на сайте**. Как получить информацию, необходимую для идентификации отдельного пользователя сайта, и сохранить ее в базе данных.
- **Вход в систему и персонализация страниц**. Как разрешить пользователям входить в систему, как создавать страницы, адаптированные под каждого пользователя, и как создавать страницы, доступные только зарегистрированным пользователям.
- **Обновление базы данных без входа пользователя в систему**. Как разрешить пользователям менять информацию в БД без предварительного входа в систему. Например, когда им нужно изменить пароль. Это включает в себя рассмотрение нового набора требований к безопасности.


Safari File Edit View History Bookmarks Develop Window Help
Tue 10 Aug 11:52
localhost
Log in / Register
CREATIVE FOLK
Print / Digital / Illustration / Photography
EVE HOWARD
MEMBER SINCE: MARCH 13, 2019
New Forest
Website design for artist retreat
POSTED IN DIGITAL BY EVE HOWARD
Quiet Invite
Invitation design for documentary
POSTED IN PRINT BY EVE HOWARD
Quiet
Book cover design
POSTED IN PRINT BY EVE HOWARD
Contact us © Creative Folk 2021

ИЗМЕНЕНИЯ В БАЗЕ ДАННЫХ

Для последних двух глав в базе данных учебного веб-сайта понадобятся три дополнительные таблицы и новые столбцы в нескольких существующих таблицах.

Следуя инструкциям, используйте phpMyAdmin:

- создайте новую базу данных с именем `phpbook-2`;
- импортируйте файл `phpbook-2.sql` из загружаемого кода для этой главы, чтобы создать таблицы в новой базе данных и добавить в них данные.

После того как вы создали эту новую базу данных, используйте phpMyAdmin, чтобы ознакомиться с изменениями. Во-первых, появились три новые таблицы. Таблица токенов представлена ближе к концу этой главы. Таблицы `comment` и `likes` будут рассмотрены в следующей главе.

Во-вторых, появились новые столбцы в таблицах `article`, `category` и `member`. В этой главе вводится новый столбец роли в таблице `member`. Новый столбец `seo_title` таблицы `article` и столбец `seo_name` таблицы `category` будут рассмотрены в следующей главе.

После того как вы ознакомились с изменениями базы данных в `phpMyAdmin`, откройте файл `config.php` в коде для этой главы и добавьте настройки для подключения к новой базе данных. Единственным отличающимся от предыдущих глав параметром будет новое имя базы данных (`phpbook-2`).

После того как вы откроете пример кода для этой главы в браузере, используйте ссылку регистрации, которая находится в правом верхнем углу каждой страницы, чтобы создать собственную учетную запись. После регистрации войдите в систему и ознакомьтесь с текущей версией сайта. Вы должны увидеть, что:

- есть и другие пользователи (и их работы);
- вы можете получить доступ к страницам администрирования только в том случае, если вы вошли в систему (и у вас есть к ним доступ).

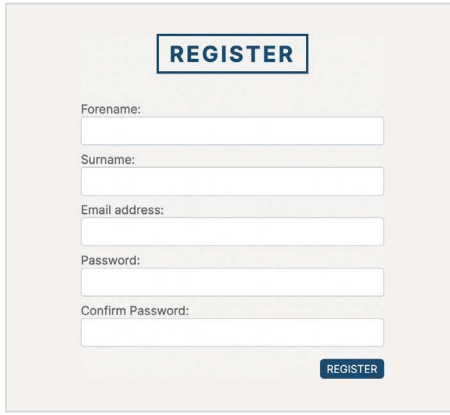
Изучив эту главу до конца, вы узнаете, какой код позволяет пользователям регистрироваться, входить в систему, загружать свои работы и запрашивать ссылку для сброса пароля.

Примечание. Загружаемый код для этой главы содержит несколько не описанных в тексте главы файлов, поскольку задачи, которые они выполняют, уже были рассмотрены в других главах.

Например, страница, позволяющая пользователям загружать работы, похожа на страницу, позволяющую администраторам создавать публикации. Основные различия заключаются в том, что пользователь обязательно должен добавить изображение, и публикация сразу же становится доступной для просмотра. Аналогичным образом страница, на которой пользователь может отредактировать личную информацию, работает по тому же принципу, как и страница, используемая администраторами для редактирования категорий.

РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЕЙ

Посетители должны заполнить регистрационную форму, чтобы стать пользователем сайта. Их данные хранятся в таблице `member` базы данных.



Страница `register.php` использует форму, показанную выше, чтобы позволить посетителям зарегистрироваться на сайте.
Когда форма отправляется, данные сначала проверяются, а потом новый метод `create()` класса `Member` добавляет данные в таблицу `member`, используя показанные в главе 14 приемы.

Регистрация вводит две новые концепции:

- **роли** (определяют, на выполнение каких задач у пользователя есть допуск);
- **хеш пароля** (веб-сайты хранят его вместо фактического пароля, вводимого пользователем при регистрации).

РОЛИ

Веб-сайты часто позволяют разным пользователям просматривать разные страницы и выполнять разные задачи. **Роли** используются для определения, какие задачи может выполнять участник. Образец сайта различает:

- **Посетитель** (`visitor`): кто-то, кто не вошел в систему. Они могут только просматривать работы на сайте.
- **Пользователь** (`member`): кто-то, кто зарегистрировался и вошел в систему. Они могут редактировать свой профиль, загружать новые работы и редактировать существующие.
- **Заблокирован** (`suspended`): зарегистрированный пользователь, чей допуск к использованию сайта был приостановлен. Таким пользователям будет запрещено входить в систему.
- **Администратор** (`admin`): владелец или сотрудник сайта. Они могут просматривать страницы администрирования, создавать категории, удалять работы и обновлять роли пользователей.

В таблице `member` в новом столбце с именем `role` хранится роль каждого участника. Его значением будет `member`, `suspended` или `admin` (он не содержит `visitor`, потому что это статус людей, которые не вошли в систему).

Примечание. При регистрации на учебном сайте автоматически устанавливается роль `admin`, чтобы позволить вам получить доступ к страницам администрирования без изменения роли в базе данных вручную. На действующем сайте для новых пользователей должна по умолчанию ставиться роль `member`, а администратор может ее затем изменить на другую.

ХЕШИ ПАРОЛЕЙ

По соображениям безопасности веб-сайты не должны хранить пароли пользователей. Вместо этого они хранят односторонне зашифрованную версию пароля, называемую **хешем**. Невозможно расшифровать хеш обратно в исходный текст[76].

Пользователь — единственный человек, который должен знать свой пароль. Сайт может проверять пароли пользователей, но сами пароли не должны храниться в базе данных.

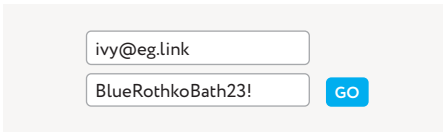
Когда пользователь регистрируются, специальный алгоритм превращает его пароль в **хеш**, который обычно выводится в виде случайного набора буквенно-цифровых символов. В базе данных хранится хеш вместо пароля.

PHP содержит встроенные функции для хеширования паролей. Каждый раз, когда функция используется для преобразования какой-либо строки в хеш, она выдает один и тот же набор символов.

Когда зарегистрированный пользователь входит на сайт, введенный им пароль снова проходит через алгоритм хеширования. Если полученное значение совпадает с хешем, хранящимся в базе данных, это значит, что пользователь ввел правильный пароль.

ВВОД: ПАРОЛЬ

Когда пользователь вводит пароль, функция хеширования конвертирует его в хеш:



РЕЗУЛЬТАТ: ХЕШ

База данных сохранит хеш (она не хранит реальный пароль):

email	пароль
ivy@eg.link	$2y$10$XTeGk6Z7XG1Gs26.MVvCIOANsdgFjZOYEMDWYlmlca4cOKyMwjufi

Хеш не может быть преобразован обратно в исходный пароль, и поэтому, даже если кто-то получит доступ к базе данных, он не сможет получить пароли пользователей. Это защищает как ваш собственный сайт, так и пользователей, которые, возможно, использовали тот же пароль на других сайтах. Независимо от того, сколько символов содержит пароль, его хеш будет состоять из одного и того же количества символов, поэтому хеш не дает подсказки относительно длины пароля. Используемый в PHP на данный момент алгоритм хеширования по умолчанию возвращает строку определенного формата, длина которой всегда равна 60 символам, из которых под собственно хеш отводится 31.

Для краткости эту строку тоже называют хешем, хотя это и не совсем корректно.
Чтобы сделать хеш еще более безопасным, функция хеширования паролей в PHP перед хешированием добавляет к паролю случайный набор символов, называемый **солью** (salt), а затем сохраняет его вместе с хешем. Когда пользователь входит в систему, PHP:

- выделяет из сохраненной строки соль и хеш пароля;
- используя эту соль, создает хеш введенного пароля;
- сравнивает сохраненный хеш с полученным.

Если они совпадают, пользователь ввел правильный пароль.

СОЗДАНИЕ И ПРОВЕРКА ХЕШИРОВАННЫХ ПАРОЛЕЙ

Функция PHP `password_hash()` создает хеш из пароля. Функция PHP `password_verify()` проверяет, что хеш от введенного пользователем пароля совпадает с сохраненным.

Встроенная в PHP функция `password_hash()` принимает пароль и возвращает хеш пароля. Эта функция принимает три параметра:

- хешируемый пароль;
- имя используемого алгоритма хеширования;
- необязательный массив настроек для этого алгоритма (на учебном сайте они не используются).

На сайте php.net можно ознакомиться с набором констант, которые можно использовать для указания алгоритма хеширования, используемого в коде, см. http://notes.re/php/pwd_hash/.

На учебном сайте используется константа `PASSWORD_DEFAULT`. Она указывает на алгоритм хеширования PHP по умолчанию. На момент написания статьи это был алгоритм `bcrypt`, но он может быть изменен по мере создания более сильных алгоритмов.

```
password_hash($password, $algorithm[, $options]);
```

ПАРОЛЬ — АЛГОРИТМ — ПАРАМЕТРЫ

Встроенная в PHP функция `password_verify()`принимает пароль, введенный пользователем, и создает для него хеш, а затем сравнивает полученное значение с сохраненным хешем. Если значения совпадают, значит, пользователь ввел правильный пароль. Функция `password_verify()` принимает два параметра:

- только что введенный пользователем пароль;
- сохраненный для этого пользователя хеш.

Вам не нужно указывать алгоритм или соль, использованные при создании хеша, поскольку они уже содержатся в строке, которую возвращает функция `password_hash()` и которая была сохранена в базе данных.

Функция возвращает значение:

- `true`, если хеши совпадают;
- `false`, если значения отличаются.

```
password_verify($password, $hash);
```

ВВЕДЕННЫЙ ПОЛЬЗОВАТЕЛЕМ ПАРОЛЬ — СОХРАНЕННЫЙ В БД ХЕШ

РЕГИСТРАЦИЯ НОВЫХ ПОЛЬЗОВАТЕЛЕЙ (ЧАСТЬ 1)

Страницы регистрации работают по тому же принципу, что и страницы администрирования, добавляющие данные в БД. При отправке формы данные проходят валидацию. Если в них нет ошибок, новый метод класса `Member` добавит пользователя в базу данных.

Файл `register.php` позволяет посетителям зарегистрироваться на сайте. После того как посетитель заполнит форму, если данные введены без ошибок, то метод `create()` объекта `Member` добавит их в базу данных. Если же данные не прошли валидацию, то будет выведена форма с сообщениями об ошибках.

1. Включение строгой типизации, и пространство имен класса `Validate` импортируется, чтобы этот класс можно было использовать на странице, не вводя полное имя.
2. Подключение файла `bootstrap.php`.
3. Переменные `$member` и `$errors` инициализируются как пустые массивы, следовательно, их можно будет добавить в массив `$data`, используемый шаблоном Twig (шаг 12), даже если в них не было добавлено никаких элементов на шагах 4–11.
4. Конструкция `if` проверяет, была ли форма отправлена.
5. Если отправлена, то данные собираются в массив `$member`. Значение поля подтверждения пароля сохраняется в отдельной переменной, поскольку его значение не будет добавлено в базу данных (оно используется только для проверки, что пользователь дважды ввел один и тот же пароль).
6. Данные проходят валидацию. Если какие-либо из них неверны, сообщения об ошибках пишутся в массив `$errors`. В классе `Validate` для этой главы есть два новых метода. Они проверяют, что адрес электронной почты соответствует формату и что пароль соответствует минимальным требованиям.
7. Все значения в массиве `$errors` объединяются в одну строку, которая записывается в переменную `$invalid`.
8. Конструкция `if` проверяет, не содержит ли `$invalid` какой-либо текст. Если нет, то данные прошли проверку. Если содержит, то в массиве `$errors` будет по крайней мере одно сообщение об ошибке.
9. Если ошибок нет, вызывается метод `create()` объекта `Member` для добавления пользователя в базу данных. Метод `create()` возвращает значение `true`, если участник добавлен успешно, или значение `false`, если адрес электронной почты уже используется (а если возникает какая-либо другая проблема, то выбрасывается исключение и выполнение кода в функции останавливается). Возвращаемое значение сохраняется в переменной `$result`.
10. Конструкция `if` проверяет, содержит ли `$result` значение `false`. Если да, то в элемент массива `$errors` с ключом `email` будет записано сообщение, информирующее пользователя, что адрес электронной почты уже используется.
11. Противоположный результат означает, что информация о пользователе была добавлена в базу данных в шаге 9, и поэтому здесь он перенаправляется на страницу входа в систему, а в строке запроса отправляется сообщение об успешной регистрации.
12. Данные, которые должны отображаться в шаблоне Twig, сохраняются в массиве с именем `$data`.
13. Метод `render()` объекта Twig использует шаблон `register.html` для создания HTML-кода, который затем выводится в браузер.

PHP c16/public/register.php

```php
<?php
declare(strict_types = 1);                  // Строгая типизация
use PhpBook\Validate\Validate;              // Импорт класса валидации

include '../src/bootstrap.php';             // Файл настроек
$member = [ ];                              // Инициализация массива данных пользователя
$errors = [ ];                              // Инициализация массива ошибок

if ($_SERVER['REQUEST_METHOD'] == 'POST') {          // Если форма была отправлена
    // Get form data
    $member['forename'] = $_POST['forename'];         // Получение имени
    $member['surname']  = $_POST['surname'];          // Получение фамилии
    $member['email']    = $_POST['email'];            // Получение email пользователя
    $member['password'] = $_POST['password'];         // Получение пароля
    $confirm            = $_POST['confirm'];          // Получение подтверждения пароля

    // Валидация данных формы
    $errors['forename'] = Validate::isText($member['forename'], 1, 254)
        ? '' : 'Forename must be 1-254 characters';
    $errors['surname']  = Validate::isText($member['surname'], 1, 254)
        ? '' : 'Surname must be 1-254 characters';
    $errors['email']    = Validate::isEmail($member['email'])
        ? '' : 'Please enter a valid email';
    $errors['password'] = Validate::isPassword($member['password'])
        ? '' : 'Passwords must be at least 8 characters and have:<br>
                A lowercase letter<br>An uppercase letter<br>A number
                <br>And a special character';
    $errors['confirm']  = ($member['password'] = $confirm)
        ? '' : 'Passwords do not match';
    $invalid            = implode($errors);          // Объединение сообщений об ошибках

    if (!$invalid) {                                 // Если ошибок нет
        $result = $cms->getMember()->create($member); // Сохранение пользователя в БД
        if ($result === false) {                     // Если ошибка
            $errors['email'] = 'Email address already used'; // Сообщение об ошибке
        } else {     // В противном случае отправить на страницу входа в систему
            redirect('login.php', ['success' => 'Thanks for joining! Please log in.']);
        }
    }
}

$data['navigation'] = $cms->getCategory()->getAll();         // Категории для навигации
$data['member']     = $member;                               // Данные пользователя
$data['errors']     = $errors;                               // Сообщения об ошибках

echo $twig->render('register.html', $data);                  // Рендеринг шаблона
```

РЕГИСТРАЦИЯ НОВЫХ ПОЛЬЗОВАТЕЛЕЙ (ЧАСТЬ 2)

Справа, в верхнем блоке кода показан шаблон Twig, используемый для отображения регистрационной формы.

В загружаемом коде код этого шаблона содержит еще несколько элементов и атрибутов, используемых для управления отображением. Часть из них здесь не показаны, чтобы вы могли сосредоточиться на основном функционале, а также чтобы код поместился на странице.

1. Шаблон Twig расширяет шаблон `layout.html` и определяет новое содержимое для блоков `title` и `description`. Это содержимое перезаписывает текст, выводящийся по умолчанию в тегах `<title>`и `<meta>` в шаблоне `layout.html`.
2. Блок `content` содержит регистрационную форму.
3. Форма отправляется на ту же страницу PHP.
4. Если данные формы были неверными, посетителю будет показано предупреждающее сообщение.
5. Отображение полей ввода.

В коде формы используется два массива, в которых содержатся данные, если форма была отправлена, но в данных были выявлены ошибки:

- `member` — массив введенных пользователем данных. Он используется для заполнения значений в полях ввода, чтобы посетителю не нужно было повторно вводить все свои данные;
- `errors` — это массив, содержащий сообщения об ошибках для каждого не прошедшего валидацию элемента данных. Эти сообщения отображаются под полями ввода.

Оба этих массива были инициализированы как пустые массивы в верхней части файла `register.php` в шаге 3 на предыдущей странице.

Метод `create()` класса `Member` добавляет пользователя в базу данных. Он действует по тому же принципу, который использовался для создания категорий. Если пользователь добавлен успешно, метод возвращает значение `true`. Если адрес электронной почты уже используется, он возвращает значение `false`.

6. Метод `create()` принимает один параметр: массив, содержащий данные пользователя, и возвращает логическое значение.
7. Функция `password_hash()` хеширует введенный пользователем пароль.
8. Код добавления пользователя заключается в блок `try`, чтобы поймать исключение дублирования.
9. Запрос `INSERT` добавляет имя пользователя, его фамилию, адрес электронной почты и хеш пароля в таблицу `member` базы данных. База данных создает значения для столбцов `id`, `joined` и `role` автоматически.
10. Выполнение запроса SQL.
11. Если выполнение кода дошло до этого места, это значит, что запрос выполнился успешно, и поэтому метод возвращает значение `true`.
12. Если же при выполнении запроса возникнет ошибка, то будет выброшено исключение и запущен код в блоке `catch`.
13. Если код ошибки 1062, это указывает на то, что запись не может быть добавлена, поскольку такой email уже есть в базе данных и его добавление нарушит ограничение уникальности. В этом случае функция возвращает значение `false`.
14. Если же это какая-то другая ошибка, то исключение создается повторно с использованием ключевого слова `throw`, чтобы оно могло быть поймано в процессе обработки исключений по умолчанию.

TWIG c16/templates/register.html

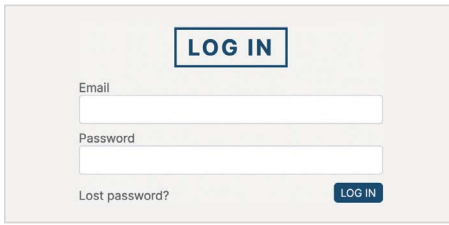
```
{% extends 'layout.html' %}
{% block title %}Register{% endblock %}                                    (1)
{% block description %}Register for Creative Folk{% endblock %}
{% block content %}                                                        (2)
<main class="container" id="content">
  <section class="header"><h1>Register</h1></section>
  <form method="post" action="register.php" class="form-membership">       (3)
    {% if errors %}<div class="alert alert-danger">Please correct errors</div>{% endif %}   (4)
    <label for="forename">Forename: </label>
    <input type="text" name="forename" value="{{ member.forename }}" id="forename">
    <div class="errors">{{ errors.forename }}</div>
    <label for="surname">Surname: </label>
    <input type="text" name="surname" value="{{ member.surname }}" id="surname">
    <div class="errors">{{ errors.surname }}</div>
    <label for="email">Email address: </label>
    <input type="email" name="email" value="{{ member.email }}" id="email">   (5)
    <div class="errors">{{ errors.email }}</div>
    <label for="password">Password: </label>
    <input type="password" name="password" id="password">
    <div class="errors">{{ errors.password }}</div>
    <label for="confirm">Confirm password: </label>
    <input type="password" name="confirm" id="confirm">
    <div class="errors">{{ errors.confirm }}</div>
    <input type="submit" class="btn btn-primary" value="Register">
  </form>
</main>
{% endblock %}
```

PHP c16/src/classes/CMS/Member.php

```
(6) public function create(array $member): bool
{
(7)     $member['password'] = password_hash($member['password'], PASSWORD_DEFAULT); // Хеш
(8)     try {                                              // Чтобы поймать исключение
(9)         $sql = "INSERT INTO member (forename, surname, email, password)
                VALUES (:forename, :surname, :email, :password);"; // SQL-запрос
(10)        $this->db->runSQL($sql, $member);          // Выполнение запроса
(11)        return true;                               // Возврат значения true
(12)    } catch (\PDOException $e) {                   // Ловим только PDOException
            if ($e->errorInfo[1] === 1062) {           // Если ошибка дублирования email
(13)            return false;            // Возвращаем false, чтобы сообщить о дублировании
            }                                          // В противном случае
(14)        throw $e;                                  // Повторный выброс исключения
        }
}
```

ВХОД В СИСТЕМУ И ПЕРСОНАЛИЗАЦИЯ

Когда пользователи возвращаются на сайт и входят в систему, их просят указать адрес электронной почты, чтобы **идентифицировать** их, и пароль, чтобы **аутентифицировать**, что они те, за кого себя выдают.



