Майк МакГрат

PHP 7

ДЛЯ НАЧИНАЮЩИХ

С ПОШАГОВЫМИ ИНСТРУКЦИЯМИ

Веб-разработка — это просто!

Мировой компьютерный бестселлер

Mike McGrath

PHP7

In Easy Steps

Майк МакГрат

PHP 7

ДЛЯ НАЧИНАЮЩИХ

С ПОШАГОВЫМИ ИНСТРУКЦИЯМИ

Москва
2018

УДК 004.43
ББК 32.973.26-018.1
М15

Mike McGrath
PHP7 in easy steps, 5th edition
By Mike McGrath. Copyright ©2015 by In Easy Steps Limited.
Translated and reprinted under a licence agreement from the Publisher:
In Easy Steps, 16 Hamilton Terrace, Holly Walk, Leamington Spa,
Warwickshire, U.K. CV32 4LY.

МакГрат, Майк.

М15 PHP7 для начинающих с пошаговыми инструкциями / Майк МакГрат. — Москва : Издательство «Э», 2018. — 256 с. — (Программирование — это просто).

ISBN 978-5-699-98594-4

Посвященная самому популярному на сегодняшний день языку программирования, эта книга помогает освоить азы PHP7 даже тем новичкам, которые не знакомы с этим языком, а также с программированием вообще. Благодаря традиционно доступному изложению, присущему всем книгам серии «Программирование для начинающих», обилию иллюстраций и примеров, а также множеству полезных советов, эта книга — лучшее пособие для начинающих программистов.

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-699-98594-4

© Райтман М.А., перевод на русский язык, 2017
© Оформление. ООО «Издательство «Э», 2017

Оглавление

1 Подготовка рабочей среды 9

Введение в PHP 10
Установка веб-сервера Abyss 13
Установка PHP-движка 17
Интеграция Abyss и PHP 20
Внедрение PHP-кода. 23
Правила написания сценариев. 25
Заключение. 28

2 Хранение значений 30

Переменные 31
Заключение строк в кавычки 34
Массивы 36
Сортировка массивов 40
Размерность массивов 43
Типы данных 45
Константы. 48
Суперглобальные переменные. 51
Заключение. 54

3 Операторы 56

Арифметические операторы 57
Операторы сравнения 59
Условные операторы 62
Логические операторы 65
Побитовые операторы 67
Операторы инкремента и декремента 70
Приоритет операторов 71
Заключение. 73

4 Условные конструкции 74

Конструкция if 75
Конструкция else 77
Конструкция switch 80
Работа с циклами 83
Циклы while и do while 85
Прерывание циклов 88
Заключение. 90

5 Функции 92

Определение функций 93
Передача аргументов 95
Параметры функции 98
Область видимости 100
Возврат значений. 103
Обратный вызов 105
Заключение. 108

6 Строки 110

Сравнение символов 111
Поиск текста 113
Извлечение подстроки 116
Изменение регистра символов. 117
Форматирование строк. 118
Вывод даты/времени 121
HTML-сущности 124
Заключение. 126

7 Классы и объекты 128

Инкапсуляция данных. 129
Создание объектов. 132
Инициализация членов класса 134
Конструкторы и деструкторы. 137
Наследование свойств 139

Полиморфизм 142
Заключение. 144

8 Работа с файлами 146

Чтение файлов. 147
Чтение строк 149
Чтение символов 151
Запись в файл 152
Добавление текста 155
Обработка ошибок 157
Перехват исключений. 160
Заключение. 163

9 Разработка веб-форм 165

Выполнение действий. 166
Проверка заполнения формы 169
Проверка введенных данных 171
Фильтрация данных 174
Передача скрытых данных. 176
Обработка данных формы 179
Веб-формы с сохранением данных 181
Выгрузка файлов 184
Группировка элементов формы 187
Передача данных в сценарий. 190
Заключение. 192

10 Сохранение данных 194

Передача cookie-данных 195
Установка cookie-записей 196
Получение cookie-записей. 198
Просмотр cookie-записей 201
Передача данных сессии 203
Настройка сессий. 204
Получение данных сессии 206
Просмотр данных сессии. 209

Заключение. 211

11 Подключение баз данных 213

Подключение компонентов 214
Создание форума 217
Страницы форума 219
Предоставление формы 222
Обработка сообщений 224
Проверка работоспособности 227
Заключение. 230

12 Подключение веб-служб 231

Загрузка данных 232
Получение узлов 234
Получение атрибутов 237
Добавление RSS-лент 239
Настройка параметров 242
Выбор компонентов 245
Заключение. 248

Предметный указатель 249

1

Подготовка рабочей среды

Добро пожаловать в захватывающий мир интерактивных веб-сайтов, созданных с помощью языка PHP. В этой главе рассказывается, как сформировать среду динамической разработки с использованием веб-сервера и PHP-движка.

- **Введение в PHP**
- **Установка веб-сервера Abyss**
- **Установка PHP-движка**
- **Интеграция Abyss и PHP**
- **Внедрение PHP-кода**
- **Правила написания сценариев**
- **Заключение**

Введение в PHP

Наиболее привлекательные современные веб-сайты обеспечивают индивидуальный подход к каждому пользователю, динамически реагируя на текущие условия — имя пользователя, время суток, даты последних публикаций, содержание корзины покупок и т.д.

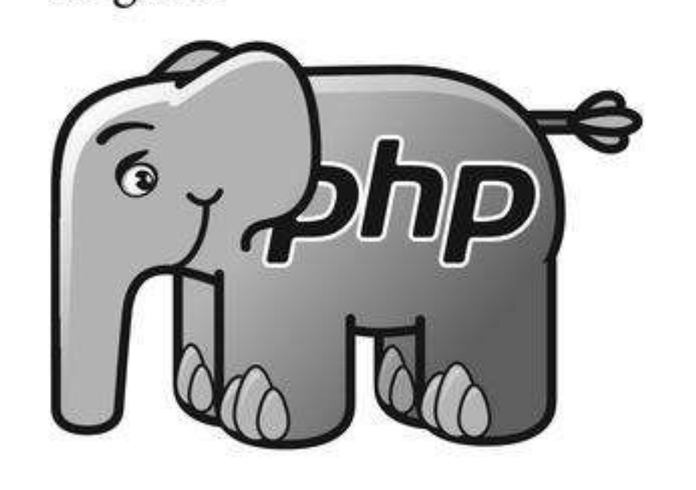
Многие из этих динамических веб-сайтов написаны на языке PHP.

Что такое PHP?

PHP — широко используемый универсальный сценарный язык, который особенно подходит для веб-разработки и может быть внедрен в HTML-код. Этот язык был создан программистом Расмусом Лердорфом в виде набора сценариев для поддержки собственного веб-сайта. Первый релиз под названием «Personal Home Page Tools (PHP Tools) version 1.0*» увидел свет 8 июня 1995 года.

Официальный логотип проекта PHP. Официальный веб-сайт проекта доступен по адресу **php.net**.

Набор инструментов был дополнен в 1997 году с выходом версии 2.0. А с выходом в 1998 году значительно переработанной версии 3.0 изменилось и название — на рекурсивный акроним «PHP: Hypertext Preprocessor**». Следующую версию, 4.0, выпущенную в 2000 году, отличали увеличение производительности, надежности и расширяемости за счет использования нового ядра Zend Engine.



Это elePHPant*** — талисман проекта PHP, созданный французским художником Винсентом Понтиером.

---

* Инструменты для создания персональных веб-страниц (*англ.*). — *Прим. пер.*

** PHP: препроцессор гипертекста (*англ.*). — *Прим. пер.*

*** Игра слов elephant (слон) и PHP. — *Прим. пер.*

Пятая версия языка увидела свет в 2004 году и была выпущена как свободное программное обеспечение группы PHP-разработчиков. В основном изменения коснулись обновления ядра Zend (Zend Engine 2). Планируемая экспериментальная версия PHP6, в которой предполагалось внедрить нативную поддержку Юникода, была признана бесперспективной. Текущая версия, PHP7, была выпущена в 2015 году и основана на последней версии ядра, Zend Engine 3, который обеспечивает повышенную производительность. Сегодня PHP-движок установлен на более чем 20 млн веб-сайтов и 1 млн веб-серверов.

Новинка! Примечания с таким значком, которые вам будут встречаться в этой книге, описывают новые функции, появившиеся в версии PHP7.

Почему PHP так популярен?

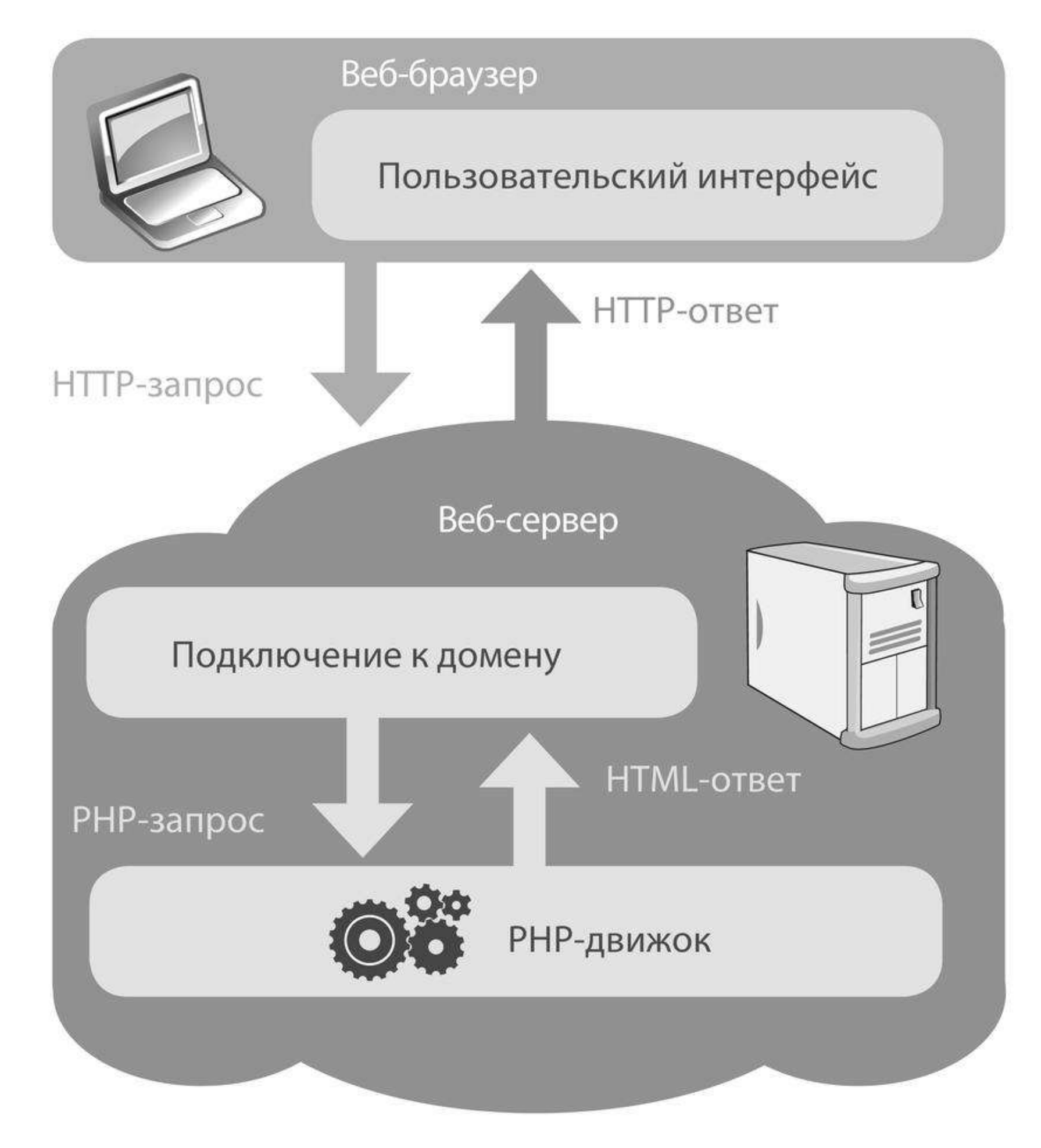
- Язык PHP наряду с тем, что очень прост и понятен начинающим пользователям, предлагает множество дополнительных возможностей профессиональным программистам.
- PHP-код заключается в специальные начальные и конечные теги, которые позволяют входить/выходить из «режима PHP», чтобы выполнять инструкции внутри HTML-документа.
- PHP-код выполняется на веб-сервере (на стороне сервера), в отличие от JavaScript-кода, запускаемого в браузере (на стороне клиента). Клиент получает результаты выполнения сценария, не зная, какой исходный код при этом использовался. В последнее время обработка на стороне сервера стала называться «облаком».

Что такое «облако»?

Всякий раз, когда пользователь переходит на веб-сайт в браузере, он запрашивает веб-страницу с веб-сервера и получает ее в ответ по протоколу HTTP. Если веб-страница содержит PHP-сценарий, веб-сервер сначала обратится к PHP-движку для обработки кода перед отправкой ответа в веб-браузер.

Совет

HTTP (Hypertext Transfer Protocol — Протокол передачи гипертекста) — универсальный коммуникационный стандарт, позволяющий любому компьютеру подключаться к любому веб-серверу для получения доступа к файлам через интернет.



Последующие страницы описывают, как подготовить среду разработки интерактивных веб-сайтов путем установки следующих серверных технологий на собственном компьютере:

- Веб-сервер — Abyss Web Server X1 (Personal Edition (бесплатно)) 2.11.2
- PHP-движок — PHP 7.1.3

На заметку

Примеры, приведенные в этой книге, созданы и проверены в указанных версиях программного обеспечения и могут потребовать внесения изменений при работе с другими версиями.

Установка веб-сервера Abyss

Abyss X1 представляет собой бесплатный компактный веб-сервер для операционных систем Windows, macOS и Linux, дистрибутив которого доступен для скачивания по адресу **aprelium.com**.

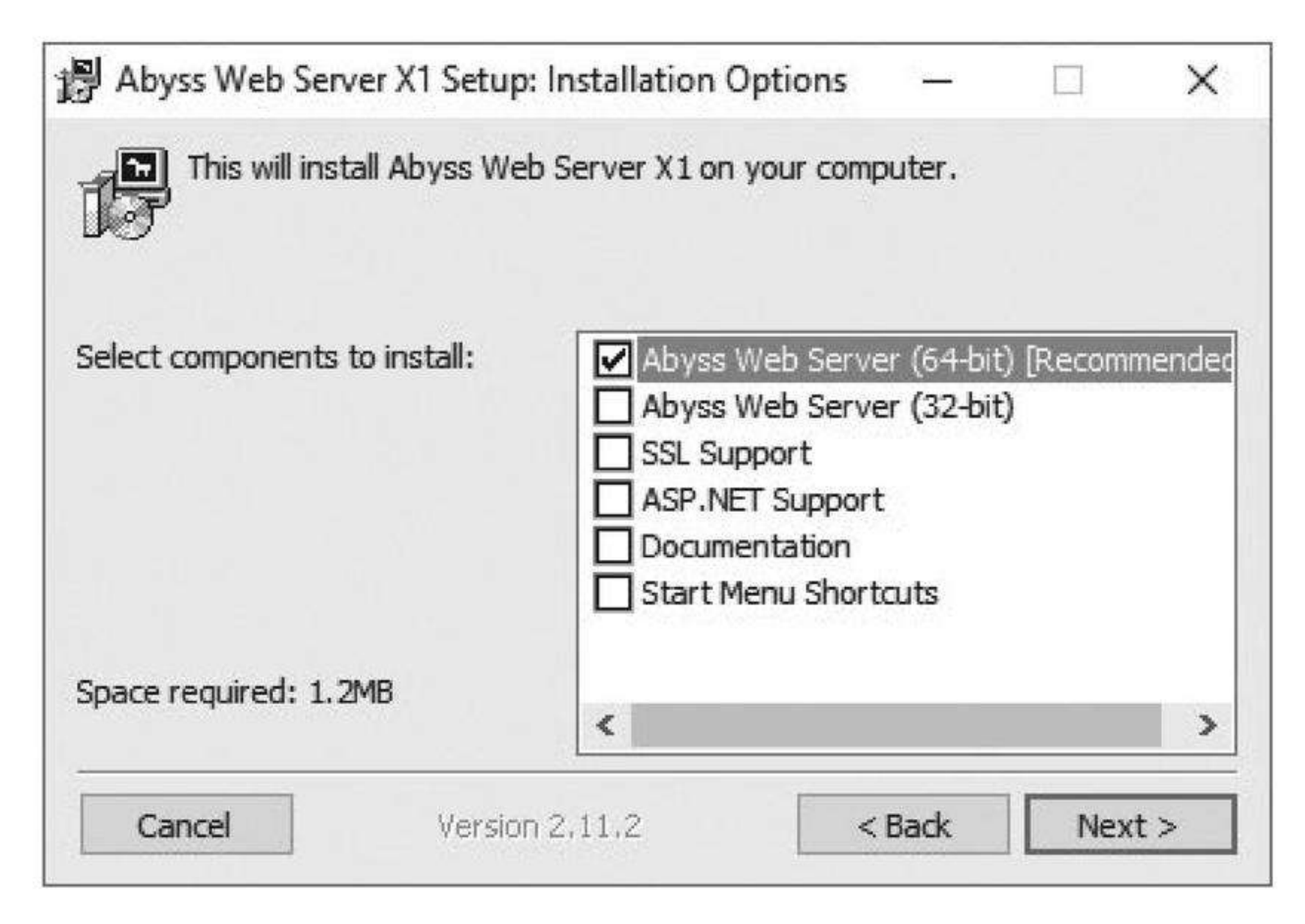
Несмотря на небольшие размеры сервер Abyss X1 поддерживает множество мощных функций, в том числе и касающихся генерации динамического контента с помощью серверных сценариев, в результате представляя собой идеального компаньона для PHP.

Веб-сервер Abyss X1 устанавливается на компьютер для обеспечения среды разработки интерактивных PHP-сайтов следующим образом.

1. Загрузите дистрибутив веб-сервера Abyss X1 для операционной системы, используемой на вашем компьютере, со страницы **aprelium.com/abyssws/download.php**.

Подробные инструкции по установке веб-сервера Abyss доступны на странице **aprelium.com/abyssws/start.html**.

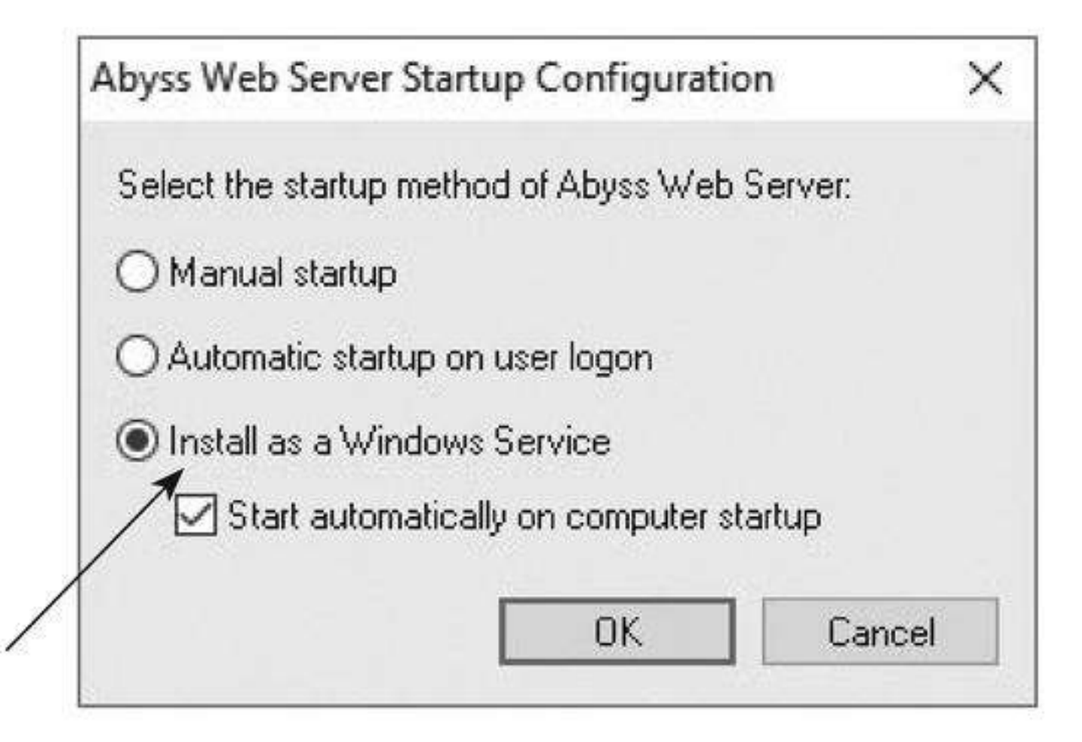
2. Запустите программу установки и нажмите кнопку **I Agree** (Я согласен), соглашаясь с условиями лицензионного соглашения. В следующем окне установите флажок **Abyss Web Server** (Веб-сервер Abyss) и нажмите кнопку **Next** (Далее).



На заметку

Дистрибутив веб-сервера Abyss для операционной системы Windows представляет собой исполняемый файл с именем *abwsx1.exe*, который необходимо запустить для установки веб-сервера.

3. Предложенный каталог установки *C:\Abyss Web Server* оставьте без изменений и нажмите кнопку **Install** (Установить). При появлении запроса, показанного на рисунке, установите переключатель в положение **Install as a Windows Service** (Установить как службу Windows) и нажмите кнопку **OK**.



Внимание

Если вы установите переключатель в положение **Manual Startup** (Запуск вручную), значок веб-сервера Abyss не будет отображаться на панели уведомлений для быстрого запуска/остановки сервера и доступа к консоли сервера. Вместо этого для доступа к консоли в браузере потребуется использовать адрес **http://localhost:9999** или **http://127.0.0.1:9999** (значение **localhost** — это псевдонимом IP-адреса 127.0.0.1).

После завершения процесса установки и запуска веб-сервера, автоматически откроется веб-браузер, используемый в вашей системе по умолчанию, и вы увидите страницу консоли веб-сервера Abyss.

Совет

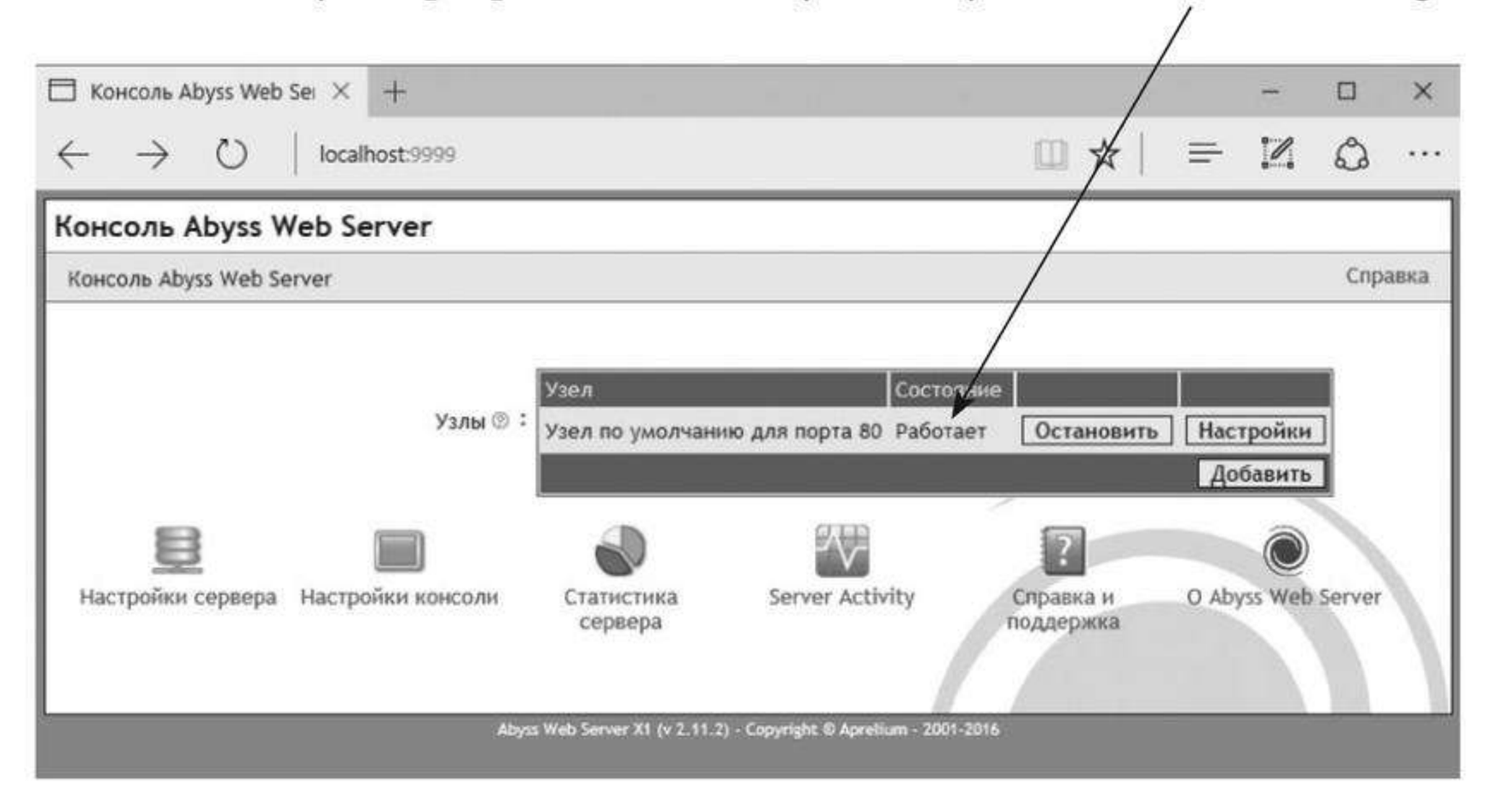
Совет. Чтобы установить русский язык для локализации интерфейса консоли веб-сервера Abyss, скачайте нужный файл со страницы **aprelium.com/abyssws/languages/** и сохраните его в каталог *C:\Abyss Web Server\lang*. После этого перезапустите веб-сервер Abyss (откройте окно диспетчера задач, щелкните правой кнопкой мыши по пункту **AbyssWebServer** на вкладке **Службы** (Services) и выберите команду **Перезапустить** (Restart)). Затем в окне консоли по адресу **http://localhost:9999** выберите пункт **Console Configuration** → **Language** (Настройки консоли → Язык) и нажмите кнопку соответствующего языка, файл которого вы скопировали ранее.

4. Выберите язык интерфейса, а затем введите желаемые логин и пароль для дальнейшего доступа к консоли веб-сервера Abyss.

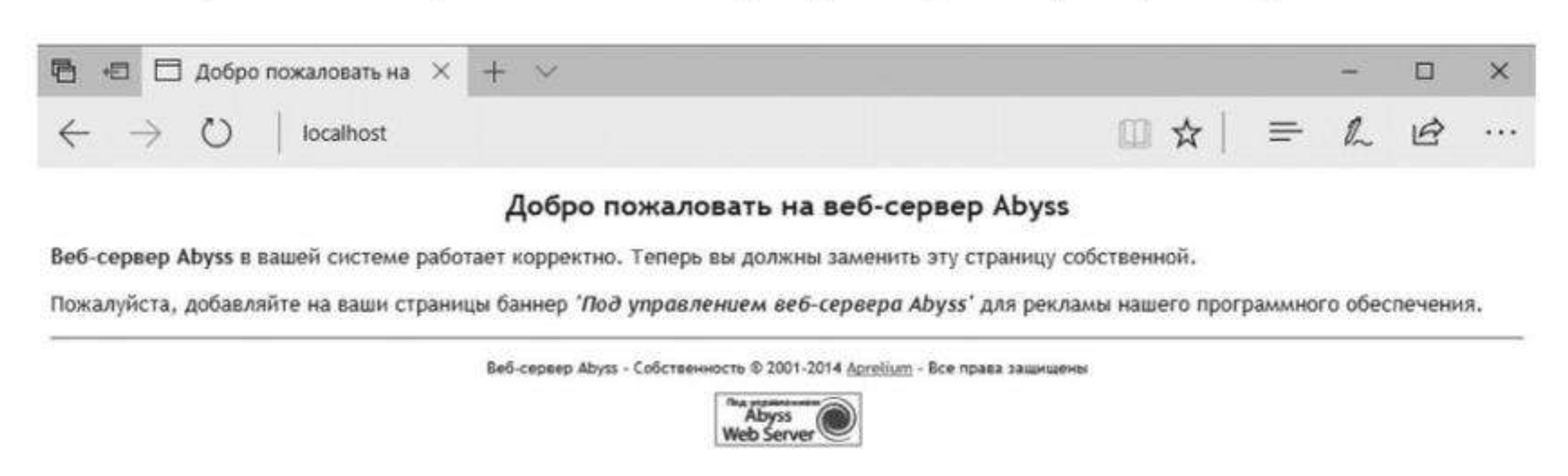
Совет

На странице консоли Abyss нажмите кнопку **Настройки** (Configure), щелкните мышью по значку **Общие** (General), чтобы просмотреть (изменить) используемые HTTP-порт (должно быть указано значение **80**) и каталог размещения документов (в котором будут сохраняться ваши веб-страницы; должно быть указано значение **htdocs/**).

5 Теперь, авторизуйтесь в системе с указанными на предыдущем шаге логином и паролем, чтобы получить доступ к консоли веб-сервера Abyss. В столбце статуса сервера должен быть указан пункт **Работает** (Running).



6 Введите **http://localhost/** в адресной строке браузера, а затем нажмите клавишу **Enter**. Вы увидите стандартную страницу Abyss с приветствием.



Совет

Если при переходе по адресу **http://localhost/** вместо страницы с приветствием веб-сервера Abyss вы видите ошибку, вероятно, какое-либо приложение в системе занимает порт 80, используемый по умолчанию. Часто такая ошибка возникает при одновременном использовании программы Skype. Для решения проблемы выполните действия, приведенные на веб-странице **goo.gl/GlvNMX**.

Установка PHP-движка

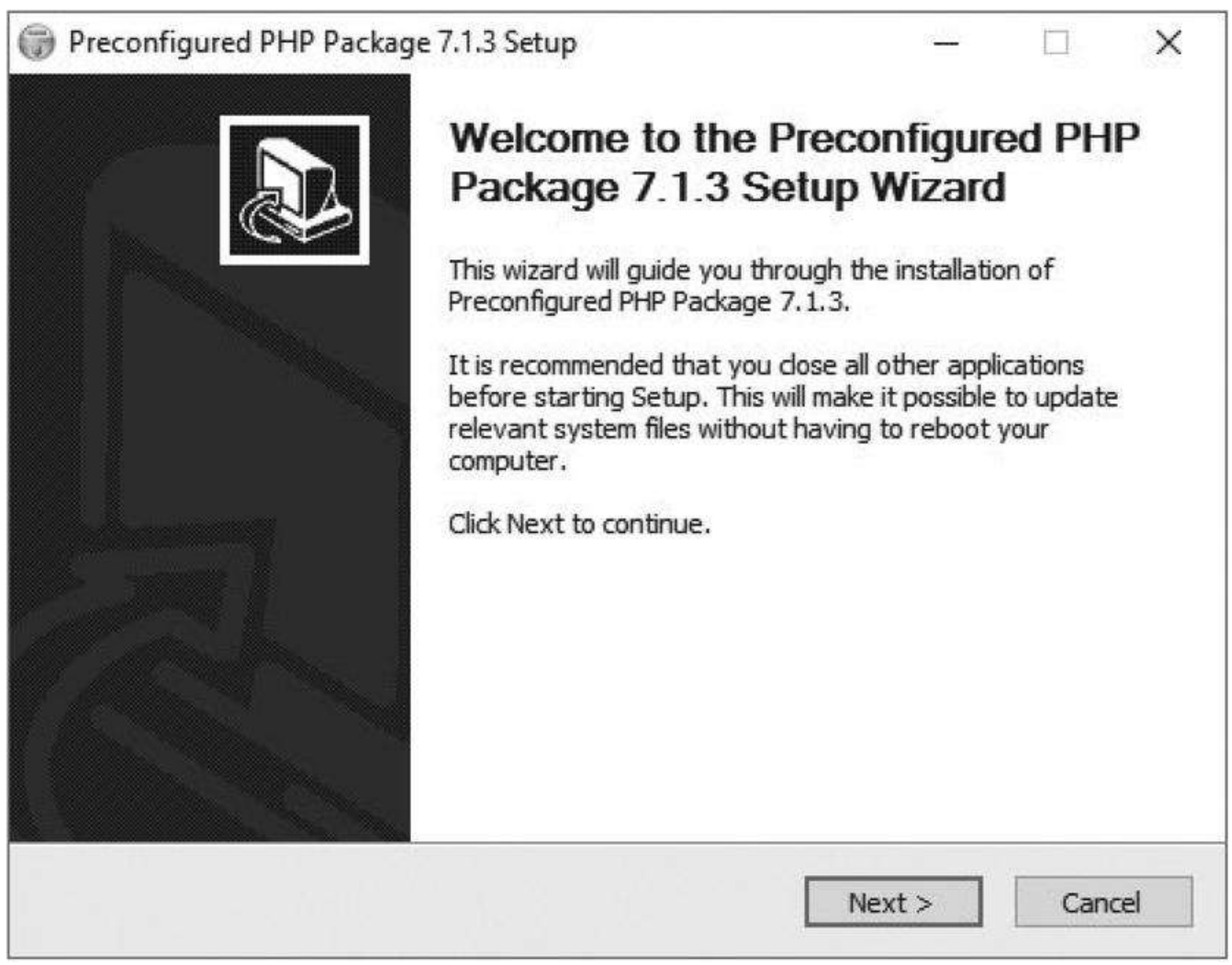
«Движок» интерпретатора PHP, который обеспечивает выполнение PHP-сценариев на веб-страницах, доступен в версиях для операционных систем Windows, macOS и Linux и может быть бесплатно загружен с сайта **php.net**.

Полная инструкция по установке PHP-движка доступна на веб-странице **php.net/manual/ru/install.php**.

Кроме того, вы можете скачать дистрибутив, специально сконфигурированный под веб-сервер Abyss для операционной системы Windows с сайта **aprelium.com**. Эта версия дистрибутива рекомендуется для простой и быстрой установки.

1. Загрузите дистрибутив PHP-движка для используемой операционной системы с сайта **aprelium.com/downloads**.

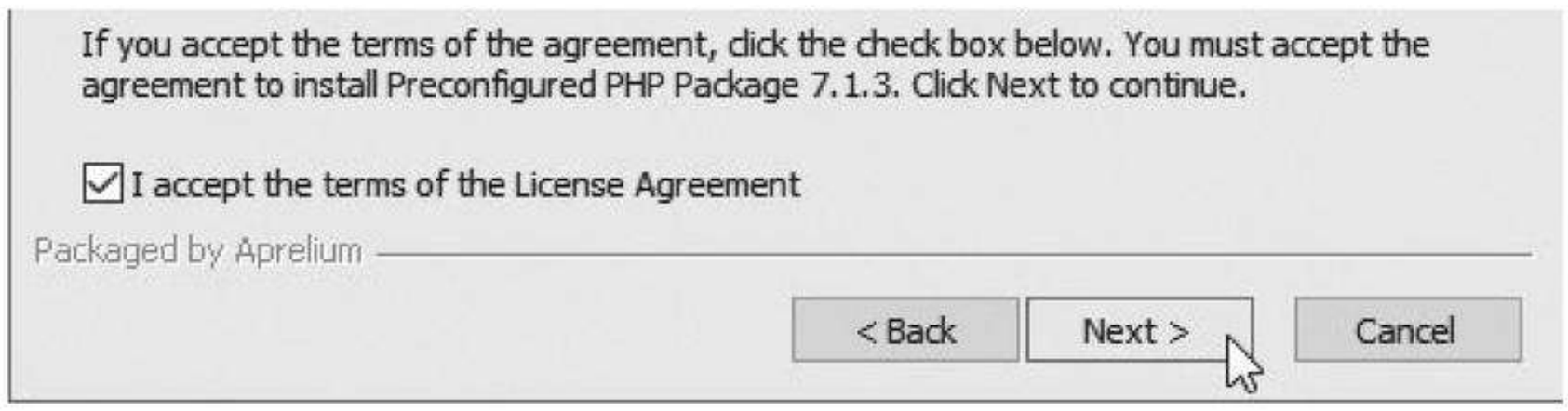
2. Запустите загруженный исполняемый файл, чтобы запустить программу установки. В открывшемся окне мастера установки нажмите кнопку **Next** (Далее), чтобы приступить к процессу инсталляции.



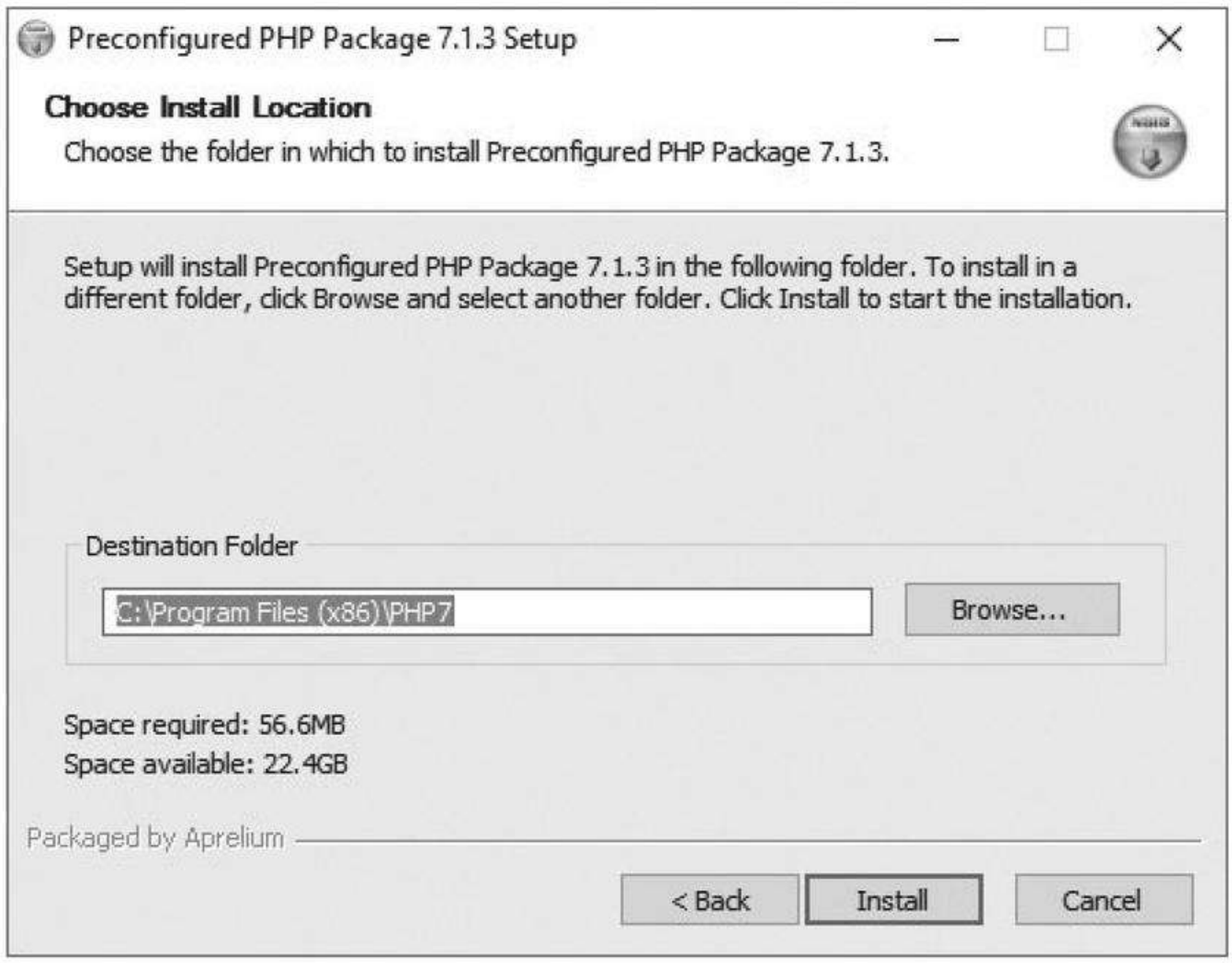
Внимание

Если вы хотите установить PHP-движок для работы с веб-сервером Abyss в операционной системе Windows, дистрибутив которого планируете скачать с сайта **php.net**, рекомендуется выбрать одну из потокобезопасных версий, помеченных как VC14 Thread Safe, — в этом случае PHP устанавливается как модуль и требуется меньше зависимостей Windows.

3. Далее примите условия лицензионного соглашения, установив соответствующий флажок, и нажмите кнопку **Next** (Далее), чтобы продолжить установку.



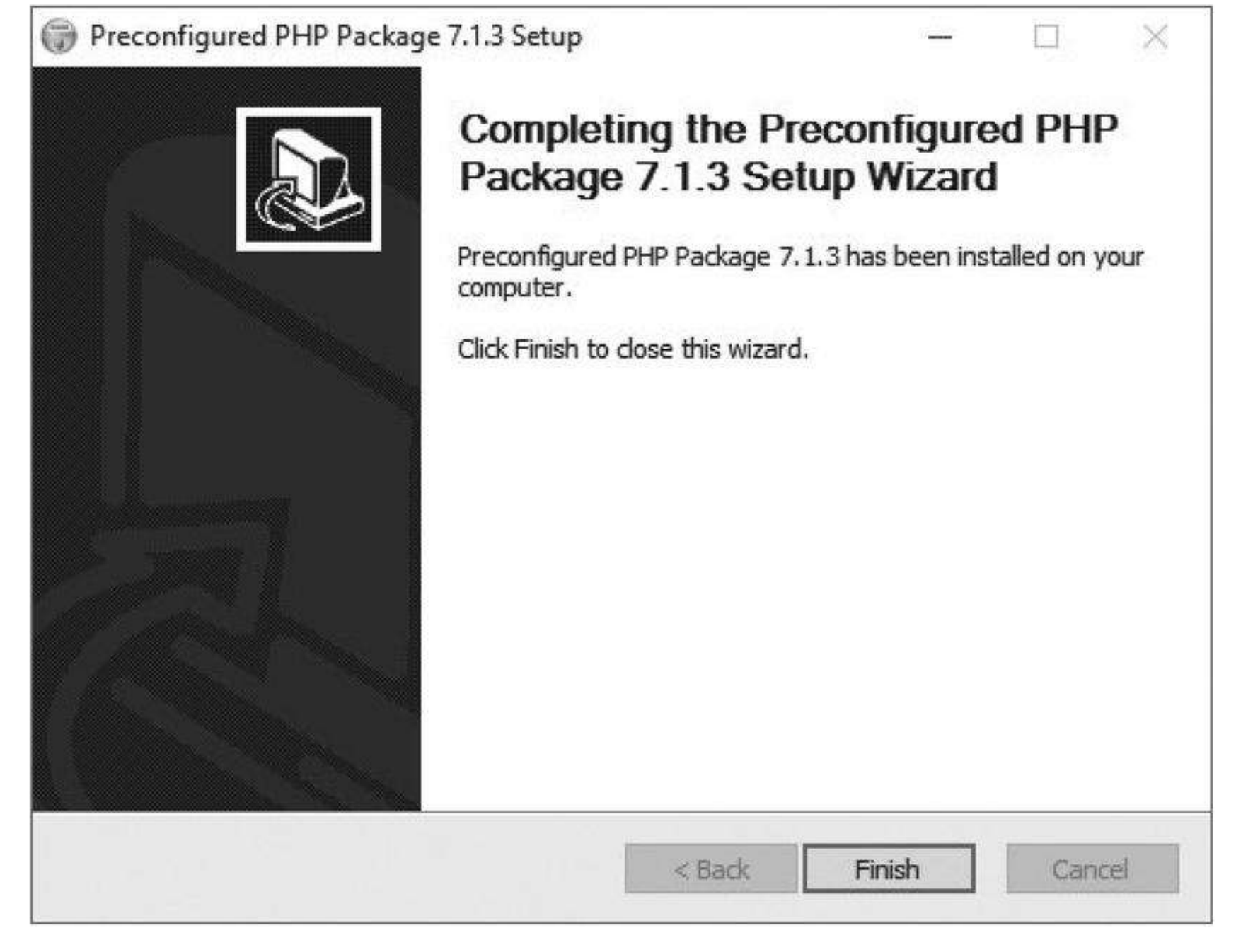
4. Оставьте без изменений предложенный каталог установки, *C:\Program Files (x86)\PHP7* и нажмите кнопку **Install** (Установить).



Совет

Каталог установки PHP-движка понадобится при настройке веб-сервера Abyss для интеграции с PHP-движком, поэтому запомните этот путь.

5. Наконец, после завершения установки, нажмите кнопку **Finish** (Готово), чтобы закрыть окно мастера установки.



На заметку

На заметку. После завершения установки PHP-движка веб-сервер все еще не может выполнять PHP-сценарии, так как на нем пока не настроена поддержка сценариев и путь к движку PHP-интерпретатора. Все это вы выполните на следующей странице.

Интеграция Abyss и PHP

Веб-сервер Abyss должен быть настроен на поддержку PHP-сценариев и привлекать при их обнаружении PHP-интерпретатор. Настройка производится в консоли веб-сервера Abyss. Необходимо ассоциировать файловое расширение.*php* с PHP-сценариями, а также указать расположение PHP-движка в используемой системе, чтобы обеспечить их интерпретацию.

1. Перейдите по адресу **http://localhost:9999** в окне браузера, чтобы запустить консоль веб-сервера Abyss. Авторизуйтесь, а затем нажмите кнопку **Настройки** (Configure) для доступа к странице настроек.

Подробные инструкции по настройке веб-сервера Abyss доступны в интернете по адресу **aprelium.com/abyssws/start.html**.

2. Щелкните мышью по значку **Параметры сценариев** (Scripting Parameters), чтобы открыть страницу настроек сценариев.

3. Убедитесь, что флажок **Включить выполнение сценариев** (Enable Scripts Execution) установлен, а затем нажмите кнопку **Добавить** (Add) в таблице **Интерпретаторы** (Interpreters), чтобы открыть страницу **Интерпретаторы — Добавить** (Interpreters — Add).

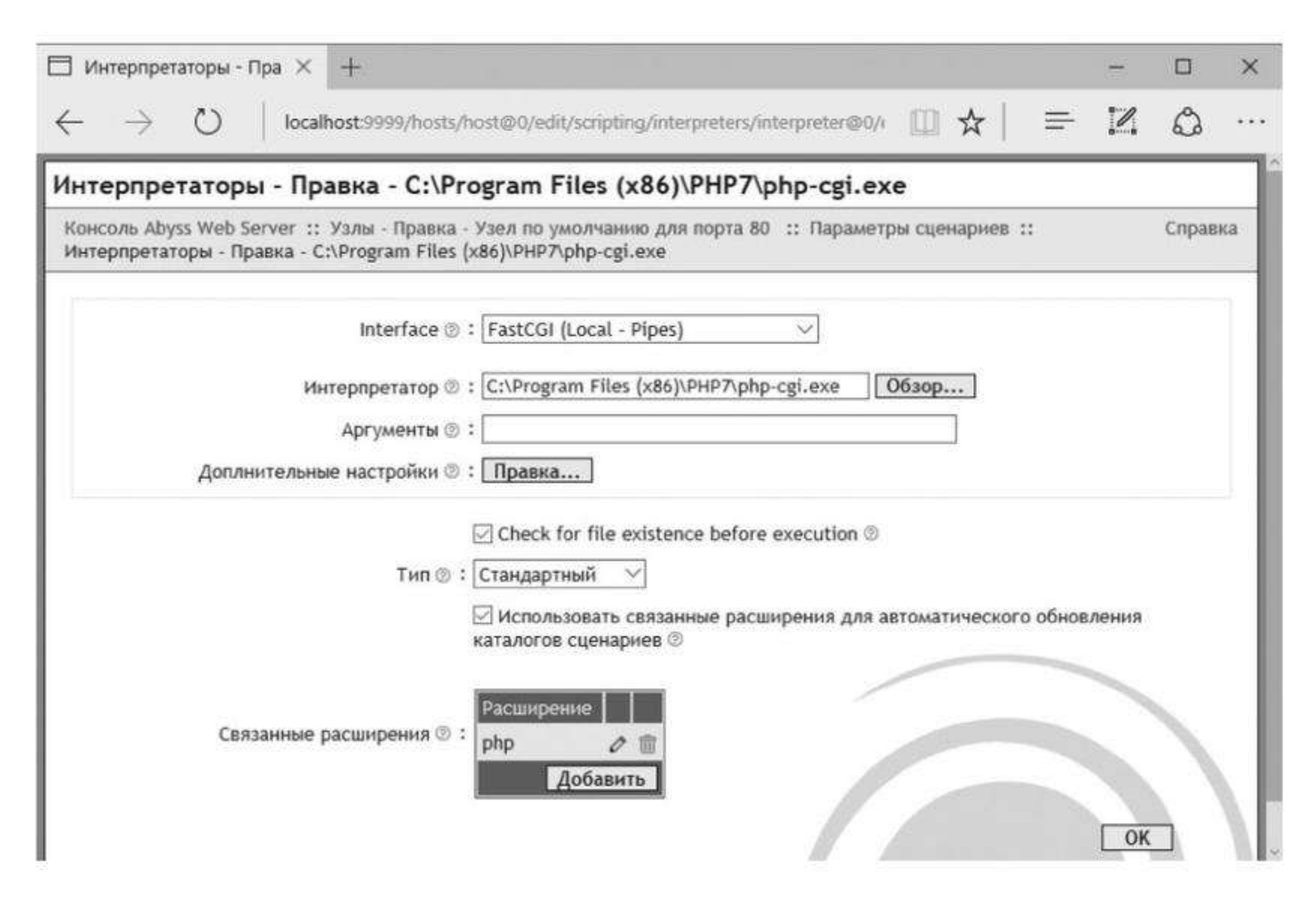
4. В раскрывающемся списке **Interface** (Интерфейс) выберите пункт **FastCGI (Local — Pipes)**.

5. В поле **Интерпретатор** (Interpreter) укажите путь к файлу *php-cgi.exe* в вашей операционной системе.

6. Нажмите кнопку **Добавить** (Add) в таблице **Связанные расширения** (Associated Extensions) и добавьте значение **php** качестве связанного расширения. Страница настроек должна выглядеть так, как показано на следующем рисунке.

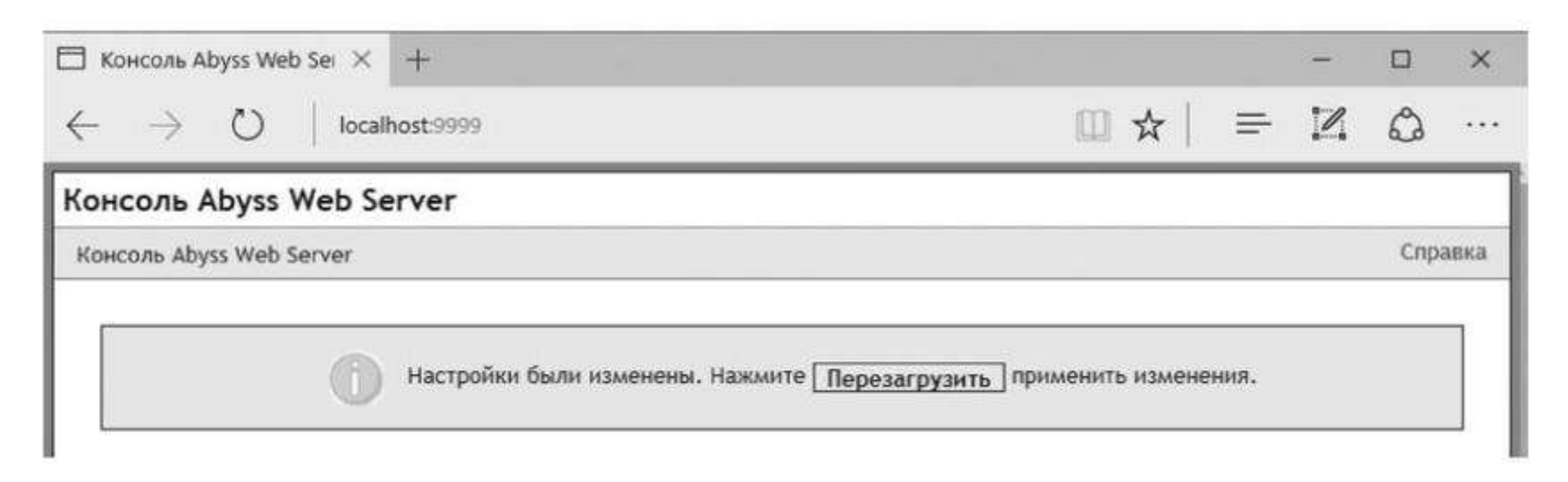
На заметку

Доменное имя **localhost** — псевдоним для доменного IP-адреса **127.0.0.1**, поэтому консоль веб-сервера Abyss также доступна и по адресу **http://127.0.0.1:9999**.



7. Нажмите кнопку **OK** для подтверждения настроек.

8. Нажмите кнопку **Перезагрузить** (Restart) на главной странице настроек консоли, чтобы внесенные изменения вступили в силу.



Внимание

Документы могут интерпретироваться PHP-движком, работающим на веб-сервере через протокол HTTP. Вы не можете напрямую открыть PHP-файл в браузере. Всегда используйте адрес **http://localhost/** для работы с примерами из этой книги.

Веб-сервер Abyss должен быть правильно настроен и запущен в системе, чтобы файлы с расширением *.php* могли обрабатываться PHP-движком. Правильность настройки можно проверить путем создания простого PHP-сценария для обработки в вашем веб-браузере с помощью сервера Abyss.

1. Откройте текстовый редактор и в точности введите указанный ниже код.

```
<?php phpinfo();?>
```

phpinfo.php

2. Сохраните сценарий в файл под именем *phpinfo.php* в каталоге документов сервера Abyss. В нашем случае это путь *C:\Abyss Web Server\htdocs*.

Совет

Файлы всех примеров из этой книге можно бесплатно скачать по адресу: **https://eksmo.ru/files/PHP7_MikeMcGrath.zip**.

3. С точностью введите строку **http://localhost/phpinfo.php** в адресную строку веб-браузера, чтобы увидеть, как веб-сервер Abyss обрабатывает веб-страницу, содержащую информацию о версии PHP-движка.

System	Windows NT SPLIFF-GURU 10.0 build 14393 (Windows 10) i586
Build Date	Mar 14 2017 23:32:46
Compiler	MSVC14 (Visual C++ 2015)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	CGI/FastCGI
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\Program Files (x86)\PHP7\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20160303
PHP Extension	20160303
Zend Extension	320160303

Внимание

PHP-сценарии чувствительны к регистру, поэтому вы должны вводить указанный код сценария, используя только символы в нижнем регистре (т.е. строчными буквами).

Внедрение PHP-кода

PHP-сценарий могут встраиваться в HTML-документы. Это означает, что и PHP-, и HTML-код могут беспроблемно сосуществовать в одном файле. Главное, чтобы внедряемый PHP-код находился внутри тегов `<?php` и`?>`, распознаваемых PHP-движком для интерпретации кода. Как правило, PHP-код выдает содержимое в тело HTML-страницы, которая затем передается в веб-браузер пользователя.

1. Запустите простой текстовый редактор (например, Notepad++) и создайте HTML5-документ со следующим шаблонным кодом и пустым телом (разделом `body`).

hello.php

```
<! DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Начинаем работу с PHP</title>
</head>
<body>

</body>
</html>
```

2. Добавьте следующие теги в тело страницы, чтобы внести в них в дальнейшем PHP-код.

```
<?php

?>
```

3. Теперь внутри PHP-тегов добавьте описательный комментарий и строку кода, выводящего контент в теле страницы. На следующем рисунке показан код документа в программе Notepad++.

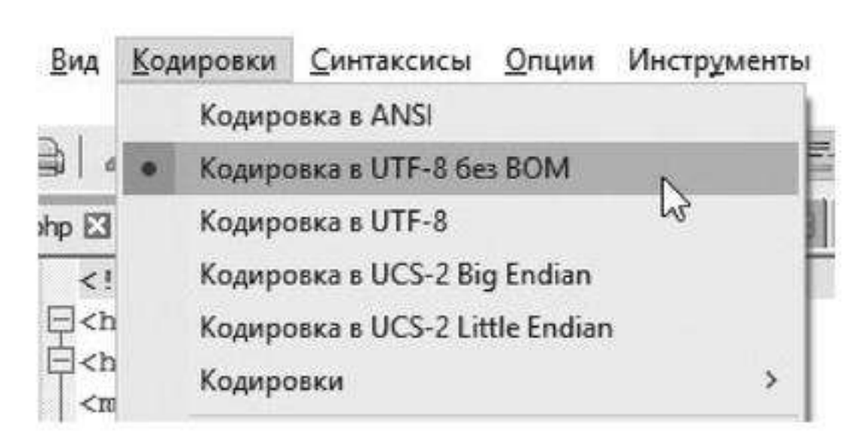
```
# Традиционное приветствие.
echo '<h1>Привет, мир!</h1>';
```

Совет

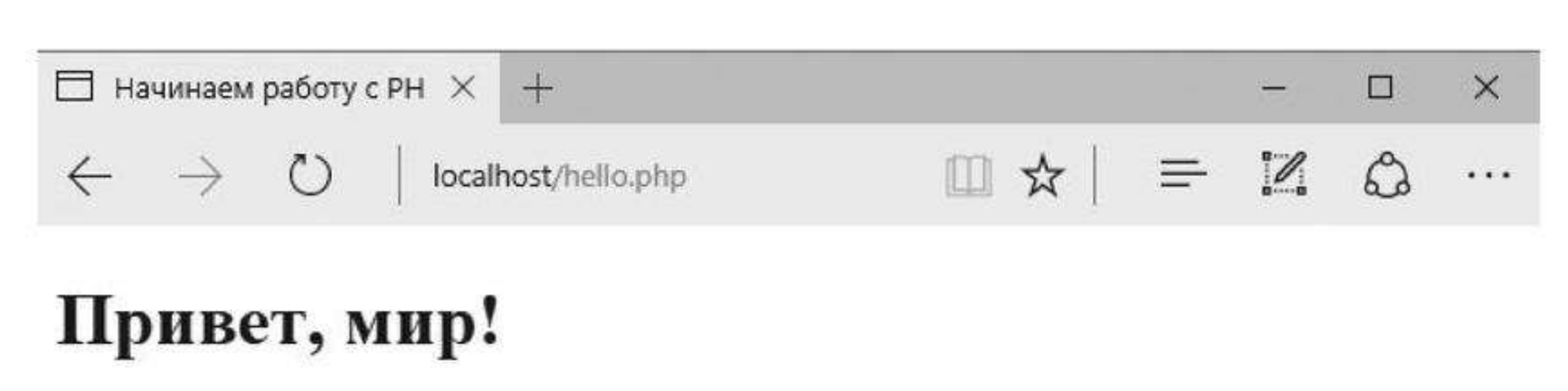
Обратите внимание на то, что описательный комментарий, добавленный внутрь PHP-тегов, носит исключительно информативный характер и не отображается на странице в браузере.

```
1  <!DOCTYPE HTML>
2  <html lang="en">
3  <head>
4  <meta charset="UTF-8">
5  <title>Начинаем работу с PHP</title>
6  </head>
7  <body>
8  <?php
9  #Традиционное приветствие.
10 echo '<h1>Привет, мир!</h1>' ;
11 ?>
12 </body>
13 </html>
```

4 Выберите в свойствах документа кодировку UTF-8, а затем сохраните его под именем *hello.php* в каталог */htdocs* сервера Abyss.



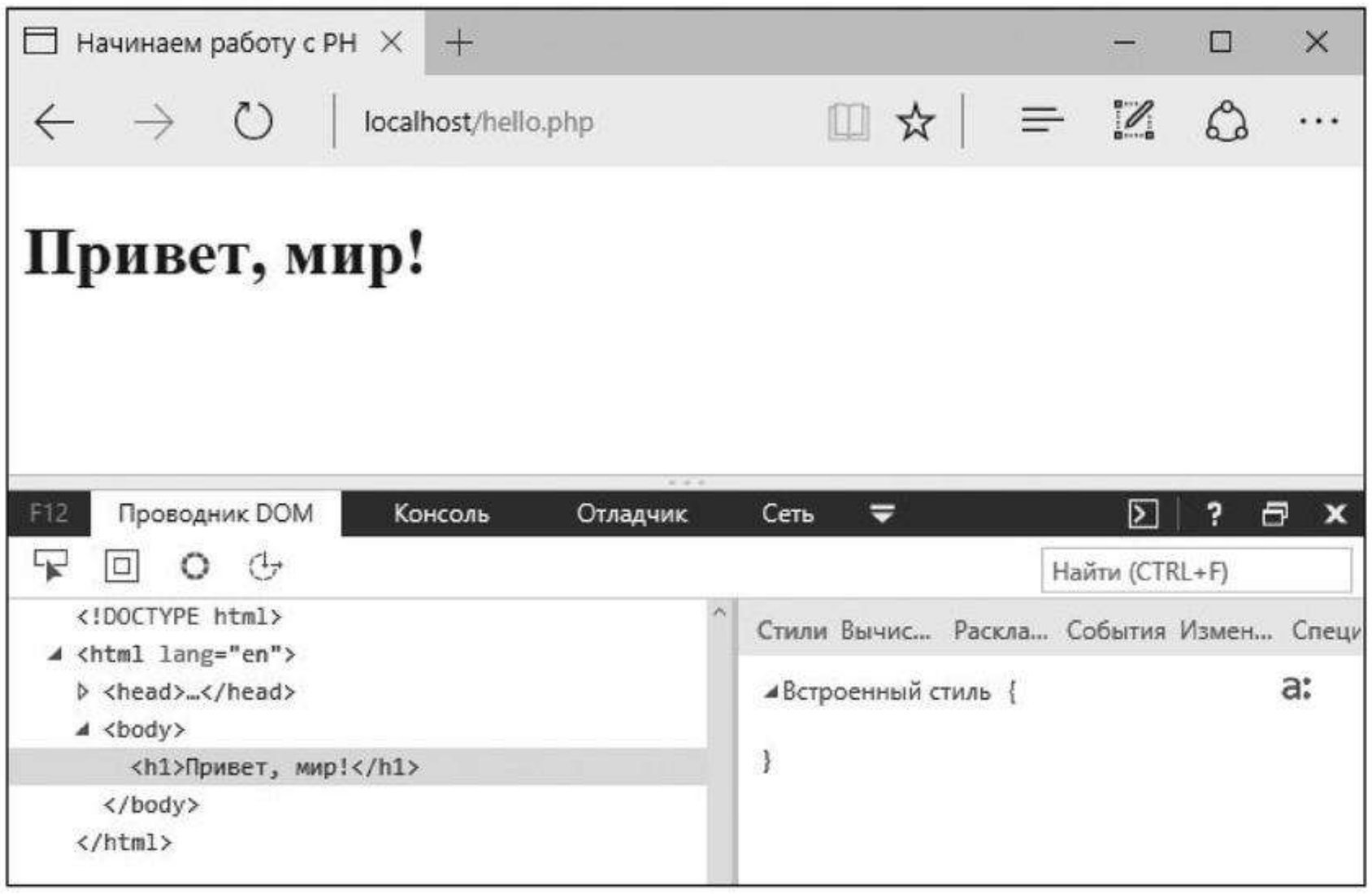
5 Далее введите адрес **http://localhost/hello.php** в адресной строке веб-браузера, чтобы увидеть, как веб-сервер Abyss обрабатывает веб-страницу, содержащую контент, созданный внедренным PHP-кодом.



Внимание

Программа Блокнот (Notepad) в операционной системе Windows автоматически добавляет скрытый маркер последовательности байтов (Byte Order Mark, BOM) в файл, в то время как другие редакторы (например, Notepad++, как показано выше) позволяют его не использовать. Notepad++ — это бесплатный текстовый редактор для программистов, доступный на сайте **notepad-plus-plus.org**.

6 Выберите команду **Просмотреть источник** (View Source) на странице в веб-браузере, чтобы увидеть, что PHP-движок разместил контент в теле страницы, включая HTML-теги заголовков, `<h1>` и `</h1>`.



На заметку

На заметку. Обратите внимание, что PHP-инструкция `echo` в точности выводит все содержимое, заключенное в одиночные кавычки (`'`).

Аналогичным образом PHP-код может внедряться в документы более ранних версий языка HTML. Другие примеры в этой книге приведены в сокращенном виде: указывается только внедряемый PHP-код, а шаблонный HTML-код опускается.

Правила написания сценариев

Оформление тегов

Когда PHP-движок получает входные данные с веб-сервера, он считывает их сверху вниз, во время процесса, называемого синтаксическим анализом или парсингом. В процессе синтаксического анализа PHP-движок (т.е. анализатор, или парсер) ищет открывающие и закрывающие теги `<?php` и `?>` и определяет, что контент между этими тегами — код сценария, который он должен интерпретировать. Код за пределами тегов `<?php` и `?>` полностью игнорируется,

что позволяет использовать в PHP-файлах контент разного типа и допускает встраивание PHP-кода в HTML-код следующим образом.

Совет

Внедряемый PHP-код всегда должен оборачиваться в открывающие и закрывающие теги <?php и ?>.

```
<p>Игнорируется PHP-движком и отображается в браузере</p>
<?php echo 'Код сценария, который должен быть
проанализирован';?>
<p>Также игнорируется PHP-движком и отображается
в браузере</p>
```

Для корректного отображения страниц очень важно, чтобы при внедрении в HTML-документ все ваши PHP-сценарии заключались в открывающие и закрывающие теги <?php и ?>.

Единственное исключение из этого правила касается случаев, когда PHP-сценарий написан в отдельном PHP-файле, который содержит только PHP-код. В этом случае желательно опускать закрывающий тег ?>.

```
<?php   echo 'Вывести это сначала';
        echo 'Вывести это потом';
```

Когда PHP-код используется для вывода только одной текстовой строки в HTML-документе, вы можете использовать сокращенный вариант написания открывающих и закрывающих тегов: <?= и ?>. Так, строка

```
<?= 'Привет!'; ?>
```

эквивалентна строке

```
<?php echo 'Привет!'; ?>
```

Более сложные PHP-сценарии могут выводить текст только при соблюдении определенных условий, как показано в следующем примере.

```
<?php if ($expression == true): ?>
     Вывести этот текст, только если выражение истинно.
<?php else: ?>
     В противном случае вывести этот текст.
<?php endif; ?>
```

На заметку

Соблюдение условий полностью объясняется и демонстрируется в главе 3 этой книги.

Оформление инструкций

Каждая инструкция в языке PHP должна завершаться символом точки с запятой (;) аналогично тому, как каждое предложение в русском языке завершается точкой. Точка с запятой распознается анализатором как метка конца отдельной инструкции, которую он должен интерпретировать. Таким образом, блок PHP-кода, содержащий две инструкции, может выглядеть следующим образом:

```
<?php echo 'Первая инструкция'; echo 'Вторая инструкция';?>
```

Закрывающий тег ?> блока PHP-кода автоматически подразумевает точку с запятой, поэтому при необходимости вы можете опустить точку с запятой, завершая последнюю инструкцию в блоке PHP-кода:

```
<?php echo 'Первая инструкция'; echo 'Вторая инструкция'?>
```


Совет

Никаких проблем с завершением последней инструкции в блоке PHP-кода не возникнет, поэтому при желании вы можете так поступить.

Оформление комментариев

Целесообразно добавлять комментарии в код PHP-сценариев, чтобы не возникало проблем в его понимании другими программистами, а также и для вас самих при чтении кода спустя какое-то время. Все пробелы и текст комментариев полностью игнорируются PHP-анализатором, поэтому вы можете добавлять неограниченное количество комментариев без снижения производительности.

Однострочные комментарии могут начинаться с хэш-символа # или, как альтернатива, с двух символов слеша //.

Многострочные (блок) комментарии заключаются в символы /* и */, так же, как и в языке программирования C.

```
<?php
     echo 'Первая инструкция'; // Однострочный комментарий.

     /* Это многострочный комментарий,
     Содержащий две строки текста. */

     echo 'Вторая инструкция';

     echo 'Финальная инструкция';
     # Другой однострочный комментарий.
?>
```

Совет

Однострочные комментарии предваряются символами // так же, как и в языке программирования C ++. А символ # для однострочных комментариев используется в командной оболочке Bash в среде Unix/Linux.

Заключение

- PHP — это сценарный язык, особенно подходящий для веб-разработки, так как может внедряться в HTML-код.
- PHP-код выполняется на стороне сервера в облаке, в отличие от сценариев JavaScript, которые выполняются на стороне клиента в браузере.
- Если веб-страница содержит сценарий, веб-сервер сначала вызывает PHP-движок для обработки кода перед отправкой ответа в веб-браузер.
- Локальная среда разработки может быть настроена путем установки веб-сервера и PHP-движка на вашем собственном компьютере.
- URL-адрес **http://localhost** — это псевдоним для числового IP-адреса **http://127.0.0.1**.
- Для доступа к консоли веб-сервера Abyss требуется перейти по адресу **http://localhost:9999** и указать логин и пароль для авторизации.

- Для простой и быстрой установки PHP-движка рекомендуется использовать предварительно сконфигурированный пакет дистрибутива PHP для веб-сервера Abyss.
- Веб-сервер должен быть настроен специальным образом (путь к движку, ассоциация расширения и настройки обработки) на поддержку сценариев и передачу их PHP-движку.
- Инструкция с PHP-кодом `phpinfo()` может использоваться для вывода веб-страницы, содержащей информацию о версии установленного PHP-движка.
- Документы, содержащие PHP-сценарий, лучше всего сохранять в Юникод-кодировке UTF-8, в том числе и для поддержки русского языка.
- Команда **echo** языка PHP в точности выводит текстовый контент, содержащийся в последующих кавычках.
- Любой внедряемый PHP-код должен быть заключен в теги `<?php` и `?>`, чтобы его мог обработать PHP-движок.
- Каждая инструкция в языке PHP должна завершаться символами ; (точка с запятой).
- Однострочные комментарии могут начинаться с хэш-символа # или двух символов слеша //.
- Многострочный блок комментариев оборачивается в символы /* и */.

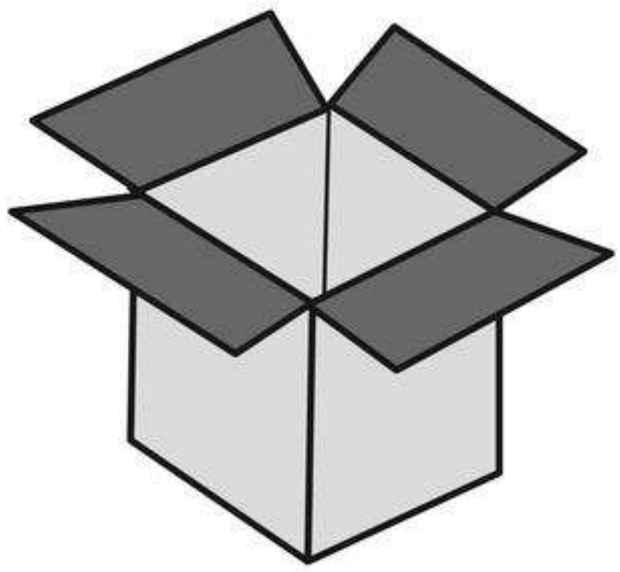
2

Хранение значений

Эта глава познакомит вас с различными контейнерами PHP, в которых могут храниться данные.

- Переменные
- Заключение строк в кавычки
- Массивы
- Сортировка массивов
- Размерность массивов
- Типы данных
- Константы
- Суперглобальные переменные
- Заключение

Переменные



Переменная — это именованный контейнер в PHP-коде, в которой могут быть сохранены данные. К сохраненному значению можно обращаться, указывая имя переменной, а также изменять его в процессе выполнения сценария.

Будучи разработчиком сценария, вы можете присвоить переменной любое имя, при условии соблюдения следующих трех соглашений о присвоении имен.

- Имена должны начинаться с символа доллара (`$`), например `$name`.
- В именах допускается использовать латинские буквы, цифры и символы подчеркивания, но не пробелы, например `$subtotal_1`.
- Сразу после символа $ должна указываться буква или символ подчеркивания, но не число.

Внимание

Внимание! В языке PHP `$this` — это специальная переменная, поэтому вы не можете использовать слово `this` в качестве имени переменной.

Обратите внимание, что имена переменных в языке PHP чувствительны к регистру, поэтому `$name`, `$Name` и `$NAME` — три отдельные независимые переменные.

В языке PHP переменные «слабо типизированы». Это означает, что они могут содержать данные любого типа, в отличие от «сильно типизированных» переменных в некоторых других языках, где тип данных необходимо указывать при создании переменной.

На заметку

Переменные, доступные во всем сценарии, должны иметь уникальные имена. См. раздел «Область видимости» главы 5 для получения дополнительной информации об области видимости переменных.

Таким образом, переменная в PHP-сценарии может успешно содержать как целое число или число с плавающей запятой, так и строку текстовых символов или логическое значение TRUE/FALSE, а также объект или пустое значение NULL.

Переменная создается в PHP-сценарии простым присвоением ей имени. Переменной может быть назначено начальное значение (такая переменная называется инициализированной) с помощью оператора присваивания =. Такие инструкции, как и все остальные в языке PHP, должны заканчиваться точкой с запятой:

```
$body_temp = 36.6;
```

Значение, содержащееся в переменной, можно отобразить, обратившись к переменной по имени, например так:

```
echo $body_temp;
```

Часто значение переменной выводится в составе строки, при этом сама строка и имя переменной заключаются в двойные кавычки.

Внимание

Не следует путать предназначение одиночных и двойных кавычек. Помните, что PHP-движок обрабатывает переменные в составе только тех строк, которые заключены в двойные кавычки.

```
echo "Температура тела составляет $body_temp градусов
Цельсия";
```

Двойные кавычки инструктируют PHP-движок анализировать всю строку и заменить имена переменных на их сохраненные значения. Это не работает, если строка заключена в одиночные кавычки!

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

variable.php

```
<?php
     # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, создающую и инициализирующую переменную.

```
$body_temp = 36.6;
```

3. Затем добавьте инструкцию, выводящую отдельное значение переменной.

```
echo $body_temp;
```

4. Затем добавьте инструкцию, выводящую значение переменной в составе строки, заключив ее в двойные кавычки.

```
echo "<p>Температура тела составляет $body_temp
градусов Цельсия ";
```

Совет

Обратите внимание на то, что переменные, создаваемые в основном теле сценария, как в данном примере, доступны «глобально», т.е. внутри всего PHP-сценария.

5. Добавьте следующее инструкцию, присваивающую переменной новое значение.

```
$body_temp = 97.88;
```

6. И, наконец, добавьте инструкцию, выводящую новое значение переменной в составе строки.

