Кириченко А.В., Никольский А.П., Дубовик Е.В.

WEB НА ПРАКТИКЕ

CSS, HTML, JavaScript, MySQL, PHP

для fullstack-разработчиков

НиТ
издательство

Кириченко А.В., Никольский А.П., Дубовик Е.В.

WEB НА ПРАКТИКЕ

CSS, HTML, JavaScript, MySQL, PHP

для fullstack-разработчиков

"Наука и Техника"

г. Санкт-Петербург

УДК 004.42
ББК 32.973

Кириченко А.В., Никольский А.П., Дубовик Е.В.

Web на практике. CSS, HTML, JavaScript, MySQL, PHP для fullstack-разработчиков — СПб.: Наука и Техника, 2021. — 432 с., ил.

ISBN 978-5-94387-271-6

Разработка многофункционального сайта, как правило, требует нескольких разных специалистов, но в данной книге мы расскажем, как все сделать самому! Fullstack-разработчик — это разработчик, который обладает знаниями всех технологий (полным стеком) для создания полноценных многофункциональных веб-сайтов. Данная книга посвящена Fullstack-разработке сайта. В книге рассмотрен полный цикл создания полноценных сайтов и Интернет-порталов:

- Идея или постановка целей и задач сайта.
- Создание макета дизайна сайта.
- Верстка. Создание frontend'а.
- Программирование backend'а.
- Базовое наполнение контентом.
- Разворачивание на хостинге.

В книге приведено описание всех ключевых технологий web-разработки (HTML5, CSS3, JavaScript, PHP, MySQL), знание которых необходимо fullstack-разработчикам.

Также приведен и разобран реальный пример разработки полноценного образовательного Интернет-портала (его фронтенда и бэкенда), исходные коды которого можно скачать с сайта издательства.

ISBN 978-5-94387-271-6

Контактные телефоны издательства:
(812) 412 70 26

Официальный сайт: www.nit.com.ru

Содержание

Глава 1.

Общая методика Fullstack-разработки сайта

Данная книга посвящена Fullstack-разработке сайта. Данная книга и практический пример написаны коллективом (сотрудниками) ООО "Цифровые Бизнес-Платформы". Книга состоит из нескольких глав, в которых дается описание технологий полного стека web-разработки. При их рассмотрении приводятся либо небольшие примеры, либо фрагменты большого практического примера - проекта образовательного веб-портала. Скриншоты портала можно посмотреть на страницах 319-322. Полностью данному проекту посвящены главы 7 (разработка фронтенда) и 8 (разработка бэкэнда). К книге, на сайте издательства прилагается программный код примера. В главе 9 описано, как развернуть практический пример (или ваш сайт) на хостинге. Обратите, пожалуйста, внимание, что код примера распространяется в учебных целях (не для коммерческого использования), а контентное оформление (изображения и т.п.) либо исключено из кода, либо принадлежит их правообладателям.

В этой части мы поговорим об этапах разработки сайта, о том, какие технологии вам понадобятся и о том, как стать Fullstack-разработчиком. В следующих частях будет приведено описание ключевых технологий web-разработки (HTML5, CSS3, JavaScript, PHP, MySQL), а также будет приведен и разобран реальный пример сайта (его фронтенда и бэкенда), исходные коды которого можно скачать с сайта http://nit.com.ru.

1.1. Этапы разработки сайта

Разработка сайта состоит из следующих этапов:

1. Идея или постановка целей и задач сайта.
2. Создание технического задания (далее – ТЗ).
3. Создание макета дизайна сайта.
4. Верстка.
5. Программирование.
6. Базовое наполнение контентом.
7. Тестирование.
8. Сдача готового сайта клиенту.

Как правило, для практически каждого из этих этапов должен быть отдельный специалист. На первом этапе пригодятся знания маркетолога, дизайнер и верстальщик должны реализовать третий и четвертый этапы. Пятый этап – за программистом. Контент-менеджер будет заниматься шестым этапом, тестировщик – седьмым. Руководитель – восьмым. Но поскольку речь идет о Fullstack-разработчике – эдаком универсальном солдате, которых в мире фриланса очень много, ему придется самостоятельно заниматься каждым этапом и быть чуть-чуть маркетологом, верстальщиком, дизайнером, программистом и тестировщиком. Казалось бы, все это сложно, но невыполнимых задач нет! Рассмотрим эти этапы подробнее.

1.1.1. Идея

Все начинается с идеи. Заказчик (или вы сами, если создаете сайт для себя) должен четко понимать, для чего сайт нужен, какие функции он должен выполнять, и что должно получиться на выходе. Далеко не каждый заказчик может самостоятельно определить его цели и задачи. Разработчик или его представитель/помощник должны помочь определиться с целями и функционалом. В этом прямая заинтересованность разработчика – он должен максимально точно понимать, что хочет заказчик, чтобы не получилось, что заказчик хотел одно, а разработчик сделал другое. Такие ситуации, к сожалению, бывают чаще, чем можно себе представить. Они приводят к недоразумениям и спорным ситуациям. Затем все равно приходится переделывать проделанную работу, а это время и нервы. Во избежание таких ситуаций нужно четко понимать, чего хочет заказчик. Если заказчик сам не знает, что он хочет, нужно помочь ему определиться.

Для этого нужно произвести подготовительный этап: проанализировать, какие задачи должен решать сайт, и вообще, зачем сайт нужен заказчику. Будет ли это Интернет-магазин или же сайт-визитка? А может это будет лидогенератор для Интернет-магазина – сайт, который будет заманивать потенциальных клиентов в Интернет-магазин – такие сайты набирают популярность в последнее время и способны в несколько раз повысить продажи. Все это оговаривается с заказчиком. Он должен сам понимать зачем ему сайт, в чем он поможет его компании и с чем справиться не сможет.

После постановки задачи нужно определиться с целевой аудиторией. Нужно понимать, для кого создается ресурс, что может предложить заказчик, и как на его предложение отреагирует целевая аудитория. От целевой аудитории зависит структура сайта, контент, дизайн и функционал.

В заключение этого этапа проводится маркетинговый анализ, который включает анализ конкурентов, сезонности товаров и услуг заказчика, определяются всевозможные "фичи" (от англ. *feature* – особенность), способные привлечь внимание посетителей. Вся эта информация также согласовывается с заказчиком.

Казалось бы, этот этап занимает достаточно много времени, но от того, насколько качественно он будет проделан, зависит вся дальнейшая работа.

1.1.2. Техническое задание

На основании собранной на первом этапе информации составляется техническое задание. Звучит абсурдно, но техническое задание для себя же составляет сам разработчик. По-хорошему, заказчик должен прийти к разработчику с уже готовым техническим заданием. Но на практике заказчик не всегда приходит даже с четким пониманием задач сайта, не говоря уже о техническом задании.

Поэтому техническое задание составляется вместе с клиентом. Данный документ включает детальные характеристики ресурса:

- Пожелания по дизайну (цветовое оформление, применение фирменного стиля или его разработка, соотношение графических элементов к текстовым);
- Семантическое ядро;
- Структуру сайта, количество страниц, категорий, блоков;
- Функционал сайта (только стандартный или разработка дополнительных модулей, подробное их описание и цели);
- Применяемые технологии;
- Технические требования к ресурсу.

Техническое задание – это основа, на которую полагается каждый специалист, участвующий в разработке. Поэтому участие заказчика в составлении ТЗ – необходимо. ТЗ оговаривается и редактируется до тех пор, пока клиент не даст согласие и не подпишет документ. Только после этого проект переходит в стадию создания. Важно, чтобы заказчик подписал техническое задание. Если возникнет ЧП и заказчик будет говорить, что разработчик создал не то, что нужно, рассудить сможет только техническое задание.

Если в нем оговорено, что кнопки на сайте должны быть зелеными, а заказчик утверждает, что хотел красные кнопки – всегда можно прикрыться техническим заданием. В этой книге мы не будем учить вас, как составлять техническое задание. Необходимую информацию всегда можно найти в Интернете.

1.1.3. Создание макета сайта

Разработка макета сайта – один из главных этапов разработки. На данном этапе разрабатывается макет дизайна сайта. А дизайн имеет ключевую важность – ведь это то, что видит посетитель в первую очередь, оценивает и принимает решение, остаться на этом сайте или перейти на другой.

Проблема в том, что создать уникальный сайт в наше время практически нереально. Сайтов уже миллиарды, и вряд ли у вас будет что-то принципиально новое. Но дизайн сайта и его эргономика (удобство работы с ним) все равно имеют значение. Если человеку будет неудобно работать с вашим Интернет-магазином, он купит товар у конкурентов. Конечно, можно поспорить, что решает цена, но если на вашем сайте будет неудобный и запутанный чекаут, клиент просто не сможет оформить заказ, плюнет и закажет данный товар на другом сайте. Да и не всегда можно снижать цену, поскольку есть определенные наценки, гарантирующие бизнесу рентабельность: если снизить цену, упадет доход и бизнес уже не будет рентабельным. Демпингование может помочь только в самом начале – при старте проекта, дальше оно будет только вредить бизнесу.

При создании макета дизайнер, на основании ТЗ, рисует кнопки, баннеры и другие графические элементы. Другими словами, тот прототип, который был создан на первом этапе разработки сайта, получает эстетичный внешний вид, воспроизводится в цветах, выбранными заказчиком. Если компания имеет корпоративный стиль, то дизайн разрабатывается в соответствии с ним. Бывает, сначала разрабатывается фирменный стиль, а после - дизайн на его основе.

При создании макета дизайнер создает дизайн не каждой страницы, а только нескольких основных. Например, если речь идет об Интернет-магазине, то, как правило, рисуются три страницы – главная (витрина), страница каталога (список продуктов) и карточка товара. Если необходимы доработки, дизайнер выполняет их и снова показывает макет заказчику. Дизайн дорабатывается до тех пор, пока он не будет утвержден.

В последнее время широко распространена практика динамического прототипирования при разработке макетов сайта – когда создаются не просто картинки страниц сайта, но их делают кликабельными, то есть смена картинок происходит в зависимости от того, по какой части изображения произошел щелчок мыши. Таким образом, уже на этапе создания макета сайта имитируется его реальная работа. Для этого широко используется такие инструменты как FIGMA, Axure и т.п.

1.1.4. Верстка

Верстка – это процесс "оживления" макета. При создании макета дизайнер использует редактор для работы с изображениями, например, Photoshop или Illustrator, возможно с использованием инструментов динамического прототипирования (Figma, Axure и т.п.). На этапе верстки верстальщик с помощью языка HTML и таблицы стилей CSS переводит дизайн в рабочий проект. Грубо говоря, верстка – это процесс преобразования рисунка, созданного в Photoshop, в HTML-документ, который при его открытии в браузере выглядит максимально подобно рисунку-макету.

На этом же этапе разрабатывается и адаптивная версия сайта, нормально отображающаяся на устройствах с разными размерами экрана. Наличие такой версии очень важно в последнее время, так как если сайт не адаптировать для мобильных устройств, Google и другие поисковики не будут отправлять трафик с мобильных устройств на такой сайт. Еще в 2019 году 52% мирового Интернет-трафика приходилось на мобильные устройства, то есть смартфоны по популярности у конечных пользователей обогнали настольные компьютеры. В 2021 году данный показатель только увеличится.

1.1.5. Программирование

Итак, у нас есть HTML-документ, который сам по себе ничего не делает – он отображает только лишь статический контент, заполненный дизайнером. Как минимум, программисту нужно написать код, который бы получал реальный контент из базы данных и выводил бы его в дизайне сайта. Также нужно разработать ряд сложных функций: подключить возможность опла-

ты с помощью банковских карт, интегрировать сайт с бухучетом (с той же 1С), разработать панель управления сайтом. Все это занимает достаточно много времени, но без этого сайт так и останется макетом, хоть и конвертированным в формат HTML.

Очень часто код, выполняющий все эти сложные задачи, пишется на PHP. Также для некоторых функций вам понадобится знание JavaScript – как правило, когда нужно выполнить какие-то операции прямо в браузере пользователя, без их передачи на сервер. Или наоборот, когда нужно получить данные с сервера (или отправить данные на сервер) без перегрузки страницы. Мы еще поговорим о необходимых знаниях.

1.1.6. Наполнение контентом

Следующий этап – базовое наполнение контентом. Никто не наполняет сайт контентом полностью (если иное не оговорено договором и техническим заданием). Как правило, создаются базовые материалы, добавляется несколько продуктов, загружаются кое-какие фото и видео. В больших командах этим занимается контент-менеджер, в Fullstack-разработке эта задача ложится на плечи программиста. Впрочем, размещение контента позволяет совместить еще одну задачу – тестирование. Если вы добавляете контент и происходит какая-то ошибка (например, не загружается фото статьи), то вы сразу можете ее исправить. Так что подобная организация даже чем-то ускоряет разработку.

1.1.7. Тестирование

После создания сайта происходит его тестирование. Существуют различные методы тестирования, рассмотрение которых выходит за рамки этой книги, поскольку этому вопросу можно посвятить отдельную книгу и даже не одну. Минимальное тестирование заключается хотя бы в том, чтобы проверить, работает ли каждая кнопка сайта так, как от нее требует ТЗ. Если все кнопки работают, контент добавляется и отображается нормально, можно сказать, что сайт готов. Конечно, можно провести еще нагрузочное тестирование и посмотреть, как будет себя вести сайт при большом количестве одновременных посетителей.

1.1.8. Жизнь после сдачи проекта

При сдаче проекта исполнитель передает заказчику все необходимые доступы к сайту. Если не произошло критических ситуаций и разработчик с заказчиком не разругались до этого момента, сотрудничество не заканчивается. Ведь есть еще много задач:

1. Обновление статей, каталога товаров.
2. Обучение работе с сайтом.
3. Поддержка сайта.
4. Продвижение ресурса.

Нужно отметить, что поэтапное создание сайта контролируется заказчиком. По окончании каждого этапа готовый результат демонстрируется клиенту и согласовывается с ним. Это помогает заказчику контролировать работу исполнителя, а исполнителю понимать, доволен ли клиент результатами его работы.

1.2. Стек WEB-технологий fullstack-разработчика разного уровня

Универсальный солдат, он же Fullstack-разработчик, должен быть знаком со множеством технологий. Попробуем разобраться, что же ему понадобится. Мы не будем плодить уровни – можно придумать сколько хочешь уровней, добавляя к каждому из них новое знаний. Мы рассмотрим лишь базовый и продвинутый уровень. Базового уровня хватит для разработки простенького сайта, а продвинутый позволит разработать сайт практически любой сложности.

1.2.1. Базовый уровень

Вам необходимо разбираться в следующих технологиях:

- **HTML и CSS** – ведь нарисованный макет нужно "порезать", то есть создать HTML-документ, который в браузере будет отображаться, как макет, нарисованный в графическом редакторе.
- **Основы языка PHP.** Основной код движка сайта вам придется писать на PHP, поэтому без знаний этого языка программирования, увы, никак.
- **Объектно-ориентированный подход в программировании.** Это основы основ. Вы вряд ли будете писать тот же PHP код полностью с нуля, для ускорения работы вам придется использовать сторонние классы, а они вряд ли напишутся без ООП. В процедурном стиле уже давно никто не программирует. Даже класс подключения к базе данных реализован в ООП-стиле.
- **Концепция MVC (Model View Controller).** Концепция MVC позволяет разделить модель данных (реальные данные, хранящиеся в БД), представление (дизайн сайта) и контроллер (функционал).
- **Основы SQL.** Для работы с базой данных вам придется изучить основы языка SQL.
- **Основы JavaScript.** В современном мире редко какой фронтенд (дизайн) обходится без функций JavaScript, поэтому без понимания хотя бы основ этого языка вам не обойтись.

Все эти технологии описаны в этой книге. Во второй части будет рассмотрен язык разметки HTML, в третьей – CSS, в четвертой – JavaScript. Пятая часть раскрывает тайны программирования на PHP, в ней же объясняется объектно-ориентированное программирование (ООП). В шестой части книги мы поговорим о MySQL.

1.2.2. Продвинутый уровень

Здесь уже понадобятся следующие знания:

- **Система контроля версий** – работать над проектом даже средней сложности без системы контроля версий (даже если работаете над проектом сами) – дело неблагодарное. Ведь можно внести ненароком неправильные изменения в код, и вернуть как было уже не получится. А вот система контроля версий как раз позволяет сделать это: вы всегда сможете откатиться к предыдущей версии кода, сравнить две версии кода и т.д. А когда над проектом работает несколько человек, система контроля версий просто **must have**. Стандартом де-факто является Git – практически все серьезные разработчики используют именно Git (если не считать некоторых софтверных гигантов, которые используют собственные разработки).
- **WebSockets**. Это технология, которая позволяет создавать интерактивное соединение между клиентом (браузером) и сервером для обмена сообщениями в режиме реального времени. Веб-сокеты, в отличие от HTTP, позволяют работать с двунаправленным потоком данных. Предлагаю смотреть на это как на AJAX нового поколения. Основная фишка — это отсутствие необходимости постоянно запрашивать новые данные с сервера. При необходимости сервер сам отправит данные, а браузер их получит.
- **Перекомпиляторы**. Пусть само слово компилятор в контексте интерпретируемых языков программирования звучит странно, но здесь речь идет о программах, которые позволяют собрать все используемые стили в один файл перед заливкой в **production**. Очень популярны перекомпиляторы во фронтенде, так как для бэкенда используются полноценные ООП-языки, и перекомпиляторы используются для других целей – для ускорения и кэширования (пример такого компилятора Zend OPcache).
- **Composer** — пакетный менеджер уровня приложений для языка программирования PHP, который предоставляет средства по управлению зависимостями в PHP-приложении. Простыми словами – это скрипт, написанный на PHP, который скачива-

ет необходимые вам библиотеки и автоматически формирует единственный специальный файл, подключив который, Вы подключите к проекту все скачанные библиотеки и фреймворки.

- **Основные фронтенд библиотеки JavaScript – jQuery и jQueri UI**. Данные библиотеки предоставляют множество полезных при разработке интерфейса пользователя компонентов, в том числе и визуальных. Использование данных библиотек существенно ускорить разработку интерфейса пользователя.

- **HTML-фреймворк Bootstrap**. Разметка HTML-документа в современном мире выполняется с использованием стилей Bootstrap. Это настолько удобно, что один раз попробовав, вы больше не сможете от него отказаться – это как наркотик! Да и множество проектов создано с использованием Bootstrap. Понимая его разметку и стили, вы без проблем разберетесь в проектах сторонних разработчиков.

- **Бандлер (bundler)**. Это инструмент для сборки модулей в единые пакеты, имеющий доступ к файловой системе. Бандлер ищет все выражения require (имеющих ошибочный, с точки зрения браузера, синтаксис) и меняет их на настоящее содержимое каждого требуемого файла.

- **Основы администрирования Linux/UNIX**. Рано или поздно ваши проекты перекочуют с хостинга на виртуальный сервер (VPS/VDS). Если на хостинге все ПО настроено провайдером, то на VDS вам придется все делать самостоятельно. А это означает, что вы должны уметь развернуть (установить и настроить) веб-сервер Apache/nginx, сервер баз данных MySQL, интерпретатор PHP и еще много другого софта. Также вы должны иметь знания по работе в командной строке, разбираться с правами доступа к файлам и каталогам, уметь диагностировать возникающие при работе серверов неисправности – читать логи и понимать, что в них написано.

- **Тестирование.** Отдельная и большая (или больная!) тема. Существуют различные методы тестирования, чаще всего используется модульное тестирование (unit testing), как самый дешевый вид тестирования. Суть идеи легко понять на простом примере. Создавая тест, вы пишете некую обёртку, которая вы-

полнит одну из функций и проверит ожидаемый результат. Если вы поменяете что-либо в другой функции, от которой зависит первая функция, то чтобы проверить, что изменение ничего не сломало — вам достаточно запустить созданные ранее тесты. Надеюсь понятно, насколько это может быть полезно в большом проекте. К сожалению, рассмотрение методов тестирования выходит за рамки этой книги, как уже было отмечено.

- **Фреймворки PHP**. Фреймворк позволяет существенно ускорить разработку, поскольку предоставляет разработчику уже готовые инструменты для выполнения рутинных задач. На сегодняшний день можно выделить наиболее популярные PHP-фреймворки (приводятся в порядке убывания популярности):
  - **Laravel** — один из самых популярных фреймов, по данным некоторых изданий – самый популярный. Ключевые функции, доступные "из коробки": шаблонизатор "Blade", Eloquent ORM для работы с БД и создания моделей (речь про модели из концепции MVC), механизм автоматической загрузки классов PHP без необходимости подключать файлы из определений в **include**, миграции (система управления версиями БД), есть свой формат пакетов, которые позволяют создавать и подключать модули Composer к приложению на Laravel. Многие дополнительные возможности уже доступны в виде таких модулей. Очень гибкий и, в целом, понятный инструмент. Будет хорошим стартом для дальнейшего изучения — при понимании используемых технологий, многие концепции других фреймворков будут понятнее. Также немаловажно, что данный фреймворк использует множество компонентов следующего фреймворка — Symfony, что тоже послужит хорошим подспорьем при дальнейшем расширении кругозора.
  - **Symfony** — в качестве ключевых плюсов упоминаются универсальность, стабильность, однако тут достаточно высокий порог вхождения. Из коробки имеет два ORM-инструмента: Propel и Doctrine. Более сложный и более тяжелый фреймворк по сравнению с Laravel.
  - **Zend** — самый высокий порог вхождения, взамен высокая стабильность и минимальные зависимости между частями

проекта. Создан для разработки больших приложений корпоративного уровня (enterprise). Интересно, что Zend создан и поддерживается разработчиками PHP, а, например, виртуальная машина в интерпретаторе языка PHP именуется не иначе как Zend Engine. Очень "тяжелый" движок, поэтому при разработке средних по сложности сайтов принято использовать Laravel – он попросту работает быстрее. Монстрообразный Zend принято применять для сложных сайтов, где низкая производительность компенсируется предоставляемыми функциями и масштабируемостью.

- **Yii** — считается одним из наиболее производительных и при этом простых фреймворков, хотя однозначных подтверждений этому нет. У него огромное сообщество и высокая популярность. Имеется множество расширений, например, для работы со Sphinx. Одна из основных функций — генератор кода Gii, разумеется также есть миграции БД.
- **CodeIgniter** — достаточно старый фреймворк, постепенно уходящий на задний план. Постепенно он затухает и уступает место более молодым продуктам. Ключевые функции - это собственный гибкий шаблонизатор, шаблоны для работы с БД (очень похожи на синтаксис SQL) и мощные возможности кэширования на стороне сервера. Считается, что это самый минималистичный и легкий фреймворк с одной из лучших документаций.

- **JavaScript-фреймворки**. Подобно PHP-фреймворкам, данные фреймворки позволяют создавать полноценные серверные приложения, но написанные на JavaScript (TrueScript), а не на PHP. Рассмотрим основные продукты:
  - **React**. Библиотека для "реактивного" взаимодействия между элементами интерфейса и предоставляющая шаблонизатор JSX, позволяющий описывать компоненты интерфейса с помощью синтаксиса JS. Основная идея, пожалуй, в использовании собственной системы событий, которая и обеспечивает эту самую "реактивность". React придумали ребята из Facebook.
  - **Angular**. Мощнейший фреймворк, предоставляющий огромные возможности для разработки приложений, на

нём можно реализовывать как "классические" небольшие SPA, так и действительно огромные проекты. Этот фрейм в полной мере соответствует концепции MVC: предоставляет возможность использовать компоненты, имеет свою удобную систему событий и вообще, по моему субъективному мнению, всячески хорош. Если вы планируете разрабатывать большой серьезный проект, я бы рекомендовал изучать именно его. С выходом версии 2.0. проект разделился на два независимых — это AngularJS и просто Angular (нумерация версий начинается с двойки), не следует их путать.

» **Vue**. Популярный ныне фреймворк. Он так же, как и React, использует виртуальный DOM, а вторая его версия поддерживает шаблонизатор JSX. Судя по всему, это попытка упростить чрезмерно сложные фреймворки типа React или Backbone, выбросив лишнее. Поскольку, например, для реализации своей системы событий тут необходимо подключать дополнительный модуль (Vuex), а ядро фреймворка используется в первую очередь как шаблонизатор и селектор элементов.

1.2.3. Если нет таланта дизайнера

Как разработать дизайн сайта, если таланта дизайнера нет? В этом случае помогут сайты, содержащие платные и бесплатные шаблоны сайтов. Иногда можно попросту модифицировать имеющийся шаблон и показать его заказчику. Это проще, нежели создавать новый шаблон с нуля. Стоимость разработки нового макета у дизайнера (если это не вы), может составить несколько десятков тысяч рублей, поэтому за хороший макет не жалко отдать 10-50 долларов (средняя стоимость шаблона, шаблоны продаются обычно в долларах, поэтому и цена тоже в долларах). Сайты с шаблонами вы можете найти по запросам (не хочется рекламировать какой-то сайт):

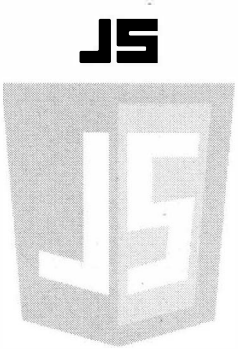
free html templates

html templates

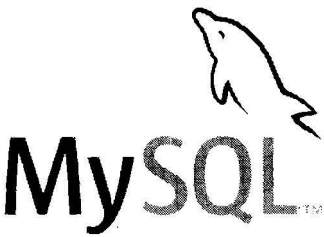
site templates

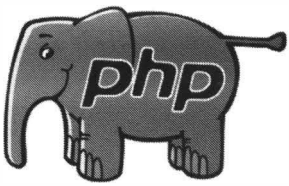
Пора приступить к следующей части книги, в которой мы рассмотрим язык разметки HTML. Это основа основ в веб-разработке. Ведь ваш PHP-скрипт должен сгенерировать именно HTML-документ и отправить его в браузер пользователя. Без знания HTML вы просто не будете знать, что именно нужно сгенерировать в ответ на запрос пользователя. Именно поэтому знание HTML, можно сказать, является краеугольным камнем веб-разработки.


HTML
5


JS


CSS
3


MySQL


php

Глава 2.

ОСНОВЫ HTML 5

2.1. Язык HTML и HTML-документы как составные элементы интернет-сайтов

Сеть Интернет (World Wide Web) – сеть глобального пользования, и поэтому вся информация в сети должна быть представлена на универсальном языке, который понимали бы все пользователи. Языком публикации, используемым в World Wide Web, является HTML (Hyper Text Markup Language – язык разметки гипертекста). Техническую информацию об этом языке (его спецификацию) на английском языке можно найти на сайте World Wide Web Consorcium – w3c.com

Все интернет-сайты и интернет-страницы по своей сути представляют собой HTML-документы или наборы html-документов. В простейшем случае, html-документ хранится на сервере, и при обращении к нему по адресу из браузера он загружается и отображается в окне браузера. В то же время существуют языки программирования и скрипты, написанные на них, которые позволяют не хранить html-страницу на сервере, а динамически формировать эти страницы, исходя из различных данных и действий пользователя. Такой возможностью обладает язык PHP, основы которого мы также рассмотрим в данной книге. Кроме того, существуют языки программирования для написания скриптов, которые выполняют различные функции. Скрипты встраивают в HTML-документ. Таким языком является

язык JavaScript, которому посвящен отдельный раздел книги. В итоге html-документ представляет собой некий каркас, который определяет, что есть в документе и в виде каких структурных элементов в него включено. При этом содержимое этих структурных элементов может находиться непосредственно в самом html-документе (например, текст), а может подгружаться из какого-либо источника: например, из файла изображения, если в html-документе указано, что на этом месте должна быть картинка. Язык HTML позволяет задавать только структуру и не позволяет программировать алгоритмы, поэтому HTML является языком разметки.