Страница `login.php` позволяет пользователям войти в систему. При отправке формы входа на сервер вызывается новый метод под названием `login()` класса `Member`. Он ищет введенный адрес электронной почты в таблице `member` базы данных и получает данные, включая хеш пароля.

Если пароль, введенный пользователем при входе в систему, создает тот же хеш, который сохранила для него база данных, сайт предполагает, что участник — тот, за кого он себя выдает, и разрешает войти в систему. Как только пользователь войдет в систему, сайт выполнит две ключевые задачи:

- **создаст сессию**, чтобы запомнить его во время текущего посещения. В ней будут храниться ключевые данные об участнике и тот факт, что он вошел в систему;
- **персонализирует страницы** для этого участника с помощью специфичной для него информации.

СЕССИИ

Как только участник вошел в систему, для него создается сессия. Она позволяет сайту идентифицировать этого участника каждый раз, когда он запрашивает другую страницу во время текущего посещения сайта. В ней хранятся идентификатор, имя и роль участника, поскольку они используются на панели навигации для:

- добавления ссылки на страницу профиля участника. Ссылка использует идентификатор участника в строке запроса;
- отображения имени в качестве текста ссылки для объекта ссылки;
- добавления ссылки на страницу администрирования, если роль участника — администратор.

Поскольку для создания панели навигации каждой странице необходимо будет работать с сессиями, будет создан новый класс `Session`. Он поможет работать с данными в суперглобальном массиве `$_SESSION`:

- если пользователь вошел в систему, данные его сессии добавляются в свойства объекта `Session`;
- если нет, то значениям этих свойств будут автоматически присвоены значения по умолчанию.

Класс `Session` также содержит методы для создания, обновления и удаления сеансов. Класс группирует код, используемый для работы с сессиями, и уменьшает объем кода, необходимого для каждой страницы. Объект `Session` создается в файле `bootstrap.php`, а также доступен в глобальной переменной Twig, чтобы все шаблоны могли получить доступ к данным сессии.

Как только участник вошел в систему, сайт может создавать страницы, адаптированные для этого пользователя, на основе информации, хранящейся о нем в базе данных.

ПЕРСОНАЛИЗАЦИЯ

Как только сайт сможет идентифицировать отдельного пользователя, он сможет настраивать страницы на основе предпочтений и профиля этого пользователя.

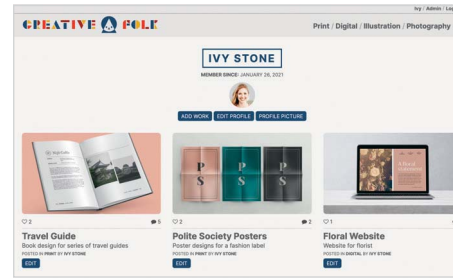
На левой странице уже описано, как при входе пользователя в систему на панели навигации отображается ссылка на страницу его профиля, а если пользователь относится к группе администраторов сайта, на ней отображается ссылка на страницы администрирования.

Кроме того, когда пользователь посещает свою страницу `member.php`, на ней отображаются ссылки, позволяющие ему добавлять или редактировать свои работы и обновлять свой профиль (как показано справа).

Тот же самый файл `member.php` используется для отображения сведений и работы каждого участника сайта, но эти дополнительные ссылки отображаются только тогда, когда участник вошел в систему и просматривает свою страницу.

Помимо создания персонализированных страниц этот раздел защищает страницы администрирования, поскольку они должны просматриваться только администраторами. До этого момента любой желающий мог их просмотреть.

Чтобы защитить страницы администрирования, в файл `functions.php` добавлена новая функция, называемая `is_admin()`.
Она вызывается в начале каждой страницы администратора. Если пользователь не вошел в систему и не является администратором, он не сможет просмотреть страницу.



Загружаемый код содержит следующие файлы, которые не приводятся в книге, поскольку они похожи на уже знакомые вам файлы:

- `work.php` позволяет пользователям загружать и редактировать свои работы. Он похож на файл `article.php` в главе 14 (за исключением того, что здесь изображение является обязательным и для параметра `published` сразу устанавливается значение `true`);
- `profile-edit.php` позволяет пользователю редактировать свой профиль. Он похож на страницу администратора, используемую для редактирования категорий;
- `profile-pic-delete.php` удаляет изображение профиля. Он похож на файл `image-delete.php`, применяемый для удаления изображений статей;
- `profile-pic-upload.php` позволяет пользователям загружать новые изображения профиля. Он использует ту же технику, что и файл `article.php` для загрузки изображений.

ВХОД В СИСТЕМУ (ЧАСТЬ 1)

Когда пользователь возвращается на сайт, страница входа `login.php` позволяет ему войти в систему. Если он вводит правильные сведения, создается новый сеанс, в котором сохраняются сведения о нем, на время этого посещения сайта.

1. Включение строгой типизации, импорт в пространство имен класса `Validate`, подключение файла `bootstrap.php`.
2. Инициализация переменной `$email` и массива `$errors`. Они необходимы для создания массива `$data`, требующегося шаблонам Twig в шаге 15.
3. Если строка запроса содержит элемент `success`, его значение сохраняется в переменной `$success`. Это значение добавляется в строку запроса при регистрации нового пользователя.
4. Конструкция `if` проверяет, была ли форма отправлена.
5. Если была, то получаем адрес электронной почты и пароль из суперглобального массива `$_POST` и присваиваем их переменным `$email` и `$password`.
6. Валидация адреса электронной почты и пароля. Если есть ошибки, то сообщения об этом сохраняются в массиве `$errors`.
7. Все значения в массиве `$errors` объединяются в одну строку и помещаются в переменную `$invalid` с помощью функции PHP `implode()`.
8. Конструкция `if` проверяет, содержит ли переменная `$invalid` непустое значение.
9. Если да, то в массив `$errors` добавляется элемент `message` с сообщением, в котором пользователю предлагается повторить попытку.
10. В противном случае данные были введены корректно.
11. Вызывается метод `login()` класса `Member`. Он проверяет, есть ли адрес электронной почты в базе данных и правильно ли пользователь ввел пароль. Если сведения верны, метод возвращает данные пользователя в виде массива. Если нет, то он возвращает значение `false`. Возвращаемое им значение помещается в переменную `$member`. Конструкции `if`, `elseif` и `else` обрабатывают результат выполнения метода.
12. Если переменная `$member` содержит данные пользователя и при этом его роль — `suspended`, то в массив `$errors` пишется сообщение о том, что учетная запись заблокирована.
13. Иначе, если в `$member` содержится непустое значение, пользователь успешно вошел в систему.
14. Для этого посетителя создается сессия с использованием метода `create()` нового объекта `Session`.
15. Если вход в систему прошел успешно, то пользователь перенаправляется на страницу своего профиля (а остальной код на странице прекращает выполнение). С этого момента в навигации:

- ссылка для входа в систему заменяется на ссылку выхода из системы;
- появляется ссылка на страницу профиля пользователя;
- отображаются ссылки на раздел администрирования, если пользователь относится к группе администраторов.

16. В противном случае пользователь с таким паролем не был найден, поэтому в массив `$errors` пишется сообщение, в котором пользователю предлагается повторить попытку.
17. Массив `$data` получает данные, необходимые в шаблоне, метод Twig `render()` создает страницу.

PHP c16/public/login.php

```php
<?php
declare(strict_types = 1);                    // Строгая типизация
use PhpBook\Validate\Validate;                // Импорт пространства имен

include '../src/bootstrap.php';               // Файл настроек

$email   = '';                                // Инициализация переменной email
$errors  = [ ];                               // Инициализация массива ошибок
$success = $_GET['success'] ?? null;          // Получение сообщения из строки запроса

if ($_SERVER['REQUEST_METHOD'] == 'POST') {               // Если форма отправлена
    $email    = $_POST['email'];                          // Получение адреса e-mail
    $password = $_POST['password'];                       // Получение пароля
    $errors['email']    = Validate::isEmail($email)
        ? '' : 'Please enter a valid email address';      // Валидация e-mail
    $errors['password'] = Validate::isPassword($password)
        ? '' : 'Passwords must be at least 8 characters and have:<br>
                A lowercase letter<br>An uppercase letter<br>A number<br>
                And another character';                   // Валидация пароля
    $invalid = implode($errors);                          // Слияние ошибок в строку

    if ($invalid) {                                       // Если есть ошибки
        $errors['message'] = 'Please try again.';         // Сообщить об этом
    } else {                                              // Если ошибок не было
        $member = $cms->getMember()->login($email, $password); // Вход в систему
        if ($member and $member['role'] == 'suspended') {      // Если заблокирован
            $errors['message'] = 'Account suspended';          // Сообщить об этом
        } elseif ($member) {                        // Для всех остальных пользователей
            $cms->getSession()->create($member);               // Записываем в сессию
            redirect('member.php', ['id' => $member['id'],]);  // Перенаправление
        } else {                                               // Пользователь не найден
            $errors['message'] = 'Please try again.';          // Сообщаем об этом
        }
    }
}

$data['navigation'] = $cms->getCategory()->getAll(); // Категории для навигации
$data['success']    = $success;                      // Сообщение об успешном завершении
$data['email']      = $email;          // Адрес email, если не удалось войти в систему
$data['errors']     = $errors;                       // Массив ошибок
echo $twig->render('login.html', $data);             // Рендеринг шаблона
```

Примечание. Сообщения об ошибках не должны показывать, что адрес электронной почты указан правильно, но пароль неверен, поскольку это подтверждает, что адрес электронной почты был зарегистрирован на сайте.

Упражнение. После того как вы воспользовались объектом `Session`, добавьте конструкцию `if` между шагами 2 и 3, чтобы проверить, вошел ли уже пользователь в систему. Если вошел, перенаправьте его на страницу пользователя.

ВХОД В СИСТЕМУ (ЧАСТЬ 2)

В первом блоке кода показан шаблон Twig для формы входа в систему. В загружаемом коде форма содержит больше HTML-элементов и атрибутов для управления отображением, но здесь некоторые из них были удалены, чтобы вы могли сосредоточиться на важном функционале и чтобы код помещался на странице.

1. Шаблон Twig расширяет шаблон `layout.html` и определяет новое содержимое для блоков `title` и `description`.
2. Блок `content` содержит форму входа в систему.
3. Форма отправляется на ту же страницу PHP.
4. Если переменная `success` содержит значение (оно не равно `null`), это значит, что пользователь только что зарегистрировался и ему отображается сообщение об этом.
5. Если массив ошибок не пустой, то отображается значение элемента `warning`.
6. В форме есть поля ввода для адреса электронной почты и пароля. Если в массиве `$errors` есть сообщения об ошибках, они отображаются под соответствующими полями ввода (в сообщении об ошибке для пароля используется фильтр Twig `raw`, поскольку он использует HTML-разметку — см. шаг 7 на предыдущей странице)[77].
Второй блок кода показывает новый метод `login()` класса `Member`. Он проверяет правильность адреса электронной почты и пароля. Если данные верны, он возвращает информацию об этом пользователе. Если нет, то он возвращает значение `false`.
7. Методу `login()` требуется адрес электронной почты и пароль.
8. Переменная `$sql` содержит SQL-запрос для получения данных пользователя по его адресу электронной почты.
9. Результат запроса записывается в переменную `$member`.
10. Если такой пользователь не был найден, метод `login()` завершает выполнение, возвращая значение `false`.
11. Если же код продолжает выполняться, это значит, что такой пользователь найден. В этом случае функция PHP `password_verify()` создает хеш из пароля, указанного при входе в систему, и проверяет, соответствует ли он их хешу в базе данных. Она возвращает значение `true`, если они совпадают, и `false`, если нет. Результат сохраняется в переменной с именем `$authenticated`.
12. Тернарный оператор проверяет, содержит ли переменная `$authenticated` значение `true`. Если это так, метод возвращает массив `$member`. Если нет, то он возвращает значение `false`.
В последнем блоке кода показан фрагмент файла `bootstrap.php`, содержащего код, используемый на каждой странице. Как показано ранее, при входе пользователя в систему его идентификатор, имя, фамилия и роль сохраняются в сессии, потому что каждая страница должна иметь доступ к этим данным, чтобы создать панель навигации. Как вы видели в главе 9, когда сайт использует сеансы, каждая страница сайта должна:

- вызвать функцию PHP `session_start();`
- проверить, содержит ли суперглобальный массив `$_SESSION` данные, к которым страница пытается получить доступ, прежде чем обращаться к ним (если этого не сделать, может произойти ошибка `Undefined index`).

Вместо того чтобы повторять код для этой операции на каждой странице, файл `bootstrap.php` (подключаемый на каждой странице) создает объект, используя новый класс `Session` (показан далее). Этот код помещается в метод `__construct()`, чтобы он выполнялся сразу при создании объекта. Если пользователь вошел в систему, код берет данные из суперглобального массива `$_SESSION` и сохраняет их в свойствах объекта `Session`.
13. В файле `bootstrap.php` создается объект класса `Session`.
14. Затем он сохраняется в глобальной переменной Twig, чтобы получить доступ к его свойствам из любого шаблона.

TWIG c16/templates/login.html

```
{% extends 'layout.html' %}
① {% block title %}Log In{% endblock %}
  {% block description %}Log in to your Creative Folk account{% endblock %}
② {% block content %}
  <main class="container" id="content">
③   <form method="post" action="login.php" class="form-membership">
      <section class="header"><h1>Log in:</h1></section>
④     {% if success %}<div class="alert alert-success">{{ success }}</div>{% endif %}
⑤     {% if errors %}<div class="alert alert-danger">{{ errors.message }}</div>{% endif %}

      <label for="email">Email: </label>
      <input type="text" name="email" id="email" value="{{ email }}" class="form-control">
      <div class="errors">{{ errors.email }}</div>
      <label for="password">Password: </label>
⑥     <input type="password" name="password" id="password" class="form-control">
      <div class="errors">{{ errors.password|raw }}</div>
      <input type="submit" class="btn btn-primary" value="Log in"><br>
      <p><a href="password-lost.php">Lost password?</a></p>
    </form>
  </main>
  {% endblock %}
```

PHP c16/src/classes/CMS/Member.php

```
⑦ public function login(string $email, string $password)
  {
      $sql = "SELECT id, forename, surname, joined, email, password, picture, role
⑧               FROM member
               WHERE email = :email;";        // SQL для получения данных пользователя
⑨     $member = $this->db->runSQL($sql, [$email])->fetch(); // Выполнение запроса
      if (!$member) {                                       // Если не найден
⑩         return false;                                     // Возврат значения false
      }                                                     // В противном случае
⑪     $authenticated = password_verify($password, $member['password']); // Пароль ОК?
⑫     return ($authenticated ? $member : false);          // Или пользователь, или false
  }
```

PHP c16/src/bootstrap.php

```
  $loader = new Twig\Loader\FilesystemLoader(APP_ROOT . '/templates'); // Настройка Twig
  $twig   = new Twig\Environment($loader, $twig_options);  // Среда Twig
  $twig->addGlobal('doc_root', DOC_ROOT);   // Глобальная переменная Twig с корнем сайта
⑬ $session = $cms->getSession();            // Создание сессии
⑭ $twig->addGlobal('session', $session);     // Глобальная переменная с сессией
```

ИСПОЛЬЗОВАНИЕ СЕССИЙ ДЛЯ ХРАНЕНИЯ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ

В верхней части сайта всегда показывается, вошел пользователь в систему или нет.

- Если вошел, заголовок содержит ссылку на его страницу, использующую имя пользователя в качестве текста ссылки и его идентификатор в строке запроса.
- Если он не вошел в систему, то выводятся ссылки на страницы `login.php` и `register.php`.

Если пользователь вошел в систему и относится к администраторам, в заголовке также отображается ссылка на раздел администрирования.

Чтобы создать эти ссылки, каждая страница должна знать идентификатор пользователя, его имя и роль, поэтому эта информация хранится в сессии.

Как показано на предыдущей странице, в файле `bootstrap.php`, подключаемом на каждой странице, создается объект пользовательского класса `Session`, показанный справа. Класс `Session` используется, чтобы группировать код для создания, чтения, обновления и удаления переменных сессии в одном месте. В этом классе определено три свойства[78]:

- `id` содержит идентификатор пользователя;
- `forename` содержит его имя;
- `role` содержит информацию о его роли.

При создании этого объекта его метод `__construct()`:

- Вызывает `session_start()`.
- Присваивает значения свойствам объекта. Если в сессии содержатся данные пользователя, то они присваиваются свойствам объекта. В противном случае присваиваются значения по умолчанию.

1. Объявление пространства имен для класса.
2. Объявление класса `Session`.
3. Объявление трех свойств. Они хранят идентификатор, имя и роль пользователя.
4. Метод `__construct()` автоматически запускается при создании объекта сессии с использованием этого класса.
5. Функция PHP `session_start()` стартует новую сессию или загружает существующую.
6. Если суперглобальный массив `$_SESSION` содержит ключи с именами `id`, `forename` и `role`, их значения сохраняются в свойствах объекта `Session`. Если нет, то в этих свойствах сохраняются значения по умолчанию.
7. Метод `create()` вызывается при входе пользователя в систему. В качестве параметра ему требуется массив, содержащий данные пользователя.
8. Функция PHP `session_regenerate_id()` обновляет идентификатор сессии, используемый как в файле сессии, так и в cookie.
9. Идентификатор, имя пользователя и роль пользователя добавляются в суперглобальный массив `$_SESSION`.
10. В данном случае для создания или обновления сессии требуется один и тот же код. Чтобы не повторять его, метод `update()` просто вызывает метод `create()`. Таким образом метод `update()` становится **псевдонимом** метода `create()`, поскольку это альтернативное имя, используемое для вызова кода, расположенного в методе `create()`[79].
11. Метод `delete()` вызывается, когда пользователь нажимает на ссылку выхода из системы в навигации. Его задача — завершить сессию.

PHP c16/src/classes/CMS/Session.php

```php
<?php
namespace PhpBook\CMS;                                  // Пространство имен

class Session
{                                                       // Определение класса Session
    public $id;                                         // id пользователя
    public $forename;                                   // Имя пользователя
    public $role;                                       // Роль

    public function __construct()
    {                                                   // Выполняется при создании объекта
        session_start();                                // Стартовать или обновить сессию
        $this->id       = $_SESSION['id'] ?? 0;         // Установка свойства id
        $this->forename = $_SESSION['forename'] ?? '';  // Установка свойства forename
        $this->role     = $_SESSION['role'] ?? 'public'; // Установка свойства role
    }

    // Создать новую сессию – также используется для обновления существующей
    public function create($member)
    {
        session_regenerate_id(true);                    // Смена id сессии
        $_SESSION['id']       = $member['id'];   // Добавление в сессию id пользователя
        $_SESSION['forename'] = $member['forename'];    // Добавление имени
        $_SESSION['role']     = $member['role'];        // Добавление роли
    }

    // Обновить текущую сессию – псевдоним для create()
    public function update($member)
    {
        $this->create($member);
    }

    // Удаление текущей сессии
    public function delete()
    {
        $_SESSION = [ ];                // Очистить суперглобальный массив $_SESSION
        $param    = session_get_cookie_params(); // Получение параметров cookie сессии
        setcookie(session_name(), '', time() - 2400, $param['path'], $param['domain'],
            $param['secure'], $param['httponly']);      // Очистить сессионный cookie
        session_destroy();                              // Уничтожить сессию
    }
}
```

ПЕРСОНАЛИЗАЦИЯ ПАНЕЛИ НАВИГАЦИИ

Объект `Session`, созданный в файле `bootstrap.php`, присваивается глобальной переменной Twig, так что каждый шаблон может получить доступ к его свойствам. В шаблоне `layout.html`:

1) конструкция `if` проверяет, содержит ли свойство `id` объекта `Session` значение 0 (это указывает, что пользователь не вошел в систему);
2) если это так, то в шаблоне отображаются ссылки для входа и регистрации;
3) в противном случае это означает, что пользователь вошел в систему;
4) создается ссылка на личную страницу прользователя с указанием его имени в тексте ссылки;
5) конструкция `if` проверяет, содержит ли свойство роли объекта `Session` значение `admin`. Если это так, отобразится ссылка на раздел администрирования;
6) ссылка на файл `logout.php` позволяет пользователю выйти из системы. Файл `logout.php` находится в загружаемом коде. Он просто вызывает метод `delete()` объекта `Session`, а затем перенаправляет пользователя на домашнюю страницу.

c16/templates/layout.html — TWIG

```
1 {% if session.id == 0 %}
2   <a href="login.php" class="nav-item nav-link">Log in</a> /
2   <a href="register.php" class="nav-item nav-link">Register</a>
3 {% else %}
4   <a href="member.php?id={{ session.id }}">{{ session.forename }}</a> /
5   {% if session.role == 'admin' %}
5     <a href="admin/index.php">Admin</a> /
5   {% endif %}
6     <a href="logout.php">Logout</a>
  {% endif %}
```

РЕЗУЛЬТАТ



ДОБАВЛЕНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ НА СТРАНИЦУ ПРОФИЛЯ

Файл `member.php` отображает профиль зарегистрированного пользователя и краткие описания его работ. Если пользователь вошел в систему и просматривает свой профиль, ему отображаются ссылки для редактирования информации о себе и добавления новой работы.

1. В шаблоне `member.html` оператор Twig `if` проверяет, совпадает ли идентификатор пользователя (хранится в сессии) с идентификатором пользователя, чья работа отображается на экране. Если они совпадают, новые ссылки отображаются под профилем.

2. На странице `article-summaries.html` другой оператор Twig `if` проверяет, совпадает ли идентификатор пользователя, просматривающего страницу, с идентификатором пользователя, разместившего публикацию. Если да, то будет добавлена ссылка для редактирования этой работы.

Файл `work.php`, позволяющий пользователям загружать работы, находится в загружаемом коде. Он похож на файл `article.php` в разделе администратора, но требуется изображение, участник — это автор, и отсутствует возможность публикации.