```
echo " ($body_temp градусов Фаренгейта)</p>";
```

7. Сохраните документ в каталог */htdocs* вашего веб-сервера под именем *variable.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть значения переменных.

36.6

Температура тела составляет 36.6 градусов Цельсия (97.88 градусов Фаренгейта)

На заметку

Каждая инструкция в языке PHP должна завершаться символом точки с запятой (;) аналогично тому, как каждое предложение в русском языке завершается точкой.

Заключение строк в кавычки

«Строка» текста может быть сохранена в качестве значения переменной практически так же, как и числовое значение, только текст нужно заключить в кавычки, обозначающие начало и конец значения. Для этой цели могут быть использованы как одиночные, так и двойные кавычки, но необходимо использовать кавычки одного типа, обозначая начало и конец текстовой строки. Например, обе следующих инструкции допустимы в качестве присвоения значений переменным:

```
$song_title = "Летний блюз";
$song_title = 'Летний блюз';
```

Если нужно использовать текстовую строку, которая сама по себе содержит кавычки, вы можете указывать их вместе с символом обратного слеша, \, или использовать в строке кавычки другого типа. Так, обе следующих инструкции содержат кавычки и допустимы в качестве присвоения значений переменным:

```
$song_title = "\"Летняя пора\" — ария Джорджа Гершвина";
$song_title = '"Летняя пора" — ария Джорджа Гершвина';
```

Второй способ, касающийся использования кавычек другого типа, более прост для восприятия и поэтому применяется в примерах этой книге.

На заметку

Имена переменных не могут содержать пробелы, вместо них используется символ подчеркивания.

Строковые значения могут выводиться в составе строки текста, при этом составная строка должна быть заключена в двойные кавычки, например:

```
echo "Классическая мелодия $song_title очень популярна";
```

Напоминаем, что двойные кавычки инструктируют PHP-движок анализировать составную строку и заменять имена переменных их сохраненными значениями. Этот способ не сработает, если строка заключена в одиночные кавычки!

Строковые значения могут быть соединены вместе («конкатенированы») в одну строку с помощью оператора конкатенации:

```
$hi = 'Привет';
$bye = 'Пока';
$song_title = $hi. $bye;     # 'ПриветПока'
```

Пробелы и знаки препинания также могут быть добавлены при конкатенации строк. Строка кода, показанная выше, изменена в следующем примере и включает запятую с пробелом:

```
$song _ title = $hi. ', '. $bye; # 'Привет, Пока'
```

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

string.php

```
<?php
     # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, создающую и инициализирующую две переменные.

```
$phrase = 'Правда редко бывает чистой';
$author = 'Оскар Уайльд';
```

3. Затем добавьте инструкцию, выводящую отдельное значение переменной.

```
echo $phrase;
```

Совет

Вы можете также использовать краткую форму записи инструкцию конкатенации с помощью оператора присваивания с конкатенацией, `.=`. В этом случае вы можете записать инструкцию `$a = $a. $b` в краткой форме `$a .= $b`.

4. Затем добавьте инструкцию, выводящую значение переменной в составе строки. Не забывайте, что присваиваемое значение указывается в двойных кавычках.

```
echo "<p>Часто говорят, что <q>$phrase</q> </p>";
```

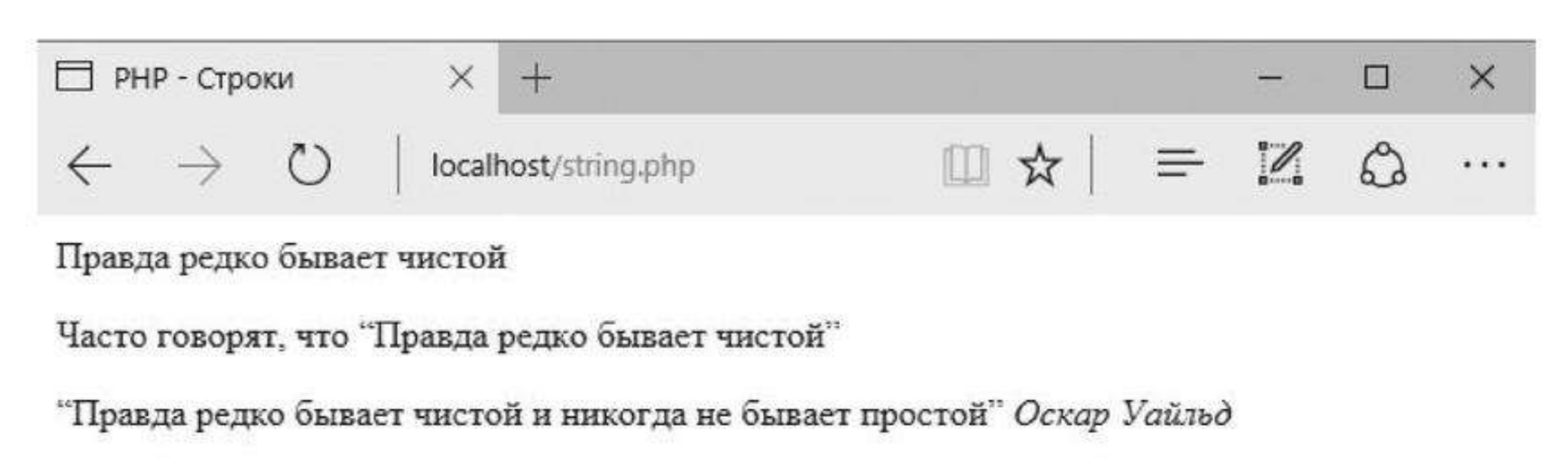
5. Добавьте инструкцию конкатенации переменной и строки текста.

```
$phrase = $phrase. ' и никогда не бывает простой';
```

6. И, наконец, добавьте инструкцию, выводящую текущие значения переменных в составе строки.

```
echo "<p><q>$phrase</q><cite>$author</cite></p>";
```

7 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *string.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть значения переменных.

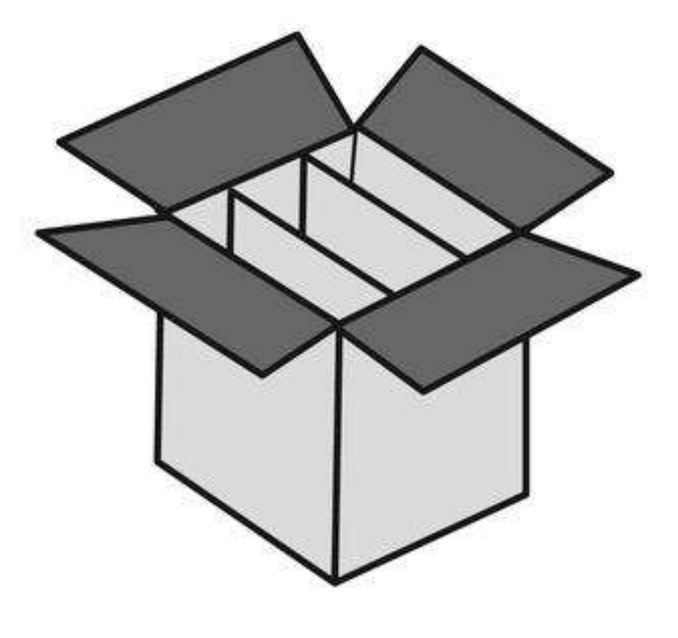


Совет

Примите для себя правило именовать переменные в одном стиле, например строчными буквами (`$my_var`) или в «ГорбатомРегистре» (`$myVar`).

Массивы

Индексированные массивы



Переменная массива может хранить несколько элементов данных в последовательном массиве «элементов», которые нумеруются (индексируются), начиная с нуля. Таким образом, первое значение хранится в элементе массива под номером ноль.

К значениям отдельных элементов массива можно обращаться, указывая имя переменной, а затем номер (индекс) элемента в квадратных скобках.

Например, код `$days[0]` ссылается на значение, сохраненное в первом элементе переменной массива с именем `days`. Аналогичным образом код `$days[1]` ссылается на значение, сохраненное во втором элементе массива, и так далее.

Массивы могут быть созданы группами инструкций, которые последовательно присваивают значения элементам, или с помощью одной инструкциии, содержащей PHP-функцию `array()` для создания элементов и присваивания им значений. Первый способ выглядит так:

```
$days[] = 'Понедельник'; $days[] = 'Вторник';
$days[] = 'Среда';
```

А второй так:

```
$days = array('Понедельник', 'Вторник', 'Среда');
```

Обе инструкции создают массив, в котором к каждому значению, хранящемуся в каждом элементе, можно обращаться, используя соответствующий номер (индекс), например `$days[0]`.

На заметку

Квадратные скобки обозначают «элемент» и не требуются после имени переменной при создании массива с помощью функции `array()`. Они необходимы только при обращении к элементу.

Ассоциативные массивы

При создании массива для каждого элемента можно дополнительно указать имя ключа, который применяется для обращения к значению соответствующего элемента:

```
$months['янв'] = 'Январь';
$months['фев'] = 'Февраль';
$months['мар'] = 'Март';
```

Или так:

```
$months =
array('янв' => 'Январь', 'фев' => 'Февраль', 'мар' => 'Март');
```

В обоих случаях создается массив, к каждому из значений элементов которого можно обращаться, используя соответствующее имя ключа, например `$months['янв']`.

Перебор массивов

Все элементы массива могут перебираться с помощью цикла `foreach`. На каждой итерации значение текущего элемента присваивается переменной:

```
foreach($months as $value)
{
   # Использовать текущее значение $value во время каждой
   # итерации.
}
```

Совет

Конструкция `foreach` используется специально для массивов и представляет собой простой способ перебрать все ключи и значения элементов массива. См. главу 4, чтобы получить дополнительные сведения о циклах.

К ключам также можно получить доступ во время каждой итерации цикла следующим образом:

```
foreach($months as $key => $value)
{
   # Использовать текущие ключи $key и значение $value
   # во время каждой итерации.
}
```

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

array.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, создающую и инициализирующую переменную массива со значениями.

```
$days = array('Понедельник', 'Вторник', 'Среда');
```

3. Затем добавьте инструкцию, выводящую значения всех элементов массива в виде маркированного списка.

```
foreach($days as $value) {echo "&bull; $value ";}
```

Совет

Вы можете установить ключ первого элемента равным 1, чтобы индексирование начиналось с единицы, а не нуля, следующим образом: `$months = array(1 => 'Январь', 'Февраль', Март');`.

4. Затем добавьте инструкцию, создающую и инициализирующую переменную массива, как с ключами, так и значениями.

```
$months = array('янв' => 'Январь', 'фев' => 'Февраль',
'мар' => 'Март');
```

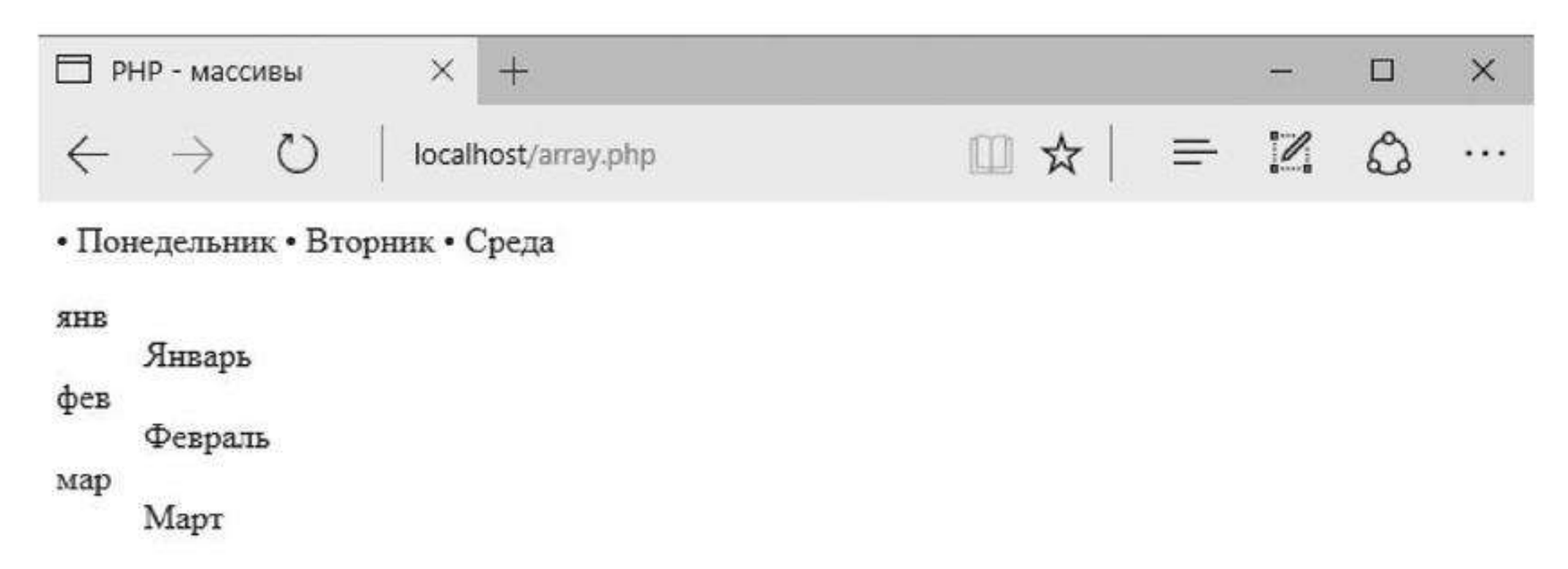
Внимание

Если вы указываете имена ключей, вы не можете обращаться к элементам по их индексу. Кроме того, если вы укажете одно и то же имя ключа дважды, второе значение заменит первое.

5. И, наконец, добавьте инструкцию для вывода ключа и значения всех элементов в виде списка определений.

```
echo '<dl>';
foreach($months as $key => $value)
{echo "<dt>$key<dd>$value";}
echo '</dl>';
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *array.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы просмотреть значения переменной массива.



Совет

Вы можете легко создавать массивы, содержащие последовательные числа или буквы с помощью функции `range()`, например так:

```
$six = range(1, 6);
$a_z = range('a','z');
```

Сортировка массивов

Язык PHP предоставляет две удобные функции для преобразования переменных массивов и строк, поскольку они часто используются вместе. Функция `implode()` преобразует массив в строку и добавляет определенный разделитель после каждого значения. Обычно она используется для создания списка значений, разделенных запятыми (CSV*), например:

```
$csv_list = implode(', ', $array);
```

С другой стороны, функция `explode()` преобразует строку в массив. При этом указывается разделитель, используемый для разбиения строки. Следующий код создает массив из списка элементов, разделенных запятыми:

```
$array = explode(', '. $csv_list);
```

PHP также предоставляет три полезные функции для сортировки элементов массива по возрастанию, по буквам и цифрам (А–Я, 1–9).

- Функция `sort()` осуществляет сортировку по значению с отбрасыванием исходного ключа.
- Функция `asort()` выполняет сортировку по значению с сохранением исходного ключа.
- Функция `ksort()` выполняет сортировку по ключу.

Существующие значения массива сортируются с помощью простой инструкции, к примеру так:

```
$makers = array('ВАЗ', 'УАЗ', 'ГАЗ');

sort($makers);
```

* Сокращение от англ. Comma-Separated Values — значения, разделенные запятыми. — *Прим. перев.*

Элементы массива также могут быть отсортированы в алфавитном порядке по убыванию (9–1, Я–А) с помощью трех аналогичных функций.

- Функция `rsort()` выполняет сортировку по значению с отбрасыванием исходного ключа.
- Функция `arsort()` выполняет сортировку по значению с сохранением исходного ключа.
- Функция `krsort()` осуществляет сортировку по ключу.

Совет

Совет. Убедиться, является ли переменная массивом, можно с помощью функции `is_array()`. Например, `is_array($var);`.

Важно учитывать, что присвоенные имена ключей будут отброшены при использовании функций `sort()` и `rsort()`, поэтому их следует использовать только в тех случаях, если взаимосвязи ключ/значение не имеют существенного значения. В противном случае следует использовать функции, учитывающие имена ключей — `asort()` и `arsort()` соответственно.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

sort.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, создающую и инициализирующую переменную массива с именами ключей и значениями.

```
$cars = array('УАЗ' => 'Патриот', 'ГАЗ' => 'Сайбер',
'ВАЗ' => 'Нива');
```

3. Затем добавьте инструкции, выводящие все ключи и значения массива в виде маркированного списка внутри списка определений.

```
echo '<dl><dt>Исходный порядок элементов:<dd>';
foreach($cars as $key => $value)
{echo ' &bull; ', $key. ' '. $value;}
```

Совет

Совет. Для подсчета количества элементов в массиве вы можете использовать функцию `count()`. Например, так: `$num = count($array);`.

4 Затем добавьте инструкции, выводящие содержимое массива, отсортированное по значению.

```
asort($cars);
echo '<dt>Сортировка по значению:<dd>';
foreach($cars as $key => $value)
{echo ' &bull; ', $key. ' '. $value;}
```

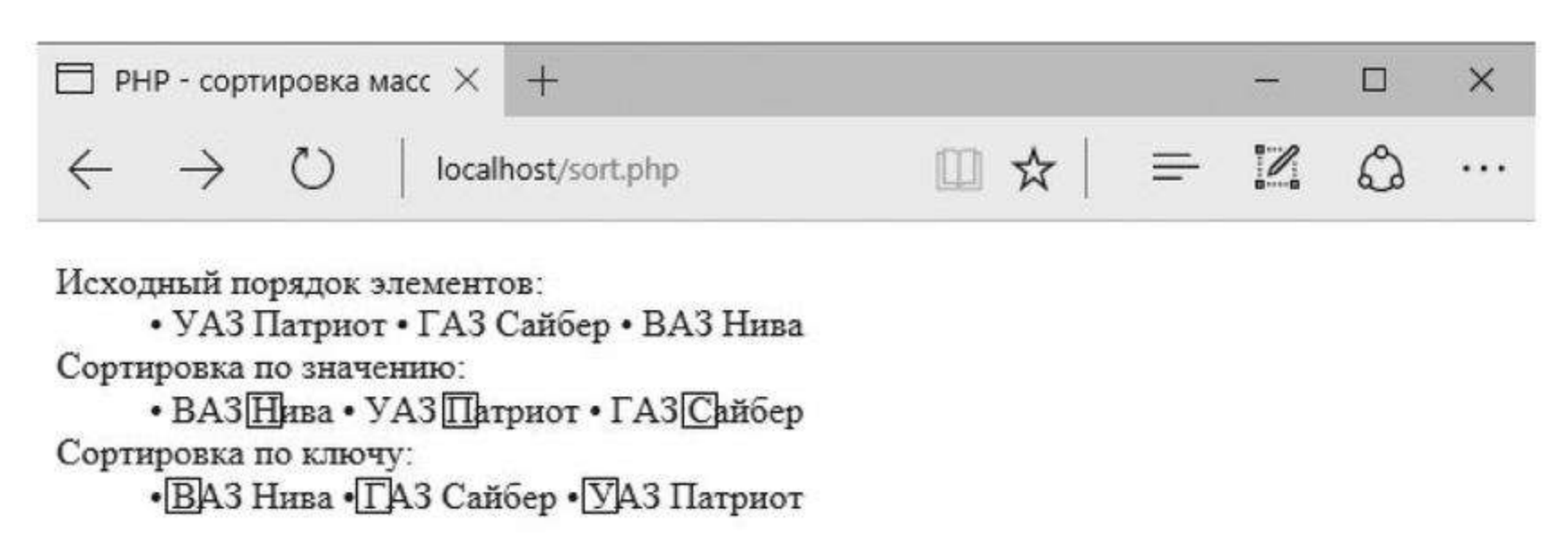
5 И, наконец, добавьте инструкции, выводящие содержимое массива, отсортированное по ключу.

```
ksort($cars);
echo '<dt>Сортировка по ключу:<dd>';
foreach($cars as $key => $value)
{echo ' &bull; ', $key. ' '. $value;}
echo '</dl>';
```


Внимание

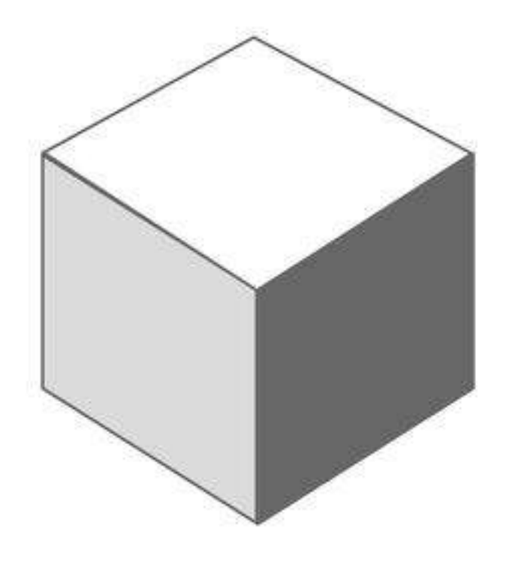
Не используйте функции `sort()` и `rsort()`, если ключи в данном массиве важны!

6 Сохраните документ в каталог */htdocs* вашего веб-сервера под именем *sort.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть отсортированные значения массива.



Размерность массивов

Нумерацию элементов массива корректнее называть *индексацией элементов массива*. Так как нумерация элементов начинается с нуля, иногда употребляют термин «нулевая индексация». Массив, созданный с одним индексом, называется одномерным.Элементы в нем расположены в одной строке:



Значение элемента	А	Б	В	Г	Д
Индексный номер	[0]	[1]	[2]	[3]	[4]

В массивах также могут использоваться несколько индексов — для представления нескольких размерностей. Массив с двумя индексами называется двумерным (или многомерным), в котором элементы расположены на нескольких строках:

Первый индекс	[0]	А	Б	В	Г	Д
	[1]	Е	Ж	З	И	К
Второй индекс		[0]	[1]	[2]	[3]	[4]

В случае с многомерными массивами к значению, содержащемуся в каждом элементе, обращение происходит посредством указания каждого индекса. Например, в двумерном массиве, показанном выше, элемент [1] [2] содержит букву «З».

Двумерные массивы полезны для хранения табличной информации, например значений системы координат XY. А использование трехмерного массива (с тремя индексами) позволяет хранить данные системы координат XYZ.

Создание многомерного массива в PHP — это, по сути, создание внешнего массива, элементы которого сами являются массивами. Разумеется, можно дополнительно присвоить имя ключа каждому элементу — в данном случае для представления каждого внутреннего массива. К отдельным значениям можно обращаться с помощью имени ключа и внутреннего индекса:

```
$matrix['Письмо'][0]
```

Внимание

Трехмерные и более сложные массивы существенно затрудняют чтение исходного кода и повышают вероятность возникновения ошибок.

При включении в строку переменные массива, содержащие имена ключей в кавычках, необходимо всегда заключать в фигурные скобки:

```
echo "Значение элемента: {$matrix['Письмо'][0]}";
```

Все внутренние массивы в многомерном массиве можно перебрать с помощью цикла `foreach`. Вложенный цикл `foreach` позволяет, в свою очередь, перебирать каждый отдельный элемент.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

matrix.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, создающую и инициализирующую два обычных массива и двумерный массив с именами ключей.

```
$letters = array('А', 'Б', 'В');
$numbers = array(1, 2, 3); $matrix =
array('Буква' => $letters, 'Число' => $numbers);
```

3. Далее добавьте инструкцию, выводящую отдельное значение переменной.

```
echo "<p>Начало: {$matrix['Буква'][0]} </p>";
```

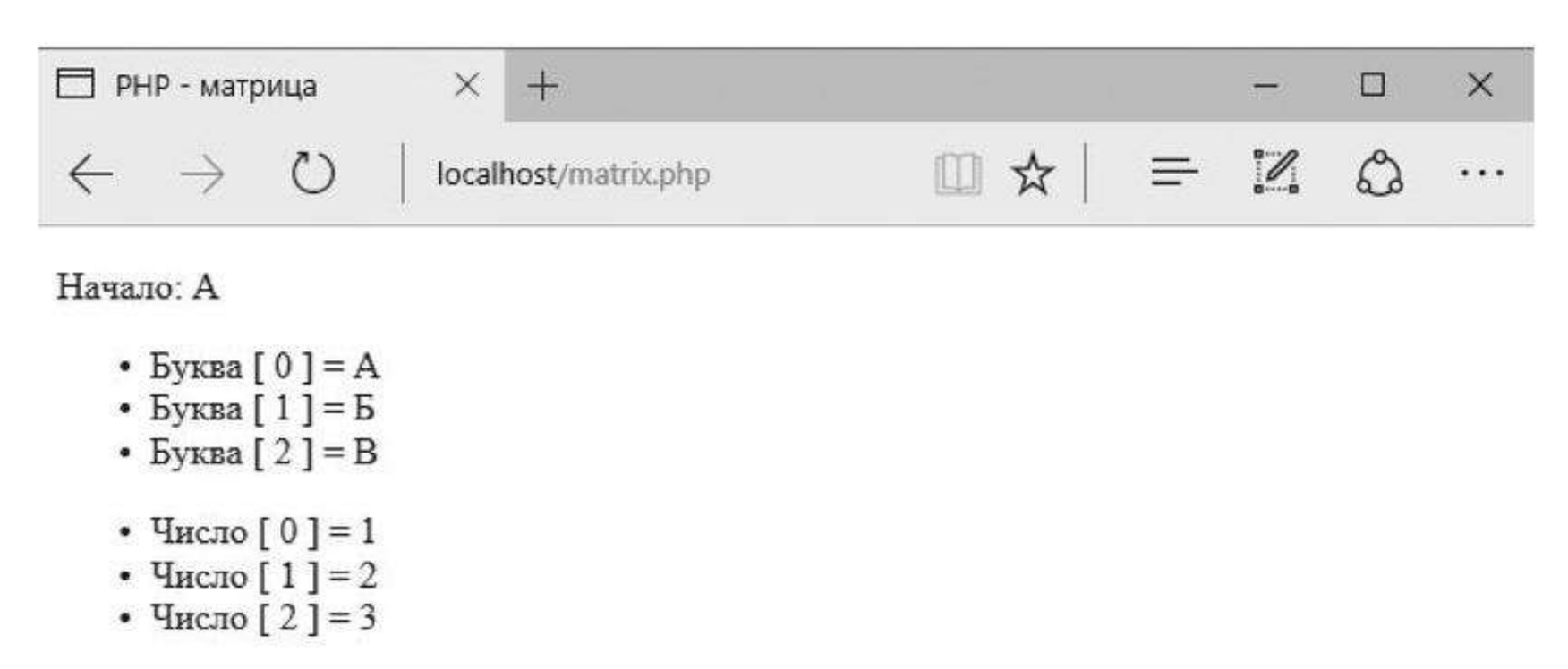
Совет

Многомерные массивы распространены шире, чем вы могли бы подумать. Например, HTML-форма, допускающая выбор нескольких пунктов из списка, реализует такое поведение на основе многомерного массива.

4 И, наконец, добавьте инструкцию для вывода ключа и значения всех элементов массива в виде двух маркированных списков.

```
foreach($matrix as $array => $list)
{
 echo '<ul>';
 foreach($list as $key => $value)
 {echo "<li>$array [$key] = $value ";}
 echo '</ul>';
}
```

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *matrix.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы просмотреть значения переменной.



На заметку

Для вывода в составе строки переменных массива, в которых используются ключи, заключенные в кавычки, следует поместить их в фигурные скобки.

Типы данных

В языке PHP переменные могут хранить данные различного типа, поэтому важно точно знать, что представляет собой каждый тип данных.

- **Строка** — последовательность символов, в которой каждый символ имеет размер в один байт, вплоть до максимальной длины в 2 Гб*. Строки,

* Ограничение касается 32-разрядных систем. В 64-разрядных системах нет каких-либо достижимых ограничений для длины строки. — *Прим. перев.*

заключенные в одиночные кавычки, рассматриваются в качестве литералов, в то время как строки в двойных кавычках будут интерпретироваться (специальные символы, значения переменных и т.п.).

- **Целое число** — недробное число в диапазоне от –2147483648 до 2147483647, указанное в десятичной (основание 10), шестнадцатеричной (основание 16 — с префиксом 0x) или восьмеричной (основание 8 — с префиксом 0) системе.
- **Числа с плавающей запятой** — компьютерная реализация экспоненциальной записи чисел. Также известны как «числа двойной точности».
- **Булев тип** — логическое выражение, значение которого может быть только истинным (`TRUE`) или ложным (`FALSE`).
- **Массив** — упорядоченная карта значений с несколькими данными, при котором ключи соответствуют значениям. Ключи — это номера индекса (по умолчанию) или явно указанные метки.
- **Объект** — класс, содержащий сохраненные свойства данных и предоставляющий методы обработки данных.
- **Ресурс** — ссылка на внешний ресурс, создаваемая и используемая специальными функциями.
- **`NULL`** — переменная без значения. Это переменная, которая не была инициализирована (ей не было присвоено никакого значения), которой была присвоена константа `NULL` или которая была удалена с помощью функции `unset()`.

Совет

Создание и использование объектов классов в PHP описано в главе 7.

Тип данных любого элемента можно проверить, указав его в качестве параметра функции `gettype()`. Например, если переменная `$num` содержит целочисленное значение, функция `gettype($num)` вернет целое число, а если переменная `$str` содержит строковое значение, функция `gettype($str)` вернет строку, и так далее. Если тип данных не может быть определен, функция `gettype()` вернет «неизвестный тип».

Возможно будет более полезна функция `var_dump()`, которая проверяет тип данных элемента и выводит структурированную информацию о них, включая тип и значение.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

types.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, создающую файловый ресурс и массив.

```
$filestream = fopen('index.html', 'r');
$data = array('PHP', 1, 2.3, TRUE, NULL, array(), new
Directory, $filestream);
```

Совет

Создание и использование файловых ресурсов в полной мере продемонстрировано в главе 8.

3. Теперь добавьте инструкцию для вывода типа данных, хранящегося внутри каждого элемента массива.

```
foreach($data as $type)
{
 var_dump($type);
 echo '<br> ';
}
```

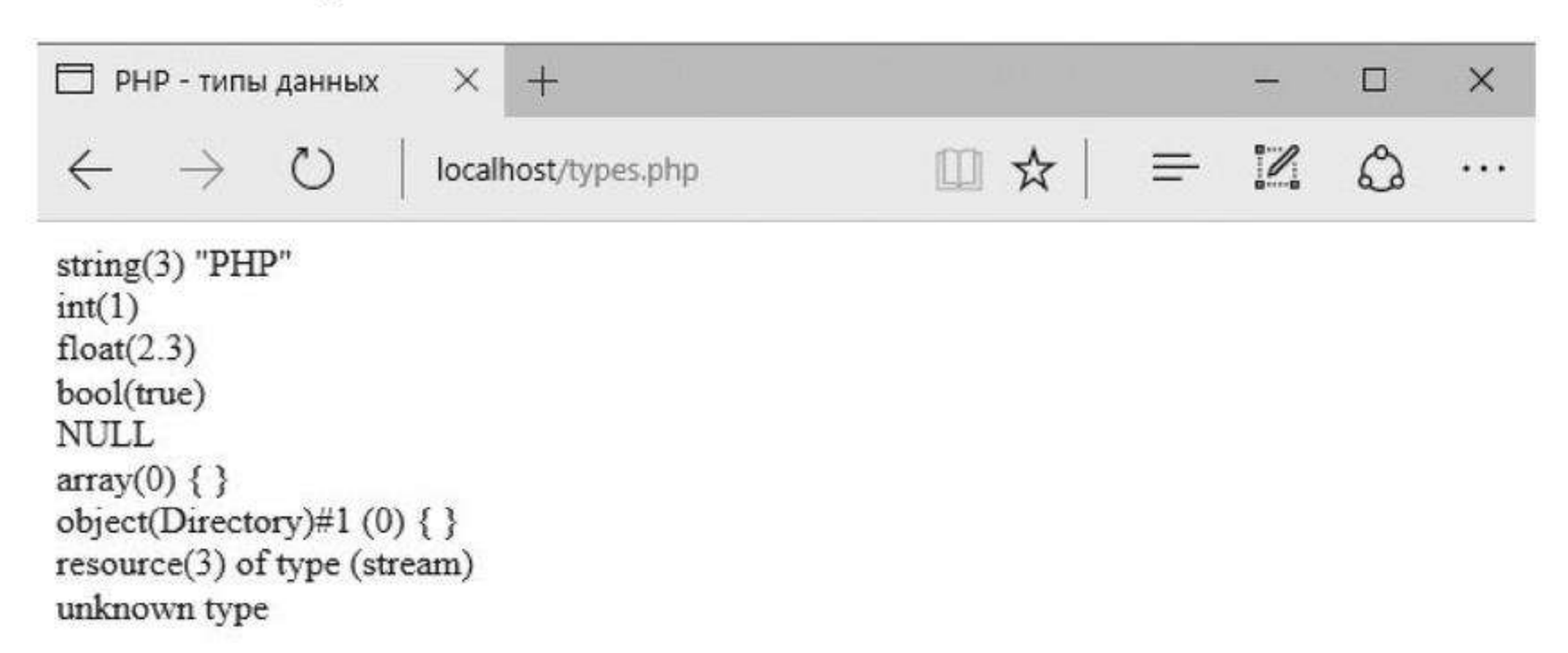
4. И, наконец, добавьте инструкцию для закрытия файлового ресурса и повторного запроса типа данных.

```
fclose($filestream);
echo gettype($filestream);
```

Совет

Булевы константы `TRUE/FALSE` не чувствительны к регистру, поэтому их можно записывать в виде `false/true`.

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *types.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть типы данных и значения.



Константы

Фиксированные значения данных, которые в сценарии никогда не изменятся, в PHP-коде нужно присваивать константам, а не переменным. Константа представляет собой именованный контейнер, хранящий числовые или строковые значения, к которым можно обратиться по имени константы.

В отличие от переменных, константа не может содержать логические значения `TRUE` или `FALSE`, представлять собой объект или содержать пустое значение `NULL`. Кроме того, константа не может быть изменена по мере выполнения сценария.

Совет

Совет. При создании переменной, всегда задумывайтесь о том, будет ли ее значение когда-либо изменяться. Если нет, возможно, лучше будет создать константу вместо переменной, к примеру, `define('PI', 3.14);`.

Константа обеспечивает неизменность значения. Использование констант — одна из лучших практик программирования.

Имена констант подчиняются тем же правилам именования, что и переменные. Но по принятому соглашению в именах констант используются прописные буквы — так их легко отличить от переменных.

Константы создаются с помощью функции `define()`, позволяющей указать имя константы и значение, которое она будет содержать, следующим образом:

```
define('name', значение);
```

Что наиболее важно, имена констант не предваряются символом $, поэтому они не могут быть включены в состав строк для вывода — PHP-интерпретатор не способен отличить имя константы от обычной строки, набранной прописными буквами. Рассмотрим следующие инструкции:

```
define('USER', 'Михаил');
echo "Привет, USER"
```

В результате будет выведена строка `Привет, USER`, а не `Привет, Михаил`, как хотелось бы.

Поэтому, чтобы включить значение константы в выводящуюся строку, к нему нужно обратиться отдельно и «конкатенировать» с текстовой строкой с помощью PHP-оператора конкатенации `.`, как показано ниже:

```
echo 'Привет, '. USER;
```

В результате будет выведена строка `Привет, Михаил`, как и ожидалось.

Константы массивов, в которых все элементы содержат фиксированные значения, также могут быть созданы с помощью функции `define()`:

```
define('НЕДЕЛЯ', ['Пн', 'Вт', 'Ср', 'Чт', 'Пт']);
```

Новинка! Возможность создавать константы массивов с помощью функции `define()` впервые стала доступна в версии PHP 7.

Наряду с пользовательскими константами можно использовать также и предопределенные константы, встроенные в PHP-интерпретатор, например `PHP_VERSION` и `PHP_OS`, которые выводят версию движка и название серверной операционной системы.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

constant.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, создающую и инициализирующую константу.

```
define('USER', 'Михаил');
```

3. Добавьте инструкцию, создающую и инициализирующую константу массива.

```
define('PETS', ['котенок', 'щенок', 'хомячок']);
```

Внимание

Вы не можете включать константы в состав строк, а также изменять значения констант и предварять их имена символом доллара.

4. Затем добавьте инструкцию, выводящую два значения констант в конкатенированную строку.

```
echo '<p>Привет, '. USER. '. Как поживает твой '. PETS[1]. '?
</p>';
```

5. Добавьте инструкцию, выводящую значение версии PHP-движка с помощью предустановленной константы в конкатенированную строку.

```
echo '<p>Ты используешь PHP-движок версии '. PHP_VERSION;
```

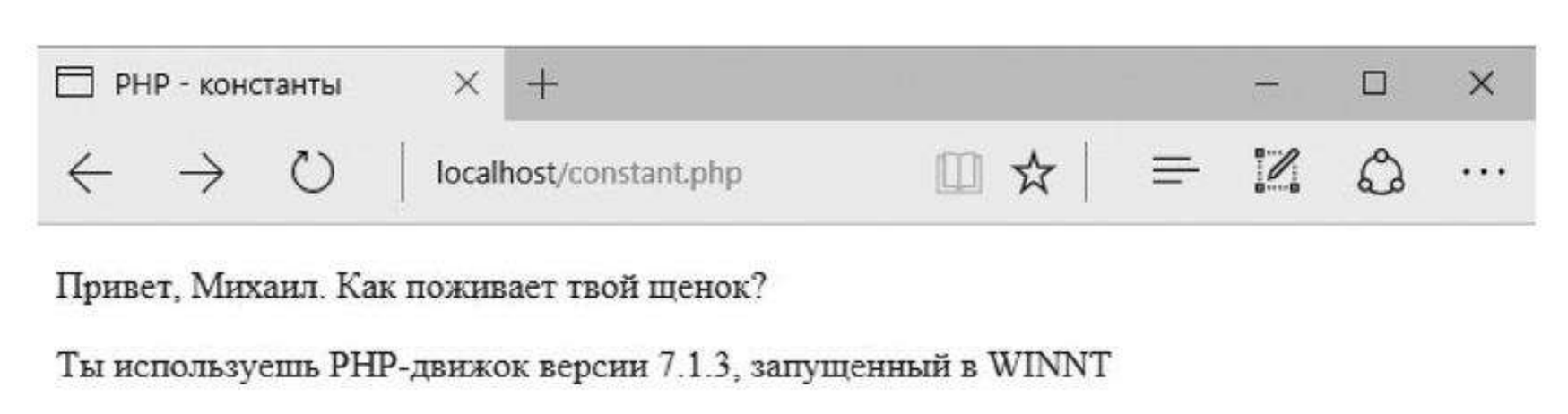
6. И, наконец, добавьте инструкцию, выводящую название операционной системы хоста с помощью предустановленной константы в конкатенированную строку.

```
echo ', запущенный в '. PHP_OS. '</p>';
```


На заметку

На заметку. Имена констант всегда нужно указывать прописными буквами, чтобы в коде их было легко отличить от переменных.

7. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *constant.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.



Суперглобальные переменные

В языке PHP существуют так называемые *суперглобальные переменные*. Это встроенные переменные, которые доступны глобально в любом месте и любом сценарии. Среди них:

Совет

Суперглобальная переменная `$_FILES` описывается в разделе «Выгрузка файлов» главы 9.

- `$GLOBALS` — ассоциативный массив, содержащий ссылки на все переменные, которые в настоящее время определены в глобальной области видимости сценария, в зависимости от контекста. Имена переменных являются ключами этого ассоциативного массива. Например:

Совет

Суперглобальная переменная `$_POST` описывается в разделе «Выполнение действий» главы 9.

- `$_SERVER` — это массив, содержащий служебную информацию, включающую заголовки, пути и местоположения сценариев, которая создается веб-сервером.
- `$_GET` — ассоциативный массив переменных, передаваемых в сценарий с помощью URL-параметров.

Совет

Суперглобальная переменная `$_COOKIE` описывается в разделе «Получение cookie-записей» главы 10.

- $_POST — ассоциативный массив переменных, передаваемых в сценарий через протокол HTTP методом POST.
- $_FILES — ассоциативный массив элементов, загруженных в сценарий через протокол HTTP методом POST.
- $_COOKIE — ассоциативный массив переменных, передаваемых в сценарий через протокол HTTP из cookie-файлов на компьютере пользователя.
- $_SESSION — ассоциативный массив, содержащий переменные, доступные для каждого сценария на веб-сайте, на время сессии пользователя.
- $_REQUEST — ассоциативный массив, который по умолчанию содержит данные переменных $_GET, $_POST и $_COOKIE.

Совет

Суперглобальная переменная $_SESSION описывается в разделе «Настройка сессий» главы 10.

- $_ENV — ассоциативный массив переменных, передаваемых текущему сценарию средой оболочки, в которой запускается PHP-анализатор. В различных системах используются разные оболочки.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

supers.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкцию, выводящую сведения о программном обеспечении используемого сервера.

```
echo 'Веб-сервер: '.$_SERVER['SERVER_SOFTWARE'].'<br>';
```

3. Добавьте инструкцию, выводящую имя текущего сценария.

```
echo 'Сценарий: '.$_SERVER['PHP_SELF'].'<br>';
```

4. Затем добавьте инструкции, выводящие имя хоста и метод запроса страницы.

```
echo 'Имя хоста: '.$_SERVER['HTTP_HOST'].'<br>'; echo
'Метод запроса: '.$_SERVER['REQUEST_METHOD'];
```

Совет

Суперглобальная переменная `$_SERVER['PHP_SELF']` описывается в разделе «Липкие формы» главы 9.

5. Добавьте инструкцию, выводящую передаваемые параметры URL-адреса.

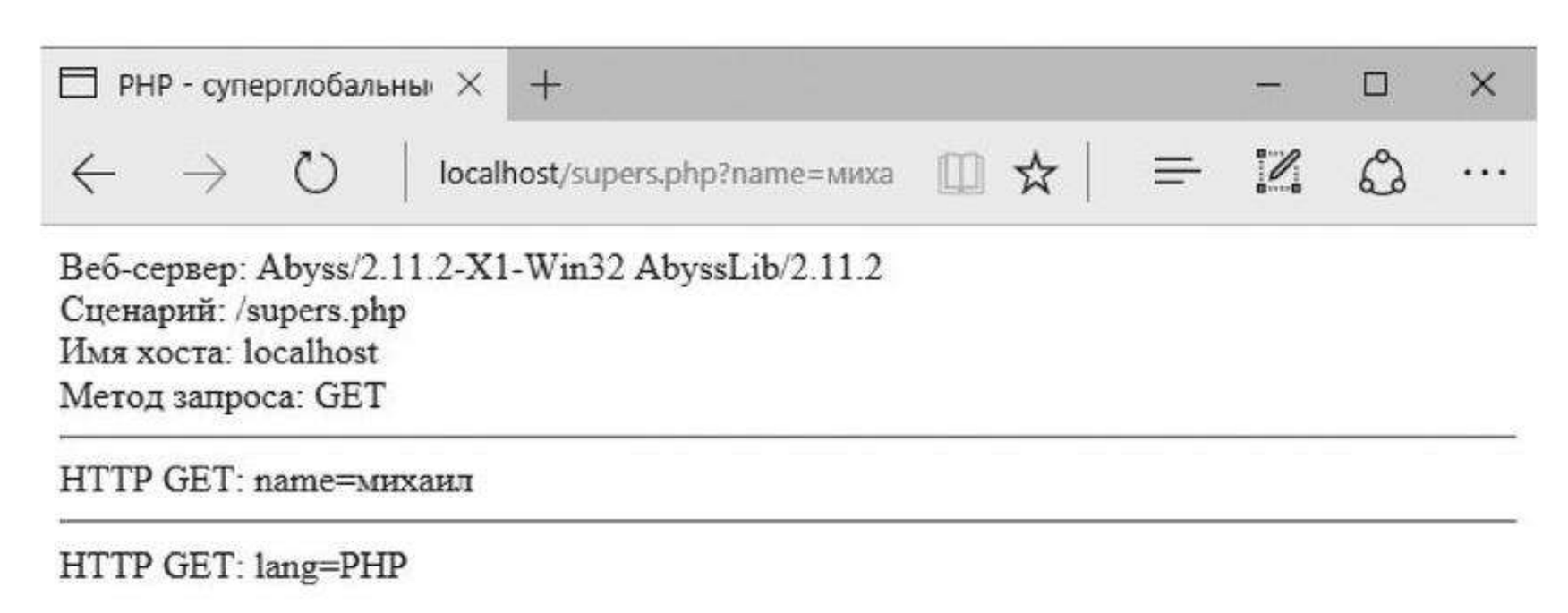
```
foreach($_GET as $key => $value)
{echo '<hr>HTTP GET: '.$key.'='.$value;}
```

6. Сохраните файл под именем *supers.php* в каталоге */htdocs* вашего веб-сервера.

Совет

Суперглобальная переменная `$_SERVER['REQUEST_METHOD']` описывается в разделе «Обработка данных формы» главы 9.

7. Откройте страницу в браузере через протокол HTTP, добавив параметры к имени сценария в адресной строке, например **?name=mike&lang=PHP**. Посмотрите, как эти параметры передаются в сценарий.



Совет

HTTP-метод `GET` описывается в разделе «Настройка параметров» главы 12.

Заключение

- Переменная — это именованный контейнер, в котором могут храниться изменяемые данные.
- К значению, сохраненному в переменной, можно обратиться, указав имя этой переменной.
- Имена переменных должны начинаться с символа доллара, `$`, и содержать латинские буквы, цифры и символы подчеркивания; при этом первый символ после `$` должен быть или буквой, или символом подчеркивания.
- Переменная может хранить целое число, число с плавающей запятой, текстовую строку, логическое значение, объект или значение `NULL`.
- Переменная инициализируется с помощью оператора присваивания, =, который назначает этой переменной начальное значение.
- Значение переменной может быть выведено в составе строки, если при этом сама строка и имя переменной заключены в двойные кавычки.
- Строки могут быть заключены в одиночные или двойные кавычки, при этом внутренние кавычки можно вывести с помощью обратного слеша.
- Строковые значения могут быть объединены в одну строку с помощью оператора конкатенации.
- Массив может хранить несколько элементов данных в виде последовательных значений, которые по умолчанию нумеруются начиная с нуля.
- Каждому элементу массива могут быть назначены имена ключей.
- Цикл `foreach` используется для перебора всех элементов массива.
- Функции `implode()` и `explode()` позволяют объединить элементы массива в строку и преобразовать строку в массив с помощью разделителя.
- Содержимое массивов может сортироваться в алфавитном порядке по значению или ключу.
- Многомерный массив содержит значения элементов, являющихся массивами.
- Функции `gettype()` и `var_dump()` используются для проверки типа данных.
- Константа — это именованный контейнер, которому может быть присвоено неизменяемое (фиксированное) значение: целое число, число с плавающей запятой или текстовая строка.

- Имена констант не предваряются префиксом `$` и указываются прописными буквами.
- Константы инициализируются с помощью функции `define()`.
- Суперглобальные переменные всегда доступны глобально в любой позиции любого сценария.

3

Операторы

В этой главе вы познакомитесь с различными операторами языка PHP: арифметическими, условными, логическими и другими.

- Арифметические операторы
- Операторы сравнения
- Условные операторы
- Логические операторы
- Побитовые операторы
- Операторы инкремента и декремента
- Приоритет операторов
- Заключение

Арифметические операторы

Арифметические операторы, часто применяемые в PHP-сценариях, перечислены в таблице ниже, с указанием действия, которое они выполняют.

Оператор	Действие
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Деление по модулю
**	Возведение в степень

Совет

Числа, используемые вместе с операторами для формирования выражений, называются «операндами» — в выражении 2 + 3 числа 2 и 3 — это операнды.

Операторы для сложения, вычитания, умножения и деления действуют так, как вы себе представляете. Но необходимо быть внимательным и для ясности группировать выражения, в которых используется более одного оператора. Сравните:

```
a = b * c – d% e / f;          # Неясное выражение.
```

и

```
a = (b * c) – ((d% e) / f);    # Ясное выражение.
```

На заметку

Использование в арифметических выражениях скобок позволяет указать приоритет выполнения операций — заключенных в скобки в первую очередь. См. раздел «Приоритет операторов» далее в этой главе для получения дополнительной информации.

Оператор деления по модулю % делит первое заданное число на второе и возвращает остаток от операции. Оператор позволяет определить, является ли число четным или нечетным.

Оператор возведения в степень ** позволяет возвести первый операнд в степень второго операнда.

```
a ** b        # Результат возведения a в степень b
```

Вы можете сокращать арифметические выражения, используя совместно с арифметическими операторами также и оператор присваивания, =. Например:

```
a += b;    # Эквивалент выражения a = (a + b);
a -= b;    # Эквивалент выражения a = (a – b);
a *= b;    # Эквивалент выражения a = (a * b);
a /= b;    # Эквивалент выражения a = (a / b);
```

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

arithmetic.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкции, создающие и инициализирующие две переменные.

```
$a = 5;
$b = 2;
```

3. Затем добавьте инструкции, выводящие результаты простых арифметических операций с использованием значений переменных.

```
$result = $a + $b; echo "Сложение: $result <br>";
$result = $a – $b; echo "Вычитание: $result <br>";
$result = $a * $b; echo "Умножение: $result <br>";
$result = $a / $b; echo "Деление: $result <br>";
```

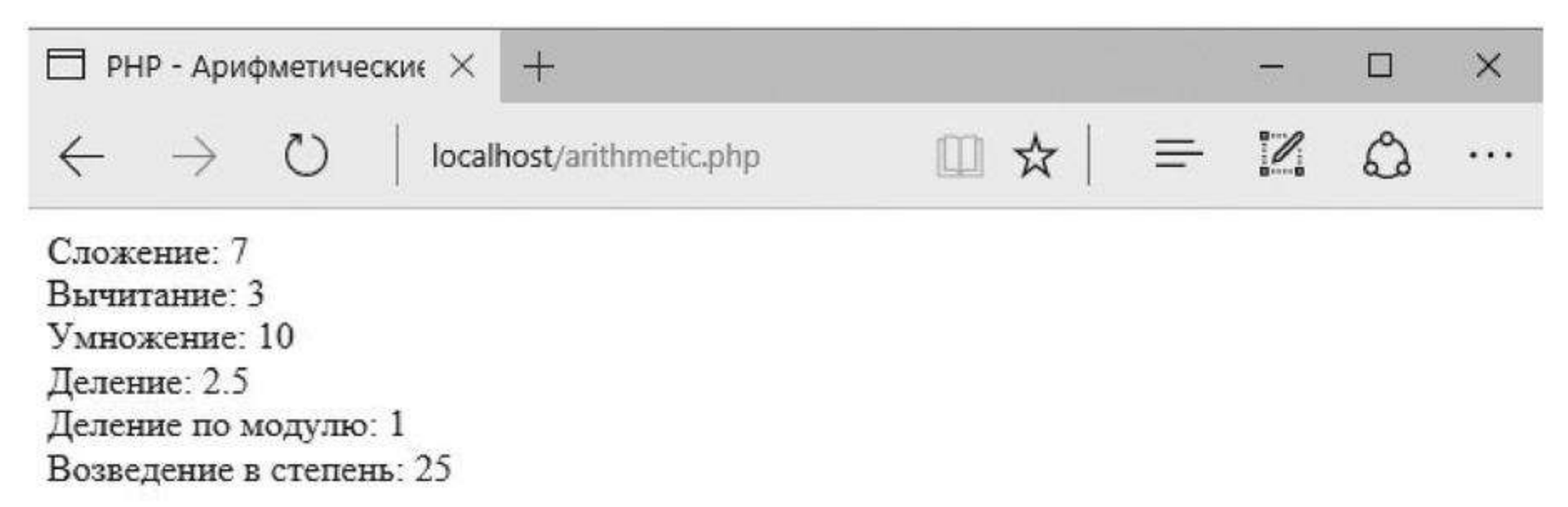
Совет

Язык PHP содержит ряд полезных функций для работы с числами. К примеру, функция `round($a)` округляет значение `$a` до ближайшего целого числа, а `number_format($a)` форматирует число `$a` с разделением групп запятыми.

4. И, наконец, добавьте инструкцию для вывода результатов деления по модулю и возведения в степень значений переменных.

```
$result = $a% $b; echo "Деление по модулю: $result <br>";
$result = $a ** $b; echo "Возведение в степень: $result";
```

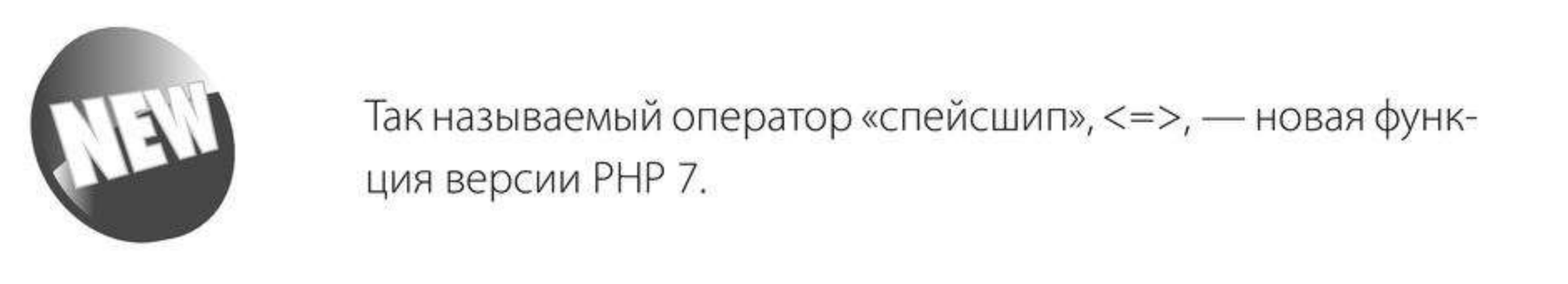
5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *arithmetic.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера и вычисления значений переменных.



В версии PHP 7 появилась новая функция `intdiv()`, которая выполняет целочисленное деление на два операнда и возвращает целое число. Например, функция `intdiv(10,3)` возвращает число 3.

Операторы сравнения

Операторы, которые обычно используются в PHP-сценариях для сравнения двух значений, приведены в следующей таблице.



Так называемый оператор «спейсшип», <=>, — новая функция версии PHP 7.

Оператор равенства, ==, сравнивает два операнда и возвращает логическое значение `TRUE`, если их значения численно равны, в противном случае будет возвращено значение `FALSE`. Символы и строки также можно сравнить как числа, согласно значениям их ASCII-кода. Как противоположность

используется оператор неравенства !=, который возвращает значение TRUE при неравенстве операндов, и FALSE при их равенстве.

Оператор	Тип сравнения
==	Равно
===	Идентично
!==	Не идентично
!= <>	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
<=>	Меньше, равно или больше

Совет

Операторы равенства и неравенства полезны для проверки состояния двух переменных при использовании условного ветвления в PHP-сценарии.

Оператор «Больше», >, сравнивает два операнда и возвращает значение TRUE, если первый из них больше второго, и значение FALSE, если он равен или меньше второго. Оператор «Меньше», <, выполняет аналогичное сравнение, но в обратную сторону: возвращает значение TRUE, если первый операнд меньше второго, FALSE в противном случае. Добавление оператора = к символу > или < немного изменяет поведение операторов «Больше» и «Меньше», возвращая значение TRUE также если оба операнда равны.

Т.н. оператор «Спейсшип» (космический корабль), <=>, возвращает целочисленное значение 1, если первый операнд больше второго, –1, если первый операнд меньше второго, или 0, если оба операнда равны.

PHP-функция `var_dump()` используется для просмотра типа и значения результатов сравнения, выполненных с помощью любого оператора.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

comparison.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкции, создающие и инициализирующие пять переменных.

```
$zero = 0; $nil = 0; $one = 1; $upr = 'A'; $lwr = 'a';
```

3. Далее добавьте инструкции, выводящие результаты проверки равенства значений переменных.

```
echo "0 == 0: "; var_dump($zero == $nil);
echo "<br>0 == 1: "; var_dump($zero == $one);
echo "<br>A == a: "; var_dump($upr == $lwr);
echo "<br>A!= a: "; var_dump($upr == $lwr);
```

Совет

ASCII-код прописной буквы «A» имеет значение 65, а строчной «a» — 97, поэтому результатом их сравнения будет `FALSE`.

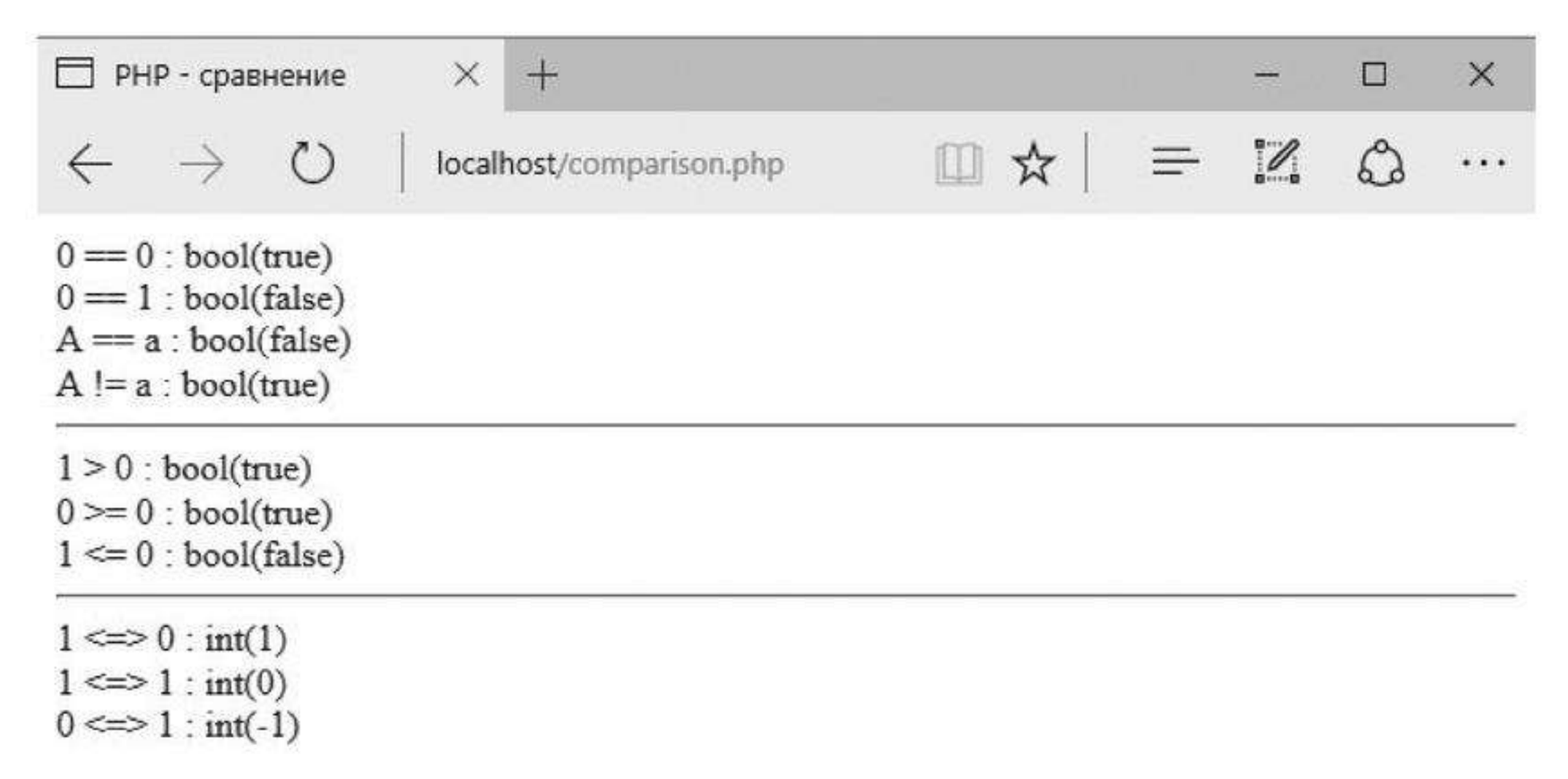
4. Добавьте инструкции, выводящие результаты сравнения значений переменных.

```
echo "<hr>1 > 0: "; var_dump($one > $nil);
echo "<br>0 >= 0: "; var_dump($zero >= $nil);
echo "<br>1 <= 0: "; var_dump($one <= $nil);
echo "<hr>1 <=> 0: "; var_dump($one <=> $nil);
echo "<br>1 <=> 1: "; var_dump($one <=> $one);
echo "<br>0 <=> 1: "; var_dump($nil <=> $one);
```

Совет

Совет. Логическое значение `TRUE` может быть представлено числом 1 в языке PHP.

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *comparison.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.



Условные операторы

Условный оператор

Условный оператор, `?:`, также известный как «тернарный», принимает три значения. Этот оператор сначала вычисляет логическое значение `TRUE/FALSE` выражения, а затем возвращает один из двух указанных результатов в зависимости от результатов вычисления. Синтаксис условного оператора выглядит следующим образом:

```
(проверочное-выражение)? результат-если-истинно: результат-если-ложно;
```

Условный оператор используется для определения, является ли указанное число тем или другим, для правильного употребления форм слова в случае использования единственного и множественного чисел и т.п. Так вы сможете избежать неправильных фраз, таких как «1 чемоданов».

Внимание

Не следует использовать больше одного условного оператора в одной инструкции, поскольку результаты могут быть непредсказуемыми.

```
$verb = ($number == 1)? 'угадали': 'не угадали';
echo "Вы $verb! Вы назвали число $number.";
```

В данном примере, если переменной `$number` присвоено значение `1`, переменной `$verb` присваивается значение `'угадали'`, а при любом другом значении переменной `$number` — значение `'не угадали'`.

Условный оператор также может быть использован для определения, является ли указанное число четным или нечетным (согласно равенству). Это вычисляется путем анализа остатка от деления числа на два, а затем подходящее значение выводится в строке:

```
$parity = ($number% 2 == 0)? 'нечетное': 'четное';
echo "$number — $parity число";
```

В этом примере, если операция деления по модулю возвращает целое число, переменной `$parity` присваивается значение `'четное'`, в противном случае — значение `'нечетное'`.

Оператор объединения со значением NULL

Оператор объединения со значением `NULL` (`??`) — новая функция в версии PHP 7.

Язык PHP поддерживает особое значение `NULL`, встроенную константу, которая представляет собой переменную без какого-либо значения, даже нуля. Переменные, которым не было присвоено значение, обрабатываются как `NULL`. Оператор объединения со значением `NULL` может использоваться для обхода операндов слева направо, после чего он возвращает значение первого операнда, если тот не равен `NULL`, или второго, если первый равен `NULL`. Если ни одному из операндов не присвоено значение (в т.ч. и не `NULL`), то данный оператор сам вернет в результате `NULL`.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

sort.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкции, создающие и инициализирующие три переменные.

```
$a = NULL; $b = 8; $c = 'PHP — это весело!';
```

3. Затем добавьте инструкции для вывода результата угадывания числа.

```
$verb = ($b == 1)? 'угадали': 'не угадали';
echo "Вы $verb! Вы назвали число $b. <hr>";
```

4. Добавьте инструкции, выводящие результат проверки числа на четность.

```
$parity = ($b% 2!= 0)? 'нечетное': 'четное';
echo "$b – $parity число<hr>";
```

5. И, наконец, добавьте инструкции, чтобы отобразить значения, отличные от NULL.

```
$result = $a?? $b?? $c; echo "abc: $result <br>";
$result = $c?? $b?? $a; echo "cba: $result <br>";
```

На заметку

Обратите внимание на то, что значение NULL, присвоенное переменной $a, игнорируется.

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *condition.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.

Логические операторы

Логические (булевы) операторы, часто используемые в PHP-сценариях, перечислены в следующей таблице.

Оператор		Действие
and	&&	Логическое «И»
or	||	Логическое «ИЛИ»
xor		Исключающее «ИЛИ»
!		Логическое «НЕ»

На заметку

Оператор `&&` — альтернативная форма оператора `and`. Аналогичным образом, оператор `||` — другая форма оператора `or`. Смысл двух разных вариантов для операторов Логическое «И» и Логическое «ИЛИ» в том, что они работают с различными приоритетами.

Логические операторы используются с операндами, которые имеют логические значения `TRUE` или `FALSE`, или в выражениях, которые могут быть преобразованы в `TRUE` или `FALSE`.

Оператор `and` анализирует два операнда и возвращает `TRUE`, если обоим операндам присвоено значение `TRUE`. В противном случае оператор `and` вернет значение `FALSE`. Этот оператор используется в условном ветвлении, когда вариант выполнения PHP-сценария определяется путем проверки двух условий. Если оба условия выполнены, будет запущена одна ветвь кода сценария, если одно или оба условия не выполнены, будет запущена другая ветвь кода.

В отличие от оператора `and`, который требует истинности (значения `TRUE`) обоих операндов, оператор `or` проверяет оба операнда и возвращает значение `TRUE`, если хотя бы одному из операндов присвоено значение `TRUE`. Если ни один из операндов не имеет значения `TRUE`, то оператор `or` вернет `FALSE`. Этот оператор полезен в случаях, когда требуется, чтобы для выполнения сценария одно или оба проверяемых условия были выполнены.

Оператор `xor` проверяет оба операнда и возвращает `TRUE`, если только один из двух операндов истинный — но не оба. Если ни одному из операндов

не присвоено значение TRUE или если оно присвоено обоим операндам, то оператор xor вернет FALSE.

На заметку

Термин «булев» относится к системе математической логики, которую развивал английский математик Джордж Буль (1815–1864).

Оператор логического отрицания ! является «унарным» — т.е. используется с одним операндом*. Он возвращает обратное значение указанного операнда, поэтому, если переменной $var присвоено значение TRUE, инструкция ! $var вернет значение FALSE. Оператор ! пригодится для переключения значения переменной в цикле с инструкциями вида $var =! $var. При каждой итерации цикла значение обращается, словно вы щелкаете выключателем освещения.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

logic.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Теперь добавьте между PHP-тегами инструкции, создающие и инициализирующие две переменные с логическими значениями.

```
$yes = TRUE; $no = FALSE;
```

3. Затем добавьте инструкции с оператором Логическое «И».

```
$result = ($no && $no)? 'TRUE': 'FALSE';
echo "no И no вернет $result <br>";
$result = ($yes && $no)? 'TRUE': 'FALSE';
echo "yes И no вернет $result <br>";
```

* Если используется два операнда, то операторы называются бинарными. — *Прим. перев.*

```
$result = ($yes && $yes)? 'TRUE': 'FALSE';
echo "yes И yes вернет $result <hr>";
```

Совет

Язык PHP также поддерживает оператор `xor` (Исключающее «ИЛИ»), выполняющий сравнение по методам «И» и «НЕ».

4. Добавьте инструкции с операторами Логическое «ИЛИ» и Логическое «НЕ».

```
$result = ($no || $no)? 'TRUE': 'FALSE';
echo "no ИЛИ no вернет $result <br>";
$result = ($yes || $no)? 'TRUE': 'FALSE';
echo "yes ИЛИ no вернет $result <br>";
$result = ($yes || $yes)? 'TRUE': 'FALSE';
echo "yes ИЛИ yes вернет $result <hr>";
$result = (! $yes)? 'TRUE': 'FALSE';
echo "НЕ yes вернет $result <br>";
```

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *logic.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.

На заметку

Обратите внимание на то, что инструкция `FALSE && FALSE` возвращает `FALSE`.

Побитовые операторы

Байт содержит восемь битов, каждый из которых содержит `1` или `0` для хранения двоичного значения и представления десятичного числа в диапазоне от 0 до 255. Бит определяет десятичный разряд только в том случае, когда этот бит содержит 1. Разряды обозначаются справа налево от «младшего двоичного разряда» (LSB, Least Significant Bit) до «старшего двоичного разряда» (MSB, Most Significant Bit). Двоичное число в таблице разрядов ниже представляет собой десятичное число 50.

№ бита	8 MSB	7	6	5	4	3	2	1 LSB
Десятичное	128	64	32	16	8	4	2	1
Двоичное	0	0	1	1	0	0	1	0

На заметку

Многие PHP-программисты не пользуются побитовыми операторами, но мы рекомендуем прочитать этот раздел, чтобы знать, что они собой представляют и как могут использоваться.

Можно управлять отдельными частями байта, используя побитовые операторы. Они позволяют вычислять и управлять отдельными битами в целом числе.

Оператор	Название	Действие с двоичным числом
`|`	ИЛИ	Возвращает `1` в каждом бите, если любой из двух сравниваемых битов равен `1`. Пример: `1010 |0101 = 1111`
`&`	И	Возвращает `1` в каждом бите, если оба из двух сравниваемых битов равны `1`. Пример: `1010 && 1100 = 1000`
`~`	Отрицание	Возвращает `1` в каждом бите, если ни один из двух сравниваемых битов не равен `1`. Пример: `1010 ~ 0011 = 0100`
`^`	Исключающее ИЛИ	Возвращает `1` в каждом бите, если только один из двух сравниваемых битов равен `1`. Пример: `1010 ^ 0100 = 1110`
`<<`	Сдвиг влево	Перемещает каждый бит, равный `1`, на заданное число битов влево. Пример: `0010 << 2 = 1000`
`>>`	Сдвиг вправо	Перемещает каждый бит, равный `1`, на заданное число битов вправо. Пример: `1000 >> 2 = 0010`

Совет

Каждая из двух половин байта называется «нибблом*» (4 бита). Двоичные числа в примерах, приведенных в таблице, представляют собой значения, хранящиеся в ниббле.

При разработке для устройств с ограниченными ресурсами часто возникает потребность в применении побитовых операторов. К примеру, оператор ^ (исключающее ИЛИ) позволяет обмениваться значениями между двумя переменными без необходимости использования третьей переменной.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

bitwise.php

```
<?php
   # Сюда добавляется php-код.
?>
```

2. Добавьте между PHP-тегами инструкции, создающие и инициализирующие две переменные с целочисленными значениями.

```
$x = 5; $y = 10;
```

3. Далее добавьте инструкцию, выводящую присвоенные значения.

```
echo "X: $x, Y: $y <br>";
```

4. Добавьте три инструкции с оператором ^ для изменения значений переменных с помощью перемещения битов на двоичном уровне.

```
$x = $x ^ $y; /* 1010 ^ 0101 = 1111 (десятичное 15) */
$y = $x ^ $y; /* 1111 ^ 0101 = 1010 (десятичное 10) */
$x = $x ^ $y; /* 1111 ^ 1010 = 0101 (десятичное 5) */
```

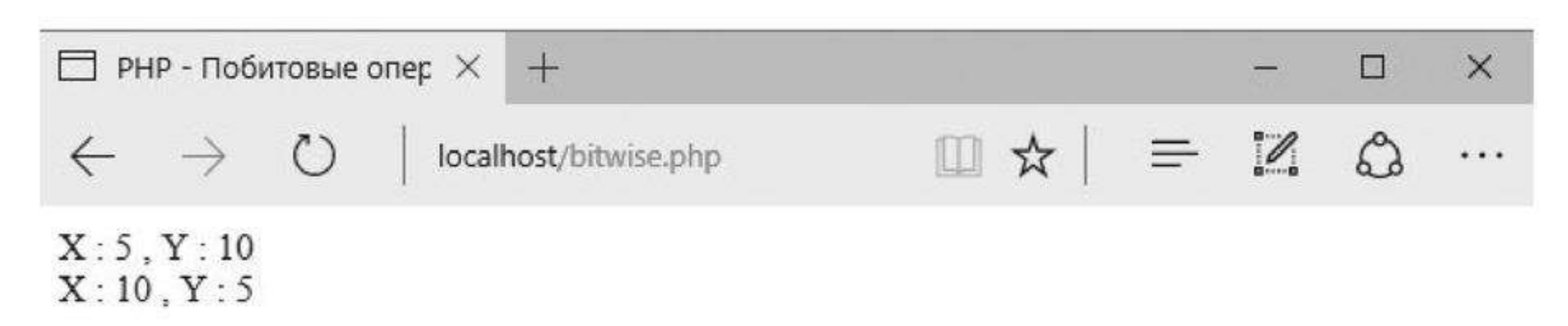
5. И, наконец, добавьте инструкцию, выводящую измененные значения переменных.

```
echo "X: $x, Y: $y <br>";
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *bitwise.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат работы оператора ^.

---

* Или «полубайтом». — *Прим. перев.*



На заметку

Не путайте битовые операторы с логическими. Битовые операторы управляют двоичными числами, в то время как логические операторы — логическими значениями.

Операторы инкремента и декремента

В языке PHP операторы инкремента, ++, и декремента, –, изменяют заданное число на единицу и возвращают полученное значение. Они чаще всего используются для подсчета итераций в цикле. Оператор инкремента увеличивает значение на единицу, а оператор декремента уменьшает значение на единицу. Каждый из этих операторов могут указываться как до, так и после переменной, содержащей изменяемое число, — эффект будет разным! При указании оператора перед переменной (префикс) значение изменяется на единицу, а затем возвращается измененное значение. При указании оператора после переменной (постфикс) сначала возвращается текущее значение, а потом оно изменяется на единицу.

1. Подготовьте шаблонный HTML-документ, описанный в разделе «Внедрение PHP-кода» главы 1, а затем добавьте следующий PHP-код в тело страницы.

change.php

```
<?php
   # Сюда добавляется php-код.
?>
```

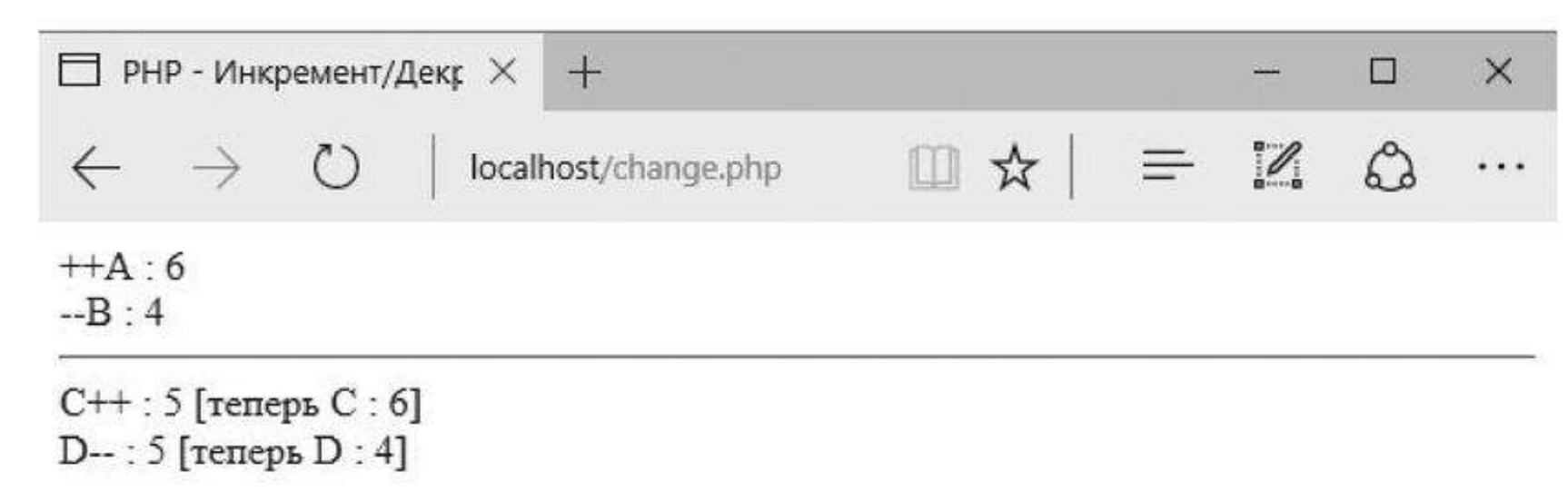
2. Теперь добавьте между PHP-тегами инструкции, создающие и инициализирующие четыре переменные с одним и тем же целочисленным значением.

```
$a = $b = $c = $d = 5;
```

3. Далее добавьте инструкции для изменения каждого значения по очереди и вывода полученных значений.

```
echo "++A: ". ++$a. "<br>—B: ". —$b ."<hr>";
echo "C++: ". $c++. "[теперь C: ". $c ."]<br>";
echo "D—: ". $d—. "[теперь D: ". $d ."]<br>";
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *change.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.



Совет

Обратите внимание на то, каким образом оператор конкатенации, `.`, используется в данном примере для формирования строк. Вы также можете использовать оператор присваивания с конкатенацией, `.=`, чтобы присоединить одну строку к другой.

Приоритет операторов

Приоритет оператора в PHP определяет порядок обработки выражений интерпретатором. Например, выражение `1 + 5 * 3` в итоге выдает результат 16, а не 18, потому что оператор умножения, `*`, имеет более высокий приоритет, чем оператор сложения, `+`. Для указания приоритета могут использоваться скобки, в этом случае выражение `(1 + 5) * 3` в результате выдаст 18, а не 16.

Когда операторы имеют равный приоритет, играет роль их «ассоциативность», определяющая, в какую сторону (справа налево или слева направо) выражения будут выполняться. Например, оператор вычитания, –, левоассоциативный, поэтому выражение `8-4-2` группируется следующим образом: `(8-4) — 2` и в итоге приводит к результату 2. В таблице ниже приведены наиболее востребованные операторы, отсортированные по убыванию их приоритетов

(т.е. с более высоким приоритетом вверху). Операторы, размещенные в одной строке, имеют одинаковый приоритет, и порядок их выполнения определяется исходя из их ассоциативности.

Оператор	**Ассоциативность**
**	правая
++ — ~	правая
!	правая
* /%	левая
+ -.	левая
<< >>	левая
< <= > >=	нет
==!= ===!== <> <=>	нет
&	левая
^	левая
|	левая
&&	левая
||	левая
??	правая
?:	левая
= += -= *= **= /= .=%= &= |= ^= <<= >>=	правая
and	левая
xor	левая
or	левая

Внимание

Если вы внимательны, то заметите, что альтернативные формы логических операторов имеют разные уровни приоритета. Это может привести к неожиданным результатам в таких выражениях, как `$bool = true and false`, в сравнении с `$bool = true && false`. Проверьте каждое из них с помощью функции `var_dump($bool)`, чтобы увидеть разницу.

Заключение

- Арифметические операторы позволяют создавать выражения с операндами для их сложения (+), вычитания (–), умножения (*), деления (/), деления по модулю (%) и возведения в степень (**).
- Оператор присваивания, =, может быть объединен с любым арифметическим оператором, позволяя выполнить арифметическое вычисление, а затем присвоить результат.
- Операторы сравнения позволяют создавать выражения с операндами для проверки их равенства (==), неравенства (!=), идентичности (===) или ее отсутствия, (!==), а также какой из них больше (>), меньше (<), больше или равен (>=), меньше или равен (<=) или меньше, равен или больше (<=>).
- Условный оператор ?: вычисляет логическое значение TRUE/FALSE выражения, а затем возвращает один из двух указанных результатов в зависимости от результатов вычисления.
- Оператор объединения со значением NULL может использоваться для обхода операндов слева направо, после чего он возвращает значение первого операнда, если тот не равен NULL, или второго, если первый равен NULL.
- Логические операторы and, &&, or, || и xor позволяют создавать выражения с двумя сравниваемыми операндами и возвращают значение TRUE или FALSE.
- Оператор ! (Логическое «НЕ») возвращает обратное логическое значение единственного операнда.
- Побитовые операторы |, &, ~, ^, << и >> позволяют вычислять и управлять отдельными битами внутри целочисленного значения.
- Операторы инкремента, ++, и декремента, –, при указании перед переменной изменяют заданное число на единицу и возвращают полученное значение.
- Операторы инкремента, ++, и декремента, –, при указании после переменной изменяют значение на единицу, а затем возвращают измененное значение.
- Приоритет операторов определяет, каким образом выражения группируются для вычисления.
- Если операторы имеют одинаковый приоритет, за группирование выражений отвечает ассоциативность операторов.
- Функция var_dump() используется для просмотра типа и значения результатов вычисления выражения.
- Специальная константа NULL представляет собой переменную, которая не имеет никакого значения, даже нуля.

4

Условные конструкции

Эта глава познакомит вас с различными условными конструкциями языка PHP, которые определяют вариант выполнения сценария.

- **Конструкция `if`**
- **Конструкция `else`**
- **Оператор `switch`**
- **Конструкция `foreach`**
- **Цикл `while`**
- **Оператор `break`**
- **Заключение**

Конструкция `if`

Ключевое слово `if` используется для условного выполнения фрагментов кода, в зависимости от результатов проверки логического значения (`TRUE` или `FALSE`) данного выражения. Инструкции в фигурных скобках после проверки будет выполнены только в том случае, если определено, что выражение истинно (`TRUE`). Если выражение оказывается ложно (`FALSE`), PHP-анализатор игнорирует инструкции и попросту переходит к следующей части сценария.

Совет

Конструкция `if` — одна из наиболее важных функций большинства языков программирования, включая PHP.

Синтаксис конструкции `if` выглядит следующим образом:

```
if (выражение) {инструкция-если-выражение-истинно;}
```

В конструкцию может быть вложено несколько инструкций, которые будут выполняться при истинности выражения. При этом каждая инструкция должна заканчиваться точкой с запятой.

```
if (выражение)
{
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
}
```

Несколько выражений можно проверять с помощью логического оператора `&&` или `and`, чтобы последующие инструкции выполнялись, только если оба выражения истинны:

```
if ((выражение) && (выражение))
{
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
}
```

Совет

Когда код содержит лишь одну инструкцию для выполнения, фигурные скобки могут быть опущены, но для порядка рекомендуется всегда заключать инструкции в фигурные скобки.

Кроме того, конструкции `if` могут вкладываться друг в друга, позволяя проверять несколько выражений. В этом случае инструкции будут выполняться, только если все выражения, проверенные по очереди, будут истинны:

```
if (выражение)
{
   if (выражение)
   {
      инструкция-если-выражение-истинно;
      инструкция-если-выражение-истинно;
      инструкция-если-выражение-истинно;
   }
}
```

Вложенные конструкции `if` могут использоваться для реализации альтернативных путей выполнения сценария.

1. Начните сценарий с добавления между PHP-тегами инструкции, инициализирующей целочисленную переменную.

if.php

```
<?php
$num = 8;
```

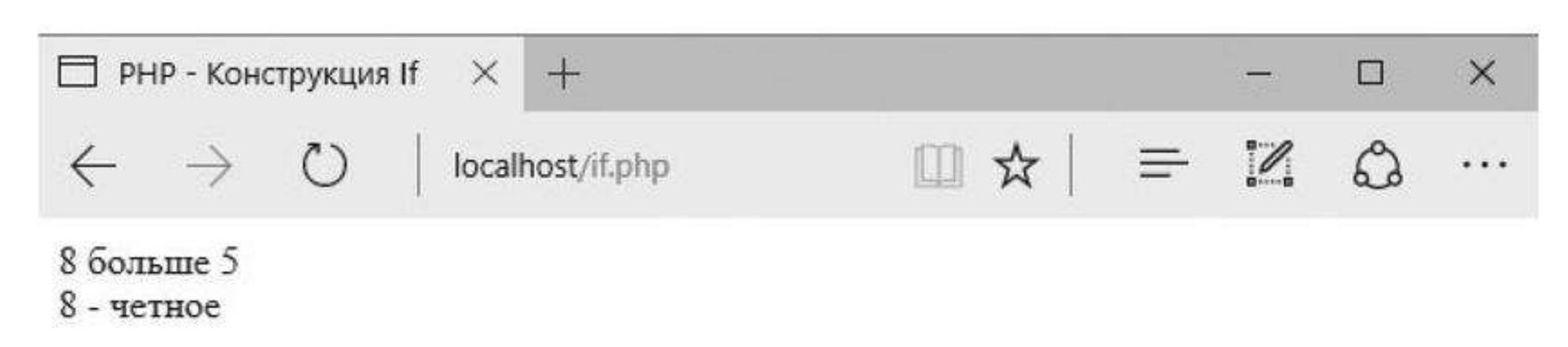
2. Далее добавьте инструкции, выводящие разный результат после проверки величины числа.

```
if ($num > 5) {echo "$num больше 5<br>";}
if ($num <= 5) {echo "$num меньше 6<br>";}
```

3. Теперь добавьте инструкции, выводящие разный результат после проверки четности числа.

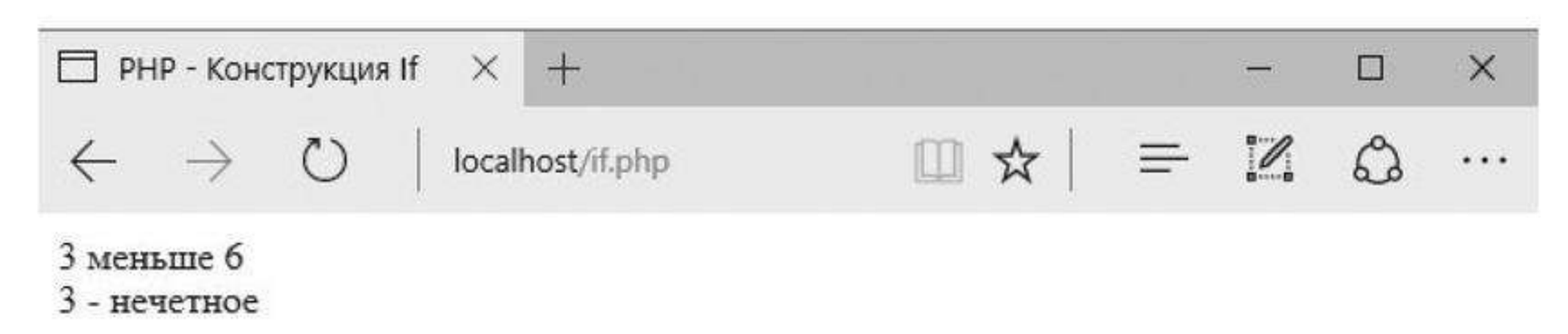
```
if ($num% 2 == 0) {echo "$num — четное<br>";}
if ($num% 2!= 0) {echo "$num — нечетное<br>";}
?>
```

4 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *if.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.



5 Измените значение переменной, сохраните сценарий и обновите страницу в браузере, чтобы увидеть результат выполнения кода.

```
$num = 3;
```



На заметку

Если проверяемое выражение ложно, PHP-анализатор пропускает блок инструкций и переходит к следующему фрагменту кода.

Конструкция `else`

Зачастую предпочтительнее расширить конструкцию `if`, добавив инструкции `else`, заключив их в фигурные скобки. Они будут выполняться, если проверяемое выражение будет ложным (`FALSE`). Синтаксис следующий:

```
if (выражение)
{
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
}
```

```
else
{
   инструкция-если-выражение-ложно;
   инструкция-если-выражение-ложно;
   инструкция-если-выражение-ложно;
}
```

Дополнительные выражения также могут быть включены в код. Для этого используется конструкция `elseif`, инструкции которой также заключаются в фигурные скобки и выполняются, если дополнительное выражение истинно.

Конструкция `else` также может быть добавлена. Ее инструкции заключаются в фигурные скобки и выполняются, когда выражения конструкций `if` и `elseif` ложны. Синтаксис следующий:

Совет

Ключевое слово `elseif` также можно записать в виде двух отдельных слов: `else if`.

```
if (выражение)
{
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
}
elseif (выражение)
{
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
   инструкция-если-выражение-истинно;
}
else
{
   инструкция-если-выражение-ложно;
   инструкция-если-выражение-ложно;
   инструкция-если-выражение-ложно;
}
```

Описанный фундаментальный подход в программировании, который предполагает выполнение сценария в разных направлениях в зависимости от результата проверки выражений, известен под названием «условное ветвление».

Внимание

Вы можете встретить PHP-сценарии, в которых используется символ двоеточия (`:`), а не фигурные скобки. Так делать не рекомендуется, потому что конструкции `if` и `else if` обрабатываются правильным образом только при использовании фигурных скобок.

1. Начните сценарий с добавления между PHP-тегами инструкции, выводящей результат только в том случае, если проверяемое выражение истинно (`TRUE`).

else.php

```
<?php
if (4 > 2) {echo '<p>Да, 4 больше, чем 2 <br>';}
```

2. Затем добавьте инструкцию, которая выводит результат только в том случае, если оба проверяемых выражения истинны.

```
if ((4 > 2) && (8 > 4))
{echo '4 больше, чем 2, а 8 больше, чем 4<br>';}
```

3. Теперь добавьте инструкцию, выводящую разный результат, если проверяемое выражение истинно или ложно.

```
if (4 > 8)
{echo '4 больше, чем 8 <br>';}
else
{echo '4 меньше, чем 8 <br>';}
```

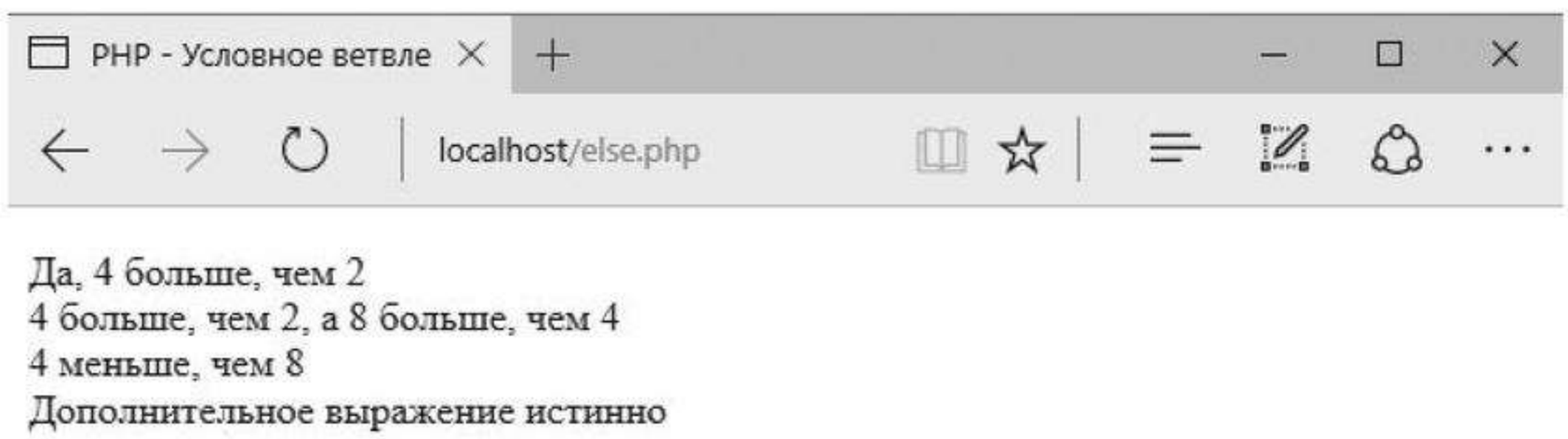
4. Наконец, добавьте инструкцию, выводящую разный результат, если одно из проверяемых выражений истинно, либо если оба выражения ложны.

```
if (4 > 8)
{echo 'Это выражение истинно</p>';}
elseif (8 > 4)
{echo 'Дополнительное выражение истинно</p>';}
else
{echo 'Оба выражения ложны</p>';}
?>
```

На заметку

Вы можете вкладывать условные конструкции `if` (одну внутрь другой), чтобы инструкции во вложенной конструкции выполнялись, только если оба выражения истинны.

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *else.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера с условным ветвлением.



Конструкция `switch`

Условное ветвление с помощью групп инструкций в конструкциях `if-elseif-else` может выполняться более эффективно с помощью конструкции `switch`, в которой проверяется только одно условие выражения.

Инструкция `switch` необычна. Она принимает заданное значение в качестве параметра, а затем ищет совпадения с полученным значением в списке инструкций `case`. В каждой конструкции `switch` будет выполнен код той инструкции `case`, в которой было обнаружено совпадение.

Важно завершать каждую инструкцию `case` ключевым словом `break`, чтобы работа конструкции `switch` завершалась после обнаружения первого совпадения (т.е. без поиска дополнительных совпадений в том же списке инструкций той же конструкции `switch`) — если, разумеется, это не является обязательным требованием.

Список инструкций `case` может сопровождаться одной финальной инструкцией `default`, код которой будет выполняться в том случае, если совпадений не обнаружено ни в одной из инструкций `case`. Общий синтаксис конструкции `switch` выглядит следующим образом:

Совет

Рекомендуется включать в конструкции `switch` инструкцию `default` — например, для вывода сообщения об ошибке, если код конструкции по каким-то причинам не работает должным образом.

```
switch (параметр)
{
 case значение: инструкции-если-совпадение-обнаружено;
 break;
 case значение: инструкции-если-совпадение-обнаружено;
 break;
 case значение: инструкции-если-совпадение-обнаружено;
 break;
 default: инструкции-если-совпадений-не-обнаружено;
}
```

Конструкция `switch` может содержать множество инструкций `case`, но все они должны содержать разные значения (нельзя присваивать одно и то же значение двум и более инструкциям `case`).

В случае, если несколько сверяемых значений приводят к выполнению одних и тех же инструкций, только последнее из них требует указания выполняемой инструкции ключевого слова `break` для завершения работы с конструкцией `switch`. К примеру, в коде, представленном ниже, для значений 0, 1 и 2 выводится одно и то же сообщение (`Меньше, чем 3`), хотя и команда `echo` указана только для значения `2`.

На заметку

Инструкция `default` не обязательно должна указываться в конце конструкции `switch`, но ее логично помещать именно туда, так в этом случае не требуется указывать инструкцию `break`.

```
switch ($number)
{
 case 0:
 case 1:
 case 2: echo 'Меньше, чем 3'; break;
 case 3: echo 'Равно 3'; break;
 default: echo 'Больше 3 или меньше нуля';
}
```

1. Начните сценарий с добавления между PHP-тегами инструкций, инициализирующих две переменные, одну с целочисленным значением, а вторую — с буквенным.

switch.php

```
<?php
$number = 2; $letter = 'Б';
```

2. Затем добавьте конструкцию `switch`, инструкции которой сверяют целочисленное значение переменной.

```
switch ($number)
{
 case 1: echo 'Единица<br>'; break;
 case 2: echo 'Двойка<br>'; break;
 case 3: echo 'Тройка<br>'; break;
 default: echo 'Неизвестное число<br>';
}
```

3. Теперь добавьте конструкцию `switch`, инструкции которой сверяют буквенное значение переменной.

```
switch ($letter)
{
 case 'А': echo 'Буква "А"<br>'; break;
 case 'Б': echo 'Буква "Б"<br>'; break;
 case 'В': echo 'Буква "В"<br>'; break;
 default: echo 'Неизвестная буква<br>';
}
```

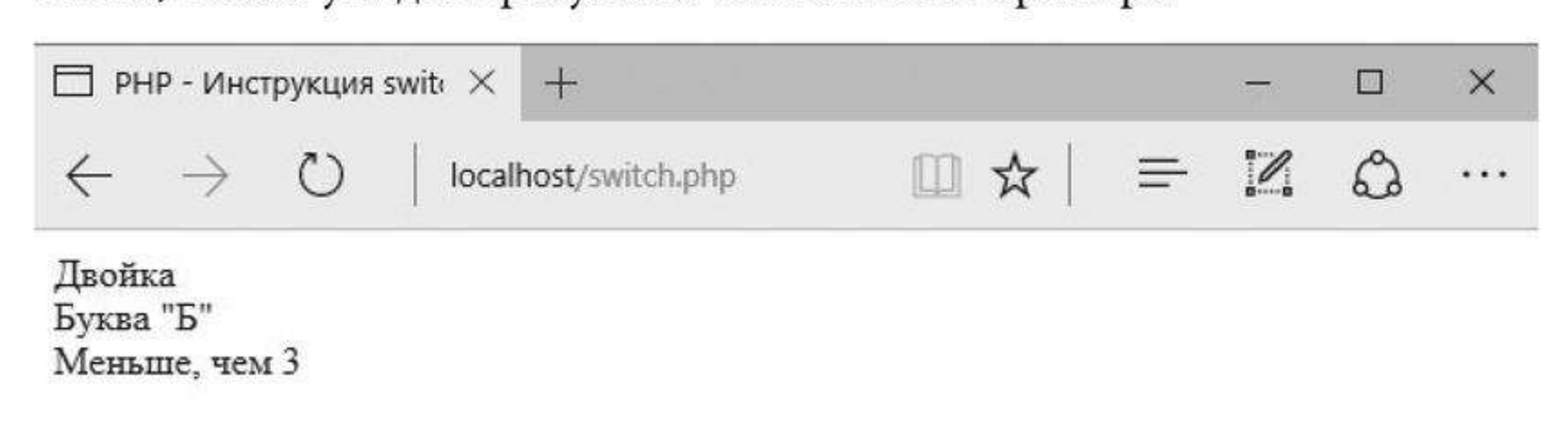
4. Наконец, укажите конструкцию `switch`, инструкции которой сверяют несколько целочисленных значений.

```
switch ($number)
{
 case 0: case 1: case 2: echo 'Меньше, чем 3<br>'; break;
 default: echo '3 или больше, либо меньше нуля';
}
?>
```

Совет

В конструкциях `switch` ключевое слов `case`, проверяемое *`значение`* и символ двоеточия (`:`) распознаются как уникальная «метка».

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *switch.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.



Работа с циклами

Цикл представляет собой фрагмент кода, который автоматически повторяется в сценарии. Одно полное выполнение всех инструкций внутри цикла называется «итерацией» или «проходом». Продолжительность выполнения цикла определяется условной проверкой, выполняемой внутри него. Пока проверяемое выражение истинно, цикл будет выполняться. Как только проверяемое выражение стало ложным, цикл завершает свою работу.

В языке PHP доступны циклы четырех типов:

- `for` — выполняется заданное число итераций;
- `while` — выполняется, пока выражение в начале каждой итерации определяется как истинное;
- `do while` — выполняется, пока выражение в конце каждой итерации определяется как истинное;
- `foreach` — итерации выполняются для обхода всех элементов массива, с которыми вы познакомились в главе 2.

Пожалуй, самым интересным является цикл `for`, который имеет следующий общий синтаксис:

```
for (инициализатор; выражение; итератор)
{
 инструкция-если-выражение-истинно;
 инструкция-если-выражение-истинно;
 инструкция-если-выражение-истинно;
}
```

Инициализатор используется для установки начального значения счетчика количества итераций, выполненных циклом. Для этой цели используется целочисленная переменная, которой обычно присваивается имя `i`.

Совет

Итератор в цикле может быть как инкрементом, так и декрементом, который отсчитывает количество итераций, а не подсчитывает.

После каждой итерации цикла определяется логическое значение проверяемого выражения, и выполнение цикла продолжается только в том случае, если выражение истинно. Если проверяемое выражение становится ложным, цикл завершается немедленно без выполнения инструкций. После каждой итерации счетчик цикла обновляется, а затем выполняются инструкции.

Циклы могут быть вложены друг в друга, обеспечивая полное выполнение всех итераций вложенного цикла при каждой итерации внешнего цикла.

1. Начните сценарий с добавления между PHP-тегами кода цикла `for`, состоящего из трех итераций и подсчитывающего их.

forloop.php

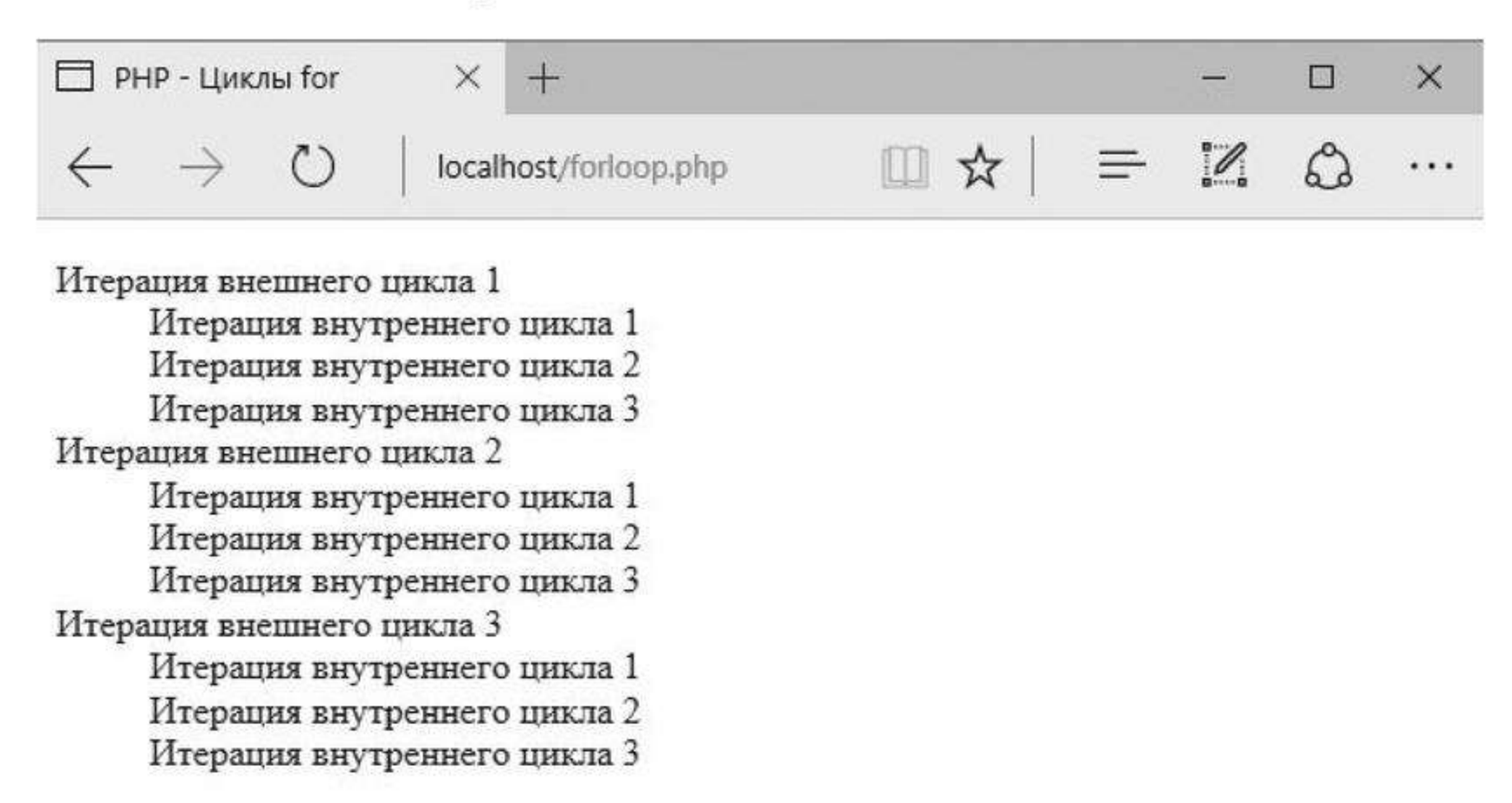
```
<?php for ($i = 1; $i < 4; $i++)
{
 echo "<dt>Итерация внешнего цикла $i";

 # Сюда будет помещен код внутреннего цикла.
}
?>
```

2. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *forloop.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения цикла и счетчик итераций.

3. Теперь добавьте вложенный цикл `for` для вывода его счетчика итераций (трех) в виде списка.

```
for ($j = 1; $j < 4; $j++)
{
 echo "<dd>Итерация внутреннего цикла $j";
}
```

4 Сохраните документ еще раз, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения цикла и вывод счетчика итераций.



На заметку

Счетчик цикла также может отсчитывать итерации. Для этого надо сменить инкремент в итераторе на декремент, используя код `$i–` вместо `$i++`.

Циклы `while` и `do while`

Цикл `while` — альтернатива цикла `for`, описанного в предыдущем примере. В цикле `while` также указываются инициализатор, проверяемое выражение и итератор, но они не так аккуратно вложены в одну пару скобок, как в цикле `for`.

Вместо этого инициализатор указывается перед блоком цикла, а проверяемое выражение — в скобках после ключевого слова `while`. Затем, в фигурных скобках, указываются инструкции и итератор, которые должны выполняться во время каждой итерации.

Внимание

Убедитесь, что проверяемое выражение в ходе цикла в определенный момент становится ложным, чтобы избежать бесконечного цикла, который никогда не завершит свою работу.

```
инициализатор
while (выражение)
{
   выполняемые-инструкции;
   итератор;
}
```

Синтаксис цикла `do while` выглядит чуть иначе: блок цикла заключен в фигурные скобки и предварен ключевым словом `do`, а инструкция `while` располагается уже после блока цикла:

```
инициализатор
do {
   выполняемые-инструкции;
   итератор;
}
while (выражение);
```

Совет

В последующих главах этой книги вы узнаете о том, как в языке PHP цикл `while` используется для извлечения записей из базы данных MySQL.

Оба цикла, `while` и `do while`, выполняются, пока проверяемое выражение остается истинным. Как только выражение становится ложным, цикл завершает работу. Поэтому крайне важно, чтобы тело цикла содержало код, который в определенный момент мог изменить результат проверки выражения — в противном случае цикл с бесконечным числом итераций приведет к зависанию системы. Существенное различие между циклами `while` и `do while` в том, что цикл `while` не выполнит ни одной итерации, если при первой проверке выражение окажется ложным. А цикл `do while` наоборот при запуске выполнит одну итерацию, после чего выполнит проверку выражения. Если необходимо выполнить цикл хотя бы однократно, выберите цикл `do while`, в противном случае выбор между циклами `for` и `while` зависит от ваших личных предпочтений. Цикл `for` рекомендуется использовать для выполнения определенного количества итераций, а цикл `while` — для выполнения итераций, пока определенное условие не будет выполнено. Циклы прекрасно подходят для использования с массивами, так как при каждой итерации может осуществляться последовательное считывание или запись элементов массива.

1. Начните сценарий с добавления между PHP-тегами инструкцию, создающую массив.

whileloop.php

```
<?php
$numbers = array(10, 20, 30);
```

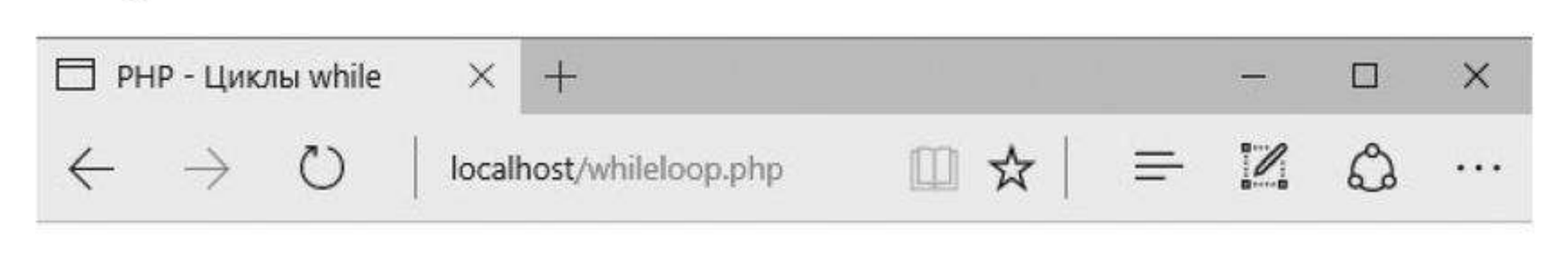
2. Далее добавьте код цикла `while` для вывода каждого значения массива.

```
echo '<dt>Цикл while: ';
$i = 0;
while ($i < 3)
{
 echo "<dd>Элемент $i = $numbers[$i] ";
 $i++;
}
```

3. Теперь добавьте цикл `do while` для вывода значения каждого элемента массива.

```
echo '<dt>Цикл do while: ';
$i = 0;
do
{
 echo "<dd>Элемент $i = $numbers[$i] ";
 $i++;
}
while ($i < 3);
?>
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *whileloop.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть, что элементы массива с помощью разных циклов отображаются одинаково.



Цикл while:
Элемент 0 = 10
Элемент 1 = 20
Элемент 2 = 30
Цикл do while:
Элемент 0 = 10
Элемент 1 = 20
Элемент 2 = 30

На заметку

Измените значение проверяемого выражения в этом примере, сделав его равным нулю, т.е. замените строки кода `while ($i < 3)` в каждом цикле на `while ($i = 0)` — вы увидите, что будет выполнена только одна итерация цикла `do while`.

Прерывание циклов

Ключевое слово `break` используется для досрочного прекращения работы цикла, когда выполнено заданное условие. Инструкция `break` располагается внутри блока инструкций цикла и предшествует проверяемому выражению. Если проверка выражения возвращает значение `TRUE`, цикл завершается немедленно и программа переходит к следующей задаче. Например, во вложенном внутреннем цикле происходит переход к следующей итерации внешнего цикла.

1. Начните сценарий с добавления между PHP-тегами цикла `for`, который выполняет три итерации.

breakcontinue.php

```
<?php for ($i = 1; $i < 4; $i++)
{
 # Вложенный цикл вставляется сюда (шаг 2).
}
```

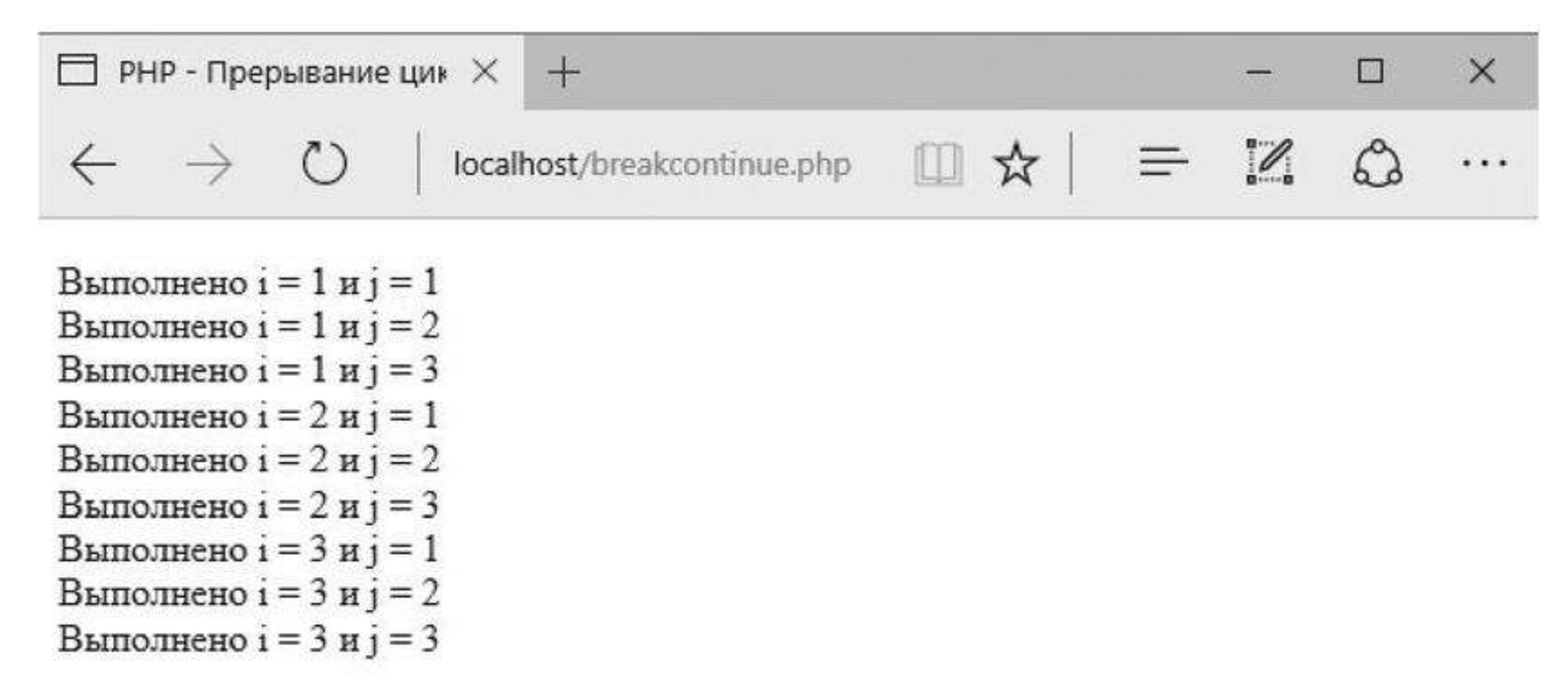
2. Далее напишите код вложенного цикла `for`, который также выполняет три итерации.

```
for ($j = 1; $j < 4; $j++)
{
 # Сюда вставляются инструкции (шаги 5 и 6).
}
?>
```

3. Теперь добавьте инструкцию, которая выводит значения обоих счетчиков циклов при каждой итерации.

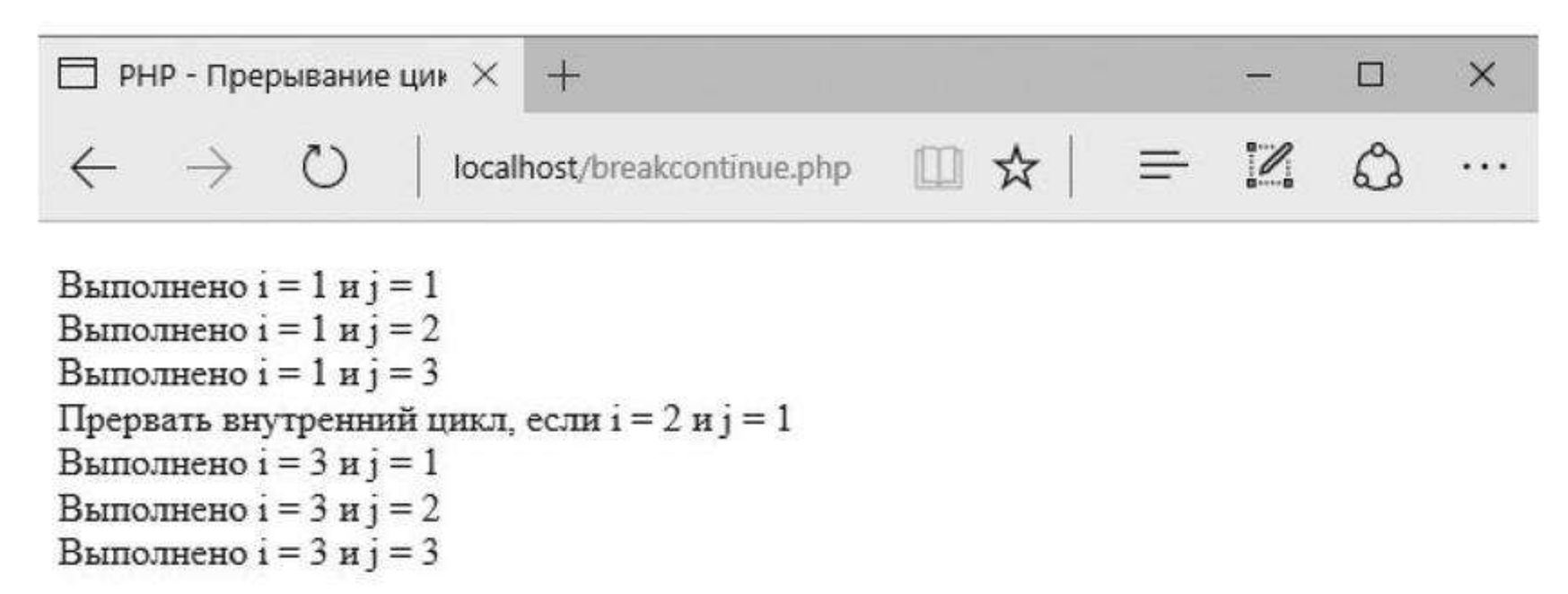
```
echo "Выполнено i = $i и j = $j <br>";
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *breakcontinue.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть значения счетчиков обоих циклов.



5 Затем добавьте следующую инструкцию `break` в самом начале блока кода внутреннего цикла, чтобы прерывать внутренний цикл. После чего сохраните документ и просмотрите изменения.

```
if ($i == 2 && $j == 1)
{
 echo "Прервать внутренний цикл, если i = $i и j = $j <br>";
 break;
}
```



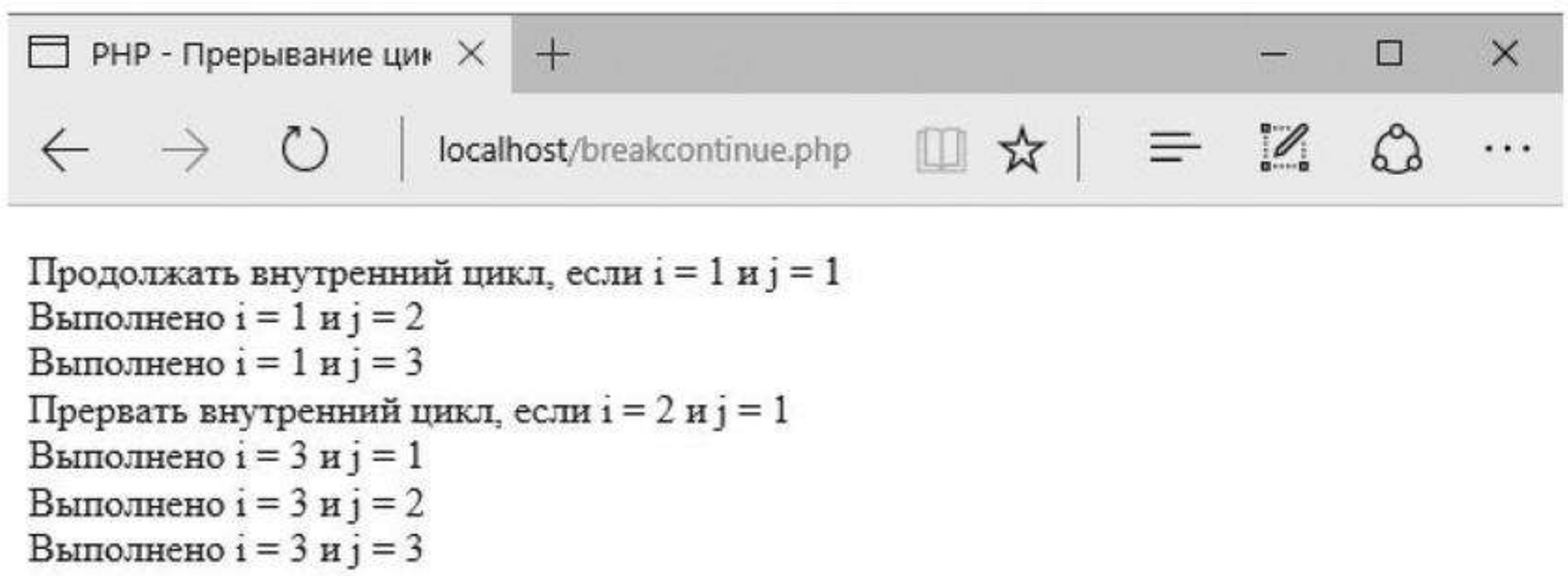
На заметку

В данном примере инструкция `break` прерывает все три итерации внутреннего цикла, когда внешний цикл пытается запустить его повторно.

Ключевое слово `continue` можно использовать, чтобы пропустить одну итерацию цикла, когда выполнено заданное условие. Инструкция `continue` указывается внутри блока кода цикла и предшествует проверяемому выражению. Если проверка выражения возвращает значение `TRUE`, текущая итерации завершается.

6 Теперь добавьте следующую инструкцию `continue` в начале кода внутреннего цикла, чтобы пропустить первую итерацию внутреннего цикла. Затем сохраните документ и просмотрите изменения.

```
if ($i == 1 && $j == 1)
{
 echo "Продолжать внутренний цикл, если i = $i и j = $j
 <br>";
 continue;
}
```



На заметку

В данном примере инструкция `continue` просто пропускает первую итерацию внутреннего цикла, когда внешний цикл пытается запустить его в первый раз.

Заключение

- Ключевое слово `if` выполняет простую проверку условия, вычисляя логическое значение `TRUE` или `FALSE` проверяемого выражения.
- Инструкции в фигурных скобках после проверяемого выражения будут выполнены только в том случае, если выражение истинно (`TRUE`).
- Несколько выражений можно проверить с помощью логического оператора `&&` или вкладывая условные конструкции `if` друг в друга.
- Ключевое слово `else` определяет инструкции, которые выполняются, если проверяемое выражение ложно (`FALSE`).

- Ключевое слово `elseif` используется для добавления альтернативного проверяемого выражения и инструкций, которые будут выполнены, если выражение истинно (`TRUE`).
- Группы инструкций `if-elseif-else` могут выполняться более эффективно с помощью конструкции `switch`.
- Ключевое слово `case` используется для указания значения, на совпадение с которым проверяется значение в коде конструкции `switch`.
- Каждая инструкция `case` должна заканчиваться ключевым словом `break`, чтобы конструкция `switch` завершала выполнение при обнаружении совпадения.
- Инструкция `default` определяет код, который будет выполняться в том случае, если во всех инструкциях `case` совпадений не обнаружено.
- Цикл представляет собой фрагмент кода, который автоматически повторяется до тех пор, пока проверяемое выражение не станет ложным.
- В языке PHP используются циклы `for`, `while`, `do while` и `foreach`.
- Цикл `for` включает инициализатор, проверяемое выражение и итератор, который увеличивает (или уменьшает) счетчик итераций.
- Итератор всегда должен быть включен в блок инструкций циклов `while` и `do while`.
- В отличие от цикла `while`, цикл `do while` при запуске однократно выполняет вложенные инструкции, прежде чем будет выполнена проверка выражения.
- Ключевое слово `break` используется для досрочного прерывания цикла, если выполнено заданное условие.
- Ключевое слово `continue` используется для пропуска одной итерации цикла, если выполнено заданное условие.

5

Функции

Эта глава расскажет вам, как создавать многократно используемые блоки кода с помощью функций.

- Определение функций
- Передача аргументов
- Параметры функции
- Область видимости
- Возврат значений
- Анонимные функции
- Заключение

Затем в PHP-сценарии можно указать вызов определенной функции по имени с помощью кода *имя_функции*`();`. После вызова функции будут выполнены инструкции, которые она содержит, если это требуется. Хотя интерпретатор выполняет код сценария сверху вниз, вызов функции может оказаться перед определением функции. Это не самая удачная идея, поэтому вы всегда должны определять функции, прежде чем они будут вызываться.

Функции могут содержать любой допустимый PHP-код и даже определения дополнительных функций. В этом случае внутренняя функция не будет определена до тех пор, пока внешняя функция не вызвана.

1. Начните PHP-сценарий с инструкции, определяющей функцию `greet()`.

function.php

```
<?php function greet()
{
 echo 'Привет, пользователь!<br>';
}
```

2. Затем добавьте код вызова функции, определенной на предыдущем шаге, чтобы она выполнила вложенные в нее инструкции.

```
greet();
```

3. Теперь определите еще одну функцию, которая содержит вложенное определение функции и инструкцию.

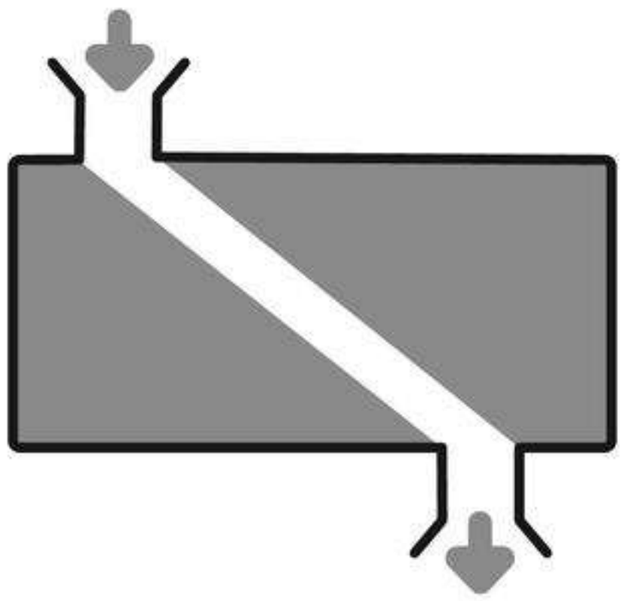
```
function outer()
{
 function inner()
 {
  echo 'Вложенная функция вызвана.<br>';
 }
 echo 'Вложенная функция создана.<br>';
}
```

4. Наконец, добавьте инструкции для вызова каждой из функций, определенных ранее.

```
outer();
inner();
?>
```

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *function.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения каждой функции.

Определение функций



Язык PHP содержит множество встроенных функций, которые могут быть вызваны по имени в сценариях для выполнения заранее определенных действий, таких как, например, сортировка массивов, как это описано в главе 2. Кроме того, вы можете создавать собственные функции для выполнения нужных действий в результате их вызова. Функции очень удобны, так как позволяют разделять код PHP-сценариев на модули, упрощая чтение и поддержку кода.

Пользовательские имена функций могут содержать любую комбинацию латинских букв, цифр и символа подчеркивания, но должны начинаться либо с буквы, либо с символа подчеркивания. Разумеется, во избежание конфликта пользовательские имена не должны совпадать с именами встроенных функций, поэтому, к примеру, вы не можете создать пользовательскую функцию с именем `sort`.

В языке PHP, в отличие от переменных, имена функций не чувствительны к регистру символов, поэтому `cube()`, `Cube()` и `CUBE()` — это отсылка на одну и ту же функцию. Тем не менее рекомендуется всегда использовать один и тот же регистр имен, как в определении, так и в вызове функции.

На заметку

В языке PHP имена переменных чувствительны к регистру, а имена функций — нет.

В PHP-сценариях определение функции осуществляется указанием ее имени после ключевого слова `function`, после чего пишутся круглые скобки (для параметров) и фигурные скобки, содержащие инструкции, которые будут выполняться каждый раз при вызове данной функции. Общий синтаксис функции показан ниже.

```
function имя_функции ()
{
 выполняемая_инструкция;
 выполняемая_инструкция;
 выполняемая_инструкция;
}
```

Совет

Удалите вызов функции `outer()`, чтобы увидеть ошибку с сообщением о неопределенной функции, когда сценарий попытается вызвать внутреннюю функцию.

Передача аргументов

Данные передаются функциям в качестве «аргументов». Это относится и к некоторым встроенным функциям. Например, функция `sort()` принимает в качестве аргумента сортируемый массив. Функции могут быть созданы с неограниченным количеством параметров, но, во избежание ошибок, при вызове аргументы должны предоставляться для каждого параметра.

На заметку

Параметры указываются при объявлении функции после ее имени в скобках, в виде списка инструкций, разделенных запятыми. Значения ***аргументов*** передаются функции при вызове в виде списка значений, разделенных запятыми.

Функция принимает аргументы в виде списка «параметров», разделенного запятыми и указанного в скобках после имени функции, например так:

```
function имя_функции ($параметр_1, $параметр_2,
$параметр_3)
{
 выполняемая_инструкция;
 выполняемая_инструкция;
 выполняемая_инструкция;
}
```

Имена параметров должны соответствовать правилам именования, используемым в языке PHP для имен переменных. В идеале имена должны быть описательными, чтобы было понятно, какие данные они принимают. Значения аргументов во время вызова должны передаваться в правильном порядке, для корректного использования инструкциями внутри функции.

При определении функции можно указать тип данных, которые должны быть переданы от вызывающей программы, добавив его перед именем параметра. Допустимые типы данных включают в себя `int`, `float`, `string`, `bool`, `callable` и `array`. При непосредственном указании типа данных вызов, пытающийся передать функции данные неправильного типа, приведет к ошибке.

Возможность указания типов данных аргументов впервые появилась в версии PHP 7.

По умолчанию аргументы в функцию передаются по значению. Это означает, что, если в вызове функции указано имя переменной, только копия данных, присвоенных этой переменной, будет передана функции. Таким образом, если вы измените значение аргумента внутри функции, то вне функции значение переменной останется прежним.

Если вы хотите разрешить функции изменять свои аргументы, вы должны передавать их по ссылке. Так функция сможет оперировать непосредственно исходными данными. Изменение значения аргумента повлияет на исходные данные в переменной, когда она передается по ссылке. Если вы хотите, чтобы аргумент всегда передавался по ссылке, вы можете указать символ амперсанда (`&`) перед именем параметра в определении функции.

1. Начните PHP-сценарий с инструкции, инициализирующей две переменные с одинаковым целочисленным значением.

arguments.php

```
<?php
$a = $b = 5;
```

2. Затем определите функцию, которая принимает два целочисленных аргумента — один по значению, а другой по ссылке.

```
function modify(int $val, int &$ref)
{
// Сюда вставляются инструкции.
}
```

3. Теперь добавьте инструкции для вывода переданных значений, а затем инкрементируйте эти значения и также выведите их.

```
echo "Переданные значения: $val, $ref<br>";
$val++;
$ref++;
echo "Инкрементированные значения: $val, $ref<hr>";
```

4. После кода функции добавьте инструкцию ее вызова, передав значения переменных в качестве аргументов.

```
modify($a, $b);
```

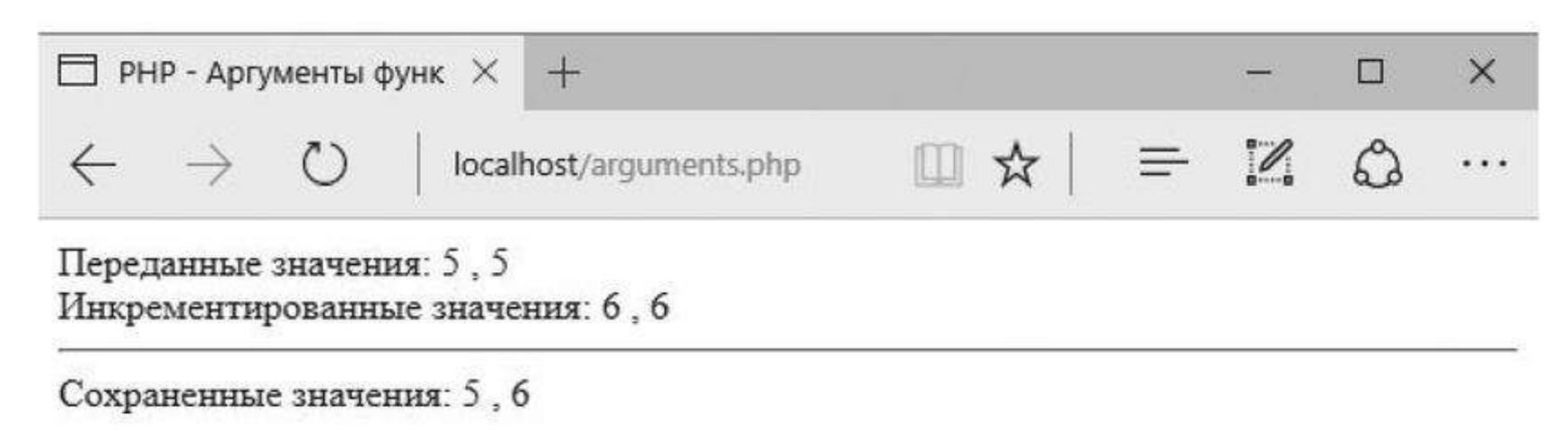
На заметку

Объявление типа `int` в данном примере определяет, что только целочисленные значения данных могут быть переданы в качестве аргументов функции.

5. Наконец, добавьте инструкцию для вывода значений, хранящихся в каждой переменной после выполнения функции.

```
echo "Сохраненные значения: $a, $b";
?>
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *arguments.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат обработки значений переменных.



Параметры функции

Если необходимо, при определении функции можно указать значения аргументов по умолчанию. Значения присваиваются к имени параметра, например так:

```
function имя_функции ($параметр_1, $параметр_2 =
"Строка по умолчанию")
{
 выполняемая_инструкция;
 выполняемая_инструкция;
 выполняемая_инструкция;
}
```

Внимание

Если в определении функции значения по умолчанию указаны для нескольких параметров, нельзя пропустить тот или иной параметр при вызове функции и присвоить значения определенным параметрам.

После этого при вызове функции будут определены один или два значения аргумента. Если указан только один аргумент, его значение будет использоваться взамен первого параметра, а для второго будет использоваться значение по умолчанию. Все параметры, для которых установлены значения по умолчанию, должны находиться после параметров, для которых значения по умолчанию не заданы. В противном случае вызов функции без правильного числа аргументов вызовет ошибку.

Если необходимо создать функцию, принимающую неизвестное (переменное) количество аргументов, в PHP используется специальный символ многоточия, (оператор ...). Он указывается в определении функции следующим образом:

```
function имя_функции (...$параметры)
{
 выполняемая_инструкция;
 выполняемая_инструкция;
 выполняемая_инструкция;
}
```

Когда в определении функции в качестве префикса имени параметра указывается оператор ..., этому параметру может быть передано несколько аргументов

для создания массива. В данном случае массив может быть обработан, как и в других ситуациях, для работы с переданными в аргументах значениях.

Кроме того, оператор ... может быть использован для «распаковки» массива, указанного в вызове функции, и передачи значений его элементов в виде отдельных аргументов следующим образом:

```
function имя_функции ($параметр_1, $параметр_2,
$параметр_3)
{
 echo $параметр_1 + $параметр_2 + $параметр_3;
}

$arr = [1, 2, 3];

имя_функции (...$arr); # Выводы 6
```

pfrfmeters.php

1. Начните PHP-сценарий с определения функции, содержащей строковые значения по умолчанию, присвоенные ее двум параметрам.

```
<?php function drink(string $tmp = 'горячий',
string $flavor = 'чай')
{
 // Сюда помещаются инструкции.
}
```

2. Затем добавьте инструкцию, выводящую значения по умолчанию или значения, переданные с аргументами.

```
echo "Пейте $tmp $flavor.<br>";
```

3. После кода функции добавьте вызовы функций, которые передают строковые аргументы.

```
drink(); drink('холодный'); drink('охлажденный', 'лимонад');
```

4. Теперь определите функцию, которая может принимать несколько целочисленных аргументов и отображать их общее количество.

```
function add(int ...$numbers)
{
 $total = 0;
 foreach($numbers as $num) {$total += $num;}
 echo "<hr>Итого: $total";
}
```

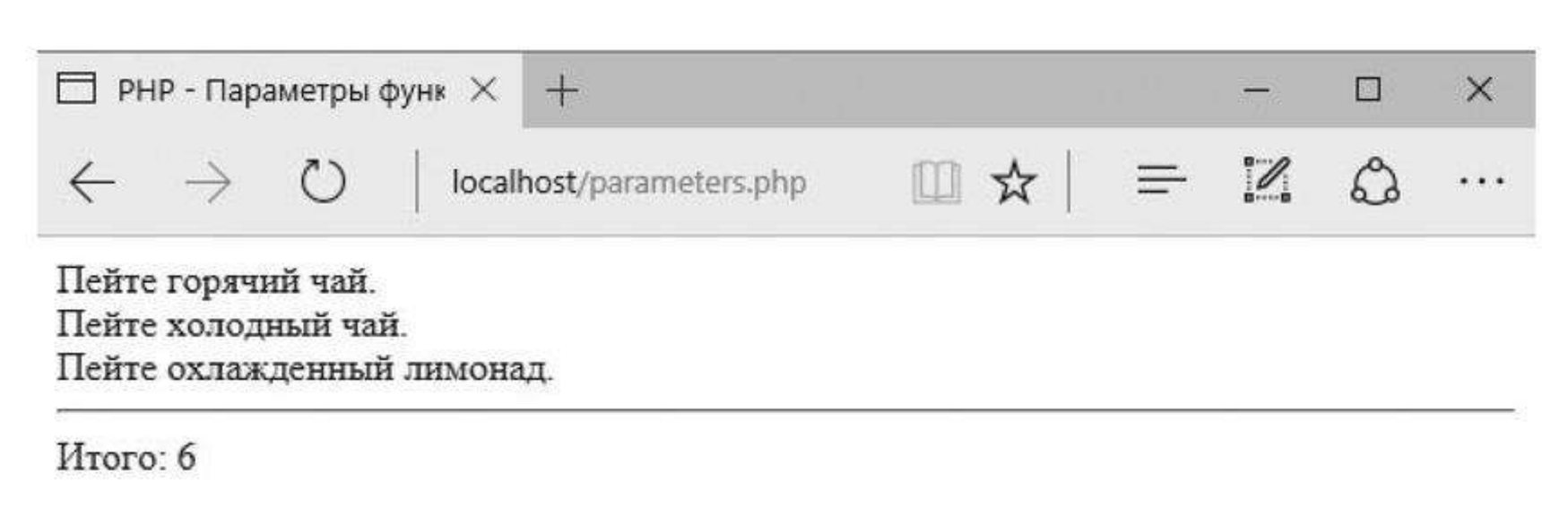
Совет

Можно указывать обычные параметры перед параметром, принимающим несколько аргументов с помощью оператора `...` Только те аргументы, которые превышают обычные элементы, будут добавлены в массив финального параметра.

5. Наконец, после кода функции добавьте инструкцию вызова функции из предыдущего шага и передайте три значения.

```
add(1, 2, 3);
?>
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *parameters.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат передачи аргументов и значения по умолчанию.



Область видимости

При создании пользовательских функций важно понимать, как доступность переменной ограничивается ее «областью видимости»:

- переменные, созданные вне функции, имеют «глобальную» область видимости, но обычно недоступны из любой функции;
- переменные, создаваемые внутри функции, имеют «локальную» область видимости, так как доступны только внутри этой конкретной функции.

Так как переменная, созданная внутри функции, недоступна «снаружи», ее имя может быть таким же, как имя переменной вне функции или внутри другой функции, конфликта имен при этом не возникнет. Это позволяет глобальной

переменной хранить значение, отличное от значения локальной переменной с тем же именем.

Если вы хотите использовать глобальную переменную внутри функции, вы должны сначала добавить инструкцию с ключевым словом `global`, чтобы объявить, что с указанным именем создается глобальная переменная:

```
global имя_переменной;
```

Внимание

Глобальные переменные в языке PHP отличаются от аналогов в других языках программирования, таких как C, где они автоматически становятся доступны функциям. В языке PHP глобальные переменные могут использоваться в функциях, только если сначала явно объявлены внутри блока функции с помощью ключевого слова `global`.

Область видимости переменной, созданной в пределах функции, может быть изменена на глобальную путем объявления с использованием ключевого слова `global` — но это нерекомендуемый подход, которого следует избегать.

Код функции может содержать вызовы других функций, выполняющих, в свою очередь, собственные инструкции, а также вызовы самих себя, что называется рекурсией. Как и в случае с циклами, рекурсивная функция должна изменить состояние проверяемого условия, чтобы не выполняться бесконечно. Условием может быть значение счетчика, передаваемое в качестве аргумента при вызове начальной функции и инкрементируемое при успешных рекурсивных вызовах, пока счетчик не превышает значение, указанное в проверяемом условии. С другой стороны, статическая локальная переменная может быть объявлена для хранения инкрементируемого значения:

```
static имя_переменной;
```

В отличие от других локальных переменных, которые теряют свое значение, когда выполнение программы выходит из этой области видимости, статические переменные сохраняют свое значение. Это означает, что обновленное значение инкремента, хранящегося в статической локальной переменной, распознается при каждом вызове функции.

Рекурсивная функция может ссылаться на инкрементируемое при успешных рекурсивных вызовах значение счетчика глобальной переменной, пока счетчик не превышает значение, указанное в проверяемом условии.

1. Начните PHP-сценарий с инициализации глобальной целочисленной переменной и вывода значения переменной.

scope.php

```
<?php
$number = 1; echo "Глобальное число:
$number<br>";
```

2. Далее определите и вызовите функцию, которая инициализирует одноименную локальную переменную и выводит значение этой переменной.

```
function show_local()
{
 $number = 100; echo "Локальное число: $number<hr>";
}
show_local();
```

3. Теперь определите и вызовите рекурсивную функцию, которая инкрементирует значения глобальной и статической переменных во время каждого вызова.

```
function recur()
{
 global $number; static $letter = 'A';
 if($number < 14)
 {
  echo "$number:$letter | ";
  $number++; $letter++; recur();
 }
}
recur();
```

На заметку

Если вы удалите ключевое слово `static` из данной функции, переменная `$letter` будет содержать одно и то же значение `'A'` при каждом вызове функции.

4. Наконец, после кода функции добавьте инструкцию для вывода измененного значения глобальной переменной.

```
echo "<hr>Глобальное число: $number";
?>
```

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *scope.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат обработки значений глобальных и локальных переменных.

Совет

В языке PHP доступен специальный массив `$GLOBALS`, который содержит **ссылки на все переменные глобальной области видимости**. Как вариант в этом примере можно использовать код `$GLOBALS['number']`, чтобы получить значение.

Возврат значений

Пользовательские функции в языке PHP позволяют вернуть значение вызывающей программе, если добавить инструкцию с ключевым словом `return`:

```
function имя_функции ()
{
 выполняемая_инструкция;
 выполняемая_инструкция;
 return значение;
}
```

Инструкция `return` необязательна, и если она не указана, то по умолчанию функции возвращают вызывающей программе значение `NULL`.

Функции позволяют возвращать только один элемент вызывающей программе, поэтому вы не сможете вернуть несколько значений. Однако вы можете присвоить несколько значений элементам массива, затем функция `return` вернет вызывающей программе единый массив, из которого будут распакованы значения.

Инструкция `return` может включать в себя проверяемое выражение, согласно которому возвращаться будет единственный соответствующий результат. Например:

```
function square(int $number)
{
 return $number * $number;
}

echo square(3); # Вывод 9
```

Чтобы функции возвращали значения только определенного типа, в объявлении функции можно при необходимости указать тип данных возвращаемого значения, например `int`, `float`, `string`, `bool` или `array`:

```
function имя_функции (): int
{
 выполняемая_инструкция;
 выполняемая_инструкция;
 return значение;
}
```

Возможность указания типа возвращаемых данных впервые появилась в версии PHP 7.

Если тип возвращаемых данных указан, функция, пытающаяся вернуть вызывающей программе данные неправильного типа, в результате приведет к ошибке.

1. Начните PHP-сценарий с определения функции, которая должна возвращать только массив данных.

return.php

```
<?php
function supply(): array
{
 // Сюда помещаются инструкции.
}
```

2. Далее добавьте инструкцию возврата массива из значений четырех различных типов.

```
return array (75, 3.142, 'Супер PHP', True);
```

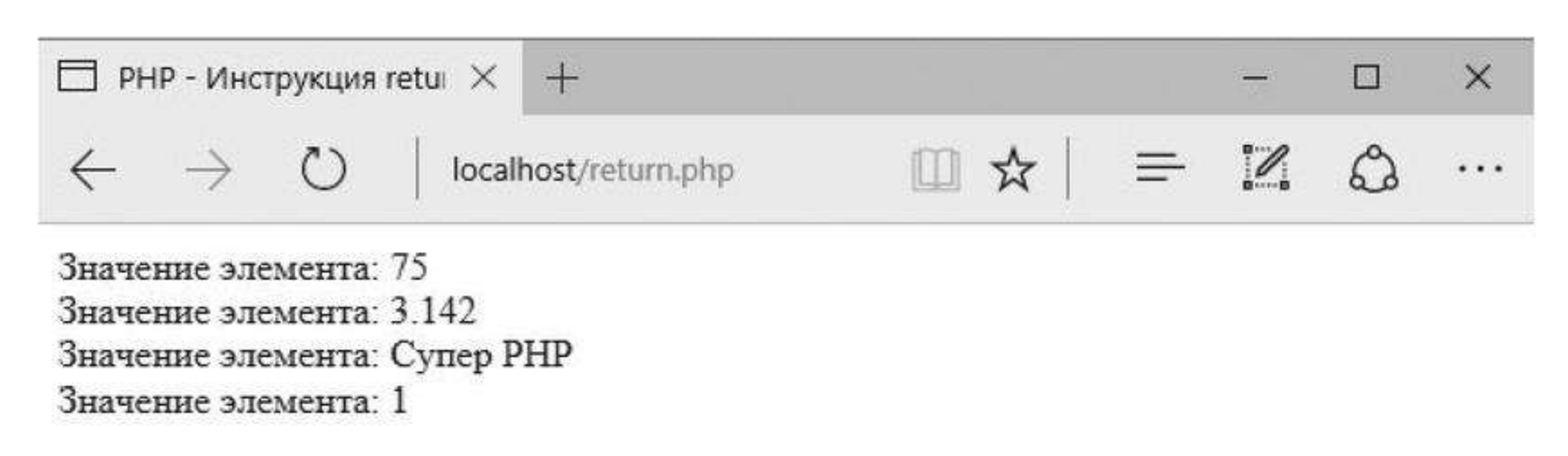
3. После кода функции добавьте инструкцию вызова функции и присвоения возвращенных значений массива переменной.

```
$array = supply();
```

4. Наконец, добавьте цикл для вывода каждого значения, возвращенного из функции.

```
foreach($array as $data)
{
 echo "Значение элемента: $data<br>";
}
?>
```

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *return.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат возврата значений.



На заметку

В языке PHP логическое значение `TRUE` может быть численно представлено как 1.

Обратный вызов

Язык PHP позволяет создавать «анонимные» функции, которым не присвоено имя, указываемое в определении. Как и другие функции, они могут принимать аргументы, если указаны параметры, и при необходимости могут возвращать значение при вызове. В отличие от других определений функций, являющихся конструкциями, определение анонимной функции представляет собой выражение. Это означает, что каждое определение анонимной функции должно заканчиваться точкой с запятой.

```
function ()
{
 выполняемая_инструкция;
 выполняемая_инструкция;
 return значение;
};
```

Внимание

Не забудьте добавить символ `;` (точка с запятой) после каждого определения анонимной функции.

Самостоятельно анонимные функции малопригодны, так как не имеют имен и их нельзя вызвать, но могут быть полезны в других качествах:

- после присвоения переменной анонимная функция может быть вызвана через имя переменной;
- передаваться в качестве вызываемого аргумента другой функции, после чего анонимную функцию можно позднее вызывать с помощью «обратного вызова»;
- возвращаемая из функции анонимная функция сохраняет доступ к переменным возвращающей функции в виде «замыканий».

Назначив анонимную функцию переменной вы по сути присваиваете функции имя и может вызвать ее и передать ей аргументы, как это делается с любой другой функцией, используя имя переменной. Аналогично, когда вы передаете анонимную функцию другой функции, она может быть вызвана обратно с передачей аргументов, как в случае и с любой другой функцией, используя имя параметра. Использование ключевого слова `use` и префикса `&` позволяет анонимной функции ссылаться на локальную переменную в возвращающей функции.

1. Начните PHP-сценарий с кода переменной, которой присваивается анонимная функция, принимающая один аргумент.

callback.php

```
<?php
$hello = function ($user) {echo "Привет,
$user!<br>";};
```

2. Затем добавьте инструкцию, которая вызывает анонимную функцию и передает один аргумент.

```
$hello('Михаил');
```

3. Теперь определите обычную функцию, которая принимает единственный аргумент обратного вызова и вызывает эту функцию, передавая строку.

```
function greet(callable $anon)
{
 $anon('Катя');
}
```

4. Добавьте инструкцию для вызова функции, определенной на предыдущем шаге, передавая в качестве аргумента анонимную функцию, которая была присвоена переменной на первом шаге.

```
greet($hello);
```

5. Далее определите еще одну обычную функцию, содержащую одну локальную переменную и возвращающую анонимную функцию, которая ссылается на значение локальной переменной.

```
function meet(): callable
{
 $time = 'утро';
 return function($name) use(&$time)
 {return "Доброе $time, $name!";};
}
```

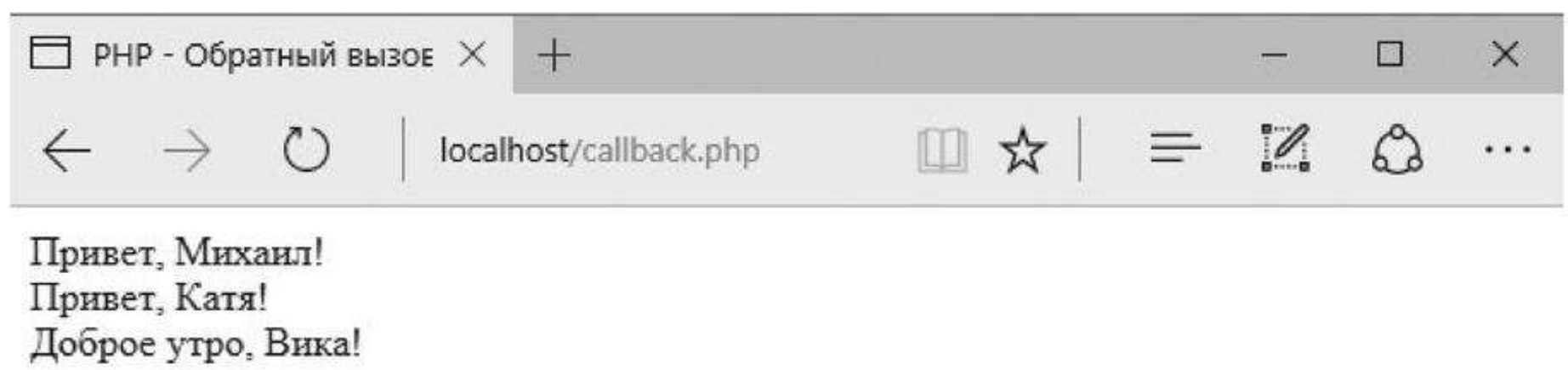

На заметку

Не забудьте добавить префикс `&`, чтобы создать ссылку на значение переменной.

6. Присвойте переменной функцию, определенную на предыдущем шаге, а затем вызовите эту функцию, передав строковый аргумент.

```
$meeting = meet();
echo $meeting('Вика');
?>
```

7. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *callback.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат возвращения значений от анонимных функций.



Совет

Обратите внимание на то, что вызываемая завершающая функция получает строку `'утро'` из родительской области видимости.

Заключение

- Пользовательские функции позволяют разделять код PHP-сценариев на модули, которые проще читать и поддерживать.
- Имена функций не зависят от регистра букв и должны начинаться с латинской буквы или символа подчеркивания.
- Функция определяется путем указания ее имени после ключевого слова `function`, затем в круглых скобках задают ее параметры и в фигурных скобках ее инструкции.
- Функция, принимающая аргументы, определяется со списком параметров, разделенных запятыми и указанных в круглых скобках после имени функции.
- По желанию в определении функции можно указать тип данных, которые должны быть переданы от вызывающей программы, указав тип данных перед именем параметра.
- Данные аргументов обычно передаются по значению, но могут передаваться и по ссылке, если имя параметра сопровождает префикс `&`.
- По желанию при определении функции для аргумента можно указать значение по умолчанию, присвоив его имени параметра.
- Если имя параметра содержит оператор `...`, становится возможным передать этому параметру несколько аргументов, чтобы создать массив.

- Переменные, созданные вне функции, имеют глобальную область видимости, но не доступны из блока функции, если они не задекларированы в ней с помощью ключевого слова `global`.
- Переменные, создаваемые внутри функции, имеют локальную область видимости и доступны только внутри этого конкретного блока функции.
- Локальные переменные теряют свои значения, когда выполнение программы выходит из этой области видимости, а статические (с ключевым словом `static`) переменные сохраняют.
- Ключевое слово `return` возвращает один элемент вызывающей программе, при этом в определении функции можно указать тип возвращаемых данных.
- В определении анонимной функции не указывается ее имя, и, так как по сути это выражение, определение должно заканчиваться точкой с запятой.
- Использование ключевого слова `use` и префикса `&` позволяет анонимной функции ссылаться на локальную переменную в возвращающей функции.

6

Строки

Эта глава познакомит вас с различными способами управления текстовыми строками в PHP-сценариях.

- Сравнение символов
- Поиск текста
- Извлечение подстроки
- Изменение регистра символов
- Форматирование строк
- Вывод даты/времени
- HTML-сущности
- Заключение

Сравнение символов

Язык PHP предоставляет ряд встроенных функций, позволяющих выполнить сравнение двух текстовых строк. Простая функция `strcmp()` принимает два строковых аргумента для сравнения и возвращает одно из трех целочисленных значений в зависимости от результата. Если строки в точности совпадают, `strcmp()` вернет `0`. Если первый строковый аргумент больше второго, `strcmp()` вернет `1`, а если меньше, то `-1`.

Внимание

Строго говоря, возвращаемое функцией `strcmp()` значение представляет собой целое положительное число, ноль или отрицательное целое число — но обычно это значение `1`, `0` или `-1`.

Функция `strncmp()` работает аналогичным образом, но принимает третий целочисленный аргумент, позволяющий указать количество символов с начала строк для сравнения.

Функции `strcmp()` и `strncmp()` учитывают регистр символов в строках, т.е. прописные и строчные буквы, при сравнении строк. Если необходимо не учитывать регистр, вы можете использовать альтернативные функции — `strcasecmp()` и `strncasecmp()`.

Чтобы сравнить результаты работы каждой из четырех функций, нужно разобраться, как фактически выполняется сравнение. Каждый символ имеет стандартный числовой ASCII-код. Строчные буквы от «a» до «z» находятся в диапазоне значений 97–122, а прописные буквы «A» — «Z» — в диапазоне 65–90. Сравнивая первое значение в каждом диапазоне, мы увидим, что строчная «a» (97) имеет больший код, чем «A» (65). Поэтому, если указать это в качестве аргументов функции, `strcmp()` вернет `1`, так как значение первого аргумента больше второго. С другой стороны, если вы укажете «A» (65) в качестве первого аргумента и «a» (97) в качестве второго, то функция `strcmp()` вернет `-1`, так как значение первого аргумента будет меньше второго. При сравнении строк каждый символ сравнивается в последовательности, чтобы гарантировать, что строки, содержащие одни и те же символы, но в разном порядке, не считались по ошибке равны. Например, при сравнении «ABC» и «BAC» код первого символа, «A» (65), будет меньше, чем первый символ во втором значении, «B» (66), поэтому функция `strcmp()` вернет `-1`, так как первый аргумент меньше второго.

На заметку

ASCII — это аббревиатура англ. фразы American standard code for information interchange (Американский стандарт кода обмена информацией), обозначающей стандарт, который определяет универсальную схему кодирования символов.

ASCII-код любого символа можно определить, задав символ в качестве аргумента PHP-функции `ord()`, а длина строки определяется ее указанием в качестве аргумента PHP-функции `strlen()`.

1. Начните PHP-сценарий с инструкций инициализации четырех переменных со строковыми значениями.

characters.php

```
<?php
$str1 = 'PHP для начинающих';
$str2 = 'PHP для начинающих';
$str3 = 'PHP Для Начинающих';
$str4 = 'php для начинающих';
```

2. Далее добавьте инструкции для сравнения строк.