TWIG c16/templates/member.html

```
{% if session.id == member.id %}
<nav class="member-options">
  <a href="work.php" class="btn btn-primary">Add work</a>
  <a href="member-edit-profile.php" class="btn btn-primary">Edit profile</a>
  <a href="member-edit-picture.php" class="btn btn-primary">Profile picture</a>
</nav>
{% endif %}
```

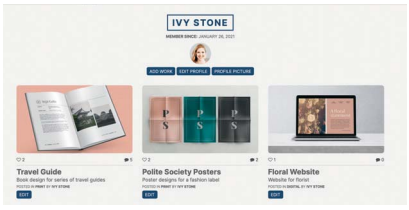
(1)

TWIG c16/templates/article-summaries.html

```
{% if session.id == article.member_id %}
  <a href="work.php?id={{ article.id }}" class="btn btn-primary">Edit</a>
{% endif %}
```

(2)

РЕЗУЛЬТАТ



Страницы для редактирования профиля и страницы для загрузки или удаления изображений профиля находятся в загружаемом коде. Вы можете попробовать написать код для этих скриптов самостоятельно, чтобы проверить свои навыки:

- страница для редактирования профиля похожа на код для редактирования категории;
- код для добавления/удаления изображений профиля похож на код для добавления или удаления изображений в статьях.

ОГРАНИЧЕНИЕ ДОСТУПА К СТРАНИЦАМ АДМИНИСТРИРОВАНИЯ

В предыдущих главах любой желающий мог получить доступ к страницам администратора сайта. В этой главе доступ к этим страницам могут получить только пользователи с ролью администратора.

1. На каждой странице администрирования вызывается новая функция с именем `is_admin()` сразу после включения файла `bootstrap.php`. Ей требуется роль участника в качестве аргумента. Если пользователь не вошел в систему, объект `Session` устанавливает для роли значение `public`.
2. Определение функции `is_admin()` добавляется в файл `functions.php`.
3. Конструкция `if` проверяет, содержит ли роль значение `admin`.
4. Если не содержит, пользователь перенаправляется на домашнюю страницу, а не на страницу login.php, чтобы люди, не являющиеся администраторами, не могли угадывать URL-адреса страниц администратора.
5. Команда `exit` останавливает выполнение любого другого кода на странице, вызвавшей функцию. Если пользователь — администратор, выполняется остальная часть страницы.

c16/public/admin/article.php **PHP**

```php
<?php
// Часть A: Настройка
declare(strict_types = 1);                    // Используем строгие типы
use PhpBook\Validate;                         // Импорт пространства имен Validate

include '../../src/bootstrap.php';            // Включение файла настройки
(1) is_admin($session->role);        // Проверка на принадлежность к группе администраторов
```

c16/src/functions.php **PHP**

```php
(2) function is_admin($role)
{
(3)     if ($role !== 'admin') {                    // Если роль не равна admin
(4)         header('Location: ' . DOC_ROOT);        // Отправить на домашнюю страницу
(5)         exit;                                   // Остановить выполнение кода
    }
}
```

Обычно требуется, чтобы пользователь вошел в систему, прежде чем он сможет обновлять информацию в базе данных. До сих пор в этой главе пользователь должен был войти в систему, чтобы сделать это.

В редких случаях вы может потребоваться обновить определенную информацию в базе данных без входа в систему, но это требует дополнительных мер безопасности, о которых вы узнаете далее.

ТОКЕНЫ, ПОЗВОЛЯЮЩИЕ ОБНОВЛЯТЬ ИНФОРМАЦИЮ В БД ПО ССЫЛКЕ ИЗ ПИСЬМА

Сайты иногда позволяют пользователю обновлять информацию в базе данных без входа в систему. Часто это происходит, когда ему было отправлено электронное письмо, содержащее ссылку, например ссылку для сброса пароля. В ссылке используется одноразовый **токен** для идентификации пользователя, для которого она предназначена.

Если пользователь забудет свой пароль, он не сможет войти на сайт, чтобы сбросить его, поэтому сайт должен предложить ему другой способ безопасного обновления пароля.

Одно из решений — отправка пользователю по электронной почте ссылки на страницу, которую он может использовать для обновления своего пароля. Поскольку ссылка отправляется на адрес электронной почты этого пользователя, он должен быть единственным человеком, способным ее использовать.

При этом в ссылке должна быть какая-то информация, по которой можно определить пользователя, которому нужно сбросить пароль. Но это не должен быть адрес электронной почты или идентификатор из столбца `id` таблицы `member`, поскольку их легко угадать и, соответственно, хакер сможет легко подделать такую ссылку, перейти по ней и сбросить пароль другого пользователя, а затем войти в его учетную запись.

Вместо этого, когда пользователь просит сбросить свой пароль, создается уникальный одноразовый токен для его идентификации пользователя. Токен представляет собой случайный уникальный набор символов, его не так просто угадать, например:

```
0d9781153ed42ea7d72b4a4963dbd4f7fbc1d09
bca10a8faae55d5dd66441521881a4e51eb17cd
62596b156f11218d31436e5ae3381bcb50acbf3
1dd2c5cd197
```

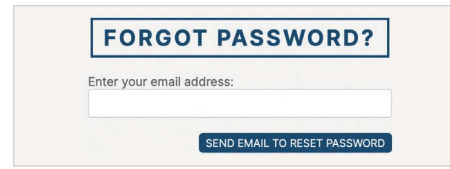
Этот токен:

- хранится в базе данных в новой таблице под названием `token`;
- используется для идентификации пользователя.

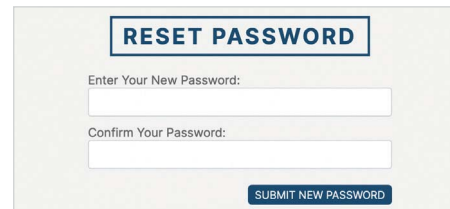
Когда пользователь нажимает на ссылку, содержащую токен, сайт делает запрос к таблице `token` в базе данных, чтобы определить, для какого пользователя он был создан.

На нескольких следующих страницах показано, как используются токены, когда пользователь хочет сбросить свой пароль.

Сначала пользователь вводит свой адрес электронной почты на странице `password-lost.php`.



Когда эта форма отправляется, сайт проверяет, есть ли пользователь с указанным адресом электронной почты. Если есть, сайт добавляет новый токен в таблицу `token` базы данных и отправляет пользователю по электронной почте ссылку на страницу `password-reset.php`. Она позволяет обновить свой пароль. Токен используется в ссылке, чтобы определить, какой участник пытается обновить свой пароль.



При обновлении пользователем пароля используется новый метод класса `Member` под названием `passwordUpdate()`.

ХРАНЕНИЕ ТОКЕНОВ В БАЗЕ ДАННЫХ

Новый класс, называемый Token, используется для создания объекта Token. Этот класс позволяет генерировать токены, хранить их в новой таблице token в базе данных и возвращает идентификатор пользователя, для которого токен создан.

В новой таблице токенов должны храниться как минимум токен и идентификатор пользователя, для которого он создан. Для дополнительной безопасности каждый токен также содержит:

- **время хранения**, чтобы токен не оставался действительным долгое время после того, как был предназначен для использования. Для нашего сайта это время составляет четыре часа после создания токена;
- **назначение**. Сайт может использовать токены для нескольких задач, и сохранение цели позволяет сайту проверять, что токен используется по назначению.

Еще один популярный повод использования токенов — при регистрации пользователей на сайте. Им может быть отправлена ссылка по электронной почте, на которую они должны нажать, прежде чем смогут войти в систему. Это подтверждает, что их адрес электронной почты был указан правильно.

Класс с именем Token используется для создания объекта Token, содержащего два метода:

- **`create()`** создает новый токен и сохраняет его в базе данных.
- **`getMemberId()`** проверяет, не истек ли срок действия токена и используется ли он по назначению. Если все верно, он вернет идентификатор участника, для которого был создан.

Объект Token создается с использованием нового метода с названием getToken() объекта CMS. Он хранится в свойстве под названием token в объекте CMS (используя тот же подход, что и для объектов Article, Category и Member).

token

token	member_id	expires	purpose
a730730065407fa0a0508cc7f06930ed962...	4	2021-03-08 14:04:01	password_reset
4fbb47d3ebd4c0f3269ef669e4123cc8a2d...	12	2021-03-08 14:05:09	password_reset
ba5fde0992dfc85b39397bf4df89ecaa25d...	9	2021-03-08 14:05:38	password_reset

Токен состоит из 64 случайных символов. Он генерируется с использованием двух встроенных методов PHP:

- `random_bytes()`создает строку случайных байтов, ее параметр — количество возвращаемых байтов;
- `bin2hex()` преобразует двоичные данные в шестнадцатеричные.



PHP c16/src/classes/CMS/Token.php

```php
<?php
namespace PhpBook\CMS;                              // Пространство имен
class Token
{
    protected $db;                                  // Объект Database

    public function __construct(Database $db)
    {
        $this->db = $db;                        // Объект Database присвоен свойству $db
    }
    public function create(int $id, string $purpose): string
    {
        $arguments['token']     = bin2hex(random_bytes(64));                  // Токен
        $arguments['expires'] = date('Y-m-d H:i:s', strtotime('+4 hours')); // Срок хранения
        $arguments['member_id'] = $id;                                    // ID пользователя
        $arguments['purpose']   = $purpose;                                  // Назначение
        $sql = "INSERT INTO token (token, member_id, expires, purpose)
                VALUES (:token, :member_id, :expires, :purpose);";    // SQL
        $this->db->runSQL($sql, $arguments);                              // Выполнение SQL
        return $arguments['token'];                                       // Возврат токена
    }
    public function getMemberId(string $token, string $purpose)
    {
        $arguments = ['token' => $token, 'purpose' => $purpose,];  // Токен и назначение
        $sql = "SELECT member_id FROM token WHERE token = :token
                    AND purpose = :purpose AND expires > NOW();"; // SQL для получения id
        return $this->db->runSQL($sql, $arguments)->fetchColumn(); // Возврат id / false
    }
}
```

1. Объект должен работать с базой данных, поэтому метод `__construct()` получает в качестве аргумента переменную с объектом базы данных и сохраняет его в свойстве `$db`.

2. Метод `create()` создает новый токен и сохраняет его в таблице `token` базы данных.

3. Создание токена и сохранение его в массиве с `$arguments`, который будет использоваться при выполнении запроса SQL.

4. Дата и время истечения срока действия токена (4 часа от текущего времени) добавляются в массив `$arguments`.

5. Идентификатор пользователя и назначение токена также добавляются в массив.

6. Переменная `$sql` содержит SQL-запрос для добавления токена в базу данных.

7. Выполнение запроса.

8. Новый токен возвращается из метода.

9. Метод `getMemberId()` проводит валидацию токена. Он возвращает идентификатор пользователя, такой токен есть и не просрочен, или значение `false` в противном случае.

10. Токен и его назначение сохраняются в массиве.

11. SQL-запрос ищет строку в таблице `token`, содержащую указанные токен и назначение и у которой время действия больше текущего. Если все три условия выполняются, то запрос вернет идентификатор пользователя.

12. Выполняется запрос SQL и функция возвращает его результат. Это будет либо идентификатор пользователя, либо `false`, если условия не совпали.

ЗАПРОС НА СБРОС ПАРОЛЯ

Страница password-lost.php (справа) отображает форму, позволяя пользователю ввести адрес электронной почты и запросить ссылку для обновления пароля. После отправки сайт:

- проверяет, есть ли в БД пользователь с таким адресом, и получает его идентификатор;
- создает токен для сброса пароля и сохраняет его в базе данных;
- создает и отправляет электронное письмо со ссылкой на страницу, которая сбросит пароль.

1. Включение строгой типизации, импорт класса Validate, подключение bootstrap.php, инициализация двух переменных, которые понадобятся в шаблоне Twig.
2. Конструкция if проверяет, была ли отправлена форма.
3. Если отправлена, то адрес электронной почты проверяется на соответствие формату. Если нет, то в переменную $error записывается сообщение об ошибке. В противном случае записывается пустая строка.
4. Конструкция if проверяет, содержит ли ли переменная $error пустую строку.
5. Если да, то вызывается новый метод класса Member, который называется getIdByEmail() (представлен ниже), и ему передается адрес электронной почты. Метод ищет пользователя с таким адресом в базе данных. Если находит, то возвращает идентификатор этого пользователя. Если нет, то возвращает значение false. Полученное значение сохраняется в переменной $id.
6. Еще одна конструкция if проверяет, был ли найден идентификатор.
7. Если да, то вызывается метод create() объекта Token для создания нового токена. Назначение токена устанавливается в значение password_reset. Созданный токен сохраняется в переменной $token.
8. Создается ссылка на страницу password-reset.php. Она содержит новый токен в строке запроса. Чтобы создать эту ссылку, сайту необходимо знать имя домена сайта. Оно содержится в новой константе с именем DOMAIN, объявленной в файле config.php. Если вы еще не сделали этого, откройте этот файл и добавьте свое имя хоста к этой константе.
9. Создание темы и текста электронного письма.
10. Создание объекта Email с использованием класса Email. Затем отправляется электронное письмо. Если отправка прошла успешно, переменная $sent получит значение true.
11. В массив $data пишется информация, необходимая Twig, и вызывается метод render().
12. Шаблон password-lost.html (показан во втором блоке кода справа) создает форму. Оператор Twig if проверяет, содержит ли переменная sent значение false. Если да, пользователю будет показана форма для запроса ссылки на сброс пароля. Если нет, пользователь увидит сообщение о том, что ему отправлены инструкции для сброса пароля по электронной почте.

c16/src/classes/CMS/Member.php **PHP**

```
public function getIdByEmail(string $email)
{
    $sql = "SELECT id FROM member
             WHERE email = :email;";                    L для поиска пользователя
    return $this->db->runSQL($sql, [$email])->fetchColumn(); // Возврат результата
}
```

PHP c16/public/password-lost.php

```php
<?php
declare(strict_types = 1);                          // Строгая типизация
use PhpBook\Validate\Validate;                      // Импорт пространства имен Validate
include '../src/bootstrap.php';                     // Файл начальной загрузки
$error = false;                                     // Сообщение об ошибке
$sent  = false;                                     // Было ли отправлено письмо

if ($_SERVER['REQUEST_METHOD'] == 'POST') {         // Если форма отправлена
    $email = $_POST['email'];                       // Email пользователя
    $error = Validate::isEmail($email) ? '' : 'Please enter your email'; // Проверка
    if ($error === '') {                                // Если формат адреса правильный
        $id = $cms->getMember()->getIdByEmail($email); // Получение id пользователя
        if ($id) {                                                      // Если найден
            $token = $cms->getToken()->create($id, 'password_reset');   // Токен
            $link  = DOMAIN . DOC_ROOT . 'password-reset.php?token=' . $token; // Ссылка
            $subject = 'Reset Password Link';               // Тема + текст письма
            $body  = 'To reset password click: <a href="' . $link . '">' . $link . '</a>';
            $mail  = new \PhpBook\Email\Email($email_config);  // Объект Email
            $sent  = $mail->sendEmail($mail_config['admin_email'], $email,
               $subject, $body);                            // Отправка письма
        }
    }
}
$data['navigation'] = $cms->getCategory()->getAll();        // Категории для навигации
$data['error']      = $error;                               // Ошибки валидации
$data['sent']       = $sent;                                // Была ли отправка

echo $twig->render('password-lost.html', $data);            // Рендеринг шаблона
```

TWIG c16/templates/password-lost.html

```twig
{% extends 'layout.html' %}
{% block title %}Password Reset{% endblock %}
{% block content %}...
  {% if sent == false %}
  <form method="post" action="password-lost.php" class="form-membership"> ...
    <label for="email">Enter your email address: </label>
    <input type="text" name="email" id="email" class="form-control"><br>
    <input type="submit" name="submit" value="Send email to reset password" class="btn">
    <span class="errors">{{ error }}</span><br>
  </form>
  {% else %}
  <p>If your address is registered, we will email instructions to reset your password.</p>
  {% endif %}...
{% endblock %}
```

СБРОС ПАРОЛЯ

Когда пользователь нажимает на ссылку в электронном письме о сбросе пароля, он отправляется на страницу `password-reset.php` (справа).

Если токен, полученный в строке запроса, является валидным, сайт показывает пользователю форму для обновления его пароля.

1. Если строка запроса содержит токен, он присваивается переменной `$token`. Если нет, то пользователь отправляется страницу `login.php`.
2. Метод `getMemberId()` объекта `Token` запрашивает из БД идентификатор пользователя, проверяя валидность токена. Если токен найден, соответствует назначению и не просрочен, то идентификатор пользователя сохраняется в переменной `$id`.
3. Если переменная `$id` пустая, то пользователь перенаправляется на страницу `login.php`. Если же id возвращается, это значит, что форма может быть отображена и обработана.
4. Конструкция `if` проверяет, была ли форма отправлена.
5. Если да, то получение и подтверждение пароля присваиваются переменным.
6. Значения проверяются, чтобы убедиться, что они соответствуют требованиям к паролю и что оба поля содержат один и тот же пароль. Любые ошибки сохраняются в массиве `$errors`.
7. Все ошибки объединяются в одну строку в переменной `$invalid`.
8. Если ошибки были обнаружены, в массив добавляется элемент `message` с сообщением об этом.
9. Если ошибок не было, то вызывается новый метод класса `Member` под названием `passwordUpdate()` для обновления пароля этого пользователя (показано ниже).
10. Получение данных пользователя.
11. Они используются для создания и отправки пользователю электронного письма, в котором сообщается, что пароль был обновлен.
12. Затем пользователь перенаправляется на страницу входа в систему с сообщением об успешном завершении. В нем говорится, что пароль был обновлен.
13. Информация, необходимая Twig, пишется в переменную `$data`.
14. Шаблон `password-reset.html` отображает форму. Его можно найти в загружаемом коде.
15. Для нового метода класса `Member`, называемого `passwordUpdate()` (ниже), требуется указать идентификатор пользователя и его новый пароль.
16. Создается хеш нового пароля.
17. Запрос SQL обновляет хеш пароля для этого пользователя, и функция возвращает значение `true`.

c16/src/classes/CMS/Member.php **PHP**

```
⑮ public function passwordUpdate(int $id, string $password): bool
  {
⑯     $hash = password_hash($password, PASSWORD_DEFAULT);    // Хеширование пароля
      $sql = 'UPDATE member
                 SET password = :password
⑰              WHERE id = :id;';                             // SQL для обновления пароля
      $this->db->runSQL($sql, ['id' => $id, 'password' => $hash,]); // Выполнение SQL
      return true;                                          // Возврат значения true
  }
```

PHP c16/public/password-reset.php

```php
<?php
declare(strict_types = 1);                                  // Строгая типизация
use PhpBook\Validate\Validate;                              // Импорт класса

include '../src/bootstrap.php';                             // Файл начальной загрузки
$errors = [ ];                                              // Инициализация массива

$token = $_GET['token'] ?? '';                              // Получение токена
if (!$token) {                                              // Если токена нет
    redirect('login.php');                                  // Перенаправление
}
$id = $cms->getToken()->getMemberId($token, 'password_reset'); // Получение id пользователя
if (!$id) {                                                 // Если id отсутствует
    redirect('login.php', ['warning' => 'Link expired, try again.',]); // Перенаправление
}

if ($_SERVER['REQUEST_METHOD'] == 'POST') {                 // Если форма отправлена
    $password = $_POST['password'];                         // Получение нового пароля
    $confirm  = $_POST['confirm'];                          // Получение подтверждения пароля
    // Валидация паролей и проверка на эквивалентность
    $errors['password'] = Validate::isPassword($password)
        ? '' : 'Passwords must be at least 8 characters and have:<br>
                A lowercase letter<br>An uppercase letter<br>A number
                <br>And a special character';               / Недействительный пароль
    $errors['confirm']  = ($password === $confirm)
        ? '' : 'Passwords do not match';                    // Пароли не совпадают
    $invalid = implode($errors);                            // Объединение сообщений об ошибках

    if ($invalid) {                                         // Если пароль не прошел валидацию
        $errors['message'] = 'Please enter a valid password.'; // Сообщение об ошибке
    } else {                                                // Иначе
        $cms->getMember()->passwordUpdate($id, $password);  // Обновление пароля
        $member  = $cms->getMember()->get($id);             // Получение данных участника
        $subject = 'Password Updated';                      // Создание темы и текста письма
        $body    = 'Your password was updated on ' . date('Y-m-d H:i:s') .
            ' - if you did not reset the password, email ' . $email_config['admin_email'];
        $email   = new \PhpBook\Email\Email($email_config); // Создание объекта email
        $email->sendEmail($email_config['admin_email'], $member['email'], $subject, $body);
        redirect('login.php, ['success' => 'Password updated']); // Перенаправление
    }                                                       // на вход в систему
}

$data['navigation'] = $cms->getCategory()->getAll();        // Категории для навигации
$data['errors']     = $errors;                              // Массив с ошибками
$data['token']      = $token;                               // Токен
echo $twig->render('password-reset.html', $data);           // Рендеринг шаблона
```

ЗАКЛЮЧЕНИЕ
РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЕЙ

- При регистрации на сайте пользователь должен предоставить уникальный идентификатор (например, адрес электронной почты) и пароль, чтобы подтвердить, что он тот, за кого себя выдает.
- Информация о пользователе сохраняется в базе данных и используется при следующих посещениях сайта.
- Когда пользователь возвращается и входит в систему, информация об этом сохраняется в сессии, которая позволяет идентифицировать его на все время этого посещения.
- Вместо того чтобы хранить сам пароль, в базе данных необходимо хранить только хеш пароля.
- Роли определяют, что разрешено делать пользователям.
- Токены могут использоваться для идентификации пользователей без указания личных данных, таких как адрес электронной почты или идентификатор.
- Токены следует использовать, когда пользователям разрешено изменять информацию в БД без входа в систему.

17

ДОБАВЛЕНИЕ НОВОГО ФУНКЦИОНАЛА

В этой главе показано, как добавлять на сайт новую функциональность. URL-адреса будут изменены, чтобы сделать их оптимизированными для SEO, а пользователи смогут ставить лайки и комментировать статьи.

Более понятные URL-адреса помогают в поисковой оптимизации (SEO, Search engine optimisation) сайта, поскольку в них используются ключевые слова, такие как заголовки статей или названия категорий.

Например, URL страницы https://eg.link/article.php?id=24 изменится на другой, содержащий ее название: https://eg.link/article/24/travel-guide.

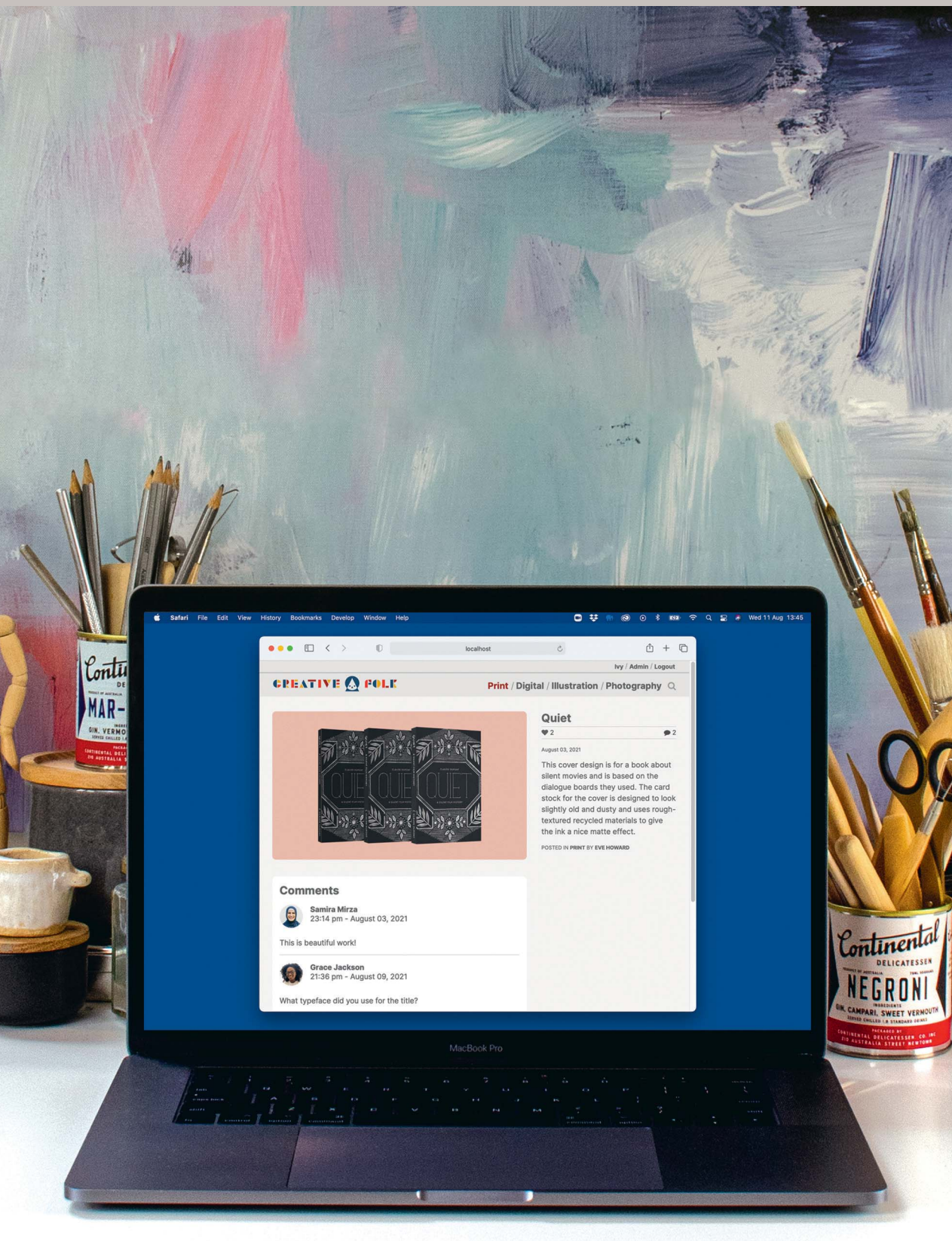
После того как вы научитесь изменять URL-адреса учебного сайта по этой новой схеме, во втором разделе к нему будут добавлены две новые функции, позволяющие зарегистрированным пользователям:

- ставить публикациям отметку **лайк** («Нравится»);
- **комментировать** публикации, чтобы добавить свое мнение или отзыв.

Добавление новой функциональности на сайт включает в себя:

- составление списка действий, которые смогут выполнять пользователи;
- определение того, какие данные необходимо будет сохранить в базе данных;
- реализация функциональности в PHP-коде и шаблонах.

Навыки, о которых вы узнаете в этом заключительном разделе, также могут быть применены и при разработке новых сайтов.


Safari File Edit View History Bookmarks Develop Window Help
Wed 11 Aug 13:45
localhost
Ivy / Admin / Logout
CREATIVE FOLK
Print / Digital / Illustration / Photography
Quiet
2
2
August 03, 2021
This cover design is for a book about silent movies and is based on the dialogue boards they used. The card stock for the cover is designed to look slightly old and dusty and uses rough-textured recycled materials to give the ink a nice matte effect.
POSTED IN PRINT BY EVE HOWARD
Comments
Samira Mirza
23:14 pm - August 03, 2021
This is beautiful work!
Grace Jackson
21:36 pm - August 09, 2021
What typeface did you use for the title?
MacBook Pro
Continental
DELICATESSEN
NEGRONI
GIN, CAMPARI, SWEET VERMOUTH

ОПТИМИЗИРОВАННЫЕ ДЛЯ SEO URL-АДРЕСА[80]

Использование в URL слов, описывающих содержимое страницы, помогает в поисковой оптимизации (SEO): такие страницы определяются в поисковых системах как более заметные. Это также облегчает чтение URL-адресов человеком.

До сих пор в этой книге в URL каждой страницы учебного сайта использовался путь к файлу PHP, который должен быть выполнен. Если на странице требовалось получить данные из БД, в строке запроса указывался идентификатор этих данных.

Многие веб-сайты используют более описательные, дружественные SEO-адреса, а не пути к файлам. Когда посетитель запрашивает эти описательный URL, веб-сервер преобразует его в путь к файлу, вызывает этот файл и сообщает ему, какие данные он должен отображать. Этот метод известен как **перезапись URL-адресов (URL rewriting)**.

Существует несколько способов написания дружественных URL. Ниже, рядом со старыми URL-адресами, использованными в предыдущих главах, представлен новый, дружественный SEO-формат. Он будет использоваться в этой главе. Новые, дружественные адреса будут содержать до трех частей, каждая из которых будет отделена символом слеша.

1. Все адреса начинаются одинаково, пути к файлу, но без расширения.php. Существующие имена файлов (за вычетом расширения) подходят для SEO-адресов, потому что они описывают назначение страницы.
2. Далее, если в старом URL-адресе была строка запроса, содержащая идентификатор данных, которые нужно получить из базы данных, после имени файла будет идти слеш и идентификатор запрашиваемых данных.
3. Адреса страниц статей и категорий будут также содержать еще один слеш, за которым идут ключевые слова, чтобы помочь поисковым системам индексировать эти страницы:

- на страницах статей используется название статьи;
- страницы категорий используют название категории.

СТАРЫЙ URL-АДРЕС	НОВЫЙ URL-АДРЕС
https://localhost/register.php	https://localhost/register
https://localhost/login.php	https://localhost/login
https://localhost/category.php?id=2	https://localhost/category/2/digital
https://localhost/category.php?id=4	https://localhost/category/4/photography
https://localhost/article.php?id=19	https://localhost/article/19/forecast
https://localhost/article.php?id=24	https://localhost/article/24/travel-guide
https://localhost/member.php?id=2	https://localhost/member/2
https://localhost/admin/article.php?id=24	https://localhost/admin/article/24

Поскольку новые URL-адреса больше не содержат пути к файлам, все такие запросы будут отправляться в один файл с названием `index.php`. Он возьмет SEO-адрес, разделит его по слешам, а затем сохранит каждую часть как отдельный элемент в массиве. Для общедоступных страниц части массива будут содержать:

- путь к файлу, который должен обработать запрос (без расширения);
- идентификатор данных в базе данных (если используется);
- SEO-заголовок (если есть).

Далее файл `index.php` возьмет значение из первого элемента массива и добавит к нему расширение.php, получив таким образом путь к файлу, такой же как в старых URL-адресах, а затем подключит этот файл, который и будет обрабатывать запрос, точно так же как если бы к этому файлу обратились напрямую. В таблице ниже показаны:

- путь (это часть URL-адреса после хоста);
- массив, созданный при разделении пути;
- описание частей массива.

ПУТЬ	ЧАСТИ	
`register`	`$parts[0] = 'register';`	Создается массив только с одним элементом. Файл `index.php` должен подключить страницу `register.php` для регистрации нового пользователя

`category/2/digital`	`$parts[0] = 'category';`
	`$parts[1] = '2';`
	`$parts[2] = 'digital';`
		Создается массив с тремя элементами, указывающими:
		● подключаемый файл — `category.php`
		● идентификатор категории — 2
		● ключевые слова для SEO — `digital`

`article/15/seascape`	`$parts[0] = 'article';`
	`$parts[1] = '15';`
	`$parts[2] = 'seascape';`
		Создается массив с тремя элементами, указывающими:
		● подключаемый файл — `article.php`
		● идентификатор публикации — 15
		● ключевые слова для SEO — `seascape`

Страницы администрирования не должны индексироваться поисковыми системами, и поэтому ключевые слова для SEO в их адресах не нужны. Когда URL страницы администрирования разделяется по слешам и превращается в массив, его элементы будут иметь немного другую структуру:

- признак страницы администрирования;
- имя файла, который должен обрабатывать запрос;
- идентификатор данных, с которыми он будет работать (если нужен).

После создания массива код в файле `index.php` проверяет, равно ли значение первого элемента слову `admin`. Если да, то это запрос к странице администрирования, и тогда путь к файлу создается другим способом, объединяя в одну строку:

- значение первого элемента;
- слеш;
- значение второго элементе;
- расширение файла.`php`.

ПУТЬ	ЧАСТИ	

`admin/article/15`	`$$parts[0] = 'admin';`
	`$parts[1] = 'article';`
	`$parts[2] = '15';`
		Состав этого массива говорит о том, что:
		● нужно подключить файл `admin/article.php`
		● идентификатор публикации равен 15

`admin/category/1`	`$parts[0] = 'admin';`
	`$parts[1] = 'category';`
	`$parts[2] = '1';`
		Состав этого массива говорит о том, что:
		● нужно подключить файл `admin/category.php`
		● идентификатор категории равен 1

Примечание: в URL-адресах некоторые символы являются управляющими, а также многие символы должны кодироваться в специальный формат. Чтобы избежать этого, из SEO-адресов желательно удалять нетекстовые символы, например: `/ ? : ; @ = & " < > # % { } | \ ^ ~ [ ] ` `.

Эти символы должны быть удалены из названия статьи или категории, прежде чем они будут использованы в качестве ключевых слов в SEO-адресе и это новое значение будет сохранено в базе данных.

ОБНОВЛЕННАЯ ФАЙЛОВАЯ СТРУКТУРА

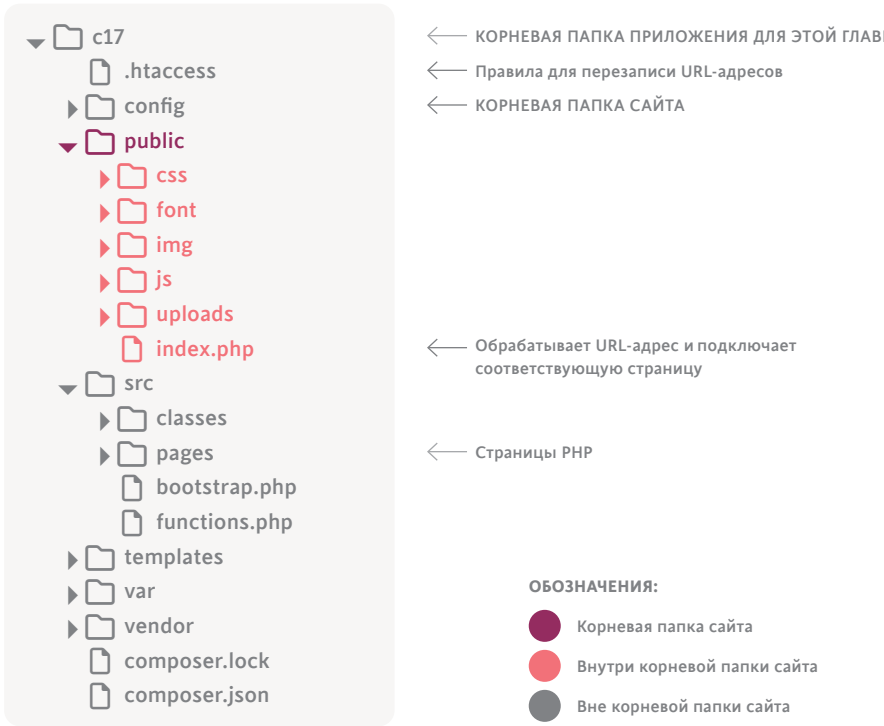
Файл `index.php` — единственный файл PHP, оставшийся в корневой папке сайта. Все остальные PHP-скрипты перемещены выше корня сайта, в каталог `src/pages`, поскольку прямой доступ к ним не требуется, а индексный файл может подключить их из любой папки.

Ниже вы можете увидеть новую файловую структуру, которая будет использоваться в этой главе.

Все файлы PHP перемещены из публичного каталога (представляющего собой корневой каталог сайта) в каталог `src/pages`.

В общей папке появились два новых файла:

- `.htaccess` содержит правила для направления всех запросов на страницу `index.php` (это файл конфигурации для Apache);
- `index.php` обрабатывает запрошенный URL и подключает соответствующую страницу PHP из папки `src/pages`.



ВНЕДРЕНИЕ SEO-АДРЕСОВ

При добавлении нового функционала необходимо предварительно спланировать, какие данные будут храниться в БД, какой код необходимо написать в PHP и каким образом обновится интерфейс. Ниже мы разберем все эти вопросы на примере добавления SEO-адресов.

КАКИЕ ДАННЫЕ ХРАНИТЬ И КАК ЭТО ДЕЛАТЬ

При создании или редактировании категории или публикации для них генерируется SEO-заголовок, а затем он сохраняется в базе данных.

Полученные заголовки сохраняются в столбце `seo_title` таблицы `article` и в столбце `seo_name` таблицы `category`. Ниже показана обновленная таблица `category`.

category

id	name	description	navigation	seo_name
1	Print	Inspiring graphic design	1	print
2	Digital	Powerful pixels	1	digital
3	Illustration	Hand-drawn visual storytelling	1	illustration

КАК РЕАЛИЗОВАТЬ НОВУЮ ФУНКЦИОНАЛЬНОСТЬ В КОДЕ

Поскольку SEO-адреса не используют путь к файлу PHP, потребуется настроить веб-сервер таким образом, чтобы все запросы к несуществующим файлам направлялись на `index.php`. Для этого понадобится настроить файл `.htaccess` в корневой папке сайта.

После этого страница `index.php` будет обрабатывать запрошенный URL в следующем порядке:

- получение идентификатора подлежащих обработке данных и сохранение его в переменной;
- подключение нужного PHP-файла для обработки запроса.

При добавлении или редактировании публикаций или категорий:

- новая функция под названием `create_seo_name()` создает SEO-заголовок;
- существующие методы классов `Article` и `Category` сохраняют его в БД.

Когда информация о публикации или категории читается из базы данных, SEO-заголовок передается в шаблон Twig для создания новых ссылок.

ОБНОВЛЕНИЕ ИНТЕРФЕЙСА

В шаблонах Twig все ссылки нуждаются в обновлении. Там, где есть SEO-заголовок, он добавляется к ссылке.

Ниже показан SEO-адрес страницы публикации. В нем используется идентификатор статьи, а затем ее SEO-заголовок.

```
<a href="article/{{ article.id }}/{{ article.seo_title }}">
```

ПЕРЕЗАПИСЬ URL-АДРЕСА

В веб-сервере Apache есть встроенный механизм перезаписи URL-адресов. С его помощью можно определить правила, согласно которым запрос на один URL должен быть преобразован в запрос на другой URL-адрес.

SEO-адреса используются только для страниц, отображаемых с помощью PHP. Они не используются для других файлов, запрашиваемых браузером, таких как файлы изображений, CSS, JavaScript или шрифтов, — их URL остаются прежними. Таким образом, механизм перезаписи URL-адресов веб-сервера Apache необходимо настроить следующим образом:

- отдавать файлы изображений, CSS, JavaScript и шрифтов, как и раньше, поскольку их URL-адреса не изменились;
- отправлять все остальные запросы в файл `index.php`.

Файл `index.php` будет разбирать запрошенный URL, а затем подключать нужный файл.

Файлы `.htaccess` управляют настройками веб-сервера Apache (включая механизм перезаписи URL-адресов). В папке `c17` находится файл `.htaccess` с настройками кодировки символов и размеров загружаемых файлов. В этот файл добавлены правила для управления механизмом перезаписи URL-адресов.

1. Сначала включается механизм перезаписи URL-адресов.
Далее инструкции для механизма перезаписи состоят из двух частей в отдельных строках:

- **Condition** (условие) указывает, когда должно выполняться правило;
- **Rule** (правило), которое состоит из двух частей: сначала идет шаблон для запросов, которые оно должно обрабатывать, а затем действие, которое необходимо совершить.

2. В условии сказано: следующее ниже правило должно быть выполнено, только если запрос относится к несуществующему на сервере файлу, что означает:

- все существующие файлы, такие как изображения, CSS, JavaScript и шрифты, не попадают под это условие и, следовательно, последующее правило не будет выполняться;
- поскольку SEO-адреса не ведут на существующие файлы, для них это правило должно будет выполниться.

3. Правило указывает, что любой запрос должен обрабатываться файлом `index.php`, расположенным в корневой папке сайта.

`c17/public/.htaccess` **PHP**

```
...
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule . public/index.php
```

Файл `.htaccess` может содержать несколько условий, за каждым из которых следуют правила для выполнения, если это условие выполнено.

Существуют и другие инструменты для перезаписи URL-адресов, но в этой книге не хватит места, чтобы охватить их все.

ИЗМЕНЕНИЕ URL-АДРЕСОВ

Чтобы избежать путаницы, ссылки на другие страницы сайта, а также на файлы изображений, CSS, JavaScript и шрифтов должны быть абсолютными, то есть начинаться от корня сайта. Обычно это просто слеш. Но поскольку код для каждой главы учебного сайта находится в отдельных папках, распложенных в реальном корне сайта, то, чтобы получить рабочие адреса, необходимо будет использовать константу.

В предыдущих главах ссылки на другие страницы сайта, а также ссылки на файлы изображений, CSS, JavaScript и шрифтов использовали относительные URL-адреса, то есть они достраивались от адреса текущей страницы PHP. Это работало, поскольку файловая система сайта была очень несложной, PHP-файлы располагались всего в двух папках.

Однако новые SEO-адреса заставляют страницы выглядеть так, как будто они находятся в разных папках. Например, эти два URL-адреса выглядят так, как будто они указывают на разные (но при этом несуществующие) папки:

```
/category/1/print
/article/22/polite-society-posters
```

Поэтому относительные ссылки не будут работать и необходимо использовать только абсолютные пути, которые начинаются от корня сайта, а не достраиваются браузером от текущей папки.

Обычно на сайтах используется просто слеш в качестве пути к корневой папке. Но поскольку в загружаемом коде версия кода сайта для каждой главы находится в своей папке, то начиная с 14 главы внутри каждой папки с кодом есть папка `public`, которая служит корнем сайта. Следовательно, чтобы получить абсолютный путь для локальных ссылок, в них должен присутствовать путь к папке `public`. Он будет добавляться в начале **всех** локальных ссылок на сайте.

Этот путь хранится в константе, определенной в файле `config.php`. Затем файл `bootstrap.php` добавляет его в глобальную переменную Twig под названием `doc_root`, и поэтому он может использоваться во всех шаблонах. Ниже вы можете увидеть его в начале ссылок на страницы (которые получили SEO-заголовки) и файлов изображений.

TWIG c17/templates/article-summaries.html

```
{% для статей в папке articles %}
<article class="summary">
<a href="{{ doc_root }}article/{{ article.id }}/{{ article.seo_title }}">
  {% if article.image_file %}
  <img src="{{ doc_root }}uploads/{{ article.image_file }}" alt="{{ article.image_alt }}">
  {% else %}
  <img src="{{ doc_root }}uploads/blank.png" alt="">
  {% endif %}

...

  {% if session.id == article.member_id %}
    <a href="{{ doc_root }}work/{{ article.id }}" class="btn btn-primary">Edit</a>
  {% endif %}</article>
{% endfor %}
```

ОБРАБОТКА ЗАПРОСОВ

Любой запрос к несуществующему адресу будет перенаправлен в файл `index.php`. Следовательно, если это будет существующий файл изображения, CSS, JavaScript или шрифта, то веб-сервер передаст его в браузер как есть, а вот SEO-адреса будут перенаправляться в `index.php`, который разобьет запрошенный URL в массив, а затем подключит PHP-файл, нужный для обработки этого запроса[81].