```
echo "'$str1' в сравнении с '$str2': ".strcmp($str1,
$str2).'<br>';
echo "'$str1' в сравнении с '$str3': ".strcmp($str1,
$str3).'<br>';
echo "'$str1' в сравнении с '$str4': ".strcmp($str1,
$str4).'<hr>';
```

3. Теперь добавьте инструкцию сравнения без учета регистра сравниваемых символов.

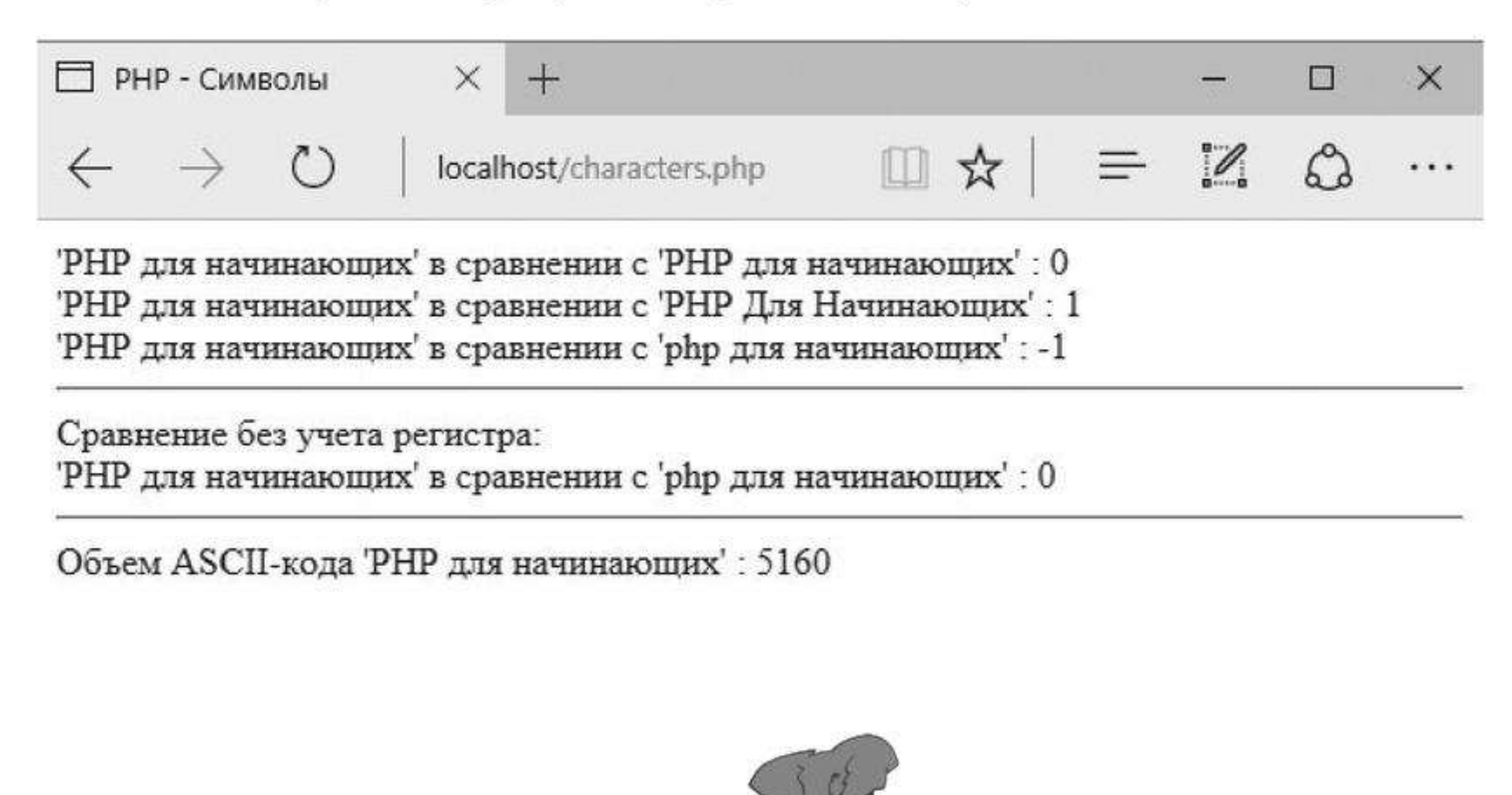
```
echo 'Сравнение без учета регистра:<br>';
echo "'$str1' в сравнении с '$str4': ".strcasecmp($str1,
$str4);
```

4. Наконец, добавьте цикл для подсчета и отображения общего объема ASCII-кода первой строки.

```
$total = 0;
for ($i = 0; $i < strlen($str1); $i++)
{
 $total += ord($str1[$i]);
}
```

```
echo "<hr>Объем ASCII-кода $str1: $total";
?>
```

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *characters.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат сравнения строк.



На заметку

Порядок символов не играет роли для общего объема ASCII-кода: две строки с разным порядком слов (или букв) будут эквивалентны в плане объема кода, но не будут соответствовать друг другу.

Поиск текста

В языке PHP доступно несколько встроенных функций для поиска среди строк определенных подстрок или отдельных символов. Простая функция `strpos()` принимает два аргумента, позволяющие указать строку, в которой будет осуществляться поиск, и искомую подстроку. Поиск осуществляется последовательно от начала строки до обнаружения совпадения. Если совпадение обнаружено, поиск останавливается и функция `strpos()` возвращает целое число, представляющее собой индекс символа в строке, соответствующего первому вхождению совпадения. Если совпадение не обнаружено, функция `strpos()` возвращает значение `FALSE`.

Функция `strrpos()` работает аналогичным образом, но возвращает индекс символа при поиске в обратном порядке, от конца строки.

Другие поисковые функции в PHP позволяют вернуть часть искомой строки, а не числовой индекс позиции символа совпадения.

Функция `strstr()` принимает два аргумента, позволяя указать строку, в которой должен осуществляться поиск, и искомую подстроку. Поиск осуществляется последовательно от начала строки до обнаружения совпадения. При обнаружении совпадения поиск останавливается, и функция `strstr()` возвращает оставшуюся часть искомой строки, откуда начинается первое вхождение совпадения. Если совпадение не обнаружено, функция `strstr()` возвращает `NULL`.

Внимание

Обратите внимание на то, что функции `strrstr()` для поиска в обратном порядке (по аналогии с функцией `strrpos()`) не существует.

Если регистр символов не имеет значения, вы можете использовать функцию `stristr()`, позволяющую выполнять поиск без учета регистра. В результате будет возвращена оставшаяся часть искомой строки, с которой начинается первое вхождение совпадения, без учета регистра.

Функция `strchr()` работает аналогичным образом, но принимает два аргумента, позволяющие указать строку для поиска и один искомый символ. Как можно предположить, функция `strrchr()` осуществляет поиск в обратном порядке, от конца строки. Обе функции возвращают оставшуюся часть искомой строки, с которой начинается первое вхождение совпадения.

1. Начните PHP-сценарий с инструкций, инициализирующих переменную со строковым значением и сообщающих ее длину*.

search.php

```
<?php
$str = 'Most Users usually find PHP useful.';
echo "'$str'<br>Длина строки: ".strlen($str);
```

2. Далее добавьте инструкции для поиска строки в прямом и обратном направлении и вывода позиции данной подстроки.

```
echo "<br>Первое вхождение 'us': ".strpos($str, 'us');
echo "<br>Последнее вхождение 'us': ".strrpos($str, 'us');
```

* Здесь и далее в главе примеры строк приведены на английском языке, так как при поиске кириллического текста некорректно определяется длина строк и регистр символов. То же относится к некоторым функциям, касающимся файловых потоков, в главе 8. — *Прим. перев.*

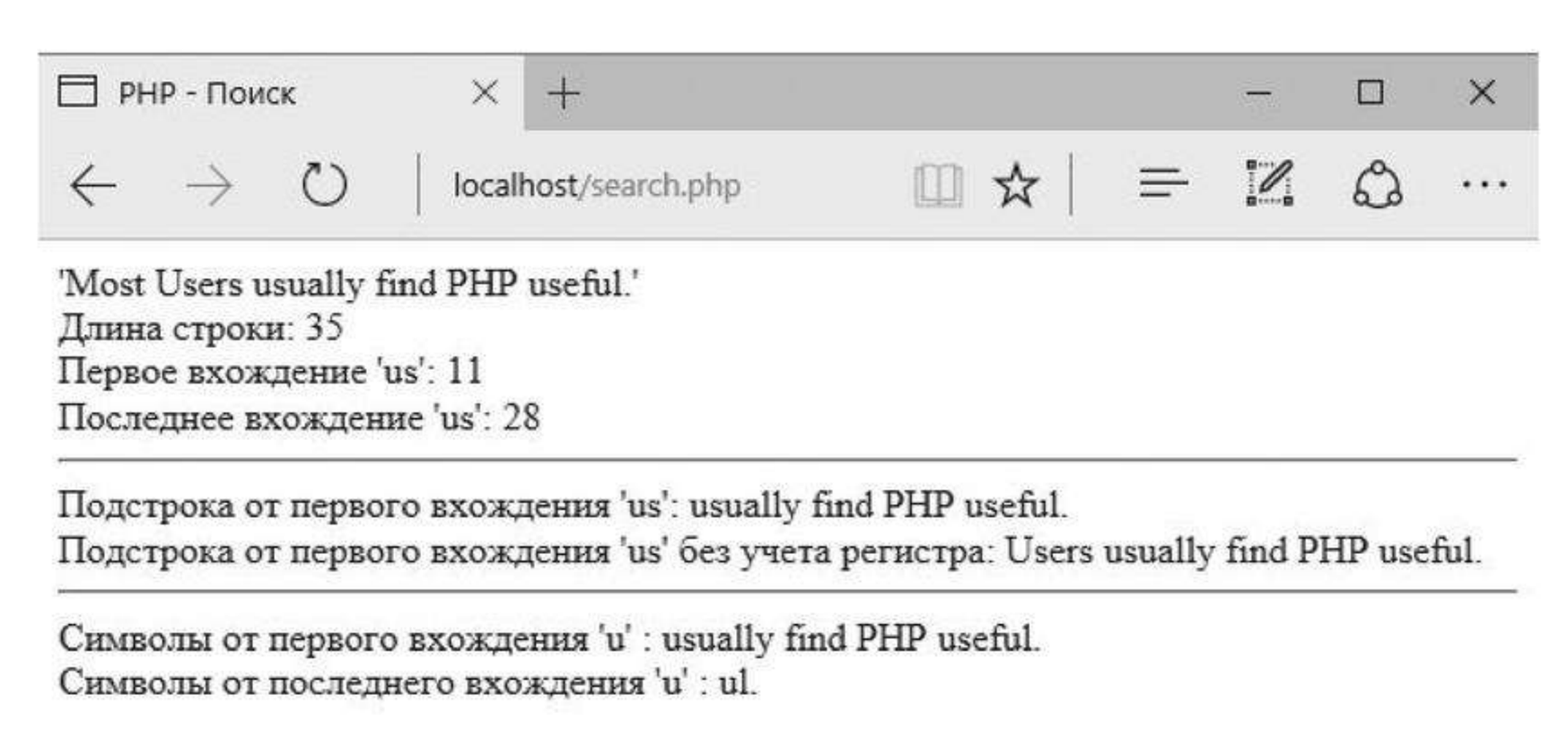
3. Теперь добавьте инструкции для поиска строки с учетом и без учета регистра и вывода строки, оставшейся после подстроки.

```
echo "<hr>Подстрока от первого вхождения 'us':
".strstr($str, 'us');
echo "<br>Подстрока от первого вхождения 'us' без учета
регистра: "
.stristr($str, 'us');
```

4. И, наконец, добавьте инструкции для поиска строки в прямом и обратном направлении и вывода строки, оставшейся после совпавшего символа.

```
echo "<hr>Символы от первого вхождения 'u':
".strchr($str, 'u');
echo "<br>Символы от последнего вхождения 'u':
".strrchr($str, 'u');
?>
```

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *search.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения поиска.



На заметку

В строке есть три вхождения сочетания `'us'`, но подходят только первое и третье.

Извлечение подстроки

Из строки можно извлечь подстроку с помощью функции `substr()`, передав ей аргументы с указанием строки и позиции, откуда следует начинать извлечение. С помощью третьего необязательного аргумента можно указать длину подстроки. Вы можете заменить подстроку с помощью функции `substr_replace()`, передав ей следующие аргументы: строку, подстроку, позицию и длину. Общее количество вхождений подстроки в строке можно определить с помощью функции `substr_count()`.

1. Начните PHP-сценарий с инструкций, инициализирующих переменную со значением в виде строки и счетчиком подстрок.

substring.php

```
<?php
$str = 'SQL in easy steps features SQL queries';
echo "'$str'<br>'SQL' счетчик: ".substr_count($str, 'SQL');
```

2. Далее добавьте инструкции для извлечения двух подстрок.

```
echo '<hr>Позиция 27: '.substr($str, 27);
echo '<hr>Позиция 4, длина 13: '.substr($str, 4, 13);
```

3. Теперь добавьте инструкцию, инициализирующую переменную со строковым значением.

```
$sub = 'PHP & MySQL';
```

4. Наконец, добавьте инструкции для замены содержимого подстроки новым значением и вывода измененной строки полностью.

```
$str = substr_replace($str, $sub, 0, 3);
echo "<hr>'$str'";
?>
```

На заметку

Аргумент длины в функции `substr_replace()` определяет длину заменяемой подстроки. В этом примере 3 означает, что нужно заменить три символа SQL более длинной строкой.

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *substring.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат извлечения подстрок и измененную строку.

Изменение регистра символов

Язык PHP предоставляет функции, позволяющие легко изменить регистр символов заданной строки. Функция `strtolower()` изменяет все буквы на строчные, а `strtoupper()` — на прописные. Кроме того, функция `ucfirst()` меняет на прописную первую букву строки, а функция `ucword()` — первую букву каждого слова в строке.

1. Начните PHP-сценарий с инструкций, инициализирующих переменную со строковым значением и выводящих это значение.

case.php

```
<?php
$str = "C++ Programming in easy steps";
echo "Исходная строка: '$str' <hr>";
```

2. Далее добавьте инструкции, выводящие строку строчными буквами.

```
$ver = strtolower($str);
echo "Строчные буквы: '$ver' <br>";
```

3. Теперь добавьте инструкции для вывода строки прописными буквами.

```
$ver = strtoupper($str);
echo "Прописные буквы: '$ver' <hr>";
```

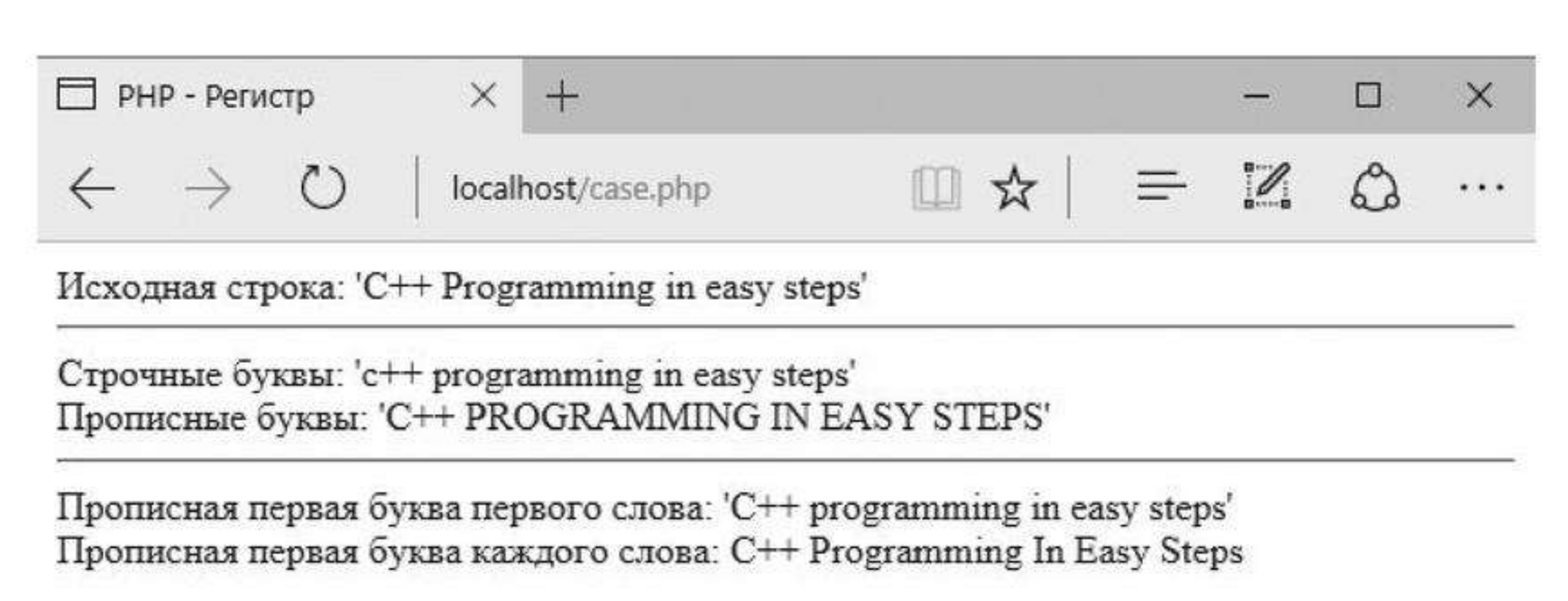
4 Наконец, добавьте инструкции, чтобы вывести варианты строки с прописной первой буквой предложения и прописными первыми буквами всех слов строки.

```
$ver = ucfirst(strtolower($str));
echo "Прописная первая буква первого слова: '$ver' <br>";
echo 'Прописная первая буква каждого слова:
'.ucwords($ver);
?>
```

Внимание

Функция `ucfirst()` обрабатывает только первый символ строки. Другие прописные буквы останутся без изменений, если они не менялись на строчные с помощью функции `strtolower()`, как показано в примере.

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *case.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат изменения регистра букв в строке.



Форматирование строк

В языке PHP доступно несколько встроенных функций для удобного форматирования строк путем управления символами. Простая функция `strrev()` принимает один строковый аргумент и возвращает эту строку в обратном порядке символов, словно «читает задом наперед».

Функция `strrpt()` принимает два аргумента, позволяющие указать строку и целое число, определяющее, сколько раз данная строка должна повторяться в выводе.

Функция `strpad()` принимает три аргумента, позволяющие указать строку, общую длину строки возвращаемых символов и шаблонный символ или подстроку, которыми следует дополнить исходную строку до указанной во втором аргументе длины. Шаблонные символы по умолчанию добавляются в конце исходной строки, но могут быть добавлены и в начало с помощью четвертого аргумента, использующего специальный флаг `STR_PAD_LEFT`.

Функция `trim()` требует один строковый аргумент и возвращает строку, в которой удалены все пробельные символы (пробелы, возврата строки, табуляции), из начала и конца исходной строки. С помощью опционального второго аргумента вместо пробельных символов, использующихся по умолчанию, можно указать символ или группу символов, которые следует удалить из начала и конца исходной строки.

Кроме того, если вы хотите удалить символы только с начала исходной строки, вы можете использовать функцию `ltrim()`, а если с конца строки — то функцию `rtrim()`.

Совет

PHP-функции `implode()` и `explode()` могут использоваться для преобразования массивов в строки и строк в массивы. Для подробной информации см. раздел «Сортировка массивов» в главе 2.

С помощью функции `strtok()` строку можно разделить на подстроки, указав в качестве аргументов строку и маркер-разделитель, которым должна быть разделена строка. Например, если вы хотите разделить строку на отдельные слова, можно указать символ пробела в качестве маркера-разделителя. Первая часть строки до маркера-разделителя будет возвращена функцией. Обратите внимание, что функция `strtok()` запоминает свою позицию в строке, так что в последующих вызовах нужно указать только маркер-разделитель для возврата очередной части строки, до тех пор, пока не будет обнаружен последний разделитель. Подобное разбиение строки лучше всего достигается с помощью цикла, осуществляющего последующие вызовы функции `strtok()`, пока не будет достигнут конец строки.

1. Начните PHP-сценарий с инструкций, инициализирующих переменную со строковым значением и выводящих это значение.

format.php

```
<?php
$str = '| PHP String Fun |'; echo "Исходная строка: $str";
```

2. Далее добавьте инструкции для вывода обращенной, повторяющийся и обрезанной версий исходной строки.

```
echo '<hr>Обращенная строка: '.strrev($str);
echo '<hr>Повторяющаяся строка: '.str_repeat($str, 3);
echo '<hr>Обрезанная строка: '.trim($str, '|');
```

3. Теперь добавьте инструкцию, выводящую строку длиной в 30 символов, дополненную в начале звездочками.

```
$pad = str_pad($str, 30, '*', STR_PAD_LEFT);
echo "<hr>Подстановка символов: $pad";
```

4. Наконец, добавьте инструкции для разбиения строки на подстроки и вывода каждой отдельной части с разделением символом многоточия.

```
echo '<hr>Разделение строки: ';
$token = strtok($str, ' ');
while ($token)
{
echo "$token … "; $token = strtok(' ');
}
?>
```

На заметку

Разделитель следует указывать при каждом последующем вызове функции `strtok()`, чтобы функция ожидала следующей части строки.

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *format.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат форматирования строки несколькими способами.

Вывод даты/времени

PHP-функция `date()` возвращает при вызове дату в виде строки, но требует строковый аргумент, позволяющий указать формат даты. Для форматирования даты и времени доступно множество параметров, указываемых в виде букв. К примеру, функция `date('n/j/y')` возвращает дату в виде `4/22/16`. Распространенные параметры перечислены в таблице ниже с кратким описанием и примерами.

Внимание

Полное наименование дня недели форматируется с помощью строчной латинской буквы L.

Параметр	Описание	Пример
Y	Порядковый номер года, 4 цифры	2014
y	Номер года, 2 цифры	14
n	Порядковый номер (одна или две цифры) месяца	7
m	Порядковый номер месяца (две цифры)	07
F	Полное наименование месяца	February
M	Сокращенное наименование месяца, 3 символа	Feb
j	День месяца (одна или две цифры)	4
d	День месяца (две цифры)	04
l	Полное наименование дня недели	Monday
D	Сокращенное наименование дня недели, 3 символа	Mon
S	Английский суффикс порядкового числительного дня месяца, 2 символа	th
H	Часы в 24-часовом формате	14
i	Минуты	45
s	Секунды	30
a	Ante meridiem (англ. «до полудня») или Post meridiem (англ. «после полудня») в нижнем регистре	am

Совет

Дополнительные параметры PHP-функции `date()` рассматриваются в интернете по адресу **php.net/manual/ru/function.date.php**.

В качестве часового пояса по умолчанию используется всемирное координированное время (англ. Universal Time Clock, UTC), но вы можете использовать собственный часовой пояс, указав его в виде значения аргумента функции `date_default_timezone_set()`. Также можно просмотреть текущий часовой пояс с помощью функции `date_default_timezone_get()`.

Помимо этого вы можете создать собственное отображение меток времени (включая названия дней недели и месяцев на русском языке) с помощью функции `strtotime()`. Она принимает дату в виде строкового аргумента и возвращает метку времени, которая может быть передана в качестве опционального второго аргумента функции `date()`. Аргумент функции `strtotime()` указывается в виде текстовой даты, например «July 4, 2017». Кроме того, вы можете указать в качестве значения аргумента такие параметры, как `"yesterday"`, `"+3months"` или `"next Wednesday"`, и функция `strtotime()` попытается обработать его и вернуть соответствующую метку времени.

1. Начните PHP-сценарий с инструкций, выводящих текущие дату, день недели, время и часовой пояс по умолчанию.

date.php

```
<?php
echo 'Дата: '.date('jS F Y').'<br>';
echo 'День недели: '.date('l').'<br>';
echo 'Время: '.date('h: i: s a').'<br>';
echo 'Часовой пояс: '.date_default_timezone_get().'<hr>';
```

2. Далее добавьте инструкции для изменения часового пояса по умолчанию и вывода уведомления, а также вывода текущего времени.

```
date_default_timezone_set('Europe/Moscow');
echo 'Текущий часовой пояс: '.date_default_timezone_get();
echo '<br>Текущее время: '.date('h: i: s').'<hr>';
```

Совет

Обратите внимание, что при форматировании строк вы можете использовать символы-разделители, такие как и `/`.

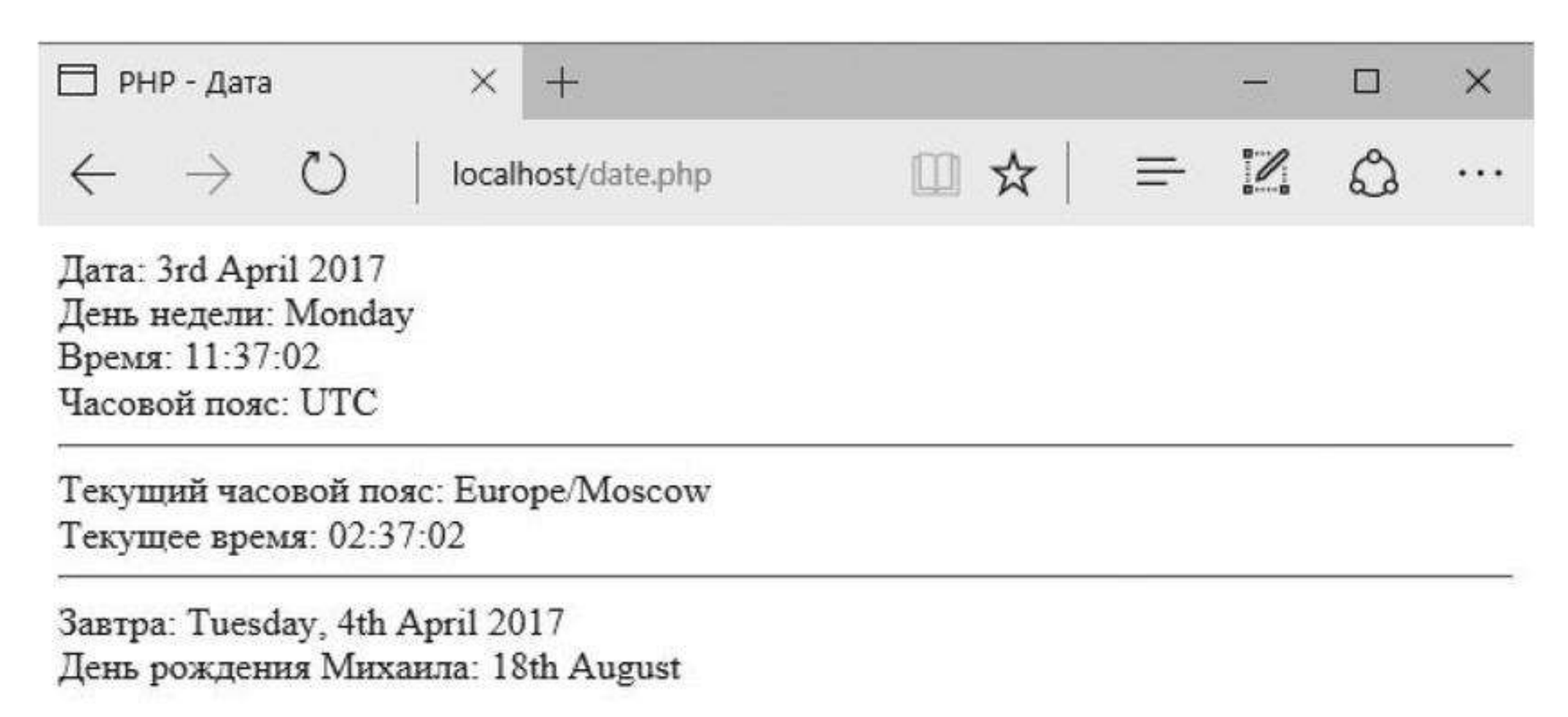
3. Теперь добавьте инструкции, создающие метку времени для отображения следующего дня и выводящие название дня недели и дату.

```
$d = strtotime('tomorrow');
echo 'Завтра: '.date('l, jS F Y', $d).'<br>';
```

4. Наконец добавьте инструкции, создающие метку времени для отображения праздничной даты с выводом дня и названия месяца.

```
$d = strtotime('August 18, 1979');
echo 'День рождения Михаила: '.date('jS F', $d);
?>
```

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *date.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат вывода строк с датами, отформатированных различными способами.



На заметку

В некоторых странах используется формат даты Месяц/День/Год, в то время как в России и других странах распространен формат День/Месяц/Год, поэтому вы можете выбрать предпочтительный вариант оформления даты.

HTML-сущности

В языке PHP есть две функции, предназначенные для обработки HTML-кода, а точнее, для замены специальных символов (которые несут особый смысл и должны быть представлены в виде HTML-сущностей, чтобы сохранить их значение) эквивалентными «сущностями». Функция `htmlentities()` преобразует все символы, указанные ниже, в эквивалентные HTML-сущности.

" & < > ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä
Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è
é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ Œ œ Š š Ÿ ƒ ˆ ˜ Α Β Γ Δ Ε Ζ Η Θ Ι
Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ ς σ τ υ φ
χ ψ ω ϑ ϒ ϖ – — ‘ ’ ‚ “ ” „ † ‡ • … ‰ ′ ″ ‹ › ‾ ⁄ € ℑ ℘ ℜ ™ ℵ ← ↑
→ ↓ ↔ ↵ ⇐ ⇑ ⇒ ⇓ ⇔ ∀ ∂ ∃ ∅ ∇ ∈ ∉ ∋ ∏ ∑ − ∗ √ ∝ ∞ ∠ ∧ ∨ ∩ ∪ ∫ ∴ ∼ ≅
≈ ≠ ≡ ≤ ≥ ⊂ ⊃ ⊄ ⊆ ⊇ ⊕ ⊗ ⊥ ⋅ ⌈ ⌉ ⌊ ⌋ ⟨ ⟩ ◊ ♠ ♣ ♥ ♦

Совет

PHP-функция `get_html_translation_table()` с аргументом-константой `HTML_ENTITIES` или `HTML_SPECIALCHARS` возвращает массив всех преобразованных сущностей.

Если вы волнуетесь за преобразование нелатинских символов, то можете использовать альтернативную функцию `htmlspecialchars()` для преобразования только значимых специальных символов HTML:

- `&` (амперсанд) — `&`
- `"` (двойные кавычки) — `"`
- `'` (одиночные кавычки) — `'`
- `<` (меньше) — `<`
- `>` (больше) — `>`

Положительным эффектом от преобразования HTML-кода с помощью PHP можно считать предотвращение исполнения с помощью веб-браузера вредоносного JavaScript-кода. Преобразование HTML-тегов в сущности гарантирует, что HTML-элемент `script` не будет распознан, а содержащийся в нем JavaScript-код выполнен.

1. Создайте PHP-сценарий с инструкциями, инициализирующими переменную с фрагментом HTML-кода и выводящими его.

encode.php

```
<?php
$html = '<script>window.location="index.html"</script>';
// Сюда помещаются инструкции.
echo $html;
?>
```

2. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *encode.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы, как предполагается, увидеть результат выполнения HTML-кода.

Браузер интерпретирует код и перейдет к главной странице сервера, поскольку распознает HTML-элемент `script`.

3. Вернитесь к PHP-сценарию и добавьте инструкцию, преобразующую специальные символы в HTML-сущности.

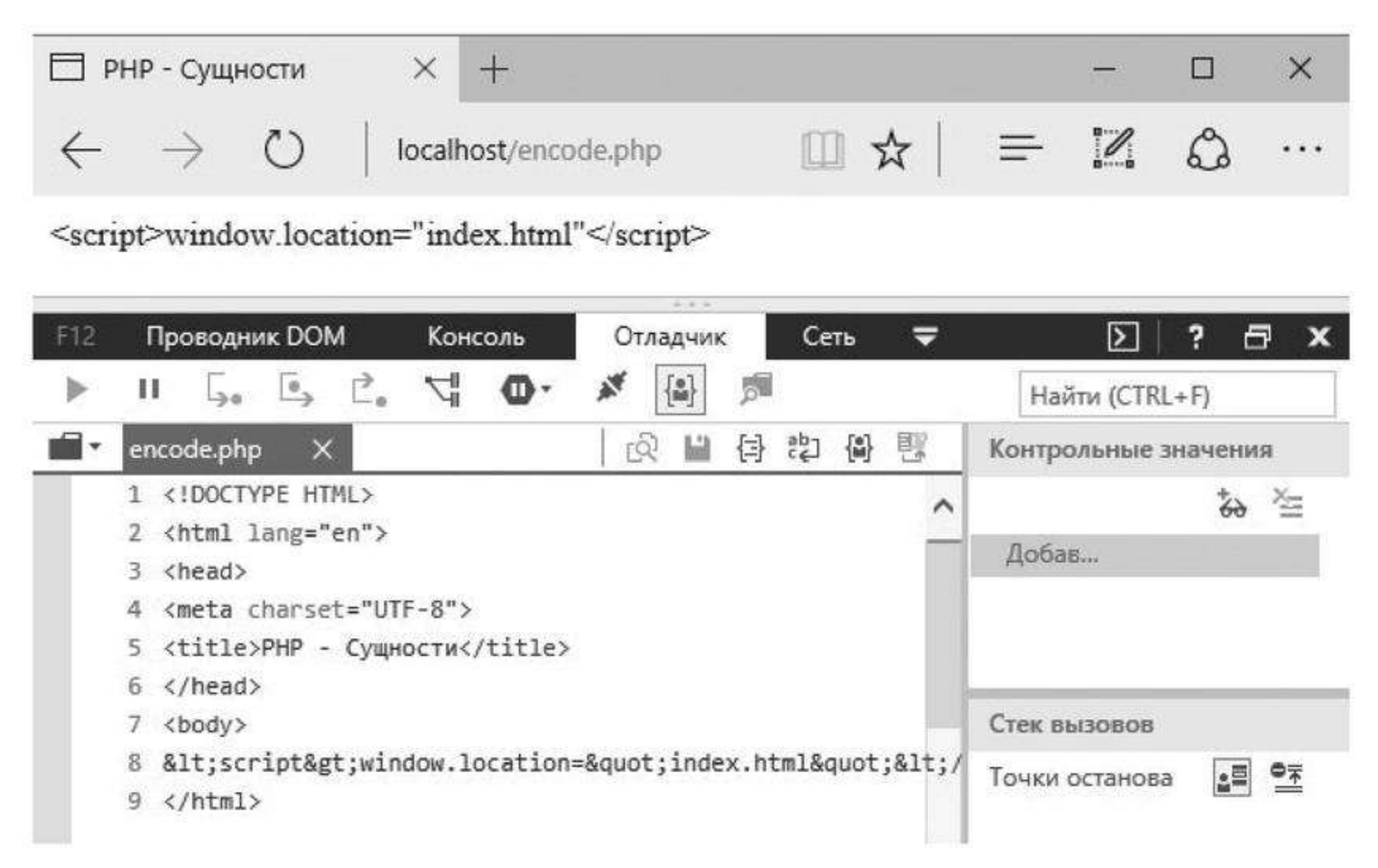
```
$html = htmlspecialchars($html);
```

4. Сохраните отредактированный документ и снова откройте страницу через протокол HTTP, чтобы увидеть фрагмент HTML-кода.

Совет

PHP-функция `html_entity_decode()` используется для обратного преобразования всех сущностей в их исходный формат символов.

5. Используйте инструмент просмотра исходного кода в веб-браузере, чтобы увидеть сущности, эквивалентные специальным символам, преобразованным средствами PHP.



Заключение

- Встроенные в PHP функции сравнения строк — `strcmp()`, `strncmp()`, `strcasecmp()` и `strncasecmp()` — как правило возвращают результат `1` (больше), `0` (равно) или `-1` (меньше).
- Сравнение строк основано на числовых ASCII-значениях каждого символа, сравниваемых в последовательности.
- ASCII-значение любого символа можно определить с помощью функции `ord()`.
- Длину строки можно определить с помощью функции `strlen()`.
- Численный индекс позиции подстроки в строке используется для поиска с помощью функций `strpos()` и `strrpos()`.
- Поисковые функции `strstr()`, `stristr()`, `strchr()` и `strrchr()` возвращают оставшуюся часть искомой строки.

- Функции `substr()`, `substr_replace()` и `substr_count()` позволяют извлекать и заменять подстроки, а также подсчитывать количество вхождений указанной подстроки.
- Строки можно обратить с помощью функции `strrev()` или повторить с помощью функции `strrpt()`.
- Заполняющие символы могут быть добавлены в строку с помощью функции `strpad()` и удалены с помощью функций `trim()`, `ltrim()` и `rtrim()`.
- Строка может быть разделена на отдельные фрагменты с помощью разделителей, переданных функции `strtok()`.
- Функция `date()` требует указания строкового аргумента, который определяет параметры форматирования возвращаемой строки даты.
- По умолчанию дата в языке PHP указывается в часовом поясе UTC (всемирное координированное время).
- Функция `date_default_timezone_set()` используется для указания другого часового пояса вместо стандартного UTC.
- Функция `strtotime()` возвращает метку времени для любой даты или времени согласно полученным аргументам.
- Функция `htmlentities()` преобразует все строковые символы, для которых имеется эквивалентная им HTML-сущность.
- Функция `htmlspecialchars()` преобразует только значимые специальные символы HTML для защиты распознавания их браузером.

7

Классы и объекты

Эта глава рассказывает, как создавать и использовать виртуальные объекты для объектно ориентированного программирования (ООП) на языке PHP.

- **Инкапсуляция данных**
- **Создание объектов**
- **Инициализация членов класса**
- **Конструкторы и деструкторы**
- **Наследование свойств**
- **Полиморфизм**
- **Заключение**

Инкапсуляция данных

Класс представляет собой структуру данных, которая может содержать одновременно и переменные, и функции. Все это «члены» класса. Переменные называются «свойствами», а функции — «методами» класса.

Доступ к членам класса извне контролируется «спецификаторами доступа»* в объявлении класса. Как правило при этом запрещается доступ к свойствам класса, но разрешается доступ к методам, которые могут хранить и извлекать данные из свойств. Этот метод «сокрытия данных» гарантирует, что сохраненные данные безопасно инкапсулируются в переменные члены класса, и это первый принцип объектно ориентированного программирования (ООП).

Объявление класса начинается с ключевого слова `class`, за которым следует пробел, а потом имя класса. Оноа должно начинаться с прописной буквы и подчиняется обычным правилам именования в языке PHP. Далее следуют объявление свойств класса, а затем объявление методов класса. Код похож на ранее рассмотренное объявление переменных и функций, но каждое объявление должно начинаться со спецификатора доступа. Синтаксис объявления класса выглядит следующим образом:

```
class ИмяКласса
{
 спецификатор-доступа $перем1, $перем2, $перем3;

 спецификатор-доступа имяФункции1 ($арг1, $арг2, $арг3)
 {
  выполняемые-инструкции;
 }
}
```

Совет

Принято начинать имена классов с прописной буквы, а имена объектов — со строчной.

* Также называемыми *модификаторами*. — *Прим. перев.*

В качестве спецификатора доступа может быть указано одно из следующих ключевых слов: `public`, `private` или `protected`. Они обозначают права доступа для членов данного класса:

- Спецификатор `public` (общедоступный) определяет, что члены доступны из любой позиции видимости класса.
- Спецификатор `private` (закрытый) определяет, что члены доступны только другим членам того класса, в котором они определены.
- Спецификатор `protected` (защищенный) определяет, что члены доступны только другим членам того же класса и членов классов, производных от этого класса.

Совет

Производные классы обсуждаются далее в этой главе — см. раздел «Наследование свойств».

Любой реальный объект может быть определен с помощью его атрибутов и действий. Например, собаке присущи такие атрибуты, как возраст, вес и цвет, а также действия, которые она может выполнять, к примеру, лай. Механизм классов в языке PHP предоставляет возможность создания в сценарии виртуального объекта собаки, чьи свойства класса могут представлять собственные атрибуты, а методы класса — собственные действия.

```
class Dog
{
 private $age;
 private $weight;
 private $color;

 public function bark() {echo 'ГАВ!';}

 // Включая методы для хранения данных в свойствах.

 // Включая методы для извлечения данных из свойств.
}
```

На заметку

Поскольку класс в PHP-коде не может полностью эмулировать реальный объект, задача программиста состоит в том, чтобы инкапсулировать все необходимые атрибуты и действия как свойства и методы класса.

Важно признать, что объявление класса только определяет структуру данных — чтобы создать объект, который должен объявить «экземпляр» этой структуры данных. Для этого следует присвоить переменной копию (экземпляр) класса, используя ключевое слово `new`, например:

```
$fido = new Dog();     // Создание экземпляра с именем
                       // "Фидо" определенной в программе
                       // структуры данных "Dog".
```

Экземпляр класса (объект) представляет собой рабочую копию структуры данных исходного класса с сохранением всего функционала. Поскольку метод в приведенном выше примере был объявлен общедоступным (на это указывает слово `public`), он доступен в сценарии за пределами объявления класса. Это означает, что этот метод может быть непосредственно вызван при обращении к объекту с помощью специального оператора `->`, как показано ниже:

```
$fido->bark();          // Выводит ГАВ!
```

Внимание

Обратите внимание, что необходимо добавить круглые скобки после имени класса в инструкции создания экземпляра.

Оператор `->` может быть использован для ссылки на любой видимый элемент экземпляра класса, но свойства переменной, как показано в примере выше, были объявлены закрытыми (`private`), поэтому инкапсулируются в объект и не являются непосредственно доступными. Принцип инкапсуляции в языке PHP определяет группировку вместе данных и функциональности в членах класса — это характеристики возраста, веса, цвета и действие «лай» в данном классе `Dog`.

Создание объектов

Для присвоения и извлечения данных закрытых (`private`) членов класса к нему должны быть добавлены специальные общедоступные (`public`) методы доступа. Это так называемые «сеттеры», которые используются для присвоения данных, и «геттеры» — для извлечения данных. Методы доступа часто именуются в соответствии с именем переменной, которой адресуются, с указанием первой буквы имени переменной как прописной и приставкой `set` или `get` соответственно. Например, методы доступа, адресуемые переменной `age`, могут быть названы `setAge()` и `getAge()`.

Сеттеры и геттеры могут ссылаться на свойства объекта класса с помощью специальной «псевдопеременной» `$this`, за которой следует оператор `->` и имя свойства. К примеру, код `$this->age` используется для ссылки на свойство `age`:

1. Начните PHP-сценарий с инструкции, объявляющей класс `Dog`.

object.php

```
<?php
class Dog
{
 // Здесь указывается код членов класса (шаги 2-5).
}
```

2. Между скобками объявления класса `Dog` объявите три закрытых переменных.

```
private $age;
private $weight;
private $color;
```

3. После закрытых переменных добавьте общедоступный метод для вывода строки при вызове.

```
public function bark() {echo 'ГАВ! ГАВ! <br>';}
```

Совет

Обратите внимание, что в объявлении класса все методы объявлены общедоступными, а все переменные — закрытыми. Данный принцип «открытый интерфейс, закрытые данные» является ключевым при создании классов.

4. Добавьте общедоступные сеттеры для присвоения индивидуальных значений аргументов каждой из закрытых переменных.

```
public function setAge (int $yrs)
{$this->age = $yrs;}

public function setWeight (int $lbs)
{$this->weight = $lbs;}

public function setColor (string $fur)
{$this->color = $fur;}
```

Фидо

5. Добавьте общедоступные геттеры для получения индивидуальных значений аргументов каждой из закрытых переменных.

```
public function getAge() {return $this->age;}

public function getWeight() {return $this->weight;}

public function getColor() {return $this->color;}
```

6. После объявления класса `Dog`, объявите экземпляр класса `Dog` с именем `fido`.

```
$fido = new Dog();
```

7. Добавьте инструкции вызова каждого из сеттеров для присвоения данных.

```
$fido->setAge(3);
$fido->setWeight(15);
$fido->setColor('коричневого');
```

8. Добавьте инструкции вызова каждого из геттеров для получения присвоенных значений.

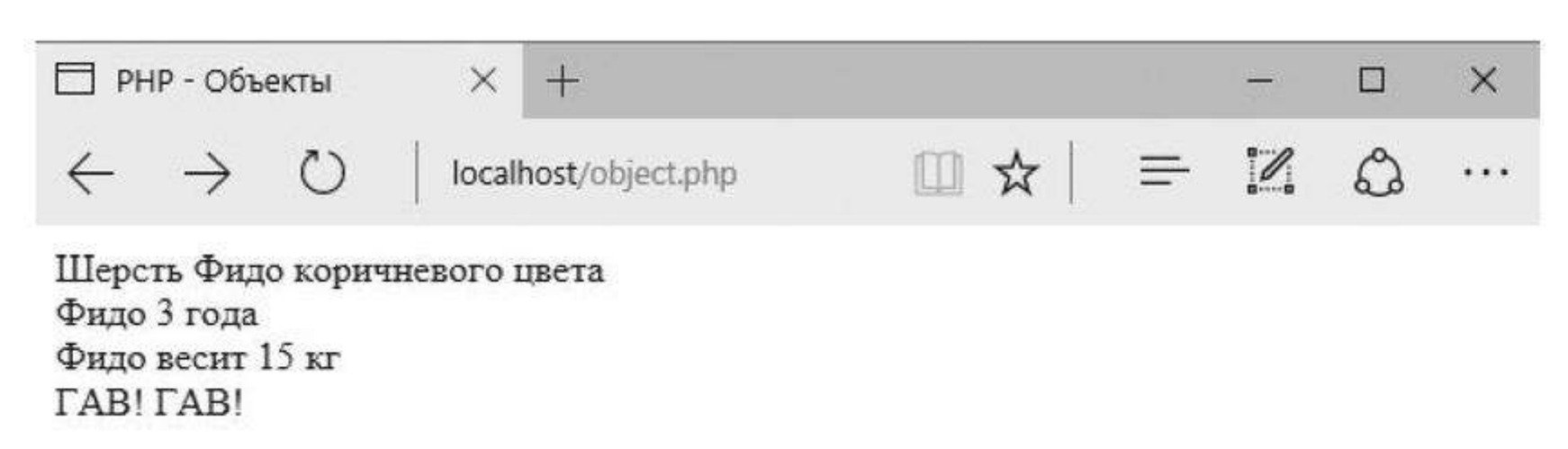
```
echo 'Шерсть Фидо '.$fido->getColor().' цвета<br>';
echo 'Фидо '.$fido->getAge().' года<br>';
echo 'Фидо весит '.$fido->getWeight().' кг<br>';
```

Нововведением в языке PHP 7 является возможность создания «анонимных классов» с простой функциональностью и без имени. К примеру, `$util = new class {public function log($s) {echo $s;}};` может быть вызван с помощью кода `$util->log('Data');`.

9 Теперь добавьте обычный метод вывода.

```
$fido->bark();
?>
```

10 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *object.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат вывода свойств класса и вызова метода.



Совет

Описанный PHP-сценарий ***object.php*** будет использоваться с изменениями на протяжении следующих нескольких страниц, по мере того, как в него будут вноситься новые функции.

Инициализация членов класса

Программа может легко создать несколько объектов, просто объявив несколько экземпляров класса. Каждый объект при этом сможет иметь уникальные атрибуты путем присвоения персональных значений с помощью сеттеров.

Часто удобно комбинировать сеттеры в единый метод, который принимает аргументы для каждой закрытой переменной. Это означает, что все значения могут быть назначены с помощью вызова одного метода, а этот метод будет содержать несколько инструкций.

Опционально каждый параметр может содержать допустимый тип данных, передаваемый каждым аргументом, а также, если необходимо, значения по умолчанию каждого параметра.

1. Создайте копию предыдущего примера из файла *object.php* и переименуйте копию в файл *initializer.php*.

initializer.php

2. В объявлении класса `Dog` замените код трех сеттеров одним комбинированным сеттером, определив в нем типы данных и значения по умолчанию аргументов.

```
public function setValues
(int $yrs = 2, int $lbs = 8, string $fur = 'черного ')
{
 // Сюда вставляются инструкции.
}
```

3. В блоке определения метода добавьте три инструкции, присваивающие переменным класса значения от переданных аргументов.

```
$this->age = $yrs;
$this->weight = $lbs;
$this->color = $fur;
```

4. После измененного кода определения класса замените три вызова сеттеров одним вызовом комбинированного сеттера с передачей трех аргументов.

```
$fido->setValues(3, 15, 'коричневого');
```

Совет

Попробуйте передать методу инициализации данные неправильного типа и посмотрите, как возникнет ошибка.

5. Объявите второй экземпляр класса Dog с именем pooch.

```
$pooch = new Dog();
```

6. Добавьте второй вызов в комбинированный сеттер, передав три аргумента для нового объекта.

```
$pooch->setValues(4, 18, 'серого ');
```

7. Добавьте инструкции вызова каждого геттера для извлечения присвоенных значений.

```
echo '<hr>Дворняжка: '.$pooch->getAge().' года ';
echo $pooch->getWeight().' кг '.$pooch->getColor();
```

8. Добавьте второй вызов к обычному методу вывода.

```
$pooch->bark();
```

9. Теперь объявите третий экземпляр класса `Dog`, присвоив ему имя `rover`.

```
$rover = new Dog();
```

10. Добавьте второй вызов в комбинированный сеттер, в этот раз не передавая никаких аргументов для нового объекта.

```
$rover->setValues();
```

11. Добавьте инструкции вызова каждого геттера для получения присвоенных значений, которые в коде указаны как используемые по умолчанию.

```
echo '<hr>Рык: '.$rover->getAge().' года ';
echo $rover->getWeight().' кг '.$rover->getColor();
```

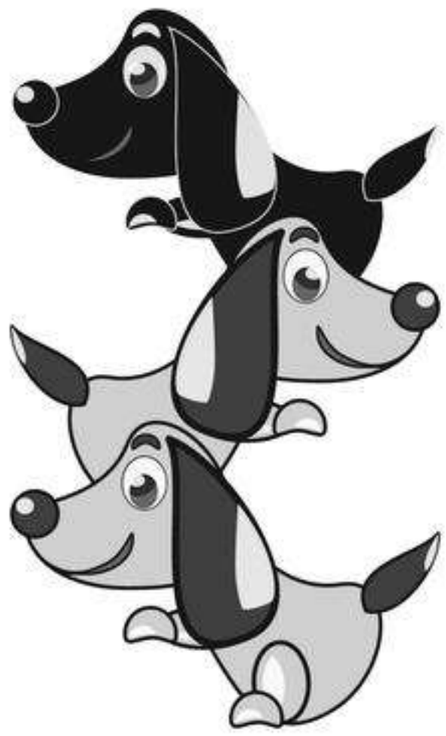
На заметку

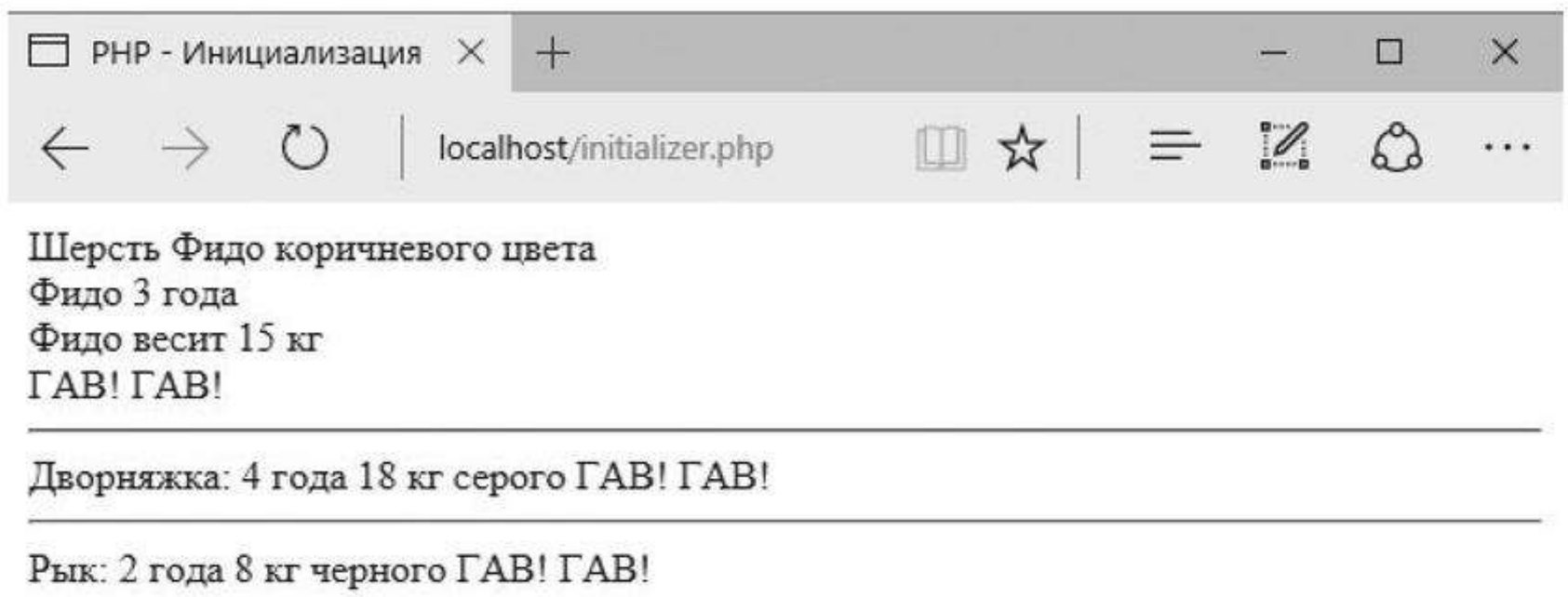
Вы должны по-прежнему вызвать метод инициализации, чтобы присвоить значения по умолчанию переменным класса, если не передано никаких аргументов.

12. Добавьте третий вызов к обычному методу вывода.

```
$rover->bark();
```

13. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *initializer.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат вывода значений с помощью комбинированного метода инициализации.





Конструкторы и деструкторы

Члены переменной класса могут быть инициализированы с помощью специального метода под названием «конструктор», который вызывается всякий раз при создании экземпляра класса. Конструктор всегда носит имя `__construct()` и может принимать аргументы для установки начальных значений переменных класса.

У конструктора есть коллега, «деструктор» — метод, который вызывается всякий раз при уничтожении экземпляра класса. Как вы могли догадаться, деструктор всегда имеет имя `__destruct()`.

Конструкторы и деструкторы не возвращают значения и вызываются автоматически — их нельзя вызвать явно и, следовательно, их объявления не нуждаются в спецификаторах доступа.

Значения для инициализации переменных класса передаются конструктору в инструкции, создающей объект, в скобках после его имени.

1. Создайте копию предыдущего примера из файла *initializer.php* и переименуйте копию в файл *constructor.php*.

constructor.php

2. В объявлении класса `Dog` замените код метода `setValues` на конструктор.

```
function __construct
(int $yrs = 2, int $lbs = 8, string $fur = 'черного ')
{
 $this->age = $yrs;
 $this->weight = $lbs;
 $this->color = $fur;
}
```

3. Теперь добавьте простой деструктор.

```
function __destruct() {echo 'Объект уничтожен!';}
```

4. После объявления класса `Dog` измените код инструкции, создающей объект `fido`: передайте значения конструктору в порядке инициализации свойств.

```
$fido = new Dog(3, 15, 'коричневого');
```

Совет

Определение метода класса также известно как «реализация».

5. Аналогичным образом измените инструкцию, создающую объект `pooch`, чтобы значения передавались конструктору.

```
$pooch = new Dog(4, 18, 'серого ');
```

6. Теперь измените инструкцию, создающую объект `rover`, чтобы значения по умолчанию также передавались конструктору.

```
$rover = new Dog();
```

7. Удалите инструкции вызова метода `setValues` всех трех объектов — теперь вместо этого метода используется конструктор.

На заметку

Несмотря на то, что начальные значения переменных определяются конструктором, вы также можете добавить сеттеры, чтобы впоследствии изменить эти значения, которые, в свою очередь, могут быть получены с помощью геттеров.

Язык PHP предоставляет несколько функций опроса классов, такие как `class_exists(`*класс*`)`, `method_exists(`*класс, метод*`)` и `property_exists(`*класс, свойство*`)`, которые возвращают значение `TRUE` в случае успешного завершения. Вы также можете вывести список членов класса с помощью функций `get_class_vars(`*класс*`)` и `get_class_methods(`*класс*`)` для просмотра только «видимых» членов.

8. Добавьте инструкции, выводящие общее количество доступных переменных в классе `Dog`.

```
$items = get _ class _ vars('Dog');
echo '<br>Переменные класса Dog: '.count($items);
```

9. Добавьте инструкции для вывода имен доступных методов класса `Dog`.

```
echo '<br>Методы класса Dog: ';
$items = get _ class _ methods('Dog');
foreach($items as $item) {echo "$item, ";}
```

10 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *constructor.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.

Совет

Совет. Число видимых переменных равно нулю. На время измените их спецификатор доступа на общедоступный (`public`), и вы увидите значение 3 — перед вами инкапсуляция в действии.

Наследование свойств

PHP позволяет создавать как полностью новый класс (что показано в предыдущих примерах), так и класс, «производный» от существующего класса. Важно отметить, что дочерний класс наследует члены родительского класса (базового), от которого он произведен, в дополнение к своим собственным членам.

Возможность наследования членов от родительского класса позволяет дочерним классам наследовать некоторые общие свойства, которые были определены в родительском классе. К примеру, родительский класс «Многоугольник» может определять свойства ширины и высоты, которые являются общими для всех многоугольников. Классы «Прямоугольник» и «Треугольник» могут быть произведены от класса «Многоугольник» и унаследовать свойства ширины и высоты, в дополнение к своим собственным членам, определяющих их уникальные характеристики.

Сила наследования чрезвычайно велика — это второй принцип объектно ориентированного программирования (ООП).

Объявление дочернего класса содержит имя дочернего класса, затем ключевое слово `extends` и имя родительского класса. Рассмотрим наследование в действии.

inheritance.php

1. Начните PHP-сценарий с объявления базового класса с именем `Polygon` — т.е. тот самый «Многоугольник» из примера выше. У этого класса есть две закрытые переменные, инициализированные с помощью конструктора, и два общедоступных геттера.

```
<?php
class Polygon
{
 private $width, $height;

 function _ _construct(int $w = 4, int $h = 5)
 {
  $this->width = $w;
  $this->height = $h;
 }

 public function getWidth() {return $this->width;}
 public function getHeight() {return $this->height;}
}
```

2. После кода класса `Polygon` объявите класс `Rectangle` («Прямоугольник» из примера выше), который является производным (дочерним) от класса `Polygon`, и добавьте уникальный метод.

```
class Rectangle extends Polygon
{
 public function area()
```

```
  {
   return ($this->getWidth() * $this->getHeight());
  }
}
```

Внимание

Не следует путать экземпляры классов и дочерние классы: экземпляр — это копия класса, в то время как дочерний класс — это новый класс, унаследовавший свойства родительского класса, от которого он произведен.

3. После кода класса `Rectangle` объявите класс `Triangle` («Треугольник» из примера выше), который является производным (дочерним) от класса `Polygon`, и добавьте уникальный метод.

```
class Triangle extends Polygon
{
 public function area()
 {
  return ($this->getWidth() * $this->getHeight() / 2);
 }
}
```

4. После кода класса `Triangle` добавьте две инструкции, создающие по экземпляру каждого производного класса, используя значения по умолчанию.

```
$rect = new Rectangle();
$trio = new Triangle();
```

5. Наконец, добавьте две инструкции для вывода значения, возвращаемого уникальным методом каждого производного (дочернего) класса.

```
echo 'Площадь прямоугольника: '.$rect->area().'<hr>';
echo 'Площадь треугольника: '.$trio->area();
?>
```

На заметку

На заметку. В языке PHP дочерний класс всегда зависит от одного родительского класса — множественное наследование не поддерживается.

6 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *inheritance.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.

Полиморфизм

Три краеугольных камня объектно ориентированного программирования (ООП) — это инкапсуляция, наследование и полиморфизм. На предыдущих страницах вы узнали, как данные могут быть инкапсулированы в PHP-класс и как дочерние классы наследуют свойства своего родительского класса, поэтому теперь мы можем исследовать финальный принцип ООП — полиморфизм.

Термин «полиморфизм» (от греч. πολύ- — много и μορφή — форма) означает способность назначить другой смысл или предназначение объекту в соответствии с контекстом. Например, в реальном мире каждый знает, как использовать кнопку (объект) — вы просто нажимаете ее. Результат нажатия кнопки зависит от того, что с ней связано (контекст), но то, как используется эта кнопка, на результат не влияет.

В мире объектно ориентированного программирования полиморфизм позволяет классам иметь различную функциональность, разделяя общий интерфейс. Интерфейсам не нужно знать, какие классы используют, поскольку они, как и кнопки в реальном мире, используются одинаковым способом, но приводят к разным результатам, зависящим от контекста. Таким образом, полиморфизм позволяет программистам создавать взаимозаменяемые объекты, которые могут быть выбраны в соответствии с необходимостью, без проверки условий.

Интерфейсы в PHP подобны классам, но объявляются с помощью ключевого слова `interface` и могут содержать только определения методов, а не какие-либо исполняемые инструкции. Синтаксис показан ниже.

```
interface ИмяИнтерфейса
{
 // Сюда помещаются только определения методов.
}
```

На заметку

Определение методов в объявлении интерфейса позволяет также указать их параметры.

При реализации интерфейса каждый класс должен реализовать все методы, которые определяет интерфейс, указав их инструкции выполнения. Класс присоединяется к интерфейсу добавлением ключевого слова `implements` и имени интерфейса в объявлении:

```
class ИмяКласса implements ИмяИнтерфейса
{
 // Сюда помещаются реализуемые методы.
}
```

Мы изменим пример из предыдущего раздела главы, чтобы охватить полиморфизм, и добавим в него интерфейс и одну функцию.

1. Создайте копию предыдущего примера из файла *inheritance.php* и переименуйте копию в файл *polymorphism.php*.

polymorphism.php

2. Объявите интерфейс с именем `Shape`, который содержит определение методов, уже реализованных в классах `Rectangle` и `Triangle`.

```
interface Shape {public function area();}
```

3. Измените первую строку объявлений классов `Rectangle` и `Triangle` для связи с интерфейсом.

```
class Rectangle extends Polygon implements Shape
class Triangle extends Polygon implements Shape
```

4. После блоков кода классов добавьте функцию, которая принимает аргумент типа `Shape` и вызывает определенный в нем метод.

```
function getArea(Shape $shape)
{return $shape->area();}
```

5. Измените две инструкции, создающие экземпляры каждого дочернего класса, для передачи значений аргументов.

```
$rect = new Rectangle(8, 10);
$trio = new Triangle(8, 10);
```

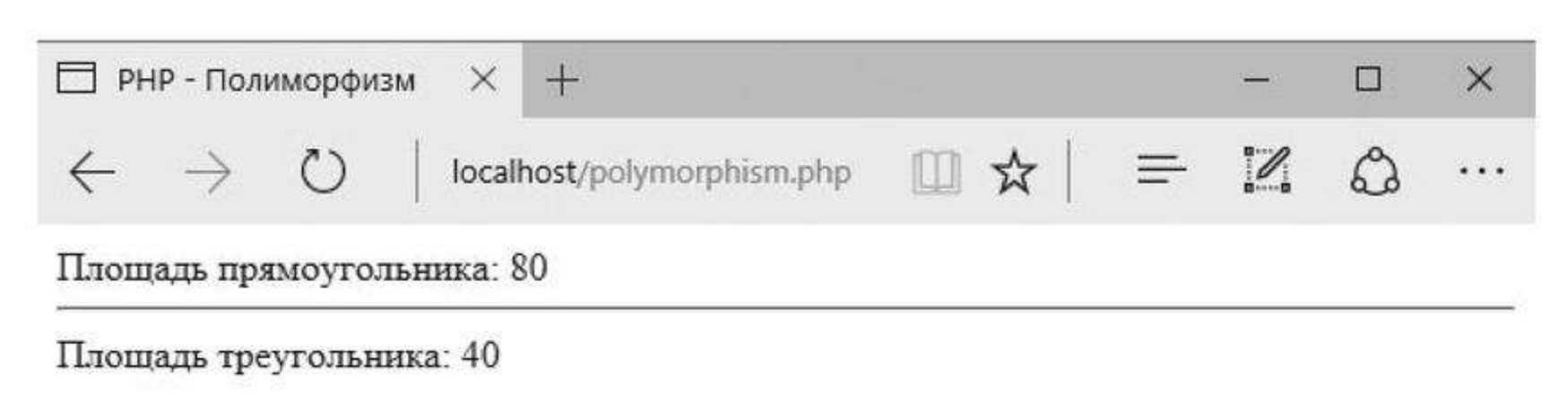
На заметку

Что тут происходит? Каждый вызов новой функции `getArea()` передает экземпляр класса в качестве аргумента. Функция `getArea()` вызывает метод `area()`, определенный в интерфейсе. Вывод возвращается после реализации метода `area()` связанного экземпляра.

6. И, наконец, измените две инструкции для вывода значения, возвращаемого уникальными методами каждого дочернего класса, на вызов каждого метода через интерфейс.

```
echo 'Площадь прямоугольника: '.getArea($rect).'<hr>';
echo 'Площадь треугольника: '.getArea($trio);
```

7. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *polymorphism.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат вывода значений через полиморфный интерфейс.



Заключение

- Класс в PHP представляет собой структуру данных, которая может содержать переменные и функции как единое целое.
- Классы объявляются с помощью ключевого слова `class` и могут содержать свойства переменной-члена и методы функции-члена.
- Доступ к членам класса извне класса определяется спецификаторами с ключевыми словами `public`, `private` и `protected`.

- Свойства переменных инкапсулируются объявлением их закрытыми (`private`), и их значения могут быть доступны только общедоступным (`public`) методам.
- Чтобы создать объект, вы должны объявить экземпляр структуры данных `class`, применив ключевое слово `new`.
- Специальный оператор `->` используется для ссылки на любой видимый член класса.
- Методы сеттер и геттер могут ссылаться на члены одного и того же класса, используя псевдопеременную `$this`.
- Единый сеттер может присваивать несколько значений аргументов или значений по умолчанию для инициализации нескольких свойств экземпляра класса.
- Конструктор (`__construct()`) вызывается при создании класса и может использоваться для инициализации свойств нового экземпляра.
- Деструктор (`__destruct()`) вызывается при уничтожении класса и может быть использован для выполнения финальных инструкций.
- Язык PHP содержит функции опроса классов, перечисления методов (`get_class_methods()`) и перечисления переменных (`get_class_vars()`).
- Три принципа объектно ориентированного программирования — это инкапсуляция, наследование и полиморфизм.
- Объявление производного (дочернего) класса включает в себя ключевое слово `extends` и имя родительского класса, члены которого будут унаследованы.
- Интерфейсы объявляются с помощью ключевого слова `interface` и в объявлении могут содержать только определения методов.
- Класс связывается с интерфейсом при объявлении с помощью ключевого слова `implements` и имени интерфейса.

8

Работа с файлами

Эта глава расскажет о чтении и записи файлов с помощью PHP и о том, как обрабатывать ошибки в сценариях.

- Чтение файлов
- Чтение строк
- Чтение символов
- Запись файла
- Добавление текста
- Обработка ошибок
- Обнаружение исключений
- Заключение

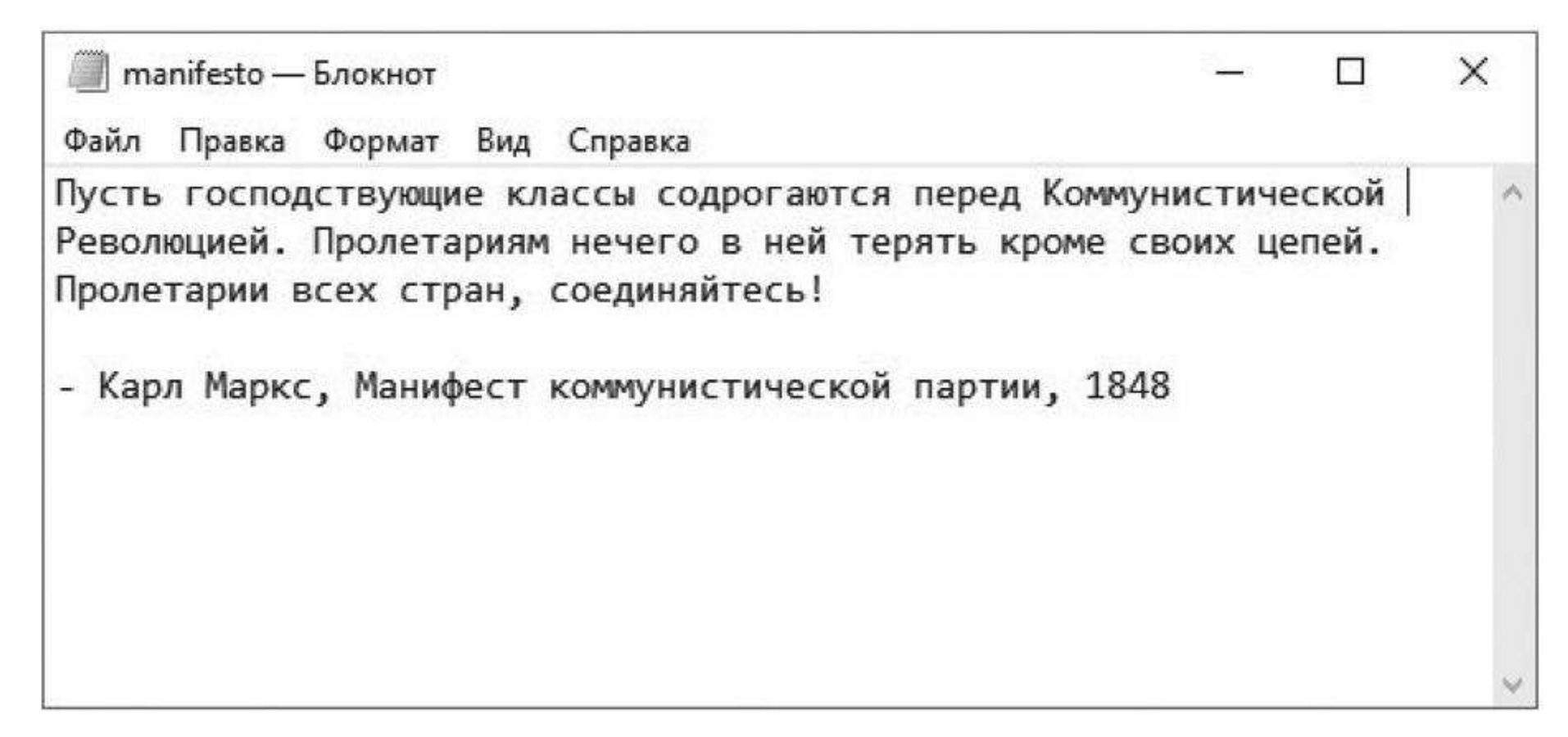
Совет

В качестве аргумента функции `die()` можно указать пояснительное сообщение.

Рекомендуется всегда добавлять в код альтернативное поведение с помощью функции `die()` в случае, если попытка создать файловый поток потерпит неудачу.

Если файловый поток создан успешно, файл может быть прочитан с указанием файлового потока и бинарного (в байтах) размера в качестве аргументов функции `fread()`. Если файл должен быть прочитан целиком, удобно использовать функцию `filesize()`, позволяющую указать бинарный размер файла, предназначенного для чтения.

После завершения любой файловой операции файловый поток должен быть прерван, будучи указанным в качестве аргумента функции `fclose()`.



1. Начните PHP-сценарий с инструкций, открывающих простой текстовый файл для чтения или завершающих сценарий в случае ошибки.

read.php

```
<?php
$filestream = fopen('manifesto.txt', 'r')
or die('Невозможно открыть файл!');
```

2. Затем добавьте инструкцию, позволяющую прочитать текстовый файл в случае его успешного открытия и выводящую все его содержимое.

```
echo fread($filestream, filesize('manifesto.txt'));
```

Чтение файлов

Возможность читать и записывать файлы, расположенные на веб-сервере, полезна для журналирования и обеспечения веб-страниц контентом. Для обработки файлов сначала должен быть создан объект «файлового потока» с помощью PHP-функции `fopen()`. Ей требуется передать два аргумента, определяющие имя или путь к файлу и режим работы с файлом, который определяет характер выполняемой операции и помещает файловый указатель в соответствующей позиции в файле. Основные файловые режимы перечислены в таблице ниже.

Внимание

Режим работы с файлом следует указывать в кавычках, если он передается в качестве аргумента функции `fopen()`.

Режим	Действие
r	Открывает существующий файл для чтения. Файловый указатель помещается в начало файла.
w	Удаляет все содержимое существующего файла или создает новый файл, если тот не существует. Файловый указатель помещается в начало файла.
a	Открывает существующий файл и сохраняет все содержимое файла. Файловый указатель помещается в конец файла.
r+	Открывает файл для чтения или записи. Файловый указатель помещается в начало файла.
w+	Открывает файла для записи или чтения. Удаляет все содержимое существующего файла или создает новый файл, если тот не существует. Файловый указатель помещается в начало файла.
a+	Открывает файл для чтения или записи и сохраняет все содержимое файла. Файловый указатель помещается в конец файла.

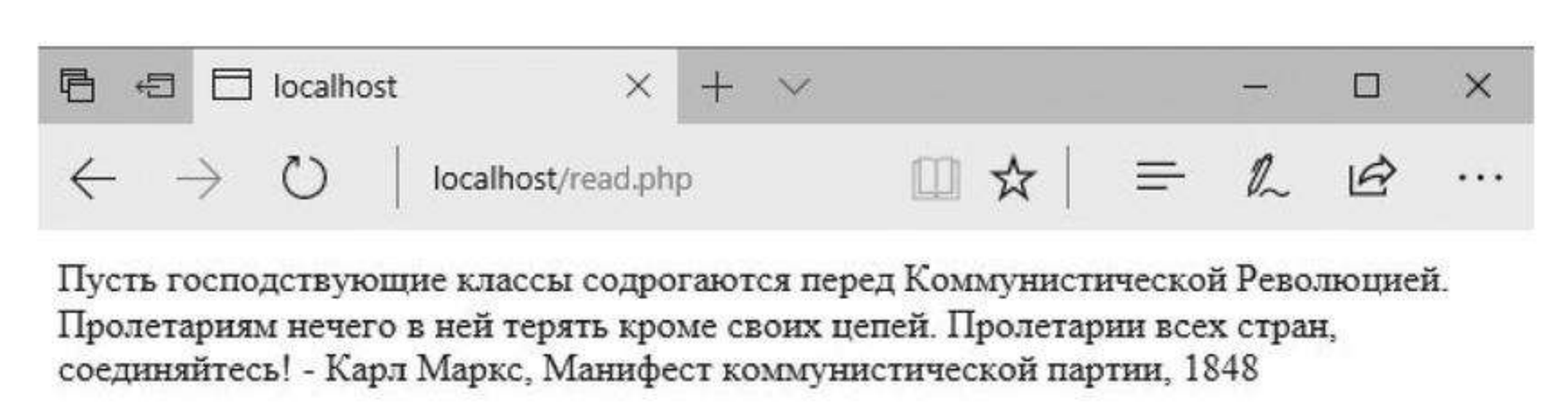
3 Теперь не забудьте закрыть файловый поток.

```
fclose($filestream);
```

На заметку

В отличие от блока PHP-кода внутри HTML-документа, отдельный файл с PHP-сценарием не требует указания закрывающего тега ?>.

4 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *read.php* вместе с простым текстовым файлом *manifesto.txt*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть содержимое текстового файла, отображенное в окне браузера.



На заметку

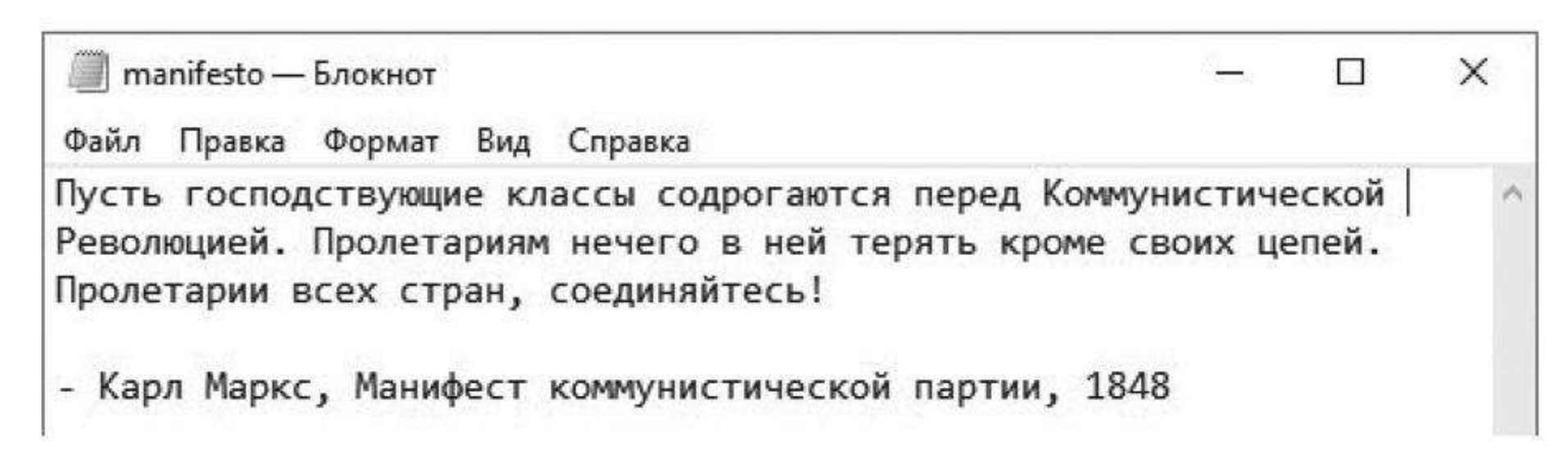
Обратите внимание, что в данном примере в выводе не соблюдаются концы строк. Мы рассмотрим эту возможность в следующем примере.

Чтение строк

После создания файлового потока его содержимое можно считывать построчно, указав файловый поток в качестве аргумента PHP-функции `fgets()`. Для чтения каждой строки вы можете вызвать эту функцию несколько раз в цикле. Функция `feof()` позволяет определить, когда достигнут конец файла, и завершает цикл.

На заметку

Функция `feof()` возвращает значение `TRUE`, когда достигнут конец файла.



1. Начните PHP-сценарий с инструкций, открывающих простой текстовый файл для чтения или завершающих сценарий в случае ошибки.

readlines.php

```
<?php
$filestream = fopen('manifesto.txt', 'r')
or die('Невозможно открыть файл!');
```

2. Затем добавьте цикл для чтения содержимого текстового файла построчно в случае его успешного открытия и вывода всего его содержимого.

```
while(!feof($filestream))
{echo fgets($filestream).'<br>';}
```

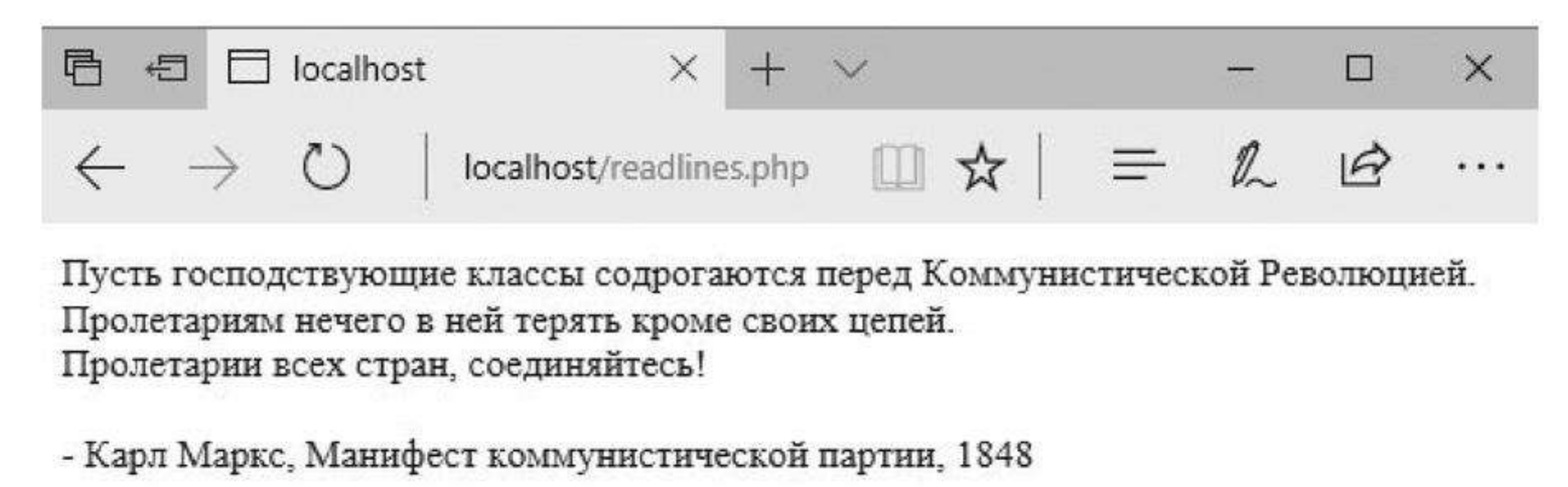
3. Теперь не забудьте закрыть файловый поток.

```
fclose($filestream);
```

Совет

Следующая после последней строки кода строка на самом деле не пуста — она содержит невидимый символ перехода на новую строку.

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *readlines.php* вместе с простым текстовым файлом *manifesto.txt*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть содержимое текстового файла, отображенное в окне браузера построчно.



Чтение символов

После создания файлового потока его содержимое можно считывать посимвольно, указав файловый поток в качестве аргумента PHP-функции `fgetc()`. Для чтения каждого символа вы можете вызывать эту функцию несколько раз в цикле. Функция `fgetc()` возвращает значение `FALSE`, когда достигнут конец файла, и завершает цикл.

Внимание

Программа Блокнот (Notepad) в операционной системе Windows автоматически добавляет скрытый маркер последовательности байтов (Byte Order Mark, BOM) в файл, в то время как другие редакторы (например, Notepad++, как показано выше) позволяют его не использовать. Notepad++ — это бесплатный текстовый редактор для программистов, доступный на сайте **notepad-plus-plus.org**.

1. Начните PHP-сценарий с инструкций, открывающих простой текстовый файл для чтения или завершающих сценарий в случае ошибки.

readchars.php

```
<?php
$filestream = fopen('title.txt', 'r')
or die('Невозможно открыть файл!');
```

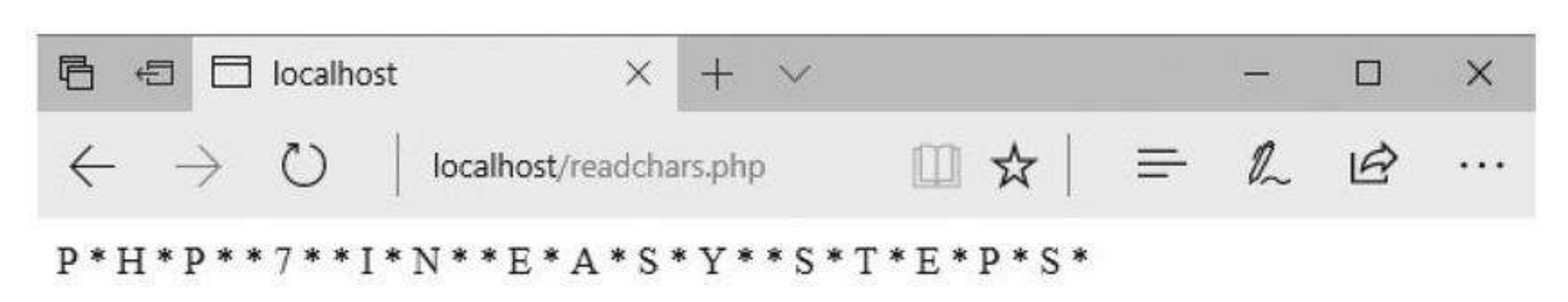
2. Затем добавьте цикл для посимвольного чтения содержимого текстового файла в случае его успешного открытия и вывода всего его содержимого.

```
while($char = fgetc($filestream))
{echo $char.' * ';}
```

3. Теперь не забудьте закрыть файловый поток.

```
fclose($filestream);
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *readchars.php* вместе с простым текстовым файлом *title.txt*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть содержимое текстового файла, отображенное в окне браузера, посимвольно.



На заметку

Функция `fgetc()` возвращает значение `FALSE`, если символы, получаемые из потока, больше не обнаруживаются.

Запись в файл

После создания файлового потока в режиме записи генерируется новый текстовый файл. В него может быть записан текст, если указать файловый поток и имя файла в PHP-функции `fwrite()`. В целях форматирования текста можно использовать неотображающиеся скрытые символы, позволяющие создавать переносы строк и отступы.

Символ	Описание
\n	Новая строка. Переход на следующую новую строку.
\r	Возврат каретки. Переход к началу строки.
\t	Отступ. Перемещение вдоль строки к следующей позиции табуляции (отступа).

Внимание

Строки, содержащие неотображающиеся символы, должны заключаться в двойные кавычки, чтобы PHP-движок смог их интерпретировать.

1. Начните PHP-сценарий с инструкций, присваивающих текстовые строки, включая неотображающиеся символы для форматирования.

write.php

```
<?php
$poem = "\r\n\tНет, не смотрел никто из нас ";
$poem .= "\r\n\tС такой тоской в глазах ";
$poem .= "\r\n\tНа лоскуток голубизны ";
$poem .= "\r\n\tВ тюремных небесах ";
```

2. Далее присвойте потоку имя файла, в который будет осуществляться запись.

```
$filename = 'poem.txt';
```

3. Напишите функцию создания простого текстового файла для записи и завершения сценария при возникновении ошибки.

```
$filestream = fopen($filename, 'w')
or die('Невозможно открыть файл!');
```

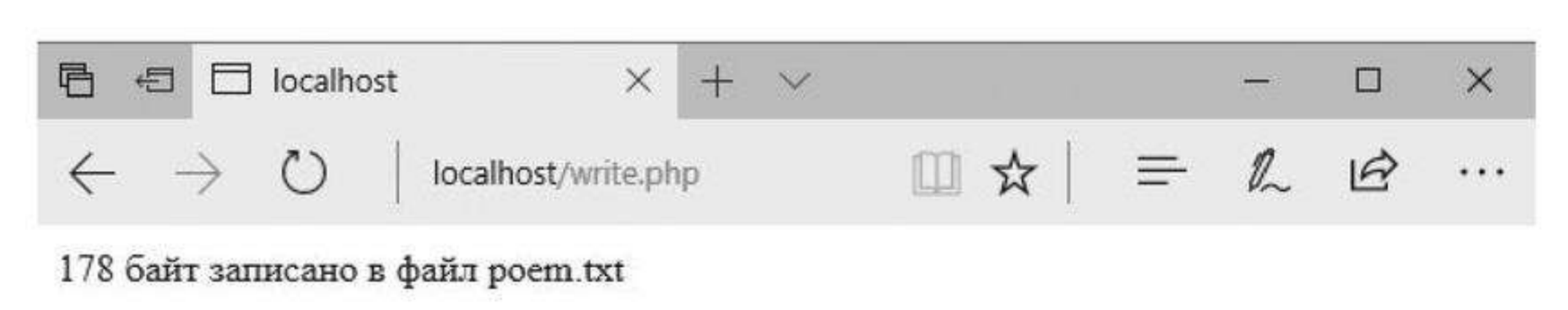
4. Напишите функцию для записи текста в файл и вывода подтверждения о том, сколько всего байт было записано.

```
$num = fwrite($filestream, $poem);
if($num)
{
 echo "$num байт записано в файл $filename";
}
```

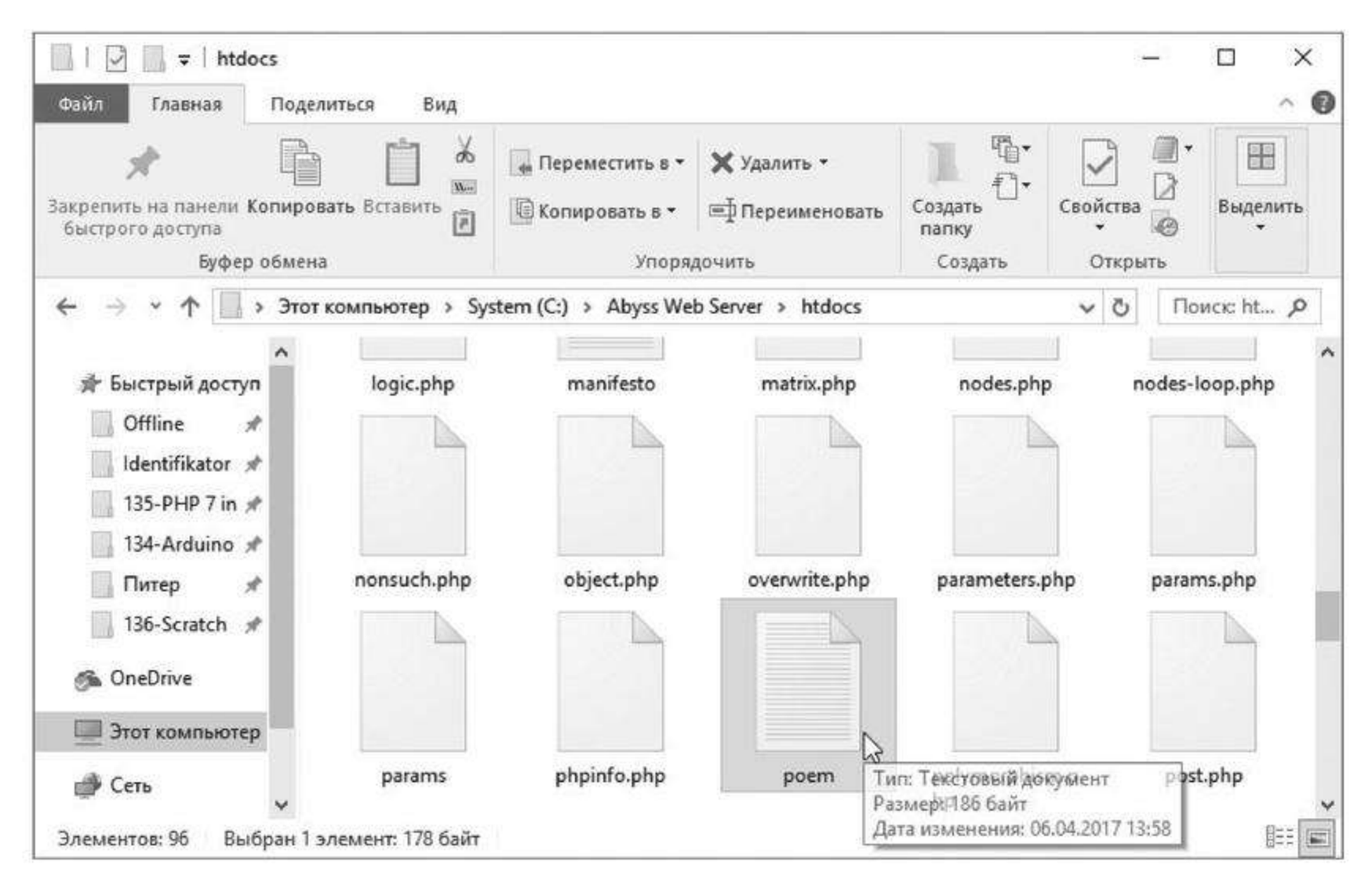
5. Теперь не забудьте закрыть файловый поток.

```
fclose($filestream);
```

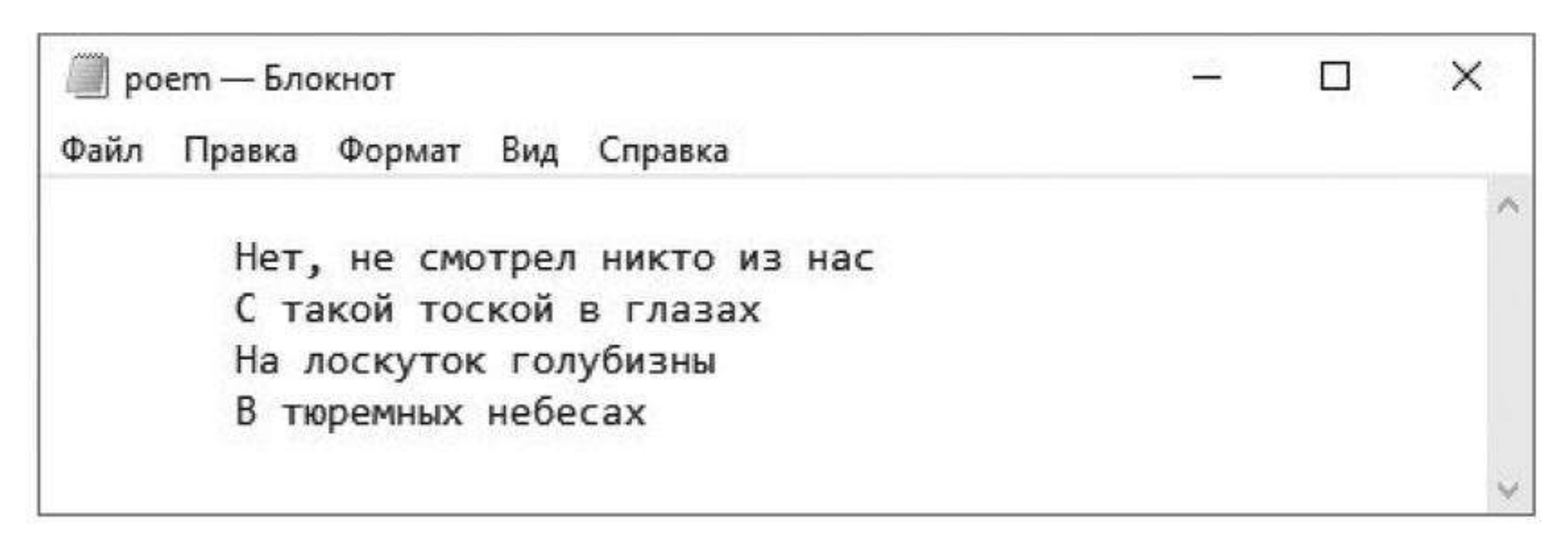
6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *write.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы выполнить запись файл и увидеть подтверждение этого.



7 Используя менеджер файлов, например Проводник (Windows Explorer), перейдите к каталогу */htdocs* вашего веб-сервера и откройте файл, созданный с помощью PHP-сценария *write.php*.



8 В файле вы увидите текст, записанный туда с помощью PHP-сценария.



Совет

Вы можете запустить этот сценарий несколько раз, и каждый раз содержимое файла будет стираться, а затем осуществляться новая запись.

Добавление текста

После создания файлового потока в режиме добавления текста вы можете открыть текстовый файл, чтобы добавить содержимое в его конец. Текст добавляется путем указания файлового потока и имени файла в PHP-функции `fwrite()`. Процесс идентичен записи файла, за исключением того, что файловый указатель позиционируется после уже имеющегося содержимого, прежде чем начинается запись текста в файл.

1. Начните PHP-сценарий с инструкций по добавлению текста, включая непечатаемые символы для форматирования.

append.php

```
<?php
$info = "\r\n\r\n\tБаллада Рэдингской тюрьмы";
$info .= "\r\n\t\t\tОскар Уайльд, 1898";
```

2. Далее присвойте потоку имя файла, в который будет осуществляться запись.

```
$filename = 'poem.txt';
```

3. Напишите код открытия текстового файла для добавления текста и завершения сценария при возникновении ошибки.

```
$filestream = fopen($filename, 'a')
or die('Невозможно открыть файл!');
```

На заметку

Помните, что неотображающиеся символы должны быть заключены в двойные кавычки для их корректной интерпретации.

4. Напишите функцию для записи текста в файл и вывода подтверждения о том, сколько всего байт было добавлено.

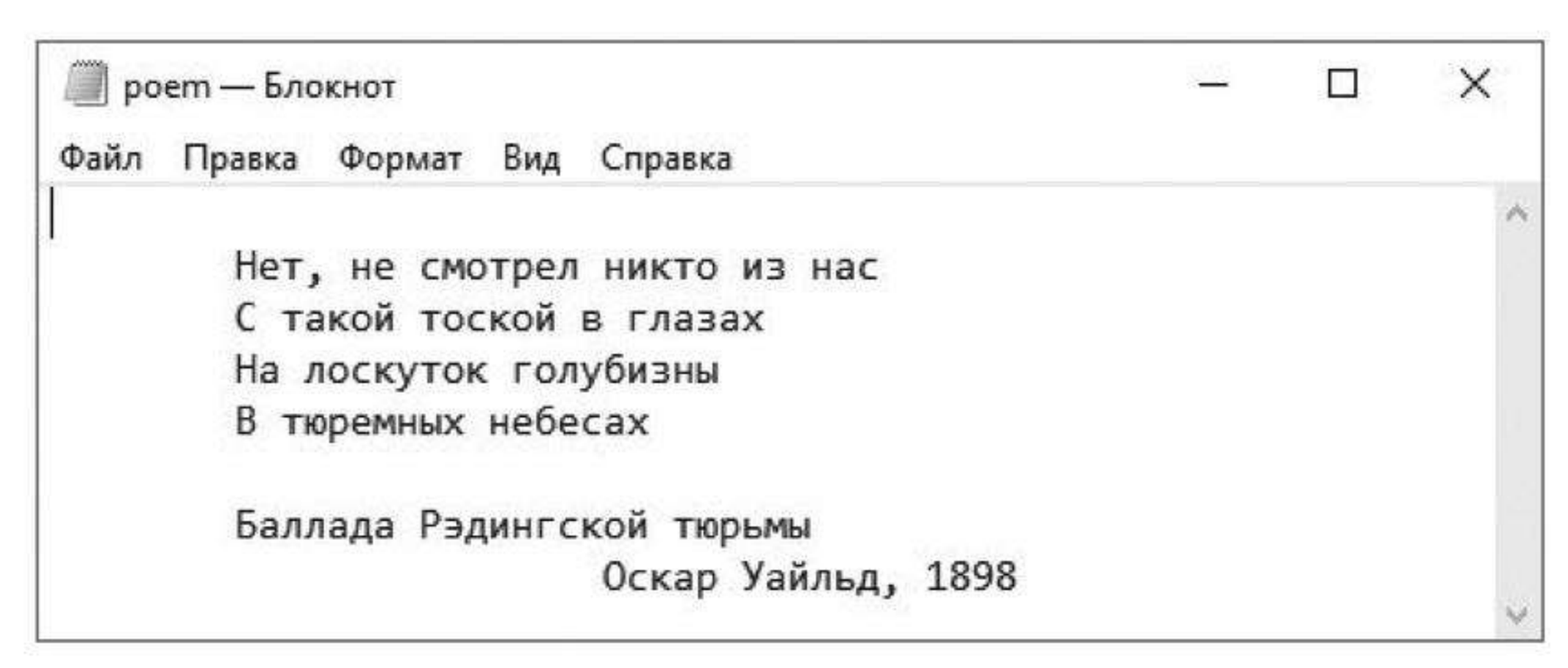
```
$num = fwrite($filestream, $info);
if($num)
{
 echo "$num байт добавлено в файл $filename";
}
```

5. Теперь не забудьте закрыть файловый поток.

```
fclose($filestream);
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *append.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть сообщение о количестве байт, добавленных в файл.

7. Перейдите к директории */htdocs* вашего веб-сервера и откройте файл, чтобы увидеть текст, добавленный в PHP-сценарий.



Перезапись текста

Важно учитывать, что, если вы откроете в режиме записи файл с содержимым, данное содержимое будет перезаписано.

1. Начните PHP-сценарий с инструкций, открывающих простой текстовый файл для записи или завершающих сценарий в случае ошибки.

overwrite.php

```
<?php
$filestream = fopen('poem.txt', 'w')
or die('Невозможно открыть файл!');
```

2. Затем добавьте инструкцию, позволяющую записать новое содержимое в текстовый файл в случае его успешного открытия.

```
fwrite($filestream, 'Файл перезаписан!');
```

3. Теперь не забудьте закрыть файловый поток.

```
fclose($filestream);
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *overwrite.php*, а затем откройте эту страницу в браузере через протокол HTTP. Также откройте текстовый файл, чтобы увидеть перезаписанное содержимое.

Внимание

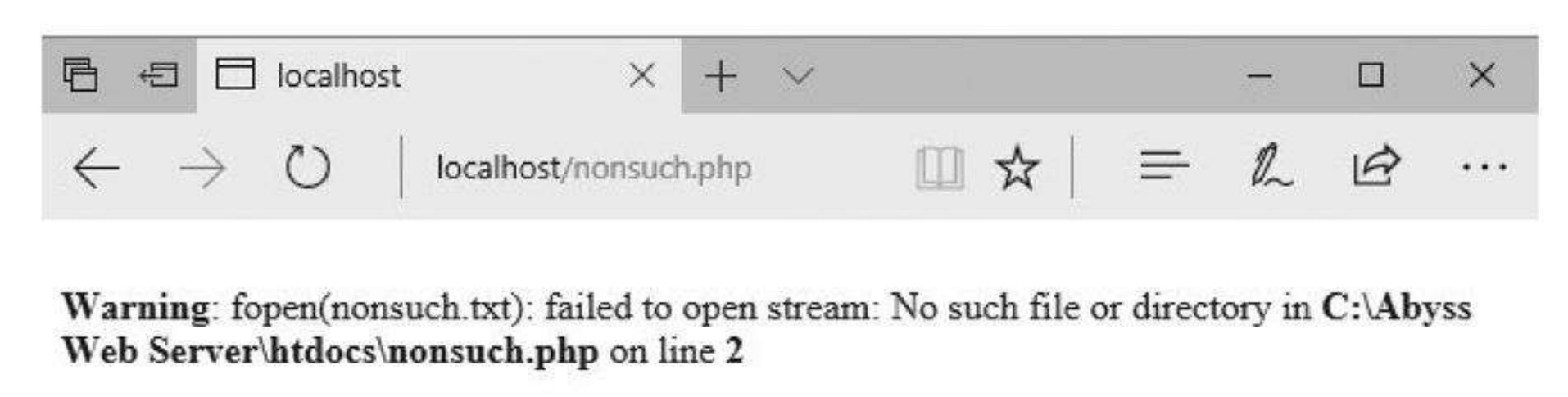
Будьте осторожны при работе с файлами в режиме записи, так как можно случайно перезаписать важные данные.

Обработка ошибок

Вы можете предвидеть возможные ошибки и завершать работу сценария в случае их возникновения с помощью функции `die()` или `exit()`. Без этого стандартный обработчик ошибок, встроенный в PHP-интерпретатор, хоть и полезен при возникновении ошибок, но сообщения обработчика просто ужасны. К примеру, если выполнить код `fopen('nonsuch.txt', 'r')`, обращающийся к несуществующему файлу, вы увидите сообщение, показанное ниже.

Совет

Функции `die()` и `exit()` выполняют идентичные операции в PHP, разница лишь в названиях. В одних случаях функция `die()`, возможно, будет более целесообразной, чтобы завершить работу сценария при обнаружении ошибки, а `exit()` — в других.



Не всегда желательно завершать сценарий при возникновении ошибки. Вы можете создать собственный обработчик ошибок, чтобы обеспечить более удобный для пользователя диалог с системой, чем это предусмотрено встроенный PHP-обработчиком. Пользовательский обработчик ошибок представляет собой обычную функцию со следующим синтаксисом:

```
function error_handler(уровень, сообщение, файл, строка,
контекст)
```

Только первые два параметра обязательны, остальные опциональны. Первый из них — это целочисленное значение, представляющее собой константу, указывающую на серьезность произошедшей ошибки.

Уровень	Константа	Описание
2	`E_WARNING`	Некритическая ошибка во время выполнения.
8	`E_NOTICE`	Некритическая возможная ошибка.
256	`E_USER_ERROR`	Сгенерированное пользователем сообщение о критической ошибке.
512	`E_USER_WARNING`	Сгенерированное пользователем сообщение о некритической ошибке.

Совет

Основные уровни ошибок перечислены в данной таблице, а информацию о дополнительных уровнях вы найдете на веб-сайте **php.net/manual/ru/errorfunc.constants.php**.

Функция пользовательского обработчика ошибок назначается, как вы догадались, для обработки ошибок указанием имени данной функции в качестве аргумента встроенной в PHP функции `set_error_handler()`.

Обработчик ошибок вызывается либо при возникновении ошибки, либо явно с помощью функции `trigger_error()` с указанием сообщения об ошибке в качестве ее аргумента.

1. Начните PHP-сценарий с определения функции пользовательского обработчика ошибок, передав ей уровни ошибки и соответствующие сообщения.

error.php

```
<?php
function error_handler($level, $message)
{
 switch($level)
 {
  case 2: $str = 'Внимание!'; break;
  case 8: $str = 'Предупреждение!'; break;
  default: $str = 'Ошибка!';
 }
 echo "<b>$str</b><br>$message<hr>";
}
```

2. Далее назначьте функцию обработки ошибок.

```
set_error_handler('error_handler');
```

3. Теперь для проверки попробуйте получить доступ к неинициализированной переменной и открыть несуществующий файл.

```
echo($var);
$file = fopen('nonsuch.txt', 'r');
```

4. Явно вызовите пользовательский обработчик ошибок.

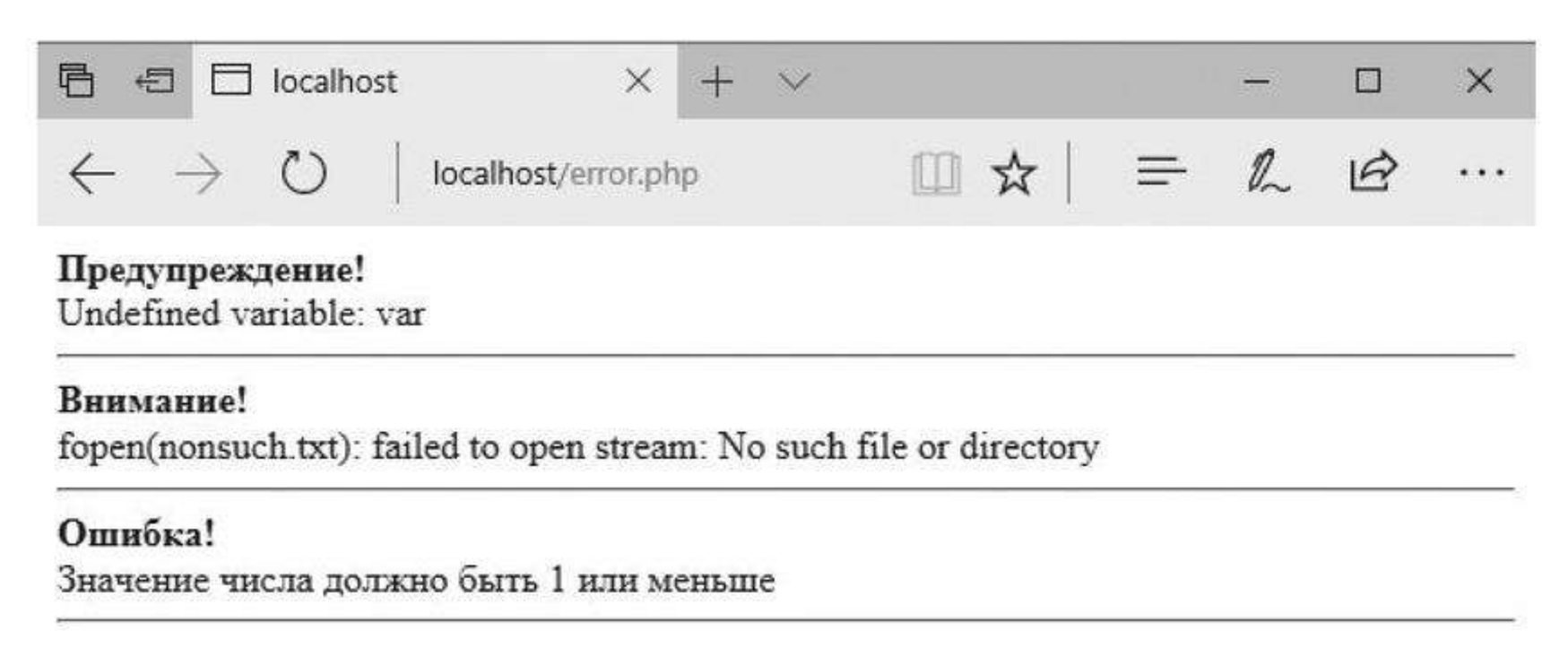
```
$number = 2;
if($number >= 1)
```

```
{
 trigger _ error('Числовое значение должно быть 1 или
 меньше');
}
```

На заметку

Обратите внимание, что сценарий продолжает работу и после сообщения о каждой ошибке; их появление не приводит к завершению работы.

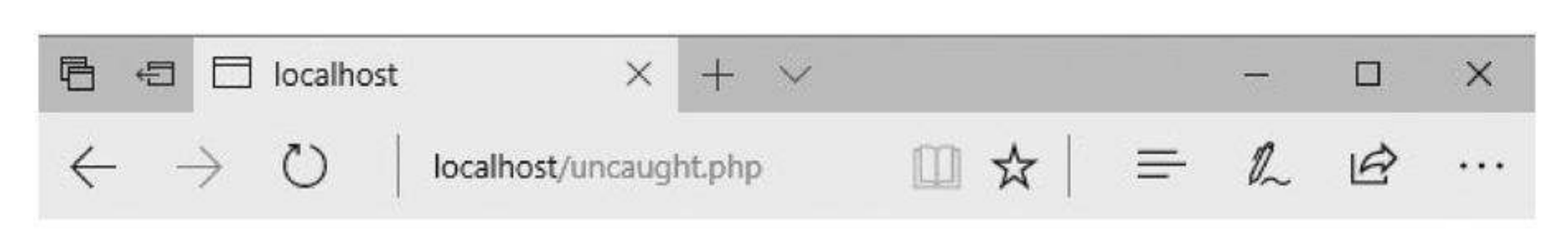
5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *error.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть уровни обнаруженных ошибок и соответствующие сообщения.



Перехват исключений

Можно определить поведение интерпретатора в случаях, когда в выполняемом сценарии обнаруживается определенная сбойная ситуация, называемая исключением. Когда возникает, т.е. «запускается» исключение, его обработка называется «перехватом». Если исключение не перехвачено, PHP-интерпретатор определит его как критическую ошибку с неприглядным сообщением «Неперехваченное исключение»*, например так:

* Текст «Uncaught Exception» на рисунке.



Fatal error: Uncaught Exception: Число: 5 Значение должно быть больше 10 in C:\Abyss Web Server\htdocs\uncaught.php:7 Stack trace: #0 C:\Abyss Web Server\htdocs\uncaught.php(10): check_size(5) #1 {main} thrown in **C:\Abyss Web Server\htdocs\uncaught.php** on line 7

Внимание

Каждый блок `try` должен быть ассоциирован с соответствующим блоком `catch`, в противном случае возникнет критическая ошибка «Неперехваченное исключение».

Основная обработка исключений заключается в прогнозировании возникновения ошибки: предпринимается попытка выполнить действие в пределах блока кода `try` и определить, каким образом сценарий должен выполняться в соответствующем блоке кода `catch`. Блок перехвата передается классу `Exception`, содержащему методы `getMessage()`, `getFile()` и `getLine()`, для извлечения данных об исключении.

Экземпляр класса `Exception` можно создать, используя ключевое слово `new` и явно запустив его с помощью ключевого слова `throw`.

Пользовательский обработчик исключений также может быть создан как дочерний класс `Exception`, передавая собственные методы.

1. Начните PHP-сценарий с определения функции, которая запустит исключение, если проверка условия провалена.

exception.php

```
<?php
function check _ size($num)
{
 if($num < 10)
 {throw new
 Exception("Число: $num<br>Значение должно быть больше
 10");}
}
```

2. Спрогнозируйте специфичную ошибку и сообщите об исключении.

```
try {check _ size(5);}
```

```
catch(Exception $e)
{    echo '<b>Исключение размера!</b><br>'.
$e->getMessage().'<hr>';    }
```

3. Теперь создайте класс для обработки пользовательских исключений.

```
class CustomException extends Exception
{
 public function get_details()
 {
  $details = 'Файл: '.$this->getFile().
  '<br>Строка: '.$this->getLine().
  '<br>'.$this->getMessage();
  return $details;
 }
}
```

На заметку

Более подробную информацию о производных классах и наследовании см. в главе 7.

4. Теперь добавьте еще одну функцию, которая сгенерирует исключение, если проверка условия закончилась неудачей.

```
function check_parity($num)
{
 if($num% 2!== 0)
 {
  throw new CustomException("Число: $num<br>Значение
  должно быть четным");
 }
}
```

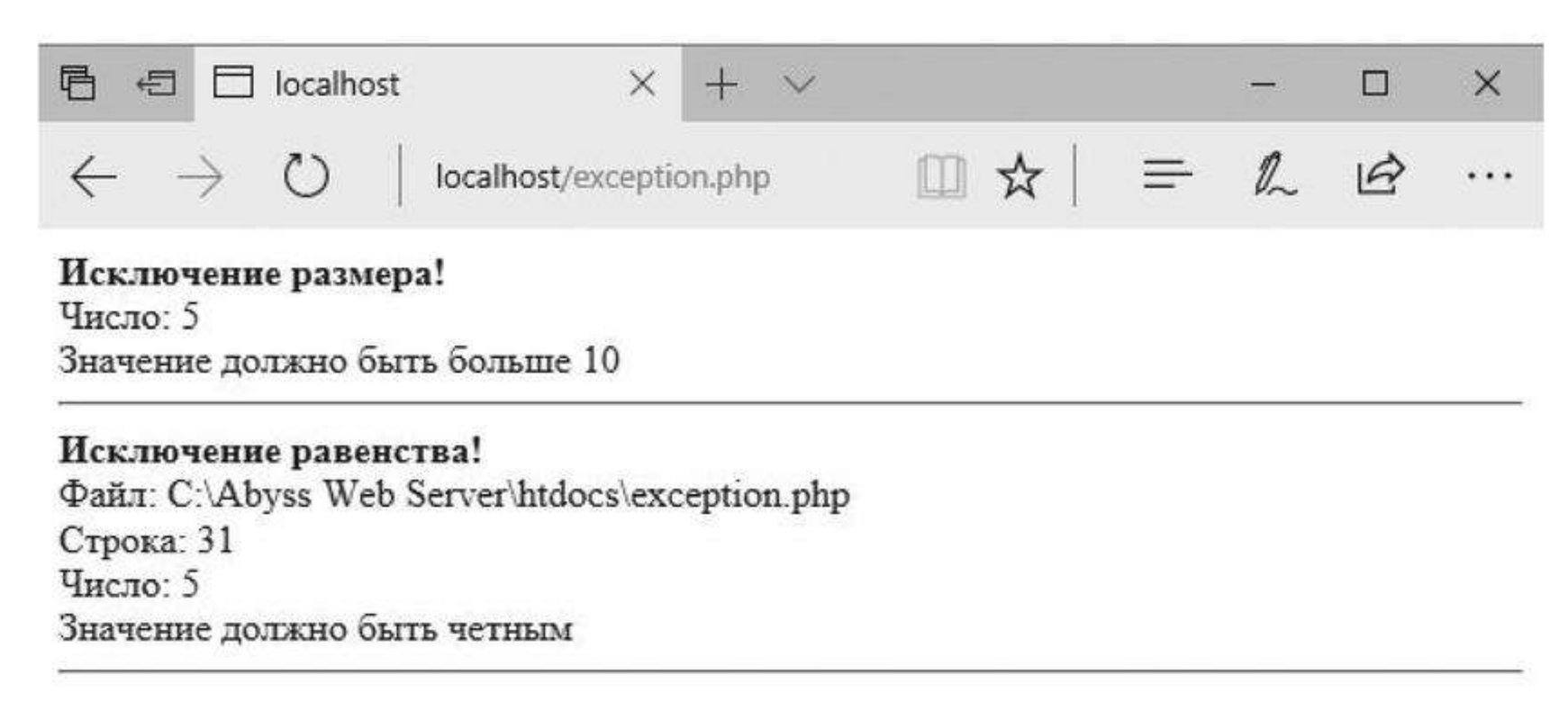
5. И, наконец, спрогнозируйте специфичную ошибку и сообщите об исключении.

```
try {check_parity(5);}
catch(CustomException $e)
{echo '<b>Исключение равенства!</b><br>'.
$e->get_details().'<hr>';}
```

Совет

Блок `try` может содержать несколько инструкций `throw` для определения нескольких ошибок и может сопровождаться несколькими блоками `catch` для обработки этих исключений.

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *exception.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат перехвата исключений и соответствующие сообщения.



Заключение

- Функция `fopen()` используется для создания файлового потока и принимает два аргумента, определяющие имя файла и режим доступа к файлу.
- Рекомендуется обеспечивать альтернативное поведение сценария с помощью функции `die()` на случай, если попытка создать файловый поток потерпит неудачу.
- Функция `fread()` используется для чтения файлового потока в режиме `'r'`, а функция `filesize()` может быть применена для указания размера файла в байтах.
- После завершения любых операций с файлом файловый поток следует закрыть с помощью вызова функции `fclose()`.
- Функция `fgets()` используется для чтения содержимого файла построчно, а функция `feof()` при этом позволяет определить конец файла.
- Функция `fgetc()` используется для чтения содержимого файла посимвольно и возвращает `FALSE` при достижении конца файла.

- Текст может быть записан в файл, если указать файловый поток и имя файла в качестве аргументов функции `fwrite()`.
- Функция `fwrite()` будет перезаписывать существующий текст в файле при использовании режима `'w'` и добавлять текст к существующему при использовании режима `'a'`.
- Для создания пользовательского обработчика ошибок имя функции должно быть указано в качестве аргумента функции `set_error_handler()`.
- Функция обработчика ошибок вызывается либо при возникновении ошибки, либо явно с помощью функции `trigger_error()`.
- Исключение — это возникновение определенной сбойной ситуации или ошибки.
- Если исключение не перехвачено, PHP-интерпретатор посчитает его критической ошибкой и выдаст сообщение «Неперехваченное исключение».
- Блок кода `try` используется для прогнозирования конкретной ошибки, а связанный с ним блок `catch` — для определения поведения сценария при возникновении данной ошибки.
- Класс `Exception` содержит методы `getMessage()`, `getFile()` и `getLine()`, которые могут извлекать сведения об исключении.
- Пользовательский обработчик исключений может быть создан в качестве дочернего класса `Exception`, позволяя обеспечить специальные методы исключения.

9

Разработка веб-форм

Эта глава познакомит вас с приемами использования PHP-кода для обработки данных веб-форм.

- Выполнение действий
- Проверка заполнения формы
- Проверка введенных данных
- Фильтрация данных
- Передача скрытых данных
- Обработка данных формы
- Веб-формы с сохранением данных
- Выгрузка файлов
- Группировка элементов формы
- Передача данных в сценарий
- Заключение

Выполнение действий

Обработка данных, переданных из веб-формы на HTML-странице, пожалуй, основная задача, выполняемая с помощью PHP-сценариев.

Атрибут `action` HTML-элемента `form` используется для назначения PHP-сценария, который будет обрабатывать данные, полученные из веб-формы. Кроме того, атрибут `method` HTML-элемента `form` используется для определения, каким образом данные передаются в сценарий, — как правило, с помощью метода `POST`. Таким образом, HTML-элемент `form` с атрибутами может выглядеть следующим образом:

```
<form action="script.php" method="POST">
```

Язык PHP содержит специальный «суперглобальный» массив `$_POST`, в котором хранятся данные, полученные из веб-формы и переданные с помощью метода `POST`. Этот массив содержит элементы с теми же именами, что и каждый из элементов веб-формы, и их значениями. К примеру, рассмотрите следующий код:

```
<input type="text" name="email" >
```


Внимание

В данном случае регистр символов очень важен, поэтому прописные и строчные буквы должны указываться в точности, как написано в книге. Например, варианты `$_POST['EMail']` и `$_Post['email']` неправильны.

При передаче данных из веб-формы PHP создает элемент массива с именем `$_POST['email']`, содержащий значение, введенное в это поле пользователем.

Для успешного распознавания PHP-переменных с теми же именами, что и элементы формы, этим переменным могут быть присвоены значения, хранящиеся в соответствующих элементах массива `$_POST`. Затем значения могут выводиться на ответной странице, созданной с помощью сценария обработчика действий.

1. Создайте HTML-документ, содержащий форму с тремя полями ввода текста для передачи данных.

action.php

```
<form action="action _ handler.php"
method="POST">
<dl>
<dt>Имя:
<dd><input type="text" name="name">
<dt>Адрес email:
<dd><input type="text" name="mail">
<dt>Примечание:
<dd><textarea rows="5" cols="20" name="comment" >
</textarea>
</dl>
<p><input type="submit" ></p>
</form>
```

2. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *action.php*.

3. Затем создайте HTML-документ, содержащий сценарий для назначения значений из веб-формы одноименным PHP-переменным для вывода.

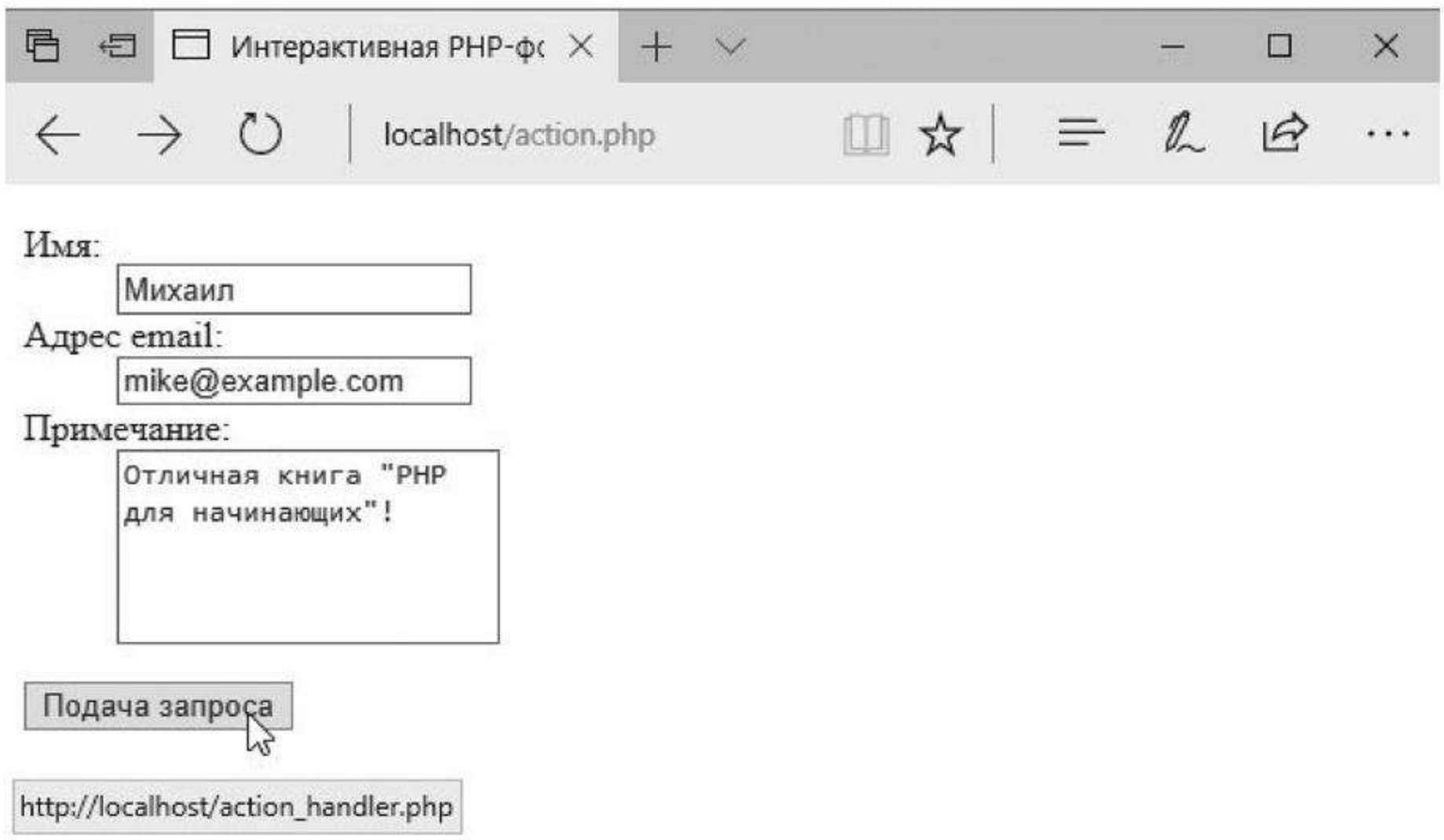
action_handler.php

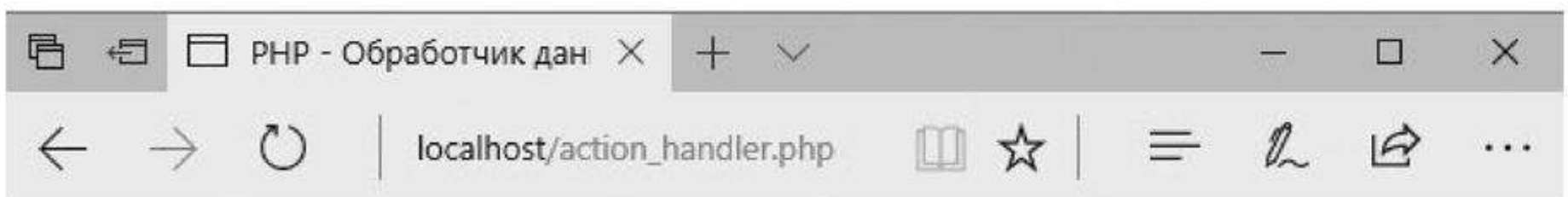
```
<?php
$name = $ _ POST['name'];
$mail = $ _ POST['mail'];
$comment = $ _ POST['comment'];
echo "<p>Благодарим за примечание, $name …</p>";
echo "<p><i>$comment</i></p>";
echo "<p>Ответ будет отправлен на адрес $mail</p>";
?>
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *action_handler.php*, а затем откройте страницу *action.php* в браузере через протокол HTTP и введите любые данные. После нажатия кнопки передачи данных в веб-форме вы увидите ответную страницу с данными, обработанными PHP-сценарием.

Совет

Существует два метода передачи данных веб-формы — `GET` и `POST`. Метод `GET` присоединяет пары имя=значение к URL-адресу и в основном используется для запроса данных, например извлечения записей из базы данных. Метод `POST` позволяет передавать больший объем данных и более безопасен, поскольку данные не добавляется к URL-адресу. В основном метод `POST` используется, когда должно быть выполнено некоторое действие — например обновление записи в базе данных. Метод `POST` используется для большинства примеров в этой книге.





Благодарим за примечание, Михаил ...

Отличная книга "PHP для начинающих"!

Ответ будет отправлен на адрес mike@example.com

Проверка заполнения формы

С помощью PHP можно легко проверить, заполнил ли пользователь представленные поля HTML-формы или нет. Для этого используется встроенная функция `isset()`. Эта функция принимает имя переменной в качестве аргумента и возвращает `TRUE`, только если значение этой переменной отлично от `NULL` — гарантируя, таким образом, что переменная была «установлена» с некоторым значением. Пустые элементы формы передают `NULL`, если пользователь не ввел в них значение.

Проверка условий, выполненная с помощью функции `isset()`, позволяет выделить элементы формы, которые не были заполнены, либо передать ответную страницу с сообщением.

1. Создайте HTML-документ, содержащий форму с тремя положениями переключателя, настроенных как единый элемент формы, передающий данные.

isset.php

```
<form action="isset _ handler.php" method="POST">
<fieldset>
<legend>К какому типу относится язык PHP?</legend>
Структурный<input type="radio"
name="definition" value="Структурный"> <br>
Процедурный<input type="radio"
name="definition" value="Процедурный"> <br>
Объектно ориентированный<input type="radio"
name="definition" value="Объектно ориентированный">
</fieldset><p><input type="submit" ></p>
</form>
```

2. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *isset.php*.

3. Затем создайте HTML-документ, содержащий PHP-сценарий, инициализирующий переменную с полученным значением, либо `NULL`, если значение не было передано.

isset_handler.php

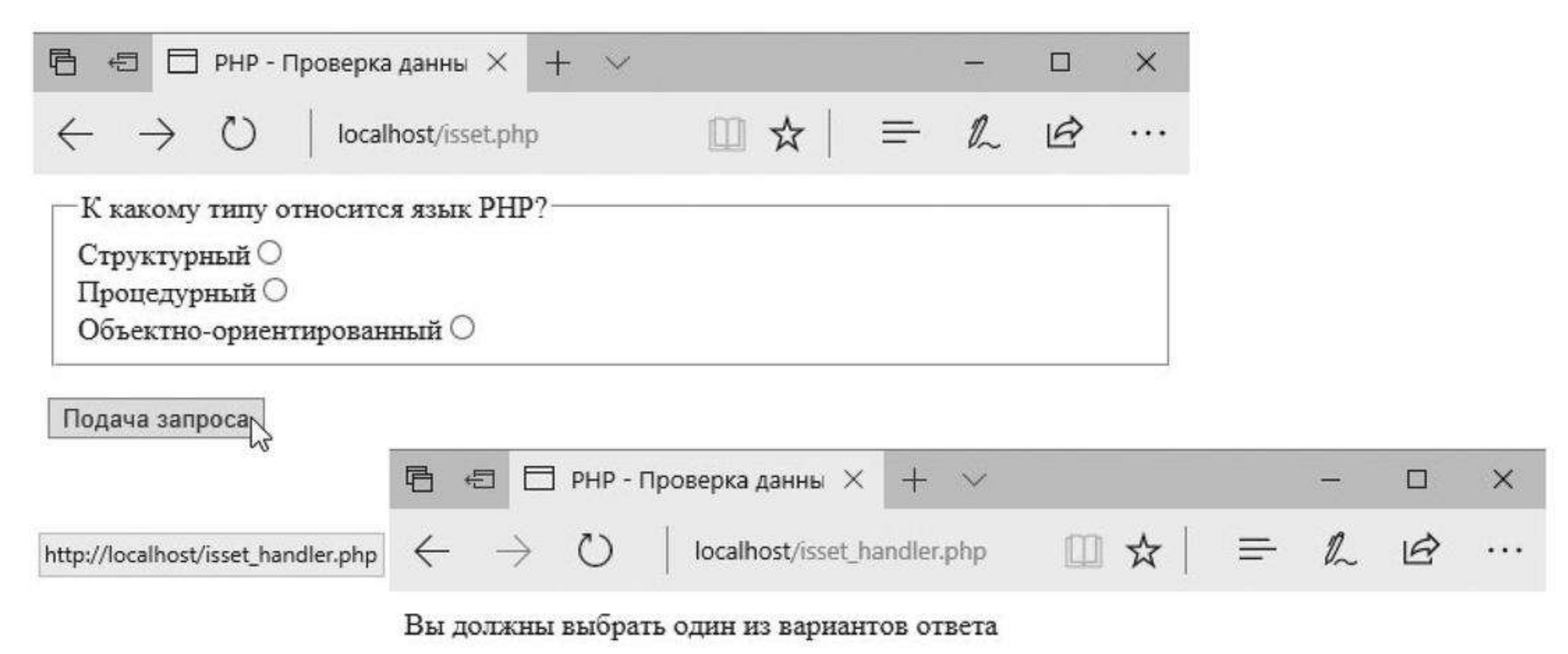
```
<?php
if (isset($ _ POST['definition']))
{
 $definition = $ _ POST['definition];
```

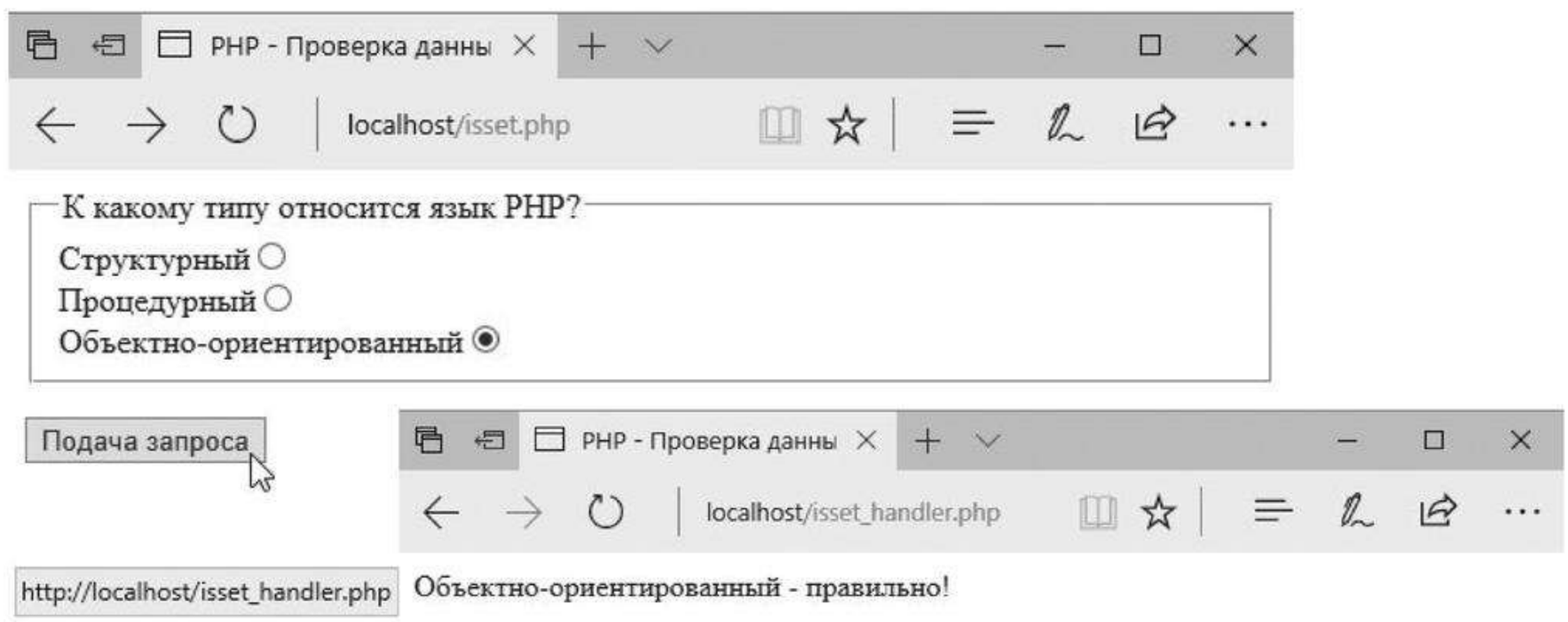
```
}
else
{
 $definition = NULL;
}

# Сюда вставляются инструкции ответа (шаг 4).

?>
```

4. И, наконец, добавьте инструкции для вывода ответа, соответствующего значению переменной.

```
if ($definition!= NULL)
{
 if ($definition!= 'Объектно ориентированный')
 {echo "$definition — неправильно!";}
 else
 {echo "$definition — правильно!";}
}
else
{echo 'Вы должны выбрать один из вариантов ответа';}
?>
```

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *isset_handler.php*, а затем откройте страницу *isset.php* в браузере через протокол HTTP и установите переключатель в одно из положений. После нажатия кнопки передачи данных в веб-форме вы увидите ответную страницу с данными, обработанными PHP-сценарием.





Совет

Совет. Проверка значения на ложность технически эффективнее, чем проверка на истинность, поскольку требует проверки меньшего числа вариантов.

Проверка введенных данных

Поскольку пользователи могут случайно или намеренно отправлять формы с пустыми полями или с данными, указанными в неправильном формате, то необходимо с помощью PHP-кода проверять, заполнены ли необходимые поля и допустимы ли передаваемые данные.

Чтобы проверить, ввел ли пользователь какое-либо значение в текстовое поле, вы можете использовать функцию `empty()`, которая принимает переменный аргумент и возвращает `TRUE`, если его значение является пустой строкой, нулем, `NULL` или `FALSE`.

Чтобы убедиться, что пользователь ввел данные в числовом формате, язык PHP предоставляет функцию `is_numeric()`, которая принимает переменный аргумент и возвращает `TRUE`, только если его значение является числом.

Более тщательные проверки данных можно выполнить с помощью PHP-функции `preg_match()`, которая принимает два аргумента: «регулярное выражение» и переменную для поиска по шаблону. Как правило, такой прием используется для проверки допустимости формата адреса электронной почты.

1. Создайте HTML-документ, содержащий форму с двумя текстовыми полями для ввода данных.

valid.php

```
<form action="valid _ handler.php"
method="POST">
<fieldset>
<legend>Введите количество и адрес email</legend>
<p>Количество: <input type="text" name="quantity"></p>
<p>Адрес email: <input type="text" name="email"></p>
</fieldset><p><input type="submit" ></p>
</form>
```

2. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *valid.php*.

3. Теперь создайте HTML-документ, содержащий PHP-сценарий, инициализирующий переменную с полученным значением, либо NULL, если значение не было передано.

valid_handler.php

```
<?php
if (!empty ($ _ POST['quantity']))
{
$quantity = $ _ POST['quantity];
# Сюда вставляется код проверки формата введенных данных
# (шаг 4)
}
else
{$quantity = NULL; echo 'Необходимо указать
количество<br>';}
```

4. Добавьте инструкции для проверки, является ли введенное значение количества численным.

```
if (!is _ numeric($quantity))
{$quantity = NULL; echo 'Количество нужно указать в виде
числа<br>';}
```

5. Теперь добавьте инструкции, инициализирующие еще одну переменную с полученным значением, либо NULL, если значение не было передано.

```
if (!empty ($ _ POST['email']))
{
 $email = $ _ POST['email];
```

```
 # Сюда вставляется код проверки формата введенных
 # данных (шаг 6)
}
else
{$email = NULL; echo 'Необходимо указать адрес
электронной почты';}
```

6. Затем добавьте инструкции, определяющие, что адрес электронной почты указан в допустимом формате.

```
$pattern = '/\b[\w. –]+@[\w. –]+\.[A-Za-z]{2,6}\b/';
if (!preg _ match($pattern, $email))
{$email = NULL; echo 'Адрес электронной почты указан
в недопустимом формате';}
```

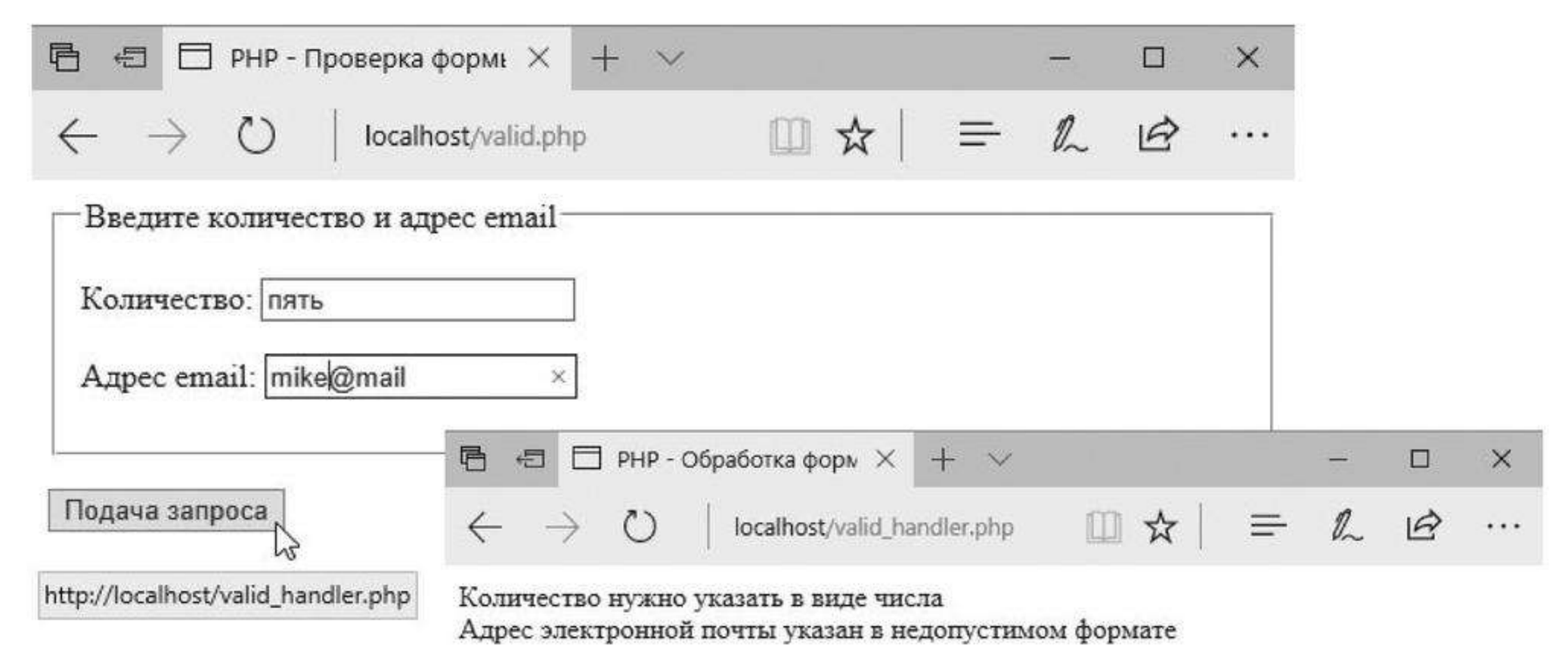
Совет

Тема регулярных выражений выходит за рамки этой книги, поэтому вам нужно в точности скопировать пример кода, проверяющего допустимость формата введенного адреса электронной почты.

7. Наконец, добавьте инструкцию для вывода переданных корректных данных.

```
if (($quantity!= NULL) && ($email!= NULL))
{echo "Адрес email: $email<br>Количество: $quantity ";}
?>
```

8. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *valid_handler.php*, а затем откройте страницу *valid.php* в браузере через протокол HTTP и отправьте данные формы, чтобы увидеть ответ сервера.



Фильтрация данных

Для простой проверки введенных пользователем данных язык PHP содержит функцию `filter_var()`, которая может использоваться как для проверки, так и для «очистки» входных значений. Эта функция требует два аргумента, позволяющие указать переменную, значение которой должно быть отфильтровано, и идентификатор применяемого фильтра.

Идентификатор фильтра	Описание
FILTER_VALIDATE_BOOLEAN	Логическое значение
FILTER_VALIDATE_EMAIL	Проверяет, что значение является корректным адресом электронной почты
FILTER_VALIDATE_FLOAT	Проверяет, что значение является корректным числом с плавающей точкой
FILTER_VALIDATE_INT	Проверяет, что значение является корректным целым числом
FILTER_VALIDATE_IP	Проверяет, что значение является корректным IP-адресом
FILTER_VALIDATE_MAC	Проверяет, что значение — это корректный MAC-адрес
FILTER_VALIDATE_REGEXP	Проверяет значение на соответствие регулярному выражению
FILTER_VALIDATE_URL	URL format
FILTER_SANITIZE_EMAIL	Удаляет недопустимые символы
FILTER_SANITIZE_ENCODED	Кодирует строку в формат URL-адреса
FILTER_SANITIZE_MAGIC_QUOTES	Экранирует строку с помощью слешей
FILTER_SANITIZE_NUMBER_FLOAT	Удаляет все символы, кроме цифр
FILTER_SANITIZE_NUMBER_INT	Удаляет все символы, кроме цифр
FILTER_SANITIZE_SPECIAL_CHARS	Экранирует HTML-символы ', ", <, > и &
FILTER_SANITIZE_STRING	Удаляет все HTML-теги
FILTER_SANITIZE_URL	Удаляет недопустимые символы
FILTER_CALLBACK	Вызывается пользовательская функция для фильтрации данных

На заметку

Фильтры проверки возвращают значение TRUE, если проверка прошла успешно, и FALSE в противном случае.

Совет

Некоторые фильтры позволяют указать дополнительные параметры. Например, в качестве параметра фильтра FILTER_VALIDATE_INT можно указать определенный числовой диапазон.

1. Создайте HTML-документ, содержащий сценарий, начинающийся с создания стилизованного заголовка HTML-страницы.

filter.php

```
<?php
$hdr = '<h1 style="color: red">PHP для начинающих</h1>';
echo $hdr;
```

2. Затем сбросьте форматирование заголовка страницы путем удаления HTML-тегов и выведите отфильтрованную версию текстовой строки.

```
$hdr = filter_var($hdr, FILTER_SANITIZE_STRING);
echo "Отфильтрованный заголовок: $hdr";
```

3. Теперь добавьте код для проверки допустимости формата указанного адреса электронной почты.

```
function validate($email)
{
 if (filter_var($email, FILTER_VALIDATE_EMAIL))
 {echo("<hr>$email — ДОПУСТИМЫЙ адрес электронной
 почты");}
 else
 {echo("<hr>$email — НЕДОПУСТИМЫЙ адрес электронной
 почты");}
}
```

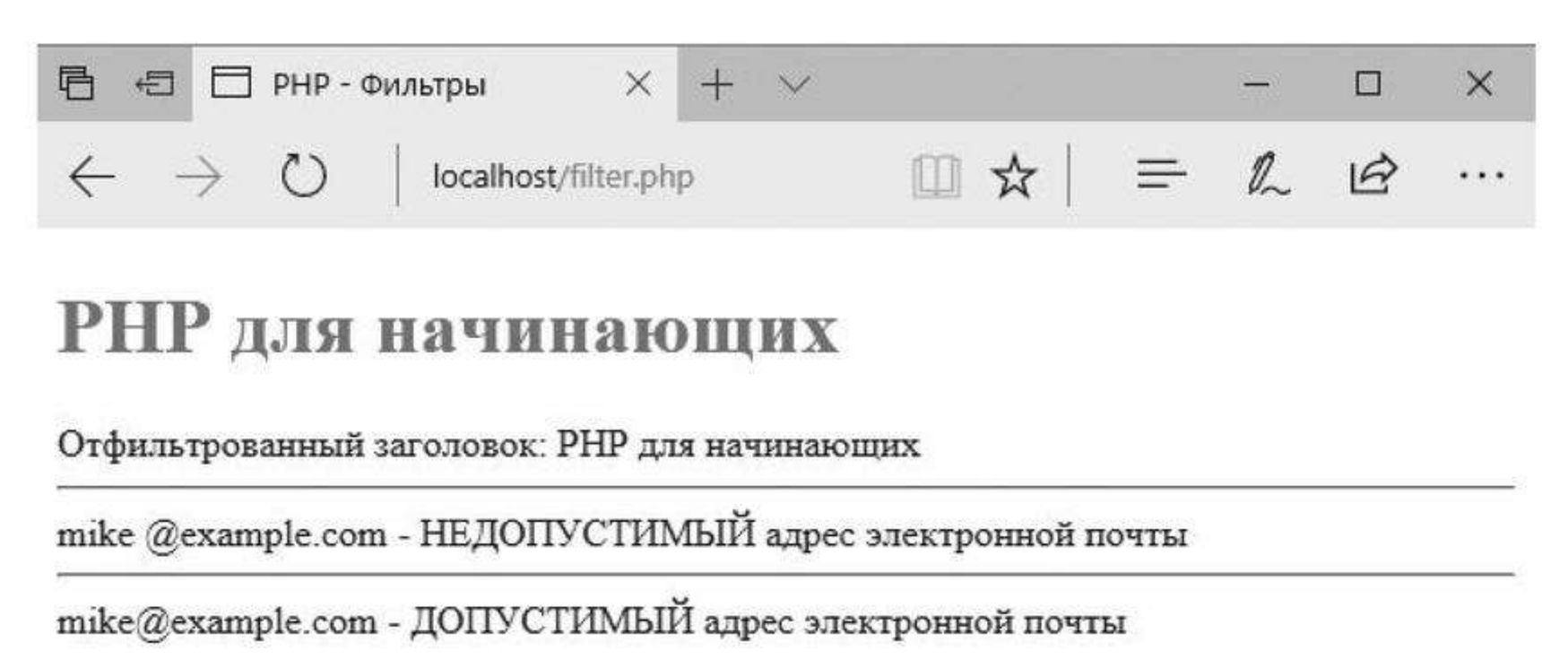
4. Укажите адрес электронной почты, содержащий недопустимый пробел и код проверки адреса. Затем отфильтруйте адрес электронной почты и укажите код повторной проверки адреса.

```
$email = 'mike @example.com';
validate($email);
$email = filter _ var($email, FILTER _ SANITIZE _ EMAIL);
validate($email);
?>
```

Внимание

Передаваемые пользовательские данные могут привести к сбоям в работе вашего веб-сайта, поэтому их всегда следует проверять.

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *filter.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть фильтрацию в действии.



Передача скрытых данных

Помимо данных, вводимых пользователем в поля HTML-формы, вы можете передавать данные, сгенерированные PHP-движком с использованием скрытых элементов HTML-формы. Как и в случае обычных элементов формы, скрытые поля передают пары имя=значение сценарию обработчика формы, но в данном случае значения предоставляются PHP-движком, а не пользователем.

Создаваемые с помощью скрытых элементов форм данные обычно касаются авторизационных данных пользователя либо даты и времени передачи данных формы.

1. Создайте HTML-документ, содержащий PHP-сценарий для установки часового пояса и инициализации двух переменных.

hidden.php

```
<?php
date _ default _ timezone _ set('UTC');
$time = date(' H: i, j F ');
$user = 'Михаил';
```

2. Теперь добавьте инструкцию для создания полноценной формы, содержащей обычное поле ввода и два скрытых элемента, значения которых задаются переменными.

```
echo '
<form action="hidden _ handler.php" method="POST">
<fieldset>
<legend>Оставьте комментарий</legend>
<textarea rows="5" cols="20" name="comment">
</textarea>
<input type="hidden" name="user" value=" '. $user .' ">
<input type="hidden" name="time" value=" '. $time.' ">
</fieldset><p><input type="submit" ></p></form> ';
?>
```

Внимание

Дата и время в этом примере информируют о создании формы, а не передачи ее данных пользователем.

3. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *hidden.php*.

4. Теперь создайте HTML-документ, содержащий PHP-сценарий, инициализирующий переменную либо с переданным значением, либо `NULL`, если значение не было предоставлено.

hidden_handler.php

```
<?php
if (!empty ($ _ POST['comment']))
{
 $comment = $ _ POST['comment'];
}
```

```
else
{
 $comment = NULL;
 echo 'Необходимо указать комментарий';
}
```

5 Теперь добавьте инструкции, инициализирующие две переменные, если значения скрытых элементов формы были установлены.