1. Файл `index.php` подключает файл `bootstrap.php`, чтобы не делать это на каждой подключаемой странице.

Затем запрошенный URL-адрес преобразуется в массив.

2. Запрошенный путь (часть после имени хоста) берется из суперглобального массива PHP `$_SERVER`, преобразуется в нижний регистр и сохраняется в переменной `$path`.
3. Часть пути до папки `public` для этой главы (которая будет корневой папкой сайта) удаляется из переменной. Для этой главы удаляется `phpbook/section_d/c17/public/` (этот шаг необходим потому, что веб-сервер обслуживает сразу несколько версий учебного сайта).
4. Функция PHP `explode()` используется для разделения пути по слешам и сохранения каждой части данных в виде отдельного элемента в массиве, называемом `$parts`.

Если запрашивается общедоступная страница, первый элемент указывает используемый файл. Если это страница администратора, то первый элемент содержит значение `admin`, а второй элемент указывает используемый файл. Например:

ПУТЬ	ЧАСТИ

`article/15/seascape`	`$parts[0] = 'article';`
	`$parts[1] = '15';`
	`$parts[2] = 'seascape';`

`admin/article/15`	`$parts[0] = 'admin';`
	`$parts[1] = 'article';`
	`$parts[2] = '15';`

5. Код проверяет, относится ли запрос к общедоступной странице или странице администратора, проверяя значение первого элемента в массиве `$parts`. Если первый элемент массива не равен строке `admin`, это означает, что запрос предназначен для страницы, которую могут просматривать все посетители.
6. Первый элемент массива `$parts` определяет имя файла, который должен быть подключен для обработки запроса. Например, если пользователь запрашивает страницу публикации, значение будет равно `article` (как показано в таблице слева внизу).

Когда пользователь запрашивает домашнюю страницу, то в `$path` будет пустая строка, и в результате `$parts` будет состоять из одного элемента, который содержит пустую строку. В этом случае в `$page` надо записать слово `index`.

Для этого мы используем еще одну сокращенную версию тернарного оператора, называемую Элвис-оператором.

Вместо того чтобы писать следующее:

```
$page = $parts[0] ? $parts[0] : 'index';
```

Используется это сокращение:

```
$page = $parts[0] ?: 'index';
```

Если `$parts[0]` содержит непустое значение, оно сохраняется в `$page`. В противном случае в `$page` запишется значение `index`.

7. Если второй элемент массива содержит значение, это будет идентификатор данных, с которыми работает страница. Если идентификатор присутствует во втором элементе массива, он сохраняется в переменной `$id`. В противном случае в `$id` запишется `null`.

PHP c17/public/index.php

```php
<?php
include '../src/bootstrap.php';                    // Файл настроек

$path  = mb_strtolower($_SERVER['REQUEST_URI']); // Перевод в нижний регистр
$path  = substr($path, strlen(DOC_ROOT));        // Удалить часть до DOC_ROOT
$parts = explode('/', $path);                    // Разделить на массив по /

if ($parts[0] != 'admin') {                      // Если это не страница администратора
    $page = $parts[0] ?: 'index';                // Название страницы или index
    $id   = $parts[1] ?? null;                   // Получить ID или null
} else {                                         // Если страница администратора
    $page = 'admin/' . ($parts[1] ?? '');        // Название страницы
    $id   = $parts[2] ?? null;                   // Получение ID
}
$id = filter_var($id, FILTER_VALIDATE_INT);       // Валидация ID

$php_page = APP_ROOT . '/src/pages/' . $page . '.php'; // Путь к файлу PHP
if (!file_exists($php_page)) {                     // Если файл не существует
    $php_page = APP_ROOT . '/src/pages/page-not-found.php'; // Показать страницу 404
}
include $php_page;                                 // Подключить PHP-файл
```

8. Если пользователь запрашивает страницу администратора, первый элемент в массиве `$parts` будет содержать слово `admin`. Второй элемент будет содержать название страницы.

9. В переменной `$page` начинает создаваться путь к файлу, обрабатывающему запрос. Он начинается со строки `admin/`, за которой следует название страницы.

Если URL-адрес закончился строкой `admin/` (в нем не указывалась страница), не было бы никаких `parts[1]`, поэтому оператор объединения с null заменяет значение пустой строкой[82].

10. Если в URL есть идентификатор, он сохраняется в переменной `$id`. В противном случае `$id` получит значение `null`.

11. Функция PHP `filter_var()` проверяет, является ли значение, хранящееся в переменной `$id`, целым числом. Это позволяет избежать данной проверки на каждой странице PHP, использующей идентификатор. К этому моменту на странице определено три переменные:

- `$parts` (массив частей URL-адреса);
- `$page`: название страницы (если это страница администратора, названию предшествует `admin/`);
- `$id`: идентификатор.

12. Путь к странице, обрабатывающей запрос, собирается в переменной `$php_page`. Она создается путем объединения:

- значения в `APP_ROOT` (создано в файле `bootstrap.php`);
- пути к страницам PHP `/src/pages/`;
- значения в переменной `$page`;
- расширения файла `.php`.

13. Функция PHP `file_exists()` проверяет, существует ли на сервере файл, путь к которому был создан в шаге 12.

14. Если нет, то значение, содержащееся в `$php_page`, заменяется путем к файлу `page-not-found.php` (который заканчивается командой `exit`, останавливающей выполнение любого другого кода).

15. Подключение файла, имя которого содержится в `$php_page`. Подключаемые файлы PHP выполняются точно так же, как если бы они выполнялись, будучи запрошенными напрямую через URL (подключаемый код работает так, как будто просто был скопирован и вставлен туда, где находится директива PHP include).

СОЗДАНИЕ SEO-ЗАГОЛОВКОВ

Когда создается или редактируется публикация или категория, вызывается новая функция, `create_seo_name()`, которая используется для создания SEO-заголовка размещенной работы или категории.

Поскольку URL не может содержать пробелов или определенных символов (например, `/ ? = & #`), в файл `functions.php` была добавлена функция `create_seo_name()`для создания дружественных SEO-заголовков из названий публикаций и категорий. Получившийся заголовок содержит только буквы A–z, цифры 0–9 и тире.

Также делается попытка использовать функцию PHP `transliterator_transliterate()`, чтобы заменить не входящие в кодировку ASCII символы ближайшим эквивалентом ASCII (на основе фонетического сходства). Например, Über будет заменен на Uber, а École — на Ecole. Для работы транслитерации необходимо установить расширение PHP (`php-intl`), поэтому в коде используется метод PHP `function_exists()`, чтобы проверить, доступна ли эта функция перед ее вызовом. Функция вызывается только в том случае, если она доступна. Для получения дополнительной информации см. http://notes.re/php/transliteration.

1. Функция `create_seo_name()` принимает строку в качестве параметра и возвращает оптимизированную для SEO-адреса версию этой строки.
2. Текст преобразуется в нижний регистр.
3. Любые пробелы удаляются из начала и конца.
4. PHP-функция `function_exists()` проверяет, доступна ли функция `transliterator_transliterate()`. Если доступна, то она вызывается, чтобы попытаться заменить символы, отличные от ASCII, эквивалентами ASCII.
5. Функция `preg_replace()` заменят пробелы на тире.
6. Затем удаляет все, кроме тире, A–z или 0–9.
7. Возвращается оптимизированное название работы или категории.

c17/src/functions.php — PHP

```php
function create_seo_name(string $text): string
{
    $text = strtolower($text);                  // Преобразование текста в нижний регистр
    $text = trim($text);                        // Удаление пробелов из начала/конца
    if (function_exists('transliterator_transliterate')) { // Если есть расширение
        $text = transliterator_transliterate('Latin-ASCII', $text); // Транслитерация
    }
    $text = preg_replace('/ /', '-', $text);          // Заменить пробелы на тире
    $text = preg_replace('/[^-A-z0-9]+/', '', $text); // Удалить все, кроме тире, A-z, 0-9
    return $text;                                     // SEO-заголовок
}
```

СОХРАНЕНИЕ SEO-ЗАГОЛОВКА

Функция `create_seo_name()` вызывается при создании или редактировании публикации или категории. Затем полученный заголовок передается в метод, который обновляет базу данных.

1. При создании или редактировании категории в файле `category.php` (теперь в `src/pages/admin`) вызывается функция `create_seo_name()` и в нее передается название категории в качестве аргумента.

Возвращаемый SEO-заголовок записывается в массив данных категории. Затем этот массив передается методам `create()` или `update()`объекта `Category`.

PHP c17/src/pages/admin/category.php

```
$category['name']        = $_POST['name'];                        // Получение заголовка
$category['description'] = $_POST['description'];                 // Получение описания
$category['navigation']  = (isset($_POST['navigation'])) ? 1 : 0; // Получение навигации
(1) $category['seo_name']    = create_seo_name($category['name']);    // SEO-заголовок
```

2. При вызове методов `create()` или `update()` объекта `Category` в массиве `$category` добавляется новый элемент с SEO-заголовком.

3. Запросы `INSERT` и `UPDATE` обновляются, чтобы также сохранять и SEO-заголовок.

Процесс для статей такой же:

- элемент с ключом `seo_title` добавляется в массив `$article` в файлах `admin/article.php` и `work.php`;
- SEO-заголовок сохраняется в базе данных при вызове методов `create()` или `update()` класса `Article`.

PHP c17/src/classes/CMS/Category.php

```
(2) public function create(array $category): bool
    {
        try {                                              // На случай дубликата
(3)         $sql = "INSERT INTO category (name, description, navigation, seo_name)
(4)                 VALUES (:name, :description, :navigation, :seo_name);"; // SQL
      ...
```

Примечание. Столбцы `seo_name` и `seo_title` в базе данных имеют ограничения уникальности (как и столбцы `name` и `title`), чтобы обеспечить уникальность SEO-заголовков.

Код для сохранения SEO-заголовков также содержится и в загружаемом коде для главы 16, чтобы можно было использовать один и тот же дамп базы данных.

ОТОБРАЖЕНИЕ СТРАНИЦ С SEO-ЗАГОЛОВКАМИ

На страницах работ и категорий в URL-адресах используются SEO-заголовки. Код на этих страницах проверяет правильность SEO-заголовка перед отображением страницы.

Сначала необходимо обновить методы `get()` и `getAll()` классов `Article` и `Category`, чтобы они также возвращали SEO-заголовки из базы данных.

1. В SQL-запросе в методе `get()` класса `Category` запрашиваются данные из столбца `seo_name`.

Далее два блока кода удаляются из **всех** PHP-файлов, которые были перемещены в файл `src/pages`, потому что эти задачи теперь выполняются в файле `index.php`:

- подключение файла `bootstrap.php`;
- получение идентификатора из строки запроса и его валидация.

Кроме этого, в файлах `article.php` и `category.php` потребуется выполнить одну дополнительную операцию — проверить правильность SEO-заголовка в URL, поскольку несколько ссылок могут указывать на одну и ту же статью с разными названиями, например:

```
✓ http://eg.link/article/24/travel-guide
× http://eg.link/article/24/japan-guide
× http://eg.link/article/24/guide-book
```

Каждый из этих адресов содержит информацию, необходимую сайту для получения данных для страницы (у него есть название страницы для подключения и идентификатор). Однако поисковые системы могут подумать, что это разные страницы с дублирующимся контентом, и за это сайт может быть наказан. Такая ситуация может возникнуть, если на другом сайте неправильно указана ссылка на страницу или если название статьи изменилось после создания ссылки.

Поэтому код на страницах `article.php` и `category.php` проверяет, совпадает ли SEO-заголовок из URL-адреса, разбитого на части в массив `$parts` в файле `index.php`, с SEO-заголовком, сохраненным в базе данных. Если нет, то пользователь перенаправляется на правильный URL-адрес.

2. Конструкция `if` проверяет, соответствует ли сохраненное в БД значение SEO-заголовка значению, переданному в URL (т. е. третьему элементу массива `$parts`, созданного в файле `index.php`). Оба значения преобразуются в нижний регистр перед сравнением.
3. Если они не совпадают, функция `redirect()` отправляет посетителя на ту же страницу, используя правильный SEO-адрес.

И последнее, каждая ссылка в каждом шаблоне должна быть изменена, чтобы поддерживать SEO-адреса.

4. Каждая ссылка должна состоять из:

- пути к корневой папке сайта (обычно это /, но поскольку для загружаемого кода используется несколько версий сайта, это путь к общей папке для этой главы);
- имени PHP-файла без расширения.php;
- идентификатор статьи или категории;
- SEO-заголовка, если ссылка ведет на публикацию или категорию.

5. Путь к файлам изображений также должен включать путь к корневой папке сайта.

PHP c17/src/classes/CMS/Category.php

```php
public function get(int $id)
{
    $sql = "SELECT id, name, description, navigation, seo_name
              FROM category
             WHERE id = :id;";                      // SQL для получения одной категории
    return $this->db->runSQL($sql, [$id])->fetch(); // Возврат данных категории
}
```

PHP c17/src/pages/category.php

```php
<?php
declare(strict_types = 1);                              // Используем строгие типы

if (!$id) {                                        // Если нет валидного идентификатора
    include APP_ROOT . '/src/pages/page-not-found.php';  // Страница 404
}

$category = $cms->getCategory()->get($id);              // Получить данные о категории
if (!$category) {                                       // Если нет категории
    include APP_ROOT . '/src/pages/page-not-found.php';  // Страница 404
}

if (mb_strtolower($parts[2]) != mb_strtolower($category['seo_name'])) {  // Неверный
                                                                         // SEO-заголовок
    redirect('category/' . $id . '/' . $category['seo_name'], [], 301); // Перенаправление
}

$data['navigation'] = $cms->getCategory()->getAll();       // Навигация по категориям
$data['category']   = $category;                           // Текущая категория
$data['articles']   = $cms->getArticle()->getAll(true, $id); // Получить работы
$data['section']    = $category['id'];  // Идентификатор категории для навигации
```

TWIG c17/templates/article-summaries.html

```twig
<a href="{{ doc_root }}article/{{ article.id }}/{{ article.seo_title }}">
  {% if article.image_file %}
    <img src="{{ doc_root }}uploads/{{ article.image_file }}"
      alt="{{ article.image_alt }}">
  {% else %}
    <img src="{{ doc_root }}uploads/blank.png" alt="">
  {% endif %}
  ...
```

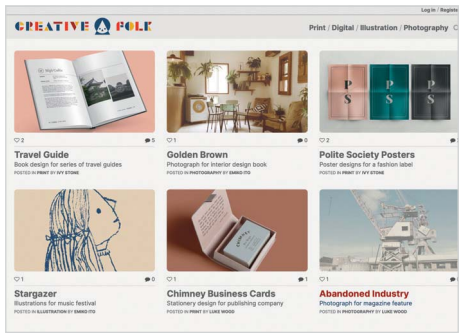
ПЛАНИРОВАНИЕ НОВОЙ ФУНКЦИОНАЛЬНОСТИ

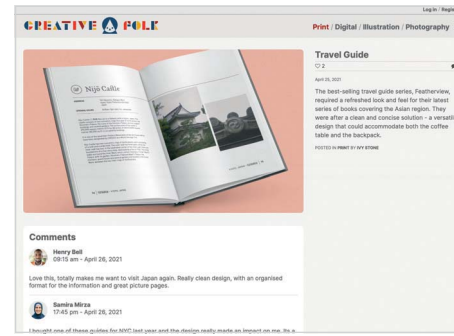
При добавлении новой функциональности начните с определения того, какие конкретно действия она позволит совершать вашим пользователям. Это упростит написание кода для реализации этой функциональности.

Прежде чем вы начнете создавать новый функционал для сайта, вы должны четко определить, какие именно действия смогут совершать пользователи. Это поможет разбить задачу на части. Например, в этой главе на всех страницах, отображающих краткие описания работ, начинает показываться количество лайков и комментариев к каждой работе.

На странице публикации количество лайков и комментариев отображается под заголовком, а сами комментарии выводятся под изображением. Кроме того, если пользователь авторизовался, то:

- иконка с сердечком становится активной и позволяет поставить оценку «Нравится» или отозвать ее;
- отображается форма для добавления комментария к публикации (в противном случае пользователю будет предложено войти в систему для комментирования).





Как только вы решите, что именно новые функции позволяют делать пользователям, вашими дальнейшими действиями будут:

1) определить, какие данные будут храниться в БД;
2) написать или обновить методы, использующиеся для получения данных из БД или сохранения данных в ней;
3) создать или обновить скрипты PHP, чтобы они могли выполнять новые задачи, когда это необходимо, и убедитесь, что в шаблоны передаются необходимые им данные;
4) создавать или обновлять шаблоны, чтобы посетители могли взаимодействовать с этими новыми функциями.

ОПРЕДЕЛИТЕ, КАКИЕ ДАННЫЕ ХРАНИТЬ И КАК ИХ ХРАНИТЬ

Сначала решите, какие данные должны будут видеть пользователи и должна ли БД хранить какие-либо новые данные.

Чтобы показывать лайки, в базе данных должно храниться следующее:

- пользователь, которому понравилась статья (уже в таблице `member`);
- работа, которая ему нравится (уже в таблице `article`).

Чтобы показывать комментарии, база данных должна хранить следующие данные:

- пользователь, оставивший комментарий (в таблице `member`);
- комментарий (новые данные);
- дата и время, когда был сделан комментарий (новые данные).

Затем решите, как сохранить новые данные в базе данных.

1. Если это дополнительные данные о чем-то, что уже представлено существующей таблицей (например, пользователь или его работа), добавьте их в соответствующую таблицу.

2. Если это совершенно новая концепция или объект, создайте новую таблицу для представления этих данных. Например, комментарии будут сохранены в новой таблице `comment`.

3. Если же новые данные описывают связь между понятиями, которые уже хранятся в базе данных, используйте таблицу **связей** между таблицами. При реализации лайков в базе данных уже есть данные о пользователях и публикациях, поэтому в таблице ссылок будут храниться идентификаторы пользователей и идентификаторы понравившихся им работ.

СОЗДАВАЙТЕ КЛАССЫ И МЕТОДЫ ДЛЯ ПОЛУЧЕНИЯ И СОХРАНЕНИЯ ДАННЫХ

Когда вы уже знаете, какие данные необходимо сохранить в базе данных, напишите классы и методы, необходимые для получения, создания, обновления и удаления этих данных.

Два новых класса будут реализовывать лайки и комментарии:

- класс `Like` получает, добавляет и удаляет лайки;
- класс `Comment` получает и добавляет комментарии пользователей.

Существующий класс `Article` также обновляется так, что методы `get()` и `getAll()` возвращают общее количество имеющихся лайков и комментариев к каждой статье.

КЛАСС LIKE

МЕТОД	ОПИСАНИЕ
`get()`	Проверяет, понравилась ли пользователю работа
`create()`	Добавляет отметку лайк в базу данных
`delete()`	Удаляет отметку лайк из базы данных

КЛАСС COMMENT

МЕТОД	ОПИСАНИЕ
`getAll()`	Получает комментарии к статье
`create()`	Добавляет комментарий в базу данных

ОБНОВЛЕНИЕ СКРИПТОВ PHP

Затем решите, нужно ли вам обновить существующие файлы PHP или создать новые для реализации этих функций.

Например, новый файл будет использоваться для сохранения в БД лайков.

Кроме того, существующая страница `article.php` проверит:

- вошел ли пользователь в систему, и если да, то понравилась ли ему статья;
- был ли оставлен комментарий (если да, комментарий проходит валидацию и затем может быть сохранен в базе данных).

ОБНОВЛЕНИЕ ФАЙЛОВ ШАБЛОНОВ

И наконец, необходимо обновить шаблоны, генерирующие выводимый в браузер HTML.

Шаблон `article-summaries.html` показывает, сколько лайков и комментариев есть к каждой статье.

Шаблон `article.html` показывает общее количество лайков и комментариев и комментарии целиком.

Если пользователь вошел в систему, шаблон добавит ссылку для того, чтобы поставить отметку «нравится» или отозвать ее, а также покажет форму, позволяющую ему оставить комментарий.

ХРАНЕНИЕ КОММЕНТАРИЕВ

Новая таблица под названием comment содержит каждый комментарий ко всем публикациям.

Столбцы в новой таблице comment (показанной ниже):

- id создается с помощью механизма автоинкремента MySQL;
- comment — это комментарий к статье;
- posted — это дата и время сохранения комментария (их БД добавляет в таблицу сама);
- article_id — это идентификатор публикации, для которой он предназначен;
- member_id — идентификатор пользователя, написавшего комментарий.

Столбцы article_id и member_id используют ограничения внешнего ключа, чтобы убедиться, что идентификаторы публикации и пользователя реально существуют.

Примечание. Всегда создавайте резервную копию базы данных перед ее изменением. Это важно, потому что добавление новой функции может случайно перезаписать или удалить данные, которые не должны быть изменены.

comment

id	comment	posted	article_id	member_id
1	Love this, totally makes me want to...	2019-03-14 17:45:13	24	1
2	I bought one of these guides for NYC...	2019-03-14 17:45:15	24	6
3	Another great piece of work Ivy,...	2019-03-14 17:53:52	3	4

Справа вы можете увидеть два SQL-запроса, использующиеся для подсчета общего количества комментариев и лайков к статье. Эти два запроса будут использоваться на странице публикации, а также при отображении списка карточек публикаций. Новая таблица для хранения лайков показана на правой странице.

ВСЕГО КОММЕНТАРИЕВ:

```
SELECT COUNT(id)
  FROM comments
 WHERE comments.article_id = article.id
```

ВСЕГО ЛАЙКОВ:

```
SELECT COUNT(article_id)
  FROM likes
 WHERE likes.article_id = article.id
```

ХРАНЕНИЕ ЛАЙКОВ

В базе данных уже есть таблицы, представляющие публикации и пользователей. Новая таблица под названием `likes` будет записывать для каждого участника каждую понравившуюся ему статью.

Чтобы это реализовать, в базе данных нужно сохранить только связь между пользователем и понравившейся ему работой (поскольку в БД уже есть данные о пользователях и работах).

Взаимосвязь описывается с помощью **таблицы связей** (она связывает данные в двух таблицах). Ее столбцами будут:

- `article_id` — идентификатор понравившейся пользователю статьи;
- `member_id` — идентификатор пользователя, которому понравилась статья.

Эта таблица связей называется `likes` и показана ниже. В ней используется множественное число likes, а не like, потому что в SQL есть ключевое слово LIKE.

Столбцы `article_id` и `member_id` используют ограничения внешнего ключа, чтобы убедиться, что идентификаторы публикации и пользователя реально существуют.

Поставить отметку можно только один раз, поэтому в таблицу добавляется **составной первичный ключ**. Он не позволяет сохранить ту же комбинацию значений `article_id` и `member_id` повторно. Это похоже на то, как таблица пользователей не позволяет двум пользователям использовать один и тот же адрес электронной почты.

См. http://notes.re/php/composite-key, чтобы узнать, как создать составной первичный ключ.

likes

article_id	member_id
1	1
2	1
1	2

member

id	forename	surname	email	password	joined	picture
1	Ivy	Stone	ivy@eg.link	0086...	2019-01-01...	ivy.jpg
2	Luke	Wood	luke@eg.link	DFCD...	2019-01-02...	*NULL*
3	Emiko	Ito	emi@eg.link	G4A8...	2019-01-02...	emi.jpg

article

id	title	summary	content	created	category_id	member_id	image_id	published	seo_title
1	PS Poster	Poster	Parts...	2019	2	2	1	1	ps-poster
2	Systemic	Leaflet	Design...	2019	2	1	2	1	systemic
3	AQ Website	New site	A new...	2019	1	1	3	1	aq-web

ОТОБРАЖЕНИЕ КАРТОЧЕК ПУБЛИКАЦИЙ С КОЛИЧЕСТВОМ ЛАЙКОВ И КОММЕНТАРИЕВ

SQL-запросы, получающие данные о публикациях, необходимо обновить, чтобы они также возвращали количество лайков и комментариев для каждой работы. Чтобы вычислить эти итоговые значения, в основной запрос добавляются два подзапроса.

Метод `getAll()` класса `Article` использует SQL-запрос для получения списка публикаций. Этот метод используется на домашней странице, странице категории, странице пользователя и странице поиска.

Чтобы получить общее количество лайков и комментариев для каждой из статей, в методе `getAll()`к существующему SQL-запросу добавляются два **подзапроса**.

Подзапросы — это дополнительные запросы, выполняющиеся в рамках другого запроса. Изменения в исходном запросе выделены на странице справа.

Каждый раз, когда основной запрос выбирает публикацию для добавления в результирующий набор, выполняются два подзапроса:

- первый подсчитывает количество лайков для этой работы;
- второй подсчитывает количество ее комментариев.

Подзапросы используют тот же синтаксис, что и другие SQL-запросы, но они заключены в круглые скобки.

Каждый из них возвращает одно значение, а после круглых скобок псевдоним указывает имя столбца в результирующем наборе, который будет содержать результат этого запроса. Те же два подзапроса были добавлены к методу `get()` класса `Article`. Вы можете увидеть их в загружаемом коде.

Примечание. Чтобы этот код поместился на странице, массив `$arguments` создается с использованием сокращенной формы записи, позволяющей присвоить двум ключам одинаковое значение в одном операторе.

1. Первый подзапрос получает количество лайков из таблицы лайков. Поскольку эти подзапросы добавляют новые поля к возвращаемой запросом информации, они помещаются после перечисления остальных полей в команде `SELECT`.

После закрывающей круглой скобки ставится псевдоним, который дает столбцу с количеством лайков в результирующем наборе имя `likes`.

2. Второй подзапрос получает количество комментариев к статье. Как и первый подзапрос, он также заключен в круглые скобки.
3. Псевдоним дает столбцу с количеством комментариев имя `comments`.
4. Шаблон `article-summaries.html`, используемый для отображения карточек публикаций на домашней странице, а также на страницах категорий, участников и поиска, обновляется для отображения количества лайков и комментариев. Эти два новых элемента данных отображаются перед заголовком статьи.

Хотя карточки статей выводятся на множестве страниц сайта, таких как страницы категорий, пользователей, поиска и домашней страницы, сами PHP-файлы в `src/pages` не нуждаются в обновлении, поскольку все они используют один и тот же метод класса `Article`, запрос в котором мы только что обновили. Затем полученные данные в виде массива передаются в шаблоны Twig.

PHP c17/src/classes/CMS/Article.php

```
public function getAll($published = true, $category = null, $member = null,
                       $limit = 1000): array
    {
    $arguments['category'] = $arguments['category1'] = $category;  // Id категории
    $arguments['member']   = $arguments['member1']   = $member;    // Id автора
    $arguments['limit']    = $limit;                  // Максимальное количество работ

    $sql = "SELECT a.id, a.title, a.summary, a.created, a.category_id,
              a.member_id, a.published, a.seo_title,
              c.name     AS category,
              c.seo_name AS seo_category,
              m.forename, m.surname,
              CONCAT(m.forename, ' ', m.surname) AS author,
              i.file     AS image_file,
              i.alt      AS image_alt,
              (SELECT COUNT(article_id)
                 FROM likes
                WHERE likes.article_id = a.id) AS likes,
              (SELECT COUNT(article_id)
                 FROM comment
                WHERE comment.article_id = a.id) AS comments

              FROM article      AS a
              JOIN category     AS c   ON a.category_id = c.id
              JOIN member       AS m   ON a.member_id   = m.id
              LEFT JOIN image   AS i   ON a.image_id    = i.id

            WHERE (a.category_id = :category OR :category1 is null)
              AND (a.member_id   = :member   OR :member1   is null) "; // SQL
    if ($published == true) {                 // Если параметр $published равен true
        $sql .= "AND a.published = 1 ";       // Только разрешенные к показу работы
    }
    $sql .= "ORDER BY a.id DESC
             LIMIT :limit;";                   // Остальные элементы запроса

        return $this->db->runSQL($sql, $arguments)->fetchAll(); // Результат запроса
}
```

TWIG c17/templates/article-summaries.html

```
<div class="social">
<div class="like-count"><span class="icon-heart-empty"></span> {{ article.likes }}</div>
<div class="comment-count"><span class="icon-comment"></span> {{ article.comments }}</div>
</div>
<h2>{{ article.title }}</h2>
```

ДОБАВЛЕНИЕ И УДАЛЕНИЕ ЛАЙКОВ

Если пользователь вошел в систему, значок сердечка на странице статьи становится ссылкой, позволяющей поставить или отозвать лайк. Код в файле по ссылке проверяет, ставил ли пользователь отметку лайк ранее. Если нет, то лайк добавляется. Если да, то отметка удаляется. Для этого в коде вызываются методы класса `Like`.

Если пользователь вошел в систему, значок сердечка на странице статьи помещается внутри ссылки[83], подобной этой:

```
<a href="/like/24"> ... </a>
```

Путь в URL ссылки начинается со слова `like`, затем указан идентификатор статьи. При переходе по этой ссылке скрипт `index.php` подключит новый файл под названием `like.php`. В этом файле:

1) условие в конструкции `if` проверяет:

- является ли идентификатор публикации пустым, или
- что пользователь не вошел в систему (в этом случае свойство `id` объекта `Session` будет иметь значение 0);

2) если любое из условий верно, посетитель не может выполнять данное действие и отправляется на страницу `Page not found`. В противном случае:

3) метод `get()` нового объекта `Like` проверяет, ставил ли этот пользователь лайк этой публикации ранее. Методу требуются идентификаторы работы и пользователя, которые передаются в него в виде индексированного массива. Результат (0 для «нет» или 1 для «да») записывается в переменную `$liked`;

4) если этот пользователь уже ставил статье лайк, вызывается метод `delete()` объекта `Like` для удаления записи из таблицы `likes` в базе данных;

5) в противном случае он еще не ставил отметку лайк, и метод `create()` класса `Like` добавляет лайк;

6) затем пользователь возвращается на страницу публикации.

Новый класс `Like` обновляет информацию в БД, если пользователь ставит или отзывает у публикации отметку «нравится». У этого класса есть три метода:

- `get()` проверяет, нравится ли пользователю работа;
- `create()` добавляет лайк к базе данных;
- `delete()` удаляет лайк из базы данных.

Каждому методу требуется два фрагмента данных, передающихся методам в виде индексированного массива[84]:

- идентификатор публикации;
- идентификатор пользователя;

7) класс `Like` аналогичен классам `Article`, `Category` и `Member`. Экземпляр этого класса создается с помощью нового метода `getLike()` объекта `CMS` и сохраняет объект класса `Database` в свойстве под названием `$db`. Метод `get()` использует SQL-функцию `COUNT()` чтобы проверить, сколько строк таблицы `likes` содержат указанные идентификаторы публикации и пользователя. Затем это число возвращается из метода.

Поскольку в этой таблице используется составной первичный ключ, метод всегда будет возвращать только 1 или 0 (если лайк уже стоит или отсутствует соответственно);

8) метод `create()` добавляет новую строку в таблицу `likes` с идентификаторами публикации и пользователя;

9) метод `delete()` удаляет из таблицы `likes` строку с переданными в него идентификаторами публикации и пользователя.

PHP c17/src/pages/like.php

```php
<?php
declare(strict_types = 1);                                  // Строгая типизация

if (!$id or $session->id == 0) {                            // Если идентификатор пустой
    include APP_ROOT . '/src/pages/page-not-found.php';     // Страница 404
}

$liked = $cms->getLike()->get([$id, $session->id]);         // Есть ли лайк
if ($liked) {                                               // Если да
    $cms->getLike()->delete([$id, $session->id]);           // Удаление
} else {                                                    // В противном случае
    $cms->getLike()->create([$id, $session->id]);           // Добавление
}
redirect('article/' . $id . '/' . $parts[2] . '/');         // Редирект на публикацию
```

PHP c17/src/classes/CMS/Like.php

```php
...
public function get(array $like): bool
{
    $sql = "SELECT COUNT(*)
              FROM likes
             WHERE article_id = :id
               AND member_id = :member_id;";                // SQL
    return $this->db->runSQL($sql, $like)->fetchColumn();   // Вернуть 1 или 0
}

public function create(array $like): bool
{
    $sql = "INSERT INTO likes (article_id, member_id)
            VALUES (:article_id, :member_id);";             // SQL
    $this->db->runSQL($sql, $like);                         // Выполнение SQL
    return true;                                            // Возврат значения true
}

public function delete(array $like): bool
{
    $sql = "DELETE FROM likes
             WHERE article_id = :article_id
               AND member_id = :member_id;";                // SQL
    $this->db->runSQL($sql, $like);                         // Выполнение SQL
    return true;                                            // Возврат значения true
}
```

ДОБАВЛЕНИЕ КОММЕНТАРИЕВ К ПУБЛИКАЦИЯМ

Если пользователь вошел в систему, форма для отправки комментария отображается под картинкой (и существующими комментариями, если они есть).

Новый класс `Comment` содержит методы для получения комментариев из таблицы `comment` базы данных или добавления их в нее:

- `getAll()`: получает все комментарии к статье;
- `create()`: добавляет комментарий в таблицу `comment`.

1. Метод `getAll()` получает все комментарии к статье, а также имя и фотографию пользователя для каждого комментария.

Метод принимает только один аргумент: идентификатор публикации, для которой он должен получать комментарии.
Чтобы получить имя и фотографию участника, оставившего комментарий, SQL использует объединение таблиц `comment` и `member`, используя:

- столбец `member_id` из таблицы `comment`;
- столбец `id` таблицы `member`.

2. Метод `create()` добавляет новый комментарий в таблицу `comment` в базе данных. Ему нужны три фрагмента данных, которые передаются методу в виде индексированного массива:

- комментарий;
- идентификатор публикации;
- идентификатор пользователя, сделавшего комментарий.

Механизм автоинкремента базы данных создает идентификатор в первом столбце таблицы `comment`. База данных также добавляет дату и время сохранения комментария в столбец `posted` таблицы.

Код на странице `article.php` дополняется, чтобы отображать существующие и сохранять новые комментарии к публикации.

3. Условие в конструкции `if` проверяет, был ли запрос отправлен запросом `POST`, что означает, что форма с комментарием была отправлена.
4. Если это так, то текст комментария присваивается переменной.
5. Для удаления нежелательной разметки из комментария создается новый объект `HTMLPurifier`.
6. Параметры конфигурации настроены таким образом, чтобы разрешить использование элементов `<br>`, `<b>`, `<i>` и `<a>`.
7. Метод `purify()` удаляет все остальные HTML-теги из комментария.
8. Если комментарий содержит от 1 до 2000 символов, в переменной `$error` сохраняется пустая строка. В противном случае в `$error` сохраняется сообщение об ошибке.
9. Если ошибки нет, комментарий, идентификатор публикации и идентификатор пользователя сохраняются в массиве с именем `$arguments`.
10. Затем вызывается метод `create()` объекта `Comment` для сохранения комментария в базе данных.
11. Страница публикации перезагружается, чтобы показать добавленный комментарий.
12. Метод `getAll()` объекта `Comment` получает все комментарии к этой статье. Они добавляются в массив `$data` для использования в шаблоне Twig.

PHP c17/src/classes/CMS/Comment.php

```php
public function getAll(int $id): array
{
    $sql = "SELECT c.id, c.comment, c.posted,
           CONCAT(m.forename, ' ', m.surname) AS author, m.picture
             FROM comment AS c
             JOIN member  AS m ON c.member_id = m.id
            WHERE c.article_id = :id;";                          // SQL
    return $this->db->runSQL($sql, ['id' => $id])->fetchAll();  // Выполнение SQL
}

public function create(array $comment): bool
{
    $sql = "INSERT INTO comment (comment, article_id, member_id)
            VALUES (:comment, :article_id, :member_id);";       // SQL
    $this->db->runSQL($sql, $comment);                          // Выполнение запроса
    return true;
}
```

PHP c17/src/pages/article.php

```php
<?php ...
if ($_SERVER['REQUEST_METHOD'] == 'POST') {                     // Если форма отправлена
    $comment  = $_POST['comment'];                              // Получение комментария
    $purifier = new HTMLPurifier();                             // Создание HTMLPurifier
    $purifier->config->set('HTML.Allowed', 'br,b,i,a[href]');   // Разрешенные теги
    $comment  = $purifier->purify($comment);                    // Очистка комментария

    $error    = Validate::isText($comment, 1, 2000)
        ? '' : 'Your comment must be between 1 and 2000 characters.
                It can contain <b>, <i>, <a>, and <br> tags.'; // Валидация комментария
    if ($error === '') {                                        // Если ошибок нет, сохранение
        $arguments = [$comment, $article['id'], $cms->getSession()->id,]; // Аргументы
        $cms->getComment()->create($arguments);                 // Создание комментария
        redirect($path);                                        // Перезагрузка страницы
    }
}

$data['navigation']  = $cms->getCategory()->getAll();         // Получение категорий
$data['article']     = $article;                              // Данные публикации
$data['section']     = $article['category_id'];               // Текущая категория
$data['comments']    = $cms->getComment()->getAll($id);       // Получение комментариев
if ($cms->getSession()->id > 0) {                  // Если пользователь вошел в систему
    $data['liked']    = $cms->getLike()->get([$id, $cms->getSession()->id]);
                                                   // Ставил ли пользователь лайк
    $data['error']    = $error ?? null;            // Сообщение об ошибке для шаблона
}
```

ОБНОВЛЕНИЕ ШАБЛОНА СТРАНИЦЫ ПУБЛИКАЦИИ

Шаблон `article.html` нуждается в обновлении, чтобы отображать новые данные, получаемые страницей `article.php`. Если пользователь вошел в систему, в нем также отображается ссылка для отметки «нравится» и форма для комментариев.

Справа вы можете увидеть изменения в шаблоне `article.html`, используемом для отображения публикаций.

Сначала речь идет о возможности поставить лайк статье.

1. Условие в блоке `if` проверяет, содержит ли свойство `id` объекта сессии значение 0, что указывает на то, что посетитель не вошел в систему.
2. Если посетитель не вошел в систему, для значка пустого сердечка делается ссылка на страницу входа.
3. В противном случае посетитель вошел в систему и для сердечка делается ссылка на страницу добавления лайков (она содержит идентификатор публикации). При нажатии на эту ссылку страница лайков либо добавляет, либо удаляет лайк этого пользователя.
4. Условие в еще одном блоке `if` проверяет значение переменной `liked`, чтобы узнать, ставил ли уже посетитель лайк этой статье.
5. Если да, появится заполненный значок сердечка.
6. Если нет, то отображается контур сердечка.
7. Закрывается блок `if` с шага 4.
8. Закрывается блок `if` с шага 1.
9. Рядом со значком сердечка отображается общее количество поставленных статье лайков. Далее отображаются комментарии.
10. Отображается количество комментариев к статье.
11. Создается цикл для вывода комментариев, лежащих в массиве `comments` (создан в шаге 12 предыдущей страницы). Если комментарии есть, то внутри цикла:
12. Отображается изображение профиля человека, написавшего комментарий, с его именем в качестве текста описания.
13. Имя пользователя отображается рядом с его фотографией.
14. Выводятся дата и время сохранения комментария. Фильтр Twig `date()` используется для форматирования значений таким же образом, как и другие даты на сайте.
15. Отображение комментария. Фильтр Twig `raw` предотвращает экранирование разметки, поскольку комментарий был запущен через `HTMLPurifier` перед сохранением.
16. Вышеприведенные операции повторяются для каждого комментария в массиве, и тег Twig `{% endfor%}` закрывает цикл.
17. Если пользователь вошел в систему, свойство `id` объекта `session` будет иметь значение больше 0.
18. Если посетитель вошел в систему, ему отображается форма, позволяющая отправить новый комментарий.
19. В противном случае посетителю будет показано сообщение, что ему необходимо войти в систему, чтобы оставить комментарий.

TWIG c17/templates/article.html

```
...
<div class="social">
  <div class="like-count">
    {% if session.id == 0 %}
      <a href="{{ doc_root }}login/"><span class="icon-heart-empty"></span></a>
    {% else %}
      <a href="{{ doc_root }}like/{{ article.id }}">
      {% if liked %}
        <span class="icon-heart"></span></a>
      {% else %}
        <span class="icon-heart-empty"></span>
      {% endif %}
      </a>
    {% endif %}
    {{ article.likes }}
  </div>
  <div class="comment-count">
    <span class="icon-comment"></span> {{ article.comments }}
  </div>
</div>

...

<section class="comments">
  <h2>Comments</h2>
  {% for comment in comments %}
    <div class="comment">
      <img src="{{ doc_root }}uploads/{{ comment.picture }}" alt="{{ comment.author }}" />
      <b>{{ comment.author }}</b><br>
      {{ comment.posted|date('H:i a - F d, Y') }}<br>
      <p>{{ comment.comment|raw }}</p>
    </div>
  {% endfor %}

  {% if session.id > 0 %}
    <form action="" method="post">
      <label for="comment">Add comment: </label>
      <textarea name="comment" id="comment" class="form-control"></textarea>
      {% if error == true %}<div class="error">{{ error }}</div>{% endif %}
      <br><input type="submit" value="Save comment" class="btn btn-primary">
    </form>
  {% else %}
    <p>You must <a href="{{ doc_root }}login">log in to make a comment</a>.</p>
  {% endif %}
</section>
```

ЗАКЛЮЧЕНИЕ
ДОБАВЛЕНИЕ НОВОГО ФУНКЦИОНАЛА

- Оптимизированные для SEO URL-адреса помогают поисковым системам индексировать страницы, их легче читать и по ним сразу видно, о чем идет речь на странице.
- Механизм веб-сервера Apache для перезаписи URL-адресов может проверять поступающие запросы и перенаправлять некоторые из них на другие страницы.
- При добавлении новой функциональности сначала определитесь с тем, какие действия могут совершать пользователи, прежде чем начнете писать код.
- Затем продумайте структуру хранения новых данных. Для новых сущностей используйте новую таблицу, дополнительные данные о существующей сущности помещаются в существующую таблицу, а связи между двумя сущностями могут использовать таблицы связей.
- Подзапрос — это запрос, который может быть вложен в другой SQL-запрос.
- Тестируйте новую функциональность на тестовом сервере (а не на реальном).
- Используйте копию базы данных для тестового сервера и создавайте резервную копию рабочей базы данных перед выкладыванием новой функциональности на боевой сервер.

ДАЛЬНЕЙШИЕ ШАГИ

Поздравляем вас с завершением этой книги. Изучив основы создания веб-сайтов на основе баз данных с использованием PHP, вы можете выбрать, что изучать дальше, исходя из того, что вы хотите делать с PHP.

Узнав, как PHP используется для создания веб-сайтов на основе баз данных, вы можете попробовать расширить код учебного приложения. Например, вы могли бы:

- создать список пользователей. Он может работать по тому же принципу, что и страниц категории, но вместо публикаций будут отображаться пользователи сайта;
- добавить страницу с отображением понравившихся определенному пользователю работ;
- создать систему обмена сообщениями для пользователей. Сообщения могут работать наподобие комментариев, но прочитать их сможет только тот, кому они были отправлены;
- внедрить разбивку на страницы на всех страницах, где выводятся карточки публикаций (так же, как это делается на странице поиска).