```
$time =
(!isset ($ _ POST['time']))? NULL: $ _ POST['time'];

$user =
(!isset ($ _ POST['user']))? NULL: $ _ POST['user'];
```

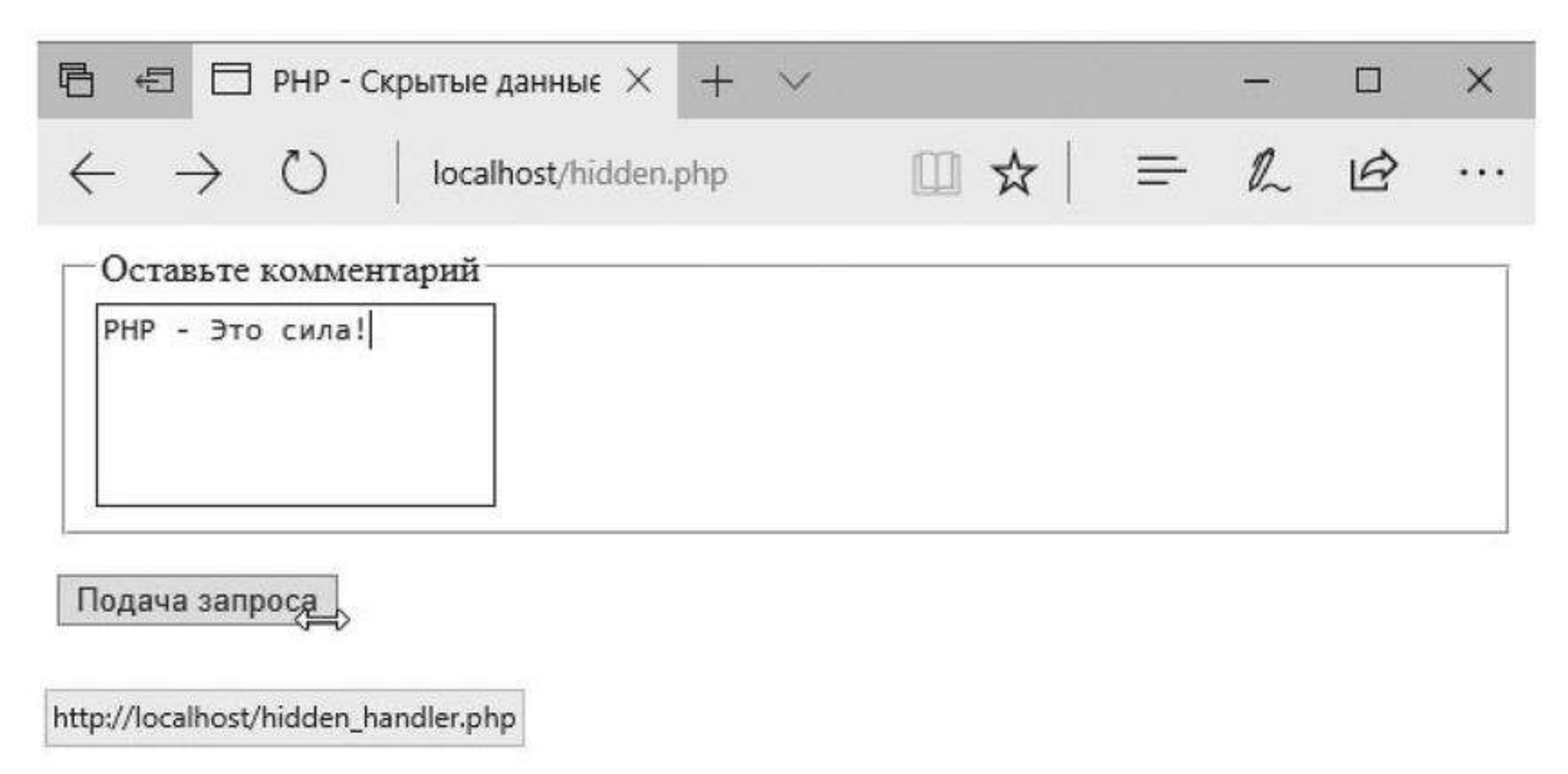
Совет

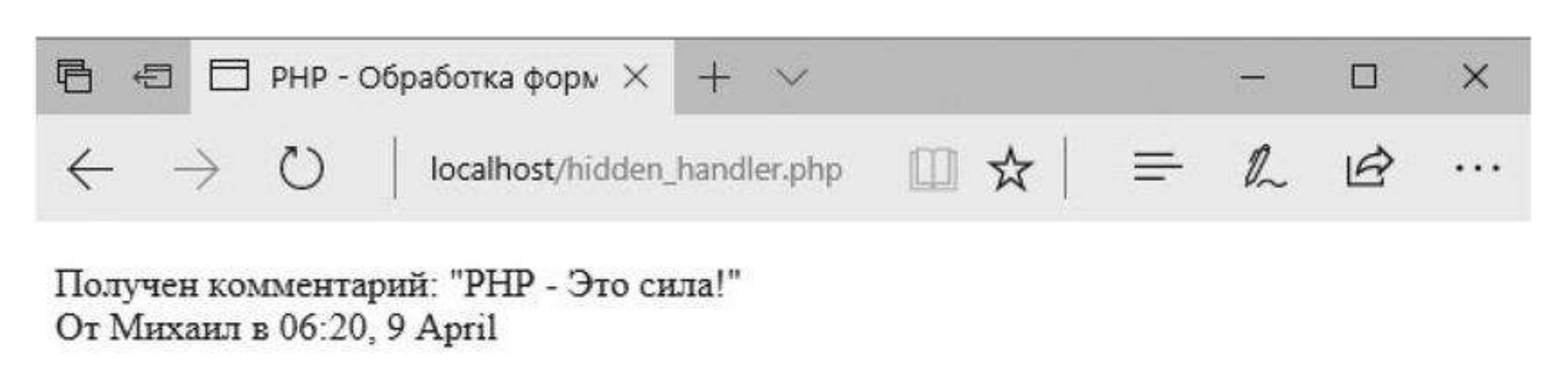
Обратите внимание, что в данном примере тернарный оператор `?:` используется вместо более емкой конструкции `if else`.

6 Наконец, добавьте инструкцию для вывода проверенных переданных данных.

```
if (($comment!= NULL) && ($time!= NULL) && ($user!= NULL))
{echo "<p>Получен комментарий: \" $comment \"<br>
                                 От $user в $time </p>"; }
?>
```

7 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *hidden_handler.php*, а затем откройте страницу *hidden.php* в браузере через протокол HTTP и отправьте данные формы, чтобы увидеть ответ сервера.





На заметку

Вы можете добавлять в вывод кавычки, предваряя их обратным слешем для правильной интерпретации. См. код данного примера.

Обработка данных формы

Предыдущие примеры в этой главе основывались на двух документах. Один использовался для вывода HTML-формы, а другой — для обработки переданных данных формы. Эта процедура тем не менее может быть выполнена с помощью одного документа, если взять на вооружение простую технику, позволяющую определить, следует ли вывести форму или ее данные должны быть обработаны.

Язык PHP предоставляет специальный «суперглобальный» массив `$_SERVER` с элементом `REQUEST_METHOD`, в котором хранится тип запроса. В случае, если для передачи данных в форме используется метод `POST`, при загрузке страницы к PHP-сценарию выполняются два запроса различного типа. Для вывода формы используется стандартный метод `GET`, осуществляемый большинством веб-страниц, в то время как для передачи данных формы применяется метод `POST`. Поэтому для определения дальнейшего поведения интерпретатора следует проверить тип запроса.

При использовании данной техники имя страницы указывается как и обработчик формы в HTML-атрибуте `action`. Вы можете отобразить форму с помощью обычных инструкций `echo`, а проверку данных осуществить таким же образом, как и в примерах с отдельным сценарием обработки формы.

1. Создайте HTML-документ, содержащий PHP-сценарий, определяющий метод запроса страницы.

single.php

```
<?php
if ($_SERVER['REQUEST_METHOD']!= 'POST')
{
 # Сюда вставляются инструкции вывода формы (шаг 2).
}
```

```
else
{
 # Сюда вставляются инструкции обработки формы(шаг 3).
}
```

2. Теперь добавьте инструкцию, выводящую форму, если страница была запрошена с помощью стандартного метода `GET`.

```
echo '
<form action="single.php" method="POST">
<fieldset>
<legend>Необходимо указать комментарий</legend>
<textarea rows="5" cols="40" name="comment">
</textarea>
</fieldset>
<p><input type="submit" ></p>
</form> ';
```

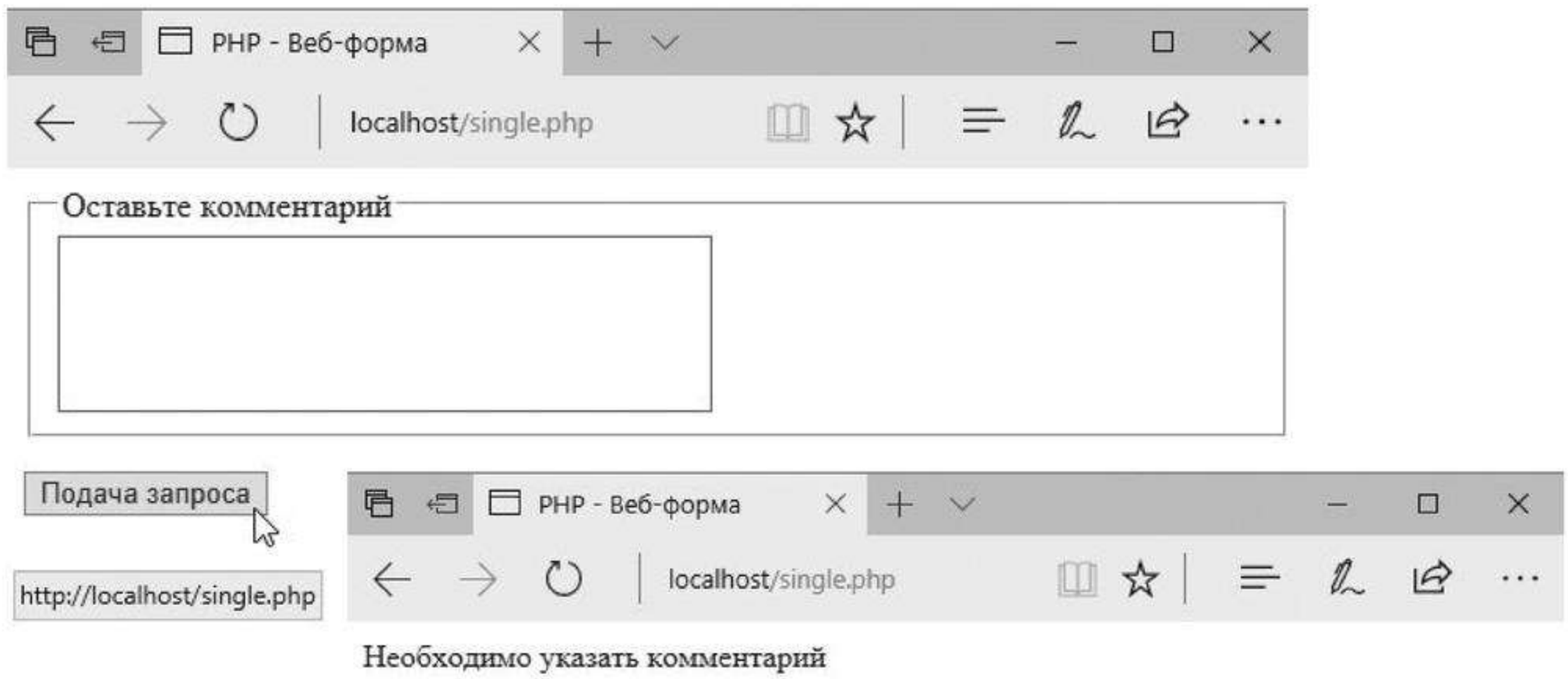
3. Затем добавьте инструкцию для обработки переданных данных формы, если страница была запрошена с помощью метода `POST`.

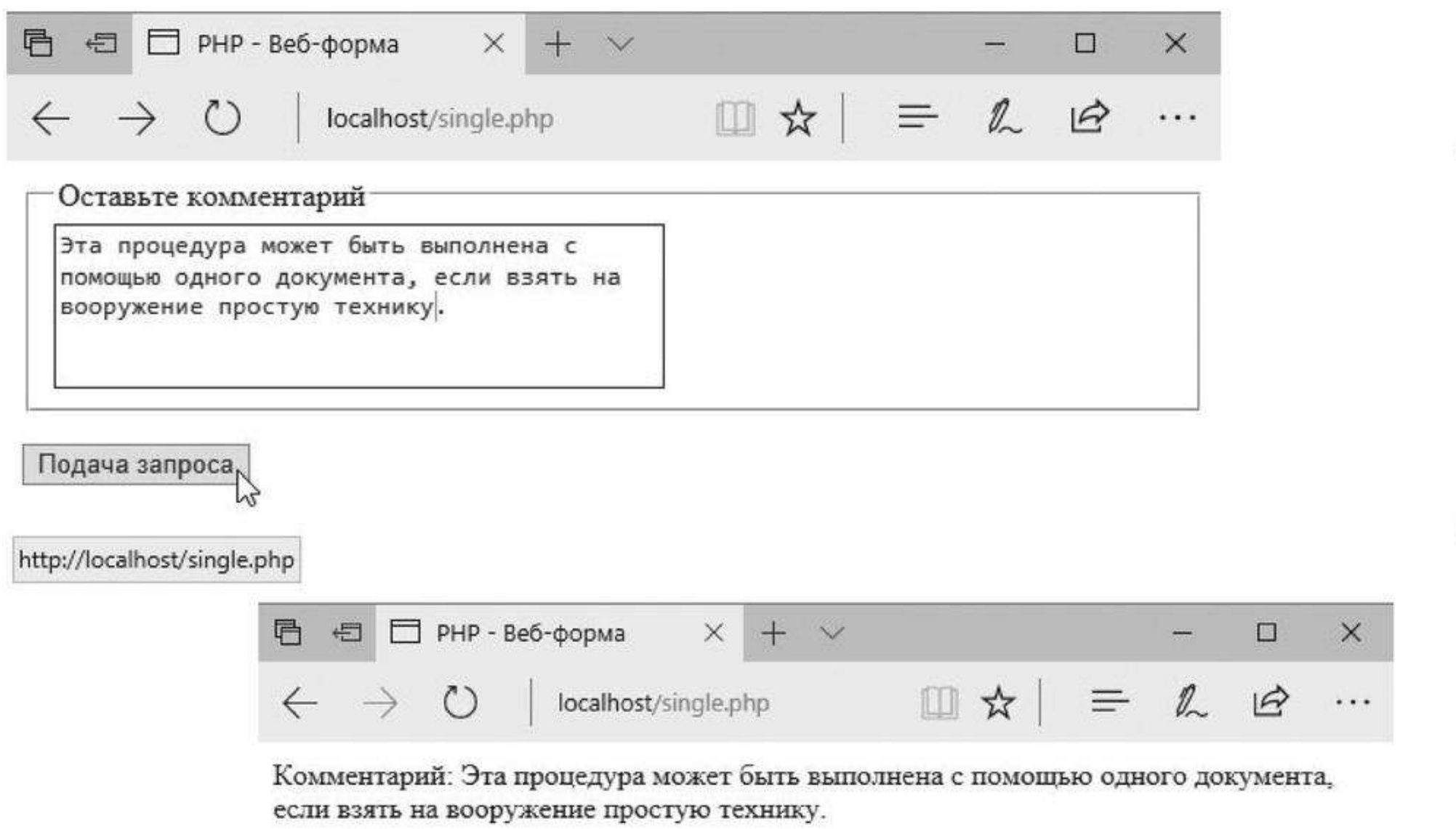
```
if (!empty ($ _ POST['comment']))
{
 $comment = $ _ POST['comment'];
 echo "Комментарий: $comment ";
}
else
{$comment = NULL; echo 'Необходимо указать комментарий';}
?>
```

Совет

Форма будет обрабатываться той же страницей и в том случае, если атрибуту `action` не будет присвоено значение (`action=""`); тем не менее рекомендуется указывать имя файла страницы.

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *single.php*, а затем откройте эту страницу в браузере через протокол HTTP и отправьте данные формы, чтобы увидеть ответ сервера.





Веб-формы с сохранением данных

Если пользователь пытается отправить на сервер данные формы, в которой были заполнены не все необходимые поля, желательно, чтобы данные, введенные в поля, сохранялись и не требовали повторного ввода. Это особенно актуально для веб-форм с большим количеством полей.

Реализовать такое поведение с помощью PHP просто. Достаточно присваивать указанные пользователем данные из массива `$_POST` атрибуту `value` HTML-тегов и затем с помощью инструкций `echo` выводить данные при повторном отображении формы для завершения ее заполнения. Для надежности инструкция должна включать функцию `isset()`, проверяющую, что элемент формы был заполнен до отправки данных формы на сервер.

Сообщения об ошибках для каждого незаполненного элемента формы указываются в массиве для информирования пользователя при повторном выводе формы, как и сообщение об успешном заполнении всех элементов формы.

1. Создайте HTML-документ, содержащий форму с двумя текстовыми полями для ввода имени и адреса электронной почты, позволяющую сохранять введенные данные.

sticky.php

```
<form method="POST"
action="<?php $_SERVER['PHP_SELF']?>" >
<p>Имя:
<input type="text" name="name"
value="<?php if (isset($_POST['name']))
echo $_POST['name];?>"> </p>
<p>Email:
<input type="text" name="email"
value="<?php if (isset($_POST['email']))
echo $_POST['email];?>"></p>
<p><input type="submit"></p>
</form>
```

Совет

Код `$_SERVER['PHP_SELF']` в этом примере используется для обеспечения работоспособности сценария на случай, если он будет переименован.

2. Теперь добавьте PHP-код для обработки переданных данных формы.

```
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
// Сюда вставляются инструкции (шаги 3-5).
}
?>
```

На заметку

Инструкция `if` проверяет, что данные формы были отправлены, определяя, что используется метод запроса `POST`, а не `GET`, который применяется для вывода незаполненной формы.

3. Затем добавьте в PHP-сценарий инструкцию для создания массива сообщений об ошибках.

```
$errors = array();
```

4. Добавьте инструкции, инициализирующие переменные с переданными значениями или добавляющие их в массив `$errors`, если поля не заполнены.

```
if (empty($_POST['name'])) {$errors[] = 'name';}
else {$name = trim($_POST['name']);}

if (empty($_POST['email'])) {$errors[] = 'email';}
else {$email = trim($_POST['email']);}
```

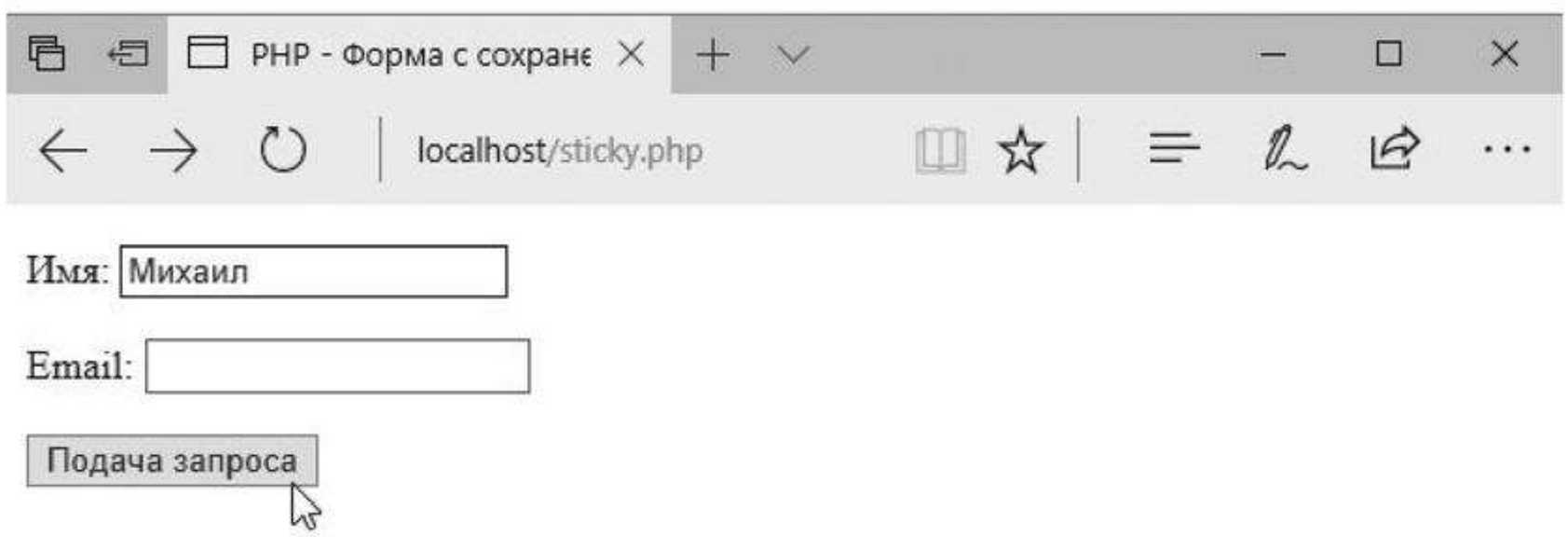
Совет

Рекомендуется использовать функцию `trim()`, как показано в примере, позволяющую удалить все начальные и конечные пробелы из переданного значения элемента формы.

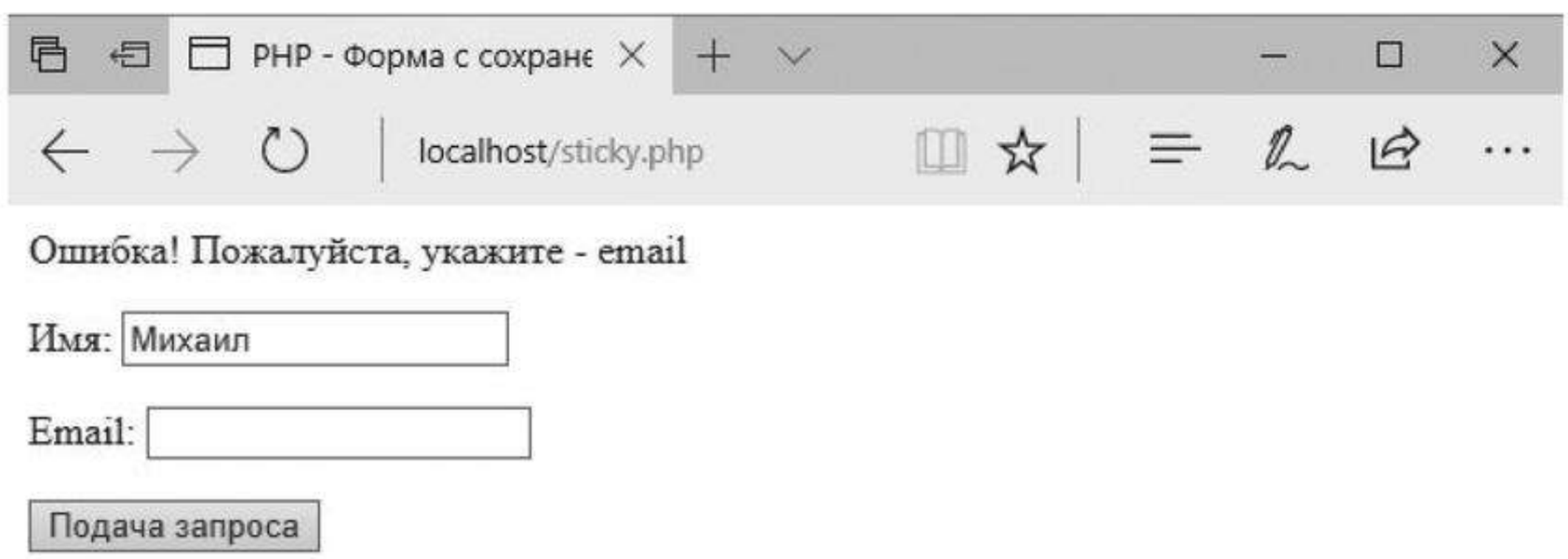
5. Теперь добавьте инструкции с сообщениями об ошибках и подтверждение успешной отправки формы.

```
if(!empty($errors))
{echo 'Ошибка! Пожалуйста, укажите ';
 foreach ($errors as $msg) {echo " — $msg ";}}
else {echo "Форма отправлена! Спасибо, $name ";}
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *sticky.php*, а затем откройте эту страницу в браузере через протокол HTTP и отправьте данные формы, чтобы увидеть ответ сервера и сохраненные значения.



http://localhost/sticky.php



Выгрузка файлов

Суперглобальная переменная `$_FILES` в языке PHP позволяет легко реализовать выгрузку файлов по методу `POST` из HTML-формы, если элемент `form` содержит атрибут `enctype` с присвоенным значением `"multipart/formdata"`. При передаче данных создается ассоциативный массив `$_FILES` с ключами, включающие имя файла (`name`), сгенерированное временное имя (`tmp_name`) и размер файла (`size`).

Имя и размер файла могут использоваться для проверки допустимости данных. Временное имя файла может быть передано функции `move_uploaded_file()` вместе с исходным именем файла для его выгрузки.

1. Создайте HTML-документ, содержащий веб-форму, поддерживающую сохранение данных, с элементом формы типа `file` и именем `image`.

upload.php

```
<form action="<?php $ _ SERVER['PHP _ SELF']?>"
method="POST" enctype = "multipart/formdata" >
Выберите изображение для выгрузки на сервер:
<input type="file" name="image" >
<input type="submit" value="Отправить файл"> </form>
```

Внимание

Для выполнения этого примера настройки PHP должны допускать выгрузку файлов. Если вам не удается выполнить пример, проверьте, что конфигурационный файл ***php.ini*** в каталоге установки PHP-движка содержит строку `file_uploads = On`.

2. Теперь добавьте PHP-код для обработки переданных данных формы.

```
<?php

if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
 // Сюда вставляются инструкции (шаги 3-7).
}

?>
```

3. Затем добавьте инструкции, инициализирующие три переменные с информацией о файле, выбранном пользователем.

```
$name = $_FILES['image']['name'];
$temp = $_FILES['image']['tmp_name'];
$size = $_FILES['image']['size'];
```

4. Теперь добавьте инструкции, проверяющие расширение файла для соответствия выбранного файла допустимому типу и завершающие сценарий в противном случае.

```
$ext = pathinfo($name, PATHINFO_EXTENSION);
$ext = strtolower($ext);
if($ext!= 'png' && $ext!= 'jpg' && $ext!= 'gif')
{echo 'Изображение должно быть формата PNG, JPG или
GIF'; exit();}
```

Совет

Совет. Обратите внимание, как функция `pathinfo()` используется для обработки расширения файла, входящего в состав его имени.

5. Добавьте инструкцию, проверяющую соответствие выбранного файла допустимому размеру и завершающую сценарий в противном случае.

```
if($size > 512000)
{echo 'Файл не должен превышать 500 Кб'; exit();}
```

6. Добавьте инструкцию, проверяющую, что имя выбранного файла не дублирует имя файла, уже находящегося на сервере, и завершающую сценарий в противном случае.

```
if(file _ exists($name))
{echo 'Файл '.$name.' уже был выгружен'; exit();}
```

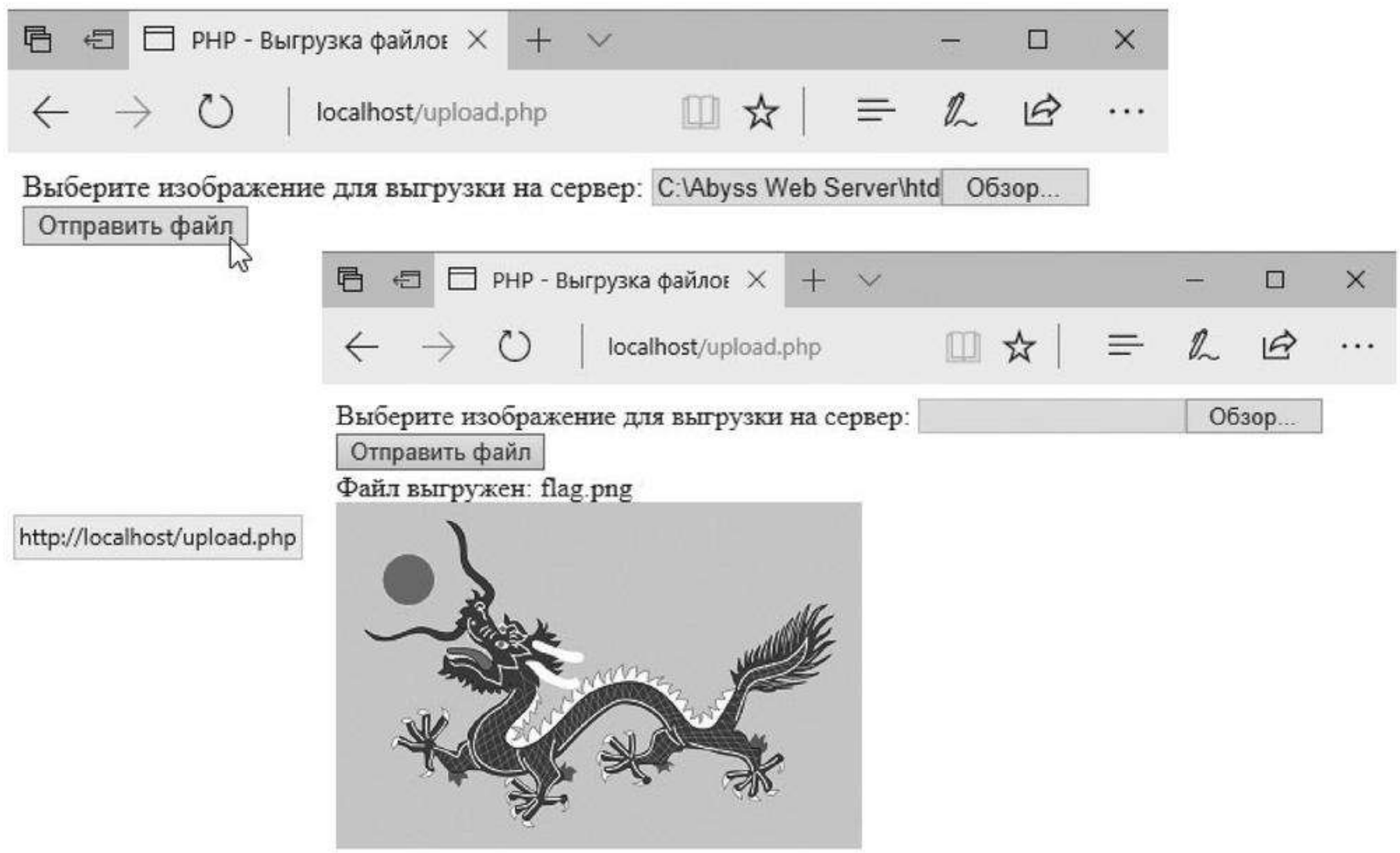
7. И, наконец, добавьте код для выгрузки файла, присвоения ему исходного имени и отображения его под веб-формой.

```
try
{
 move _ uploaded _ file($temp, $name);
 echo 'Файл выгружен: '.$name;
 echo '<br><img src="'.$name.'">';
}
catch(Exception $e)
{
 echo 'Сбой выгрузки файла!';
}
```

Совет

Аргументы функции `move_uploaded_file()` определяют (временное) имя файла и его расположение. Вы можете указать необходимый каталог назначения, например `'uploads/'.$name`.

8. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *upload.php*, а затем откройте эту страницу в браузере через протокол HTTP. Выберите графический файл и выгрузите его в локальный каталог */htdocs*.



Группировка элементов формы

Язык PHP допускает использование нескольких файлов для совместной работы и формирования из них единой веб-страницы. Это означает, что динамические веб-формы могут быть окружены статичным контентом — так называемыми шапкой и подвалом, которые могут быть с легкостью использованы многократно для формирования других страниц сайта.

PHP-функция `include()` в качестве аргумента принимает URL-адрес другого файла, который будет встроен в веб-страницу. Как правило, встраиваются HTML-файлы со статичным контентом, которые следует располагать в каталоге */htdocs/includes* вашего веб-сервера.

Язык PHP также предоставляет функцию `require()`, которая внедряет другие файлы в веб-страницы, также как и функция `include()`, но имеет некоторые отличия. Если функция `include()` не может быть выполнена, поскольку, к примеру, указанный файл не обнаружен, в браузер передается предупреждение, при этом сценарий продолжает выполняться. В отличие от нее функция `require()` выводит ошибку и прекращает работу сценария. Обычно с помощью функции `include()` встраивают статичные HTML-файлы, а функцией `require()` — динамические PHP-сценарии.

Важно отметить, что, в отличие от статичного HTML-заголовка, название веб-страницы, созданной из нескольких файлов, может быть динамически сгенерировано PHP-движком для обеспечения максимальной гибкости.

1. Создайте HTML-документ, содержащий статичный заголовок и фрагмент PHP-кода, генерирующий название страницы.

header.html

```
<! DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="UTF-8">
<title> <?php echo $page _ title;?> </title>
<link rel="stylesheet" href="includes/style.css">
</head>

<body>
<header><h1>Шапка страницы</h1></header>
```

2. Теперь создайте HTML-документ, содержащий контент подвала (нижней части страницы).

footer.html

```
<footer><p>Подвал страницы</p></footer>
</body>
</html>
```

3. Далее создайте папку с именем *includes* в каталоге */htdocs* вашего веб-сервера и сохраните в нем ранее созданные документы под именами *header.html* и *footer.html* соответственно

4. Создайте CSS-файл, форматирующий контент шапки и подвала сайта (и правила оформления для некоторых последующих примеров).

styles.css

```
header > h1 {border-bottom: 1px dashed black;
font-style: italic; font-size: x-large;}
footer > p {border-top: 1px dashed black;
font-style: italic;}
table {border-spacing: 5px; width: 530px;}
th {color: #FFF; background: #000; text-align: left;}
td {border-bottom: 1px solid black;
padding: 3px; background: #F0F0F0;
text-align: left; vertical-align: top;}
p#err _ msg {color: #F00; font-weight: bold;}
```

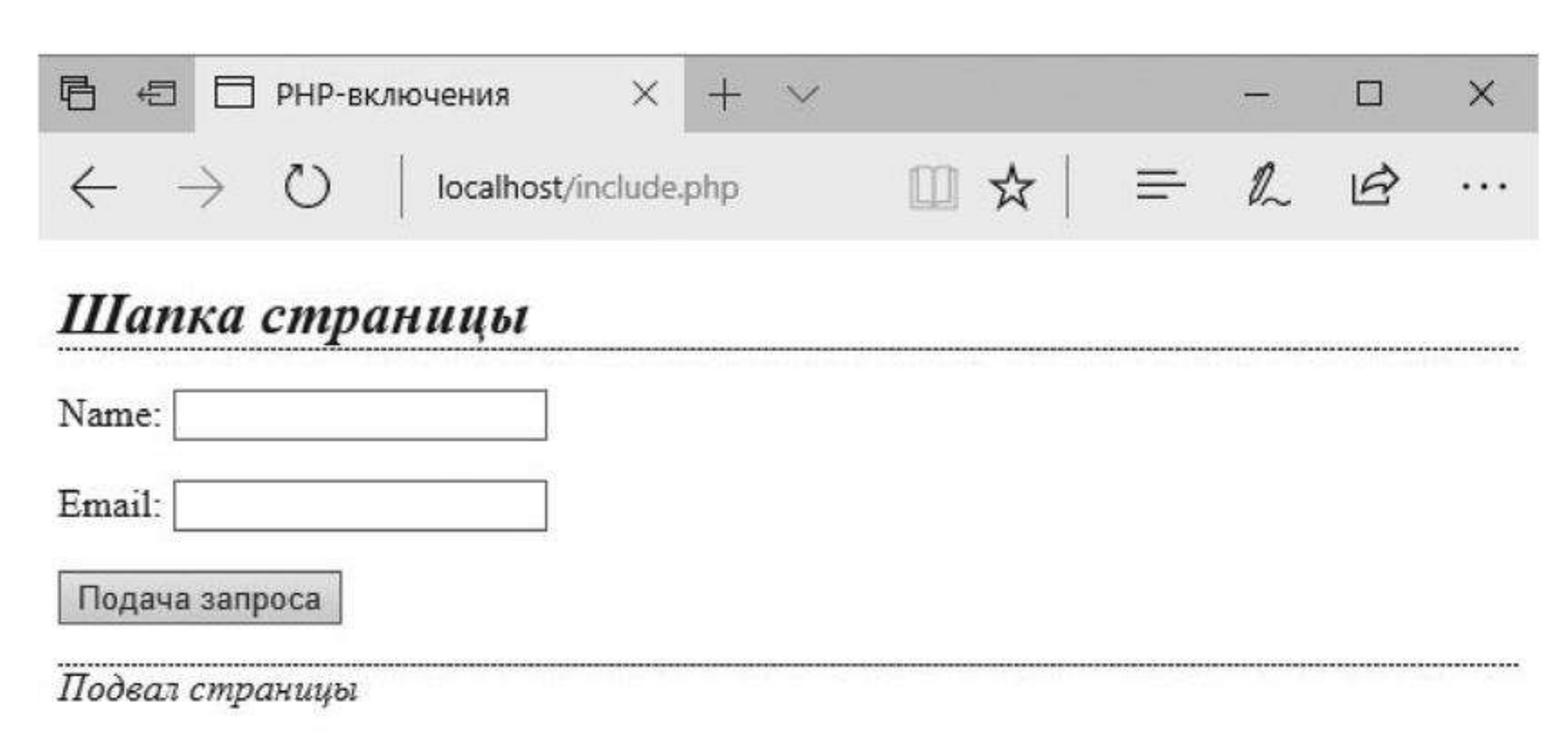
5. Сохраните таблицу стилей в CSS-файл в каталог */includes* вместе с HTML-файлами *header.html* и *footer.html*.

6. И, наконец, создайте PHP-сценарий, устанавливающий название веб-страницы и окружающий форму контентом шапки и подвала.

include.php

```
<?php
$page _ title = 'PHP-включения';
include ('includes/header.html');
echo '<form action="include.php" method="POST">
<p>Имя: <input type="text" name="name"> </p>
<p>Email: <input type="text" name="email" ></p>
<p><input type="submit"></p></form>';
include ('includes/footer.html');
?>
```

7. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *include.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть сгенерированное название страницы и форму, окруженную статичным контентом.



Совет

Вы можете использовать функцию просмотра исходного кода страницы в вашем браузере, чтобы увидеть, что весь контент из разных файлов был собран в единую веб-страницу.

Передача данных в сценарий

Среди предопределенных «суперглобальных» переменных в PHP есть массив `$_SERVER`, хранящий метод запроса страницы, и массив `$_SESSION`, хранящий пользовательские данные независимо от открытой страницы сайта до завершения сессии пользователя. Эти переменные доступны во всех областях видимости всех сценариев.

На заметку

Суперглобальный массив `$_SESSION` будет использоваться для хранения информации о пользователе в главе 10.

Когда HTML-форма передает данные с помощью метода `POST`, в массиве `$_POST` сохраняются переданные значения из полей формы в одноименных элементах массива. Таким образом, к значению, переданному из поля «город» формы, доступ можно получить с помощью запроса `$_POST['город']`.

Аналогичным образом, если данные передаются с использованием метода `GET`, массив `$_GET` также сохраняет переданные значения в именованных элементах массива. Таким образом, значение переданных данных с именем «ID» можно извлечь с помощью кода `$_GET['ID']`. Такой подход, как правило, используется для передачи данных в сценарий, посредством присоединения пары «имя=значения» к URL-адресу HTML-элемента привязки следующим образом:

```
<a href="www.example.com/script.php?id=1">
```

Сценарий получает данные, присоединенные к гиперссылке, и затем может использовать полученное значение для определения ответа.

1. Создайте HTML-документ, содержащий PHP-код, извлекающий элемент с именем `id` из массива `$_GET`.

link.php

```
<?php
if (isset ($ _ GET['id'])
{
# Сюда вставляются инструкции обработки (шаги 2-3).
}
```

2. Затем добавьте инструкцию обработки, присваивающую переданное значение одноименной PHP-переменной.

```
$id = $_GET['id'];
```

3. Теперь добавьте инструкцию обработки, выводящую тот или иной ответ, в зависимости от переданного значения.

```
switch($id)
{
 case 1: echo 'Выбрана корова<hr>'; break;
 case 2: echo 'Выбрана лошадь<hr>'; break;
 case 3: echo 'Выбрана коза<hr>'; break;
}
```

4. Наконец, добавьте код для отображения гиперссылок с присоединяемыми значениями.

```
echo '<h1>Выберите животное</h1>';
echo ' <p><a href="link.php?id=1">Корова</a> |';
echo ' <a href="link.php?id=2">Лошадь</a> | ';
echo ' <a href="link.php?id=3">Коза</a></p>';
?>
```

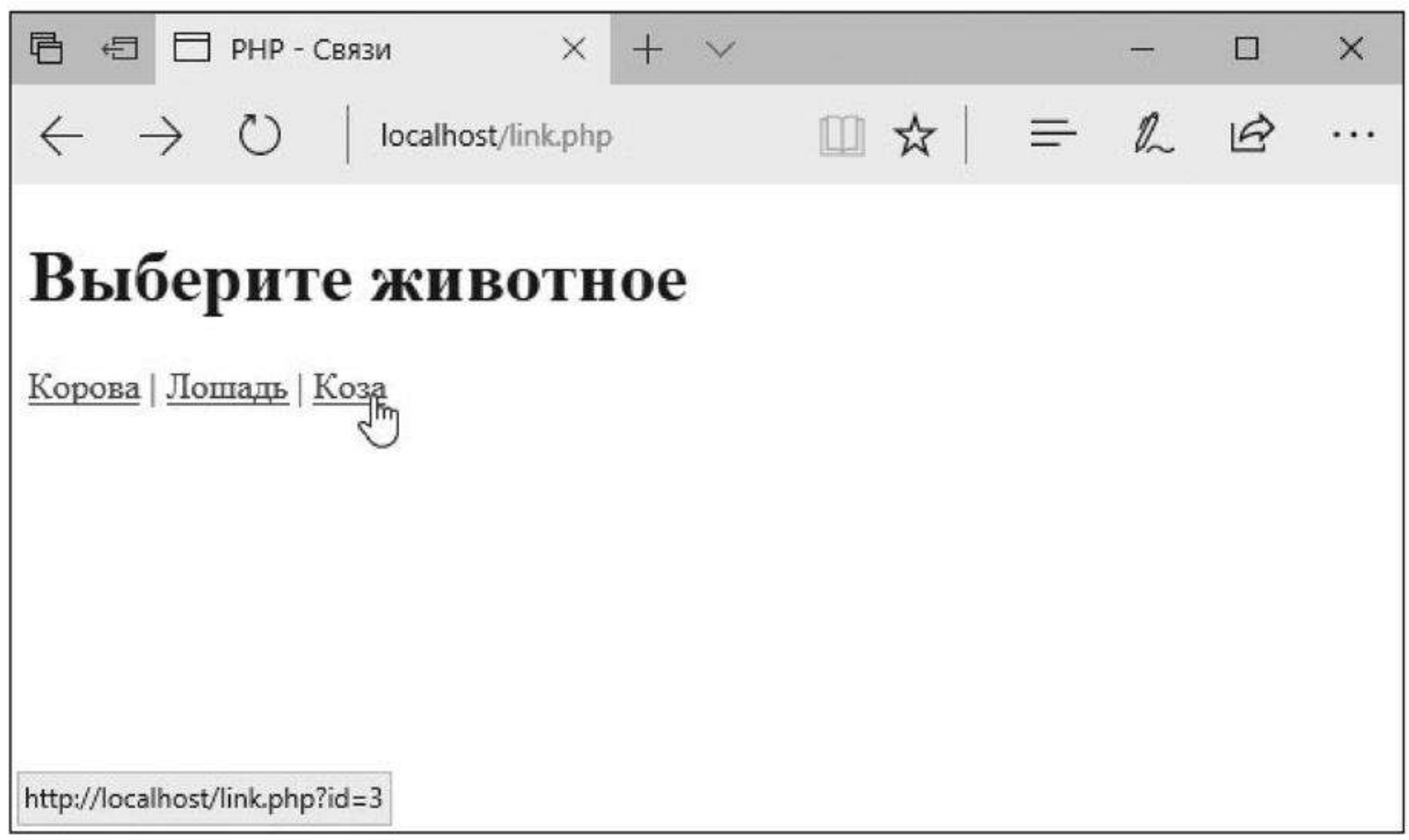

Внимание

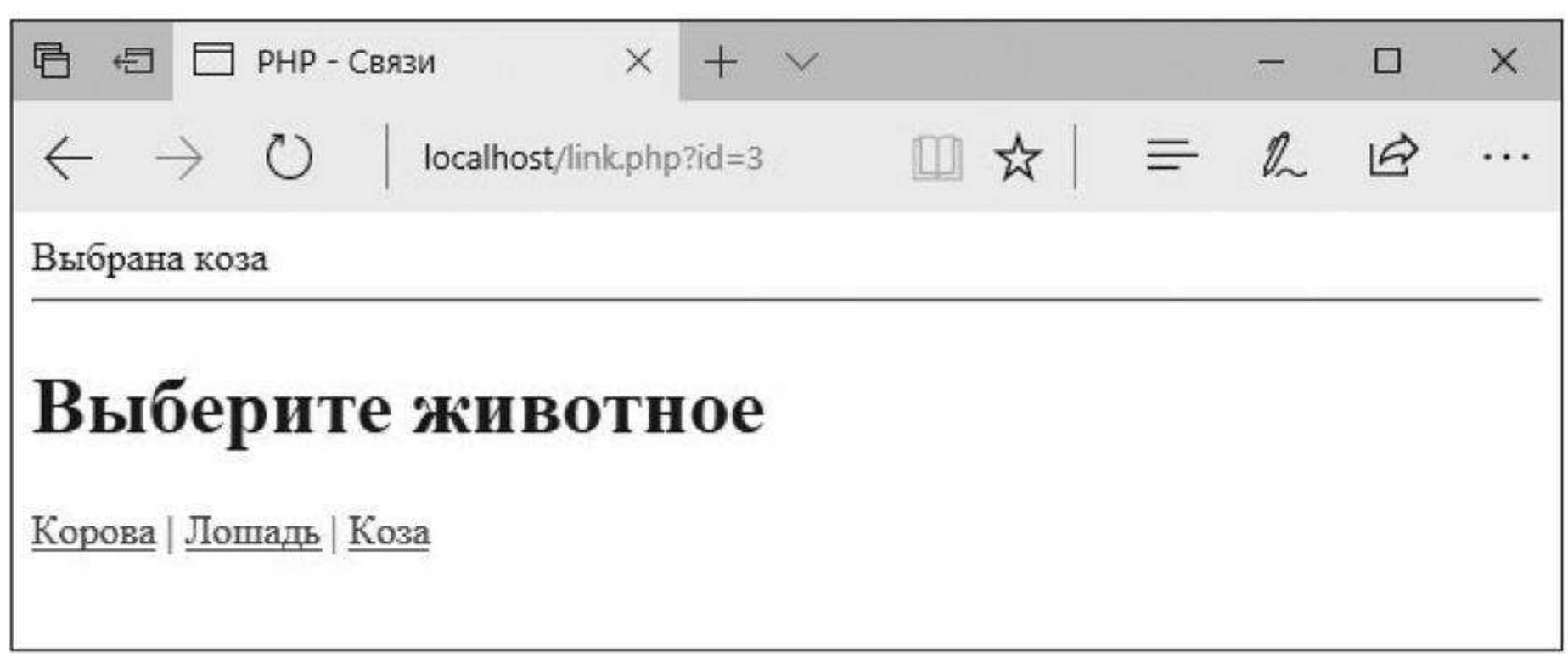
Язык PHP также предоставляет суперглобальную переменную `$_REQUEST`, включающую все содержимое массивов `$_POST` и `$_GET`. Это означает, что запросу `$_POST['city']` аналогичен запрос `$_REQUEST['city']`, но его следует избегать. Всегда используйте массив, соответствующий типу передачи вместо `$_REQUEST`.

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *sticky.php*, а затем откройте эту страницу в браузере через протокол HTTP и щелкните мышью по любой ссылке, чтобы увидеть ответ сервера.

На заметку

Обратите внимание на то, что переданная пара данных «имя=значение» отображается в адресной строке браузера.





Заключение

- Атрибут `action` HTML-элемента `form` позволяет указать PHP-сценарий для обработки переданных данных формы, а атрибут `method` данного элемента определяет, каким образом данные передаются в сценарий.
- Данные формы, переданные с помощью метода `POST`, автоматически сохраняются в суперглобальный массив `$_POST`.
- Элементы массива `$_POST` именуются идентично элементам HTML-формы, содержащим каждое переданное значение.
- С помощью PHP можно проверять, ввел ли пользователь данные в предоставленные ему элементы HTML-формы, используя встроенную функцию `isset()`.

- PHP-функция `empty()` проверяет, было ли введено значение в текстовое поле формы.
- Проверка формата введенных данных может быть выполнена с помощью функций `is_numeric()` и `preg_match()`.
- Данные, генерируемые с помощью PHP, такие как метки времени, создаваемые с помощью функции `date()`, могут передаваться от скрытых элементов формы.
- Суперглобальный массив `$_SERVER['REQUEST_METHOD']` хранит тип запроса, выполненный для загрузки страницы.
- Формы с сохранением данных предотвращают повторный ввод значений, присваивая их при передаче атрибуту `value` элемента `input`.
- PHP-функции `include()` и `require()` используются для встраивания другого контента и сценариев в веб-страницу.
- Данные, переданные с помощью метода `GET`, автоматически сохраняются в суперглобальный массив `$_GET`.
- Гиперссылки могут включать пары «имя=значение», присваиваемые при передаче данных в сценарий с помощью метода `GET`.

10

Сохранение данных

Эта глава расскажет вам, как сохранять данные с помощью cookie-записей и PHP-сессий.

- Передача cookie-данных
- Установка cookie-записей
- Получение cookie-записей
- Просмотр cookie-записей
- Передача данных сессии
- Настройка сессий
- Получение данных сессии
- Просмотр данных сессии
- Заключение

Передача cookie-данных

Данные, указанные пользователем, часто необходимо использовать на нескольких страницах веб-сайта. Пользовательские данные удобно хранить в маленьких cookie-файлах на компьютере клиента (пользователя), чтобы ему не нужно было многократно вводить одни и те же данные. PHP-функция `setcookie()` позволяет легко создавать cookie-записи, указав три аргумента в соответствии со следующим синтаксисом:

```
setcookie(имя, значение, срок-действия);
```

Совет

Функция `setcookie()` также принимает дополнительные необязательные аргументы, позволяющие указать серверный путь, домен, необходимость использовать защищенный протокол HTTPS и доступность только для HTTP-клиентов.

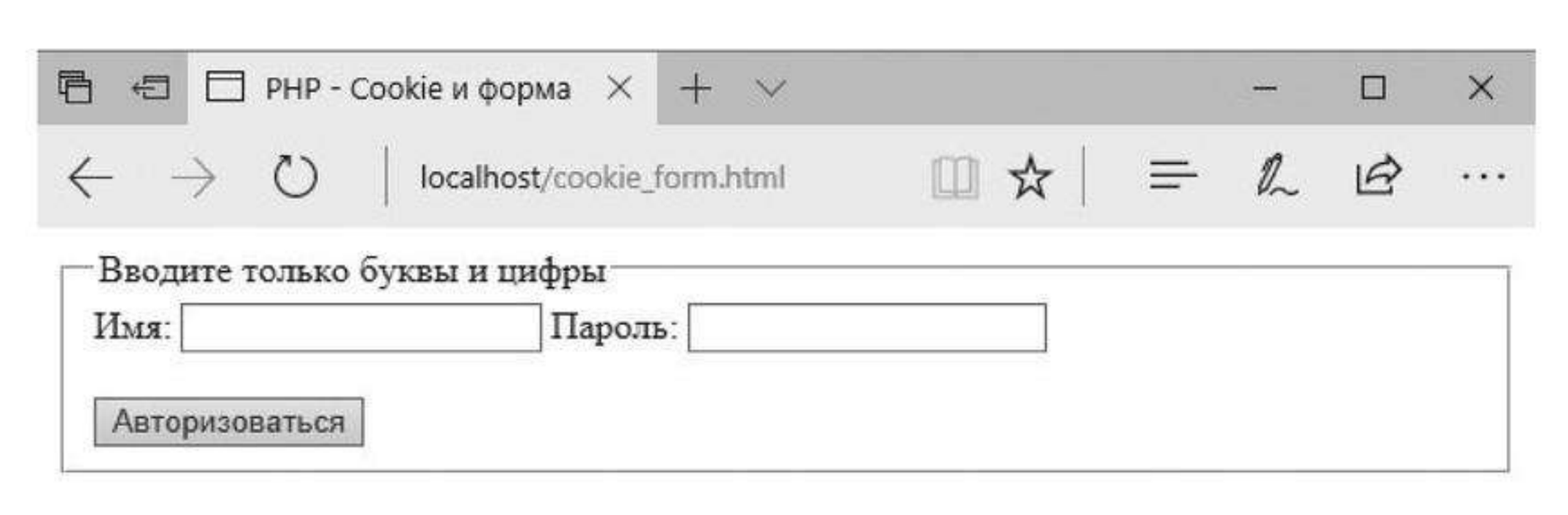
Когда данные HTML-формы передаются на сервер с помощью метода `POST`, то эти данные автоматически назначаются элементам специального глобального массива `$_POST`. Сценарий может проверять наличие данных из определенных полей формы с помощью встроенной в PHP функции `isset()`, позволяющей искать элементы с именами, соответствующими полям HTML-формы. Когда подтверждается, что поле действительно существует, соответствие имя/значение может быть сохранено в виде cookie-записи. Данный подход может быть использован для хранения имени пользователя и пароля и использования на всех страницах веб-сайта.

1. Создайте HTML-документ и добавьте следующую форму в раздел тела страницы, указав метод передачи данных и имя PHP-сценария обработки.

cookie_form.html

```
<form name = "entry" method = "POST"
                  action = "cookie _ set.php">
<fieldset>
<legend>Вводите только буквы и цифры</legend>
Имя: <input type = "text" name = "user" >
Пароль: <input type = "password" name = "pass" >
<br><br><input type = "submit" value = "Авторизоваться" >
</fieldset>
</form>
```

2. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *cookie_form.html*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть веб-форму.



Внимание

Хранение паролей в cookie-файлах — далеко не самый безопасный метод и используется здесь лишь для демонстрации использования cookie-записи для хранения данных.

Установка cookie-записей

Получив переданные данные формы, сценарий может проверить их на наличие значений полей и соответствие значений допустимым форматам. К примеру, вы можете использовать функцию `ctype_alnum()`, позволяющую проверить, содержат ли значения только буквы и цифры.

На заметку

Функция `ctype_alnum()` возвращает `TRUE`, если ее аргумент содержит только буквы и цифры.

После завершения проверки функция `setcookie()` может сохранить имена и значения переданных данных. Кроме того, с помощью функции `md5()` может быть выполнено хэш-преобразование пароля. При успешном завершении сценарий может перенаправить браузер на другую страницу с указанным URL-адресом с помощью функции `header()`.

1. Начните PHP-сценарий с функции для обработки неудачных попыток проверки данных.

cookie_set.php

```
<?php
function reject($entry)
{
 echo "Недопустимо: $entry <br>";
 echo 'Пожалуйста, <a href="cookie _ form.
 html">авторизуйтесь</a>';
 exit();
}
```

2. Затем добавьте блок кода проверки условия, устанавливающий две cookie-записи для действительных данных или вызывающий функцию обработки неудачных попыток проверки данных.

```
if(isset($ _ POST['user']))
{
 $user = trim($ _ POST['user']);
 if(!ctype _ alnum($user)) {reject('Имя пользователя');}

 if(isset($ _ POST['pass']))
 {
  $pass = trim($ _ POST['pass']);
  if(!ctype _ alnum($pass)) {reject('Пароль');}
 else
 {
  setcookie('user', $user, time()+3600);
  setcookie('pass', md5($pass), time()+3600);
  header('Location: cookie _ get.php');
  }
 }
}
else {header('Location: cookie _ form.html');}
?>
```

Совет

Обратите внимание на то, как в примере используется функция `time()` для получения значения текущего времени, а затем к нему добавляется один час (в секундах), чтобы установить срок действия cookie-записей равным одному часу.

Совет

Финальная инструкция `else` выполняет перенаправление на HTML-страницу формы, если непосредственно открыт URL-адрес сценария.

3. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *cookie_set.php*, а затем переходите к следующей странице, чтобы узнать, как извлечь сохраненные данные из cookie-файла.

Получение cookie-записей

После того как cookie-записи установлены, они автоматически присваиваются специальному глобальному массиву `$_COOKIE`. Сценарий может проверить наличие определенной cookie-записи с помощью встроенной PHP-функции `isset()`, позволяющей искать cookie-записи с указанным именем. Если наличие cookie-записи подтверждено, то ее значение может быть присвоено обычной переменной в сценарии. Данный прием может использоваться для извлечения сохраненного имени пользователя и вывода его на странице. Если же искомая cookie-запись отсутствует, то сценарий может выполнить альтернативное действие.

1. Начните PHP-сценарий с кода поиска cookie-записи и извлечения данных для вывода, если запись обнаружена.

cookie_get.php

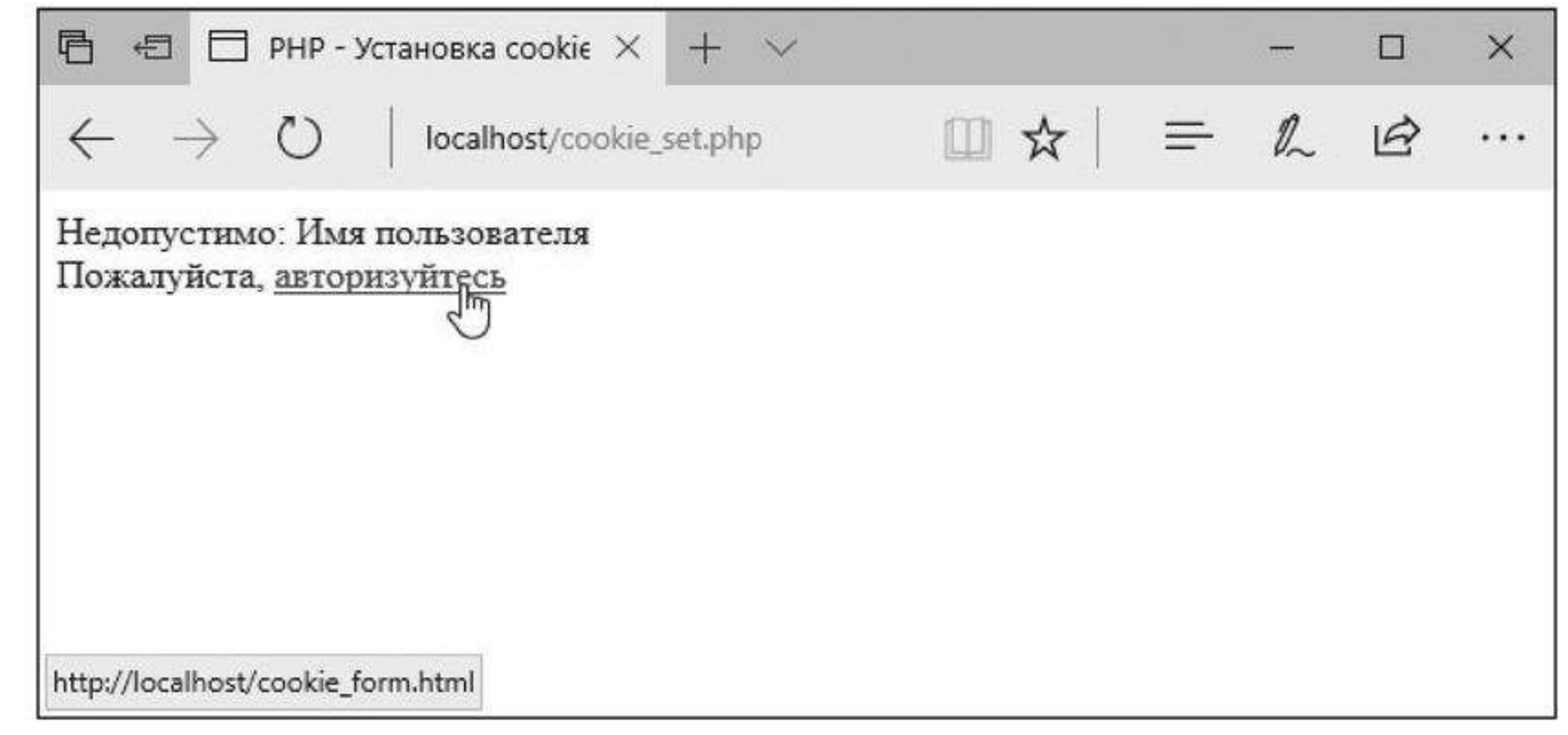
```
<?php if(isset($ _ COOKIE['user']))
{
 $user = $ _ COOKIE['user'];
 echo "<h1>Добро пожаловать, $user!</h1><hr>";
 echo '<a href="cookie _ data.php">Просмотр cookie</a>';
}
```

2. Добавьте альтернативный вывод на случай, если cookie-запись не обнаружена.

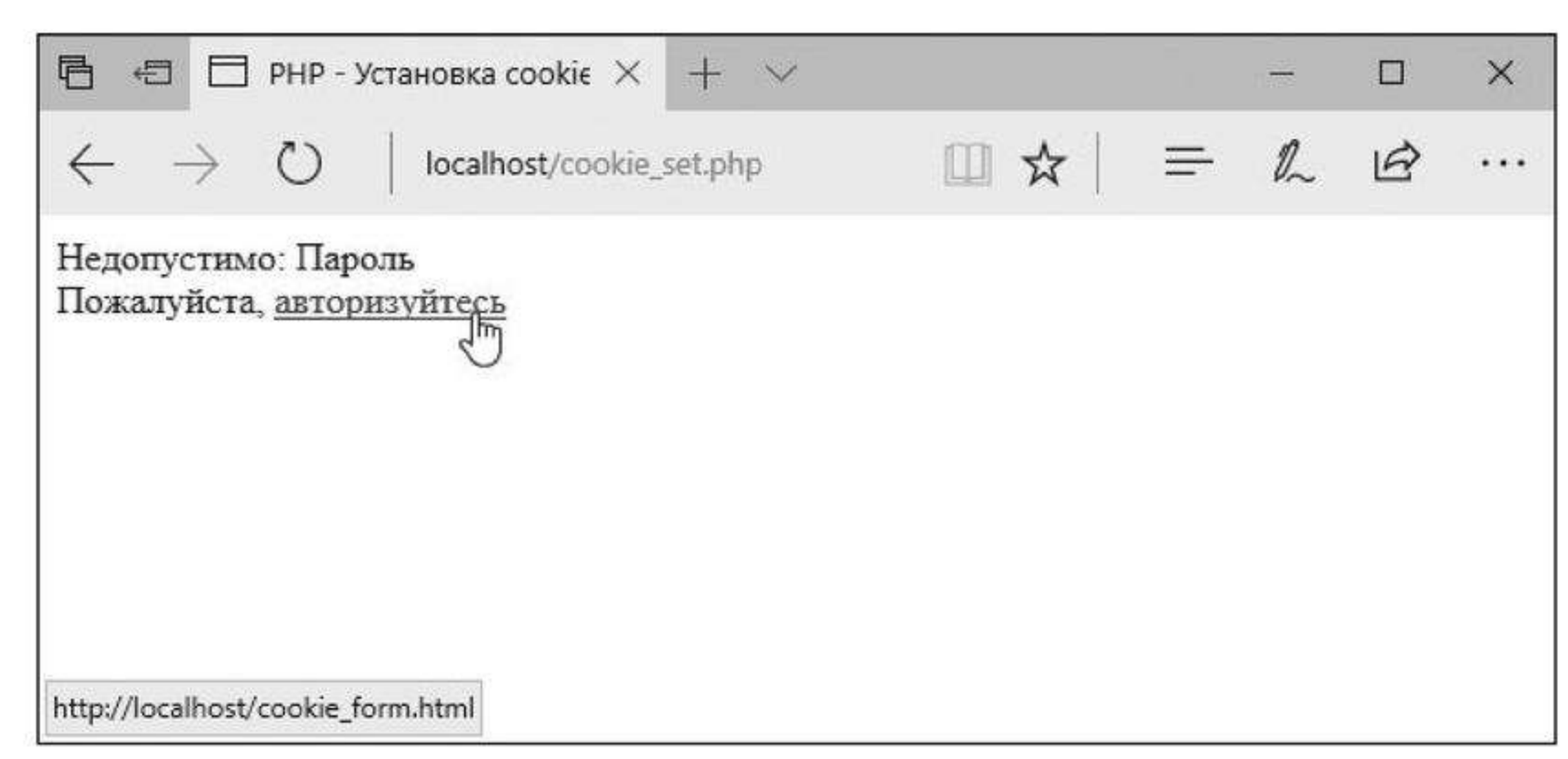
```
else
{
 echo 'Пожалуйста, <a href="cookie _ form.html">
 авторизуйтесь</a>';
}
?>
```

3. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *cookie_get.php*.

4. Теперь откройте веб-форму из файла *cookie_form.html* (созданную ранее в этой главе) через протокол HTTP и нажмите кнопку «Авторизоваться», чтобы увидеть сообщение о неудачной попытке авторизации, так как необходимые cookie-записи не установлены.



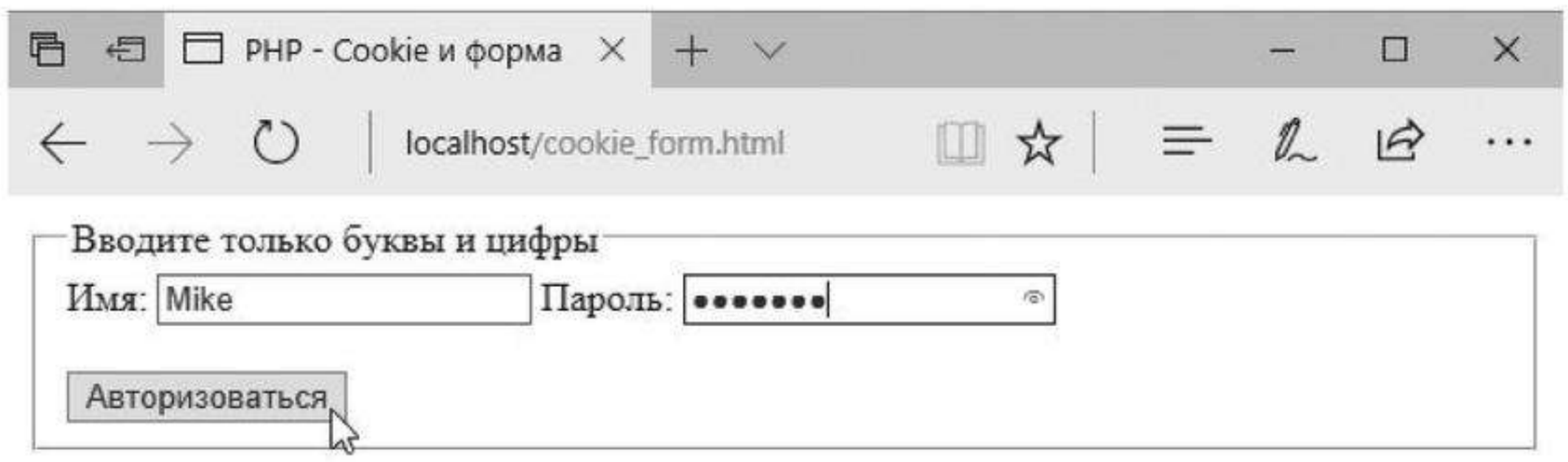
5. Щелкните мышью по ссылке «авторизуйтесь», чтобы вернуться к HTML-форме, и введите имя пользователя, а затем нажмите кнопку «Авторизоваться», чтобы увидеть, что проверка опять провалена.



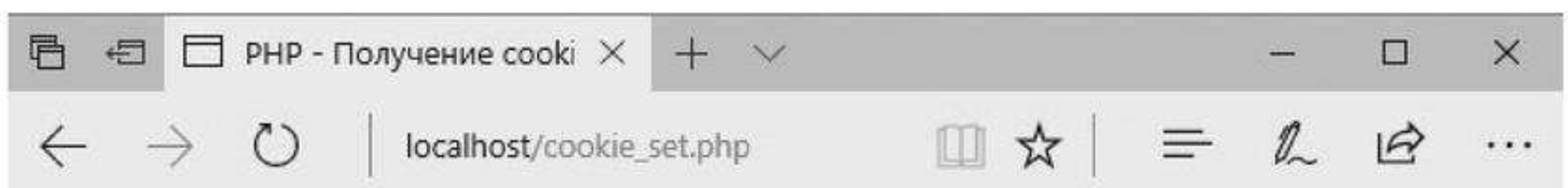
На заметку

Вы также можете перейти к HTML-форме и ввести имя пользователя или пароль, который включает в себя нелатинские буквы (например, кириллицу) или специальные символы. При авторизации вы вновь потерпите неудачу.

6 Вернитесь к HTML-форме еще раз и введите допустимые имя пользователя и пароль. Нажмите кнопку «Авторизоваться», чтобы увидеть сообщение об успешной авторизации.



http://localhost/cookie_set.php



Добро пожаловать, Mike !

Просмотр cookie

Совет

Параметры регистрационных данных можно расширить. К примеру, можно допускать символы подчеркивания либо ограничить минимальную допустимую длину пароля.

7 На следующей странице вы научитесь просматривать данные, хранящиеся в cookie-файлах, используя ссылку на странице *cookie_data.php* из этого примера.

Просмотр cookie-записей

Специальный глобальный массив `$_COOKIE` хранит имена и значения cookie-записей в виде ассоциативного массива ключей/значений. Сохраненные данные можно просматривать посредством цикличного обхода массива, чтобы увидеть все имена и значения, либо с помощью PHP-функции `var_dump()`.

1. Начните PHP-сценарий с кода, позволяющего обнаружить, существуют ли какие-либо cookie-записи. В случае успеха извлеките все найденные имена и значения.

cookie_data.php

```
<?php
if(count($_COOKIE) > 0)
{
 echo '<dl> ';
 foreach($_COOKIE as $key => $value)
 {
  echo "<dt>Ключ: $key";
  echo "<dd>Значение: $value";
 }
 echo '</dl><hr>';
 var_dump($_COOKIE);
}
```

2. Затем добавьте альтернативный вывод, если cookie-записи отсутствуют.

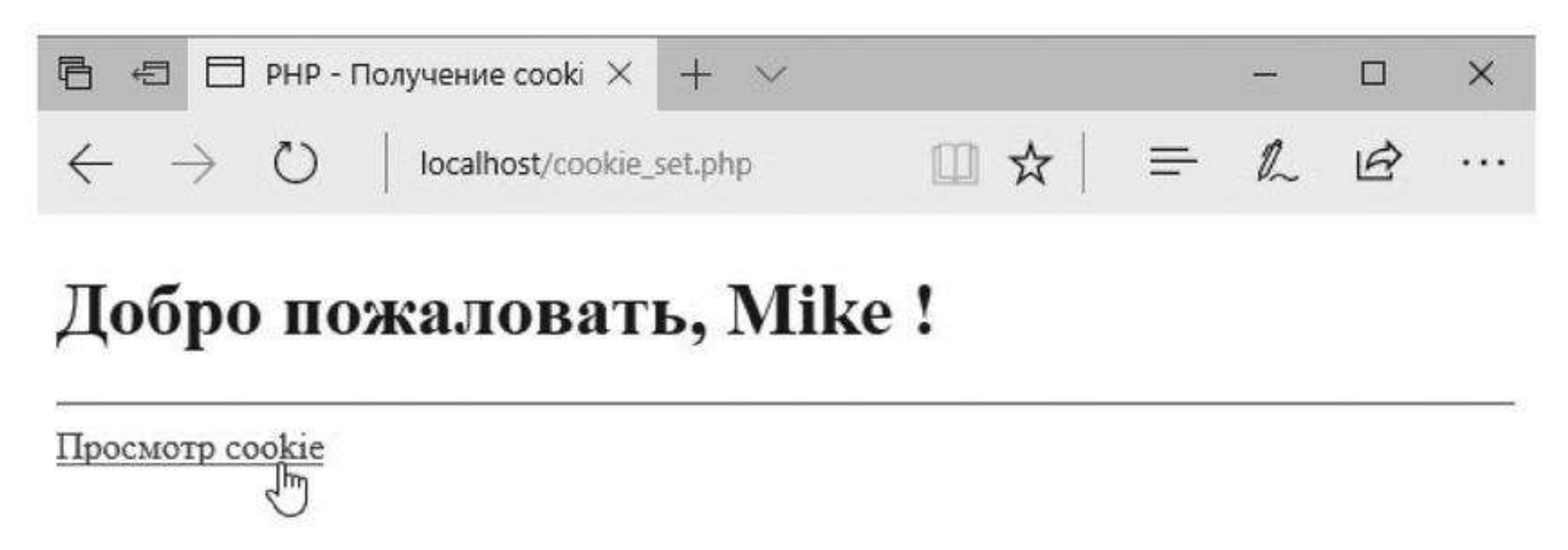
```
else
{
 echo 'Пожалуйста, <a href="cookie_form.
 html">авторизуйтесь</a>';
}
?>
```

3. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *cookie_data.php*.

Совет

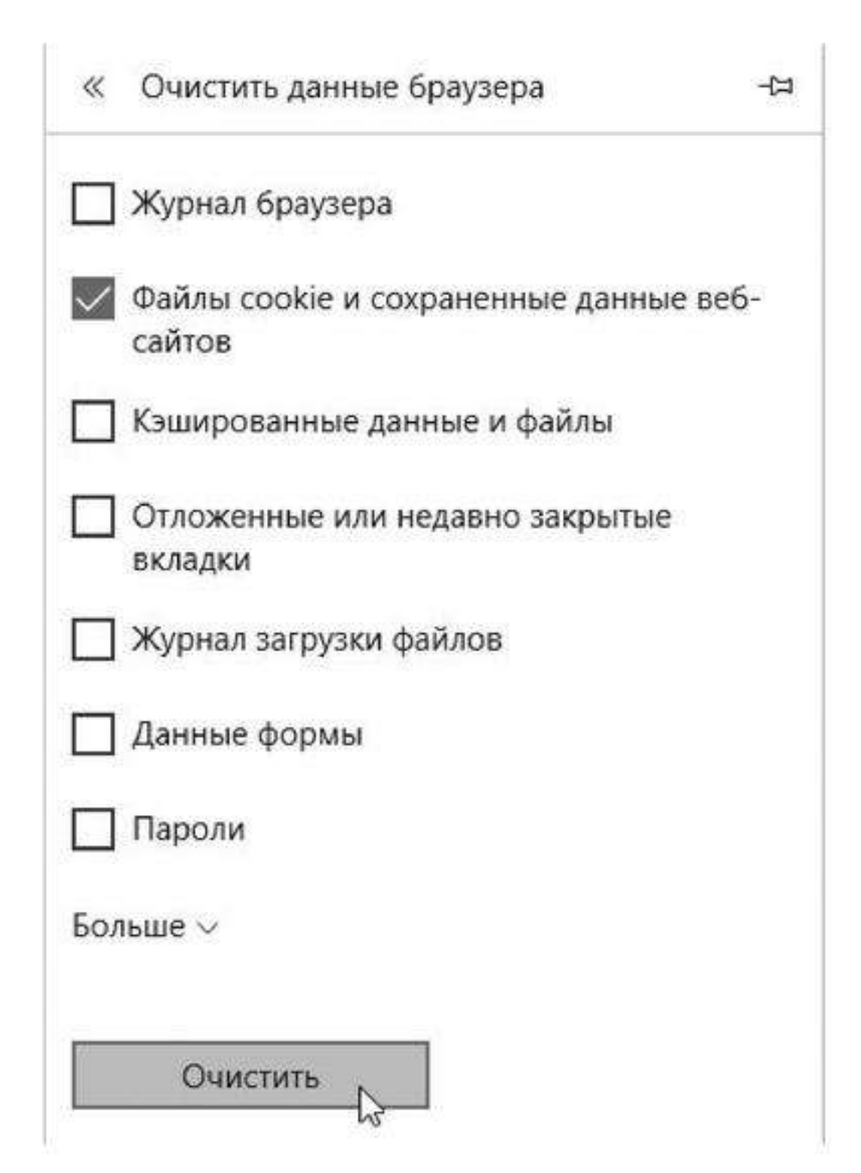
Чтобы создать новую или перезаписать существующую cookie-запись, вы можете вызвать функцию `setcookie()` в любой момент.

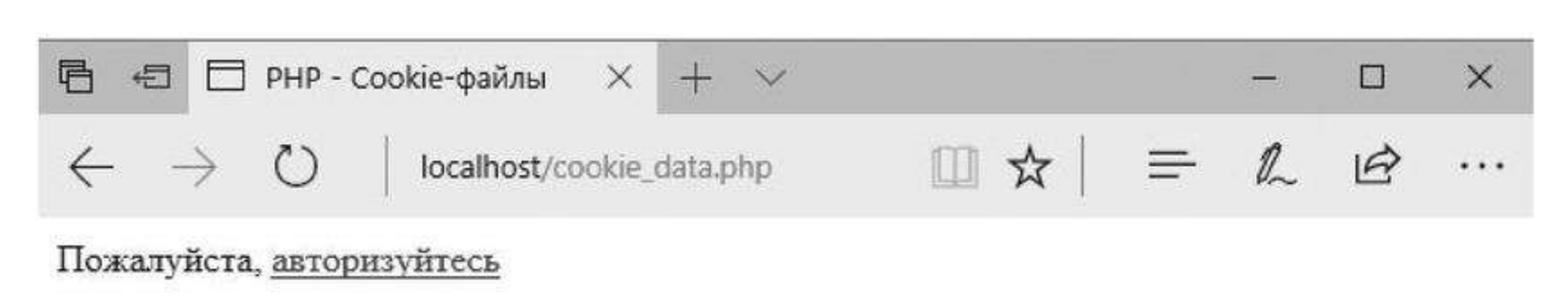
4 Теперь откройте веб-форму в файле *cookie_form.html* (см. ранее в этой главе) через протокол HTTP и введите допустимые данные для авторизации. Затем перейдите по ссылке, расположенной на странице *cookie_get.php* (см. ранее в этой главе), чтобы просмотреть cookie-записи.