Вы также можете попробовать использовать код учебного приложения для создания сайта другого типа. В этом случае сначала добавьте таблицы в базу данных, а в код PHP — классы, работающие с этими таблицами. Например, сайт о музыке может представлять разных исполнителей и жанры. Сайт о садоводстве может включать в себя различные виды растений и способы ухода за ними. Сайт о кулинарии может показывать различные рецепты и их ингредиенты. Вы также можете изучить способы предоставления доступа к БД другим программам (например, приложениям, работающим на мобильных устройствах). Это делается с помощью так называемого интерфейса прикладного программирования (**API, Application programming interface**).

PHP-программисты не всегда пишут веб-сайты с нуля — они часто создают их, используя либо фреймворк, либо другую CMS. Однако для их использования вам необходимо знать основы создания веб-сайтов на основе баз данных (как показано в этой книге).

ФРЕЙМВОРКИ

Фреймворки предоставляют код, необходимый для выполнения общих задач при создании веб-сайтов и приложений с использованием PHP. Двумя наиболее популярными фреймворками для изучения могут быть:

- **Symfony** https://symfony.com;
- **Laravel** https://laravel.com.

УПРАВЛЕНИЕ КОНТЕНТОМ

Системы управления контентом (CMS) избавляют разработчиков от необходимости создавать сайты с нуля. Три популярных приложения CMS:

- **WordPress** https://wordpress.org;
- **Drupal** https://drupal.org;
- **Joomla** https://joomla.org.

В дополнение к настройке внешнего вида сайта вы также можете написать плагины WordPress, модули Drupal или расширения Joomla, добавляющие новые функциональные возможности. Многие программисты делятся этими плагинами, модулями и расширениями с другими.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ASCII, схема кодирования, 112
Bootstrap, фреймворк, 382
Composer, инструмент, 372
  управление пакетами, 374
  установка, 373
Cookie, файл, 208, 209
  защита, 212
  настройка, 211
  получение данных, 213
  получение доступа к, 211
  сессионный, 214
  создание, 210
Echo, сокращенная запись команды, 27
  использование, 27
HTTPGET, 108
HTTPPOST, 108
HTTP-заголовок
  добавление и обновление, 138
  перенаправление пользователей, 140
ISO 8859-1, стандарт, 112
MySQL
  знакомство с, 6
  история развития, 9
Packagist, сайт, 372
PHP
  блоки, 12
  внутри HTML, 11
  добавление комментариев в код, 11
  интерпретатор, 3, 106, 113, 121
  история развития, 7
  обновление переменной в, 19
  обновление страниц, 383
  отправка текста и HTML-кода в браузер, 13
  создание страниц на, 39
  страница на, 4, 12
  язык программирования, 1
PHPMailer, пакет, 391
phpMyAdmin, инструмент, 250
  использование, 250
SEO-ориентированные названия, 427
SQL, язык, 9
SQL-запрос
  привязка значений к, 289
  привязка целого числа к, 290
Twig, шаблонизатор, 378
  доступ к данным в шаблонах Twig, 385
  использование условий в шаблонах Twig, 387
  использование циклов в шаблонах Twig, 388
  отображение данных в шаблонах Twig, 386
  параметры, 381
  структурирование файлов шаблонов, 388
  фильтры, 386
  шаблон краткого описания статьи, 390
  шаблон статьи, 391
URL-адрес, 106
  дружественный SEO, 420
  обновление, 424
  перезапись, 420, 423
UTF-8, стандарт, 112
Автозагрузка, 348
Атака с внедрением SQL-кода, 288
База данных, 6
  создание функции поиска, 305
  автоинкремент, 248, 272
  веб-сайты под управлением, 245
  внешний ключ, 250
  выяснение количества измененных строк, 315
  добавление данных в, 251, 272
  добавление данных в таблицу, 313
  дублирование данных в, 249
  значения по умолчанию, 272
  использование изменяющихся в SQL-запросе данных, 287
  использование объектов для работы с, 349
  использование одного файла для отображения множества страниц, 290
  использование строк запроса для отображения требуемой страницы, 292
  класс, 354
  настройка, 251
  обновление, 399
  обновление данных в, 273, 313
  обновление данных в таблице, 314
  объединение столбцов, 269
  объект, 350
  ограничение внешнего ключа, 277
  ограничение и пропуск результатов, 262
  ограничение количества совпадающих результатов, 263
  отображение данных на HTML-страницах, 292
  отображение разных данных на одной странице, 289
  первычный ключ, 248
  повторяющаяся запись, 276
  подключение к, 278, 282
  подсчет и группировка результатов, 260
  подсчет количества совпадающих результатов, 261
  поиск значений, 258
  поиск с помощью оператора LIKE

и подстановок, 257
поле, 248
получение данных из, 254
получение данных из двух таблиц, 263
получение данных из нескольких таблиц, 266
получение данных с помощью SQL-запроса, 283
получение и проверка данных категории, 322
получение идентификатора новой строки данных, 315
получение нескольких строк данных из, 286
получение одной строки данных из, 285
пользовательская функция PDO без параметров, 295
пользовательская функция PDO с параметрами, 296
предотвращение дублирования значений в столбцах, 316
привязка значений к SQL-запросу, 289
привязка целого числа к SQL-запросу, 290
псевдоним таблицы, 267
резервное копирование, 275
результирующийнабор, 254
реляционная, 7, 245
создание, 251
создание веб-страниц для редактирования данных в, 317
создание и обновление данных, 320
создание и редактирование статей, 328
создание, обновление и удаление категорий, 318
сортировка результатов, 260
сохранение данных категории, 323
столбцы, 248
строки, 248
таблица, 6, 248
типы данных, 249
удаление данных из, 275
удаление данных из таблицы, 314
удаление категории, 326
уникальные ограничения, 276
управление порядком строк в результирующем наборе, 258
управление сайтом с помощью нескольких PHP-файлов, 296
условие поиска, 255
учетные записи пользователей, 252
форма для создания или редактирования данных категории, 325
форматирование данных HTML-страницы, 293
хранение данных, 247
хранение токенов в, 414
Безопасное взаимодействие, 109
Безопасность транспортного уровня, 111
Безопасный протокол передачи гипертекста, 109
Библиотека, 365
HTML Purifier, 375
TinyMCE, 376
добавление HTML Purifier в CMS, 377
использование, 371
Бит, 111
Бэкенд, 379
Валидация, 363
данных в переменных, 173
данных с помощью регулярных выражений, 159
длины текста, 158
загрузки файлов, 185
нескольких входных данных с помощью фильтров, 172
параметров, 161
размера и типа файла, 184
фильтры, 168
флажков, 163
форм, 165
форм с помощью фильтров, 177
цифр, 156
Видениус, Микаэль, 9
Включаемый файл, 41
Время, 196
временные метки UNIX, 199
интервал, 204
обновление, 202
повторяющиеся события, 205
указание с помощью строки, 198
установка, 203
форматы, 197
часовые пояса, 206
Выражение, 16, 28
include, 58
match, 43, 48
require, 58
группа, 130
квантор, 129
обратный слеш, 129
регулярное, 128
Данные
валидация, 143, 147
доступ к, 142
обработка отсутствующих данных в суперглобальных массивах, 145
отсутствие, 148
получение данных, отправленных через

HTTP GET, 144
принятие решения, 143
проверка допустимости значений, 163
сбор, 142
сбор с помощью функций фильтрации, 166
требуемые, 147
экранирование, 143, 149
Дата, 196
интервал, 204
обновление, 202
повторяющиеся события, 205
указание с помощью строки, 198
установка, 203
форматы, 196
часовые пояса, 206
Динамический веб-сайт, 2, 105
Домашняя страница, 298
Жонглирование типами, 37
Зависимость, 366
внедрение, 348
менеджер, 366
Заголовок
HTTP-запроса, 107
HTTP-ответа, 107
Значение выражения-переключателя, 47
Изображение, 178
изменение размера, 187
изменение размера и обрезка с помощью Imagick, 193
изменение размера с помощью GD, 191
обрезка, 188
редактирование с помощью расширений, 189
соотношение сторон, 187
Инструкция, 13
Исключение, 223, 235
неулавливаемое, 242
обработка, 238
обработка по умолчанию, 242
обработчик по умолчанию, 238
пользовательское, 236
создание пользовательского, 240
улавливание различных типов, 240
Источник ввода, 166
Категория
класс, 355
создание, обновление и удаление, 357
Класс, 16, 85, 89, 90, 106
автозагрузка, 362
база данных, 354
валидации, 363
встроенный, 114
для создания и отправки электронных писем, 395
дочерний, 354
использование для создания объекта, 92
категория, 355
конструктор, 91, 95
определение, 91
получение данных в виде объекта с помощью, 311
предопределенный, 310
родительский, 354
создание и использование, 90
создание объекта с использованием существующего, 312
члены, 98
Код
представления, 379
приложения, 378
состояния, 107
Кодовый блок
формирование, 43
Комментарий, 11, 14
добавление в программу, 15
многострочный, 15
Константа, 137
использование, 138
Конструктор, 95
продвижение свойств, 96
Контекст, 176
Корневой каталог, 340
приложения, 342
Кэширование, 381
Лердорф, Расмус, 7
Логи ошибок, 118
Логическое значение, 38
Магический метод, 96
Массив, 20, 106
ассоциативный, 20, 21
добавление и удаление элементов, 135
изменение и сохранение значений в, 25
индекс, 22
индексированный, 21, 22
многомерный, 25, 26
обновление, 24
работа с, 133
сортировка, 136
сохранение массива в, 25
суперглобальный, 113, 114, 145
хранение в свойстве объекта, 99
элемент, 20
Межсайтовый скриптинг, 149
Метод
геттер, 98
сеттер, 98
Многобайтовые символы, 112

Модель, 86
Модификатор доступа, 98
Область видимости, 70
  глобальная, 71
  локальная, 71
Обработка запросов, 424
Обход по ключам и значениям, 56
Объединение строк, 32
Объект, 10, 16, 85, 89
  CMS, 351, 360
  базы данных, 350
  данных PHP, 246, 278
  для представления даты и времени, 201
  исключения, 235
  контейнер, 351
  метод, 85, 88, 94
  настройка режима выборки для получения, 311
  получение данных в виде, 309
  преимущества использования, 102
  свойство, 85, 87, 92
  создание и использование, 90
  создание с использованием класса, 92
  тип, 86
  хранение в свойстве объекта, 100
Оператор, 10, 13, 28
  if, 42, 44
  if... elseif, 42, 46
  if...else, 42, 45
  include, 58
  include_once, 59
  JOIN, 263, 265, 267
  LIKE, 257
  require, 58
  require_once, 59
  SQL, 254
  switch, 42, 47
  арифметический, 28, 29
  больше, 33
  больше или равно, 33
  выполняемый, 50
  идентичности, 32
  конкатенации, 28, 31
  логический, 29, 34
  логическое И, 34
  логическое ИЛИ, 34
  логическое НЕ, 35
  меньше, 33
  меньше или равно, 33
  множественный оператор возврата, 77
  неидентичности, 33
  неравенства, 32
  приоритет, 30
  присваивания с конкатенацией, 31
  равенства, 32
  сравнения, 28, 32
  строковый, 29, 31
  структура условного, 43
  тернарный, 45
  трехстороннего сравнения, 33
  условный, 41, 292
Операция, 329
Отладка, 231, 381
Отправка данных из ссылок и форм, 109
Ошибка
  веб-сервера, 244
  нефатальная, 226, 230, 242
  обработка, 222
  обработка нефатальной, 233, 234
  обработка по умолчанию, 242
  обработка фатальной, 233
  отслеживание, 231
  парсинга, 226, 227
  сообщения об, 225
  управление отображением, 223
  уровни и типы, 226
  фатальная, 226, 228, 243
Пакет, 366, 372
  управление с помощью Composer, 374
  установка, 373
Пароль
  запрос на сброс, 416
  сброс, 417
  хеш, 400
Парсинг, 226
Переключатель, 160
Переменная, 15, 16
  глобальная, 72
  значение, 16
  имя, 16
  наименование, 18
  обновление, 19
  объявление, 17
  отображение содержимого, 117
  присвоение значения, 17
  создание, 17
  статическая, 72
  строгие типы, 381
  тип данных, 16
  функции фильтрации для работы с, 172
  хранение в функции, 72
  хранение данных, 345
Подключение внешних файлов, 57
Пользователь сайта
  аутентификация, 405
  вход в систему, 405, 407
  добавление параметров на страницу профиля, 412

идентификатор, 399
идентификация, 405
использование сессий для хранения данных, 410
ограничение доступа к страницам администрирования, 413
пароль, 399
персонализация, 406
персонализация панели навигации, 412
регистрация, 400, 403
роли, 400
Поток управления, 41, 313
Приведение типов, 37
неявное, 37
явное, 37
Принцип
единственной ответственности, 102
не повторяйся, 102
Программирование
основы, 10
Пространство имен, 366
абсолютное имя, 368
глобальное, 367
импорт класса в, 370
импорт кода в, 369
использование в классах CMS, 369
конфликт имен, 366
объявление, 366
создание, 366
Протокол, 106
передачи гипертекста, 105, 106
Путь, 106, 340
абсолютный, 340
относительный, 340
Разбивка на страницы, 305
Расширение, 189
Регистр, 124
Редакторы кода, 113
Реляционная база данных, 7
Рефакторинг, 348, 362
Сборка мусора, 217
Сервер
изменение настроек, 122
отправка данных формы на, 152
подключение к SMTP-, 392
прокси-, 139
Сертификат, 110
запрос на выпуск, 110
центр сертификации, 110
Сессия, 208, 213, 406
длительность, 214
жизненный цикл, 217
запуск, 214
идентификатор, 214
перехват, 221
сессионный файл cookie, 214
создание, 215
сохранение данных, 214
файл, 214
хранение и доступ к данным в, 217
Символ
зарезервированный, 149
подстановочный, 257
Система
авторизации, 219
управления контентом, 247
управления реляционными базами данных, 248
Скрипт, 12
Сообщение об ошибках, 114, 118
Ссылка
в письме, обновляющая базы данных, 413
передача по, 347
Статический сайт, 2
Статьи
настройка страницы, 330
обновление шаблона страницы, 437
получение данных, 359
получение и валидация данных, 332
сохранение изменений, 334
удаление, 337
форма/сообщение, 336
шаблон краткого описания, 390
Страница
категории, 300
поиска, 306
статьи, 301
участника, 303
Строка, 38
замена символов в, 127
запроса, 106
подстрока, 125
позиция символа, 125
поиск символов в, 125
проверка символов в, 126
удаление символов в, 126
Строковый литерал, 13
Сущность, 143
имя, 149
номер, 149
Схема кодирования, 105, 111
изображений, 111
символов, 111
Таблица
символов, 345
ссылок, 432
Типы данных
null, 19

в базах данных, 249
логические, 19
логическое значение, 38
объединенный, 75
объектный, 89
объявление, 63, 75
приведение, 19, 37
псевдотип, 75
скалярные, 18
смешанный, 75
составные, 20, 73
строгие, 76
строка, 18, 38, 124
число, 19, 37
Токен, 414
хранение в базе данных, 414
Транзакционные письма, 391
Транзакция, 329
откат, 329
Управляющая конструкция, 15, 41
Упрощенное (сокращенное) вычисление, 35
Уровень защищенных сокетов, 111
Условие, 41
Утверждение
подготовленное, 288
Участник сайта, 399
Файл, 178
PHP-файл, получающий и отображающий данные, 384
валидация размера и типа, 184
данные, 140
заголовка, 297
загрузка из браузера, 179
загрузочный, 344
конфигурации, 343
логов, 232
очистка имени и дубликатов, 183
перемещение в пункт назначения, 182
перемещение загруженного, 183
получение данных о, 141
получение на сервере, 181
права доступа к, 182
проверка загруженного, 181
проверка загрузки, 183
структура, 342
удаление, 140
футера, 297
Фигурные скобки, 43
Фильтры валидации, 168
использование для валидации отдельных значений, 169
нескольких входных данных, 171
сбор значений с помощью, 170
Фильтры очистки, 175
применение к переменным, 176
Флажок выбора, 160
проверка установки, 162
Форма
отправка данных на сервер, 152
получение данных, 153
проверка отправки, 155
Фраза причины, 107
Фронтенд, 379
Функция, 10, 16, 62, 106, 292
COALESCE, 270
CONCAT, 270
анонимная, 362
аргумент, 63, 68
возврат значения, 63, 66
встроенная, 83, 113, 116, 123
вызов, 63, 64, 68
даты и времени, 200
для выполнения инструкций SQL, 293
документация, 82
доступ или обновление глобальных переменных внутри, 72
доступ к переменным вне, 73
и составные типы данных, 73
именованный аргумент, 80
имя, 62
использование, 63
массивов, 134
многобайтовая строковая, 127
назначение, 82
наименование, 70
необязательный параметр, 78
область видимости, 70
обновления массива, 135
обработки исключений, 238
обработки ошибок, 233
объявление, 63, 64
определение возвращаемого типа данных, 74
опциональный параметр, 64
параметр, 63, 67
пользовательская, 83
регулярного выражения, 130
рекомендации по написанию, 81
сортировки массива, 137
фильтрации, 143, 166
хранение переменной в, 72
чувствительность к регистру, 125
Хост, 106
Хеш пароля, 401
Цикл, 41, 48, 293
do while, 49, 51, 52
for, 49, 52

foreach, 49, 55
while, 49, 50
для извлечения одной строки данных за итерацию, 287
проверка условия, 50
синтаксис, 50
Число, 37
валидация, 156
работа с, 132
Шаблон, 378
блоки, 388
дочерний, 388
рендеринг, 380
родительский, 388
Шифрование, 109
Язык сценариев, 12
ключ, 110
расшифровка, 109
шифр, 110
Экранирование
данных, 149
зарезервированных символов HTML, 151
кавычек, 13
пользовательского контента, 152
Электронное письмо, 391
отправка, 393
создание, 393
Язык структурированных запросов, 254
SQL-запрос, 254
инструкция SQL, 254

ПРИМЕЧАНИЯ

1 Это важно, потому что как раз «обычно» страница на PHP не содержит никакого HTML, который встречается только на отдельных страницах, специально предназначенных для вывода HTML, о чём будет говориться в последующих главах. — *Здесь и далее прим. науч. ред., если не указано иное.*

2 Все эти термины имеют строго определённое значение, и употребление одного вместо другого может приводить к непониманию.

Оператор (*operator*) — действие, чаще всего состоит из одного-двух символов: **, <, || и т. д.

Выражение (*expression*) — элемент языка, который возвращает какое-то значение. Инструкция (*statement*) — минимальный самостоятельный элемент языка, законченная команда. Единичная инструкция заканчивается точкой с запятой и может состоять только из одного выражения, но чаще — из нескольких.
Управляющая конструкция (*control structure*) — команды, которые меняют ход выполнения инструкций.
Функция (*function*) — фактически это именованный блок кода. Вызов функции тоже является выражением.
Языковая конструкция (*language construct*) — хитрая конструкция, которая похожа по действию на функцию, но ею не является. Что-то вроде исключений в обычном языке. В первую очередь отличается от функций отсутствием скобок. Но у некоторых конструкций они обязательны.

3 Традиционно завершающая запятая ставится, если элементы добавляются каждый на своей строке, и не ставится, если элементы пишутся в одну строку.

4 Строго говоря, последние два оператора не относятся к арифметическим, а составляют свой собственный класс операторов (наряду с операторами постфиксного инкремента и декремента), но для простоты они добавлены в ту же таблицу.

5 Начиная с версии 7.1, PHP в этом случае выдает ошибку A non well formed numeric value encountered, но приводит строку к числовому типу и возвращает результат. Это же замечание относится и к следующей строке.

6 Начиная с версии PHP 7.1 вызывает предупреждение A non well formed numeric value encountered. Начиная с версии 8.0 вызывает фатальную ошибку.

7 Следует отметить, что присвоение значений true и false сделано здесь для наглядности. В реальности, если нужно получить одно из этих значений, тернарный оператор не понадобится, поскольку оператор сравнения уже возвращает результат логического типа и выражение сократится до $child = $age < 16;.

8 Можно заметить, цикл for — это просто компактная запись цикла while со счетчиком. Ср.:

```
$i = 0;
```

```
while ($i < 10) {
   echo $i;
   $i++;
}
```

9 Поскольку это не печатная продукция, header и footer обозначают всего лишь неизменные верхнюю и нижнюю часть сайта.

10 Эти две конструкции были актуальны в то время, когда в PHP не было механизма автозагрузки классов (о котором будет рассказано далее в книге). В настоящее время они не рекомендуются к использованию.

11 Также это помогает обнаруживать проблемы на ранней стадии. Если в функцию пришли данные неверного типа, то PHP выдаст ошибку, вместо того чтобы продолжать работу и выдать заведомо неверный результат.

12 Однако если в этих файлах в свою очередь вызываются другие функции с указанием типов, то имеет смысл включать строгую проверку и для них.

13 Существует общепринятый стандарт оформления кода в PHP, который называется PSR-12. В частности, он определяет то, как пишутся названия классов, свойств и методов. Все они пишутся в CamelCase, но имя класса с заглавной первой буквы, а имена свойств и методов — со строчной.

14 Такое поведение будет вызывать ошибку в новых версиях PHP, и его следует избегать.

15 Начиная с этой главы, ради экономии места в книге примеры кода часто будут даваться в сокращении. Но вы всегда можете посмотреть полный пример, открыв в редакторе соответствующий файл из загружаемого кода. Именно такой вариант работы с книгой является наиболее продуктивным — параллельно с чтением книги открывать в редакторе на компьютере соответствующий пример кода, а также выполнять его в браузере.

16 Речь идет о таком понятии, как *наследование*, и праве доступа классов-потомков к свойствам и методам классов-родителей. Об этом можно прочитать в документации PHP, в разделе про ООП.

17 Вызывается по клавише F12, далее вкладка «Сеть», далее выбор запроса из списка.

18 Методов на самом деле больше, но основными являются эти два.

19 Строго говоря, при отправке методом POST тоже можно использовать строку запроса, помимо передачи данных в теле запроса. А вот для метода GET это единственный способ передачи данных на сервер, поэтому строка запроса традиционно ассоциируется с этим методом.

20 Здесь представлена только небольшая часть массива `$_SERVER`. Вы можете посмотреть его полное содержимое с помощью функции `phpinfo()`, о которой будет идти речь ниже.

21 На самом `var_dump()` может вывести сразу несколько переменных, разделенных запятыми. Также в нее можно передавать не только переменные, но и любые выражения.

22 С помощью этой функции можно переопределить не все параметры, а только те, которые помечены как PHP_INI_ALL в списке настроек в документации. Нужно быть внимательным, поскольку попытка изменить недоступную настройку не приводит к ошибке, а молча игнорируется.

23 На самом деле при удалении ничего страшного не произойдет — для отсутствующей настройки будет взято значение по умолчанию. Но, конечно, лучше, когда все настройки на месте.

24 Некоторые настройки PHP не могут быть переопределены в .htaccess. Они помечены как PHP_INI_SYSTEM в списке настроек в документации PHP. Также файл .htaccess доступен не всегда или в нем могут не работать команды php_flag и php_value.

25 Это очень важный факт. Если искомая подстрока находится в самом начале строки, то strpos() вернет 0. А 0 при использовании в конструкции if приравнивается к false. Поэтому при использовании strpos() для проверки наличия подстроки результат всегда надо сравнивать с false, используя оператор идентичности: `if (strpos(…) === false) // не найдено` или использовать для поиска последние три функции из таблицы.

26 По какой-то причине автор книги использовал собственные названия параметров функций в таблице ниже. Общепринятые можно посмотреть в документации. Быстро найти нужную функцию в онлайн-документации PHP можно, набрав в браузере php.net/имя_функции, например php.net/strpos.

27 Помимо поиска, очень важной особенностью этих функций является возможность вернуть совпавшую с шаблоном часть строки. Это делается нестандартным способом, путем заполнения необязательного третьего параметра $matches. Функция preg_match() заполняет только первое совпадение с шаблоном, а preg_match_all() – все. Подробнее можно прочесть в документации, набрав в браузере php.net/preg_match.

28 Вкратце разница заключается в том, что абсолютный путь является однозначным, но может меняться от сервера к серверу. А относительный зависит от текущей директории. По возможности всегда следует использовать абсолютные пути.

29 На самом деле речь идет о любых данных, а не только «предоставленных посетителем». Это очень важный момент. В вопросах безопасности никогда не следует делить данные на «полученные от пользователя» и все остальные. Защищены должны быть любые данные, без какого-либо разделения. Это и проще, и надёжнее.

30 К сожалению, здесь допущена серьезная ошибка. Отправка заголовка Location: неизбежно приведет к изменению кода ответа на 302, и код 404

браузер просто не получит. Вы можете в этом убедиться, посмотрев код ответа в инструментах разработчика в браузере. Соответственно, в такой ситуации нельзя использовать перенаправление. Информация об ошибке должна быть показана на той же самой странице, после выставления кода ответа 404. Т.е. вместо `header()` следует просто подключить нужный файл, например `include 'page-not-found.php';`.

31 Это не единственная причина. Даже без всяких хакеров, просто пара символов «меньше» и «больше» (< и >) заставит браузер принять их за HTML-тег, а лишняя кавычка может раньше времени закрыть значение атрибута. Поэтому экранироваться при выводе в HTML должны любые данные, независимо от их происхождения. Кроме того, иногда бывает сложно определить, что данные именно пользовательские. Опасные данные могут прийти из того источника, который вы посчитаете безопасным.

32 На самом деле массив $_GET может заполняться и при отправке методом POST, если строка запроса присутствует в URL-адресе, указанном в атрибуте action.

33 Следует отметить, что для версий PHP меньше 8.0 эта функция не выполняла свою основную задачу — проверить, является ли введенное значение собственно числом. Если в нее передать значение «20 лет», то в PHP версии 7.4 и ранее из-за приведения типов она вернет true. В PHP 8 сравнение чисел стало более строгим, но в любом случае лучше добавить в функцию вызов is_numeric().

34 Строго говоря, тернарный оператор здесь лишний, поскольку выражение в скобках уже возвращает булево значение, то есть достаточно написать `$terms = (isset($_POST['terms']) and $_POST['terms'] == 'true');`.

35 На самом деле здесь нужно сравнение не с булевым значением true, а со строкой 'true'. Автор сам выше пишет, что все значения, в том числе булевы, приходят из браузера в виде строк. Результат сравнения, впрочем, будет один и тот же, поскольку при нестрогом сравнении любая непустая строка соответствует значению true.

36 Этот фильтр объявлен устаревшим и не рекомендуется к использованию.

37 Важно не путать этот термин с результатом работы функции basename(), которая по умолчанию возвращает имя файла вместе с расширением, но без каталога.

38 На самом деле путь может быть как абсолютным, так и относительным. Но поскольку абсолютные пути всегда предпочтительнее, то в любом случае следует воспользоваться описанным подходом.

39 Вместо `dirname(__FILE__)` короче будет написать `__DIR__`.

40 Строго говоря, cookie не является файлом. Скорее это определенные HTTP-заголовки, которыми обмениваются браузер и сервер. При этом

браузер может сохранять cookie как в файлах, так и в любом другом виде — например, современные браузеры используют встроенную базу данных.

41 PHP исходно обрабатывал такие строки как числа, чтобы облегчить жизнь пользователям, которые могут ввести число с пробелом в конце, или дописать размерность, '1 шт.'. Однако в последние годы командой разработчиков языка был взят курс на устранение подобного нестрогого поведения, и, в частности, попытка использовать строку как число стала вызывать фатальную ошибку вместо предупреждения, а попытка использовать число с некорректными символами стала вызывать предупреждение вместо уведомления (а в совсем старых версиях PHP оба этих действия не вызывали вообще никаких ошибок). Но при этом и строка '0a', и строка '1a' не являются корректными числами — о чем PHP, собственно, и предупреждает.

42 Проблема тут несколько надуманная, поскольку если в переменной будет просто 0, то конечная стоимость будет та же. И наоборот, если в переменной будет значение '1a', то PHP умножит цену на 1. Скорее проблема тут в том, что '1a' заведомо не является корректным числом, что сигнализирует о явно неверных данных, которые не стоит обрабатывать вообще.

43 Для того чтобы эта функция работала всегда, можно в самом начале кода включить буферизацию вывода, благодаря которой PHP не сразу будет передавать всю выводимую информацию в браузер, а сначала будет собирать ее в буфер. И в этом случае отправка статуса HTTP ответа будет работать всегда. Кроме того, при ошибке можно будет вызвать функцию `ob_clean()`, которая удалит всё, что выводилось ранее, и вместо этого вывести пользовательское сообщение об ошибке.

44 Говоря простым языком, исключения — это те же ошибки, только более удобные в обращении. Потому что, во-первых, их можно перехватить и продолжить работу, а во-вторых, поскольку исключения являются объектами, то их функциональность можно расширять, а также создавать любое количество новых типов ошибок.

45 По-другому можно сформулировать так: *код должен выбрасывать исключение, если не может выполнить ту работу, для которой он предназначен*. Например, если код, который должен записывать данные в БД, не может это сделать, потому что какое-то значение является некорректным, то он должен выбросить исключение. И наоборот, код, который служит для проверки введенных данных, не должен выбрасывать исключение. Поскольку задача первого кода — запись в БД, а второго — проверка данных. Для первого неверные данные являются исключением, а для второго — нет.

46 Это не совсем верно. С сайта MySQL можно скачать бесплатную программу MySQL Workbench, которая как раз и предоставляет графический интерфейс к базе данных. Также есть множество программ других производите-

лей, как платных, так и бесплатных, например SQLYog, HeidiSQL, Navicat, а также инструмент для управления БД, встроенный в редактор PHPStorm. На мой взгляд, все эти программы гораздо удобнее, чем PHPMyAdmin.

47 Термин «колонка» часто употребляется не только в своем основном значении, обозначая колонку таблицы (то есть совокупность всех полей в этой колонке), но и как синоним поля, то есть определенной ячейки одной конкретной строки. Обычно из контекста понятно, о каком значении идет речь.

48 Тип timestamp следует применять с осторожностью, поскольку выводимое значение будет зависеть от текущего часового пояса. Чтобы избежать таких неоднозначностей, лучше использовать тип datetime, который всегда возвращает ровно то значение, которое было внесено в ячейку.

49 Строго говоря, collation — это не кодировка, а набор правил сравнения символов, который базируется на кодировке. Например, кодировка (charset) utf8mb4, а набор правил сравнения (collation) на ее базе — utf8mb4_unicode_ci. Для простоты оба понятия часто называют кодировкой, хотя это и технически некорректно.

50 Важно не путать пользователя базы данных с пользователем сайта. Для сайта создается одна учетная запись, которая используется в коде сайта для выполнения всех запросов. А учетные записи пользователей сайта создаются в таблице БД.

51 Также эти 4 типа операций часто сокращают как CRUD: Create (создание), Read (чтение), Update (обновление), Delete (удаление).

52 Обычно никто не придерживается такой строгой терминологии и SQL-запросами называют любые команды, отправляемые в БД.

53 Эта нечувствительность существует не сама по себе, а задается как раз теми самыми правилами сравнения (collation), о которых говорилось выше. В наборе правил, которые были использованы при создании БД — utf8mb4_unicode_ci, — буквы ci как раз и означают нечувствительность к регистру (case insensitive) при сравнении.

54 На самом деле информация об имени таблицы для каждой колонки присутствует в результирующем наборе, просто phpMyAdmin ее не отображает. Кроме того, функции, возвращающие результат запроса в PHP, также используют только имена колонок, без таблиц. Однако получить имя таблицы можно, см. метод getColumnMeta() класса PDOStatement.

55 Обычно для этих целей используется два файла: один содержит код создания объекта PDO и настройки, а второй — данные для подключения к БД, такие как имя сервера, номер порта, имя БД, логин и пароль. Таким образом на разных серверах будет меняться только файл с данными для подключения к БД, а файл с кодом создания объекта PDO будет оставаться одним и тем же. Но здесь для простоты используется один файл.

56 Здесь важно помнить, что объект `PDOStatement` тоже хранит весь результирующий набор в памяти. Поэтому в случаях, когда запрос может вернуть очень большое количество строк, рекомендуется выполнять его так, чтобы весь набор не сохранялся в памяти объекта. Об этом можно прочитать в документации PHP в разделе «Буферизированные и небуферизированные запросы».

57 Здесь имеется в виду query string — часть адреса URL, которая содержит пары имя=значение, а не SQL- запрос.

58 Для отечественных пользователей боле привычным будет формат `'d.m.Y'`.

59 Хотя в данном случае это и сработает, но использовать индексированный массив с именованными плейсхолдерами не рекомендуется. Если в execute() передается индексированный массив, то в запросе должны использоваться позиционные плейсхолдеры, в виде знаков вопроса, напр. WHERE id =?;.

60 Важный момент — в этом месте пропущено экранирование. При выводе переменной `$term` (а по-хорошему — вообще всех выводимых переменных) она должна экранироваться функцией `html_escape()`. Это хороший пример того, как легко можно пропустить уязвимость и как ненадежно полагаться на ручное экранирование, вместо которого следует использовать подход, при котором все переменные автоматически экранируются при выводе, о чем будет рассказано в следующем разделе.

61 Передача сообщений через строку запроса обычно не применяется, а здесь используется для краткости. Но лучше для этих целей использовать сессии.

62 Здесь в примерах отлавливается исключение типа PDOException, но правильнее отлавливать исключение самого высокого уровня — Throwable. Ведь если выполнение кода прервется при возникновении любого исключения — было ли оно вызвано ошибкой БД или какой-то другой причиной, — то часть запросов в останутся невыполненными. То есть откатывать транзакцию надо при любом исключении.

63 Если ваш сервер не поддерживает расширение Imagick, то в этом месте код выдаст ошибку. Чтобы выполнить этот пример, закомментируйте следующие три строки и вместо них напишите `move_uploaded_file($temp, $destination);`, и загруженный файл скопируется как есть.

64 Помимо этого неудобство относительного пути заключается в его неуниверсальности. В отличие от абсолютного, один и тот же относительный путь нельзя использовать в разных папках сайта.

65 Тут дело даже не столько в том, что написанный вручную путь длиннее, а в том, что константа позволяет сделать абсолютный путь настраиваемым. То есть мы получаем удобство относительных путей с точностью абсолютных. В отличие от настоящих относительных путей, построенные с помощью такой константы пути всегда будут постоянными, поскольку достраи-

ваются не от текущей папки, а от одной и той же точки — корневой папки сайта.

66 Чтобы не запутаться в корневых папках, надо четко понимать различие между ними: корневая папка файловой системы является самой верхней папкой на диске. В корневой папке приложения лежат все файлы сайта, в том числе и папка файлами, доступными публично, — корень сайта. Плюс с точки зрения браузера, который не видит файловую систему сервера, а видит только фалы в корневой папке сайта, корнем сайта является просто слеш.

67 Это означает, что понятие корня сайта имеет двойное толкование: для браузера это именно корень, самый верхний уровень. Для файловой же системы (и PHP) — это папка в файловой системе, внутри которой расположены файлы сайта, доступные браузеру.

68 В данном случае речь идет не о символах алфавита, а о таблице, которую создает интерпретатор, чтобы связать имена функций и переменных с актуальными участками в памяти, содержащими код или данные.

69 Фактически это означает переделку кода под известный паттерн MVC, Model-View-Controller. Классы в папке classes — это модели, «страницы», которые запрашивает пользователь, — контроллеры, а шаблоны — это компонент View. Занимаясь профессиональной разработкой, вы увидите, что практически все современные сайты строятся по примерно такой же схеме.

70 Этот принцип называется Liskov Substitution Principle, или LSP, и представлен буквой L в известной аббревиатуре SOLID. При этом формально совпадение конструкторов для соблюдения этого принципа не требуется, поскольку в нем говорится только об использовании дочернего класса, а не о создании. Но совпадение сигнатур контроллеров в любом случае повышает совместимость классов.

71 Это очень интересная и редко используемая техника. Она позволяет использовать в запросе отключаемые условия. Если значение передано, то условие работает, а если передано значение null — то условие как бы исключается из запроса (поскольку условие: `member1 IS null` будет вычисляться для каждой строки как `null IS null`, то есть всегда возвращать true, и строка будет добавляться в результирующий набор).

72 На самом деле, если пространство имен содержит хоть один обратный слеш, оно всегда считается глобальным, и использовать для таких пространств имен ведущий слеш не обязательно (и не рекомендуется).

73 Как вариант, можно написать `use PDO;` или `use PDOException;` (об операторе `use` будет рассказано чуть ниже), и тогда можно будет обращаться к этим классам без обратного слеша.

74 Поскольку Composer — это специальным образом упакованный PHP-скрипт, то его можно вызывать так же как и любой другой, с помощью

интерпретатора PHP: `php composer.phar` (указав правильный путь, если Composer находится в другой папке).

75 Просто вывести сообщение сразу после отправки формы будет не совсем правильно. Если после отправки обновлять страницу, то письмо будет отправляться снова и снова. Правильно будет после успешной обработки любого запроса методом POST всегда делать редирект, а сообщение либо добавлять в адресную строку, как это делалось в предыдущих главах, либо — предпочтительнее — в сессию (т. н. flash messages).

76 Английское слово hash переводится как «фарш». И точно так же как фарш невозможно провернуть назад, получив из него исходный кусок мяса, хеш означает необратимое шифрование. Но при этом для паролей мы используем такую воображаемую мясорубку, которая из одного и того же куска мяса всегда выдает один и тот же фарш. Таким образом, сравнив образцы фарша, мы сможем проверить идентичность введенных паролей. Соль, упоминаемая далее, так же относится к этой кулинарной аналогии.

77 Поскольку тег `raw` является небезопасным, а используется он только ради тегов `<br>`, то их лучше было бы убрать из сообщения, а при выводе использовать фильтр `n12br`.

78 В учебном коде это сделано для экономии места, но в реальном проекте так делать не стоит. В классе сессии не должно быть свойств, относящихся к конкретной предметной области. Ведь в сессии может храниться множество разных типов данных — корзина товаров, настройки отображения, язык интерфейса. Поэтому класс должен быть нейтральным и позволять хранить любые данные. Имеет смысл сделать одно свойство, которое будет содержать массив, в который следует класть экземпляры других объектов или массивы.

79 Возникает вопрос — зачем тогда вообще делать метод update()? В будущем код этого метода может поменяться и уже не совпадать с кодом create(). В этом случае его можно будет переписать, не меняя вызовы в остальном коде.

80 Русскоязычными разработчиками часто используется термин «ЧПУ» — Человеко-Понятный URL.

81 Такая обработка запросов называется роутингом (routing). Следует учитывать, что для экономии места использованный в книге вариант роутинга является не самым оптимальным, и в реальном проекте лучше либо написать свой, либо использовать готовый пакет роутинга, например nikic/fast-route.

82 Правильнее было бы вместо пустой строки присвоить значение index, чтобы так же, как и с публичными страницами, вызывался файл admin/index.php.

83 Лучше такие элементы делать либо формой с методом POST, либо через AJAX. Если же сделать просто ссылкой, то индексирующие боты поиско-

вых систем будут постоянно переходить по этой ссылке, создавая лишнюю нагрузку на сайт.

84 Здесь это сделано для экономии места, но в реальном коде так лучше не делать и передавать нужные данные не массивом, а в виде отдельных параметров.

Все права защищены. Книга или любая ее часть не может быть скопирована, воспроизведена в электронной или механической форме, в виде фотокопии, записи в память ЭВМ, репродукции или каким-либо иным способом, а также использована в любой информационной системе без получения разрешения от издателя. Копирование, воспроизведение и иное использование книги или ее части без согласия издателя является незаконным и влечет уголовную, административную и гражданскую ответственность.

Производственно-практическое издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Дакетт Джон

PHP И MYSQL
Серверная веб-разработка

Главный редактор *Р. Фасхутдинов*
Руководитель направления *В. Обручев*
Ответственный редактор *Д. Калачева*
Литературный редактор *Д. Журавлева*
Научный редактор *Р. Шевченко*
Младший редактор *Д. Данилова*
Художественный редактор *А. Гусев*
Компьютерная верстка *Э. Брегис*
Корректоры *Е. Сербина, Р. Болдинова*

Страна происхождения: Российская Федерация
Шығарылған елі: Ресей Федерациясы

ООО «Издательство «Эксмо»
123308, Россия, город Москва, улица Зорге, дом 1, строение 1, этаж 20, каб. 2013.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru
Өндіруші: «ЭКСМО» АҚБ Баспасы,
123308, Ресей, қала Мәскеу, Зорге көшесі, 1 үй, 1 ғимарат, 20 қабат, офис 2013 ж.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Интернет-магазин : www.book24.ru

Интернет-магазин : www.book24.kz
Интернет-дүкен : www.book24.kz
Импортёр в Республику Казахстан ТОО «РДЦ-Алматы».
Қазақстан Республикасындағы импорттаушы «РДЦ-Алматы» ЖШС.
Дистрибьютор и представитель по приему претензий на продукцию,
в Республике Казахстан: ТОО «РДЦ-Алматы»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды
қабылдаушының өкілі «РДЦ-Алматы» ЖШС,
Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 251-59-90/91/92; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ
о техническом регулировании можно получить на сайте Издательства «Эксмо»
www.eksmo.ru/certification
Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Дата изготовления / Подписано в печать 27.06.2023. Формат 70x100 1/16.
Печать офсетная. Усл. печ. л. 55,74.
Тираж экз. Заказ

В электронном виде книги издательства Эксмо вы можете купить на www.litres.ru
ЛитРес: один клик до книг

ЧИТАЙ ГОРОД

12+

eksmo.ru
Официальный интернет-магазин издательства «Эксмо»

Хочешь стать автором «Эксмо»?

ЭКСМО
Издательство «Эксмо» — универсальное издательство №1 в России, является одним из лидеров книжного рынка Европы.
eksmo.ru eksmo

ISBN 978-5-04-171951-7
9 785041 719517 >

Эта книга предназначена для

разработчиков сайтов и приложений, веб-дизайнеров и программистов.

Вы научитесь

программировать на языке PHP, создавать сайты с нуля и управлять базами данных.

Эта книга – самый простой и интересный способ изучить PHP и начать работать с базами данных MySQL.

Внутри вы найдете наглядные фрагменты кода, скриншоты и иллюстрации, с помощью которых не только легко и быстро освоите азы программирования на PHP, но и примените полученные знания на практике, создав свой первый веб-сайт или приложение. Помимо этого, автор сосредотачивается на лучших практиках современного веб-дизайна, чтобы ваш продукт получился не только функциональным, но и удобным для будущих пользователей.

Никаких специальных навыков для освоения этой книги не требуется.

Джон Дакетт проектирует и разрабатывает веб-сайты уже более двадцати лет. Он работает как с небольшими стартапами, так и с глобальными брендами. Под его авторством выпущено несколько книг, посвященных разработке сайтов и приложений, веб-дизайну, программированию и веб-юзабилити.

ISBN 978-5-04-171951-7

«Это очень своевременная книга. В последние годы язык PHP (которому скоро исполнится 30 лет!) обрел второе дыхание, получив много новых возможностей. Все они представлены в этой книге, вместе с самыми передовыми приемами программирования. Здесь не будет устаревших подходов, которые, что уж греха таить, часто встречаются в обучающих материалах по PHP».

Роман Шевченко, преподаватель PHP