5 Откройте меню настроек в вашем веб-браузере и удалите существующие cookie-записи, хранящиеся в вашей системе.

6 Теперь нажмите кнопку **Обновить** (Refresh) в вашем браузере, чтобы перезагрузить страницу и увидеть альтернативный контент — подтверждение того, что cookie-записи были удалены.





Совет

Совет. В руководстве по языку PHP описывается прием удаления cookie-записей путем установки пустого значения и истекшего срока действия, например `setcookie($_COOKIE['user'], '', time()-3600)`. Кроме того, для уничтожения определенной может использоваться функция `unset()`.

Передача данных сессии

Пользовательские данные могут быть сохранены в cookie-файлы на компьютере клиента, только если пользователь разрешил использование cookie-файлов на своем компьютере. Следует учесть, что многие пользователи запрещают запись cookie-файлов, чтобы избежать отслеживания. Для обхода этой проблемы PHP имеет альтернативу. С помощью PHP пользовательские данные могут храниться на сервере, в специальном глобальном массиве `$_SESSION`. Для обеспечения работоспособности сессий в PHP необходимо, чтобы каждая PHP-страница вызывала встроенную функцию `session_start()` в начале кода страницы, прежде любых HTML-элементов. После передачи данных веб-формы на сервер методом `POST` они присваиваются элементам специального глобального массива `$_POST`. Вы можете проверять данные отдельных полей формы при помощи встроенной функции `isset()`. Если обнаружено, что искомые данные присутствуют, то соответствующее имя и значение может быть сохранено в переменной сессии. Данный способ как альтернатива cookie-файлам позволяет хранить, к примеру, имя пользователя и пароль и использовать эти данные на всех страницах веб-сайта.

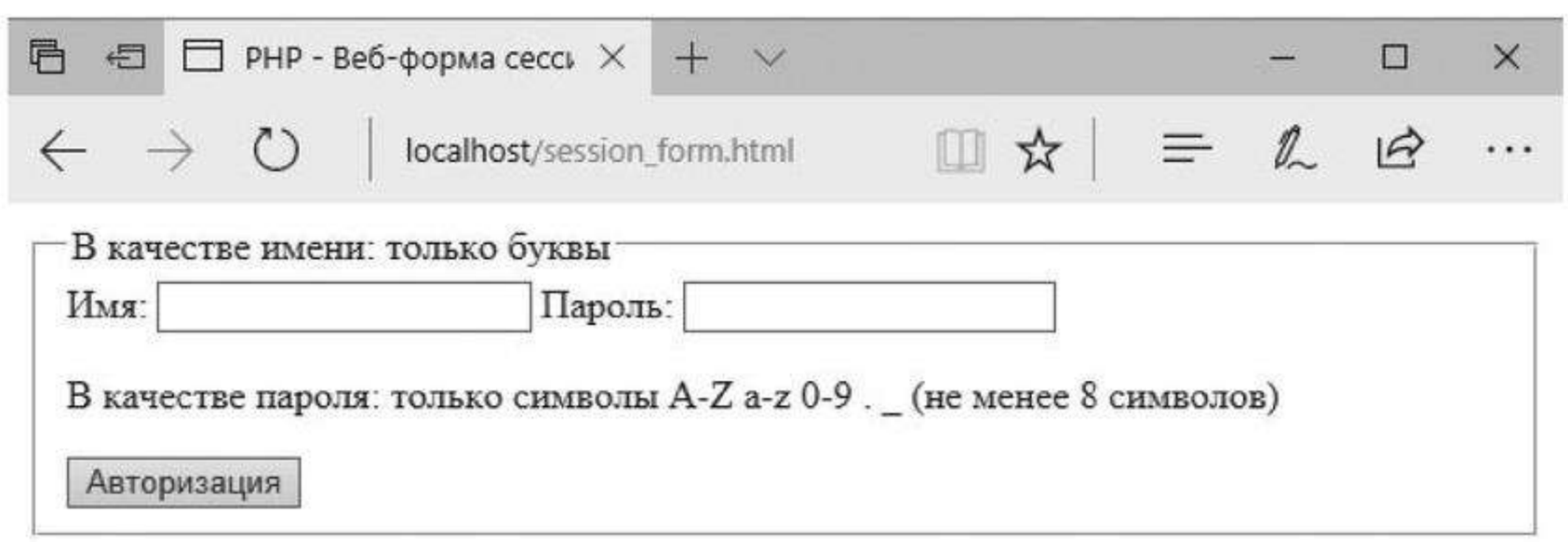
Переменные сессии доступны для PHP-сценария, только если в начале кода страницы была вызвана функция `session_start()`.

1. Создайте HTML-документ и добавить указанный ниже код веб-формы в раздел тела страницы. Укажите метод передачи данных и имя PHP-сценария, который будет их обрабатывать.

session_form.html

```
<form name = "entry" method = "POST"
                              action = "session _ set.php">
<fieldset>
<legend>В качестве имени: только буквы</legend>
Имя: <input type="text" name="user" >
Пароль: <input type="password" name="pass" >
<p>В качестве пароля: только символы A-Z a-z 0-9. _
(не менее 8 символов)</p> <input type="submit"
value="Авторизация">
</fieldset>
</form>
```

2. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *session_form.html*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат выполнения примера.



Настройка сессий

Получив отправленные пользователем данные, PHP-сценарий может проверить заполнение необходимых полей и допустимость переданных в них значений. Например, функция `ctype_alpha()` используется для проверки, что значения содержат только буквы, а функция `preg_match()` проверяет соответствие значения определенному шаблону.

После завершения проверки переданные имена и значения могут быть назначены элементам глобального массива `$_SESSION`. При успешном завершении сценарий с помощью функции `header()` перенаправляет браузер на следующую страницу с указанным URL-адресом.

На заметку

Функция `ctype_alpha()` возвращает значение `TRUE` только в том случае, если ее аргумент содержит буквы.

1. Создайте начальный PHP-сценарий, разрешающий работу с сессиями.

```
<?php session_start();?>
```

2. Начните основной PHP-сценарий с кода оповещения о неудачной попытке проверки данных.

```
<?php function reject($entry)
{
 echo "Недопустимо: $entry <br>";
 echo 'Пожалуйста, <a href="session_form.
 html">авторизуйтесь</a>';
 exit();
}
```

3. Добавьте код проверки условий, создающий две cookie-записи, если данные прошли проверку успешно, или вызывающий функцию обработки неудачных попыток авторизации.

```
if(isset($_POST['user']))
{
 $user = trim($_POST['user']);
 if(!ctype_alpha($user)) {reject('Имя пользователя');}
 if(isset($_POST['pass']))
 {
  $pass = trim($_POST['pass']);
  if(!preg_match('/^[A-Za-z0-9._]{8,}$/', $pass))
  {reject('Пароль');}
 else
 {
  $_SESSION['user'] = $user;
  $_SESSION['pass'] = $pass;
  header('Location: session_get.php');
 }
 }
} else {header('Location: session_form.html');}
?>
```

Совет

Совет. Функция `preg_match()` принимает регулярное выражение и строковый аргумент. Тема регулярных выражений выходит за рамки этой книги, поэтому вам нужно в точности скопировать пример кода, проверяющего допустимость формата введенного пароля. В этом примере пароль не шифруется, чтобы вы могли увидеть его соответствие указанным требованиям.

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *session_set.php*, а затем перейдите к следующей странице, чтобы научиться извлекать данные сессий.

Получение данных сессии

К данным сессии можно получить доступ из специального глобального массива `$_SESSION` после вызова функции `session_start()`. PHP-функция `isset()` позволяет запросить элемент с заданным именем.

Если установлено, что элемент существует, то его значение может быть присвоено обычной переменной сценария. Этот прием может быть использован для извлечения сохраненного имени пользователя и вывода на странице. Если искомый элемент отсутствует, то сценарий может выполнить альтернативное действие.

1. Создайте начальный PHP-сценарий, разрешающий работу с сессиями.

```
<?php session_start();?>
```

session_get.php

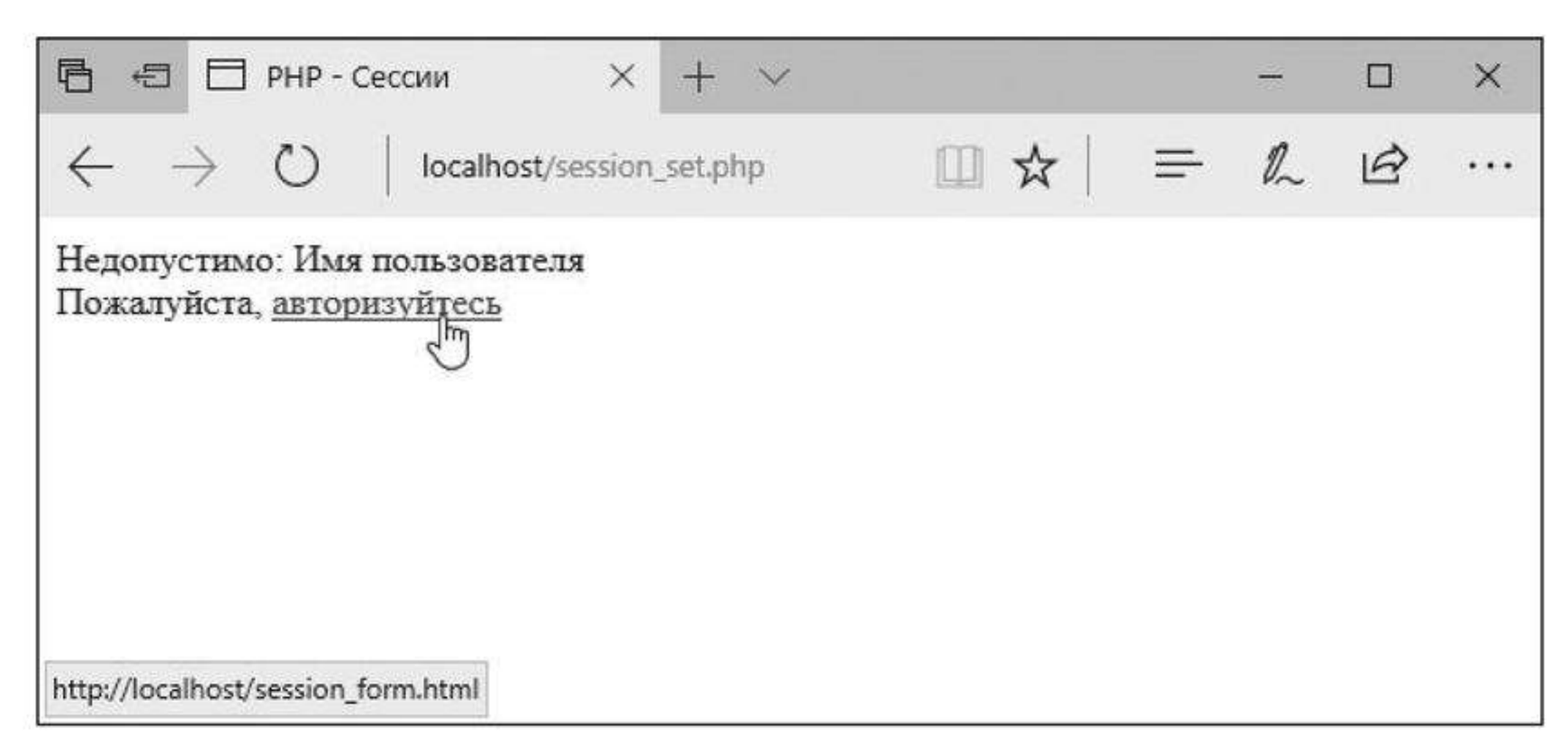
2. Начните основной PHP-сценарий с поиска переменной сессии и извлечения данных для вывода в случае, если переменная обнаружена, или выполнения альтернативного кода, если переменная сессии отсутствует.

```
<?php
if(isset($_SESSION['user']))
{
 $user = $_SESSION['user'];
 echo "<h1>Добро пожаловать, $user!</h1><hr>";
 echo '<a href="session_data.php">Просмотр сессии</a>';
}
```

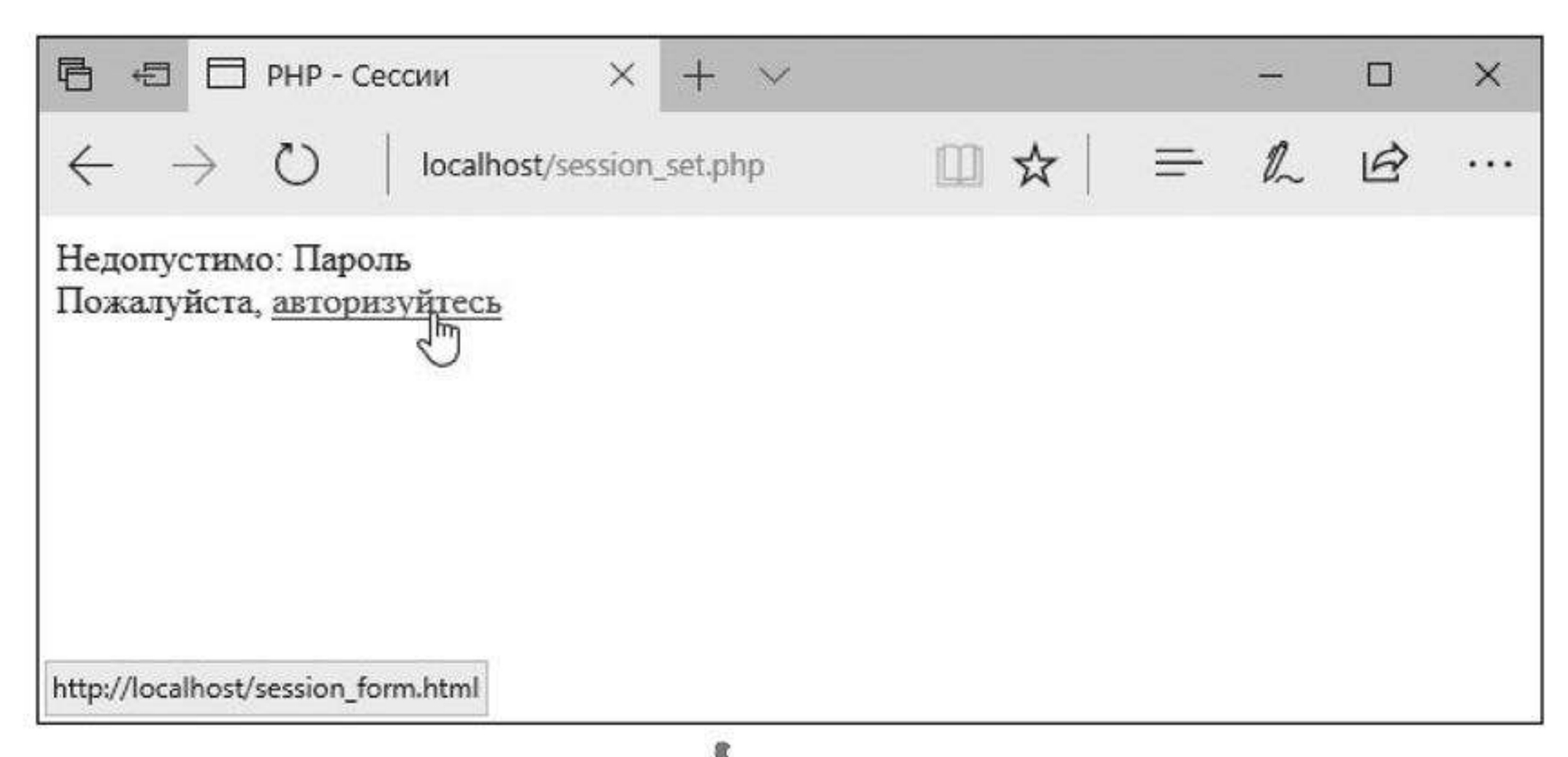
```
else
{echo 'Пожалуйста, <a href="session _ form.
html">авторизуйтесь</a>';}
?>
```

3. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *session_get.php*.

4. Теперь откройте HTML-файл *session_form.html* с веб-формой (созданный ранее в этой главе) через протокол HTTP и нажмите кнопку «Авторизация», чтобы увидеть сообщение о неудачной попытке авторизации.



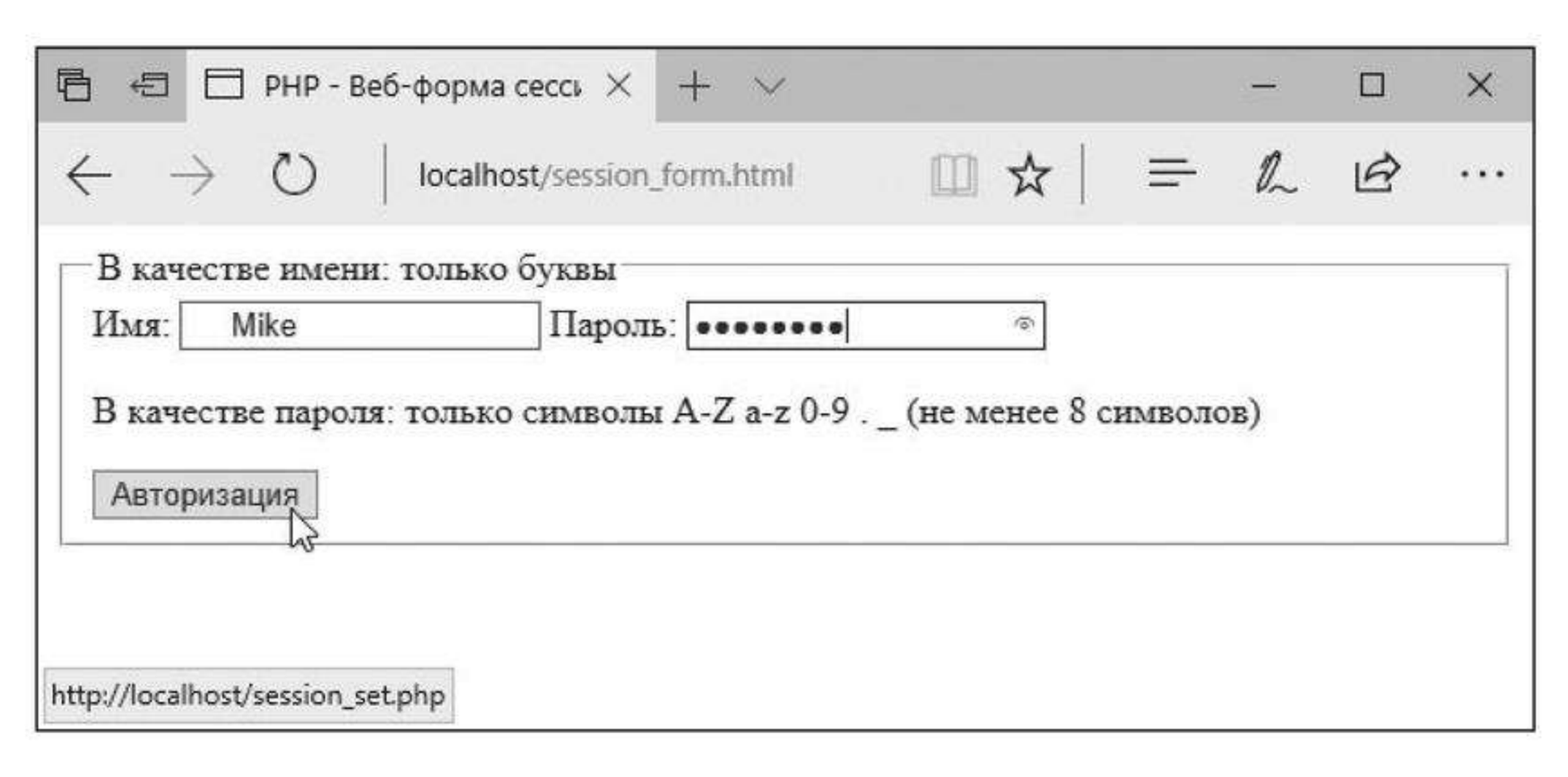
5. Щелкните мышью по ссылке «авторизуйтесь», чтобы вернуться к HTML-форме, и введите имя пользователя. Затем нажмите кнопку «Авторизация», чтобы увидеть, что попытка проверки опять провалена.

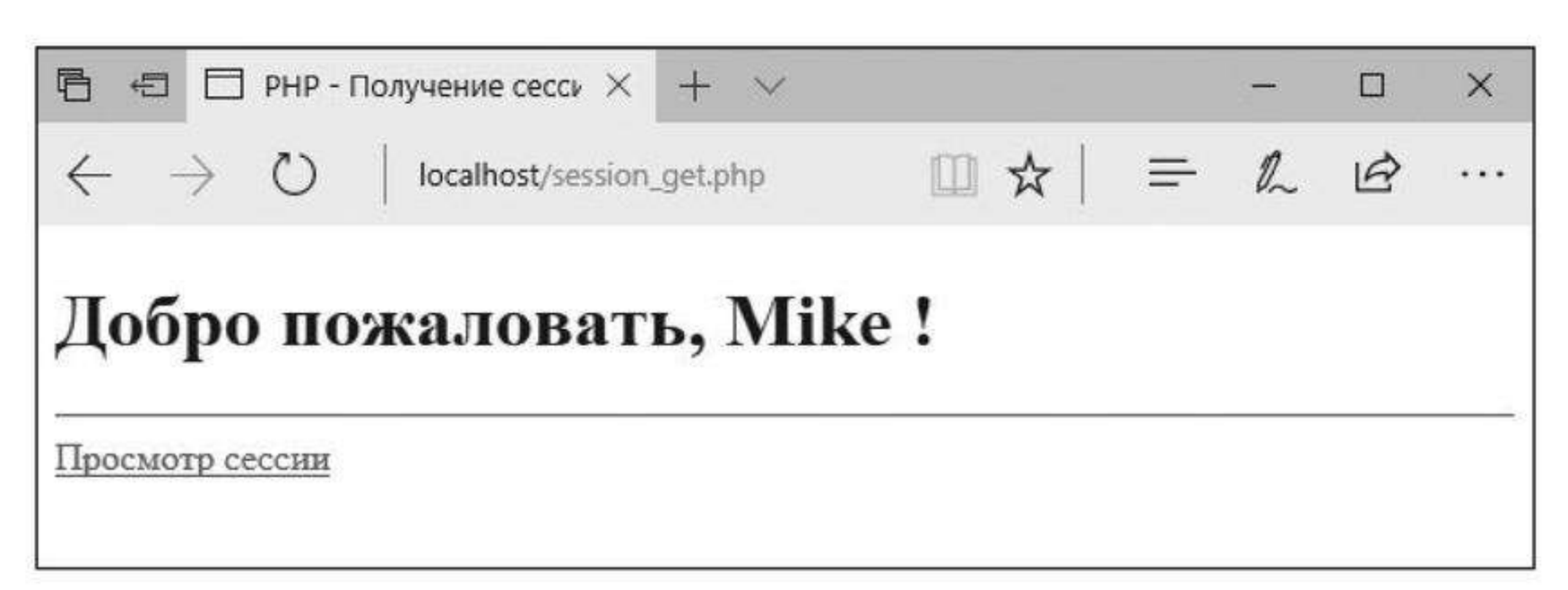


Совет

Вы также можете перейти к HTML-форме и ввести пароль, содержащий меньше необходимого числа символов или включающий в себя недопустимые символы. При авторизации вы вновь потерпите неудачу.

6 Вернитесь к HTML-форме еще раз и введите допустимые имя пользователя и пароль. Нажмите кнопку «Авторизация», чтобы увидеть сообщение об успешной авторизации.





На заметку

На заметку. Обратите внимание на то, что в этом примере перед именем пользователя добавлены пробелы. Они не вызовут проблем, так как будут удалены с помощью функции `trim()`, которая используется в сценарии *session_set.php*.

7 На следующей странице вы узнаете, как просматривать данные, хранящиеся в переменных сессии, используя ссылку на странице из файла *session_data.php*.

Просмотр данных сессии

Специальный глобальный массив `$_SESSION` хранит имена и значения cookie-записей в виде ассоциативного массива ключей/значений. Сохраненные данные можно просматривать посредством цикличного обхода массива, чтобы увидеть все имена и значения, либо с помощью PHP-функции `var_dump()`. Кроме того, каждая сессия имеет уникальный идентификационный номер, который можно увидеть с помощью PHP-функции `session_id()`.

На заметку

Уникальность каждой сессии позволяет переменным существовать без конфликтов.

Установленные переменные сессии могут быть удалены путем указания имени их элемента в PHP-функции `unset()`, а полностью сессия может быть прекращена путем вызова функции `session_destroy()`:

1. Создайте начальный PHP-сценарий, разрешающий работу с сессиями.

```
<?php session_start();?>
```

session_data.php

2. Начните основной PHP-сценарий с инструкций, удаляющих переменные сессии, завершающих сессию и подтверждающих завершение сессии.

```
<?php
function kill_session()
{
 unset($_SESSION['user']);
 unset($_SESSION['pass']);
 session_destroy();
 echo '<hr>Сессия прервана<br>';
 echo 'Идентификатор сессии: '.session_id().'<br>';
 var_dump($_SESSION);
}
```

Совет

Вы можете в любое время присвоить новое значение переменной сессии для замещения существующего сохраненного значения.

3. Теперь добавьте код, обнаруживающий установленные переменные сессии и извлекающий все сохраненные имена и значения.

```
if(count($_SESSION) > 0)
{
 echo '<dl> ';
 foreach($_SESSION as $key => $value)
 {
  echo "<dt>Ключ: $key"; echo "<dd>Значение: $value";
 }
 echo '</dl><hr>';
 // Сюда вставляются инструкции (шаг 4).
}
```

4. Добавьте инструкции для вывода идентификатора сессии и массива, а затем вызовите функцию, завершающую сессию.

```
echo 'Идентификатор сессии: '.session_id().'<br>';
var_dump($_SESSION);
kill_session();
```

5. Добавьте альтернативный вывод на случай, если сессия не обнаружена.

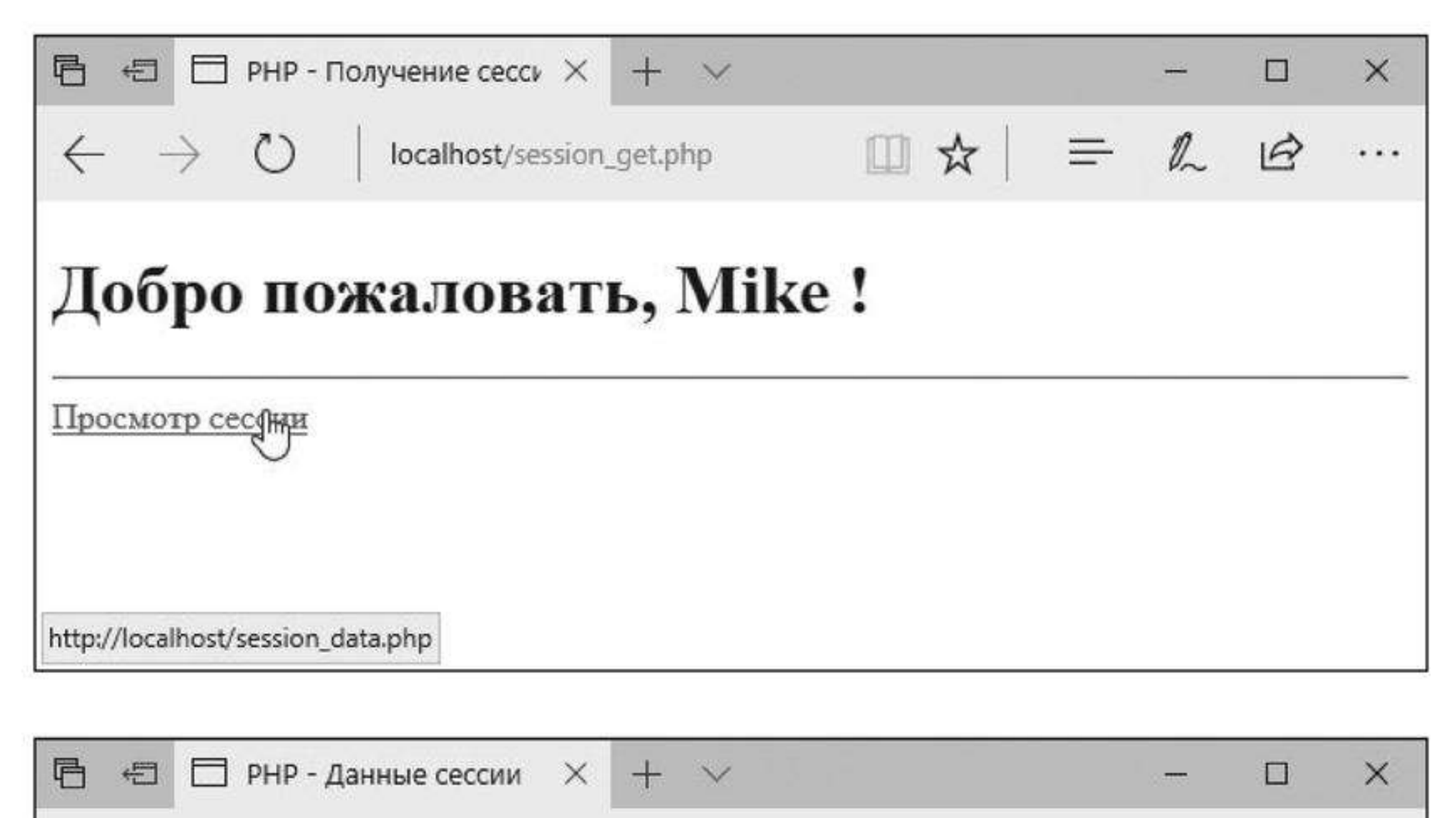
```
else
{echo 'Пожалуйста, <a href="session_form.
html">авторизуйтесь</a>';}
?>
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *session_data.php*.

7. Теперь откройте веб-форму, сохраненную в файле *session_form.html* (см. ранее в этой главе), через протокол HTTP и введите допустимые данные для авторизации. Затем перейдите по ссылке, расположенной на странице *session_get.php* (см. ранее в этой главе), чтобы просмотреть сведения о сессии.

На заметку

Обратите внимание на то, что пароль соответствует указанному шаблону.



8 Теперь нажмите кнопку **Обновить** (Refresh) в вашем браузере, чтобы перезагрузить страницу и увидеть новое содержимое — сообщение «Пожалуйста, авторизуйтесь».

Заключение

- PHP-функция `set_cookie()` может использоваться для хранения данных пользователя на компьютере клиента и их применения на различных страницах веб-сайта.
- Функции `set_cookie()` требуется передать по крайней мере три аргумента, определяющих имя, значение и срок действия.
- Данные веб-формы, переданные с помощью метода `POST`, автоматически присваиваются элементам глобального массива `$_POST`.
- Функция `isset()` позволяет искать внутри глобального массива `$_POST` элементы с именами, соответствующими именам полей формы.

- Функция `ctype_alnum()` проверяет, что значение содержит только буквы и цифры.
- Хэш-преобразование шифрует открытые пароли с помощью функции `md5()`.
- PHP-функция `header()` определяет расположение контента с указанием URL-адреса, чтобы перенаправить браузер на нужную страницу.
- После установки cookie-записи автоматически присваиваются элементам специального глобального массива `$_COOKIE`.
- Пользовательские данные могут быть сохранены на сервере в глобальном массиве `$_SESSION` как альтернатива способу хранения данных в cookie-файлах.
- Для использования сессий каждая PHP-страница должна содержать вызов функции `session_start()` в самом начале кода страницы перед любыми HTML-тегами.
- Данные формы, переданные в глобальный массив `$_POST`, могут быть присвоены элементам глобального массива `$_SESSION`.
- Функция `ctype_alpha()` проверяет, что значение содержит только буквы.
- Функция `preg_match()` используется для проверки соответствия значения указанному шаблону.
- Каждая сессия имеет уникальный идентификатор, который можно получить с помощью PHP-функции `session_id()`.
- Переменные сессии могут быть удалены с помощью функции `unset()`, а сама сессия завершена с помощью функции `session_destroy()`.
- Сохраненный контент можно просмотреть с помощью функции `var_dump()` и посредством циклического обхода массивов `$_COOKIE` или `$_SESSION`.

11

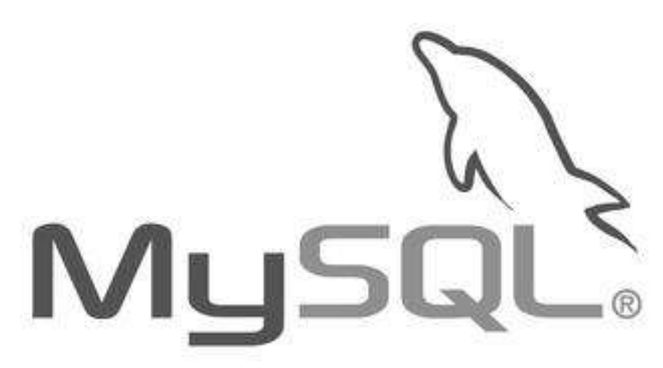
Подключение баз данных

В этой главе вы узнаете, как установить и настроить простой форум на основе PHP и базы данных MySQL.

- Подключение компонентов
- Создание форума
- Проверка работоспособности
- Обработка сообщений
- Предоставление формы
- Страницы форума
- Заключение

Подключение компонентов

Подключение к базе данных MySQL осуществляется с помощью обычного PHP-сценария. В сценарии определяется переменная с именем `dbc` (сокращение от database connection — соединение с базой данных), которой присваивается значение, возвращенное после вызова встроенной функции `mysqli_connect()`. Этому вызову требуется четыре аргумента, с помощью которых указываются хост-сервер, имя пользователя MySQL и его пароль, а также база данных, к которой осуществляется подключение. Если вызов выполнен успешно, функция `mysqli_connect()` возвращает «объект», представляющий собой подключение к серверу MySQL. Сценарий выглядит следующим образом:



Выпуск MySQL Community Edition доступен для бесплатной загрузки на сайте dev.mysql.com/downloads/

```
<?php

$dbc =
mysqli _ connect('сервер', 'пользователь',
'пароль', 'база _ данных')
OR die (mysqli _ connect _ error());
mysqli _ set _ charset($dbc, 'utf-8');
```

connect_db.php

Объект, возвращаемый успешно вызванной функцией `mysqli_connect()`, очень важен, так как используется в качестве аргумента в вызовах других PHP-функций, которые обмениваются данными с сервером MySQL. К примеру, в приведенном выше сценарии он используется в качестве первого аргумента вызываемой функции `mysqli_set_charset()`, позволяющей идентифицировать подключение.

На заметку

В целях этой главы предполагается, что база данных MySQL установлена и настроена для использования на локальном сервере, с полным привилегированным доступом к базе данных с именем `website_db`.

Если вызов функции `mysqli_connect()` завершился неудачей, сценарий выполняет альтернативную ветвь кода и вызывает функцию `die()`. Она эквивалентна функции `exit()`, немедленно завершающей работу сценария.

Опционально функции `die()` можно передать строковый аргумент, значение которого будет выведено пользователю при завершении работы сценария. В нашем примере текстовое сообщение возвращается с помощью функции `mysqli_connect_error()`, которая описывает, почему попытка подключения не удалась.

Так как сценарий, отвечающий за подключение к базе данных, содержит конфиденциальные данные пользователя, он не должен помещаться рядом с другими PHP-сценариями в каталоге */htdocs* вашего веб-сервера. Вместо этого сценарий *connect_db.php* следует поместить в родительский каталог */htdocs*, где он не будет непосредственно доступен. При этом возможность встраивания в другие PHP-сценарии сохраняется и осуществляется с помощью инструкции `require`, например так:

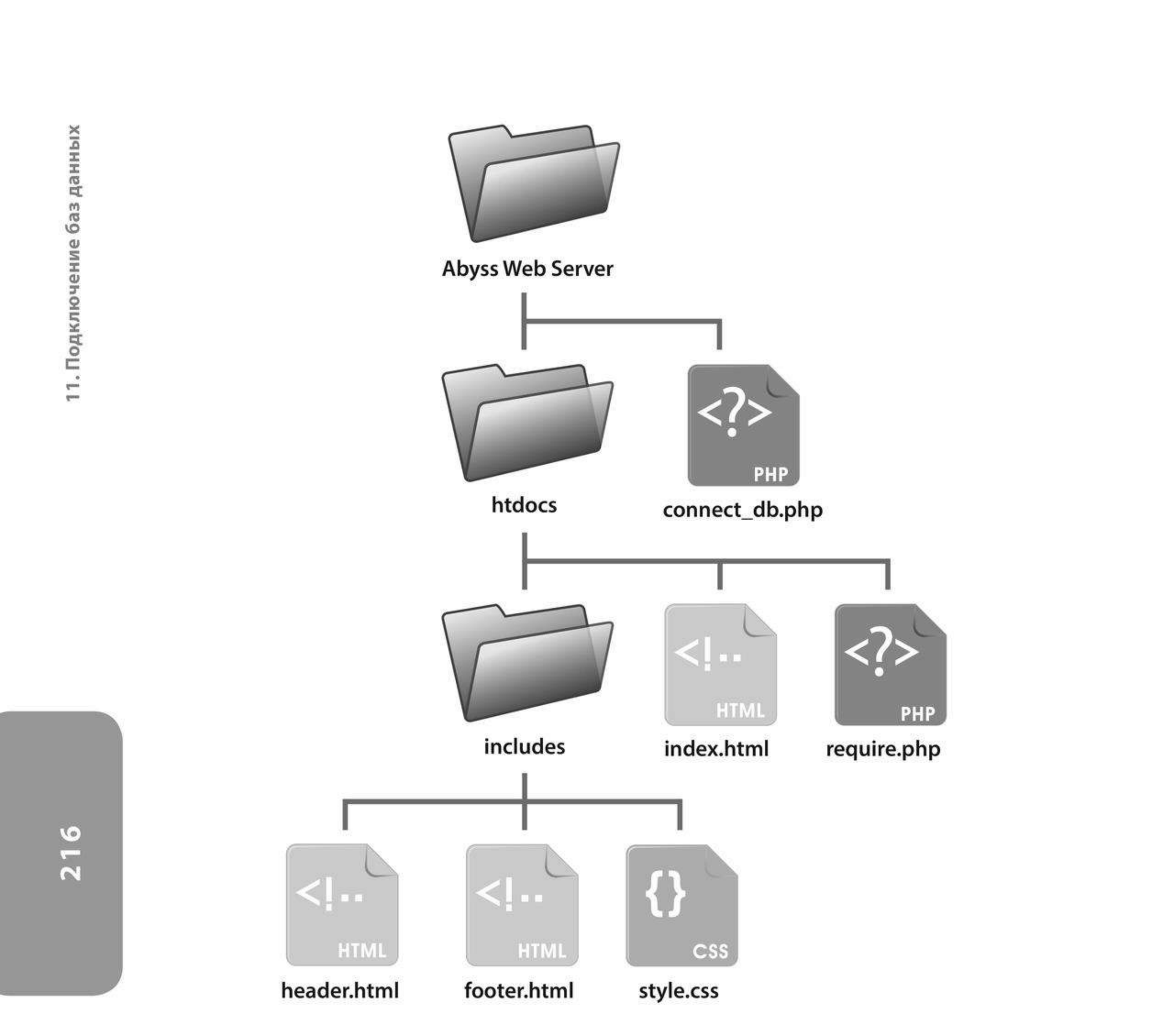
require.php

```
<?php

require ('../connect _ db.php');

if (mysqli _ ping($dbc))
{
 echo 'Сервер MySQL '. mysqli _ get _ server _ info($dbc).
                 ' на '. mysqli _ get _ host _ info($dbc);
}
```

Следуя этой рекомендации, файлы в структуре каталогов вашего веб-сервера должны быть организованы так, как показано ниже. Поэтому, если вы запрашиваете сценарий *require.php* через протокол HTTP, сценарий *connect_db.php* выполняется из его расположения в родительском каталоге и осуществляет подключение к серверу MySQL.

На заметку

В файлах, которые содержат только PHP-код, например описанных в этом примере, следует опустить закрывающий PHP-тег ?> в конце файла. Это предотвращает появление после закрывающего PHP-тега случайных пробельных символов или пустых строк, которые могут привести к нежелательным последствиям.



Внимание

Сценарий ***connect_db.php*** должен успешно подключаться к базе данных MySQL для выполнения примеров из этой главы.

Создание форума

Пользовательские сообщения (посты) могут быть сохранены в таблице базы данных MySQL с помощью PHP-сценария, чтобы впоследствии их можно было извлечь для отображения на странице форума, содержащей сообщения различных зарегистрированных пользователей. Как правило, таблица базы данных, содержащая опубликованные пользовательские сообщения, хранит идентификатор сообщения, имя и фамилию пользователя, название темы, текст сообщения, а также дату публикации сообщения.

Внимание

Таблицы в базах данных MySQL может создавать только администратор или пользователь MySQL, которому предоставлен полный набор привилегий доступа.

Имя столбца	Тип содержимого	Пример
`post_id`	Числовое	123
`first_name`	Текстовое	Михаил
`last_name`	Текстовое	Райтман
`subject`	Текстовое	Мороженое
`message`	Текстовое	Очень вкусно в жару.
`post_date`	Дата/Время	2016–07–24 12:30:00

Таблица базы данных по данной схеме создается с помощью языка SQL (Structured Query Language — язык структурированных запросов). Схема этой таблицы базы данных предполагает, что для столбцов, хранящих текстовый контент, должны быть определены типы данных `VARCHAR` или `TEXT`. Идентификаторы сообщений (первый столбец) должны быть представлены положительным целочисленным значением `INT`, которое таблица автоматически назначает с помощью SQL-атрибута `AUTO_INCREMENT`. Дата и время должны быть сохранены в качестве SQL-значения `DATETIME`. Все эти элементы требуются для внесения сообщения в таблицу базы данных, поэтому к каждому столбцу необходимо применить правило `NOT NULL`.

Совет

Подробнее изучить и научиться использовать язык структурированных запросов с базами данных можно из книги под названием «SQL in easy steps».

Команды SQL-запросов, необходимые для создания показанной выше структуры с таблицей `forum` базы данных MySQL, выполняются в оболочке MySQL Command Line Client или PHP-сценарием в веб-браузере.

1. Начните PHP-сценарий с инструкции подключения к базе данных. Код стандартного сценария подключения показан в начале этой главы.

create_forum.php

```
<?php

require('..\connect _ db.php');
```

2. Далее назначьте переменной характеристики столбцов таблицы.

```
$sql = 'CREATE TABLE IF NOT EXISTS forum ('.
'post _ id INT UNSIGNED NOT NULL AUTO _ INCREMENT,'.
'first _ name VARCHAR(20) NOT NULL,'.
'last _ name VARCHAR(40) NOT NULL,'.
'subject VARCHAR(60) NOT NULL,'.
'message TEXT NOT NULL,'.
'post _ date DATETIME NOT NULL,'.
'PRIMARY KEY (post _ id))';
```

На заметку

Тип данных `VARCHAR` ограничивает указанным числом количество символов. Тип данных `TEXT` ограничивает объем текста 65 535 символами.

3. Теперь добавим проверяющую конструкцию, сообщающую, была ли таблица создана успешно, и выдающая ошибку в противном случае.

Совет

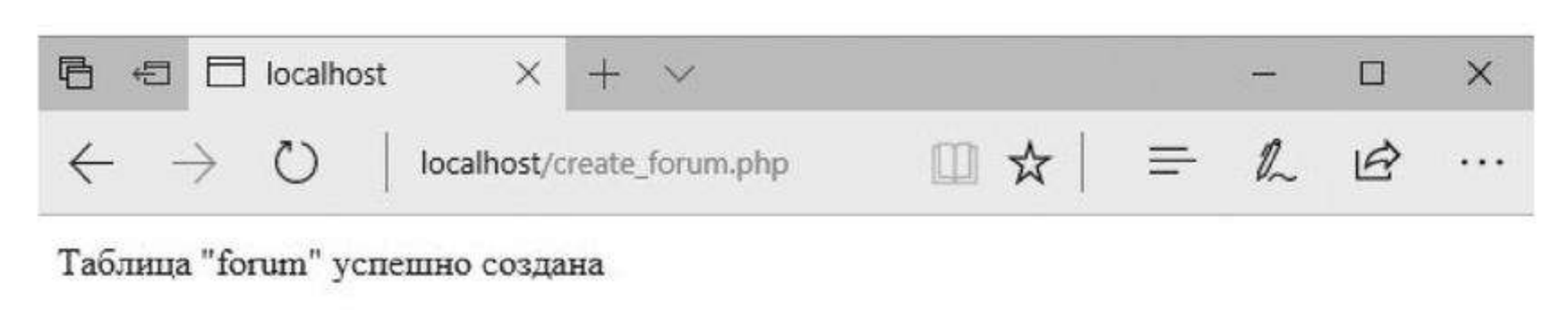
Функция `mysqli_query()` выполняет SQL-запрос, а функция `mysqli_error()` оповещает о возникновении ошибки.

```
if(mysqli_query($dbc, $sql) === TRUE)
{
 echo 'Таблица "forum" успешно создана';
}
else
{
 echo 'Ошибка создания таблицы: '.mysqli_error($dbc);
}
```

4. Не забудьте по окончании закрыть подключение к базе данных.

```
mysqli_close($dbc);
```

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *create_forum.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть результат создания таблицы.



Страницы форума

Страницы форума отображают ранее опубликованные сообщения, извлеченные из таблицы `forum` базы данных MySQL, созданной ранее в этой главе. Сообщения форума могут отображаться между шапкой и подвалом страницы, вместе с ссылкой на другую страницу, на которой пользователь может опубликовать новое сообщение в форуме. Если база данных `forum` не содержит ранее опубликованных сообщений, сценарий сообщит об их отсутствии.

PHP-функция `mysqli_query()` позволяет выполнить SQL-запрос для извлечения содержимого таблицы базы данных, а функция `mysqli_num_rows()` определяет число табличных строк, доступных в результате запроса. Если табличные строки в результате обнаружены, используется функция `mysqli_fetch_array()` для извлечения ассоциативного массива строк и данных в каждом столбце с передачей ей результата и константы `MYSQLI_ASSOC` в качестве аргументов.

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы.

forum.php

```
<?php
$page _ title = 'PHP-форум';
```

2. Далее добавьте инструкцию, встраивающую шапку страницы.

```
include ('includes/header.html');
```

3. Добавьте инструкцию, открывающую подключение к базе данных.

```
require ('..\connect _ db.php');
```

4. Извлеките все сообщения из таблицы базы данных и назначьте их переменной.

```
$sql = 'SELECT * FROM forum';
$result = mysqli _ query($dbc, $sql);
```

5. Выполните проверку, пуста ли таблица базы данных, и, если это так, выведите соответствующее оповещение.

```
if (mysqli _ num _ rows($result) > 0)
{
 // Сюда вставляются инструкции (шаг 6).
}
else
{
 echo '<p>В настоящее время сообщений нет.</p>';
}
```


На заметку

Столбцы возвращаются в виде ассоциативного массива, в котором имя строки — это ключ для каждого значения в этом столбце.

6. Затем добавьте инструкции, выводящие таблицу со списком всех опубликованных сообщений, извлеченных из таблицы базы данных, если они обнаружены.

```
echo '<table><tr><th>Автор</th><th>Тема</th><th
id="msg">Сообщение</th></tr>';
```

```
while ($row = mysqli_fetch_array($result, MYSQLI_
ASSOC))
{
 echo '<tr><td>'.
 $row['first_name'].' '.
 $row['last_name'].'<br>'.
 $row['post_date'].'</td><td>'.
 $row['subject'].'</td><td>'.
 $row['message'].'</td></tr>';
}

echo '</table>';
```

Совет

Строки таблицы базы данных возвращаются в массив, в котором имена строк используются в качестве индексов массива. Ячейки в строках таблицы будут заполнены данными, содержащимися в столбцах на строке таблицы базы данных `forum`.

7. После проверяющего блока добавьте инструкцию, создающую ссылку на страницу для создания нового сообщения.

```
echo '<p><a href="post.php">Написать сообщение</a></p>';
```

8. Теперь добавьте инструкцию, закрывающую подключение к базе данных.

```
mysqli_close($dbc);
```

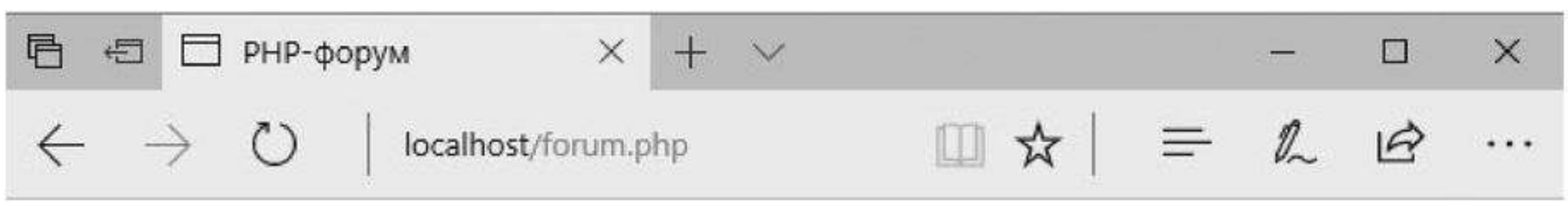
9. Добавьте финальную инструкцию, встраивающую подвал страницы.

```
include ('includes/footer.html');
?>
```

10. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *forum.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть страницу форума.

На заметку

Приемы встраивания шапки и подвала в HTML-страницу описаны в главе 9.



Шапка страницы

В настоящее время сообщений нет.

Написать сообщение

Подвал страницы

Предоставление формы

Завершив PHP-сценарий для отображения опубликованных сообщений на форуме, как это было описано в предыдущем разделе, теперь мы можем создать PHP-сценарий, с помощью которого пользователь сможет создавать новые сообщения. Для этого предоставим HTML-форму с полями для имени, фамилии и названия темы, а также текстовой областью для самого сообщения.

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы.

post.php

```
<?php
$page _ title = 'PHP — Публикация сообщения';
```

2. Далее добавьте инструкцию, встраивающую шапку страницы.

```
include ('includes/header.html');
```

3. Добавьте инструкцию для вывода веб-формы.

```
echo '<form action="process.php"
                    method="POST" accept-charset="utf-8">

Имя: <input name="first _ name" type="text">
Фамилия: <input name="last _ name" type="text">

  <p>Тема:<br>
  <input name="subject" type="text" size="64"></p>
  <p>Сообщение:<br>
  <textarea name="message" rows="5" cols="50">
```

```
    </textarea></p>
    <p><input type="submit" value="Отправить"></p>

</form>';
```

Совет

Обратите внимание, что атрибут `action` элемента `form` назначает обработчик данных с именем ***process.php***. Этот файл описан в следующем разделе.

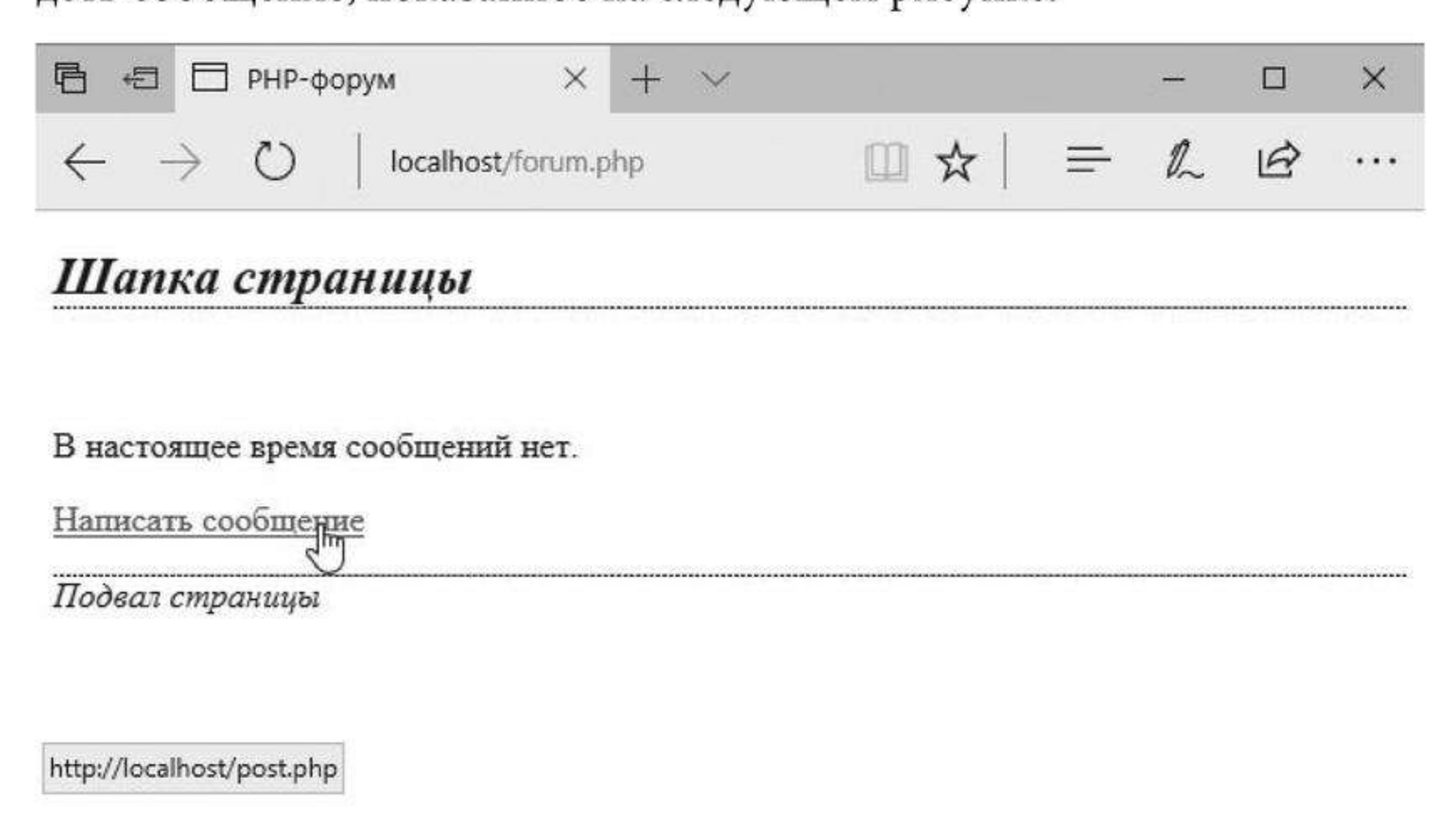
4. Теперь добавьте инструкцию, создающую ссылку на форум.

```
echo '<p>
<a href="forum.php">Вернуться к форуму</a></p>';
```

5. Добавьте финальную инструкцию, встраивающую подвал страницы.

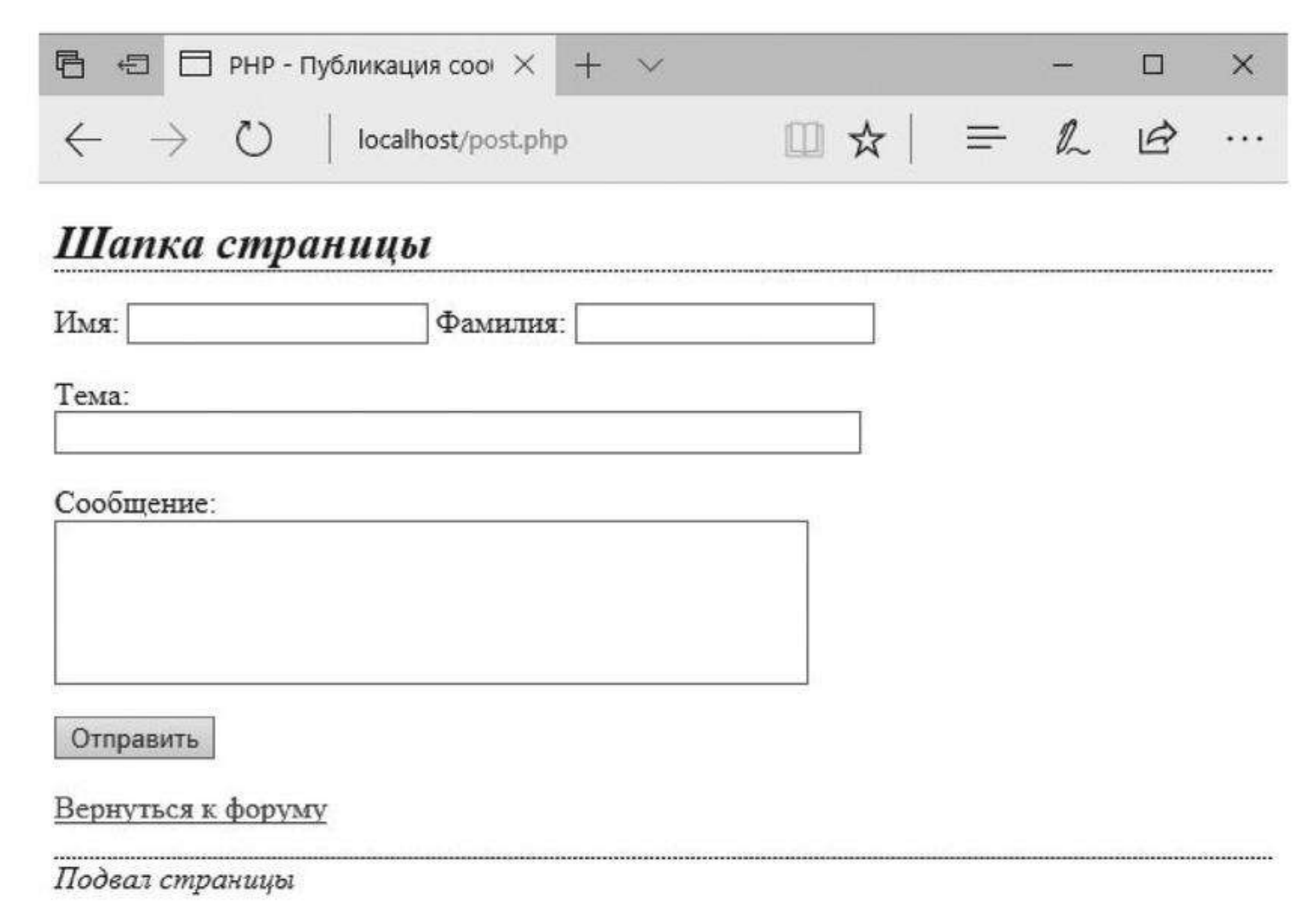
```
include ('includes/footer.html');
?>
```

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *post.php*, а затем перейдите по ссылке на страницу форума, чтобы увидеть сообщение, показанное на следующем рисунке.



Совет

В реальности пользователи форума могут сначала регистрировать свои учетные записи. В этом случае эти учетные записи можно также хранить в таблице базы данных для последующего вызова. В примере для упрощения имена пользователей просто передаются на сервер из HTML-формы.



На заметку

Страница, предназначенная для создания нового сообщения, содержит ссылку на главную страницу форума, чтобы пользователь мог на нее вернуться.

Обработка сообщений

После того как готов PHP-сценарий с формой для отправки новых сообщений на форум, можно приступить к созданию сценария для обработки передаваемых сообщений.

Сценарий для обработки данных, переданных из HTML-формы, может гарантировать, что все поля формы заполнены, проводя для этого простую проверку. Функция `trim()` используется для удаления пробельных символов, а функция `empty()` — для проверки, содержат ли поля формы значения.

После того как данные успешно прошли проверку, сценарий может сохранить переданное сообщение в таблицу `forum` базы данных MySQL. Если попытка опубликовать сообщение выполнена успешно, можно вывести страницу форума для отображения сообщений, а в противном случае — показать сообщение об ошибке.

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы.

process.php

```
<?php
$page_title = 'PHP — Ошибки';
```

2. Добавьте инструкцию, встраивающую шапку страницы.

```
include ('includes/header.html');
```

3. Теперь добавьте функцию `exit`, чтобы вывести уведомление и ссылку на страницу создания сообщения, если проверка была провалена.

```
function fail($str)
{
 echo "<p>Пожалуйста, укажите $str.</p>";
 echo '<p><a href="post.php">Написать сообщение</a>';
 include ('includes/footer.html');
 exit();
}
```

На заметку

Этот сценарий назначается в файле ***post.php*** и служит для обработки передаваемых пользователем сообщений. Форма не сохраняет данные, поэтому, если пользователь возвращается к форме из-за ошибки ввода, все поля будут пусты.

4. Добавьте код, обеспечивающий передачу данных веб-формы.

```
if(isset($_POST['message']))
{

 // Сюда вставляются инструкции (шаг 5).

 // Сюда вставляются инструкции (шаг 6).

 // Сюда вставляются инструкции (шаг 7).

}
```

5. Добавьте код, проверяющий, что пользователь заполнил все поля формы.

```
if (!empty(trim($_POST['first_name])))
{$first_name = addslashes($_POST['first_name']);}
else {fail('имя');}

if (!empty(trim($_POST['last_name])))
{$last_name = addslashes($_POST['last_name']);}
else {fail('фамилию');}

if (!empty(trim($_POST['subject'])))
{$subject = addslashes($_POST['subject']);}
else {fail('тему');}

if (!empty(trim($_POST['message'])))
{$message = addslashes($_POST['message']);}
else {fail('текст сообщения');}
```

Внимание

Обратите внимание, что в коде используется PHP-функция `addslashes()`, экранирующая апострофы и кавычки.

6. Если проверка завершена успешно, создайте подключение к базе данных.

```
require ('../connect_db.php');
```

7. Напишите инструкции, добавляющие переданные значения полей HTML-формы в таблицу `forum` базы данных MySQL.

```
$sql = "INSERT INTO forum
(first_name, last_name, subject, message, post_date)
VALUES
('$first_name', '$last_name','$subject', '$message',
NOW()) ";

$result = mysqli_query ($dbc, $sql);
```

Совет

MySQL-функция `NOW()` возвращает значение даты/времени согласно текущей метке времени.

8 И, наконец, добавьте инструкцию для оповещения об ошибке в случае сбоя и, в противном случае, перенаправления браузера на страницу форума.

```
if (mysqli_affected_rows($dbc)!= 1)
{
 echo '<p>Ошибка</p>'.mysqli_error($dbc);
 mysqli_close($dbc);
}
else
{
 mysqli_close($dbc);
 header('Location: forum.php');
}
```

Совет

Этот сценарий выводит собственные сообщения об ошибках, если обнаруживаются незаполненные поля формы, и сообщения об ошибках MySQL, если значения не могут быть добавлены в таблицу базы данных.

9 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *process.php*, а затем отправьте форму, оставив все поля незаполненными, чтобы увидеть сообщения об ошибках.

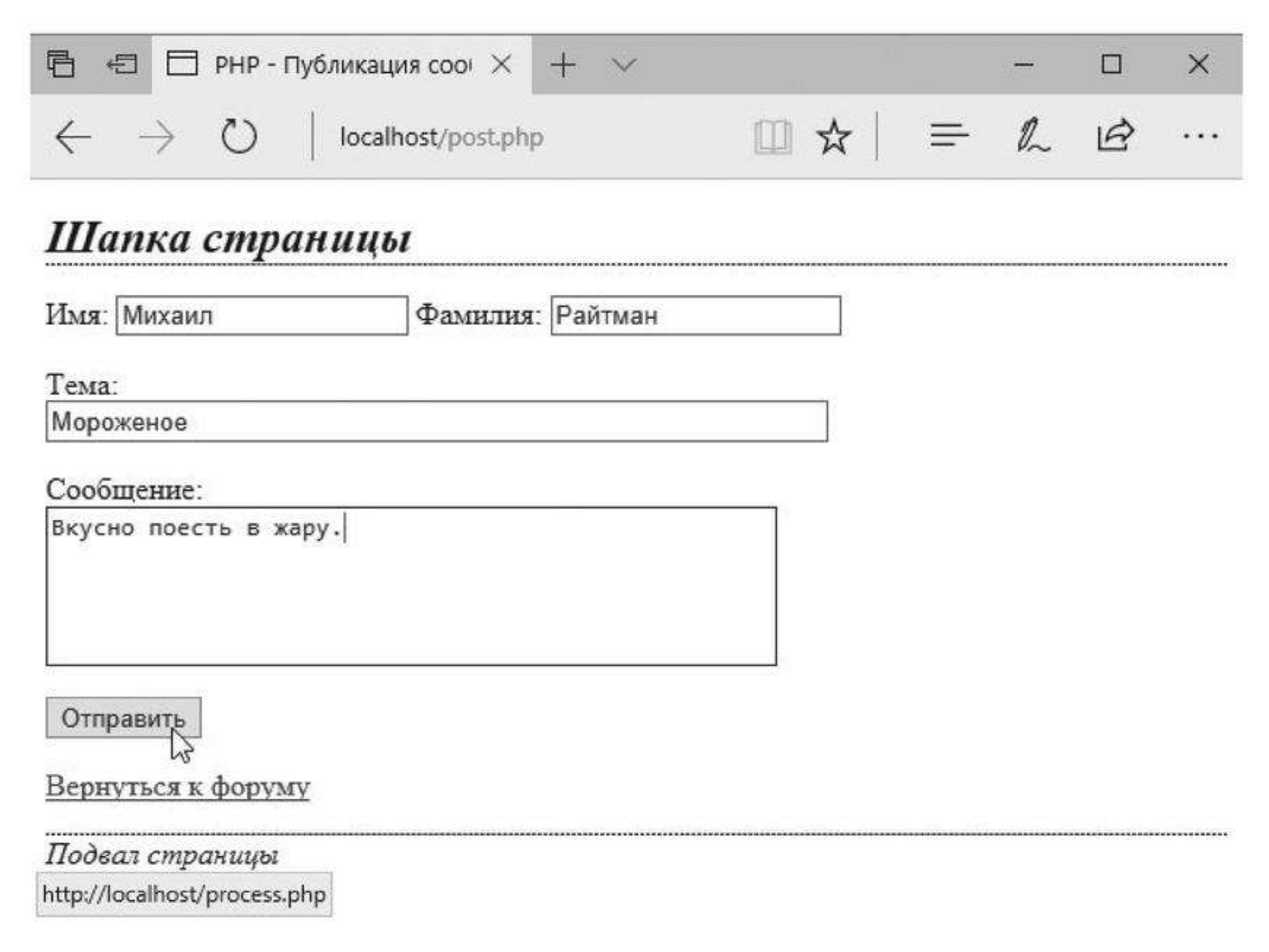
Проверка работоспособности

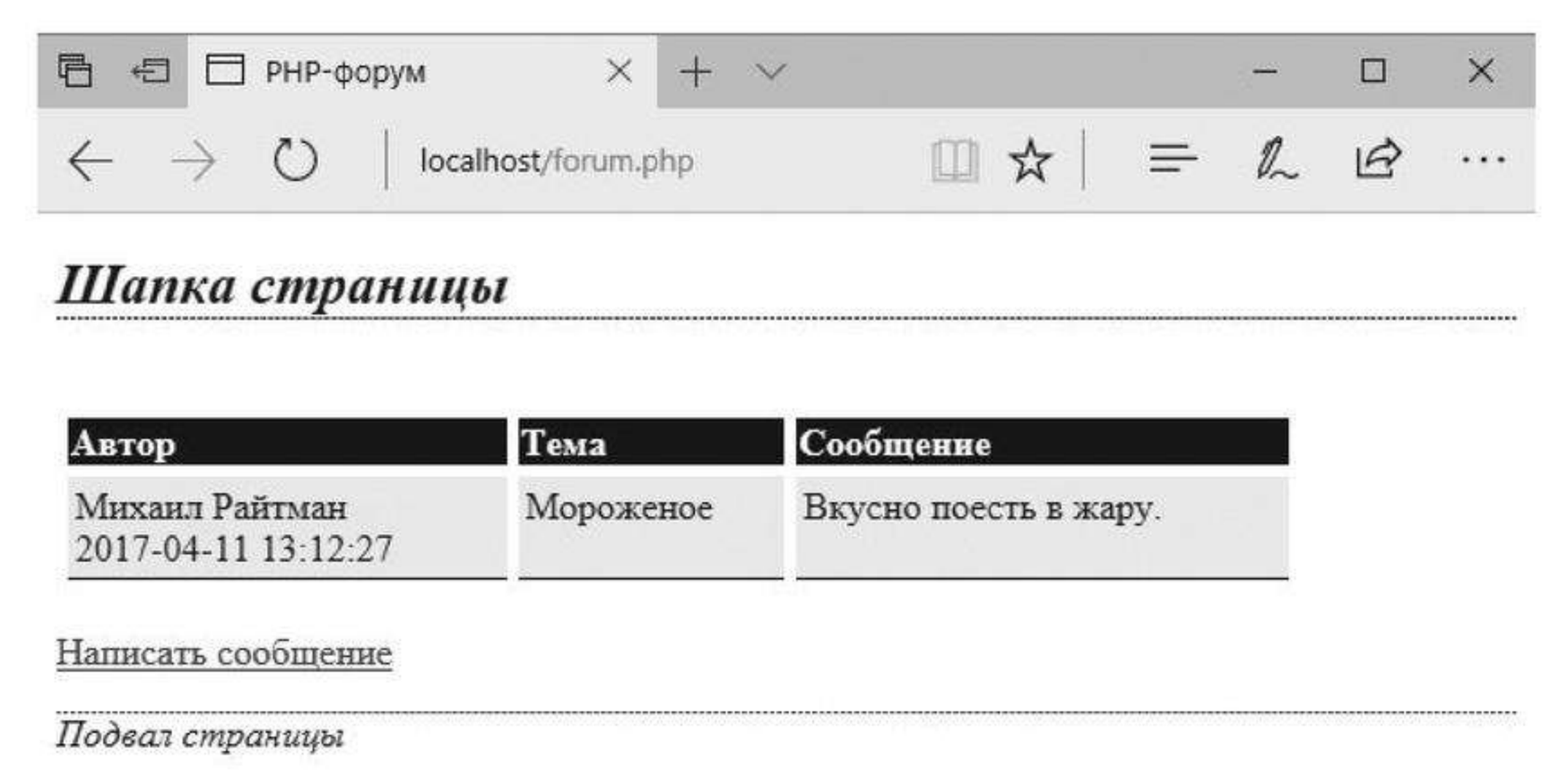
Теперь пришло время протестировать процесс публикации сообщений на форуме, чтобы убедиться в корректности выполняемых проверок и в том, что публикуемые сообщения появляются на странице форума.

1 Откройте страницу *post.php*, заполните все поля данными и отправьте форму, чтобы увидеть сообщение, появившееся на странице форума.

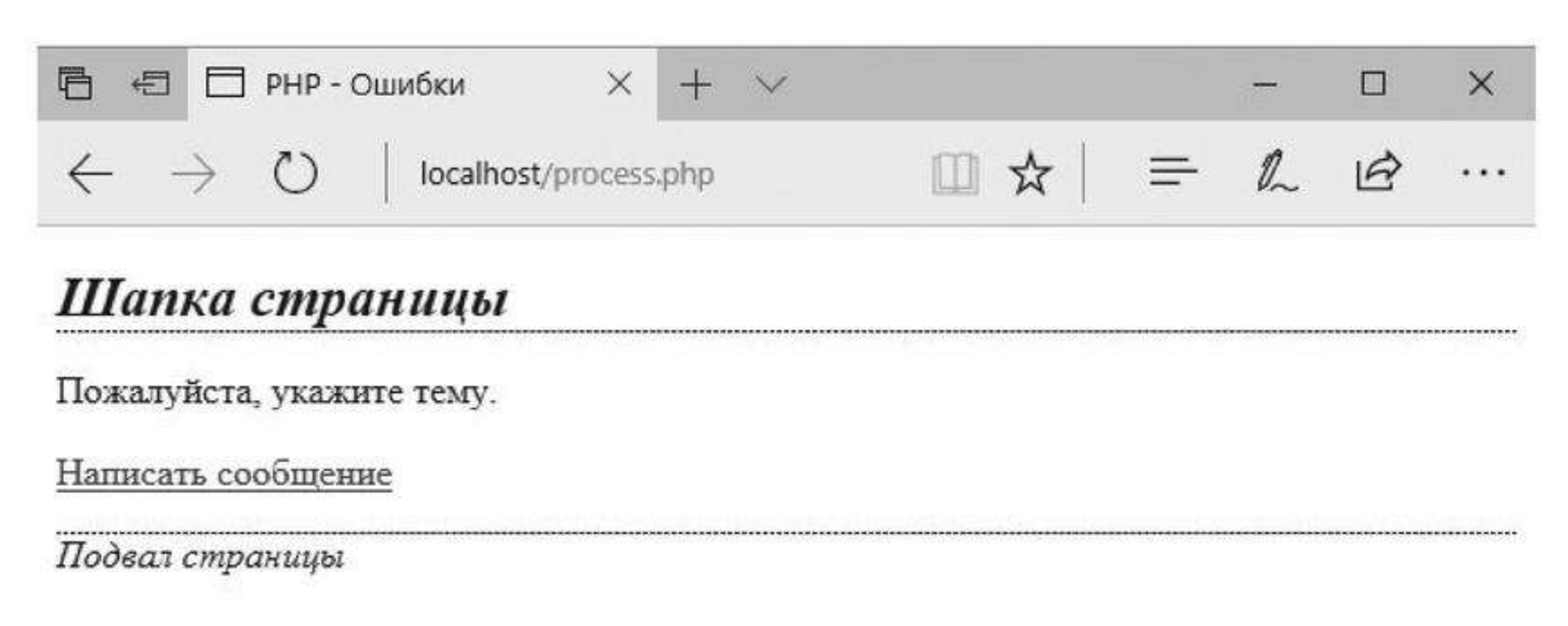
На заметку

Информация для столбца «Автор» извлекается из ячеек `first_name`, `last_name` и `POST_DATE` таблицы `forum` базы данных MySQL.





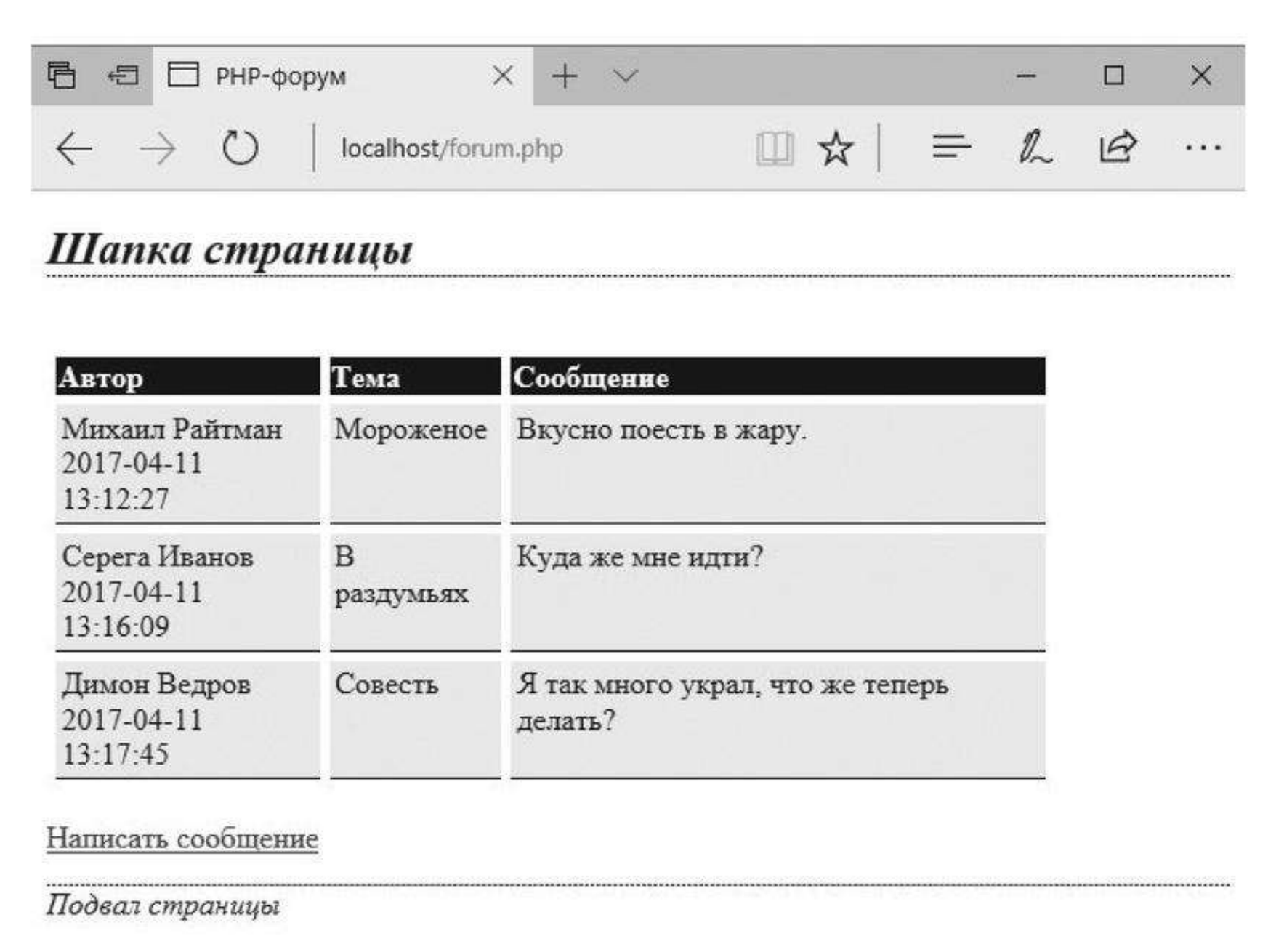
2 Затем вернитесь на страницу создания нового сообщения и отправьте форму с пустым полем «Тема», чтобы увидеть уведомление об ошибке.



Совет

Вы можете остановить службу MySQL Server, после чего отправить форму. Появится сообщение об ошибке MySQL.

3. Вернитесь на страницу создания нового сообщения и отправьте несколько сообщений от имени разных пользователей, чтобы увидеть, как они отображаются на форуме.



Совет

В данном примере новые сообщения на форуме появляются в нижней части таблицы по мере добавления, но такой порядок сортировки может быть изменен путем добавления правила `SORT` в SQL-запрос.

Заключение

- HP-сценарий может подключиться к базе данных MySQL с помощью функции `mysqli_connect()`, в качестве аргументов которой указывают сервер, имя пользователя и пароль, а также имя базы данных.
- Функция `mysqli_connect()` возвращает объект подключения к базе данных, который может быть передан в качестве аргумента другим функциям, таким как `mysqli_set_charset()`.
- Функция `die()` эквивалентна вызову функции `exit()`, позволяющей завершить работу сценария.
- Функция `mysqli_connect_error()` возвращает сообщение об ошибке, если не удается попытка установить подключение, и может использоваться в качестве аргумента функции `die()`, вызываемой при остановке сценария.
- В целях безопасности рекомендуется поместить сценарий подключения к базе данных вне обычного каталога */htdocs* и при необходимости включать его в другие сценарии с помощью инструкции `require`.
- Функция `mysqli_ping()` проверяет подключение к указанной базе данных, а затем данные могут быть извлечены с помощью функций `mysqli_get_server_info()` и `mysqli_get_host_info()`.
- Функция `mysqli_query()` выполняет SQL-запрос, а функция `mysqli_error()` оповещает о возникновении ошибки.
- Подключения к базам данных всегда должны закрываться, если они больше не требуются, посредством вызова функции `mysqli_close()`.
- Если запрос к базе данных завершен успешно, функция `mysqli_num_rows()` может определить общее количество извлекаемых строк.
- Функция `mysqli_fetch_array()` позволяет создать ассоциативный массив строк и данных в случае осуществления успешного запроса.
- PHP-функция `empty()` используется для проверки полей HTML-формы, определяя, заполнены они или нет.
- Строка данных может быть добавлена в таблицу базы данных MySQL с помощью функции `mysqli_query()`, а затем функция `mysqli_affected_rows()` может проверить и подтвердить данную операцию.
- Браузер может быть перенаправлен на новую страницу с помощью функции `header()` с указанным в ней аргументом `'Location: URL-адрес'`.

12

Подключение веб-служб

Прочитав эту главу, вы научитесь встраивать XML-контент в PHP-сценарии.

- Загрузка данных
- Получение узлов
- Получение атрибутов
- Добавление RSS-лент
- Настройка параметров
- Выбор компонентов
- Заключение

Загрузка данных

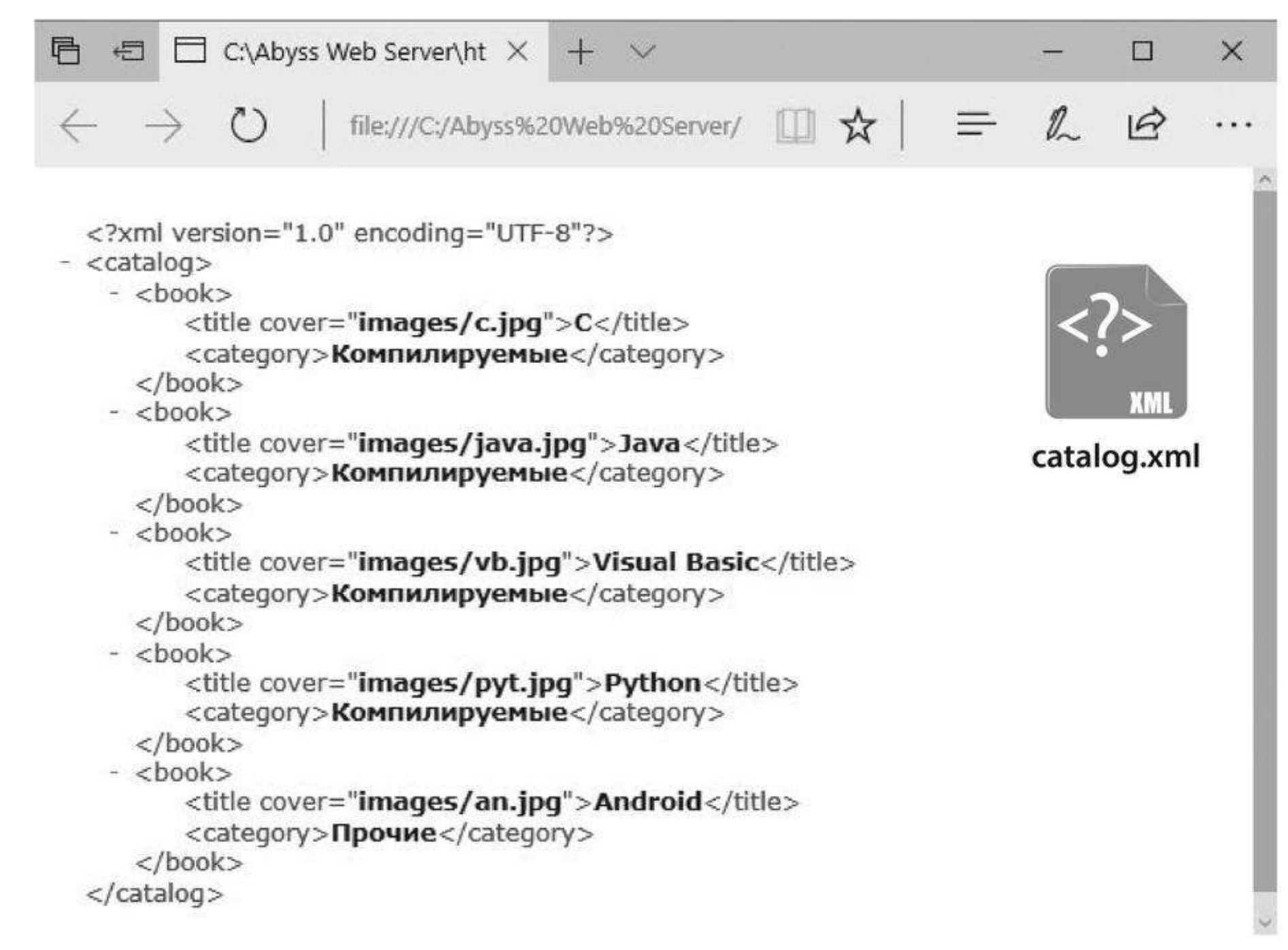
В интернете существует много веб-сервисов, бесплатно предоставляющих разнообразные данные, которые можно встраивать на пользовательские веб-страницы. Типичными примерами могут служить котировки акций, актуальные новости или прогноз погоды.

Большинство веб-сервисов предоставляют данные в формате XML (Extensible Markup Language — расширяемый язык разметки). Данный формат используется для удобства восприятия человеком машиночитаемого кода и не зависит от используемой платформы. К примеру, Java-приложение может передавать XML-данные через интернет .NET-программе и обратно.

Внимание

Все XML-документы должны начинаться с объявления типа XML-файла — как показано на рисунке, в первой строке.

Язык XML очень похож на HTML, но в XML можно использовать любые собственные имена элементов. Теги элементов окружают контент и могут включать в себя атрибуты, как и в случае с HTML. Важно, что каждый элемент имеет как открывающие, так и закрывающие теги и что элементы в XML-коде корректно вкладываются для удобного и правильного оформления.



Совет

Файл *catalog.xml*, показанный на данном рисунке, будет использоваться в нескольких следующих примерах, с помощью которых демонстрируется, как загружать XML-данные и получать доступ к значениям элементов или атрибутов.

Язык PHP предоставляет XML-анализатор (парсер) под названием SimpleXML, который может загружать XML-документы в память в виде иерархической древовидной структуры. Функция `simplexml_load_file()` требует указать имя XML-файла и путь для преобразования документа в структуру данных, содержащей SimpleXMLElement-объекты. Затем структура данных может быть просмотрена с помощью функции `var_dump()`.

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы, и укажите элемент для предварительного форматирования текста.

catalog.php

```
<?php
echo '<title>PHP — Простой XML</title><pre>';
```

2. Далее добавьте инструкцию, загружающую XML-данные или завершающую сценарий в случае сбоя.

```
$xml = simplexml_load_file('catalog.xml')
or die('Невозможно загрузить данные!');
```

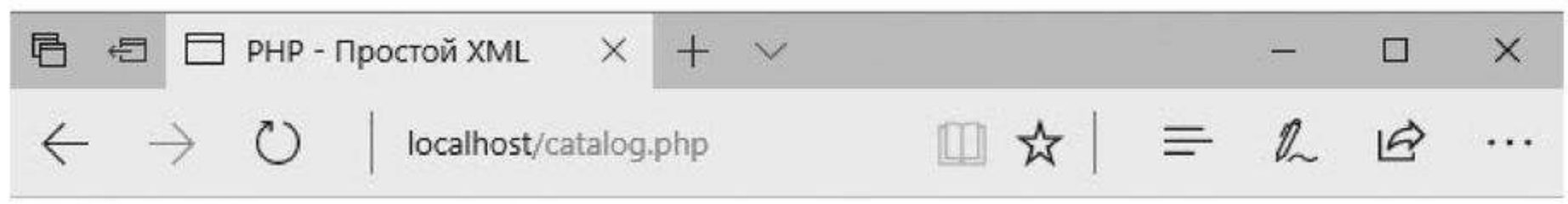
3. Теперь выведите все SimpleXMLElement-объекты.

```
var_dump($xml);
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *catalog.php*, вместе с файлом *catalog.xml*, а затем откройте этот PHP-файл в браузере через протокол HTTP, чтобы увидеть SimpleXMLElement-объекты.

Совет

Обратите внимание, что эта структура данных описывает массив с именем `book`, содержащий пять элементов массива, которые соответствуют пяти XML-элементам `book`.




```
object(SimpleXMLElement)#1 (1) {
  ["book"]=>
  array(5) {
    [0]=>
    object(SimpleXMLElement)#2 (2) {
      ["title"]=>
      string(1) "C"
      ["category"]=>
      string(26) "Компилируемые"
    }
    [1]=>
    object(SimpleXMLElement)#3 (2) {
      ["title"]=>
      string(4) "Java"
      ["category"]=>
      string(26) "Компилируемые"
    }
    [2]=>
    object(SimpleXMLElement)#4 (2) {
      ["title"]=>
      string(12) "Visual Basic"
      ["category"]=>
      string(26) "Компилируемые"
    }
    [3]=>
    object(SimpleXMLElement)#5 (2) {
      ["title"]=>
      string(6) "Python"
      ["category"]=>
      string(26) "Компилируемые"
    }
    [4]=>
    object(SimpleXMLElement)#6 (2) {
      ["title"]=>
      string(7) "Android"
      ["category"]=>
      string(12) "Прочие"
    }
  }
}
```


На заметку

Распространенность XML делает его идеальным для передачи данных через интернет.

Получение узлов

Структуры данных, содержащиеся в SimpleXML-объектах, описывают родственные отношения между внешними и внутренними XML-элементами. Каждый член структуры данных известен под термином «узел». Узлы верхнего уровня — это прямые потомки самого SimpleXML-объекта, на которые

можно ссылаться, используя имя объекта, оператор `->` и имя дочернего узла. К примеру, `$object->`*`потомок`*. На потомок дочернего узла (внук) можно ссылаться путем добавления еще одного оператора `->` и имени внучатого узла, например `$object->`*`потомок`*`->`*`внук`* и так далее для каждого последующего поколения.

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы.

```
<?php
echo '<title>PHP — Простой XML</title>';
```

nodes.php

2. Далее добавьте инструкцию, загружающую XML-данные или завершающую сценарий в случае сбоя.

```
$xml = simplexml_load_file('catalog.xml')
or die('Невозможно загрузить данные!');
```

3. Затем извлеките содержимое элемента `title`, являющегося потомком первого элемента `book`.

```
echo 'Первая книга: '.
$xml->book[0]->title.' для начинающих<br>';
```

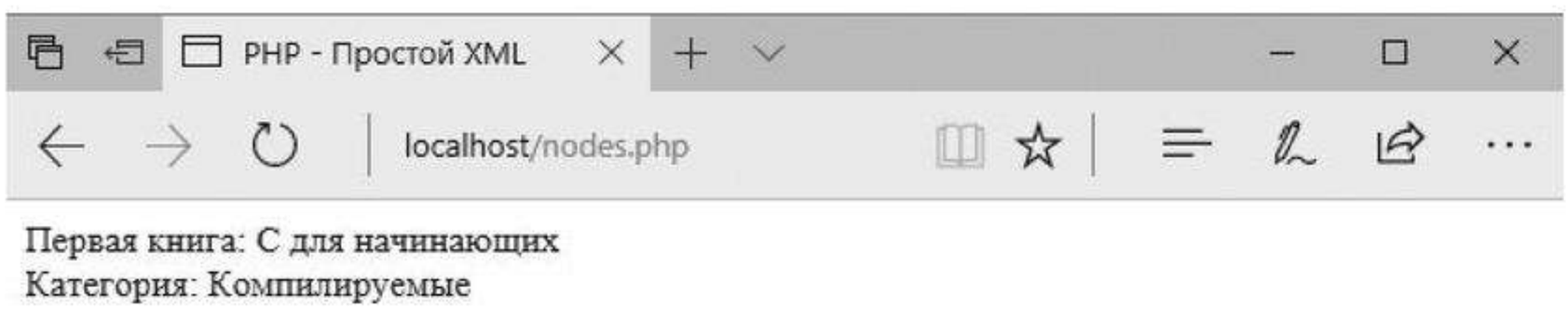
4. Теперь извлеките содержимое элемента `category`, являющегося потомком первого элемента `book`.

```
echo 'Категория: '.$xml->book[0]->category;
```

Совет

Так как пять узлов `book` расположены на одном и том же уровне иерархии, в данном примере они используются в виде массива. Следовательно, можно ссылаться на каждый узел по отдельности, используя имя узла и индекс элемента массива.

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *nodes.php*, вместе с файлом *catalog.xml*, а затем откройте данный PHP-файл в браузере через протокол HTTP, чтобы увидеть содержимое узла.



Перебор узлов

SimpleXML-объект предоставляет методы для работы с данными, содержащимися в его структуре данных. Удобный метод `children()` возвращает все дочерние узлы и может быть использован для циклического обхода узлов для извлечения содержимого каждого из них.

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы.

```
<?php
echo '<title>PHP — Простой XML</title>';
```

nodes_loop.php

2. Далее добавьте инструкцию, загружающую XML-данные или завершающую сценарий в случае сбоя.

```
$xml = simplexml_load_file('catalog.xml')
or die('Невозможно загрузить данные!');
```

3. Теперь инициализируйте счетчик итераций со значением, равным нулю.

```
$counter = 0;
```

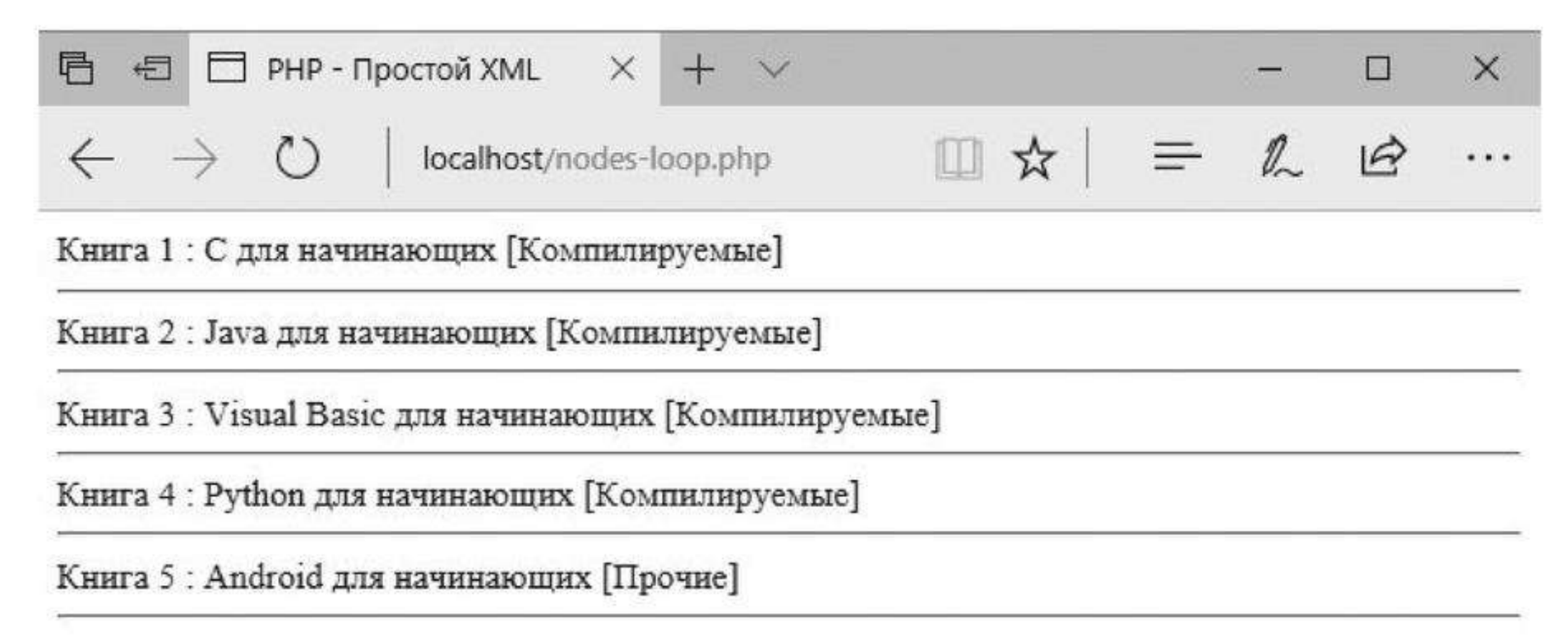
4. Добавьте цикл, инкрементирующий значение счетчика и выводящий содержимое элементов `title` и `category` всех дочерних узлов.

```
foreach($xml->children() as $book)
{
$counter++;
echo 'Книга '.$counter.': ';
echo $book->title.
' для начинающих ['.$book->category.']<hr>';
}
```

Совет

Для обхода дочерних узлов каждого поколения можно использовать вложенные циклы `foreach`.

5. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *loop.php*, вместе с файлом *catalog.xml*, а затем откройте этот PHP-файл в браузере через протокол HTTP, чтобы увидеть содержимое узла.



Получение атрибутов

На значения, содержащиеся в атрибутах SimpleXMLElement-объекта, можно ссылаться с указанием имени атрибута, заключая его в одиночные кавычки и квадратные скобки, помещая его после имени узла. К примеру, ссылка `$object->потомок['атрибут']` обращается к значению *атрибута* элемента *потомка* верхнего уровня.

attribs.php

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы.

```
<?php
echo '<title>PHP — Простой XML</title>';
```

2. Далее добавьте инструкцию, загружающую XML-данные или завершающую сценарий в случае сбоя.

```
$xml = simplexml_load_file('catalog.xml')
or die('Невозможно загрузить данные!');
```

3. Затем извлеките содержимое элемента `title`, являющегося потомком первого элемента `book`.

```
echo 'Первая книга: '.
$xml->book[0]->title.' для начинающих<br>';
```

4. Отобразите значение атрибута `cover` элемента `title`, являющегося потомком первого элемента `book`.

```
echo 'Обложка: '.$xml->book[0]->title['cover'].'<br>';
```

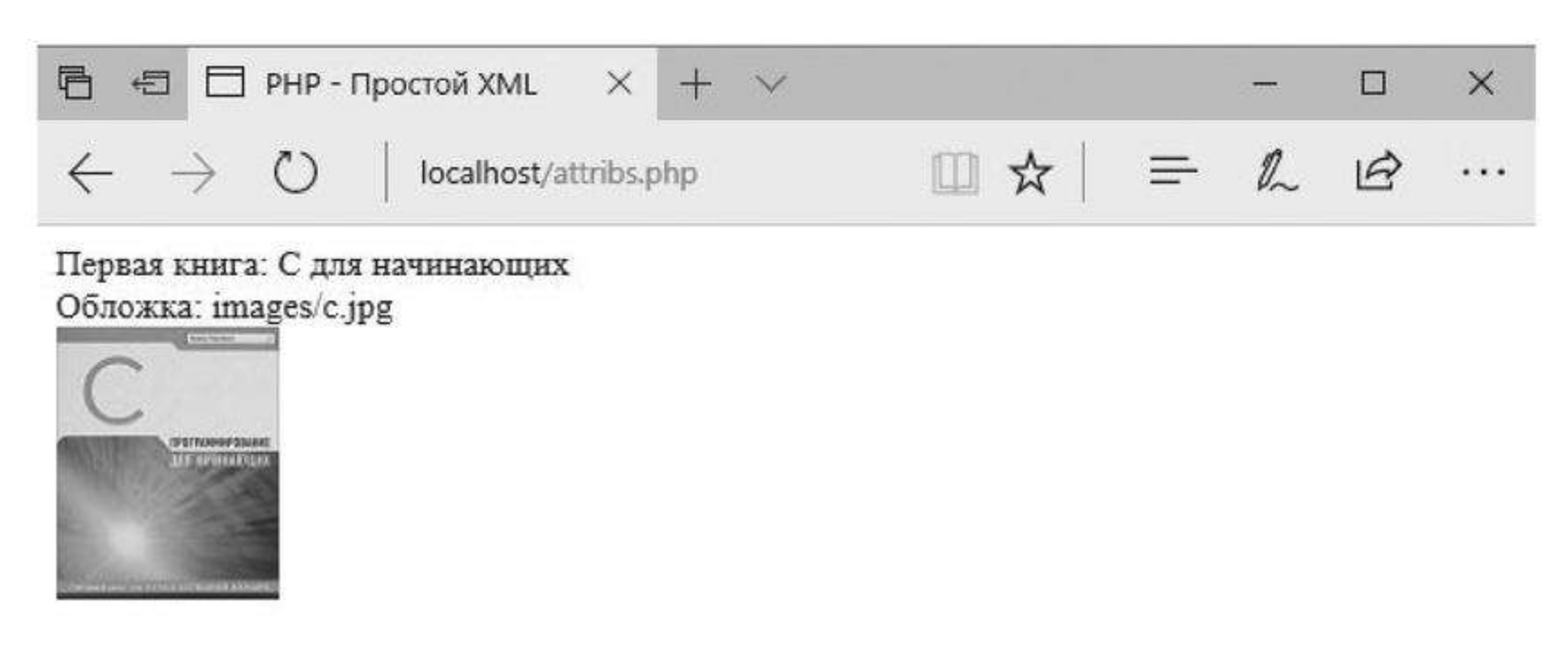
5. Используйте полученное на предыдущем шаге значение, чтобы вывести изображение обложки первого элемента `book`.

```
echo '<img src="'.$xml->book[0]->title['cover'].'">';
```

На заметку

Обратите внимание, что в этом примере каждое значение атрибута определяет путь и имя файла изображения в каталоге */htdocs/images*.

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *attribs.php*, вместе с файлом *catalog.xml*, а затем откройте этот PHP-файл в браузере через протокол HTTP, чтобы увидеть значение атрибута и изображение обложки книги.



Перебор атрибутов

Метод `children()` SimpleXML-объекта может быть еще раз использован для цикличного обхода узлов и извлечения значения указанного атрибута конкретного узла при каждой итерации.

1. Начните PHP-сценарий с инструкции, устанавливающей название страницы.

```
<?php
echo '<title>PHP — Простой XML</title>';
```

attribs-loop.php

2. Далее добавьте инструкцию, загружающую XML-данные или завершающую сценарий в случае сбоя.

```
$xml = simplexml_load_file('catalog.xml')
or die('Невозможно загрузить данные!');
```

3. Теперь инициализируйте счетчик итераций со значением, равным нулю.

```
$counter = 0;
```

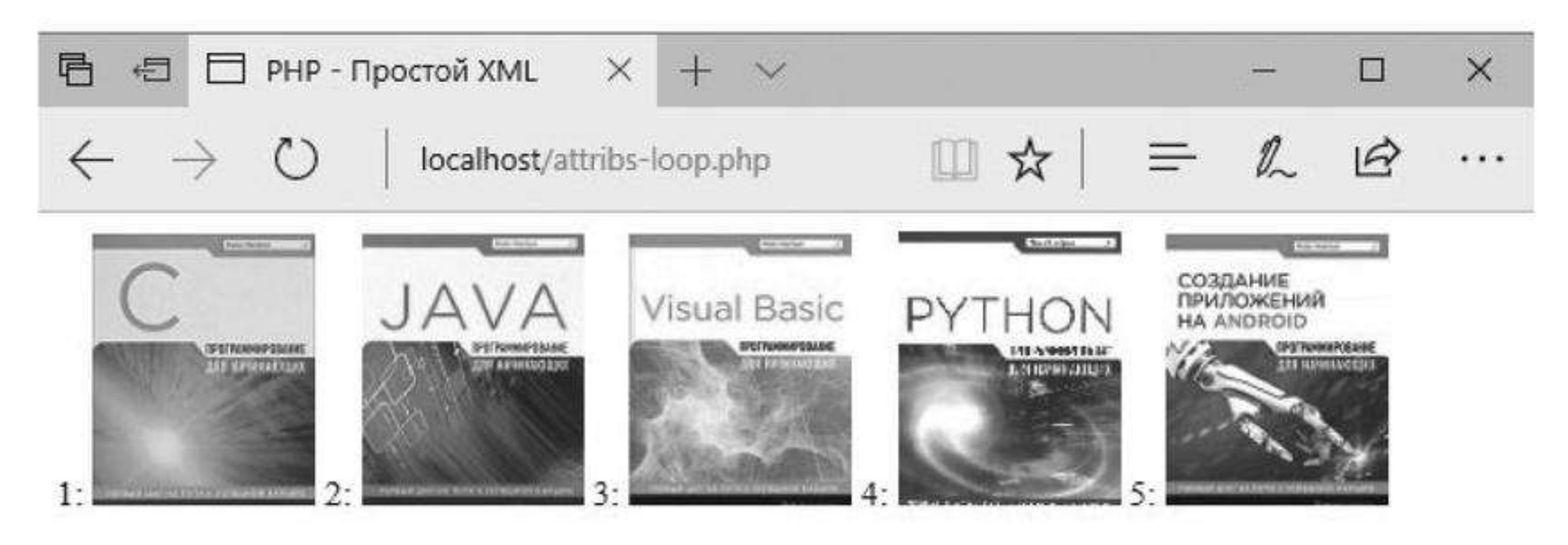
4 Добавьте цикл, инкрементирующий и выводящий значение счетчика, а затем извлекающий значение атрибута во время каждой итерации для вывода изображения обложки книги.

```
foreach($xml->children() as $book)
{
 $counter++;
 echo ' '.$counter.': ';
 echo '<img src="'.$book->title['cover'].'">';
}
```

Совет

SimpleXMLElement-объект также предоставляет метод `attributes()`, который возвращает ассоциативный массив с именами и значениями всех атрибутов в пределах элемента, например `$obj->`*`потомок`*`->attributes()`.

5 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *attribs-loop.php* вместе с файлом *catalog.xml*, а затем откройте этот PHP-файл в браузере через протокол HTTP, чтобы увидеть счетчик и изображение соответствующей обложки, отображаемое во время каждой итерации цикла.



Добавление RSS-лент

Благодаря возможности получить доступ к данным XML-документов из PHP-сценариев, как описано ранее в этой главе, мы можем легко встраивать на страницу данные из внешних источников. Многие веб-сервисы поддерживают бесплатные RSS-каналы (Rich Site Summary / Really Simple Syndication).

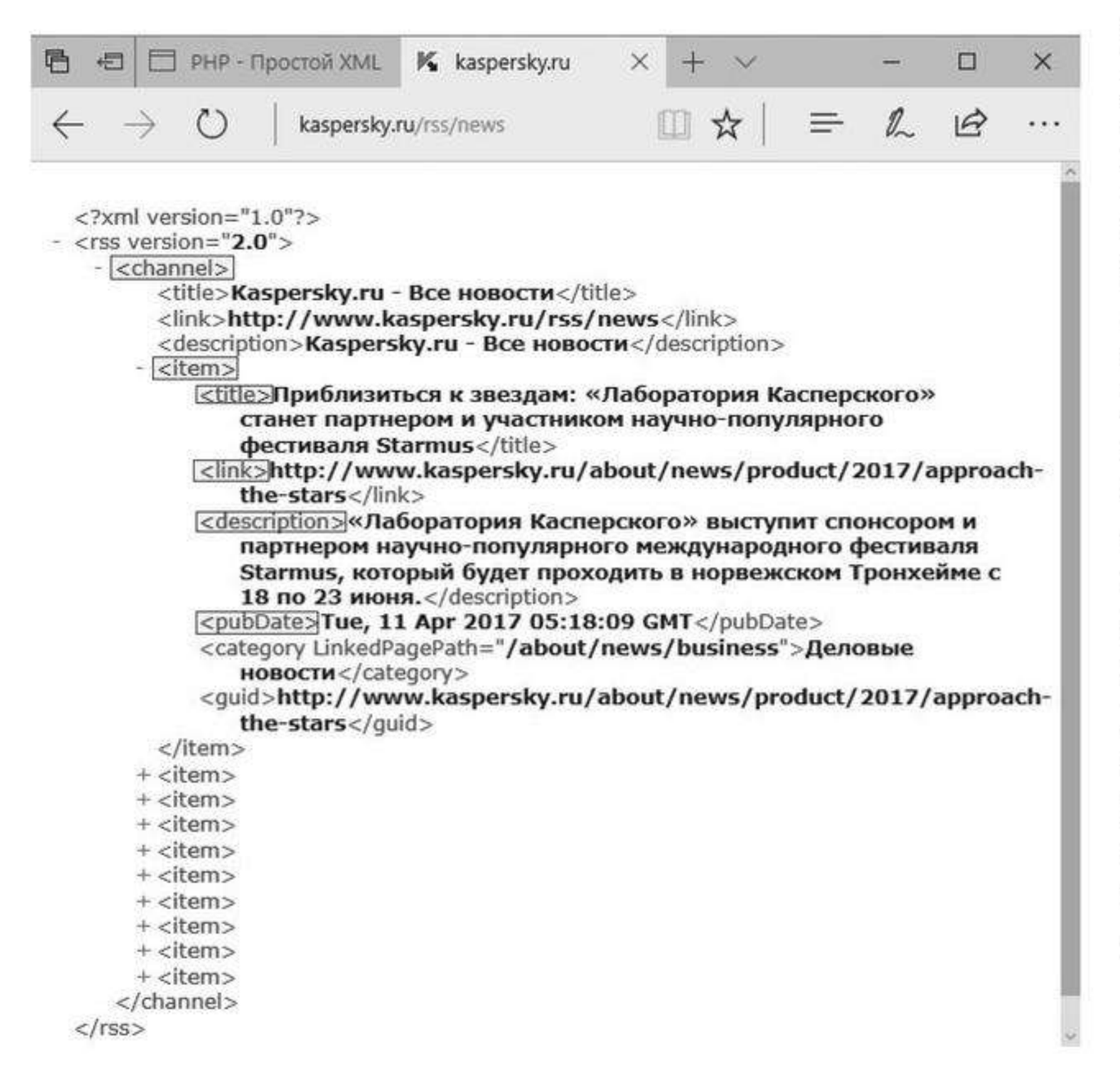
Вы можете узнать больше о RSS-каналах в интернете по адресу `rssboard.org`. Также на этом сайте можно указать URL-адрес любого RSS-канала в валидаторе, чтобы просмотреть его XML-элементы.

Выполнив поиск в интернете по запросу «каталог RSS-каналов», вы найдете множество веб-сайтов, на которых публикуются ссылки на бесплатные RSS-каналы. RSS-каналы передают регулярно обновляемые данные в формате XML.

На веб-сайте **subscribe.ru/catalog?rss** вы найдете ссылки на множество бесплатных RSS-каналов на различную тематику. Кроме того, по запросу «RSS-канал» в поисковой системе можно искать каналы, не входящие в каталог.

В данном примере рассматривается бесплатный RSS-канал компании «Лаборатория Касперского», на котором публикуется регулярно обновляемые новости, посвященные ИТ-тематике, доступный по адресу **http://www.kaspersky.ru/rss/news**. RSS-канал доставляется в веб-браузер в виде XML-документа, который может быть отображен в браузере. Кнопки рядом с внешними элементами интерактивны, вы можете щелкать по ним мышью, чтобы разворачивать (+) и сворачивать (–) вложенные элементы.

RSS — своего рода «диалект» языка XML. В RSS-файлах код обязательно должен оборачиваться в элемент верхнего уровня, `rss`, содержащий элемент `channel`. Данные содержатся во вложенных элементах.



Элементы, выделенные рамкой на данном рисунке, как и все остальные, отвечают за извлечение данных, но вызывают наибольший интерес. Каждый из элементов `item`, показанных свернутыми, содержит собственные элементы `title`, `description`, `link` и `pubDate`, отвечающие за название, описание, ссылку и дату публикации соответственно.

1. Начните PHP-сценарий с инструкции, загружающей XML-данные или завершающей сценарий в случае сбоя.

rss.php

```
<?php
$url = 'http://www.kaspersky.ru/rss/news';
$xml =
simplexml _ load _ file($url) or die('Невозможно загрузить
данные!');
```

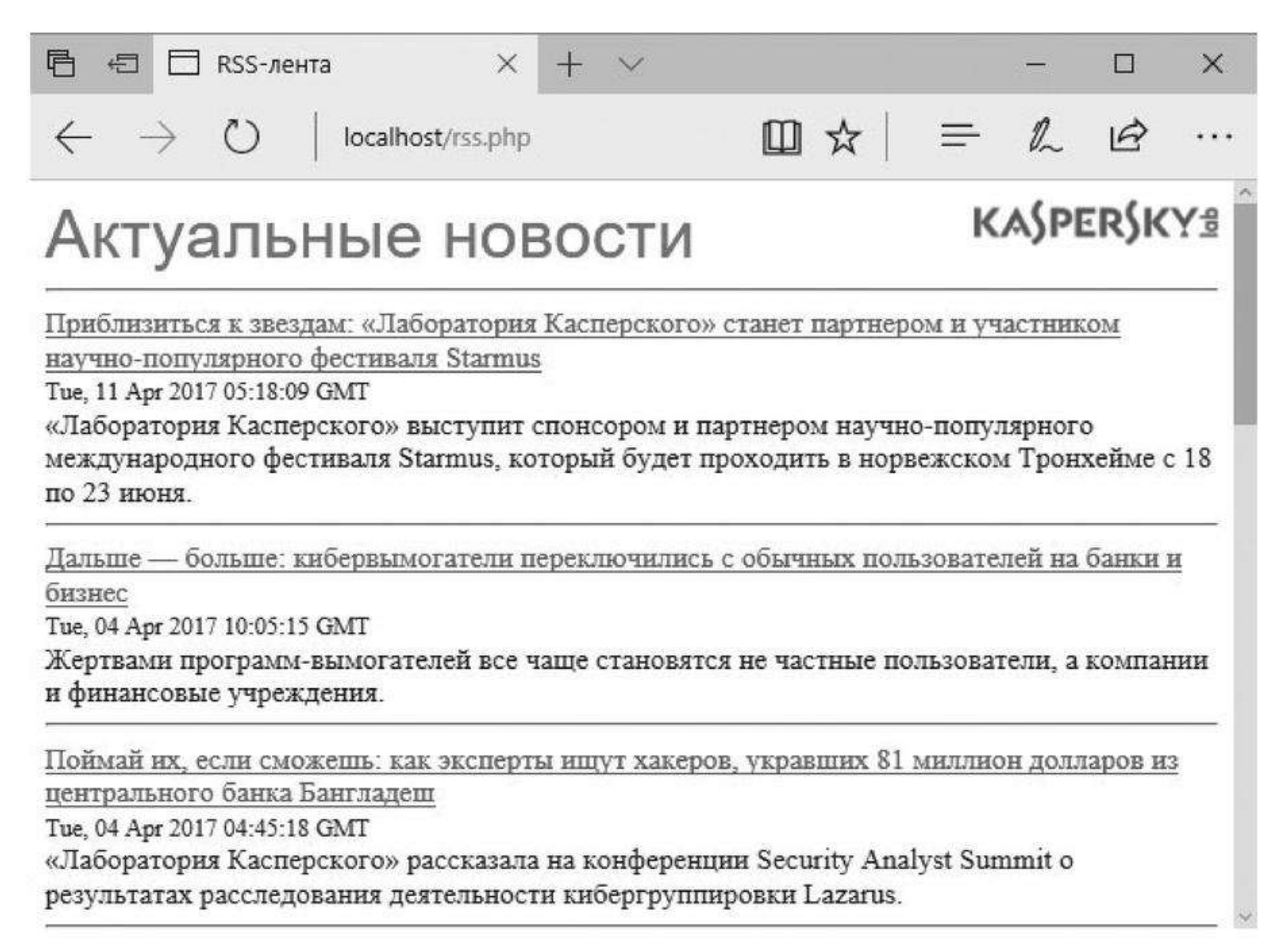
2. Далее добавьте инструкцию, встраивающую шапку страницы.

```
include('includes/rss-header.html');
```

3. Теперь добавьте код для вывода каждой новостной заметки, а также для встраивания подвала страницы.

```
foreach ($xml->channel->item as $item) {
echo '<a href="'.$item->link.'">' .$item->title.'</a>';
echo '<br><small>'.$item->pubDate.'</small><br>';
echo $item->description. '<hr>';}
include('includes/rss-footer.html');
```

4. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *rss.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть контент RSS-канала.



На заметку

Некоторые компании, предоставляющие RSS-каналы, требуют разместить сведения об авторских правах или логотип компании.

Настройка параметров

Некоторые компании предоставляют статичный URL-адрес для передачи данных, как RSS-канал «Лаборатории Касперского» из предыдущего примера, который просто доставляет XML-данные, определенные поставщиком контента. Другие веб-сервисы обеспечивают более гибкие API-интерфейсы (Application Programming Interface — интерфейсы программирования приложений), позволяющие установить параметры в URL-запросе с целью выбора, какие данные вы будете получать и в каком формате.

Дополнительная информация о компании OpenWeatherMap доступна на сайте openweathermap.org. Там же можно зарегистрироваться, чтобы получить собственный идентификатор, необходимый для работы с веб-сервисом.

Ресурс OpenWeatherMap предоставляет свободный доступ к собственному API-интерфейсу прогноза погоды, которые доступен по адресу **http://api.openweathermap.org/data/2.5/weather**. Этот API-интерфейс позволяет запрашивать данные о погоде с использованием различных форматов. Параметры, принимаемые этим API-интерфейсом, включают:

- `q` — название города (или название города с кодом страны),
- `units` — единицы измерения температуры: метрические (шкала Цельсия), имперские (шкала Фаренгейта) или стандартные (шкала Кельвина, используется по умолчанию),
- `mode` — формат данных: XML, HTML или JSON (по умолчанию),
- `appid` — обязательный идентификационный ключ, предоставляемый бесплатно при регистрации на сайте **openweathermap.org**.

Значения параметров с символом = добавляются к URL-адресу сайта **openweathermap.org** после символа? и отделяются друг от друга знаком `&`.

Ответ сайта **openweathermap.org** может быть доставлен в виде XML-файла, который можно сохранить в папку */htdocs* веб-сервера с помощью PHP-функции `asXml()`, указав имя файла. Полученный файл можно рассмотреть в веб-браузере, чтобы разобраться в структуризации извлекаемых данных, именах XML-элементов и их атрибутов.

1. Начните PHP-сценарий с присвоения строковых значений четырем переменным, которые будут использоваться для определения параметров ответа сервера.

params.php

```
<?php
$city = 'Moscow';
$units = 'metric';
$mode = 'xml';
$id = '28df3c64f387cf55026e0de3fb8beaa4';
```

2. Затем добавьте инструкцию для компоновки строки запроса с использованием указанных значений параметров.

```
$request =
'http://api.openweathermap.org/data/2.5/weather?'.
'q='.$city.'&units='.$units.'&mode='.$mode.'&APPID='.$id;
```

3. Теперь добавьте инструкцию, загружающую XML-данные с помощью запрашиваемых строковых параметров или завершающую сценарий в случае сбоя.

```
$xml = simplexml_load_file($request)
or die('Невозможно загрузить данные!');
```

4. Если ответ получен, контент следует записать в XML-файл и сохранить его в каталог */htdocs* веб-сервера.

params.xml

```
try
{
 $xml->asXml('params.xml');
 // Сюда помещаются инструкции (шаг 5).
}
catch(Exception $e)
{
 echo 'Ошибка: '.$e->getMessage();
}
```

5. Добавьте инструкцию, перенаправляющую браузер на только что созданный XML-документ.

```
header('Location: params.xml');
```

На заметку

Ответ будет содержать данные о погоде с учетом указанных условий.

6. Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *params.php*, а затем откройте эту страницу в браузере через протокол HTTP, чтобы увидеть содержимое XML-файла, записанного после извлечения данных с сервера.

localhost

localhost/params.xml

```
  <?xml version="1.0" encoding="UTF-8"?>
- <current>
  - <city id="524901" name="Moscow">
      <coord lat="55.75" lon="37.62"/>
      <country>RU</country>
      <sun set="2017-04-12T16:30:42" rise="2017-04-12T02:30:53"/>
    </city>
    <temperature unit="metric" max="7" min="4" value="5.76"/>
    <humidity unit="%" value="86"/>
    <pressure unit="hPa" value="1002"/>
  - <wind>
      <speed name="Moderate breeze" value="6"/>
      <gusts/>
      <direction name="West-southwest" value="250" code="WSW"/>
    </wind>
    <clouds name="broken clouds" value="75"/>
    <visibility value="9000"/>
    <precipitation mode="no"/>
    <weather value="broken clouds" icon="04d" number="803"/>
    <lastupdate value="2017-04-12T11:30:00"/>
</current>
```

На рисунке рамкой выделены наиболее интересные значения. Они могут быть извлечены с помощью имен элемента и атрибута.

Выбор компонентов

Отдельные компоненты данных ответа, полученного с сервера OpenWeatherMap на предыдущей странице, могут быть динамически встроены в XML-приложение с PHP-формой, поддерживающей сохранение.

1. Начните PHP-сценарий с загрузки файла шапки страницы, чтобы встроить на страницу графический баннер.

```
<?php
include('includes/weather-header.html');
```

weather.php

2. Затем добавьте условную конструкцию, проверяющую, что страница открыта после передачи данных веб-формы.

```
if($_SERVER['REQUEST_METHOD'] == 'POST')
{
 // Сюда вставляются инструкции (шаги 3-6).
}
```

3. Внутри кода условной конструкции присвойте строковые значения параметров четырем переменным.

```
$city = (empty($_POST['city']))? "Moscow"
: $_POST['city'];
$units = 'metric';
$mode = 'xml';
$id = '28df3c64f387cf55026e0de3fb8beaa4';
```

Совет

Обратите внимание на то, что значение параметра `city` по умолчанию назначается в случае, если форма передается пользователем без введенного названия города.

4. Затем внутри условной конструкции добавьте инструкцию, создающую строку запроса с использованием указанных значений параметров.

```
$request =
'http://api.openweathermap.org/data/2.5/weather?'.
'q='.$city.'&units='.$units.'&mode='.$mode.'&APPID='.$id;
```

5. Теперь внутри условной конструкции добавьте инструкцию, загружающую XML-данные на основе указанных параметров или завершающую сценарий в случае сбоя.

```
$xml = simplexml_load_file($request)
or die('Невозможно загрузить данные!');
```

6. И, наконец, добавьте инструкции для вывода извлеченных данных.

```
$icon = 'http://openweathermap.org/img/w/'.
     $xml->weather['icon'].'.png';
echo '<h1>Сегодня в городе '.$xml->city['name'];
echo ': '.$xml->weather[''].'<img src='.$icon.'></h1>';
echo '<ul><li>Температура: '.$xml->temperature['value'];
echo '&deg; '.$xml->temperature[''].' C';
echo '<li>Ветер: '.$xml->wind->speed['value'].' м/с';
echo '<li>Влажность: '.$xml->humidity['value'].'&#37;</ul>';
```

Совет

Значением атрибута `icon` является имя графического файла с расширением PNG. При конкатенации пути и расширения файла `.png` на страницу выводится соответствующее текущей погоде изображение.

7. После кода блока проверки добавьте инструкции, создающие веб-форму.

```
echo '<form method="POST" action="weather.php">';
echo '<fieldset><legend>Введите название города</legend>';
echo '<input type="text" name="city">';
echo '<input type="submit"
     name="submission" value = "Прогноз">';
echo '</fieldset></form>';
```

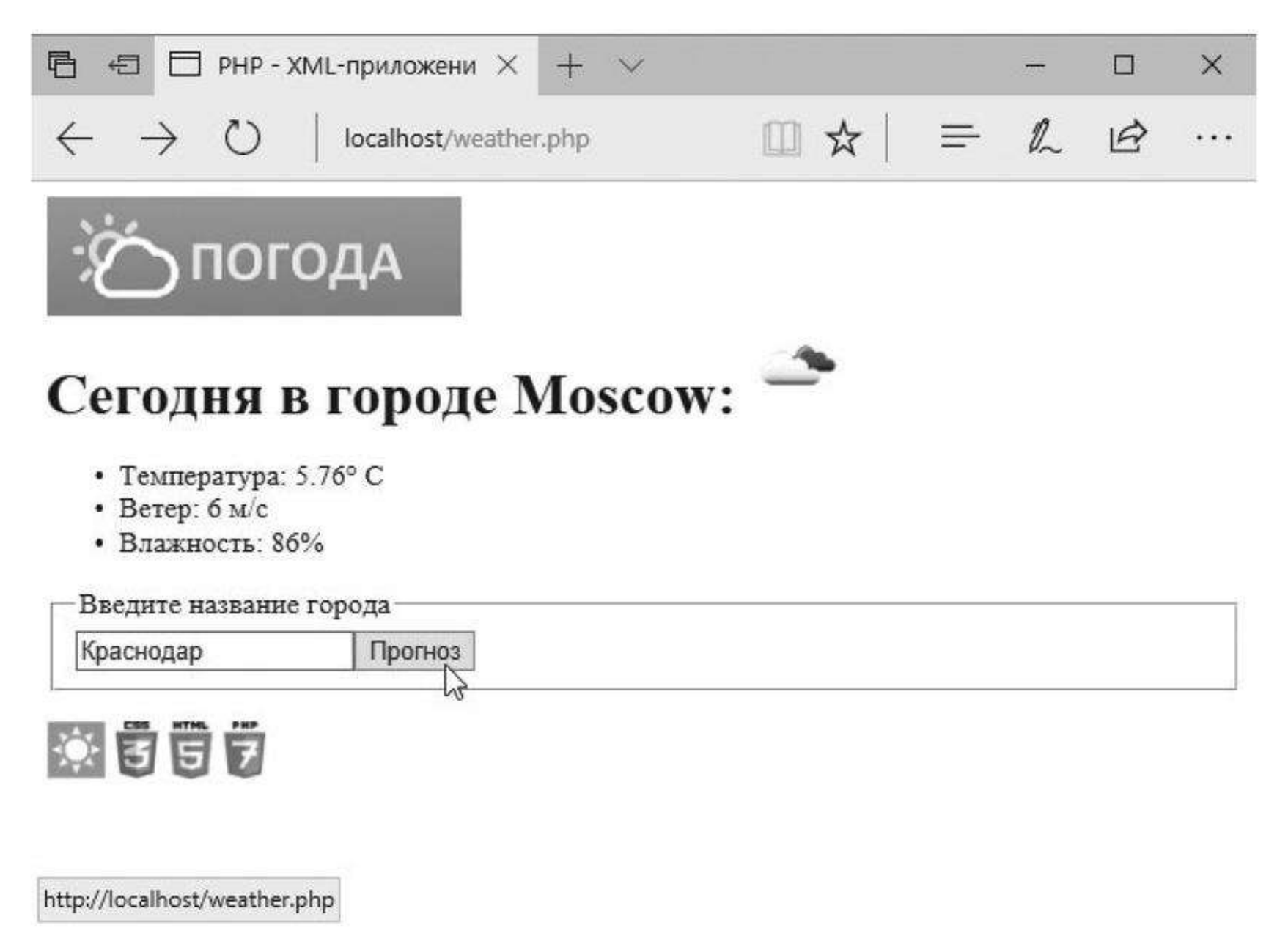
На заметку

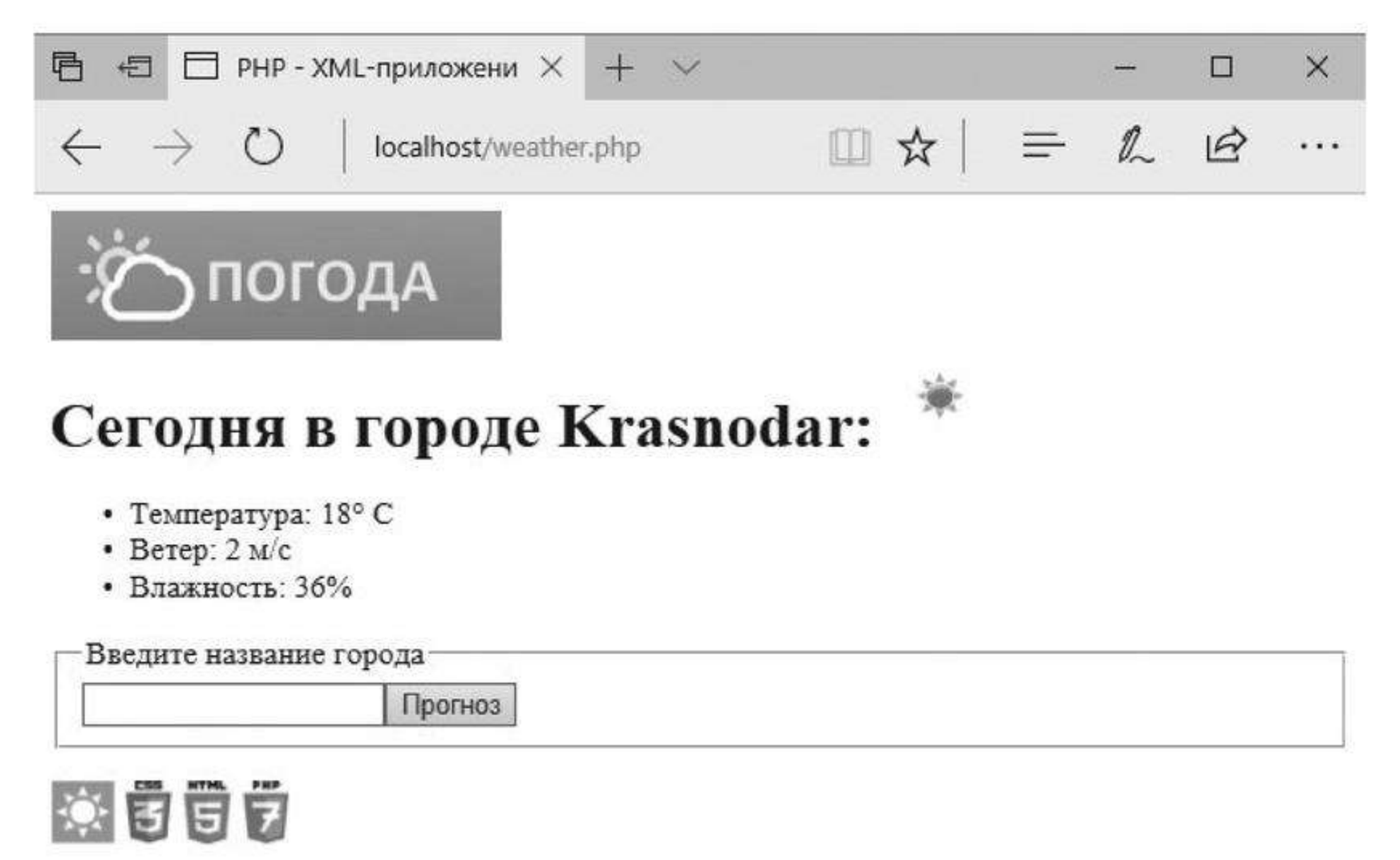
Атрибуту `action` присвоено значение с именем того же PHP-файла, чтобы форма поддерживала сохранение данных.

8. Наконец, добавьте в сценарий код страницы с шапкой файла для вывода изображения «Погода» в верхней части страницы.

```
include('includes/weather-footer.html');
```

9 Сохраните документ в каталоге */htdocs* вашего веб-сервера под именем *weather.php*, а затем откройте эту страницу в браузере через протокол HTTP и нажмите кнопку «Прогноз», чтобы увидеть погоду в городе, установленном по умолчанию (в Москве). Введите название другого города и вновь нажмите кнопку «Прогноз», чтобы увидеть погоду в указанном городе.





Заключение

- Все XML-элементы должны иметь открывающий и закрывающий теги и корректно вкладываться для поддержки правильно отформатированного кода.
- PHP-движок содержит XML-анализатор (парсер) под названием SimpleXML, который позволяет загрузить в память XML-документ в виде иерархической древовидной структуры.
- Функция `simplexml_load_file()` создает структуры данных SimpleXMLElement-объектов.
- Структура данных SimpleXML-объекта описывает отношения родитель — потомок между внешними и внутренними XML-элементами.
- Каждый член структуры SimpleXML-данных является узлом.
- Узлы верхнего уровня представляют собой прямых дочерних потомков самого SimpleXML-объекта.
- На узлы можно ссылаться используя оператор `->` для адресации к каждому поколению иерархической древовидной структуры.
- Метод `children()` возвращает все дочерние узлы и может использоваться для циклического обхода узлов дерева с целью извлечения содержимого каждого узла.
- На атрибуты можно ссылаться по имени атрибута, указанного в одиночных кавычках и квадратных скобках сразу после имени узла.
- RSS-каналы передают регулярно обновляемые данные в формате XML.
- Язык RSS — своего рода «диалект» XML, в котором все документы должны содержать элемент верхнего уровня `rss` со вложенным в него элементом `channel`.
- Веб-сервисы предоставляют постоянный URL-адрес или API-интерфейс, позволяющий задавать параметры в URL-запросе.
- Значения параметров присваиваются с помощью символа = и добавляются к URL-адресу после символа `?`. Несколько значений параметров отделяются друг от друга символом `&`.
- SimpleXML-объект может быть записан в XML-файл на стороне сервера с помощью функции `asXml()`.
- Отдельные компоненты SimpleXML-объекта могут быть динамически встроены в приложение на языке PHP.

Предметный указатель

!, логическое «НЕ», оператор, 63
!=, не равно, оператор, 59
!==, не идентично, оператор, 59
", определения строк, оператор, 31
$, имени переменной, оператор, 31
%, деления по модулю, оператор, 57
&&, логическое «И», оператор, 63
&, И, побитовый оператор, 65
(), аргументов, оператор, 95
(), приоритета, оператор, 57
**, возведения в степень, оператор, 57
*, умножения, оператор, 57
–, вычитания, оператор, 57
—, декремента, оператор, 70
', строки, оператор, 31
., конкатенации, оператор, 33
/, деления, оператор, 57
?:, условный (тернарный), оператор, 61
[], элемента массива, оператор, 35
\, экранирования, оператор, 33
^, исключающее ИЛИ, побитовый оператор, 65
{}, блока функции, оператор, 91
{}, имен ключей, оператор, 43
|, ИЛИ, побитовый оператор, 65
||, логическое «ИЛИ», оператор, 63
~, отрицание, побитовый оператор, 65
+, сложения, оператор, 59
++, инкремента, оператор, 70
<, меньше, оператор, 59
<<, сдвиг влево, побитовый оператор, 65
<=, меньше или равно, оператор, 59
<=>, «спейсшип», оператор, 59
<>, не равно, оператор, 59
=, присваивания, оператор, 31
==, равно, оператор, 59
===, идентично, оператор, 59
=>, массива, оператор, 35
>, больше, оператор, 59
->, объекта, оператор, 233
>=, больше или равно, оператор, 59
>>, сдвиг вправо, побитовый оператор, 65
«И», логический оператор, &&, 63
«ИЛИ», логический оператор, ||, 63
«НЕ», логический оператор,!, 63
«спейсшип», оператор, <=>, 59
Abyss Web Server, 11
  настройка под PHP, 16
  установка, 12
action, атрибут (HTML), 144, 178
array(), функция, 35
arsort(), функция, 37
ASCII-кода значение, 110
asort(), функция, 37
asXml(), функция, 239
break, ключевое слово, 78, 87, 89
case, ключевое слово, 78
connect_db.php, сценарий, 224
continue, ключевое слово, 87, 89
cookie-файлы, 190
  запись данных, 191
  извлечение данных, 148
count(), функция, 29
ctype_alnum(), функция, 192
ctype_alpha(), функция, 204
Current Weather Data API, 239
default, ключевое слово, 78
define(), функция, 47
die(), функция, 214
do while, цикл, 83, 85
echo, инструкция, 18
else, ключевое слово, 76, 89
elseif, ключевое слово, 76

empty(), функция, 168
Exception, класс
  getFile(), метод, 160
  getLine(), метод, 160
  getMessage(), метод, 160
exit(), функция, 214
explode(), функция, 37
extension.php, файл, 16
FALSE, логическое (булево) значение, 61, 63, 75
fclose(), функция, 147
feof(), функция, 149
fgetc(), функция, 150
fgets(), функция, 149
filesize(), функция, 147
fopen(), функция, 147
for, цикл, 83
foreach, цикл, 35, 83
form, HTML-элемент, 164
fread(), функция, 147
GET-запрос, метод, 165, 178
header(), функция, 191, 204
htdocs, иерархия, 215
htdocs, каталог, 13
HTML-сущности
  htmlentities(), htmlspecialchars(), 123
HTML-форма, 164
  атрибут action, 164
  атрибут method, 164
  скрытые поля, 176
  с сохранением данных, 180
HTTP, протокол, 11
if else, инструкция, 76
if, ключевое слово, 76
implode(), функция, 37
include(), функция, 140
includes, каталог, 141
is_array(), функция, 28
isset(), функция, 166, 180, 190
krsort(), функция, 37
ksort(), функция, 37
localhost, домен (127.0.0.1), 13, 15
ltrim(), функция, 116
md5(), функция, 191
method, атрибут (HTML), 164
mysqli_connect(), функция, 214
mysqli_connect_error(), функция, 214
mysqli_fetch_array(), функция, 218
mysqli_num_rows(), функция, 218
mysqli_query(), функция, 218
mysqli_set_charset(), функция, 214
NULL, значение, 166
OpenWeatherMap, 239
ord(), функция, 110
PHP
  внедрение кода, 18
  обзор, 10
  переменные, 31
  теги <?php?>, 18
  установка, 14
  файловое расширение.p*hp*, 17
  функции, 91
  чувствительность к регистру, 31
phpinfo(), функция, 17
POST-запрос, метод, 164, 178
preg_match(), функция, 168
private, спецификатор доступа, 127
protected, спецификатор доступа, 127
public, спецификатор доступа, 127
REQUEST_METHOD, элемент, 178
require(), функция, 184, 214
rsort(), функция, 37
RSS-канал, 237
rtrim(), функция, 216
set_error_handler(), функция, 155
setcookie(), функция, 190, 191
simplexml_load_file(), функция, 131
SimpleXMLElement-объект
  children(), метод, 134, 136
  атрибуты узла, 135
  узлы, 133
SimpleXML-анализатор, 131
sort(), функция, 37
strcasecmp(), функция, 110
strchr(), функция, 112
strcmp(), функция, 110
stristr(), функция, 112
strlen(), функция, 110
strncasecmp(), функция, 110
strncmp(), функция, 110
strpad(), функция, 116
strpos(), функция, 112

strrchr(), функция, 112
strrev(), функция, 116
strrpos(), функция, 112
strrpt(), функция, 116
strstr(), функция, 112
strtok(), функция, 116
strtolower(), функция, 115
strtotime(), функция, 121
strtoupper(), функция, 115
substr(), функция, 114
substr_count(), функция, 114
substr_replace(), функция, 114
switch, ключевое слово, 78
throw, ключевое слово, 160
trigger_error(), функция, 155
trim(), функция, 116
TRUE, логическое (булево) значение, 61, 63, 72
try catch, блок, 160
ucfirst(), функция, 115
ucword(), функция, 115
UTF-8, кодировка, 19
var_dump(), функция, 201, 207, 231
while, цикл, 83, 85
XML (расширяемый язык разметки), 231
XML-приложение, 241
аргументы, 95
арифметические операторы, 57
  +, сложения, 57
  /, деления, 57
  **, возведения в степень, 57
  %, деления по модулю, 57
  *, умножения, 57
  –, вычитания, 57
базовый класс, 138
байт, 8 бит, 65
больше, чем, операторы
  >, больше, 59
  >=, больше или равно, 59
веб-сервис, 231
  API-интерфейс, 239
  RSS-лента, 237
веб-форма *см.* HTML-форма
вложенный цикл, 83
возведения в степень, оператор, **, 57
вычитания, оператор, –, 57
геттер, метод класса, 129
гиперссылка
  присоединение данных, 186
глобальная область видимости, 99
группировка выражений, 57
дата и время
  date_default_timezone_get(), функция, 121
  date_default_timezone_set(), функция, 121
  date(), функция, 121
двойные кавычки, 31
двоичное число, 65
декремента, оператор, —, 70
деления по модулю, оператор, %, 57
деления, оператор, /, 57
деструктор, метод, 136
длина строки, 110
доступа методы, сеттер и геттер, 129
доступа спецификаторы, 127
  private, 127
  protected, 127
  public, 127
дочерний узел (потомок), 233
единого документа формы передача, 178
запись файлов
  fwrite(), 151
Значение атрибута, 180
идентично, оператор, ===, 59
извлечение классов, 138
именование, 31, 95
  деструктор класса, ~, 136
  доступа, методы, 129
  классов, 127
инкапсуляция, 128
инкремента, оператор, ++, 70
инструкций правила, 21
исключающее «ИЛИ», оператор xor, 63
итерация цикла, 83
кавычки двойные, 31
каскадные таблицы стилей (CSS), 185
класс
  базовый, 138
  извлечение, 138
  метод, 127
  объявление, 127
  сеттер и геттер, 129
  член, 127
  экземпляр, 129
ключ массива, 35

кодировка
HTML-сущности, 123
UTF-8, 19
комментирование, правила, 21
конкатенация, 33
константа, 47
конструктор, метод, 136
логические (булевы) значения
TRUE FALSE, 63
логические операторы
and, логическое «И», 63
&&, логическое «И», 63
!, логическое «НЕ», 63
||, логическое «ИЛИ», 63
or, логическое «ИЛИ», 63
xor, исключающее «ИЛИ», 63
локальная область видимости, 99
массив
индекс, 43
многомерный, 43
переменная, 35
сортировка, 37
элементы, 43
меньше чем, операторы
<, меньше, 59
<=, меньше или равно, 59
методы класса, 127
сеттер и геттер, 129
младший двоичный разряд (LSB), 65
многомерные массивы, 43
название страницы, 184
наследование, 138
не идентично, оператор, ! ==, 59
не равно, оператор, !=, 59
не равно, оператор, <>, 59
нулевая индексация, 43
облако (сторона сервера), 11
обработка исключений
try catch, блок, 160
обработка ошибок
custom error_handler(), функция, 155
try catch, блок, 160
обратный слеш, символ экранирования, \, 33
объединения со значением NULL, оператор, 61
объединения со значением NULL, оператор, 61
объект, 127
объекта, оператор, ->, 233
объектно ориентированное программирование (ООП), 127
инкапсуляция, 127
наследование, 138
полиморфизм, 140
объявление
извлечение классов, 138
классов, 127
одиночные кавычки, 33
операторы
арифметические, + - * /% **, 57
логические, && and || or xor!, 63
побитовые, & | ~ ^ << >>, 65
сравнения, ==!= <> ===!== > < >= <=?:, 59
определение строк
вывод переменной в составе строки, 31, 33
параметры, 95
переменная, 31
область видимости, 99
побитовые операторы
|, ИЛИ, 65
&, И, 65
~, отрицание, 65
^, исключающее ИЛИ, 65
<<, сдвиг влево, 65
>>, сдвиг вправо, 65
подвал страницы, встраивание, 184
подключение к базе данных, 214
полиморфизм, 140
правила
теги, инструкции, комментарии, 21
присваивания, оператор, =, 31
прописные имена констан, 47
равно, оператор, ==, 59
регистр, чувствительность кода, 31, 91
регулярное выражение, 168
серверная консоль, 13
сессии
session_destroy(), функция, 207
session_id(), функция, 207
session_start(), функция, 203
unset(), функция, 207
получение данных, 205
установка данных, 204
сеттер, метод класса, 129
сложения, оператор, +, 57

сокрытие данные, 127
сообщений публикация (форум), 232
сравнения, операторы
  ==, равно, 59
  >, больше, 59
  >=, больше или равно, 59
  ===, идентично, 59
  !=, не равно, 59
  <>, не равно, 59
  <, меньше, 59
  <=, меньше или равно, 59
  !==, не идентично, 59
  <=>, «спейсшип», 59
старший двоичный разряд (MSB), 65
строка
  обрезка, 116
  поиск, 112
  смена регистра, 115
  сравнение, 110
  форматирование, 116
строки, 33
суперглобальные переменные
  $_COOKIE, 194, 201
  $_GET, массив, 186
  $_POST, 203
  $_POST, массив, 144, 180, 186
  $_SERVER, массив, 178
  $_SESSION, 203, 204
  $_SESSION, массив, 186
таблица базы данных, 116
таблица стилей, 185
теги, 20
текст, 33
текста добавление, 153
тернарный, оператор, ?:, 59
типы данных, 31
умножения, оператор, *, 57
унарный, оператор, 63
условное ветвление, 76
  case, break, default, ключевые слова, 78, 89
  if, else, ключевые слова, 74, 89
  switch, ключевое слово, 78
условный, оператор, ?:, 61
файла режимы, 147
файловый поток, 147
фигурные скобки
  блок функции, 91
  именованные элементы массива, 43
формы данные, 144
формы с сохранением данных, 180
форума база данных, 216
форума страница, 218
функция
  передача аргументов, 95
  создание, 91
циклы
  break, ключевое слово, 87
  continue, ключевое слово, 88
  do, ключевое слово, 85
  foreach, ключевое слово, 83
  for, ключевое слово, 83
  while, ключевое слово, 85
члены класса, 126
шапка страницы, встраивание, 184
экземпляр класса, 127
  группа объектов, 134
экранирование, символ обратного слеша, \, 33
элемент, 35
язык структурированных запросов (SQL), 216

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

ПРОГРАММИРОВАНИЕ — ЭТО ПРОСТО

МакГрат Майк

PHP7 ДЛЯ НАЧИНАЮЩИХ С ПОШАГОВЫМИ ИНСТРУКЦИЯМИ

(орыс тілінде)

Директор редакции *Е. Капьёв*
Ответственный редактор *Е. Истомина*
Художественный редактор *А. Шуклин*

В оформлении обложки использован элемент дизайна:
Plasteed / Shutterstock.com
Используется по лицензии от Shutterstock.com

ООО «Издательство «Э»
123308, Москва, ул. Зорге, д. 1. Тел.: 8 (495) 411-68-86.

Өндіруші: «Э» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел.: 8 (495) 411-68-86.
Тауар белгісі: «Э»
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) 251-59-89/90/91/92, факс: 8 (727) 251 58 12 вн. 107.
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта Өндіруші «Э»

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить на сайте Издательства «Э»

Өндірген мемлекет: Ресей
Сертификация қарастырылмаған

Подписано в печать 19.10.2017. Формат 70x100 1/16.
Печать офсетная. Усл. печ. л. 20,74.
Тираж экз. Заказ

ISBN 978-5-699-98594-4

9 785699 985944 >

В электронном виде книги издательства вы можете купить на **www.litres.ru**

ЛитРес:
один клик до книг

Оптовая торговля книгами Издательства «Э»:
142700, Московская обл., Ленинский р-н, г. Видное,
Белокаменное ш., д. 1, многоканальный тел.: 411-50-74.

По вопросам приобретения книг Издательства «Э» зарубежными оптовыми покупателями обращаться в отдел зарубежных продаж
International Sales: International wholesale customers should contact Foreign Sales Department for their orders.

По вопросам заказа книг корпоративным клиентам, в том числе в специальном оформлении, *обращаться по тел.: +7 (495) 411-68-59, доб. 2261.*

Оптовая торговля бумажно-беловыми и канцелярскими товарами для школы и офиса:
142702, Московская обл., Ленинский р-н, г. Видное-2,
Белокаменное ш., д. 1, а/я 5. Тел./факс: +7 (495) 745-28-87 (многоканальный).

Полный ассортимент книг издательства для оптовых покупателей:
Москва. Адрес: 142701, Московская область, Ленинский р-н,
г. Видное, Белокаменное шоссе, д. 1. Телефон: +7 (495) 411-50-74.
Нижний Новгород. Филиал в Нижнем Новгороде. Адрес: 603094,
г. Нижний Новгород, улица Карпинского, дом 29, бизнес-парк «Грин Плаза».
Телефон: +7 (831) 216-15-91 (92, 93, 94).
Санкт-Петербург. ООО «СЗКО». Адрес: 192029, г. Санкт-Петербург, пр. Обуховской Обороны,
д. 84, лит. «Е». Телефон: +7 (812) 365-46-03 / 04. **E-mail:** server@szko.ru
Екатеринбург. Филиал в г. Екатеринбурге. Адрес: 620024,
г. Екатеринбург, ул. Новинская, д. 2щ. Телефон: +7 (343) 272-72-01 (02/03/04/05/06/08).
Самара. Филиал в г. Самаре. Адрес: 443052, г. Самара, пр-т Кирова, д. 75/1, лит. «Е».
Телефон: +7(846)207-55-50. **E-mail:** RDC-samara@mail.ru
Ростов-на-Дону. Филиал в г. Ростове-на-Дону. Адрес: 344023,
г. Ростов-на-Дону, ул. Страны Советов, 44 А. Телефон: +7(863) 303-62-10.
Центр оптово-розничных продаж Cash&Carry в г. Ростове-на-Дону. Адрес: 344023,
г. Ростов-на-Дону, ул. Страны Советов, д.44 В. Телефон: (863) 303-62-10. Режим работы: с 9-00 до 19-00.
Новосибирск. Филиал в г. Новосибирске. Адрес: 630015,
г. Новосибирск, Комбинатский пер., д. 3. Телефон: +7(383) 289-91-42.
Хабаровск. Филиал РДЦ Новосибирск в Хабаровске. Адрес: 680000, г. Хабаровск,
пер.Дзержинского, д.24, литера Б, офис 1. Телефон: +7(4212) 910-120.
Тюмень. Филиал в г. Тюмени. Центр оптово-розничных продаж Cash&Carry в г. Тюмени.
Адрес: 625022, г. Тюмень, ул. Алебашевская, 9А (ТЦ Перестройка+).
Телефон: +7 (3452) 21-53-96/ 97/ 98.
Краснодар. Обособленное подразделение в г. Краснодаре
Центр оптово-розничных продаж Cash&Carry в г. Краснодаре
Адрес: 350018, г. Краснодар, ул. Сормовская, д. 7, лит. «Г». Телефон: (861) 234-43-01(02).
Республика Беларусь. Центр оптово-розничных продаж Cash&Carry в г.Минске. Адрес: 220014,
Республика Беларусь, г. Минск, проспект Жукова, 44, пом. 1-17, ТЦ «Outleto».
Телефон: +375 17 251-40-23; +375 44 581-81-92. Режим работы: с 10-00 до 22-00.
Казахстан. РДЦ Алматы. Адрес: 050039, г. Алматы, ул.Домбровского, 3 «А».
Телефон: +7 (727) 251-58-12, 251-59-90 (91,92,99).
Украина. ООО «Форс Украина». Адрес: 04073 г. Киев, ул.Вербовая, 17а.
Телефон: +38 (044) 290-99-44. **E-mail:** sales@forsukraine.com

Полный ассортимент продукции Издательства «Э» можно приобрести в магазинах «Новый книжный» и «Читай-город».
Телефон единой справочной: 8 (800) 444-8-444. Звонок по России бесплатный.

В Санкт-Петербурге: в магазине «Парк Культуры и Чтения БУКВОЕД», Невский пр-т, д.46.
Тел.: +7(812)601-0-601, www.bookvoed.ru

Розничная продажа книг с доставкой по всему миру. Тел.: +7 (495) 745-89-14.

BOOK24.RU
ИНТЕРНЕТ-МАГАЗИН
ИНТЕРНЕТ-МАГАЗИН
BOOK24.RU

КОГДА ВЫ ДАРИТЕ КНИГУ, ВЫ ДАРИТЕ ЦЕЛЫЙ МИР

ХОТИТЕ ЗНАТЬ БОЛЬШЕ?

Заходите на сайт:
https://eksmo.ru/b2b/

Звоните по телефону:
+7 495 411-68-59, доб. 2261

ВАШ ЛОГОТИП НА ОБЛОЖКЕ

ВАШ ЛОГОТИП НА КОРЕШКЕ

ОБРАЩЕНИЕ К КЛИЕНТАМ НА ОБЛОЖКЕ

Выучите PHP7 с нуля!

В ваших руках – путеводитель по миру PHP7, он содержит всю необходимую новичку информацию об этом языке – от подготовки к работе до разработки веб-форм, подключения баз данных и веб-служб.

Из этой книги вы узнаете:

- С чего начать?
- Как работает PHP
- Что такое переменные, массивы и константы
- Как использовать операторы, конструкции и циклы

И многое другое.

Преимущества этой книги:

- Простой язык
- Понятные инструкции
- Каждый шаг проиллюстрирован
- Вы освоите веб-разработку

Замечательная книга для начинающих веб-разработчиков, которая может использоваться как в качестве самоучителя, так и в качестве справочника. Ее главное достоинство – краткость, ясность и отсутствие лишней для новичков информации.

ISBN 978-5-699-98594-4
9 785699 985944 >

Игорь Борисов, ведущий преподаватель
в области программирования
УЦ «Специалист» при МГТУ им. Н. Э. Баумана