Итак, HTML – язык разметки, предлагающий разработчикам следующие возможности:

- Представлять информацию в сети в виде электронных документов, с информационным содержимым в виде форматированного текста, таблиц, списков, фотографий;
- Включать в документы звуковые фрагменты, видеоклипы, электронные таблицы и другие приложения и элементы мультимедиа;
- Осуществлять загрузку документов посредством активизации гипертекстовой ссылки;
- Разрабатывать формы для осуществления взаимодействия с удаленными службами (поисковыми роботами, онлайн-магазинами и т.п.)

Разметка документов заключается в том, что документ представляется в виде последовательности элементов. Например, чтобы отобразить в окне браузера простейшую текстовую информацию

HTML-документ должен иметь следующий вид:

```
<!DOCTYPE html>
<HTML>
      <HEAD>
            <TITLE>Мой первый html-документ</TITLE>
      </HEAD>
      <BODY>
            <H2 align=center>Я знаю HTML</H2>
```

```
            <P>
            Привет мир!
            </P>
      </BODY>
</HTML>
```

Для создания html-документа достаточно набрать его текст в каком-либо простом текстовом редакторе и сохранить его в файле с расширением .html (например, example1.html). Чтобы открыть его, щелкните по нему дважды мышкой, и содержимое html-файла откроется в браузере.

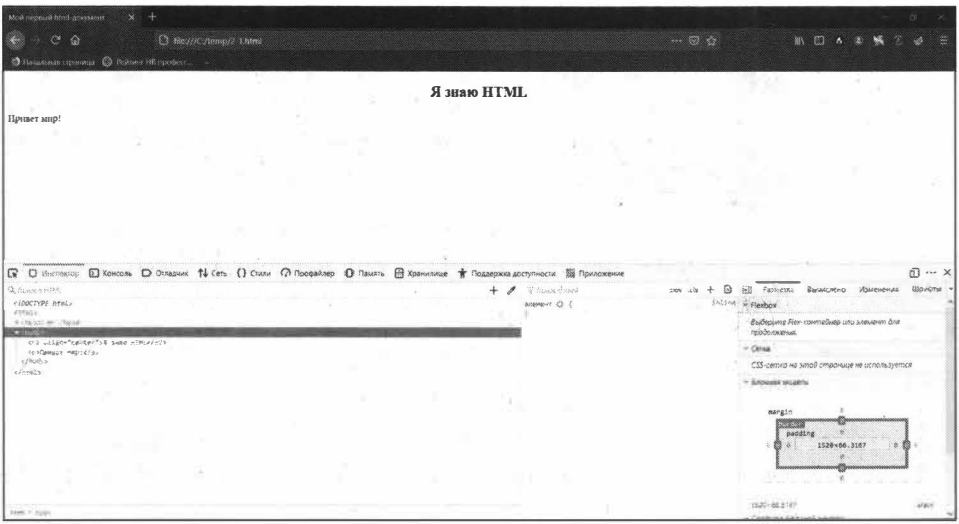


Рис. 2.1. Простейший текстовый HTML-документ

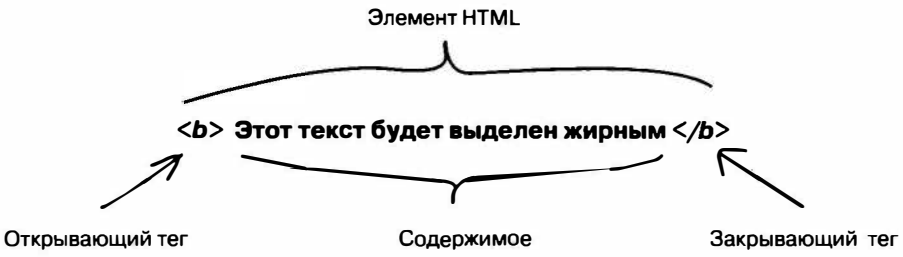
2.2. Теги (элементы) HTML основные структурные единицы HTML

Основная структурная единица, с которой происходит работа в документе HTML – это HTML-элемент, который на профессиональном жаргоне принято называть тегом. Далее в книге мы повсеместно будем использовать слово "тег", по началу еще указывая, что речь идет именно об html-элементе. Если посмотреть исходные файлы верстки сайтов в сети Интернет, элек-

тронных писем, многих печатных форм документов (например, используемые интернет-магазинами, в которых они выставляют покупателю счет), то все они представляют собой набор HTML тегов.

Как и языки программирования, язык разметки HTML постоянно развивается, публикуются новые стандарты, появляются новые, имеющие свои семантические особенности теги.

Каждое объявление тега (HTML-элемента) обычно включает три части: **начальный** (открывающий) **тег, содержимое** и **конечный** (закрывающий) **тег**. Имя тега (HTML-элемента) отображается в начальном теге (<имя элемента>) и конечном теге (</имя элемента>).



На рисунке показана структура элемента **B**, осуществляющего выделение текста жирным шрифтом

```
<b>
  Этот текст будет жирным!
</b>
```

Рис. 2.2. Структурная схема элемента B

Теги служат для разметки, и при просмотре интернет-страницы в браузере на экране не отображаются.

В HTML имеют место теги (элементы), которые позволяют авторам опускать открывающие (например, элементы HEAD и BODY) и закрывающие (например, элементы **P** и **Li**) теги. В некоторых случаях допустимо отсутствие обоих тегов.

Существуют HTML-элементы, не имеющие содержимого. Например, элемент перехода на следующую строку BR содержимого не имеет, т.к. его

функция заключается в прерывании строки текста и переходе на следующую. Элементы такого типа никогда не имеют конечных тегов.

Все теги языка HTML выделяются символами-ограничителями (< и >), между которыми прописывается имя тега и, возможно, его атрибуты.

Единственным исключением являются теги комментария. Для них предусмотрено следующее написание:

```
<!--Текст комментария -->
```

Имена тегов и их атрибутов не чувствительны к регистру, т.е. не имеет значения, заглавными или прописными буквами они написаны. Например, записи <bOdY> и <BoDy> абсолютно идентичны с точки зрения HTML.

Атрибуты – это свойства тега (элемента), которые имеют либо стандартное значение (используемое по умолчанию), либо значение, задаваемое разработчиком или сценарием (скриптом).

Набор атрибутов индивидуален для каждого тега (элемента), хотя некоторые из них могут повторяться у разных тегов (элементов).

Пары "атрибут=значение" помещаются перед закрывающей скобкой начального тега элемента, а если их несколько – разделяются пробелами. Порядок их перечисления произволен. Например,

```
<BODY id = "pap">
      ..... содержимое .....
      ..... содержимое .....
      ..... содержимое .....
</BODY>
```

Некоторые атрибуты не имеют значения, тогда достаточно просто указания имени атрибута.

По умолчанию в HTML необходимо, чтобы все значения атрибутов были заключены в двойные (код ASCII 34) или одинарные (код ASCII 39) кавычки. В некоторых случаях допустимо этого не делать, но рекомендуется кавычки применять всегда.

Значение атрибута может включать только буквы (a – z, A – Z), цифры, знаки переноса (код ASCII 45) и точки (код ASCII 46). Включение одинарных кавычек в значение атрибута допустимо, если оно заключено в двойные кавычки, и наоборот.

При интерпретации тегов все браузеры придерживаются того правила, что при обработке неправильно заданного HTML-элемента или при использовании в документе элемента разметки, не поддерживаемого данным браузером, он игнорируется.

2.3. Структура документа HTML

Можно выделить общую структуру для файлов в формате HTML. Документ в формате HTML состоит из трех основных частей:

- строки, объявляющей файл как документ на языке HTML5;
- заголовка, заключенного в тег HEAD;
- тела документа, представляющего собой тег BODY, если документ имеет классическое, однооконное представление. В теле документа содержится вся, предлагаемая пользователю, информация.

Заголовок и тело документа заключены в тег-контейнер HTML. Все остальные теги разметки располагаются либо в заголовке, либо в теле документа.

Простейший пример HTML-документа представлен ниже:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Название документа </title>
  </head>
  <body>
    Здесь располагается текст страницы
  </body>
</html>
```

Тег `<!DOCTYPE html>` позволяет браузеру, в котором открывается данная страница, определить, что для верстки был использован стандарт языка HTML версии 5.

2.4. Заголовок документа HTML

Тег <HEAD> (элемент заголовка) предназначен для содержания в нем информации о документе, а именно:

- название документа (тег TITLE) <title>
- полный URL документа (тег BASE) <base>
- управляющую информацию (тег META) <meta>
- список ссылок (тег LINK) <link>
- описание стилей (тег STYLE) <style></style>
- задание скриптов (тег SCRIPT) <script></script>

Тег HEAD имеет один необязательный атрибут *profile*, указывающий на внешний файл мета-элементов. В качестве значения атрибута задается адрес URL. Задание тега HEAD в общем виде выглядит так:

```
<HEAD profile="URL">
   ..... список элементов заголовка .....
   ..... список элементов заголовка .....
   ..... список элементов заголовка .....

</HEAD>
```

Тег TITLE – задаем название интернет-страницы

Тег TITLE задает название HTML-документа (интернет-страницы). Этот тег разметки, строго говоря, не является обязательным, однако его использование настоятельно рекомендуется. Он присваивает документу название, независимое от имени файла, которое отображается в строке заголовка браузера. Также это имя используется по умолчанию при добавлении страницы в папку "Избранное". Документу, тег TITLE которого не задан, браузер в качестве его имени будет использовать надпись "Без заголовка" или полный адрес документа.

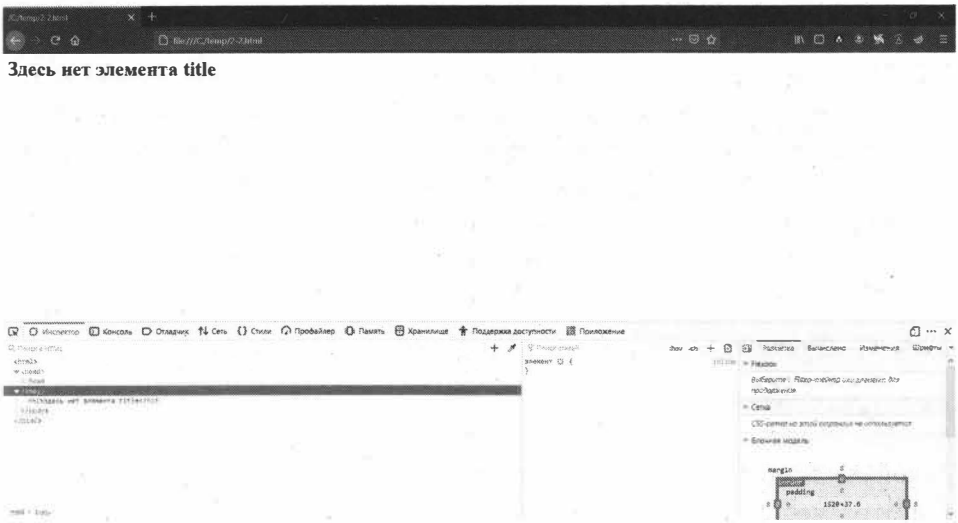


Рис. 2.3. Демонстрация отсутствия элемента TITLE

Открывающий и закрывающий теги TITLE являются обязательными. Его содержимое представляет текстовую строку неограниченной длины.

Название документа должно кратко характеризовать его содержание. Учитывая это, а также возможную минимизацию окна браузера, рекомендуется в названии документа ограничиться 50-60 символами. Содержание тега TITLE не должно включать в себя других тегов разметки. Например, нельзя с помощью тега I вывести название документа курсивом, т.е. запись:

```
<TITLE><I> название документа </I></TITLE>
```

недопустима.

Запись:

```
<I><TITLE> название документа </TITLE></I>
```

также неприемлема, так как тег **I** не является тегом заголовка.

Тег BASE

Тег BASE используется для явного задания полного URL-адреса документа. Это бывает полезно, ввиду того что общепринятым стилем задания гипертекстовых ссылок является их относительная адресация. То есть при задании ссылки на документ указывается не полный его URL-адрес, а задается его месторасположение относительно текущего адреса. Тег BASE как раз и задает адрес, относительно которого будут браться относительные ссылки в HTML-документе.

Пример относительной ссылки:

```
<A href="/BOOK1/chapter1.html">
```

Такой стиль полезен тем, что при переносе всего дерева HTML-документов в другое место (на другой сайт), не требуется изменять ссылки в самих HTML-документах.

Использование тега BASE позволяет реализовывать относительные ссылки в том случае, когда HTML-документ перемещен (или скопирован), а все остальное дерево документов, на которые он ссылается, нет. Адрес его поменялся (например, документ лежал на www.anekdot.ru, а теперь на Вашем компьютере, на диске **C**), однако при активизации относительной ссылки, она будет взята браузером относительно исходного адреса, прописанного в теге BASE.

Тег BASE имеет один обязательный атрибут *href*, значением которого является полный URL документа.

В листинге показано применение тега BASE и относительных ссылок:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 //EN">
<HTML>
  <HEAD>
    <TITLE>Документ с использованием элемента BASE</TITLE>
    <BASE href= "www.nit.center">
  </HEAD>
  <BODY>
      .... текст документа ....
      .... текст документа ....
      <A href= "\IMAGES\ret.gif">Ссылка на изображение ret.gif</A>
      .... текст документа ....
```

```
      <A href= "chapter2.html">Переход к следующей главе</A>
      .... текст документа ....
      .... текст документа ....
      .... текст документа ....
   </BODY>
</HTML>
```

В данном примере переход по относительным ссылкам задается относительно URL - адреса www.nit.center. Таким образом, заданные в этом документе ссылки в абсолютном варианте всегда (независимо от месторасположения документа) будут иметь следующий вид: www.nit.center\IMAGES\ ret.gif и www.nit.center\chapter2.html. Если бы базовый адрес не был задан, то ссылки интерпретировались бы относительно каталога, в котором находится данный документ. Соответственно, при перемещении документа изменялись бы цели относительных ссылок.

Начальный тег BASE обязателен, конечный тег запрещен.

Тег BASE часто используется для организации узлов, которые имеют так называемые "зеркала". Часть документов основного сервера переносится на "зеркальный". Обычно это делается для повышения скорости взаимодействия с удаленными пользователями и применяется в совокупности с запретом на кэширование перенесенных документов (как пользовательскими браузерами, так и proxy-серверами).

Тег BASE можно использовать и в заголовке, и в теле документа, причем несколько раз. Область действия тега BASE определяется от места его задания и до конца документа, или до следующего объявления тега BASE, если таковой имеется.

Тег LINK – подключение каскадных таблиц стилей CSS к HTML-документу

Данный тег указывает на связь, отношение между содержащим его документом и другим ресурсом сети. На данный момент единственное его практическое применение заключается в подключении внешних каскадных таблиц стилей (CSS), ответственных за визуальное представление документа.

Информация о документе, предоставляемая элементом LINK, может быть использована некоторыми поисковыми машинами для оптимизации поиска.

Пример:

```
<link rel="stylesheet" type="text/css" href="style.css">
```

В качестве значения атрибута *href* указывается путь к файлу стилей CSS.

Тег META – задаем ключевые слова для индексации поисковиками и кодировку

Тег META используется для задания некоторых свойств интернет-страницы (а именно: автора, списка ключевых слов, кодировки и т.п.), благодаря чему позволяет управлять обработкой HTML-документа. Теги META и TITLE являются наиболее используемыми при задании заголовка.

При задании META начальный тег обязателен, конечный тег запрещен.

Атрибуты:

- **name** – указывает имя свойства;
- **content** – задает значение свойства;
- **scheme** – указывает имя схемы, используемой для обработки значения свойства;
- **http-equiv** – используется вместо атрибута name для указания имени http-сообщения;
- **lang** – информация о языке. Необязательный атрибут.
- **dir** – указывает направленность текста. Необязательный атрибут.

Каждый элемент META содержит в себе пару свойство-значение. Атрибут *name* (http-equiv) указывает свойство, атрибут *content* – значение.

Например:

```
<META http-equiv=content-type content= "text/
html;charset=utf-8" >

<META name=description content="Обучение, управление
проектами, сертификация PMP">

<META name=keywords content="PMP, проект, управление
проектами, сертификация PMP">
```

В первом случая указана кодировка HTML-документа, во втором случае – описание документа, в третьем – ключевые слова. Оба последних тега META обычно имеют одинаковое или близкое значение в целях повышения эффективности обнаружения их поисковыми роботами, так как именно по этим параметрам, в значительной степени, осуществляется отбор документов.

Атрибут *http-equiv* может использоваться вместо атрибута name для задания свойств HTML-документа на уровне http-заголовка. Через атрибут *http-equiv* осуществляется доступ к полям HTTP-заголовка. Первое полезное применение этого атрибута заключалось в осуществлении принудительной перезагрузки документа браузером. Для этого, с помощью атрибута *http-equiv* используется http-оператор **refresh**. Время, через которое надо произвести перезагрузку указывается через атрибут *content*, а адрес загружаемого документа – атрибутом *url* оператора **refresh**.

Пример написания:

```
<META http-equiv="Refresh" content="5; new_document.html">
```

При таком задании через 5 секунд после загрузки текущего документа браузер автоматически перейдет к загрузке документа new_document.html.

Практически во всех HTML-документах тег META используется для их описания посредством задания списка ключевых слов и краткой информации о содержимом документа. Ключевые слова, вместе с названием документа, помогают поисковым машинам найти документ. В своих отчетах они выдают название документа, прописанное в элементе TITLE, и его краткое описание, заданное через элемент META.

Для указания списка ключевых слов и краткой информации о документе в заголовке используются два META элемента:

```
<META name="description" content="Обучение, управление
проектами, сертификация PMP">

<META name="keywords" content="PMP, проект, управление
проектами, сертификация PMP ">
```

Через тег META можно указать кодировку содержимого документа. Элемент META тогда принимает вид:

```
<META http-equiv="Content-type" content="text/
html;charset=windows-1251">
```

С помощью тега META можно запретить кэширование документа, что бывает полезно при частом обновлении документа. Для осуществления этой операции в элемент META включается оператор cache-control (в HTTP 1.0 применялся оператор Pragma). Оператор устанавливается в положение *no-cashe*. META-элемент тогда примет вид:

```
<META http-equiv="Cache-control" content="no-cache">
```

При кэшировании документа можно указать время, до которого имеет место соответствие закэшированного документа с его оригиналом на сервере. В данном случае используется HTTP-оператор Expires и элемент META принимает вид:

```
<META http-equiv="Expires" content="Mondey, 18-May-2023
00:00:01">
```

> **Примечание: в данном случае дата задается для корректировки HTTP-заголовка и поэтому должна иметь следующий формат, указанный в примере.**

Тег STYLE – задание стилевых настроек CSS3 непосредственно в HTML-документе

О самих каскадных таблицах стилей CSS3 мы поговорим в следующей главе. Сейчас же важно знать, что правила каскадных таблиц стилей можно за-

давать непосредственно в HTML-документе, к которому они должны быть применены, а можно их размещать в виде отдельного файла с расширением css и подключать его к HTML-документу по ссылке (это делается в теге заголовка LINK – см. выше).

Элемент STYLE предназначен для задания правил каскадных таблиц стилей CSS непосредственно в html-документе. При этом, согласно правилам каскадирования стилей, если какие-то из них совпадают по имени с элементами описания стиля, заданными во внешнем файле (подключенном через элемент LINK), то они заменяют элементы из внешнего файла.

Начальный тег обязателен, конечный тег обязателен.

Атрибуты:

- **type** – обязательный атрибут, указывающий язык таблицы стилей, содержащейся в данном элементе. Язык таблицы стилей указывается как тип содержимого. Обычно это text/css либо text/javascript.
- **media** – необязательный атрибут, задает целевое устройство для информации о стиле. Имеет следующий вид: media=дескриптор_усторойства (например, projection – проектор). По умолчанию установлен в значение screen (экран компьютера).

Необязательные атрибуты:

- **lang** – информация о языке;
- **dir** – информация о направленности текста

Пример использования:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional //EN">
<HTML>
        <HEAD>
        <TITLE>Пример использования элемента STYLE </TITLE>
                <STYLE type="text/css">
                /* Это комментарий в CSS */
                   H1 {border-width: 1;
                   border: solid;
                   text-align: center}
                 </STYLE>
        </HEAD>
```

```
        <BODY>
         <H1> Пример использования элемента STYLE </H1>
         Благодаря простейшей таблице стилей заголовок,
                задаваемый использованием элемента H1,
                будет отображен отцентрированным и
                заключенным в рамку. И если в дальнейшем
                в тексте документа будут присутствовать
                заголовки первого уровня (элемент H1), то
                они также будут расположены по центру и
                обведены рамкой.
        </BODY>
</HTML>
```

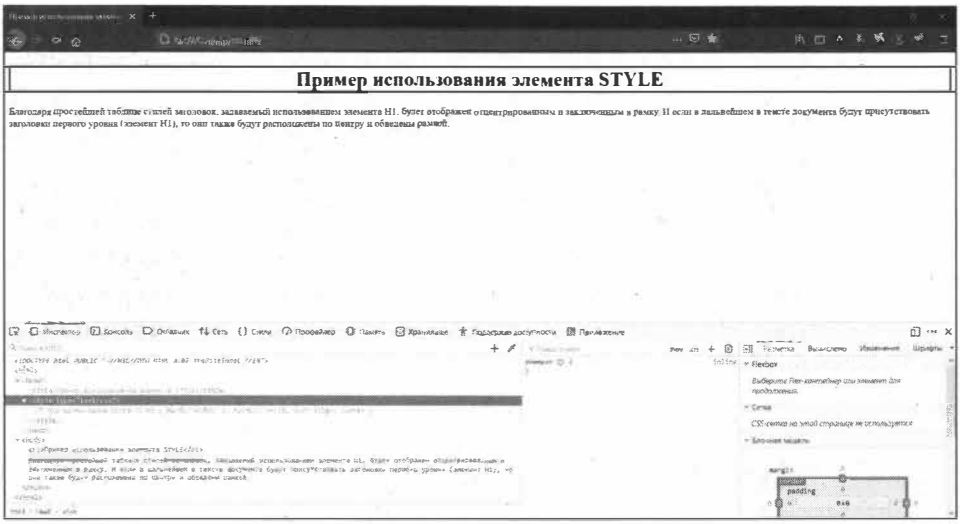


Рис. 2.4. Пример использования элемента STYLE и каскадных таблиц стилей

Обратите внимание на стиль комментариев. В коде CSS допускаются комментарии в стиле C++ и C, например:

```
// Это допустимый комментарий в CSS-коде
/* Это тоже допустимый
   многострочный
      комментарий в CSS-коде
```

А вот комментарии HTML в стиле:

```
<!-- комментарий -->
```

нельзя использовать в CSS-коде.

Тег SCRIPT – задание и подключение скриптов Javascript к HTML-документу (интернет-странице)

Тег SCRIPT служит для размещения кода сценария, написанного на языке скриптов (например: JavaScript), или подключения внешнего файла с таким скриптом. Как и в случае с CSS-стилями, сценарии могут задаваться как внутри HTML-документа, так и в отдельном файле. Тег SCRIPT может быть задан и в заголовке, и в теле HTML-документа, причем несколько раз.

Начальный тег обязателен, конечный тег обязателен.

Атрибуты:

- **type** - указывает язык, на котором написан скрипт, как тип содержимого (например, type="text/javascript"). Является обязательным атрибутом (значение по умолчанию не задано)
- **src** – необязательный атрибут, указывает месторасположение (URL) внешнего скрипта. Общий вид: src=URL
- **language** – указывает язык скрипта, содержащегося в элементе. Его значением является идентификатор языка. Является нежелательным элементом, так как идентификаторы языка не стандартизированы. Вместо него используется атрибут type. Пример задания: language="javascript".
- **defer** – необязательный логический атрибут. Его указание указывает браузеру, что скрипт не будет генерировать содержимое текста.
- **charset** – кодировка символов. Необязательный атрибут, определяемый в любом месте документа. Указывает кодировку внешнего скрипта (заданного атрибутом scr) и не относится к содержимому тега SCRIPT.

Если не задан атрибут *scr*, содержимое тега SCRIPT интерпретируется браузером как скрипт. В том случае, если этот атрибут задан, скрипт загружается с указанного URL, а все содержание элемента SCRIPT браузером игнорируется.

2.5. Тело HTML-документа. Тег BODY

В теле документа располагается содержательная его часть. В качестве тела документа определен элемент-контейнер BODY. Начальный и конечный теги являются необязательными (необходимость их использования определяется контекстом)

2.6. Персонализация и сегментация HTML-тегов. Атрибуты id и class – готовим базу для применения CSS-стилей

Важным правилом верстки сайтов является разделение структурной разметки и стилей оформления. То есть в **html** есть возможность для текста заголовка H1 (тег H1) задать его выравнивание непосредственно в теге H1, но правильно будет создать CSS-правило для тега H1, которое будет располагаться отдельно от тега H1, и применяться ко всем тегам H1 в документе. Благодаря такой организации вам, во-первых, не надо будет искать, где именно в тексте html-документа надо подправить выравнивание в случае необходимости (все стилевые настройки располагаются обычно в отдельном файле или в заголовке html-файла), а во-вторых, вам не надо будет каждый раз задавать для каждого тега H1 значение выравнивания текста – одно CSS-правило для тега H1 будет применено ко всем тегам H1 в html-документе. Соответственно, и в случае изменения выравнивания текста в CSS-правиле для тега H1 изменения будут применены ко всем тегам H1. Такой стиль верстки настолько считается правильным, что в версии HTML5 у тега H1 (и других) были исключены атрибуты задания оформ-

ления: выравнивания и т.п. Подразумевается, что все это будет делаться с помощью таблиц стилей CSS. Однако такие атрибуты все еще поддерживаются браузерами, так как они активно применялись в предыдущей версии HTML 4.01, и есть сайты, созданные в рамках предыдущего стандарта. Но мы с вами учимся создавать сайты согласно последним стандартам, а значит будем отделять определение структурных элементов – тегов HTML, и их оформление – CSS-стили.

И вот здесь появляется одна особенность: когда вы непосредственно для каждого тега задаете его внешний вид (например, выравнивание для заголовка H1), то можете для разных случаев применения одного и того же тега в HTML-документе задавать свои параметры (для одного заголовка H1 задать одно выравнивание, а для другого заголовка H1 на этой же html-странице – другое выравнивание). Когда же вы создали одно стилевое CSS-правило для тега, то оно будет применяться ко всем включениям тега на html-странице. Как в таком случае для разных заголовков H1 задать разный вид в разных частях html-страницы? Для того, чтобы у верстальщика была возможность обращаться (и писать CSS-правила) не ко всем экземплярам тега на html-странице, а какому-либо конкретному экземпляру, в HTML предусмотрена возможность для каждого тега задавать его уникальное имя – задать конкретному экземпляру тега атрибут *id*, а также возможность объединять несколько тегов под одним именем – задать для нескольких экземпляров одного тега одинаковый атрибут *class* (то есть объединить их в класс). И тогда, при обращении указывается тег и *id* конкретного тега (или *class* – при классификации экземпляров одного тега).

Итак, атрибуты *id* и *class* служат для идентификации содержащих их тегов. Через атрибут *id* тегу можно присвоить уникальное имя. Атрибут *class* причисляет тег к классу тегов, созданному разработчиком. Причем сам класс тегов образуется с первым включенным в него тегом. Если тег причисляется к нескольким классам сразу, то тогда имена классов должны быть разделены пробелами.

Атрибут *id* служит в HTML для выполнения следующих функций:

- для осуществления выборочного обращения таблицы стилей к определенным элементам (задания их стиля);
- для указания цели (якоря) гипертекстовых ссылок, что позволяет им ссылаться не только на документ в целом, но и на его отдельные части;
- для реализации ссылки на определенный элемент сценария;

- для задания имени объекта, вставляемого в документ тегом OBJECT.

Атрибут *class* служит в HTML для выполнения следующих функций:

- для осуществления выборочного обращения таблицы стилей к определенной группе элементов (задания их стиля);
- для реализации особой обработки браузером определенной (заданной разработчиком) группы элементов.

Атрибуты *id* и *class* могут быть установлены почти для всех тегов языка HTML. Именование HTML-тегов и объединение их в определенные группы позволяет осуществлять обращение к ним, что особенно полезно при применении к ним таблиц стилей.

2.7. Форматирование текста документа HTML

Подходы форматирования текста в HTML

Для форматирования текста в HTML могут использоваться два подхода: задание структуры документа и использование элементов непосредственного (физического) форматирования.

При структурном представлении текста он разбивается на фрагменты, которым ставится в соответствие определенный структурный тип, а вместе с ним и его свойства. Описание же этих свойств происходит применительно к структурному типу (в начале HTML-документа), а не к тексту. Благодаря этому достигается разделение содержательной и описательной частей HTML-документа. Фрагменты текста, которые могут быть выделены – самые разные: цитата (тег <cite>), фрагмент компьютерного кода (тег <code>), выделенный текст (тег), тег (элемент <samp>) и другие.

При непосредственном (физическом) форматировании визуальное свойство ставится в соответствие и характеризует непосредственно фрагмент

текста. Например, тег <i> выводит текст, заключенный между его начальным и конечным тегами, курсивом. тег <b> – жирным шрифтом, и т.д.

Момент разделения HTML-элементов форматирования текста на структурные и физические может оказаться достаточно сложным для неподготовленного читателя. Поэтому при первом ознакомлении с HTML им рекомендуется ориентироваться на видимый эффект, производимый тем или иным элементом разметки. Для этих целей более удобным и понятным является использование тегов физического форматирования, так как некоторые теги структурного форматирования приводят к одинаковому визуальному эффекту. Например, для выделения текста курсивом предусмотрен один тег физического форматирования <i>. Этого же видимого эффекта можно достичь при помощи четырех тегов структурного форматирования <var>, , <dfn>, <cite>, просто они еще характеризуют смысловое содержание выделенного текста. Понимание структуры документа, а также необходимость ее четкого представления произойдет при разработке и организации сложных HTML-документов.

Первое время (в ранних версиях HTML) использовался в основном метод физического форматирования, но с развитием HTML-технологий и каскадных таблиц стилей приоритетным стал структурный подход.

Теги структурного форматирования текста

- **Тег ABBR** - отмечает заключенный в себе текст как аббревиатуру. А атрибут <title> добавляет всплывающую подсказку к тексту, в которой можно дать расшифровку аббревиатуры.
- **Тег ACRONYM** - отмечает свой текст как акроним, т.е. произносимое слово, состоящее из аббревиатур. Тег ACRONYM обычно находит свое применение с использованием атрибута title, значением которого выставляется текст расшифровки аббревиатуры. Тогда, при наведении курсора на аббревиатуру, расшифровывающий текст будет отображаться браузером как примечание.
- **Тег CITE** - отмечает заключенный в себе текст как цитату. Браузерами такой текст отображается курсивом.
- **Тег CODE** - отмечает заключенный в себе текст как фрагмент компьютерного кода. Браузерами такой текст по умолчанию

отображается моноширинным шрифтом (ширина всех символов одинакова).

- **Тег DFN** - отмечает заключенный в себе текст как определение (англ. definition). Браузеры отображают содержимое тега <dfn> с помощью курсивного начертания.
- **Тег EM** - отмечает заключенный в себе текст как выделенный. Браузерами по умолчанию отображается курсивом. Является предпочтительной альтернативой тега физического форматирования I (выделяет текст курсивом).
- **Тег KBD** - отмечает заключенный в себе текст, как введенный пользователем с клавиатуры. Браузерами по умолчанию отображается моноширинным шрифтом.
- **Тег SAMP** - отмечает заключенный в себе текст, как пример (пример отчета, выдаваемого программой, сценарием и т.п.). Браузерами по умолчанию отображается моноширинным шрифтом.
- **Тег STRONG** - отмечает заключенный в себе текст, как сильно выделенный. Браузерами по умолчанию отображается жирным шрифтом. Является предпочтительной альтернативой тега физического форматирования B (выделяет текст жирным шрифтом).
- **Тег VAR** - отмечает заключенный в себе текст, как экземпляр переменной или аргумента какой-либо программы. Браузерами по умолчанию отображается курсивом.

2.8. Строки и абзацы

Текст документа обычно разбивается авторами на абзацы, текст в которых имеет законченное смысловое содержание. Это делает прочтение и понимание документа более удобным. Обычно в текстовых редакторах разделение абзацев производится переходом на следующую строку с помощью клавиши "Enter". На работу браузеров этот метод не оказывает никакого влияния. Это значит, что при отображении текста, при отсутствии в документе каких-либо указаний со стороны HTML-разработчика, браузер сам будет

осуществлять переходы на следующую строку из соображений экономного использования пространства своего окна. При изменении размеров окна соответственно будет меняться длина строк и, следовательно, визуальное представление. Все это может привести к большим неудобствам. Чтобы их избежать в языке HTML предусмотрено разбиение текста на отдельные абзацы путем заключения их в содержимое тегов **P**.

Тег P

HTML располагает своими средствами для разбиения на абзацы. Для этой функции используется тег уровня блока **P**. Текст, находящийся между начальным и конечным тегами, воспринимается браузерами как абзац. При отображении абзацы сверху и снизу выделяются пустой строкой.

Указание начального тега обязательно, конечный тег может не задаваться: при этом, абзацем будет считаться все, что расположено после начального тега.

В теге **P** могут содержаться теги уровня блока, включая и сам тег **P**.

Пример:

```
<P>
   Текст абзаца
   Текст абзаца
   <P>
   Текст вложенного абзаца
   Текст вложенного абзаца
   </P>
</P>
```

Тег BR

Место переноса строки в пределах абзаца, тем не менее, определяется браузером по-прежнему автоматически, исходя из размера окна и размера шрифта. HTML предоставляет возможность принудительного переноса строки, независимого от настроек браузера. Элементом, осуществляющим

принудительный перенос строки, является тег BR. Перенос строки происходит сразу после места его задания.

Пример:

```
Текст строки <BR>
Текст с новой строки </BR>
```

2.9. Заголовки – теги H1, H2, H3, H4, H5, H6

В языке HTML предусмотрены теги, определяющие содержащийся между их начальным и конечным тегами строку текста как заголовок. Всего определено шесть заголовков различного уровня. Каждому из них соответствует определенный размер: самым маленьким является заголовок шестого уровня, самым большим – заголовок первого уровня. HTML-элементами, соответствующими заголовкам с первого по шестой уровень, являются теги H1, H2, H3, H4, H5, H6. Все заголовки отображаются жирным шрифтом.

Теги H1, H2, H3, H4, H5, H6 являются блокообразующими элементами. Это означает, что они не могут использоваться внутри текста для выделения отдельных его фрагментов, так как строка, содержащая заголовок, может содержать только заголовок. Весь остальной текст располагается выше и ниже его.

Заголовки сверху и снизу выделяются пустыми строками.

При использовании тегов H1, H2, H3, H4, H5, H6 задание начального тега является обязательным. Конечный тег может не указываться, тогда заголовком будет считаться все, что расположено после начального тега.

2.10. Списки на интернет-страницах. Теги UL и OL

Язык HTML обладает возможностями предоставления информации в виде списков. Список служит для добавления структуры в документ. Причем эта

структура отображается визуально, например: список покупок, меню сайта, пошаговое описание каких-либо действий, толковый словарь и т.п.

В HTML различают:

- Неупорядоченные списки (список покупок)
- Упорядоченные списки (пошаговое описание, в котором каждый шаг пронумерован).
- Список определений (толковый словарь)

В любом списке должен присутствовать хотя бы один элемент списка, иначе он будет проигнорирован.

Неупорядоченные списки создаются тегом UL, упорядоченные списки – тегом OL. Списки обоих типов состоят из последовательности элементов списка, которые задаются тегом LI. Тег LI, а точнее его содержание (например, название покупки), является обособленной частью списка. Неупорядоченные списки отображаются браузерами как маркированные, а упорядоченные – как пронумерованные. Пример структуры неупорядоченного списка (маркированного):

```
<ul>
    <li>
        ...
    </li>
    <li>
        ...
    </li>
    <li>
        ...
    </li>
    <li>
        ...
    </li>
    <li>
        ...
    </li>
</ul>
```

2.11. Таблицы на интернет-страницах

Создание таблицы. Тег TABLE

Одним из самых распространенных и эффективных способов представления информации в документах является использование таблиц. В HTML применение таблиц носит более общий характер. Они позволяют разработчикам упорядочивать информацию в документе: текст, изображения, объекты, формы, поля форм и т.п. Это значит, что помимо построения табличек как таковых, теги таблиц используются как средство форматирования документа. Таблицы с неотображаемыми (невидимыми) границами долгое время использовались для верстки веб-страниц, позволяя разбивать документ на прямоугольные области, в каждую из которых может быть помещена своя информация. Подобный способ применения таблиц нашел воплощение на многих сайтах, пока ему на смену не пришел более современный способ верстки с помощью слоев.

В HTML-документе может содержаться любое количество таблиц, причем допускается вложение таблиц друг в друга. Каждая таблица является содержимым тега-контейнера **<TABLE>** и состоит, по крайней мере, из одной строки.

Тег TABLE имеет атрибут *border*, с помощью которого можно задавать параметры границ таблицы – толщину рамки вокруг таблицы и видимость рамки вокруг каждой ячейки. В контексте использования атрибута *Border* задание элемента TABLE может быть в трех вариантах:

- <TABLE> - атрибут *Border* не задан. В этом случае никакие рамки не прорисовываются. На экране отображается только содержащаяся в ячейках информация в отформатированном виде.
- <TABLE border> - атрибут *Border* задан, но не задано его значение. В этом случае отображаются рамки вокруг ячеек толщиной в 1 пиксел (эта величина изменению не подлежит) и рамка вокруг таблицы толщиной в 1 пиксел, установленной по умолчанию.
- <TABLE border=pixels> - атрибут *Border* задан и задано его значение в пикселах. В этом случае рамка вокруг таблицы отобра-

жается толщиной указанной в значении, а рамки вокруг ячеек по-прежнему отображаются толщиной в 1 пиксел.

Несмотря на наличие данного атрибута, задание параметров границ таблиц HTML рекомендуется осуществлять с помощью соответствующих правил CCS3, тем более что что при таком задании можно указывать различные параметры для разных границ таблицы.

Каждая строка задается тегом <TR> (Table Row), внутри которой осуществляется деление на ячейки. Каждая ячейка строки представляет собой тег <TH> (Table Header – ячейка заголовка) или тег <TD> (Table Data – ячейка с данными). Разница в использовании этих тегов заключается в различном визуальном представлении заключенной в них информации:

- **Тег <TH>** – содержащаяся в нем информация по умолчанию выводится полужирным шрифтом и выравнивается по центру ячейки (align=center; valign=middle);
- **Тег <TD>** – содержащаяся в нем информация по умолчанию выводится обычным шрифтом, выровненная по горизонтали влево (align=left) и по центру по вертикали (valign=middle).

Количество строк в таблице определяется количеством начальных тегов TR. Количество столбцов – максимальным количеством ячеек в одной строке среди всех строк таблицы. Остальным строкам браузер добавляет необходимое количество пустых ячеек. Пустые ячейки добавляются в строки справа для таблиц, имеющих направление слева направо, и слева для таблиц, направленных справа налево. Порядок столбцов (направление таблицы) задается атрибутом *dir* тега <TABLE>.

Пример:

```
<!DOCTYPE html>
<html>
   <head>
      <title> Пример простейшей таблицы</title>
   </head>
   <body >
   <h2>Пример простейшей таблицы</h2>
   <table border="1">
      <tr>
         <th>1-ая ячейка 1-ой строки</th>
         <th>2-ая ячейка 1-ой строки</th>
```

```
        <th>3-ая ячейка 1-ой строки</th>
      </tr>
      <tr>
        <td>1-ая ячейка 2-ой строки</td>
        <td>2-ая ячейка 2-ой строки</td>
      </tr>
      <tr>
        <td>1-ая ячейка 3-ей строки</td>
        <td>2-ая ячейка 3-ей строки</td>
        <td>3-я ячейка 3-ей строки</td>
      </tr>
      <tr>
        <td>1-ая ячейка 4-ой строки</td>
      </tr>
    </table>
    </body>
</html>
```

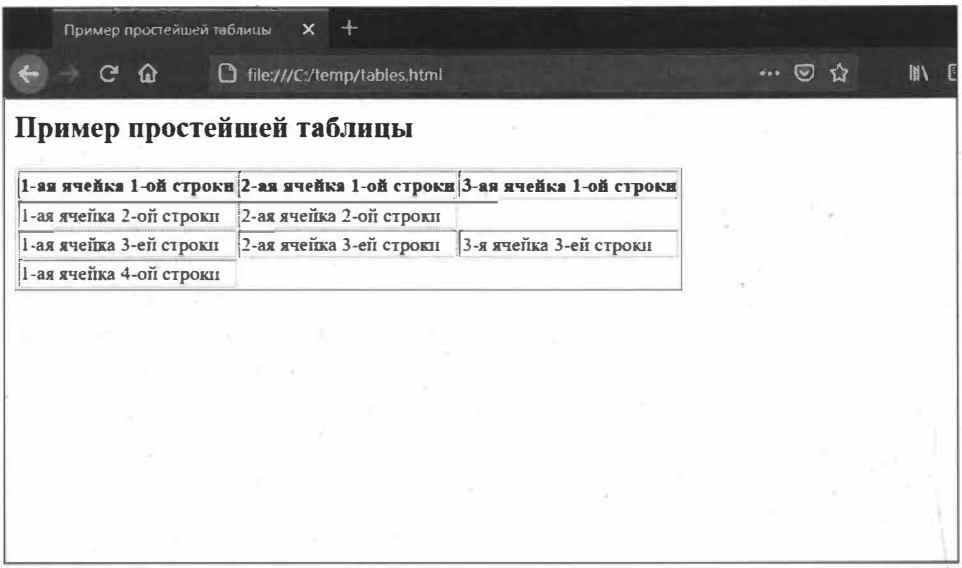


Рис. 2.5. Пример документа с простой таблицей

Таблица может иметь название, которое задается с помощью тега CAPTION.

Строки и ячейки таблицы. Теги TR, TH, TD

Содержимым **тега <TR>** является строка таблицы, а точнее ячейки, расположенные в одной строке. Количество строк в таблице определяется количеством тегов <TR>. Начальный тег обязателен, конечный тег не обязателен и обычно не указывается.

Теги <TH> и <TD> определяют ячейки таблицы. Содержимым этих тегов является информация, содержащаяся в отдельной ячейке.

У обоих начальный тег обязателен, конечный тег необязателен и обычно не указывается.

Теги <TH> и <TD> имеют следующие необязательные атрибуты:

- **headers** – атрибут, значением которого является список имен (id) ячеек, выступающих в роли заголовка (описания) для содержащей его ячейки.
- **scope** – задает область таблицы, для которой информация из данной ячейки является заголовочной (описанием). Принимает одно из следующих значений:
  - *row* – говорит о том, что информация из данной ячейки является заголовком (описанием) для всех последующих ячеек строки.
  - *col* - говорит о том, что информация из данной ячейки является заголовком (описанием) для всех последующих ячеек столбца, содержащего данную ячейку.
  - *rowgroup* - говорит о том, что информация из данной ячейки является заголовком (описанием) для всех последующих строк таблицы.
  - *colgroup* - говорит о том, что информация из данной ячейки является заголовком (описанием) для всех последующих столбцов таблицы.
- **rowspan** – атрибут, отвечающий за объединение соседних строк в рамках одного столбца (т.е. соседних ячеек в столбце) В качестве своего значения принимает натуральные числа. По умолчанию установлен в значении равном единице.
- **colspan** - атрибут, отвечающий за объединение соседних столбцов в рамках одной строки (т.е. соседних ячеек в строке). В качестве своего значе-

ния принимает натуральные числа. По умолчанию установлен в значении равном единице.

Объединим две первые ячейки в одну. Для этого надо указать атрибут colspan=2 для первой ячейки и стереть вторую или третью ячейку:

```
<table border="1">
   <tr>
      <th colspan="2">Ячейка 1</th>
      <th>Ячейка 3</th>
   </tr>
   <tr>
      <td>Ячейка 4</td>
      <td>Ячейка 5</td>
      <td>Ячейка 6</td>
   </tr>
   <tr>
      <td>Ячейка 7</td>
      <td>Ячейка 8</td>
      <td>Ячейка 9</td>
   </tr>
</table>
```

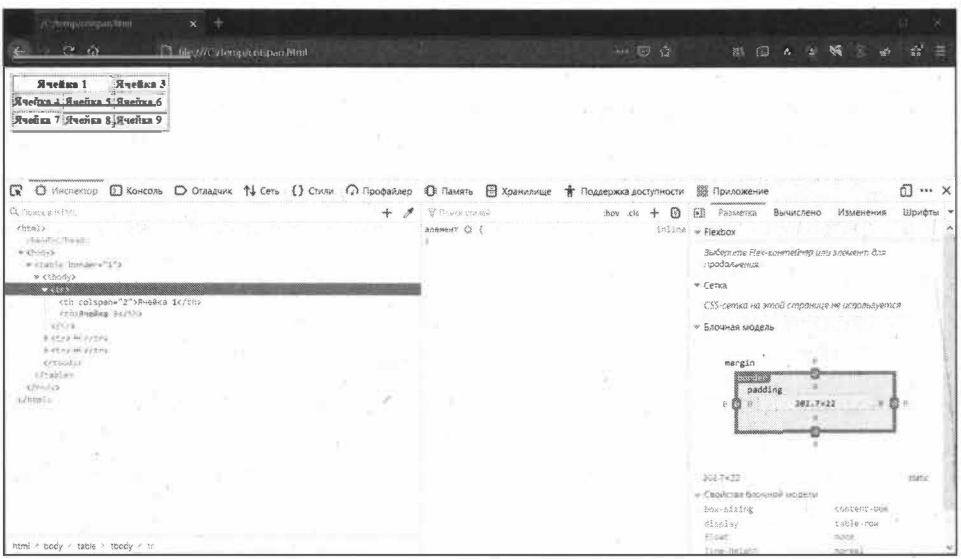


Рис. 2.6. Таблица с объединенными ячейками в одной строке

При объединении ячеек в сроке все последующие ячейки сдвигаются вправо. Поэтому если в рассматриваемой таблице не убрать вторую ячейку (или третью), то таблица примет следующий вид:

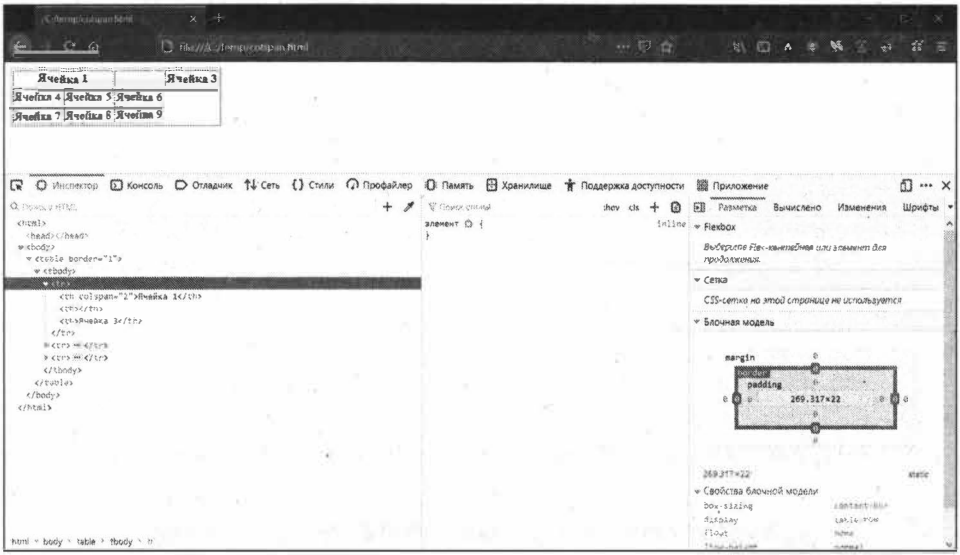


Рис. 2.7. Демонстрация сдвига следующих после объединения ячеек

Аналогичная ситуация с объединением соседних ячеек одного столбца:

```
<table border="1">
   <tr>
      <th>Ячейка 1</th>
      <th>Ячейка 2</th>
      <th>Ячейка 3</th>
   </tr>
   <tr>
      <td>Ячейка 4</td>
      <td rowspan=2>Ячейка 5</td>
      <td>Ячейка 6</td>
   </tr>
   <tr>
      <td>Ячейка 7</td>
      <td>Ячейка 9</td>
   </tr>
</table>
```

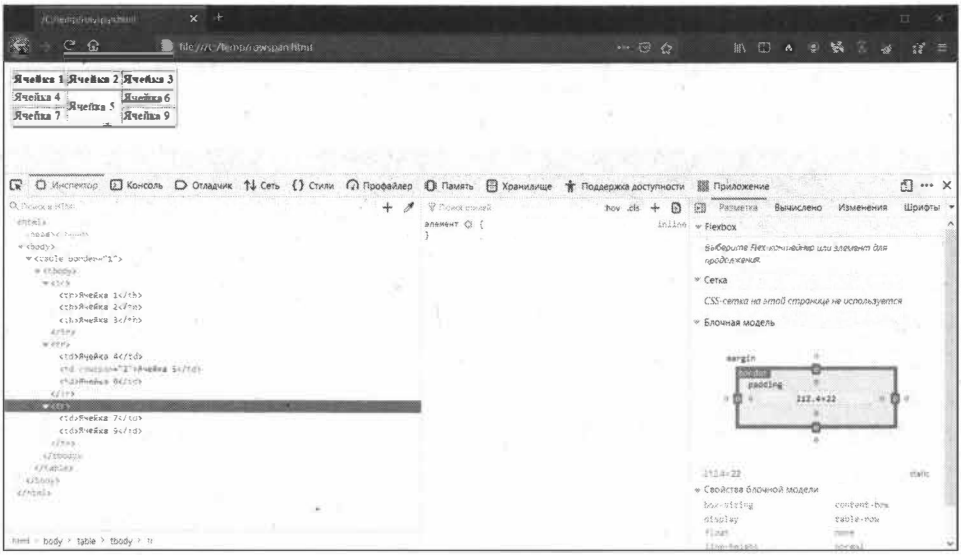


Рис. 2.8. Таблица с объединенными ячейками в одном столбце

Семантическая разметка таблицы

В HTML5 было существенно расширено семантическое разбиение таблицы – появились возможности, с помощью новых специальных тегов, объединять группы ячеек таблицы в смысловые (семантические) группы. Тегами семантической разметки таблицы являются следующие:

- Тег CAPTION – определяет название (заголовок) таблицы. Если этот элемент используется, он всегда должен быть первым вложенным элементом тэга <table>. Внешний вид и расположение заголовка по отношению к самой таблице может быть изменено с помощью стилей CSS caption-side и text-align.
- Тег COLGROUP – позволяет группировать столбцы таблицы. Таблица может отображаться вся сразу, а может последовательно. В первом случае браузер ждет загрузки содержимого всех ячеек таблицы и только после этого отображает таблицу. Для убыстрения загрузки рекомендуется организовывать последовательное отображение таблицы. В этом случае сначала загружаются и отображаются границы таблицы (если они есть), затем по очереди загружается и отображается содержимое ячеек. При этом загрузка всей таблицы по времени не меняется, но зато значительно уменьшается время ожидания первого представления таблицы (первой

отображенной информации), что делает ее загрузку гораздо привлекательней. Для обеспечения последовательного отображения таблицы браузеру необходимо указать ее предварительные размеры. Для этого и используются теги COLGROUP. У данного тега есть атрибут span = "число", который и задает количество столбцов, попадающих в область действия тега **colgroup**. Тег **colgroup** используется после тега caption (при наличии) и перед тегами строки (tr) или группы строк.

- Тег TBODY – своим открывающим и закрывающими тегам выделяет группу строк ячеек таблицы, являющихся основным содержательным телом таблицы.
- Тег TFOOT – своим открывающим и закрывающими тегам выделяет группу строк ячеек таблицы, являющихся подвалом (нижним колонтитулом) таблицы. В таблице может быть только один тег TFOOT
- Тег THEAD – своим открывающим и закрывающими тегам выделяет группу строк ячеек таблицы, являющихся заголовком (верхним колонтитулом) таблицы. Тег THEAD должен включать хотя бы одну строку TR с ячейками TH и TD.

Пример задания таблицы с полной семантической разметкой:

```
<table border="1">
   <caption>Заголовок таблицы </caption>
   <colqroup id="business">
      <col span="2">
      <col span="1">
   </colqroup>
   <thead>
      <tr>
      <th>Параметр</th><th>Значение</th><th>Значение по
умолчанию</th>
      </tr>
   </thead>
   <tfoot>
      <tr>
         <td colspan="3">Разработал: Петров Васисуалий
Эдиктович</td>
      </tr>
   </tfoot>
   <tbody>
```

```
    <tr>
      <td>Продажи 2020</td><td>120000 млн.</td><td>100000
млн</td>
    </tr>
    <tr> ... дополнительные ячейки данных ... </tr>
    <tr> ... дополнительные ячейки данных ... </tr>
  </tbody>
</table>
```

2.12. Вставка изображений в интернет-страницу

Внедрение иллюстративных изображений в HTML осуществляется посредством тега IMG, имеющего следующие атрибуты:

- **src** - обязательный атрибут, задающий URL-адрес (полный или относительный) расположения изображения
- **width**, **height** - геометрические размеры изображения, задаваемые либо в пикселах, либо в процентах видимого пространства окна браузера
- **alt** - атрибут, имеющий в качестве значения текстовую строку, которая является альтернативной текстовой информацией текущего изображения.
- **usemap** – использовать изображение совместно с клиентской навигационной картой, имя которой указывается в качестве значения этого атрибута.
- **ismap** – использовать изображение совместно с серверной навигационной картой, имя которой указывается в качестве значения этого атрибута.

Тег IMG используется только с открывающим тегом. Пример:

```
<IMG src="Img0001.jpg" width="200">
```

2.13. Базовые абстракции разметки. Верстка на основе блоков DIV

В общем случае все HTML-элементы подразделяются на элементы уровня блока (или блокообразующие элементы) и на строковые элементы. Различие между ними заключается в следующем:

- Элементы уровня блока представляют собой ярко выраженные элементы структуры. В своем содержимом такие элементы могут иметь как просто какую-либо информацию, так и другие блокообразующие или строковые элементы. Элементом уровня блока, например, является элемент абзаца P, в содержимом которого могут быть заданы другие абзацы, списки, таблицы и т.д., которые буду по отношению к нему дочерними.
- Строковые элементы представляют собой не столько элемент структуры, сколько саму информацию, обладающую определенным свойством (их еще называют элементами встраиваемой информации). Например, элемент I не создает структурное подразделение, а просто утверждает, что текст в его содержимом должен отображаться курсивом. В содержимом строковых элементов не могут быть блокообразующие элементы, только строковые.

Разность между строковыми элементами и элементами уровня блока хорошо заметна при использовании каскадных таблиц стилей (CSS): под блокообразующие элементы выделяются стилевые блоки (прямоугольные области экрана, обладающие определенными визуальными свойствами). Подробнее об этом читайте в главе, посвященной CSS3.

Строка браузера, в которой присутствует элемент уровня блока, не может содержать ничего, кроме него. Например, если на строке расположен заголовок, то на ней уже не может быть никакой другой информации.

Универсальным элементом уровня блока является элемент DIV, универсальным строковым элементом - SPAN. Смысл их существования заключается в том, что они не имеют никаких своих специфических атрибутов и свойств, а основным своим назначением имеют привнесение в HTML-документ структуры. Выделять некую информацию в блок осуществляется с той целью, чтобы потом к ней можно было бы целенаправленно обратиться: задать определенное свойство с помощью каскадных таблиц стилей, ис-

пользовать при работе сценария и т.п. Использование обычных, не универсальных элементов зачастую бывает не совсем удобно ввиду того, что они, в соответствии со своим назначением, имеют свои особенности. Например, блокообразующий элемент абзаца P выделяет свое содержимое сверху и снизу дополнительными пустыми строками.

Элементы DIV и SPAN обладают одинаковым набором необязательных глобальных атрибутов (см. конец данной главы)

2.14. Семантические теги HTML5. Верстка на основе семантической разметки

В последних релизах языка верстки HTML, в особенности версии 5, была добавлена поддержка большого количества семантических тегов. Данные изменение имеют наибольшее значение для верстальщиков, веб-разработчиков и других пользователей данного языка. Использование семантических тегов, позволяет сделать разметку более читаемой, более понятной человеку, что упрощает редактирование и поддержку подобных файлов. С другой стороны, для браузеров, в большинстве случаев, использование данных тегов аналогично использованию тега .

Использование семантических тегов позволяет оценить качество выполненной верстки. Сравните два одинаковых фрагмента разметки. В первом случае, не используются семантические теги:

```
<div>
   <div>
      …
   </div>
   <p>…</p>
</div>
<div>
   <h3>…</h3>
   <a>…</a>
</div>

<div>
…
</div>
```

Попытка проанализировать данный фрагмент разметки с точки зрение семантики практически невозможно, т.к. подобная структура может встречаться практически в любой области HTML документа, например, в шапке, в футере и в основном теле странице – теге <body>.

Во втором случае, используются семантические теги:

```
<section>
   <nav>
      ...
   </nav>
   <p>...</p>
</section>

<aside>
   <h3>...</h3>
   <a>...</a>
</aside>

<article>
...
</article>
```

Анализируя данный фрагмент разметки, можно сделать несколько выводов:

1. В разметке имеется три самодостаточных блока <section>, <aside>, <article>
2. Блок <section> представляет собой автономный раздел страницы. Если проводить аналогии со страницей-лэндингом, то это может быть раздел "Наши преимущества" и раздел "Наши услуги"
3. В блоке <aside> описано содержимое, которое может быть связано только косвенно с основным содержанием страницы. Наиболее частое применение – разметка боковых панелей.
4. Блок <article> содержит часть документа, которая может быть использована повторно в любой другой части документа, т.к. представляет собой самодостаточный фрагмент. В качестве примера можно привести запись в блоге или фрагмент статьи.

Далее приведем описание тегов семантической разметки HTML5, а в главе, посвященной рассмотрению фронтенда практического примера, будет

показано их практическое применение. Обратите внимание, что сами семантические теги практически никак не определяют внешний вид своего содержимого. Например, классический тег BODY является по своей сути семантическим тегом – он говорит, что все, что находится между его открывающим и закрывающим тегами является телом html-документа. И все. Внешний вид можно настраивать с помощью стилевых CSS-таблиц, которые применяются к этому тегу, но сам тег этим "не занимается".

- Тег ARTICLE – своими открывающим и закрывающим тегами определяет самостоятельную часть документа, страницы или сайта, предназначенную для независимого использования и обособленного восприятия – это может быть статья в блоге, статья в журнале или газете, какой-либо другой самостоятельный фрагмент содержимого. Теги ARTICLE могут быть вложенными.

- Тег ASIDE представляет собой часть документа, чье содержимое только косвенно связанно с основным содержимым документа. Чаще всего представлен в виде боковой панели, сносок или меток. Внешний вид содержимого тега ASIDE (а между его открывающим и закрывающим тегами может быть все что угодно в плане HTML) никак не определяется самим тегом ASIDE, данным тегом просто придается определенный смысл фрагменту html-документа, а выделяется этот фрагмент. А вот как именно он будет отображен – в виде боковой панели, сноски или еще как-то – это вы уже сами зададите с помощью таблиц стилей.

Пример:

```
<article>
  <p>
    Стандарт управления проектами PMP был разработан
американским институтом PMI
  </p>
  <aside>
    <p>
      Что примечательно, создатели первой версии стандарта
потом отказались от последующих версий PMP, так как, по их
мнению, те не соответствуют первоначальной идее стандарта.
    </p>
  </aside>
  <p>
    Стандарт PMP содержит в себе следующие области знаний ...
  </p>
</article>
```

- Тег HEADER – своими открывающим и закрывающим тегами определяет заголовочную часть HTML-документа. Обычно в этой части (в качестве содержимого тега HEADER) располагаются теги навигации, изображение логотипа, заголовок с названием сайта и т.п. Внутри содержимого могут быть любые теги за исключением тегов HEADER и FOOTER.
- Тег FOOTER представляет собой нижний колонтитул (футер, подвал), содержащий информацию о документе, его авторе, контакты и т.п. Такая область (набор тегов) ограждается открывающим и закрывающим тегами FOOTER и называется "подвалом" или "футером". Подвал обычно располагается внизу интернет-страницы (html-документа).
- Тег SECTION – своим открывающим и закрывающим тегами задает раздел – тематическую группу содержимого интернет-страницы – который должен восприниматься обособленно. Секция может иметь свою собственную обособленную структуру, в частности в секции может быть свой заголовок (тег HEADER) и подвал (тег FOOTER).
- Тег NAV своим открывающим и закрывающим тегами задает область html-документа, в которой собраны теги (ссылки) навигации (как внутри текущего документа, так и ведущих на другую страницу).

Пример семантической верстки:

```
<html>
   <head>
      <title> Название документа </title>
   </head>
   <body>
<header>
  Шапка сайта
</header>
<nav>
  Навигация
  <ul>
    <li><a href="#">Ссылка 1</a></li>
    <li><a href="#">Ссылка 2</a></li>
  </ul>
</nav>
<section>
  Секция 1
  <article>
    <h1>Заголовок статьи</h1>
    <p>Контент</p>
       <h2>Подзаголовок статьи</h2>
```

```
    </article>
  </section>
  <section>
    Секция 2
    <article>
      <h1>Заголовок статьи 2</h1>
      <p>Контент</p>
         <h2>Подзаголовок статьи</h2>
    </article>
  </section>

  <footer>
    <p>Дата, адрес, авторство</p>
  </footer>
  </body>
  </html>
```

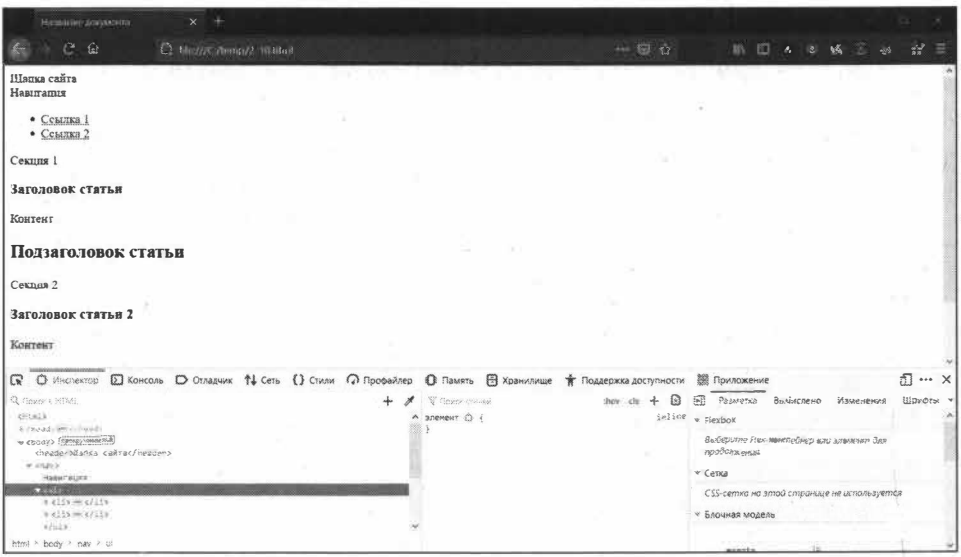


Рис. 2.10. Пример полной семантической верстки

2.15. Глобальные атрибуты HTML5

При рассмотрении различных HTML-тегов мы изучали атрибуты, свойственные тому или иному тегу. Однако есть набор атрибутов, называемых глобальными атрибутами, которые применимы абсолютно ко всем тегам HTML. То есть для любого тега могут быть заданы любые глобальные атрибуты. С двумя из глобальных атрибутов мы уже с вами знакомы – это атрибуты *id* и *class*. Осталось познакомиться с остальными. Это полезно.

- **accesskey = "символ"** – позволяет назначить клавишу на клавиатуре, нажатие на которую активирует или выделяет соответствующий тег. Допускается возможность указания нескольких символов. Содержимое данного атрибута – список разделенных пробелами символов. Браузер будет использовать первый имеющийся в раскладке клавиатуры символ из списка.
- **contenteditable** – перечислимый атрибут, указывающий, нужно ли предоставить пользователю возможность редактировать содержимое соответствующего тега. Поддерживаются следующие значения:
  - `true` редактирование разрешено;
  - `false`, редактирование не разрешено.
- **contextmenu** – атрибут, который в качестве своего значения принимает id тега MENU, который следует использовать в качестве контекстного меню для данного элемента. Использование тега MENU на данный момент не стандартизировано разработчиками браузеров, а потому не рекомендуется.
- **dir** – позволяет указать направление текста, выбранное для содержимого тега. Поле dir может принимать одно из следующих значений:
  - `ltr`, что расшифровывается как *left to right* – слева направо;
  - `rtl`, что расшифровывается как *right to left* – справа налево;
  - `auto` — направление будет выбрано автоматически.
- **draggable** – позволяет определить, можно ли перетаскивать содержимое соответствующего тега или нет. Принимает следующие значения:
  - `true` -- можно перетаскивать;
  - `false` – нельзя перетаскивать.

- **dropzone** - данный атрибут позволяет задать действия, которые выполняются после перетаскивания данных на содержимое соответствующего тега, а также задает допустимый тип перетаскиваемых данных. Поддерживаемые значения:
  - `copy` – перетаскивание создаст копию перетаскиваемого элемента;
  - `move` – перетаскиваемый элемент будет перемещен в новое расположение;
  - `link` — перетаскивание создаст ссылку на перетаскиваемые данные.
  - `file: тип файла` – задается допустимый тип файла (например, file: image/png).
- **hidden** – логический атрибут, позволяющий скрыть элемент. Атрибут просто указывается, без значения
- **style** – Содержит описание стилей CSS, которые должны быть применены к элементу.
- **tabindex** – атрибут, который в качестве значения содержит число от 0 до 32767, определяющее порядковое место соответствующего тега в последовательности перехода между тегами html-страницы при последовательном нажатии клавиши "Tab"
- **title** - в качестве значения содержит текст (например, title = "Заголовок от Васи"), предоставляющий дополнительную информацию о теге или его название.
- **translate** – задает, можно ли переводить содержимое тега при автоматическом переводе всей страницы или нет. Может принимать значения:
  - *пустая строка* или `"yes"` указывает, что содержимое тега должно быть переведено;
  - `"no"` указывает, что содержимое тега не должно быть переведено.

Глава 3.

ОСНОВЫ CSS3

Таблица стилей представляет собой написанный на языке CSS3 набор правил, применяемых к HTML-тегам документа, к которому эта таблица подключена. Эти правила задают визуальное представление содержимого HTML-тегов, иначе говоря, то, как они будут выглядеть на экране монитора (цвет, рамки, размер, выравнивание, шрифт, отступы, видимость и т.п.). Благодаря CSS3 эти параметры не надо задавать отдельно для каждого тега через его атрибуты. Таким образом, каскадные таблицы стилей позволяют отделять содержимое HTML-документа от описания его внешнего вида. А благодаря этому достигается значительное удобство, компактность описания визуальных свойств HTML-элементов и оперативность их изменения.

Для того чтобы браузер применял правила какой-либо таблицы стилей к HTML-документу, необходимо привязать таблицу и документ друг к другу.

Существует четыре метода, которыми это можно сделать:

- Внедрение - задание таблицы стилей непосредственно в заголовке самого HTML- документа, в качестве содержимого тега STYLE
- Присоединение - таблица стилей находится во внешнем файле и присоединяется к HTML-документу через тегLINK. При этом CSS-файл с внешней таблицей стиля всегда сопровождает HTML-файл документа.

- Импортирование - суть этого метода заключается в том, что текст таблицы стилей, находящейся во внешнем файле на сервере, импортируется с помощью Css-свойства @import внутрь текста HTML-файла.

- Поэлементное задание стиля, когда для всех HTML-тегов определен атрибут STYLE. Через него, используя синтаксис CSS3, можно задавать (или переопределять) стиль для каждого тега индивидуально.

3.1. Правила CSS

В основе всех версий CSS лежит общий для всех набор синтаксических правил. Именно благодаря этому обстоятельству обеспечивается преемственность различных версий CSS. В том случае, если браузер поддерживает ранние версии CSS и не поддерживает новые возможности, то последние будут просто им проигнорированы. Однако правила, которые браузер в состоянии применить, он применит.

Все обычные правила каскадных таблиц стилей состоят из двух частей: селектора и блока определений и имеют следующий вид:

селектор {блок определений}.

Селектором служит название HTML-элемента, или комбинация названий HTML-элементов, для которых и задается правило форматирования. Блок определений начинается с левой фигурной скобки "{" и заканчивается правой фигурной скобкой "}". Все, что находится перед левой фигурной скобкой считается селектором.

Пример:

Между фигурными скобками блока определений располагаются объявления. Они имеют следующий общий вид:

название свойства: значение свойства

Причем между "названием", двоеточием и "значением" может находиться любое количество пробелов.

Несколько объявлений для одного селектора могут быть объединены в группы, отделяясь друг от друга точкой с запятой.

Например, следующий набор правил:

```
P{font-size:14 pt}
P{font-family: Arial}
P{font-color: yellow}
P{background-color: blue}
```

эквивалентен одному такому правилу:

```
P     { font-size:14 pt;
font-family: Arial;
font-color: yellow;
background-color: blue}
```

Объявление может не определять никакого свойства. При этом оно называется пустым.

Например, IMG {}

```
IMG - селектор, {}- блок определений
```

3.2. Селектор классов

Благодаря использованию селектора классов разработчик может обращаться к группе разнородных HTML-элементов, принадлежащим к одному

классу (имеющим одноименное значение атрибута *class*). Селектор классов имеет следующий общий вид:

- сначала, через запятую, указываются названия HTML-элементов, а затем, через точку, следует имя класса, к которому эти элементы должны принадлежать, чтобы к ним было применено данное правило.
- тег.имя класса {определение; определение; определение}

Например, чтобы правило применялось ко всем элементам, принадлежащим к классу *superclass*, используется следующая запись: ".superclass {определение}"

Или другой пример:

```
<!DOCTYPEhtml>
<html>
   <head>
      <title>
      Пример применения селектора классов
      </title>
      <style type ="text/css">
      h3{color:green}
      h3.titlepage{color:red}
      h3.indexpage{color:blue}
      </style>
   </head>
   <body>
      <h3> Заголовок 1</h3>
      <h3 class=titlepage> Заголовок 1</h3>
      <h3 class=indexpage> Заголовок 1</h3>
   </body>
</html>
```

При таком задании, все заголовки H3, принадлежащие к классу *titlepage* (class = titlepage) будут отображаться красным цветом. Принадлежащие к классу *indexpage* (class = indexpage) – синим цветом. Все остальные H3-заголовки будут иметь зеленый цвет.

Можно также использовать классы и без указания тега. При такой записи класс можно применять к любому тегу.

.Имя класса {определение; определение; определение}

3.3. Селектор ID-имен

СелекторID-имен в своем применении аналогичен селектору классов. Отличие заключается в том, что селектор ID-имен фильтрует HTML-теги не по классу (атрибуту class), а по id-именам (атрибуту id).

Синтаксис селектора ID-имен имеет следующий вид: сначала пишется имя HTML-тега, а затем, через символ "#", прописывается его id-имя.

тег#id_имя {определение; определение; определение}

Например, правило:

```
P#famous {font-family: Heretz}
```

будет применено только к тем HTML-тегам P, у которых для атрибута **id** указано значение "famous".

Чтобы обратиться ко всем HTML-тегам, имеющим атрибут id = "famous", достаточно следующей записи:

```
#famous{определение;
      определение;
        .....
      определение}
```

В одном правиле может содержаться несколько селекторов. Допустим, разработчику требуется применить одно и тоже правило к нескольким разным HTML-тегам, не объединённым в классы и имеющим разные id-имена. Можно, конечно, написать отдельное правило для каждого такого элемента. но более эффективным и правильным будет перечислить, через запятую, селекторы HTML-элементов в списке селекторов одного правила.

Например,

```
H1, P, Q {color: lightgrey;
          font-family: Arial}
.my-class, B {color: black}
```

3.4. Разметка и форматирование документа средствами CSS3

Одной из основных технологий, составляющих суть языка CSS3, является блочная модель отображения документов. Согласно этой модели HTML-теги уровня блока, состоящие в иерархическом дереве документа, отображаются в виде прямоугольных блоков, к которым могут быть применены стилевые установки.

Сам блок представляет собой несколько вложенных друг в друга прямоугольных областей. В самой внутренней области и помещается информативное содержание HTML-тега.

Универсальным элементом уровня блока является тегDIV, универсальным строковым элементом-тег SPAN. Смысл их использования заключается только в привнесении структуры в HTML-документ.

Размеры

Размеры прямоугольной области экрана, занимаемой стилевым блоком HTML-тега, выделяются как сумма значений следующих свойств:

- горизонтальный размер:
  - » левое поле (margin-left)
  - » левая граница (border-left)
  - » левый отступ (padding-left)
  - » ширина информативной области (атрибут *width* блокообразующего элемента, который также может указываться в таблице стилей)
  - » правый отступ (padding-right)
  - » правая граница (border-right)

» правое поле (margin-right)

- вертикальный размер вычисляется аналогично горизонтальному с той разницей, что вместо правых и левых (left и right) суммируются верхние и нижние (top и bottom) размеры соответствующих областей

Геометрические размеры отступов так же, как и в случае с полями, можно задать либо каждый отдельным правилом, либо все сразу.

Ниже перечислены свойства, задающие размеры каждого поля в индивидуальном порядке:

- padding-left: размер левого отступа
- padding-right: размер правого отступа
- padding-top: размер верхнего отступа
- padding-bottom: размер нижнего отступа

Ширина отступов задается либо для каждой из них индивидуально, либо для всех четырех сразу.

Границы

Ширина отдельных границ задается следующими свойствами:

- border-top-width: ширина верхней границы
- border-bottom-width: ширина нижней границы
- border-right-width: ширина правой границы
- border-left-width: ширина левой границы

Ширина всех четырех границ может быть указана с помощью одного свойства *border-width*. Значения этих свойств либо указываются только в стандартных для CSS3 единицах длины, либо в качестве значений используются специально зарезервированные ключевые слова языка CSS3:

- thin-тонкая линия границы (2 пикселя)

- medium – средняя линия границы (4 пикселя)
- thick-толстая линия границы (6 пикселей)

Цвет границ определяется следующими свойствами:

- border-top-color: цвет верхней границы
- border-bottom-color: цвет нижней границы
- border-right-color: цвет правой границы
- border-left-color: цвет левой границы

Цвет для всех четырех границ сразу можно указать с помощью свойства *border-color*. Порядок задания цветов отдельных границ при этом аналогичен порядку задания ширин для свойства *border-width*. В том случае, если цвет границ не указан, они отображаются цветом, установленным для свойства *color* блокообразующего элемента.

Для границ должен быть также задан тип линии, которой она отображается. Ниже приведены допустимые в CSS3 названия типов и описан соответствующий им видимый результат:

- **none** - граница определена как отсутствующая. При таком задании значение border-width устанавливается равным нулю.
- **solid** - граница будет отображаться в виде одной сплошной линии. Если атрибут border-width не задан, то по умолчанию используется значение border-width: medium
- **double** - граница будет очерчена двумя сплошными линиями. Причем свойство border-width будет показывать суммарную толщину двух этих линий и расстояния между ними.
- **dotted** - граница будет отображена в виде пунктирной линии.
- **groove** - граница будет показана как вдавленная линия.
- **ridge** - граница будет отображена в виде выпуклой линии.
- **inset** - при задании такого типа границ весь блок будет отображен как вдавленный.

- **outset**- указание этого значения при задании типа границ приведет к тому, что весь блок будет отображен как выпуклый.

Цвет и фон

Как уже упоминалось, каскадные таблицы стилей были задуманы для наиболее эффективного задания внешнего вида содержимого HTML-элементов, который они принимают при отображении в окне браузера. А что же, как не цветовое оформление является основной визуальной характеристикой. Для каждого отображаемого HTML-тега может быть указано два цвета: цвет фона и цвет переднего плана. На переднем плане отображается информативное содержимое тега, проще говоря - текст. В CSS3 цвет текста задается единственным, предназначенным для этих целей свойством *color*.

Для описания фонового оформления элемента в CSS3 определено несколько возможностей.

Фон можно задать с помощью следующих свойств:

- **background-color**: указывает цвет фона;
- **background-image**: указывает фоновое изображение. Причем для этого случая может быть задано несколько дополнительных параметров фона:
  - » background-repeat: указывает наличие или отсутствие дублирования фонового изображения как по вертикали, так и по горизонтали
  - » background-attachment: это свойство указывает на то, будет или нет фоновое изображение прокручиваться при прокрутке документа.
  - » background-position: задает положение изображения в рамках стилевого блока, к которому оно принадлежит.
  - » background-origin – определяет область позиционирования фонового рисунка.
  - » background-size – масштабирует фоновое изображение, согласно заданным размерам.
  - » background-clip – определяет, как цвет фона или фоновая картинка должны выводиться под границами.

3.5. Форматирование текста средствами CSS3

Итак, со стилевыми блоками и их цветовыми и фоновыми характеристиками разобрались. Теперь остается описать разметку текста, располагающегося в информативной области блоков. С помощью средств языка CSS3 можно задать отступы и высоту строк в текстовых фрагментах, расстояния между словами и буквами. Возможности CSS3 также позволяют трансформировать текст, написанный строчными буквами в прописной текст, и наоборот. С помощью свойства *text-decoration* можно указать подчеркивание, зачеркивание или надчеркивание текста.

Общепринятым считается тот факт, что от выбора подходящего шрифта может во многом зависеть привлекательность документа. Каждый из них имеет свой индивидуальный внешний вид. Подключение определенного шрифта и задание его визуальных характеристик (прямой, курсив, жирный и т.п.) является одними из наиболее часто производимых разработчиком операций. Язык каскадных таблиц стилей CSS3 предлагает для этих целей свои возможности.

Отступы

С помощью свойства *text-indent* может быть задан отступ первого слова в первой строке абзаца. Или, говоря другими словами, определяет размер красной строки. Сдвиг производится относительно левого края текстового фрагмента (т.е. относительно левого края информативной области стилевого блока).

Выравнивание

Свойством *text-align* определяется выравнивание текста внутри информативной области стилевого блока. С помощью него текст может быть выровнен:

- left – по левому краю;

- right – по правому краю;
- center – по центру;
- justify – по ширине;
- start – аналогично значению *left*, если текст идет слева направо, и *right*, если текст идет справа налево;
- end – алогично значению *right*, если текст идет слева направо, и *left*, если текст идет справа налево.

Визуальные характеристики

Как уже упоминалось выше, с помощью свойства *text-decoration* можно придать тексту такие визуальные характеристики как: подчеркивание, зачеркивание и надчеркивание. Для этого достаточно использовать свойство *text-decoration* с одним из следующих значений:

- **underline** – каждая строка текста будет подчеркнута;
- **overline** – все строки текста будут отображены с чертой, расположенной над каждой их них;
- **line-through** – каждая строка будет отображена перечеркнутой;
- **none** – никакого декоративного оформления текста производиться не будет. Это значение выставлено по умолчанию.

Внутритекстовые интервалы

Управление размером межбуквенных интервалов осуществляется с помощью CSS-свойства *letter-spacing*, которое устанавливает расстояние между буквами, задаваемое в единицах длины.

Например,

```
H3 {color: blue;
letter-spacing: 13px}
```

Свойство **letter-spacing** может быть выставлено в значении *normal* (letter-spacing: normal), что указывает браузеру использовать стандартные интервалы текущего шрифта.

Размер интервалов между словами задается с помощью свойства **word-spacing**.

Размер междустрочных интервалов задается с помощью свойства **line-height**.

Регистр

С помощью свойства **text-transform**, определённого в рамках спецификации CSS3, можно осуществлять преобразование регистра букв текста. Или, другими словами, трансформировать строчные буквы в прописные, и наоборот. Эта возможность реализуется свойством **text-transform**, которое в соответствии со своим назначением может принимать следующие значения:

- *capitalize* - первая буква каждого слова отображается прописной (заглавной)
- *uppercase* - все буквы каждого слова отображаются прописными (заглавными)
- *lowercase* - все буквы каждого слова отображаются как строчные
- *none* - никакого преобразования регистра не производится.

Пример использования:

```
h1 {color: green;
      letter-spacing: 5px;
text-transform: uppercase}
```

3.6. Настройка шрифта средствами CSS3

С помощью свойства **font-family** браузеру указывается шрифт, которым он должен отображать текстовую информацию, для которой это свойство определено (или наследуется).

Например,

```
p {font-family:Arial}
```

В результате такого задания текст абзацев будет отображаться шрифтом Arial.

Возможности свойства **font-family** таковы, что оно позволяет указывать приоритетный список названий шрифтов, каждый из которых будет пробовать подключаться в соответствии со своим приоритетом.

Например, правило

```
body {font-family: Heretet, Times New Roman, Arial}
```

означает, что сначала браузер будет пробовать использовать шрифт Heretet. Если такой шрифт отсутствует в его распоряжении, (его просто нет на компьютере пользователя), то будет произведена попытка использования шрифта TimesNewRoman и т.д.

Стиль шрифта

С помощью свойства **font-style** можно задавать стиль шрифта. Рассмотрим значения, принимаемые свойством **font-style** и соответствующий им эффект:

- normal – шрифт отображается в своем обычном, прямом виде. Это значение используется по умолчанию.
- italic – шрифт будет отображен курсивом
- obligue – шрифт будет отображен в наклонном виде

Размер шрифта

Благодаря CSS-свойству font-size реализуется возможность задания размера шрифта. В качестве значения свойства font-size могут выступать либо ключевые слова, либо единицы длины, характеризующие высоту шрифта (ширина изменяется пропорционально высоте).

Для свойства font-size определена следующая последовательность ключевых слов:

xx-small, x-small, small, medium, large, x-large,xx-large

Последовательность задана по возрастанию. Это значит, что шрифт размером small меньше шрифта размером medium.

Однако лучше задавать шрифт в типографских пунктах (pt). Именно в этих единицах измерения задается размер шрифта во всех текстовых редакторах.

Пример:

```
B {font-size: 14 pt}
```

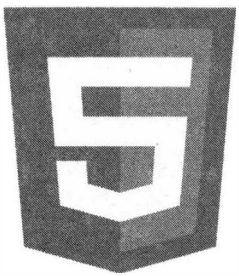
Жирность шрифта

Жирность шрифта, которым отображается текст, может быть задана через свойство **font-weight**. Каждая из этих девяти градаций задается одним числом: 100,200,300, ..., 900. Значение 100 соответствует самому бледному варианту шрифта, значение 900 – самому жирному.

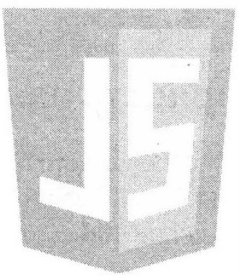
Кроме этого, допустимо использование следующих ключевых слов:

- normal – задает нормальную (обычную) жирность шрифта. Это значение используется по умолчанию.
- bold - указывает на использование полужирного варианта шрифта.

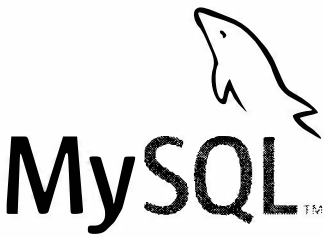
HTML

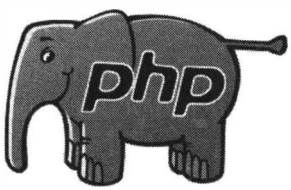


JS



CSS





Глава 4.

Основы JavaScript

4.1. Объектная модель JavaScript

В JavaScript используется объектная модель документа, в рамках которой каждый HTML-контейнер можно рассматривать как совокупность свойств, методов и событий, происходящих в браузере. По сути, это связь между HTML-страницей и браузером. Пока мы не будем вникать в ее подробности, а желающие "бежать впереди паровоза" могут узнать больше по следующей ссылке:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model

Если вы не знакомы с объектно-ориентированным программированием, тогда вы, наверное, не особо понимаете, о чем идет речь, когда мы говорим об объектах и методах. Не расстраивайтесь, пока принимайте все как есть, а дальше вы поймете, что к чему. В JavaScript все основано на классах и объектах (поскольку это объектно-ориентированный язык) и без них вы не сможете написать свои программы. Именно поэтому мы начали разговор об объектной модели JavaScript в этой вводной главе.

Объект можно воспринимать как совокупность данных и методов (функций) для их обработки. В JavaScript с некоторыми объектами также связываются определенные события.

Давайте разберемся, что такое объект, как говорится, "на пальцах". Представим, что объект - это человек. Путь наш объект называется **Human**. У такого объекта может быть масса характеристик: имя, пол, дата рождения и т.д. Все это называется свойствами объекта. Обратиться к свойству можно так:

объект.свойство

Например:

```
Human.Sex = 'M';
Human.Name = 'Николай';
Numan.YearOfBirth = '1976';
```

Метод – это функция обработки данных. Например, метод, возвращаемый год рождения человека может называться GetYear. Обращение к методу производится так:

объект.метод(параметры);

Например:

```
Human.GetYear();
```

Связать метод с функцией можно так:

объект.метод = имя_функции;

С объектом может быть связано какое-нибудь событие. Например, при рождении человека может генерироваться событие **OnBirth**. Для каждого события можно определить его обработчик – функцию, которая будет реагировать на событие. Что будет делать эта функция, зависит от события. Например, обработчик OnBirth может заносить в некую общую таблицу базы данных общую информацию об объекте: имя, пол и дату рождения. Такая таблица может использоваться для ускорения поиска нужного объекта.

Теперь рассмотрим еще одно, более абстрактное понятие – класс. Класс – это образец объекта, если хотите, класс – это тип переменной объекта. Пусть мы разработали класс **Human**, тогда объект, то есть экземпляр класса Human, может называться Human1. Вы можете создавать несколько объектов класса Human – имена у них будут разными:

```
Rodion = new Human;              // создаем объект Rodion
класса Human
```

```
Mary = new Human;          // создаем объект Mary класса Human
```

В этой книге мы еще поговорим о создании объектов, а сейчас приведенной информации вполне достаточно для восприятия следующего материала.

4.2. Комментарии в JavaScript

Комментарии в JS могут быть однострочными и многострочными. Однострочный комментарий начинается с двух знаков //, с этим типом комментариев вы уже знакомы:

```
// комментарий
i++;    // увеличиваем i
```

Комментарием является все, что находится после // и до конца строки.

Многострочный комментарий начинается символами /* и заканчивается символами */, например:

```
/* это пример
многострочного
комментария */
```

Какие комментарии использовать - зависит от вас. Однострочные комментарии удобно использовать для комментирования отдельных строчек кода. Многострочные комментарии подойдут для объяснения того, что делает целый блок, например, описать, что делает функция и какие параметры ей нужно передать.

4.3. Диалоговые окна

Для взаимодействия с пользователем, то есть для ввода данных и вывода результатов работы программы, как правило, используются HTML-формы и возможность вывода HTML-кода прямо в документ (метод document. write). Этот способ удобен тем, что вы, используя HTML и CSS, можете

оформить форму ввода данных так, как вам заблагорассудится. То же самое можно сказать и о выводе данных. Из JS-сценария вы можете выводить любой HTML-код, позволяющий как угодно оформить вывод.

Но в некоторых ситуациях этих возможностей оказывается очень много. Иногда нужно просто вывести диалоговое сообщение, например, сообщить пользователю, о том, что введенный им пароль слишком простой или слишком короткий. В этом разделе мы разберемся, как выводить диалоговые окна, позволяющие выводить короткие сообщения (например, сообщения об ошибках ввода) и обеспечивающий ввод данных.

4.3.1. Метод alert() - простое окно с сообщением и кнопкой OK

Метод alert() объекта **window** используется для отображения простого окна с сообщением и одной кнопкой - **OK**. Такое окно может использоваться, например, для отображения сообщений об ошибках (короткий/простой/неправильный пароль). Окно, кроме донесения до пользователя сообщения, больше не предусматривает никакого взаимодействия с пользователем.

Методу alert() передается только одна строка - отображаемая строка. Чтобы отобразить многострочное сообщение, разделяйте строки символом \n:

```
window.alert("Привет, мир!");
window.alert("Привет,\nмир!");
```

Первая наша программа - была не "Hello, world", но давайте не будем изменять традиции и таки напишем эту программу, хотя она будет и не первой - хоть тут мы будем отличаться от всех остальных программистов. Сценарий, демонстрирующий использование метода alert(), приведен в листинге 4.1.

Листинг 4.1. Использование метода alert()

```
<html>
<head>
  <title>Alert</title>
</head>
<body>
```

```
  <script>
    window.alert("Привет, мир!");
  </script>
</body>
</html>
```

Наш сценарий находится в теле документа (тег<body>), поэтому будет запущен сразу при загрузке HTML-файла.

4.3.2. Метод confirm() - окно с кнопками OK и Cancel

Другой часто используемый метод - метод confirm(). Он выводит окно с сообщением и двумя кнопками - **OK** и **Cancel**, позволяя пользователю выбрать одну из кнопок. Проанализировав возвращаемое методом значение (*true*, если нажата кнопка **OK** и *false* - в противном случае), вы можете выполнить то или иное действие. Для нашего примера мы будем просто выводить с помощью alert() название нажатой кнопки. Пример использования метода confirm() приведен в листинге 4.2.

Листинг 4.2. Использование метода confirm()

```
<html>
<head>
  <title>Confirm</title>
</head>
<body>
  <script>
   if (window.confirm("Нажмите OK или Отмена")) {
      window.alert("OK");
   }
   else {
      window.alert("Отмена");
   }
</script>
</body>
</html>
```

4.3.3. Метод prompt() - диалоговое окно для ввода данных

Метод prompt() отображает диалоговое окно с полем ввода, сообщением и кнопками **OK** и **Cancel**. Введенное пользователем значение можно потом будет присвоить какой-то переменной. Диалог возвращает введенную пользователем строку. Если пользователь ничего не ввел, диалог возвращает значение *null*.

Методу prompt() нужно передать два параметра - строку, которая будет отображена в качестве приглашения ввода (над полем для ввода данных) и значение по умолчанию, которое будет передано в сценарий, если пользователь поленится ввести строку и просто нажмет **OK**. Пример использования этого диалога приведен в листинге 4.3.

Листинг 4.3. Пример использования метода prompt()

```
<html>
<head>
  <title>Confirm</title>
</head>
<body>
  <script>
  const UName = window.prompt("Как тебя зовут?", "Никак");
  if (UName ==null) {
      alert("Пока!");
  }
  else {
      document.write("Привет, " + UName);
  }
</script>
</body>
</html>
```

Наш сценарий прост. Если пользователь нажмет **Отмена**, то увидит диалог с текстом "пока!" (ну не хотим мы "общаться" с пользователем, который не хочет представиться). Если пользователь нажмет **OK**, то строка из поля

ввода будет отображена в HTML-документе в виде "Привет, <введенный текст>".

4.4. Переменные в JavaScript

4.4.1. Объявление переменной

Переменная – это поименованная область памяти, хранящая данные. В других языках программирования (например, C, PHP) переменные являются типизированными, то есть при объявлении переменной определяется тип данных (число, символ, строка, массив чисел или массив символов и т.д.), которые будут храниться в этой переменной. В JavaScript, как и в PHP, переменные не являются строго типизированными, а это означает, что тип переменной зависит от данных, которые в данный момент хранятся в ней.

В отличие от того же PHP, где можно использовать переменную без ее предварительного объявления (хотя это и нежелательно, поскольку переменная не инициализирована), в JavaScript переменную можно объявить с помощью служебного слова **var** (устаревший вариант). Однако, в соответствии с новым стандартом языка JavaScript, при объявлении переменных следует использовать либо служебное слово **let** (для переменных, которые будут изменяться в ходе работы скрипта), либо служебное слово **const** (для неизменяемых переменных), причем, предпочтительно не заменять слово **const** словом **let** там, где это возможно.

Имя переменной должно начинаться с символа буквы (A-Z) или символа подчеркивания. Последующими символами могут быть цифры, буквы, а также знак $. Имя переменной не может начинаться с цифры или с символа $ (как в PHP)! Также нужно помнить, что JavaScript учитывает регистр символов, то есть переменные variable и Variable – это две разные переменные.

4.4.2. Типы данных и преобразование типов

Данные, хранящиеся в переменной, могут быть разного типа. Как вы заметили, при объявлении переменной (как это делается в других языках программирования) тип переменной (данных) не указывается.

В JavaScript переменные могут содержать следующие типы данных:

- **number** - числа, как целые, так и с плавающей точкой.
- **string** - строки.
- **boolean** - логический тип данных, может содержать два значения - *true* (истина) и *false* (ложь).
- **function** - функции. В JS мы можем присвоить ссылку на функцию любой переменной, если указать имя функции без круглых скобок.
- **object** - массивы, объекты, а также переменные со значением *null*.

Тип переменной JavaScript определяет при ее инициализации, то есть при первом присваивании значения, например:

```
Num1 = 5;       // Переменной Num1 присвоено целое значение 5, тип - number
Num2 = 5.5;     // Переменное с плавающей точкой 5.5, тип - number
Str1 = "привет"; // Переменной Str1 присвоено значение "Hello", тип - string
Str2 = 'мир';   // Также можно использовать одинарные кавычки
Str3 = null;    // Переменная не содержит данных, ее тип - object
Bool1 = true;  // Булевая (логическая) переменная со значением true
```

Оператор **typeof** возвращает строку, описывающую тип данных переменной. Давайте продемонстрируем его работу (см. листинг. 4.4). Сценарий из листинга 4.4 объявляет переменные, выполняет их инициализацию (присваивает значения), а затем выводит тип каждой переменной.

Листинг 4.4. Оператор typeof

```
<html>
<head>
  <title>typeof</title>
</head>
<body>
<script>
let Num1, Num2, Str1, Str2, Str3, Bool1;

Num1 = 5;
Num2 = 5.5;
Str1 = "привет";
Str2 = 'мир';
```

```
Str3 = null;
Bool1 = true;

document.write("<br>Num1 - " + typeof(Num1));
document.write("<br>Num2 - " + typeof(Num2));
document.write("<br>Str1 - " + typeof(Str1));
document.write("<br>Str2 - " + typeof(Str2));
document.write("<br>Str3 - " + typeof(Str3));
document.write("<br>Bool1 - " + typeof(Bool1));

</script>
</body>
</html>
```

В процессе вычислений JavaScript производит преобразование типов. Посмотрим, как оно работает. Определим две переменные: одна будет содержать число 1, а вторая – символ "1":

```
let A = 1;
let B = "1";
```

Теперь определим еще две переменные:

```
let SR = B + A;    // string result
let IR = A + B;    // integer result
```

Тип переменной определяется по типу первого присваиваемого значения. В нашем случае переменная SR будет содержать значение "11", поскольку первой была строковая переменная B. Переменная IR будет содержать значение 2 по выше описанным причинам.

Переменные в JavaScript также могут быть булевого (логического) типа. Такие переменные могут принимать одно из двух значений – *true* (истина) или *false* (ложь):

```
let Bool = true;
```

Для принудительного преобразования типов вы можете использовать две следующие функции:

- **parseInt** – преобразует строку в целое число, если это возможно.
- **parseFloat** – преобразует строку в число с плавающей запятой, если это возможно.

- **eval** - вычисляет выражение в строке, как будто это обычное выражение JavaScript.

Рассмотрим несколько примеров:

```
let A = "1";     // строковое значение "1"
let B = parseInt(A);     // переменная B теперь содержит
число 1
let C = "5.1";   // строка "5.1"
let D = parseFloat(C);  // переменной D будет присвоено
число 5.1
let E = "2 + 2";    // строка "2+2"
let F = eval(E);    // переменной F будет присвоено число 4
```

4.4.3. Локальные и глобальные переменные

Как и в других языках программирования, в JavaScript существуют локальные и глобальные переменные.

Локальной называется переменная, объявленная в какой-нибудь функции. Она доступна только в этой функции и недоступна во всем скрипте.

Глобальная переменная объявлена в теле скрипта и доступна во всех объявленных в скрипте функциях.

Глобальные переменные принято объявлять в самом начале скрипта, чтобы все функции наследовали эти переменные и их значения.

4.5. Выражения и операторы JavaScript

4.5.1. Типы выражений

Выражение – это набор переменных, констант, операторов. У любого выражения есть свое значение – результат вычисления выражения. Значение может быть числом, строкой или логическим значением.

В JavaScript есть два типа выражений: которые присваивают значение переменной; и которые просто вычисляют выражение без присваивания его значения переменной:

```
x = 3 * 2

9 - 5
```

Существуют еще так называемые условные выражения. Условные выражения определяются так:

```
(условие)? значение1: значение2
```

Если условие истинно, то выражение имеет значение 1, а если нет – значение 2. Например:

```
sedan = (doors >=4) ? true : false
```

4.5.2. Операторы присваивания

Операторы присваивания, поддерживаемые в JavaScript, описаны в таблице 4.1.

Таблица 4.1. Операторы присваивания

Оператор	Пример	Описание
+=	x += y	x = x + y
-=	x -= y	x = x – y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%= (остаток от деления)	x %= y	x = x % y

4.5.3. Арифметические операторы

Математические (арифметические) операторы в JS такие же, как и в большинстве других языков программирования, а именно:

- + - сложение (например, A = B + C;).
- - - вычитание (например, A = B - C;).
- * - умножение (например, A = B * C;).
- / - деление (например, A = B / C;).
- % - деление по модулю (например, A = B % C);
- ++ - инкремент, увеличивает значение переменной на 1 (например, i++;).
- -- - декремент, уменьшает значение переменной на 1 (например, j--;).

Как используются эти операторы, думаю, вы и так знаете. Нужно отметить только особенность операторов инкремента и декремента. Рассмотрим небольшой пример:

```
x = 7
y = x++
```

Переменной **y** будет присвоено значение **7**, а после этого переменная **x** будет увеличена на **1** (значение 8). Если же ++ указать до **x** (а не после него), то сначала переменная **x** будет увеличена на **1**, а потом уже будет присвоено новое значение переменной **y**:

```
y = ++x
```

4.5.4. Логические операторы

К логическим операторам относятся следующие операции:

- ! – унарная операция отрицания (NOT)
- && - бинарная операция И (AND), истинна, когда оба операнда истинны
- || - бинарная операция ИЛИ (OR), истинна, когда один из операндов равен true

Пример:

```
bool = true && false
```

4.5.5. Операторы сравнения

В таблице 4.2 описаны операторы сравнения, которые вы можете использовать в JavaScript при написании своих сценариев.

Таблица 4.2. Операторы сравнения в JavaScript

Оператор	Описание
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно
!=	Не равно
===	Строго равно

Обратите внимание, что в JS есть два оператора сравнения: равно (==) и строго равно (===). В чем между ними разница? Оператор ==, сравнивая значения разных типов, пробует свести типы, а затем выполнить сравнения. То есть значение **3** будет равно строке "3". Оператор ===, встретив разные типы значений, сразу вернет *false*.

4.5.6. Двоичные операторы

К двоичным операторам относятся следующие операторы:

- ~ - двоичная инверсия (A = ~B;).
- & - двоичное И (A = B & C;).
- | - двоичное ИЛИ (A = B | C;).
- ^ - двоичное исключающее ИЛИ (A = B ^ C;).
- << - сдвиг влево на один или более разрядов с заполнением младших разрядов нулями (A = B << Y;)

- >> - сдвиг вправо на один или разрядов с заполнением старших разрядов содержимым самого старшего разряда (A = B >> Z;).
- >>> - сдвиг вправо без учета знака, старшие разряды будут заполнены нулями (A = B >>> C).

Двоичные операторы выполняют поразрядные операции с двоичным представлением целых чисел.

4.5.7. Слияние строк

Для слияния (конкатенации) строк используется оператор +:

```
const str = "string1 " + "string2"
```

В результате переменная **str** будет содержать значение "string1 string2".

4.5.8. Приоритет выполнения операторов

Сейчас мы поговорим о приоритете выполнения операторов. Пусть у нас есть выражение:

```
A = 2 + 3 * 4 / 5;
```

В какой последовательности будет производиться его вычисление? Еще из школьного курса математики мы знаем, что приоритет операции умножения выше, чем сложения, поэтому сначала **3** будет умножено на **4** (в результате мы получим значение 21).

Затем полученное значение 21 будет разделено на 5 (мы получим значение 4.2), поскольку приоритет операции деления выше, чем сложения.

К полученному значению 4.2 будет добавлено значение **2** и в результате переменной **A** будет присвоено значение 6.2.

Приоритет операций можно изменить с помощью скобок, например:

```
A = (2 + 3) * 4 / 5;
```

В этом случае сначала будет вычислено выражение 2 + 3, а затем полученное значение будет умножено на **4** и разделено на **5**. Приоритет операций умножения и деления одинаковый, поэтому они выполняются слева направо, то есть *в порядке следования*. В результате будет получено значение **4** (5 * 4 и разделить на 5).

Далее перечислены операторы в порядке убывания приоритета:

- !, ~, ++, -- — отрицание, двоичная инверсия, инкремент, декремент;
- *, /, % — умножение, деление, остаток от деления;
- +, - — сложение и вычитание;
- <<, >>, >>> — двоичные сдвиги;
- & — двоичное И;
- ^ — двоичное исключающее ИЛИ;
- | — двоичное ИЛИ;
- =, +=, -=, *=, /=, %= — присваивание.

4.6. Основные конструкции языка JavaScript

К основным конструкциям языка относят условный оператор (if..else), а также операторы циклов. В этом разделе будут рассмотрены эти конструкции.

4.6.1. Условный оператор if

Прежде, чем мы будем рассматривать условный оператор **if**, настоятельно рекомендую вернуться к разделу 3.2.5 и еще раз просмотреть таблицу с операторами сравнения - так вам будет понятнее все происходящее здесь.

Условный оператор **if** имеется в большинстве языков программирования. Он позволяет выполнить определенное действие в зависимости от истинности условия. Общая форма оператора выглядит так:

if (условие)
{операторы, если условие истинно}
else {
операторы, если условие ложно}

Обратите внимание, что вторая часть (else) не обязательна.

Условие - это логическое выражение, построенное на базе операторов сравнения, именно поэтому я просил вас вернуться к разделу 3.2.5, чтобы еще раз просмотреть имеющиеся операторы сравнения. Каждый из операторов сравнения возвращает *true* в случае истинности и *false*, если проверяемый факт ложен.

Пусть у нас есть две переменные:

```
const A = 10;
const B = 5;
```

Оператор A == B вернет *false*, поскольку **A** не равно **B**. Оператор A > B вернет *true*, поскольку **A** больше, чем **B**.

Для инверсии логического значения вы можете использовать оператор !, например:

```
!(A == B)
```

Конечно, можно также использовать оператор !=, но здесь уже поступайте, как вам будет удобнее и понятнее.

В нашем случае переменные **A** и **B** не равны, поэтому оператор == вернет значение *false*, но поскольку указан оператор !, то будет возвращено значение *true*.

Рассмотрим несколько примеров:

```
const A = 10;
const B = 5;

// Будет выведено A > B
```

```
if (A > B) {
  document.write('A > B'); }

// Будет выведено B < A
if (A > B) {
  document.write('A > B'); }
else {
  document.write('B < A'); }

// Будет выведено A = B
if (!(A == B)) {
  document.write(" A != B "); }
else {
  document.write(" A = B "); }
```

Операторы **if** можно вкладывать друг в друга, что продемонстрировано в листинге 3.2. Сценарий в этом листинге пытается разделить 10 на значение одной из переменных - **A** или **B**, предварительно проверяя, не равно ли значение этих переменной 0. Сначала сценарий проверяет, не равно ли 0 значение переменной A. Поскольку A = 0, то выполнение сценария переходит на второй оператор **if**, который проверяет, не равно ли 0 значение переменной B. С переменной B все хорошо, поэтому переменной C будет присвоено значение 10 / 1 (10). В противном случае, если обе переменные равны 0, будет выведено сообщение *Division by zero*.

Листинг 4.5. Вложенность операторов if

```
<html>
<head>
  <title>if</title>
</head>
<body>
<script>

const A = 0;
const B = 1;
let C;

if (A !=0) {
  C = 10 / A;
  document.write('C = ' + C); }
```

```
else if (B != 0) {
   C = 10 / B;
   document.write('C = ' + C); }
else document.write("Деление на 0");

</script>
</body>
</html>
```

4.6.2. Оператор выбора switch

Иногда (конечно не во всех ситуациях) вместо множества вложенных операторов **if** можно использовать оператор **switch**. Оператор switch позволяет сравнить переменное или выражение с множеством значений, что позволяет избавиться от серии операторов **if** и сделать код более компактным.

Общая форма оператора **switch** выглядит так:

```
switch (<Переменная или выражение>) {
        case <Значение 1>:
                        <Оператор 1>;
                        break;
        case <Значение 2>:
                        <Оператор 2>;
                        break;
        ...
        [default:
                <Оператор>;]
}
```

Работает оператор **switch** следующим образом:

- Сначала вычисляется значение переменной или выражения;
- Затем полученное значение сравнивается с одним из значений, указанных в блоках **case**.
- Представим, что у нас 10 блоков case, и значение совпало с 5-ым блоком **case**. Тогда, если в 5-ым блоке case не указан оператор **break**, то будут выполнены действия, связанные с блоками 5-10, а также операторы из блока default. Если же указан оператор break, тогда будет выполнено только то действие, которое указано в 5-ом блоке case. Для большей однознач-

ности (если не нужно иного) я всегда рекомендую использовать оператор break для преждевременного выхода из оператора switch.

- Если вычисленное значение не совпало ни с одним из значений, указанных в блоках **case**, тогда будет выполнены операторы из блока **default**, если таковой указан. Блок default является необязательным.

Представим, что у нас есть переменная **command**, в зависимости от значения которой нужно выполнить определенные действия, например:

```
if (command == 1) alert('Выбрано действие: 1');
if (command == 2) alert('Выбрано действие: 2');
if (command == 3) alert('Выбрано действие: 3');
if (command == 4) alert('Выбрано действие: 4');
```

Код выглядит громоздко и логически не воспринимается, как один блок, а как четыре разных блока (если бы мы по этому коду построили блок-схему, то у нас бы и получилось четыре разных блока).

Весь этот громоздкий код мы можем заменить на более компактный. Пусть он занимает больше строк, зато выглядит не таким перегруженным и воспринимается как единое целое:

```
switch (command) {
      case 1: alert('Выбрано действие: 1'); break;
      case 2: alert('Выбрано действие: 2'); break;
      case 3: alert('Выбрано действие: 3'); break;
      case 4: alert('Выбрано действие: 4'); break;
      default: alert('Неизвестное действие!');
}
```

Как видите, получившийся код воспринимается не таким перегруженным, хотя занимает больше строк. К тому же оператор **switch** позволяет задавать действие по умолчанию. В конечном итоге, с его помощью можно понятнее и прозрачнее реализовывать сложные разветвления, которые кажутся запутанными, если их реализовать с помощью **if**. Однако еще раз отмечу, что switch - далеко не панацея во всех ситуациях выбора.

В листинге 4.6 приведен пример использования оператор **switch**. Сначала мы отображаем диалог ввода действия, затем анализируем, какое действие выбрал пользователь. Обратите внимание, что прежде, чем передать полу-

ченное действие оператору switch, мы сводим его к типу *number* с помощью функции parseInt().

Листинг 4.6. Пример использования оператора switch

```
<html>
<head>
  <title>Confirm</title>
</head>
<body>
  <script>
  const command = window.prompt("Введите действие", "");

  if (command == null) {
    document.write('Нажата Отмена'); }
   else
     switch (parseInt(command)) {
        case 1: alert('Выбрано действие 1'); break;
        case 2: alert('Выбрано действие 2'); break;
        case 3: alert('Выбрано действие 3'); break;
        case 4: alert('Выбрано действие 4'); break;
        default: alert('Неизвестное действие');
     }

</script>
</body>
</html>
```

4.6.3. Циклы

Если проанализировать все программы, то на втором месте после условного оператора будут операторы цикла. Используя цикл, вы можете повторить операторы, находящиеся в теле цикла. Количество повторов зависит от типа цикла - можно даже создать бесконечный цикл. В JavaScript есть три типа цикла:

- Цикл со счетчиком (for)
- Цикл с предусловием (while)
- Цикл с постусловием (do..while)

Цикл for

Начнем с цикла со счетчиком - **for**. Этот цикл используется для выполнения тела цикла четко определенного количества раз. Цикл **while**, например, удобно использовать для ожидания какого-то события (мы не знаем, сколько раз будет выполнено тело цикла, пока условие станет истинным), а цикл **for** используется тогда, когда вы точно знаете, сколько раз нужно повторить цикл. Синтаксис цикла **for**:

```
for (команды_инициализации; условие; команды_после_итерации)
        {
        <Операторы - тело цикла>
        }
```

Оператор **for** начинает свою работу с выполнения команд инициализации. Данные команды выполняются всего лишь один раз. После этого проверяется условие: если оно истинно, выполняется тело цикла. После того, как будет выполнен последний оператор тела, выполняются команды "после итерации". Затем снова проверяется условие, в случае, если оно истинно, выполняется тело цикла и поститерационные команды и т.д.

Выведем строку 0123456789:

```
for (i=0; i<10; i++) document.write(i);
```

Чтобы вывести строку 12345678910 нужно установить начальное значение счетчика в 1 и изменить условие цикла:

```
for (i=1; i<=10; i++) document.write(i);
```

Цикл **for** будет очень полезным при обработке массивов, которые мы рассмотрим в следующей главе.

Цикл while

В некоторой мере цикл **for** очень похож на цикл с предусловием (while), так как сначала проверяется условие, а потом уже выполняется тело цикла. Рассмотрим цикл с предусловием:

```
while ( логическое выражение ) {
 операторы; }
```

Сначала цикл вычисляет значение логического выражения. Если оно истинно, происходит итерация цикла (выполняется тело цикла), иначе происходит выход из цикла и выполнение следующего за циклом оператора.

Далее приведен пример, выводящий числа от 1 до 10:

```
let i = 1;

while (i < 11) {
      document.write(i + "<br>");
      i++;
}
```

Соблюдайте осторожность при использовании цикла **while**. Если вы не предусмотрите условие выхода, тогда получите бесконечный цикл. В нашем случае условием выхода является не только, что i < 11, но и сам инкремент **i**, то есть оператор, способный повлиять на условие, указанное в заголовке цикла. Если бы мы забыли указать оператор i++, то получили бы бесконечный цикл. Поскольку переменная i = 1, что меньше 11 и у нас нет другого оператора, который в теле цикла изменял бы эту переменную, то тело цикла будет выполняться бесконечно.

Цикл do..while

В PHP есть еще одна форма цикла - do while. В отличие от цикла while здесь сначала выполняются операторы (тело цикла), а затем уже проверяется условие. Если условие истинно, то начинается следующая итерация. Получается, что тело цикла будет выполнено как минимум один раз. Синтаксис цикла:

```
do
 {
 // тело цикла
 }
while (условие);
```

Пример:

```
i = 1;
do {
  document.write(i);
  i++; }
while (i < 10);
```

Операторы break и continue

В теле цикла вы можете использовать операторы **break** и **continue**. Первый прерывает выполнение цикла, а второй – прерывает выполнение текущей итерации и переходит к следующей.

Представим, что нам нужно вывести только нечетные числа в диапазоне от 1 до 20. Пример цикла может быть таким:

```
for (i=1; i<21; i++) {
 if (i % 2 == 0) continue;
 else document.write(i + " ");
}
```

В теле цикла мы проверяем, если остаток от деления i на 2 равен 0, значит, число четное и нужно перейти на следующую итерацию цикла. В противном случае нам нужно вывести наше число.

А вот пример использования оператора **break**. Не смотря на то, что цикл якобы должен выполниться 10 раз, он будет прерван, когда i будет равно 5:

```
for (i=1; i<11; i++) {
 if (i == 5) break;
 document.write(i + " ");
}
```

В результате будет выведена строка:

```
1 2 3 4
```

Вложенность циклов

В теле цикла может быть другой цикл. Вложенность циклов формально не ограничивается, однако нужно быть предельно острожным, чтобы не допустить зацикливания. Пример вложенного цикла:

```
let j = 1;
while (j < 15) {
 for (k=0; k<j; k++) document.write('*');
 document.write('<br>');
 j++;
}
```

Одно из частых применений циклов - обработка массивов. О том, что такое массивы и как с ними работать в JavaScript, мы поговорим в следующей главе.

4.7. Введение в массивы

Массив - это упорядоченный набор данных. У каждого элемента массива есть свой индекс (его также называют ключом), который используется для однозначной идентификации элемента внутри массива.

Если вы программировали на других языках программирования, то знаете, что все переменные, в том числе и массивы, должны быть объявлены в строго определенном месте программы. В JavaScript переменную, следовательно, и массив, можно объявить в любом месте программы (сценария), но до первого использования массива. А это удобно, если нужно создать массив с неизвестным числом параметров. Например, вам нужно прочитать в массив текстовый файл так, чтобы каждая строка файла представляла собой один элемент массива. В другом языке программирования нужно было объявить очень большой массив, скажем, на 10 000 элементов - ведь вы же не знаете, сколько строк будет в файле, но нужно написать более или менее универсальную программу, чтобы она умела обрабатывать большие файлы. А что делать, если в файле всего 3 строки или 10 001 строка? В первом случае вы выделили памяти во много раз больше, чем реально нужно для обработки файла. Ваша программа будет попросту поедать память, и вы ничего не сможете сделать. Во втором случае ваша программа не справится с обработ-

кой всего файла, поскольку в файле больше строк, чем может поместиться в массив.

4.7.1. Инициализация массива

Массив инициализируется, как и любая другая переменная, путем присваивания значения. Поскольку массив может содержать несколько значений, при его инициализации значения указываются в квадратных скобках и разделяются запятыми:

```
let M;
M = [5, 3, 4, 1, 2];

let Months;
Months = ["", "Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec"];
```

Нумерация элементов массива начинается с 0, а не с 1. Получить доступ к определенному элементу массива можно так:

```
num0 = M[0];   // 5
j = M[3];    // 1
```

4.7.2. Изменение и добавление элементов массива

При желании вы всегда можете изменить значение существующего элемента массива или добавить новый элемент, например:

```
M[0] = 7;
M[5] = 8;
```

Первый оператор присваивает значение 7 элементу массива M с индексом 0. Второй оператор создает новый элемент массива со значением 8, индекс элемента - 5. Теперь у нас есть вот такой массив:

```
[7, 3, 4, 1, 2, 8]
```

Посмотрим, что сделает следующий оператор:

```
M[7] = 11;
```

Этот оператор создаст два элемента массива: элемент с индексом 6 и значением *undefined* и элемент с индексом 7 и значением 11. Массив будет следующим:

```
[7, 3, 4, 1, 2, 8, undefined, 11]
```

4.7.3. Многомерные массивы

В JS вы можете создавать многомерные массивы путем присваивания любому элементу массива нового массива, например:

```
M[0] = [3, 2, 1];
```

Обратиться к элементу многомерного массива можно так:

```
j = M[0][1];
```

4.7.4. Пример обработки массива

Подробно о средствах работы с массивами мы поговорим в главе 14, когда мы будем рассматривать встроенные классы JS. Сейчас же мы рассмотрим несколько примеров для работы с массивами. У любого объекта массива есть свойство *length*, содержащее длину массива. Это свойство мы можем использовать при обработке массива.

Сценарий из листинга 4.7 вычисляет минимальный элемент массива. Сначала минимальным считается первый (с индексом 0) элемент массива. Далее в цикле **for** мы сравниваем каждый следующий элемент массива. Если сравниваемый элемент меньше нашего минимума, он становится новым минимумом.

Листинг 4.7. Вычисление минимума массива

```
<html>
<head>
  <title>Минимум в массиве</title>
</head>
<body>
  <script>
  let M = [7, 66, 55, 4, 88, 1, 8, 99, 3];

  Min = M[0];    // Минимум
  Min_ind = 0;   // Индекс Min

  document.write('Массив: <br>');

  for (i=1; i<M.length; i++) {
    if (M[i] < Min) {
      Min = M[i];
      Min_ind = i;
    } // if
    document.write(M[i] + " ");
  } //for
  document.write('<br>Минимум: ' + Min);
  document.write('<br>Индекс: ' + Min_ind);

</script>
</body>
</html>
```

Сначала мы считаем минимальным элемент с индексом 0. Далее выводим этот минимальный элемент, поскольку обработка массива начинается с индекса 1 (сравнивать 0-ой элемент с 0-ым элементом нет смысла). В цикле мы проверяем, не является ли текущий элемент минимальным и, если это так, устанавливаем новый минимум. Для идентификации минимума используются две переменных - Min (минимум) и Min_ind (индекс минимума). Также в цикле мы выводим обрабатываемый элемент.

Теперь мы решим другую подобную задачу - найдем максимальный элемент. Чтобы было интереснее, давайте сделаем двумерный массив и найдем в нем максимальный элемент. А то бы наш новый сценарий отличался бы только знаком больше от предыдущего. Согласитесь, совсем не интересно. В новом же сценарии нам нужно обойти два измерения. Для обхода перво-

го измерения мы будем использовать счетчик **i**, для обхода второго - j. Как только найдем максимальный элемент, в массив **Ind** мы запишем его "координаты" - значения счетчиков i (элемент 0) и j (элемент 1).

Листинг 4.8. Поиск максимума в двухмерном массиве

```
<html>
<head>
  <title>Максимальный элемент</title>
</head>
<body>
  <script>
  let M = [0, 0, 0];

  M[0] = [3, 2, 1];
  M[1] = [7, 8, 9];
  M[2] = [5, 6, 7];

  Max = M[0][0];    // Максимум
  Ind = [0, 0];     // Его индекс

  for (i=0; i<M.length; i++) {

    for (j=0; j<M[i].length; j++) {

      if (M[i][j] > Max) {
      Max = M[i][j];
      Ind[0] = i;
      Ind[1] = j;

      }
    }
  }

  document.write('Max ' + Max);
  document.write('<br>Ind [' + Ind[0] + '][' + Ind[1] +
']');

</script>
</body>
</html>
```

Обратите внимание, как мы обращаемся к свойству *length*. Поскольку каждый из элементов исходного массива также является массивом, то мы можем обратиться к *length* так: M[i].length. Конечно, мы и так знаем размерность массива, и можно было бы использовать значения длины 3 для каждого из счетчиков, но так корректнее.

4.8. Функции JavaScript. Основные понятия

Функция - это фрагмент JavaScript-кода, который можно вызвать из любого места основного сценария.

По сути, функция - это подпрограмма. Функция описывается с помощью ключевого слова **function**, синтаксис описания функция следующий:

```
function <Имя функции> ([<Параметры>]) {
        <Тело функции>
        [return <Значение>]
}
```

Имя функции должно быть уникальным. Для имен функции действует те же правила, что и для имени переменной. После имени функции в круглых скобках указываются параметры функции. Если функции не передаются параметры, то указываются только круглые скобки. Если параметров несколько, то они разделяются запятыми.

В фигурных скобках располагаются выражения JavaScript. Как правило, эти выражения производят какие-то манипуляции над переданными параметрами.

Функция по определению должна возвращать результат. Результат возвращается с помощью ключевого слова **return**. Конечно, иногда нужно создать просто подпрограмму, которая не возвращает никакого результата (например, форматирует сообщение в диалоговом окне и выводит это самое диалоговое окно), в этом случае ключевое слово return не обязательно и можно обойтись без него.

Рассмотрим несколько примеров функций:

```
// Функция просто выводит диалоговое окно с текстом 'Access denied'
// Использование этой функции просто короче, чем вызов windows.alert
// К тому же, когда понадобится изменить текст сообщения, тогда
// текст можно будет изменить в одном месте, а не по всему сценарию
// Функция ничего не возвращает

function denied() {
window.alert('Access Denied!');
}

// Функция возвращает сумму двух чисел. Никакой проверки, являются
// ли аргументы числами, не производится
function Sum(x, y) {
const result = x + y;
return result;
}
```

Использовать эти функции можно так:

```
denied();           // будет выведено наше сообщение
let x = Sum(2, 2); // в переменную x будет записан
                    // результат функции Sum
```

После инструкции **return** происходит выход из функции. Все операторы после оператора return не будут выполнены, например:

```
function Sum(x, y) {
      const result = x + y;
      return result;
      window.alert('Сумма'); // никогда не будет выполнен

}
```

В конструкции **return** можно указать сразу выражение, перепишем нашу функцию Sum:

```
function Sum(x, y) {
      return (x + y);
}
```

Функции можно передавать не только значения, но и значения переменных, например:

```
const a = 10;
const b = 12;

const ab = Sum(a, b);
```

Ссылку на функцию вы можете сохранить в любой переменной, например:

```
const d = denied;  // Имя функции указываем без скобок
d();       // Вызываем функцию denied() по ссылке
```

В JS допускаются также анонимные функции, то есть функции без названия:

```
const x = function() {          // ссылка на анонимную
функцию
  window.alert('Тест'); // присваивается переменной x
}
x();            // вызываем функцию через переменную x
```

Ссылку на вложенную функцию можно возвратить в качестве значения конструкции **return**, для этого используются круглые скобки два раза. Пример:

```
const x = function() {          // Ссылка на анонимную функцию
      return function() {       //   Возвращаем   ссылку   на
                                // вложенную функцию
            window.alert("Тест");
      };
   };
x()(); // Вызываем вложенную функцию
```

4.8.1. Расположение функций внутри сценария

Мы уже рассмотрели достаточно примеров функций, осталось только понять, где в HTML-документе должны находиться функции. Теоретически,

функция может находиться в любом месте сценария, но до первого момента ее использования. Чтобы не запутывать прежде всего самих себя, программисты обычно помещают описание функций в секцию HEAD (заголовок) HTML-документа. Если же функций достаточно много или код функции слишком объемный, можно вынести код в отдельный .js файл. Сейчас мы рассмотрим оба варианта.

В листинге 4.9 я описал функцию в секции HEAD, а вызов функции производится из сценария секции BODY.

Листинг 4.9. Функция помещена в HEAD

```
<html>
<head>
  <title>Функции</title>
  <script>
    function denied() {
      window.alert('Access Denied!');
    }

  </script>
</head>
<body>
  <script>

    denied();     // вызываем функцию

</script>
</body>
</html>
```

В листинге 4.10 мы подключаем JS-файл functions.js (имя файла может быть любым). Код файла functions.js приведен в листинге 4.11.

Листинг 4.10. Вызов функции из внешнего JS-файла

```
<html>
<head>
  <title>Функции</title>
```

```
  <script type="text/javascript" src="functions.js">
  </script>
</head>
<body>
  <script>

  denied();

</script>
</body>
</html>
```

Листинг 4.11. Внешний JS-файл (functions.js)

```
function denied() {
   window.alert('Access Denied!');
}
```

Не нужно создавать отдельный JS-файл для каждой функции. Вы можете создать один-единственный файл, в который вы поместите все функции, которые необходимы вашему основному сценарию.

4.8.2. Область видимости переменной: глобальные и локальные переменные

Глобальными являются все переменные, объявленные за пределами функции. Они доступны в любой части программы (сценария), в том числе и в функции.

Локальными являются переменные, объявленные в самой функции. Такие переменные доступны только функции, в которой они объявлены и недоступны в других функциях или в основной программе.

Если имя локальной переменной совпадает с именем глобальной переменной, то будет использоваться локальная переменная, а значение глобальной переменной останется без изменения.

Рассмотрим листинг 4.12. В секции HEAD мы объявляем две переменные - A и B. Они будут глобальными переменными, доступными, как в функции F1(), так и в коде из секции BODY, то есть везде по сценарию. В теле

функции мы объявляем две локальные переменные - X и B. Затем функция выводит значение переменных A, B, X. Функция вывела значения 10, 5 и 10. Как видите, используется локальная переменная B вместо глобальной переменной B, если имена переменных совпадают. Основная программа выводит значения переменных **A** и **B**, будут выведены значения 10 и 20. Выводить значение переменной X в основной программе нет смысла, так как вы получите сообщение об ошибке **Uncaught ReferenceError: X is not defined**.

Листинг 4.12. Глобальные и локальные переменные

```
<html>
<head>
  <title>Глобальные и локальные переменные</title>
  <script>
  // глобальные переменные
  const A = 10;
  const B = 20;

  function F1() {

  // локальные переменные
  const X = 10;
  const B = 5;

  document.write("<br>A = " + A);
  document.write("<br>B = " + B);
  document.write("<br>X = " + X);
  }
  </script>
</head>
<body>
  <script>

  F1();

  document.write("<HR>");
  document.write("A = " + A);
  document.write("<br>B = " + B);

</script>
</body>
</html>
```

4.9. Основные концепции объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) — это особый подход к написанию программ.

ООП базируется на трех основных принципах - **инкапсуляция**, **полиморфизм**, **наследование**.

С помощью **инкапсуляции** вы можете объединить воедино данные и обрабатывающий их код. Инкапсуляция защищает и код, и данные от вмешательства извне. Базовым понятием в ООП является класс. Грубо говоря, класс - это своеобразный тип переменной. Экземпляр класса (переменная типа класс) называется объектом. В свою очередь, объект - это совокупность данных (свойств) и функций (методов) для их обработки. Данные и методы обработки называются членами класса.

Получается, что объект - это результат инкапсуляции, поскольку он включает в себя и данные, и код их обработки. Чуть дальше вы поймете, как это работает, пока представьте, что объект - это эдакий рюкзак, собранный по принципу "все свое ношу с собой".

Члены класса могут быть открытыми или закрытыми. Открытые члены класса доступны для других частей программы, которые не являются частью объекта. Закрытые члены доступны только методам самого объекта.

Теперь поговорим о **полиморфизме**. Если вы программировали на языке C (на обычном C, не C++), то наверняка знакомы с функциями abs(), fabs(), labs(). Все они вычисляют абсолютное значение числа, но каждая из функций используется для своего типа данных. Если бы C поддерживал полиморфизм, то можно было бы создать одну функцию abs(), но объявить ее трижды - для каждого типа данных, а компилятор бы уже сам выбирал нужный вариант функции, в зависимости от переданного ей типа данных. Данная практика называется перезагрузкой функций. Перезагрузка функций существенно облегчает труд программиста - вам нужно помнить в несколько раз меньше названий функций для написания программы.

Полиморфизм позволяет нам манипулировать с объектами путем создания стандартного интерфейса для схожих действий.

Осталось поговорить о **наследовании**. С помощью наследования один объект может приобретать свойства другого объекта. Заметьте, наследование - это не копирование объекта. При копировании создается точная копия объекта, а при наследовании эта копия дополняется уникальными свойствами (новыми членами). Наследование можно сравнить с рождением ребенка, когда новый человек наследует "свойства" своих родителей, но в то же время не является точной копией одного из родителей.

Все выше сказанное было истинно для любого полноценного объектно-ориентированного языка программирования. В JavaScript поддержка ООП довольно ограничена. Например, нет ни закрытых (приватных), ни открытых свойств и методов. Все свойства и методы являются открытыми.

4.9.1. Создание пользовательских классов и объектов

Мы уже знакомы немного с классами и объектами. Создать новый объект можно с помощью встроенного класса **Object**, например:

```
let Human = new Object();
// свойства объекта
Human.firstname = "Родион";
Human.lastname = "Doe";
// формируем метод объекта
Human.getFullName = function() {
   const fname = this.firstname + this.lastname;
   return fname;
}

// просто выводим значения:
window.alert(Human.firstname);
window.alert(Human.lastname);
window.alert(Human.getFullName);
```

После того, как объект создан, в переменной **Human** сохраняется на него ссылка. В качестве значения свойства объекта можно использовать любой тип данных - число, массив, строку или другой объект. Если в качестве зна-

чения указана ссылка на функцию, то такое свойство становится методом объекта, внутри которого доступен указатель на текущий объект (this).

Если вы привыкли к другим языкам программирования и такое создание объекта вам не привычно, можете использовать фигурные скобки для определения свойств и методов объекта:

```
let Human = {
   firstname: "Родион";
   lastname: "Doe";
   getFullName: function() {
        const fname = this.firstname + this.lastname;
        return fname;
   }
}
```

В этом случае значение свойств/методов указывается через двоеточие. Если в фигурных скобках нет никаких выражений, тогда создается пустой объект:

```
let empty_obj = {};
```

Есть один очень важный момент, который вы должны осознать. Пусть нам нужно создать два одинаковых объекта, которые должны использоваться раздельно. Вот что первым приходит в голову:

```
let o1 = o2 = {};
```

Однако так нельзя делать! Поскольку создается только один объект, а ссылка на него сохраняется в двух переменных. Поэтому все изменения в переменной **o1** будут отображаться и на переменной **o2** и наоборот:

```
o1.firstname = "Родион";
document.write(o2.firstname);  // будет выведено Родион
```

Что же делать? Нужно два разных оператора инициализации переменной-объекта, например:

```
let o1 = {};
let o2 = {};
```

Если после ключевого слова **new** указывается функция, то она становится конструктором объекта. Такой функции можно передать начальные данные для инициализации объекта (можно ничего не передавать, все зависит от реализации). Функции-конструкторы удобно использовать, если нужно инициализировать несколько подобных объектов. Рассмотрим функцию конструктор:

```
function Human(firstname, lastname) {
   this.firstname = firstname;
   this.lastname = lastname;
   this.getFullName = function() {
         const fname = this.firstname + this.lastname;
      return fname;
    }
}

let Rodion = new Human("Родион", "Doe");
let Ivan = new Human("Ivan", "Ivanov");
```

Что такое класс? Класс - это тип объекта, включающий в себя переменные и функции для управления этими переменными. Да, переменные - это свойства, а функции - это методы. Создать экземпляр класса, то есть объект, можно так:

<экземпляр> = new <имя объекта> ([параметры]);

Теперь посмотрите на то, как мы создали объекты Rodion и Ivan. Мы указали служебное слово **new**, затем имя нашей функции и передали ей параметры. Другими словами, создав функцию-конструктор, мы создали класс **Human**. Вот так вот. Как я уже и говорил, поддержка ООП в JavaScript несколько изощренная. Если вы никогда не программировали на объектно-ориентированном языке программирования, то ничего необычного не заметите. Но если же вы программировали хотя бы на том же PHP, то создание класса посредством создания функции-конструктора - это дикость. Но есть то, что есть - и вам с этим придется смириться и использовать именно в таком контексте, как это предлагает JavaScript.

Зато в JavaScript есть удобный цикл for..in, позволяющий пройтись по всем свойствам объекта. Это позволяет вывести свойства объекта, которые ранее

были неизвестны. А так как у объектов нет закрытых свойств, то будут выведены абсолютно все свойства объекта. Пример:

```
for (let P in Rodion) {
      document.write(P + " = " + Rodion[P]);
}
```

Обратите внимание: мы обращаемся к объекту, как к массиву (в PHP такие массивы называются ассоциативными, где в качестве индекса может использоваться не только число, но и строка).

Оператор **in** позволяет проверить существование свойства в объекте, например:

```
if ("firstname" in Rodion) window.alert(Rodion.firstname);
```

Проверить наличие метода можно, указав его имя без скобок:

```
if (Rodion.getFullName) window.alert('getFullName exists');
```

Так нельзя проверять наличие свойства, поскольку значение **0** будет интерпретироваться как *false*, в итоге такой проверки вы получите, что свойство не существует, но на самом деле оно существует, но просто равно **0**.

С помощью оператора **instanceof** можно проверить принадлежность экземпляра классу, например:

```
if ((typeof Rodion == "object") && (Rodion instanceof
Human))
   window.alert('Родион - экземпляр Человека');
```

Удалить свойство можно так:

```
delete Ivan.lastname;
```

4.10. Практика JS

4.10.1. Работа с объектной моделью документа HTML

Прежде чем приступить к разбору оживления нашего сайта средствами языка JavaScript необходимо разобраться, как взаимодействовать со страницей и находящимися на ней элементами. Как указать, что именно в данную область страницы должен быть помещен слайдер или какой блок будет использоваться как фиксированное меню? Данные вопросы решаются благодаря использованию DOM-дерева нашей HTML страницы. DOM – это объектная модель документа (Document Object Model), согласно которой, каждый HTML-тег является объектом. Вложенные теги являются "детьми" родительского элемента, при этом, текст, находящийся внутри тегов, тоже является объектом. С помощью DOM мы можем представить нашу верстку в виде древовидной структуры (рис. 4.1):

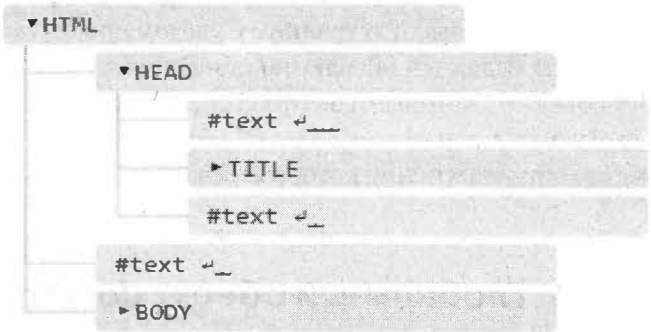


Рис. 4.1. Пример структуры DOM-дерева

Более подробно узнать об особенностях DOM-дерева и том, как современные браузеры взаимодействуют с ним, можно по ссылке на один из основных электронных учебников по JavaScript: https://learn.javascript.ru/dom-nodes.

Нам же для работы нужно в первую очередь изучить основные глобальные объекты DOM-дерева, с помощью которых мы и будем взаимодействовать с тегами на нашей странице:

1. window
2. document

4.10.2. Глобальный объект window

Объект window описывает все содержимое окна браузера, когда вы открываете страницу сайта. Внутри данного объекта содержатся все встроенные в браузер функции и свойства. К таким функциям относятся, например, alert() и prompt(), упомянутые ранее в теории оп JavaScript. Данные функции можно описать, как методы объекта window: window.alert(), window. prompt().

К данному объекту мы можем обращаться при разработке, если нам нужно узнать параметры окна браузера для задания взаимодействия пользователя с нашей страницей. Например, в свойства данного объекта записывается информация о том, сколько пикселей от верха страницы пролистал пользователь или какова полная высота страницы с учетом прокрутки. Если стили какого-либо объекта задаются абсолютно, а не относительно, то итоговые размеры этого объекта, с которыми он отобразится на странице, также можно найти в свойствах данного глобального объекта. Для этого можно использовать метод объекта window, который называется getComputedStyle().

4.10.3. Глобальный объект document

Это основной объект, с которым мы будем работать при реализации скриптов для нашего сайта. Объект документ указывает на DOM-дерево страницы и является вложенным в глобальный объект **window**. Наибольший интерес для нас представляют имеющиеся у данного объекта методы, которые и позволяют нам работать с элементами DOM-дерева. Рассмотрим эти методы далее.

4.11. Основные методы для работы с элементами DOM-дерева

К основным методам, которыми мы будем пользоваться в нашей работе, относятся три следующих ключевых слова:

1. querySelector() или querySelectorAll()
2. addEventListener()
3. classList

Рассмотрим каждое из этих ключевых слов подробнее.

4.11.1. Ключевое слово querySelector()

Данный метод относится к глобальному объекту **document**. Основное его назначение – поиск элемента DOM-дерева по указанному идентификатору. Таким образом, метод возвращает из DOM-дерева соответствующий CSS селектору элемент.

Например, вызов метода

```
document.querySelector(".user-block")
```

вернет нам элемент из DOM-дерева с классом class="user-block".

Чтобы обратиться к элементу, можно использовать следующие селекторы:

1. класс – ".user-block"
2. атрибут – "[user-block]"
3. значение атрибута – "[user-block] = 1"
4. **id** элемента – "#user-block"
5. смешанные селекторы

Особенность данного метода заключается в том, что он "проходит" по структуре DOM-дерева и возвращает первый найденный элемент, удовлетворяющий нашему условию. Т.е. если у вас на странице содержится 3 элемента с

классом "user-block", то данный метод вернет именно первый элемент. Как получить все три объекта, мы рассмотрим далее.

4.11.2 Ключевое слово querySelectorAll()

Как понятно из названия данного метода, он делает все то же самое, что и предыдущий, но возвращает не первый найденный элемент, а все элементы, удовлетворяющие заданному условию, сгруппированные в псевдомассив (на данном этапе вам нужно знать, что по структуре псевдомассив похож на обычный массив JS, но т.к. в нем содержатся элементы DOM-дерева, к нему не применимы многие свойства, работающие с массивами).

Таким образом, если у нас на странице есть три элемента с классом "user-block", то вызов метода:

```
querySelectorAll(".user-block"),
```

вернет нам псевдомассив из трех элементов [эл-нт 1, эл-нт 2, эл-нт 3].

Данный метод можно использовать, например, когда нужно обратиться ко всем формам на странице, чтобы настроить их отправку, а не создавать скрипты для каждой формы по-отдельности.

4.11.3. Ключевое слово addEventListener

Данный метод работает на указанном элементе и описывает события, которые должны происходить при изменении данного элемента или взаимодействии с ним. Самый простой пример – реализации события при клике на ссылку.

Метод имеет следующий синтаксис использования:

```
link.addEventListener("click", function() {…});
```

Вызов метода состоит из 4 основных компонентов:

1. Элемент, у которого будет "отлавливаться" событие. В нашем примере – это элемент **link**
2. Метода регистрации ("отлова") требуемого события – addEventListener()
3. События, которое будет отслеживаться. В нашем примере, это события клика – "click"
4. Функция, которая будет выполняться каждый раз при наступлении события – function() {...}

4.11.4. Ключевое слово classList

Данное ключевое слово позволяет нам манипулировать классами элемента в DOM-дереве. С помощью имеющихся у данного ключевого слова методов можно добавлять или удалять у элемента требуемые классы. Это позволяет стилизовать данные элементы, подключая/ отключая определенные группы свойств из таблицы стилей.

Использование данного ключевого слова имеет следующий синтаксис:

```
element.classList.add("selector")
```

Здесь также есть 4 ключевых компонента:

1. Элемент, которому добавляет новый класс (либо удаляется). В нашем примере – **element**
2. Свойство, имеющее методы для работы с классами – classList
3. Метод манипулирования классами. В нашем примере – метод добавления класса **add**
4. Название класса, добавляемого элементу – "selector".

Метод имеет особенность при использовании: при указании селектора класса, в отличии от метода querySelector, перед названием класса точка не ставится ("user-block", а не ".user-block").

У данного ключевого слова существует 4 метода манипуляции классами:

1. **add** - добавляет новый класс
2. **remove** – удаляет заданный класс
3. **toggle** – переключает класс (если у элемента есть указанный класс, то он удаляется; если класса нет, то он добавляется)
4. **contain** – проверяет наличие класса (возвращает *true*, если класс есть у элемента, и *false*, если класс у элемента отсутствует)

Это были основные методы для работы с элементами страницы нашего сайта, далее перейдем к примерам реализации скриптов.

4.12. Примеры скриптов

Написание скриптов происходит в отдельном файле main.js, расположенном в каталоге nit-center/src/js. Помимо создания самого файла скриптов важно не забыть подключить его ко всем страницам нашего сайта. Для этого используется специальный тег SCRIPT. Для оптимизации загрузки и отрисовки страницы данный тег помещают в самый конец тега BODY. Код подключенного скрипта для нашего сайта представлен в листинге 4.1:

Листинг 4.13. Подключение скрипта

```
<script src="./js/main.js"></script>
```

Перед написанием основных скриптов необходимо подготовить также наш основной файл main.js. Для этого в него добавляется следующий код (листинг 4.14).

Листинг 4.14. Подготовка файла main.js

```
"use strict";

window.addEventListener("DOMContentLoaded", () => {
});
```

Первая строчка кода "use strict" используется чтобы сообщить браузеру, что последующий скрипт работает в строгом режиме. О том, что такое строгий и нестрогий режимы работы можно прочитать по ссылке: **https://learn.javascript.ru/strict-mode**.

Последующие строчки с вызовом метода addEventListener объекта window используются для того, чтобы наш скрипт начал работать только после загрузки всего HTML документа. Это позволяет избежать ситуации, когда скрипт уже загрузился и начал работать, а DOM-дерево сайта еще не успело обрисоваться. В этом случае, если скрипт обратиться к элементу, которого еще нет в дереве, то он просто покажет ошибку и прекратит дальнейшее исполнение. Помещая весь основной код нашего скрипта в указанную функцию, мы гарантируем, что подобной ситуации не произойдёт.

В качестве примера мы рассмотрим следующие функции (они реализованы в прилагаемом к книге примере, который приведен в конце книги):

- Функцию, реализующее прилипание главного меню при прокрутке страницы
- Функцию, подключающую слайдер Swiper к главной странице
- Функцию отправки формы обратной связи с главной страницы

4.12.1. Скрипт фиксированного меню

Данный скрипт реализует "прилипание" полоски с элементами главного меню при прокрутке пользователем страницы на определенное расстояние (количество пикселей). Очевидно, что мы будем манипулировать тегом HEADER на наших страницах. Однако, т.к. нам нужно, чтобы при прокрутке показывался только перечень пунктов меню, нужно будет скрывать неиспользуемые теги в блоке HEADER. Сделать это можно несколькими способами: через скрипт находить требуемые блоки и скрывать их из области видимости, манипулируя их значениями свойства display; либо, как это сделано в нашем случае, прописать заранее для этих блоков свойства отображения, в зависимости от наличия у родительского блока HEADER специального класса. В нашем случае – это класс "sticky". Меняя наличие данного класса у родительского блока HEADER, мы таким образом манипулируем отображением внутренних блоков. Подробно разобрать код данной функции можно, обратившись к ее листингу:

Листинг 4.15. Функция фиксированного меню

```
const stickyMenu = () => {
    const menu = document.querySelector(".main-header"),
            menuHeight = window.getComputedStyle(menu).height,
            parentElement = menu.parentElement;

    let menuReplacer = document.createElement("div"),
        replaced;
    menuReplacer.style.height = menuHeight;

    window.addEventListener("scroll", () => {
        let scrollHeight = parseInt(menuHeight) / 2;

        if  (window.pageYOffset >= scrollHeight) {
            menu.classList.add("sticky");
            if (!replaced) {
                parentElement.prepend(menuReplacer);
                replaced = true;
            }
        } else {
            menu.classList.remove("sticky");
            if (replaced) {
                parentElement.removeChild(menuReplacer);
                replaced = false;
            }

        }
    });
};

try {
    stickyMenu();
} catch(err) {

}
```

4.12.2. Скрипт подключения слайдера

Для сайта (рассматриваемого в книге в качестве примера (см. последние главы) и код которого прилагается к книге) мы выбрали не разработку слайдера с нуля, а подключение готовой библиотеки. Библиотеки – это сторонние JS скрипты, которые вы можете интегрировать в ваш JavaScript код, и пользоваться дополнительными возможностями, не тратя времени

на длительное написание того, что уже во многих вариациях присутствует в свободном доступе в интернете. Действительно, в сети можно найти огромное количество всевозможных готовых слайдеров, решающие самые разные задачи: вертикальный слайдер, бесконечный слайдер, слайдер с видео, адаптивные слайдеры и т.д. Для нашего проекта мы выбрали популярный слайдер Swiper. Подробную инструкцию по его использованию можно найти на официальной странице данного слайдера.

Подключение сторонних библиотек и фрагментов JavaScript скриптов возможно несколькими способами. В нашем сайте используется подход подключение CDN библиотек. CDN – это технология, позволяющая оптимизировать загрузку требуемого содержимого для работы сайта. При такой реализации мы не включаем код самого слайдера в сам скрипт, а используем тег SCRIPT, чтобы подключить код слайдера к самой HTML странице. Также в нашем скрипте мы только прописываем конфигурацию (настройки) для данного слайдера, которые можно найти на официальном сайте Swiper.

Алгоритм подключение слайдера следующий:

1. Подключаем скрипт слайдера в конце тега BODY нашей главной страницы:

```
<script src="https://unpkg.com/swiper/swiper-bundle.min.
js"></script>
```

2. Далее подключаем стили для слайдера через тег LINK в блоке HEAD нашей главной страницы. Эти стили предоставляются разработчиком слайдера и необходимы для корректного отображения его слайдера на нашей странице:

```
<link rel="stylesheet" href="assets/css/plugins/swiper-
bundle.css">
```

3. После этого мы должны добавить специальные классы в нашу HTML верстку в соответствии с инструкцией разработчика. Благодаря наличию у DOM-элементов определенных классов, скрипт слайдера будет определять какой блок является оберткой слайдера, какой – областью отображения слайдов, а какие – самими слайдами. В листинге 4.16. данные классы выделены жирным:

Листинг 4.16. Добавление специальных классов в верстку слайдера:

```
<div class="slider">
  <div class="slider-col-1 main-slider-thumbs-container">
    <ul class="preview-list swiper-wrapper">
      <li class="preview-item swiper-slide" data-slide-to="1">
         <img src="assets/img/slider/slide11.jpg" width="98px" alt>
      </li>
      <li class="preview-item swiper-slide" data-slide-to="2">
         <img src="assets/img/slider/slide12.jpg" width="98px" alt>
      </li>
      <li class="preview-item swiper-slide" data-slide-to="3">
         <img src="assets/img/slider/slide13.jpg" width="98px" alt>
      </li>
   </ul>
</div>
<div class="slider-col-2">
   <div class="slider-item-field main-slider-container">
      <div class="slider-item-wrapper swiper-wrapper">
         <div class="slider-item swiper-slide">
           <a href="#"><img src="assets/img/slider/slide11.jpg"
                 width="1030px" height="360px" alt></a>
         </div>
         <div class="slider-item swiper-slide">
           <a href="#"><img src="assets/img/slider/slide12.jpg"
                 width="1030px" height="360px" alt></a>
         </div>
         <div class="slider-item swiper-slide">
            <a href="#"><img src="assets/img/slider/slide13.jpg"
                  width="1030px" height="360px" alt></a>
         </div>
       </div>
       <div class="swiper-pagination"></div>
   </div>
 </div>
</div>
```

4. На последнем этапе подключения слайдера мы, используя инструкцию разработчика, прописываем в нашем основном файле скрипта main.js необходимую конфигурацию слайдера, таким образом настраивая его. Код конфигурации слайдера представлен в листинге 4.17:

Листинг 4.17. Конфигурация слайдера Swiper

```
// Main page SWIPER slider config

const mainSlider = () => {
     var mainSliderThumbs = new Swiper(".main-slider-thumbs-
container", {
        slidesPerView: 4,
        loopedSlides: 4,
        watchSlidesVisibility: true,
        watchSlidesProgress: true,
        direction: "vertical"
    });

    var mainSlider = new Swiper(".main-slider-container", {
        spaceBetween: 10,
        loop: true,
        loopedSlides: 4,
        keyboard: {
            enabled: false,
        },
        thumbs: {
            swiper: mainSliderThumbs
        },
        pagination: {
            el: '.main-slider-container .swiper-pagination',
            clickable: true,
        },
        autoplay: {
            delay: 5000,
        }
    });
};

try {
    mainSlider();
} catch(err) {

}
```

4.12.3. Скрипт отправки форм

Для реализации скрипта "Отправка пользователем формы подписки на рассылку" мы будем использовать технологию AJAX – Asynchronous JavaScript and XML. Технология AJAX позволяет обновлять содержимое страницы (при отправке и получении) без перезагрузки самой HTML страницы.

Перед написанием скрипта отправки форм важно знать, что введенные пользователем данные (наличие формы определяется наличием тега FORM) собираются в особом формате данных, который называется FormData. Этот формат имеет свои особенности, которые нужно учитывать при конфигурации методов отправки данных сервер.

Для отправки и получения информации можно использовать несколько возможных решений. Устаревшее решение, которые сейчас уже не используется в новых проектах – XMLHttpRequest. На смену данной технологии пришли новые, более оптимизированные решения. Существуют даже специальные библиотеки для управления отправкой и получение данных с страниц сайта. Одна из популярных библиотек называется axios. Однако в нашем проекте мы будем использовать встроенный JavaScript инструментарий для работы с сервером – Fetch API. Для большинства современных сайтов Fetch API – наиболее оптимальное решение, не уступающее по своему функционалу и производительности сторонним библиотекам.

Написанный нами код формы не только управляет отправкой формы, но и предлагает хорошее взаимодействие с пользователем. При работе пользователя с формой мы информируем его обо всех основных состояниях:

1. Пользователь нажал кнопку "Подписаться" данные отправляются на сервер – пользователь видит специальный спиннер (рис. 4.2)

Рис.4.2. Спиннер, сигнализирующий об инициализации отправки

2. При успешной отправке сообщения, пользователю демонстрируется модальное окно с благодарностью за подписку.
3. Если же отправка не удалась, то пользователю информируют об этом с помощью модального окна (рис.4.3.).

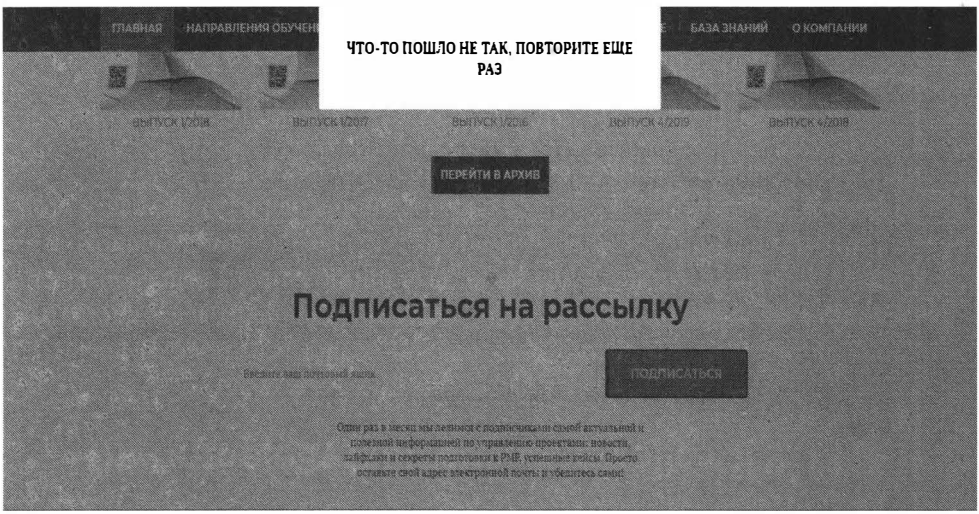


Рис. 4.3. Информирование пользователя о неудачной отправке формы

Итоговый скрипт отправки формы представлен в листинге 4.18

Листинг 4.18. Функция отправки формы:

```
// Forms

const forms = (formSelector, url) => {
    const forms = document.querySelectorAll(formSelector);

    const message = {
        loading: "assets/img/form/spinner.svg",
        success: "Благодарим за подписку на рассылку nit.center",
        failed: "Что-то пошло не так, повторите еще раз"
    };

    const postData = async (data) => {
        let result = await fetch(url, {
            method: "POST",
            body: data,
        });

        return await result.text();
    };

    function bindPostData(form) {
        form.addEventListener("submit", (evt) => {
            evt.preventDefault();

            const statusMessage = document.createElement("img");
            statusMessage.src = message.loading;
            statusMessage.style.cssText = `
                display: block;
                margin: 0 auto;
            `;
            form.insertAdjacentElement("afterend", statusMessage);

            const formData = new FormData(form);

            postData(formData)
            .then(() => {
                showThanksModal(message.success, form);
                statusMessage.remove();
            })
            .catch(() => {
                showThanksModal(message.failed, form);
                statusMessage.remove();
```

```
            })
            .finally(() => {
                form.reset();
            });
        });
    }

    forms.forEach(item => {
        bindPostData(item);
    });

    function showThanksModal(message, parent) {

        const thanksModal = document.createElement("div");
        thanksModal.classList.add("modal");
        thanksModal.classList.add("modal-show");
        thanksModal.innerHTML = `
            <div class="modal__content">
                <div class="modal__title">${message}<div>
            </div>
        `;
        document.body.style.overflow = "hidden";

        parent.appendChild(thanksModal);
        setTimeout(() => {
            thanksModal.remove();
            document.body.style.overflow = "";
        }, 4000);
    }
};

try {
    forms(".subscribe-form", "http://localhost:3000/requests");
} catch(err) {

}
```

Глава 5.

Основы языка PHP

Введение в PHP

PHP – это простой, но в то же время очень мощный язык программирования, разработанный для создания HTML-содержимого. В данной части книги приводятся основные сведения о синтаксисе языка PHP, рассматриваются его операторы, некоторые функции, а также работа с базами данных и обработка форм. Наша задача – не написать очередной самоучитель по PHP, а предоставить достаточно материала, чтобы вы могли самостоятельно разобраться в исходных кодах вашего будущего сайта.

Способы использования PHP

PHP можно использовать тремя основными способами:

- Сценарии сервера – изначально PHP разработан для создания динамического веб-контента на стороне сервера, и это до сих пор является самой сильной стороной PHP. Для генерации HTML-кода вам нужен синтаксический анализатор PHP и веб-сервер, с помощью которого можно отправлять получившиеся документы браузерам клиентов. Кроме того, PHP так же стал популярным для генерации XML-документов, графики, Flash-анимации, PDF-файлов и многого другого. В этой книге мы сконцентрируемся именно на таком способе использования PHP.
- Сценарии командной строки – PHP может выполнять сценарии из командной строки, во многом как Perl, awk, или оболочка Unix. Вы можете использовать сценарии командной строки

для задач системного администрирования, таких как резервное копирование и парсинг журнала.

- Клиентские приложения с графическим интерфейсом пользователя – используя PHP-GTK, вы можете написать кроссплатформенное приложение, обладающие графическим интерфейсом пользователя (GUI), на PHP.

PHP работает на всех основных операционных системах, от вариантов Unix, в том числе Linux, FreeBSD, Ubuntu, Debian и Solaris до Windows и Mac OS X. Он может использоваться со всеми основными веб-серверами, в том числе Apache, Microsoft IIS и серверы Netscape/iPlanet. Сам язык чрезвычайно гибок. Например, вы не ограничены выводом просто HTML или других текстовых файлов — вы можете генерировать документы любого формата. PHP имеет встроенную поддержку вывода файлов PDF, изображений GIF, JPEG, PNG, Flash-роликов. Одним из наиболее значительных качеств PHP является всесторонняя поддержка баз данных. PHP поддерживает все основные базы данных (включая MySQL, PostgreSQL, Oracle, Sybase, SQL MS, DB2 и ODBC-совместимые базы данных), а так же многие другие. Также поддерживаются NoSQL-базы данных, такие как SQLite и MongoDB. В общем, с использованием PHP создание веб-страниц с динамическим контентом, полученным из базы данных, является достаточно простым.

Установка PHP

PHP установлен на большинстве современных хостингов, поэтому вам ничего не придется делать – вам нужно только купить хостинг и поддержка PHP у вас уже будет. Если пока хостинг покупать не хочется, то можно загрузить PHP с официального сайта – www.php.net и установить на локальном компьютере. Еще лучше установить комплект XAMPP (https://www.apachefriends.org/ru/index.html), в состав которого входит веб-сервер Apache, интерпретатор PHP и сервер баз данных MySQL. Установив данный пакет программ на свой компьютер, вы сможете развернуть разработанный в книге сайт на локальном компьютере, не покупая хостинг!

Привет, мир!

Какой бы идиотской ни была данная традиция, но традиция есть традиция и изучение любого языка программирования начинается с написания простейшего приложения, выводящего одну строчку – Привет, мир!

Мы напишем такое приложение на PHP. Откройте любой текстовый редактор (мы рекомендуем использовать редакторы вроде Atom или MS Visual Studio Code – они бесплатные и обладают подсветкой синтаксиса и возможностью автодополнения PHP-кода, что очень помогает новичкам) и создайте файл hello.php, код которого приведен в листинге 5.1.

Листинг 5.1. Сценарий hello.php

```
<?php

echo "<h1><i>Привет, мир!</i></h1>";

?>
```

Загрузите этот файл на хостинг в каталог public_html или на локальный компьютер в каталог htdocs (он находится в каталоге, в который вы установили XAMPP, например, c:\xampp\htdocs). После этого откройте браузер и введите путь:

```
http://имя_сервера/hello.php
```

В случае с локальным компьютером имя сервера будет localhost. Если вы все сделали правильно, то увидите заветную строчку в браузере (рис. 5.1).

Рис. 5.1. Сценарий в работе

Разберемся, что есть что. Сам PHP-код заключен в теги <?php и ?>. Примечательно, что эти теги можно встраивать в HTML-код в произвольном месте, например, мы могли бы написать сценарий так:

```
<h1><i><?php echo "Привет, мир!"; ?></i></h1>
```

Другими словами в PHP-файле (в файле с расширением .php) кодом приложения считается все, что заключено в теги <?php ?>. Весь основной контент считается HTML-кодом и выводится без изменений. В нашем случае вся программа заключается в одном операторе echo, который выводит заключенный в кавычки текст.

Теперь, когда мы написали наше первое приложение, можно приступить к изучению синтаксиса языка.

5.1. Синтаксис PHP

Лексическая структура

Лексическая структура языка программирования – это набор основных правил, который определяет, как надо писать программы на этом языке. Это синтаксис самого низкого уровня языка и он определяет, какими должны быть имена переменных, какие символы могут использоваться для комментариев, и как операторы программы будут разделяться друг от друга.

Чувствительность к регистру

Имена пользовательских классов и функций, а также встроенные конструкции и ключевые слова, такие как *echo*, *while*, *class* и т.д., являются не чувствительными к регистру. Поэтому все три следующие строки эквивалентны:

```
echo("привет, мир");
ECHO("привет, мир");
EcHo("привет, мир");
```

С другой стороны, переменные являются чувствительные к регистру. Поэтому $name, $NAME и $NaME – это три разные переменные. 2.1.2. Операторы и точки с запятыми Оператор – это фрагмент кода PHP, делающий что-либо. Оператор может быть простым как присваивание значения пере-

менной или же сложным как цикл с многократными точками выхода. Рассмотрим небольшую подборку операторов PHP, включая вызовы функции, присваивание значения и оператор **if**:

```
echo "Привет, мир";
myFunction(42, "Привет");
$a = 1;
$name = "Федор";
$b = $a / 25.0;
if ($a == $b) {
    echo "Переменные равны!";
}
```

Для разделения простых операторов PHP использует точки с запятыми. Сложные (составные) операторы, использующие фигурные скобки, чтобы пометить блок кода, вроде проверки условия или цикла не нуждаются в точке запятой после закрывающей скобки. В отличие от других языков, в PHP точка с запятой является необходимой перед закрывающейся скобкой:

```
 if ($needed) {
echo "Она должна быть у нас!"; // точка с запятой необходима здесь
}            // здесь точка с запятой не нужна
```

Однако, вы можете не указывать точку с запятой перед закрытием тега PHP: Хороший стиль - указывать все необязательные точки с запятыми, поскольку это существенно упростит добавление кода в будущем.

Пробелы и разрывы строк

Пробелы и пробельные символы (новая строка, табуляция и т.д.) ни на что не влияют в PHP-программе. Вы можете разбить оператор на любое число строк или же записать в одну единственную строку. Например, следующий оператор:

```
raisePrices($inventory, $inflation, $costOfLiving, $greed);
```

можно записать с большим числом пробельных символов и переходов на следующую строку:

```
raisePrices ($inventory,
             $inflation,
```

```
        $costOfLiving,
        $greed);
```

или с меньшим числом пробелов:

```
raisePrices($inventory,$inflation,$costOfLiving,$greed);
```

Преимущество данной гибкости заключается в возможности форматирования вашего кода, чтобы сделать его более читабельным. Некоторые нерадивые программисты пользуются этой свободой в обратную сторону и создают совсем нечитаемый код - это не рекомендуется!

Комментарии

Комментарии предоставляют дополнительную информацию о коде людям, которые будут читать ваш код, но они полностью игнорируются PHP во время выполнения программы. Даже если вы думаете, что никто не будет читать ваш код, кроме вас, все равно рекомендуется добавлять комментарии для самого себя - спустя несколько месяцев ваш код будет выглядеть так, как будто его написал незнакомец.

В идеале ваши комментарии должны быть достаточно редкими, чтобы не мешать самому коду, но достаточно многочисленными, чтобы вы могли понять по ним, что происходит. Не нужно комментировать очевидные вещи, зато различные хитрые трюки очень нуждаются в комментариях.

Например, в следующем случае комментарий будет лишним:

```
$x = 17; // сохраняем значение 17 в переменной $x
```

Зато в следующем случае комментарий оправдан:

```
// конвертируем сущности &#nnn; в символы
$text = preg_replace('/&#([0-9])+;/e', "chr('\\1')",
$text);
```

PHP предоставляет несколько способов помещения комментариев в ваш код, все они пришли из существующих языков программирования - C, C++ и оболочки Unix. Обычно используется C-стиль для комментирования за пределами кода и стиль C++ для помещения комментариев в сам код.

Поддерживаются несколько типов комментариев:

- в стиле C++ - как было показано ранее.
- в стиле командной оболочки – когда используется символ #, например:

```
#######################
## Функции Cookie
#######################
```

- В стиле C – когда комментарием считается все, что заключено между символами /* и */:

```
/* Здесь начинается комментарий ?>
Нечто, что вы хотите отобразить в HTML HTML.
*/
```

Комментарии в стиле командной оболочки хоть и допускаются, но смотрятся чужеродно в PHP-коде. Поэтому пользуйтесь или комментариями в стиле C++ или в стиле C. Комментарии в стиле C++ лучше подходят, когда вам нужно закомментировать одну строчку или же написать комментарий в строке, чтобы пояснить ее назначение:

```
// Это комментарий в стиле C++
$server = "localhost";                    // имя сервера MySQL
```

Комментариями в стиле C удобно пользоваться, когда нужно закомментировать сразу множество строк или написать многострочный комментарий:

```
/* В этом разделе мы присвоим значения нескольким переменным.
В этом нет никакого смысла, но мы делаем это для удовольствия
*/
```

Литералы

Литерал - это значение данных, которое встречается непосредственно в программе. Примеры литераторов в PHP:

```
2001
0xFE
```

```
1.4142
"Hello"
'Hi!'
true
null
```

Идентификаторы

Идентификатор - это просто имя. В PHP идентификаторы используются для именования переменных, функций, констант и классов. Первый символ идентификатора должен быть ASCII-символом (в верхнем или нижнем регистре) или символом подчеркивания (_) или любым другим символов между ASCII 0x7F и ASCII 0xFF. После первого символа допускаются любые алфавитно-цифровые символы.

Имена переменных

Имена переменных всегда должны начинаться со знака доллара ($). После знака доллара должна идти буква или символ подчеркивания, а затем уже буквы, цифры, символы подчеркивания в любом количестве, исключая символы пробелов. Имена переменных чувствительны к регистру символов. Рассмотрим несколько примеров допустимых имен переменных:

```
$bill
$head_count
$MaximumForce
$I_HEART_PHP
$_underscore
$_int
```

А вот несколько недопустимых имен переменных:

```
$not valid // использован пробел в переменной
$| // использован недопустимый символ после доллара
$3wa // использована цифра после доллара
```

Поскольку переменные чувствительны к регистру символов, все это - разные переменные:

```
$hot_stuff
$Hot_stuff
$hot_Stuff
$HOT_STUFF
```

Имена функций

Имена функций следуют тем же правилам что и переменные, только не начинаются со знака доллара ($) и не чувствительны к регистру. Рассмотрим несколько допустимых имен функций:

```
tally
list_all_users
deleteTclFiles
LOWERCASE_IS_FOR_WIMPS
_hide
```

Все эти имена относятся к одной и той же функции:

```
howdy HoWdY HOWDY HOWdy howdy
```

Имена классов

Имена классов соответствуют стандартным правилам для PHP-идентификаторов и они также не чувствительны к регистру:

```
Person
account
```

Имя класса **stdClass** зарезервировано.

Константы

Константа - это идентификатор для простого значения. Константами могут быть только логические (булевые), целые, вещественные значения и строки. Как только константа установлена, она больше не может быть изменена. Определить константы можно с помощью функции define():

```
define('PUBLISHER', 'Nauka i Tekhnika');
echo PUBLISHER;
```

5.1.1. Ключевые слова

Ключевые слова (или зарезервированные слова) - это слова, определенные в языке для его базовой функциональности. Вы не можете использовать эти слова в именах переменных, функциях, классах или константах. Все ключевые слова являются регистро-независимыми. В таблице 5.1 приведены ключевые слова PHP.

Таблица 5.1. Ключевые слова PHP

__CLASS__	echo	insteadof
__DIR__	else	interface
__FILE__	elseif	isset()
__FUNCTION__	empty()	list()
__LINE__	enddeclare	namespace
__METHOD__	endfor	new
__NAMESPACE__	endforeach	or
__TRAIT__	endif	print
__halt_compiler()	endswitch	private
abstract	endwhile	protected
and	eval()	public
array()	exit()	require
as	extends	require_once
break	final	return
callable	finally (начиная с PHP 5.5)	static
case	for	switch
catch	foreach	throw
class	function	trait
clone	goto	try
const	global	unset()
continue	if	use
declare	implements	var
default	include	while
die()	include_once	xor
do	instanceof	yield (начиная с PHP 5.5)

В дополнение вы не можете использовать в качестве идентификаторов имена встроенных PHP-функций.

5.1.2. Типы данных

Язык PHP предоставляет восемь типов значений или типов данных. Четыре типа являются скалярными (содержат одно значение): целые числа, числа с плавающей запятой, строки и булевые (логические) значения. Два составных: массивы и объекты. Оставшиеся два являются специальными типами данных: ресурс и NULL.

Целые числа

Целые числа (integer) - это числа, не имеющее дробной части, например, 1, 12 и 256. Диапазон доступных значений зависит от вашей платформы, но обычно это от -2,147,483,648 до +2,147,483,647. Как видите, этот диапазон эквивалентен диапазону типа данных **long** в компиляторе C. К сожалению, стандарт C не задает точно диапазон типа **long**, поэтому на некоторых системах диапазон типа **integer** может отличаться от приведенного здесь. Целочисленные литералы могут быть записаны в десятичном, восьмеричном, шестнадцатеричном или двоичном видах. Десятичные значения представлены как последовательность цифр без лидирующего нуля. Последовательность может начинаться со знака плюс (+) или минус (-). Если знак не указан, подразумевается положительный знак. Примеры целых чисел, записанных в десятичной системе:

```
1998
-641
+33
```

В восьмеричной системе число представляется как последовательность цифр от 0 до 7 с лидирующим нулем. Подобно десятичным числам, восьмеричные также могут быть положительными или отрицательными. Рассмотрим несколько примеров восьмеричных значений и эквивалентные им десятичные:

```
0755 // десятичное 493
+010 // десятичное 8
```

Шестнадцатеричные значения начинаются с последовательности 0x (или с последовательности 0X), за которой следует последовательность цифр (0-9) или букв (A-F). Буквы могут быть, как в нижнем, так и в верхнем регистре, но обычно записываются в верхнем регистре. Подобно предыдущим значениям, вы можете добавить знак плюса или минуса в шестнадцатеричные числа:

```
0xFF // десятичное 255
0x10 // десятичное 16
-0xDAD1 // десятичное -56017
```

Целые числа также могут быть представлены в двоичной системе. Такие числа начинаются с последовательности 0b (или с 0B), за которой следует последовательность цифр от 0 до 1. Подобно другим системам счисления, вы можете добавить знак плюса или минуса:

```
0b00000001 // десятичное 1
0b00000010 // десятичное 2
-0b10 // десятичное -2
```

Числа с плавающей точкой

Числа с плавающей точкой (их также часто называют вещественными числами или числами с плавающей запятой) представлены в виде числовых значений с десятичными знаками. Подобно целым числам, диапазон чисел с плавающей точкой зависит от платформы машины. В PHP диапазон вещественных чисел эквивалентен диапазону типа данных **double** компилятора C. Обычно это от 1.7E-308 до 1.7E+308 с 15 знаками точности. Если вам нужно больше точности или более широкий диапазон значений, вы можете воспользоваться расширениями BC или GMP. PHP распознает числа с плавающей точкой в двух форматах. Вот один из них, который мы используем каждый день:

```
3.14
0.017
-7.1
```

Но PHP также распознает числа в научной записи:

```
0.314E1 // 0.314*10^1 или 3.14
17.0E-3 // 17.0*10^(-3) или 0.017
```

Строки

Строки очень часто используются в веб-приложениях, поэтому PHP на уровне ядра поддерживает операции по созданию и обработке строк. Строка - это последовательность символов произвольной длины. Строковые литералы должны быть заключены в двойные или одинарные кавычки:

```
'большая собака'
"красная машина"
```

В двойных кавычках вы можете использовать интерполяцию переменных, чего нельзя делать в одинарных кавычках:

```
$name = "Билл";
echo "Привет, $name\n";
echo 'Привет, $name';
```

Вывод:

```
Привет, Билл
Привет, $name
```

В двойных кавычках можно также использовать Esc-последовательности (управляющие последовательности), как показано в таблице 5.2.

Таблица 5.2. Esc-последовательности, используемые в двойных кавычках

Esc-последовательность	Какой символ представляет
\"	Двойная кавычка
\n	Новая строка
\r	Возврат каретки
\t	Табуляция
\\	Обратный слеш
\$	Знак доллара
\{	Левая фигурная скобка

\}	Правая фигурная скобка
\[	Левая квадратная скобка
\]	Правая квадратная скобка
\0 .. \777	ASCII-символ, заданный в восьмеричном виде
\x0 .. \xFF	ASCII-символ, заданный в шестнадцатеричном виде

Логические (булевые) значения

Логическое значение позволяет сказать, истинно что-либо или нет. Подобно другим языкам программирования, PHP определяет некоторые значение как *true* (истина), а другие - как *false* (ложь). Истинность или ложность определяется условным оператором, например:

```
if ($alive) { ... }
```

В PHP следующие значения эквивалентны false:

- Ключевое слово false
- Целое число 0
- Число с плавающей запятой 0.0
- Пустая строка ("") и строка, содержащая "0"
- Массив с 0 элементов
- Объект без значений и функций
- Значение NULL

Массивы

Массив содержит группу значений, доступ к каждому из значений вы можете получить по индексу – номеру позиции (номер – это число, при этом 0 соответствует первой позиции) или некоторому имени (строке), которое называется ассоциативным индексом:

```
$person[0] = "Эдисон";
$person[1] = "Ванкель";
$person[2] = "Крэппер";
$creator['Лампочку'] = "Эдисон";
$creator['Роторный двигатель'] = "Ванкель";
$creator['Туалет'] = "Крэппер";
```

Конструкция array() создает массив. Рассмотрим два примера:

```
$person = array("Эдисон", "Ванкель", "Крэппер");
$creator = array('Лампочку' => "Эдисон",
                 'Роторный двигатель' => "Ванкель",
                 'Туалет' => "Крэппер");
```

Вместо конструкции array() начиная с версии PHP 5.4 для создания массивов можно использовать сокращенную запись []. Например, перепишем определение первого из вышеприведенных массивов в сокращенной форме:

```
$person = ["Эдисон", "Ванкель", "Крэппер"]; // начиная с PHP 5.4
```

Рассмотрим несколько циклов, позволяющих "пройтись" по элементам массива (наиболее часто используется цикл **foreach**):

```
foreach ($person as $name) {
      echo "Привет, {$name}\n";
}

foreach ($creator as $invention => $inventor) {
            echo "{$inventor} создал {$invention}\n";
}
```

Вывод:

```
Привет, Эдисон
Привет, Ванкель
Привет, Крэппер
Эдисон создал Лампочку
Wankel создал Роторный двигатель
Crapper создал Туалет
```

Вы можете отсортировать элементы массива с помощью разных функций сортировки:

```
sort($person);
// $person теперь = array("Ванкель", "Крэппер", "Эдисон")
```

Объекты

PHP также поддерживает объектно-ориентированное программирование (ООП). ООП предоставляет особый подход к проектированию, упрощает отладку и обслуживание, а также делает удобным повторное использование кода.

Классы - это кирпичики объектно-ориентированного дизайна. Класс - это определение множества объектов, имеющих общую структуру, состоящую из свойств (переменных), и общее поведение, состоящее из методов (функций).

Классы определяются ключевым словом **class**:

```
class Person {
      public $name = '';
      function name ($newname = NULL)
      {
            if (!is_null($newname))
            {
                  $this->name = $newname;
            } return $this->name;
      }
}
```

Как только класс определен, вы можете создать на его основе любое число объектов (экземпляров класса). Объекты определяются с помощью ключе-

вого слова **new**, а доступ к свойствам и методам объекта можно получить с помощью конструкции ->:

```
$ed = new Person;
$ed->name('Эдисон');
echo "Привет, {$ed->name}\n";
```

Ресурсы

Ресурс (resource) – это специальная переменная, содержащая ссылку на внешний ресурс. Многие модули содержат несколько функций для взаимодействия с внешним миром. Например, у каждого расширения базы данных есть как минимум функция для подключения к базе данных, функция для отправки запроса к базе данных и функция закрытия соединения с базой данных. Поскольку у вас может быть несколько одновременных соединений с базой данных, функция подключения возвращает вам ресурс (или дескриптор) – некий уникальный идентификатор (или дескриптор), позволяющий вам идентифицировать соединение.

У каждого активного ресурса есть свой уникальный идентификатор. Каждый идентификатор - это индекс во внутренней таблице PHP, содержащей информацию обо всех активных ресурсах. В этой таблице PHP хранит информацию о каждом ресурсе, включая информацию о числе ссылок на ресурс в коде приложения. Тип resource содержит специальные указатели на открытые файлы, соединения с базой данных, области изображения и тому подобное. Когда последняя ссылка на ресурс исчезает, вызывается расширение, создавшее ресурс, чтобы освободить память, закрыть соединения и т.д. для этого ресурса:

```
$res = database_connect(); //фиктивная функция соединения с
БД database_query($res);
$res = "boo"; // соединение с БД будет автоматически закрыто,
              // т.к. переменная $res была переопределена
```

Большинство расширений предлагают отдельные функции закрытия ресурса. Их использование считается хорошим стилем и подчеркивает уровень программиста: вы явно закрываете ресурс, а не полагаетесь на автоматическую очистку.

Определить, является ли значение ресурсом, позволяет функция is_resource():

```
if (is_resource($x)) { // $x - ресурс }
```

Callback-функции

Callback-функции - это функции или методы объектов, используемые некоторыми другими функциями, например, call_user_func(). Callback-функции также могут быть созданы методом create_function() и с помощью анонимных функций:

```
$callback = function() {
      echo "вызвана callback-функция";
}
call_user_func($callback);
```

Вывод:

```
  вызвана callback-функция
```

NULL

Для типа данных NULL допустимо только одно значение, представленное регистро-независимым ключевым словом NULL. Значение NULL представляет собой переменную, у которой нет значения (подобно значению *undef* в Perl или *none* в Python):

```
$aleph = "beta";
$aleph = null; // значение переменной потеряно $aleph = Null;
// то же самое $aleph = NULL;
// то же самое
```

Функция is_null() позволяет проверить, нет ли у переменной значения:

```
if (is_null($x)) { // у переменной $x нет значения, $x = NULL; }
```

5.1.3. Переменные

По своей сути переменная – это область памяти, доступ к которой осуществляется по имени (идентификатору). Как уже было сказано ранее, перемен-

ные в PHP - это идентификаторы, название которых начинается со знака доллара ($) и так далее, например, $name.

Переменные могут хранить значение любого типа. В PHP нет проверки типа времени компиляции или времени выполнения. Вы можете легко заменить значение переменной значением другого типа:

```
$name = 'Mark';          // строка
$name = 5;          // целое число
```

Переменные переменных

Вы можете сослаться на значение переменной, имя которой сохранено в другой переменной, для этого используется дополнительный знак доллара ($). Например:

```
$foo = "bar";
$$foo = "baz";
```

После выполнение второго оператора у переменной $bar будет значение "baz". Переменная переменной берет значение переменной и рассматривает его как имя переменной.

Переменные-ссылки

В PHP можно использовать переменные-ссылки, то есть переменные, ссылающиеся на другие переменные. То есть получается, что у одной переменной будет два имени: основное и псевдоним (переменная-ссылка). Чтобы сделать $black псевдонимом для переменной $white, используйте синтаксис:

```
$black =& $white
```

Старое значение переменной $black будет потеряно. Вместо этого, $black теперь является вторым именем для значения, сохраненного в переменной $white.

```
$bigLongVariableName = "на";
$short =&$bigLongVariableName;
$bigLongVariableName .= " PHP!";
print "\$short = $short\n";
print "Long = $bigLongVariableName\n";

$short = на PHP!
Long = на PHP!

$short = "Программируем $short";
print "\$short = $short\n";
print "Long = $bigLongVariableName\n";

$short = Программируем на PHP!
Long = Программируем на PHP!
```

После присваивания две переменных работают как два альтернативных имени для одного и того же значения. Удаление (с помощью unset()) одного из имен никак не отображается на остальных именах:

```
$white = "снег";
$black =& $white;
unset($white);
print $black;
```

Вывод: снег

Функции могут возвращать значения по ссылке, например, чтобы избежать копирования больших строк или массивов.

Области видимости переменных

Область видимости переменной зависит от места объявления переменной и определяет, в каких частях программы переменная будет доступна. В PHP существует четыре типа области переменных: локальная, глобальная, статическая область и область параметров функции.

Локальная область видимости

Переменные, объявленные в функции, локальны для этой функции. Данные переменные видимы только в коде этой функции. Они не доступны за пределами функции. Также, по умолчанию, переменные, определенные за

пределами функции (называемые глобальными), недоступны внутри функции. Рассмотрим пример, в котором функция обновляет локальную переменную вместо глобальной:

```
function updateCounter() { $counter++; }
$counter = 10;
updateCounter();
echo $counter;
```

Вывод: 10

Переменная $counter внутри функции является локальной для этой функции, поскольку другое не определено явно. Функция увеличивает свою локальную переменную $counter, которая будет уничтожена, как только функция завершит свою работу.

Глобальная переменная $counter не будет изменена и ее значение так и останется равным 10.

Локальные переменные существуют только в функциях. В отличие от других языков программирования в PHP вы не можете создавать переменные с областью видимости в цикле, ветках условия или других типах блоков.

Глобальная область видимости

Переменные, объявленные за пределами функции, являются глобальными. Доступ к ним можно получить из любой части программы, но, как уже было указано ранее, они недоступны внутри функций. Чтобы разрешить доступ функции к глобальной переменной, вы можете использовать ключевое слово **global** внутри функции.

Ключевое слово **global** используется для объявления, что переменная, к которой обращается функция, является глобальной. Давайте теперь перепишем функцию updateCounter() так, чтобы она получала доступ к глобальной переменной $counter:

```
function updateCounter()
{
global $counter;
$counter++;
}
$counter = 10;
```

```
updateCounter();
echo $counter;
```

Вывод: 11

Статические переменные

Значение статических переменных сохраняется между вызовами функции, но при этом переменная видима только в самой функции. Объявить статическую переменную можно с помощью ключевого слова **static**. Например:

```
function updateCounter() {
      static $counter = 0;
      $counter++;
      echo "Статический счетчик = {$counter}\n";
}

$counter = 10;
updateCounter();
updateCounter();
echo "Глобальный счетчик = {$counter}\n";

Вывод:
Статический счетчик = 1
Статический счетчик = 2
Глобальный счетчик = 10
```

Параметры функции

Определение функции может содержать именованные параметры:

```
function greet($name)
{
      echo "Привет, {$name}\n";
}
greet("Фред");
Привет, Фред
```

Параметры функции локальны и доступны только внутри своей функции. В нашем случае переменная $name недоступна за пределами функции greet().

5.1.4. Выражения и операторы

Выражение - конструкция языка PHP, использующаяся для вычисления значения. Простейшие выражения - это обычные значения (0, "строка") и переменные. Значения вычисляются в самих себя (то есть значение **0** и есть сам ноль), а значение переменной хранится в области памяти, на которую указывает переменная. Более сложные выражения формируются с использованием простых выражений (значений и переменных) и операторов.

Оператор берет несколько значений (они называются операндами) и делает что-то с ними, например, складывает их вместе. Операторы в PHP представлены символами пунктуации, например, + ("плюс") – этот символ знаком нам еще из математики. Некоторые операторы изменяют свои операнды, но подавляющее большинство - нет.

В таблице 5.3 приведены операторы PHP, многие из них были позаимствованы из языков C и Perl. Колонка **П** означает приоритет оператора, операторы приведены в порядке понижения приоритета - от самого высокого до самого низкого. Колонка ПВ означает порядок выполнения, который может быть **Л** (слева направо), **П** (справа налево) или **Н** (неассоциативный).

Таблица 5.3. Операторы PHP

П	ПВ	Оператор	Операция
21	Н	clone, new	Создание нового объекта
20	Л	[	Индекс массива
19	П	~	Побитное отрицание (NOT)
	П	++	Инкремент
	П	--	Декремент
	П	(int), (bool), (float), (string), (array), (object), (unset)	Приведение типа
	П	@	Подавление ошибок
18	Н	instanceof	Проверка типа

17	П	!	Логическое отрицание (NOT)
16	Л	*	Умножение
	Л	/	Деление
	Л	%	Остаток от деления
15	Л	+	Сложение
	Л	-	Вычитание
	Л	.	Конкатенация строки
14	Л	<<	Побитный сдвиг влево
	Л	>>	Побитный сдвиг вправо
13	Н	<, <=	Меньше, меньше или равно
	Н	>, >=	Больше, больше или равно
12	Н	==	Равно
	Н	!=, <>	Не равно
	Н	===	Тип и значения равны
	Н	!==	Тип и значения не равны
11	Л	&	Побитное И (AND)
10	Л	^	Побитное исключающее ИЛИ (XOR)
9	Л	|	Побитное ИЛИ (OR)
8	Л	&&	Логическое И (AND)
7	Л	||	Логическое ИЛИ (OR)

6	Л	?:	Условный оператор
5	Л	=	Присваивание
	Л	+=, -=, *=, /=, .=, %=, &=, |=, ^=, ~=, <<=, >>=	Присваивание с операцией
4	Л	and	Логическое И (AND)
3	Л	xor	Логическое исключающее ИЛИ (XOR)
2	Л	or	Логическое ИЛИ (OR)
1	Л	,	Разделитель списка

Количество операндов

Большинство операторов в PHP являются бинарными, то есть они соединяют два операнда (или выражения) в одно, более сложное выражение. PHP также поддерживает некоторые унарные операторы, которые конвертируют одно выражение в более сложное выражение. Еще PHP поддерживает единственный тернарный оператор, который комбинирует три выражения в одно более сложное.

Приоритет операторов

Порядок, в котором вычисляются операторы в выражении, зависит от их относительного порядка. Например, вы можете написать:

```
2 + 4 * 3
```

Как видно из таблицы 5.3, у операторов сложения и умножения разный приоритет, приоритет у умножения выше, чем у сложения. Поэтому сначала будет выполнено умножение, а только потом сложение. Следовательно, мы получим `2 + 12` или `14` в качестве результата. Если бы приоритеты сложения и умножения поменять, мы бы получили результат `6 * 3 = 18`.

Чтобы задать определенный порядок, вы можете группировать операторы в скобки. В нашем примере, чтобы получить `18`, вам нужно использовать выражение:

```
(2 + 4) * 3
```

Вы можете составлять более сложные выражения (выражения, содержащие более одного оператора), просто помещая операнды и операторы в надлежащем порядке так, чтобы их относительный приоритет привел к желаемому результату. Однако для улучшения читабельности и восприятия кода, а также уменьшения числа ошибок настоятельно рекомендуется использовать скобки в сложных выражениях. Понимание приоритета позволяет писать код подобным образом:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

Однако такой код трудно читать, и почти всегда он не делает того, что задумал программист.

******Совет. Когда вы имеете дело со сложными правилами приоритета, помните, что все эти правила могут быть сведены к двум следующим:**

- **Умножение и деление имеют более высокий приоритет, чем сложение и вычитание**
- **Используйте скобки для всего остального.**

Порядок выполнения операторов

Порядок выполнения определяет порядок, в котором будут выполнены операторы с одним уровнем приоритета. Например, посмотрим на выражение:

```
2 / 2 * 2
```

У операторов умножения и деления одинаковый приоритет, но результат выражения зависит от того, какая операция будет выполнена первой:

```
2 / (2 * 2)    // 0.5
(2 / 2) * 2    // 2
```

Порядок выполнения операторов деления и умножения - слева на право. Это означает, что в нашем примере результат будет равен 2.

Неявное приведение типа

Многие операторы ожидают получить операнды определенного типа, например, бинарные математические операторы требуют, чтобы оба оператора были одинакового типа. Как мы уже знаем, переменные PHP могут хранить целые числа, числа с плавающей точкой, строки и др. При необходимости PHP преобразовывает значения одного типа в другой.

Преобразование значения одного типа в другой называется приведением типа (в англ. литературе - *casting*). Используемое неявное приведение типа в PHP называется манипуляцией с типом (*type juggling*). Правила неявного приведения типа для арифметических операторов представлены в таблице 5.4.

Таблица 5.4. Правила неявного приведения типов для бинарных арифметических операторов

Тип первого операнда	Тип второго операнда	Преобразование
Целое число	Число с плавающей запятой	Целое значение будет конвертировано в число с плавающей запятой
Целое число	Строка	Строка будет конвертирована в число. Если значение после преобразование будет числом с плавающей запятой, целое число (первый операнд) будет преобразовано в число с плавающей запятой
Число с плавающей запятой	Строка	Строка будет конвертирована в число с плавающей запятой

Некоторые другие операторы ожидают операнды других типов, поэтому у них есть свои собственные правила приведения типов. Например, оператор конкатенации строк перед конкатенацией преобразует в строку оба операнда.

```
3 . 2.74  // результатом будет строка 32.74
```

Вы можете использовать строку везде, где PHP ожидает число. Строка должна начинаться с целого числа или с числа с плавающей точкой. Если в

строке не было найдено число, строка преобразуется в число 0. Если строка содержит точку (.) или букву E (или e), строка будет преобразована в число с плавающей точкой. Точнее, обработка строки происходит слева направо и заканчивается на первом символе, который не может быть обработан логически. Например:

```
"9 Lives." - 1;          // 8 (integer)
"3.14 Pies" * 2;               // 6.28 (float)
"9 Lives." - 1;          // 8 (float)
"1E3 Points of Light" + 1;     // 1001 (float)
```

Арифметические операторы

Арифметические операторы - это операторы, используемые нами практически каждый день. Практически все арифметические операторы являются бинарными, однако, арифметическое отрицание и унарный плюс (арифметическое утверждение или пустая арифметическая операция) являются унарными. Эти операторы требуют числовых значений, а все нечисловые значения будут конвертироваться в числовые, согласно правил, описанных в предыдущем разделе. К арифметическим оператором относятся:

Сложение (+)

Результат сложения - сумма двух операндов.

Вычитание (-)

Результат вычитания - разница между двумя операндами, то есть значение второго операнда вычитается из первого.

Умножение (*)

Результат оператора умножения - это произведение двух операндов. Например, `3 * 4` равно `12`.

Деление (/)

Результатом оператора деления - это частное двух операторов. Результатом деления двух целых чисел может быть или целое число (`4 / 2`) или число с плавающей запятой (`1 / 2`).

Остаток отделения (%)

Оператор `%` преобразует оба операнда в целые числа и возвращает остаток от деления первого операнда на второй. Например, `10 % 6 = 4`.

Арифметическое отрицание (-)

Оператор арифметического отрицания или просто унарный минус возвращает операнд, умноженный на `-1`, что изменяет знак числа. Например, `-(3 - 4)` эквивалентно `1`. Арифметическое отрицание отличается от оператора вычитания. Арифметическое отрицание - это унарный оператор и он ставится всегда перед операндом. Вычитание - это бинарный оператор и он находится между операндами.

Унарный плюс или пустая арифметическая операция (+)

Унарный плюс возвращает операнд, умноженный на `+1`, что никак не изменяет его значения и вообще не имеет никакого эффекта. Этот оператор используется для явного указания знака значения, например, `+(3 - 4)` равно `-1`, как и `(3 - 4)`.

Оператор конкатенации строки

Манипулирование со строками - это основная работа PHP-приложения, поэтому для конкатенации (объединения) строк в PHP используется отдельный оператор конкатенации - (`.`). Оператор конкатенации присоединяет второй операнд (указанный справа от точки) к первому операнду (указанному слева от точки) и возвращает объединенную строку. При необходимости перед конкатенацией операнды преобразуются в строки. Например:

```
$n = 5;
$s = 'Здесь было ' . $n . ' уток.';
// $s = 'Здесь было 5 уток'
```

Оператор конкатенации очень эффективен, поскольку множество операций в PHP сводится к конкатенации строк.

Операторы инкремента и декремента

В программировании одними из наиболее часто используемых операций являются инкремент (увеличение значения на единицу) или декремент (уменьшение значения на единицу). Унарные операторы инкремента (++)

или декремента (--) позволяют быстро выполнить эти часто используемые операции. Данные операторы уникальны тем, что они работают только с переменными. Операторы изменяют значения операндов и возвращают значение.

Существует два способа использования операторов инкремента и декремента в выражениях. Если вы поместите оператор инкремента/декремента перед операндом, он сразу возвратит новое значение операнда (увеличенное или уменьшенное). Если вы поместите оператор после операнда, он вернет исходное значение операнда (перед увеличением или уменьшением). В таблице 5.5 приведены различные варианты операции.

Таблица 5.5. Операции инкремента и декремента

Оператор	Название	Возвращаемое значение	Эффект на $var
$var++	Пост-инкремент	$var	Возвращает значение $var, затем увеличивает $var на единицу.
++$var	Пре-инкремент	$var + 1	Увеличивает $var на единицу, затем возвращает значение $var.
$var--	Пост-декремент	$var	Возвращает значение $var, затем уменьшает $var на единицу.
--$var	Пре-декремент	$var – 1	Уменьшает $var на единицу, затем возвращает значение $var.

Приведем пример:

```
<?php
echo "<h4>Пост-инкремент</h4>";
$var = 5;
echo "Должно быть 5: " . $var++ . "<br />\n";
echo "Должно быть 6: " . $var . "<br />\n";

echo "<h4>Пре-инкремент</h4>";
$var = 5;
echo "Должно быть 6: " . ++$var. "<br />\n";
```

```
echo "Должно быть 6: " . $var . "<br />\n";

echo "<h4>Пост-декремент</h4>";
$var = 5;
echo "Должно быть 5: " . $var-- . "<br />\n";
echo "Должно быть 4: " . $var . "<br />\n";

echo "<h4>Пре-декремент</h4>";
$var = 5;
echo "Должно быть 4: " . --$var . "<br />\n";
echo "Должно быть 4: " . $var . "<br />\n";
?>
```

Операторы инкремента/декремента могут быть применены как к числам, так и к строкам. Увеличение на 1 алфавитного символа превращает его в следующий по алфавиту символ. Как показано в таблице 5.6, инкремент `"z"` или `"Z"` превращает ее обратно в `"a"` или `"A"` и увеличивает предыдущий символ на единицу (или вставляет новую `"a"` или `"A"` если это первый символ строки), как будто символы - это система счисления с основанием 26.

Таблица 5.6. Автоматический инкремент букв

Увеличиваем это	Получаем это
"a"	"b"
"z"	"aa"
"spaz"	"spba"
"K9"	"LO"
"42"	"43"

Операторы сравнения

Как подразумевает название, операторы сравнения используются для сравнения операндов. Результат таких операторов всегда `true`, если сравнение истинно или `false` в противном случае.

Операндами операторов сравнения могут быть числа, строки или одно число и одна строка. В зависимости от типов операндов способы проверки истинности немного отличаются. Чтобы понять, как работают операторы сравнения, давайте рассмотрим таблицу 5.7.

Таблица 5.7. Типы сравнения, выполняемого операторами сравнения

Первый операнд	Второй операнд	Сравнение (тип)
Число	Число	Числовое
Строка, полностью состоящая из чисел	Строка, полностью состоящая из чисел	Числовое
Строка, полностью состоящая из чисел	Число	Числовое
Строка, полностью состоящая из чисел	Строка, не полностью состоящая из чисел	Лексикографическое
Строка, не полностью состоящая из чисел	Число	Числовое
Строка, не полностью состоящая из чисел	Строка, не полностью состоящая из чисел	Лексикографическое

Нужно отметить очень важную особенность: если сравниваются две строки, состоящие из чисел, то они будут преобразованы в числа и будут сравниваются как числа. Исключение составляет оператор тождественного равенства (=== см. ниже). Если у вас есть две строки, состоящие из чисел, и вам нужно сравнить их лексикографически, используйте функцию `strcmp()`.

К операторам сравнения относятся:

Равно (==)

Если оба операнда равны, этот оператор возвращает `true`, в противном случае он возвращает `false`

Тождественно равно (===)

Если оба оператора равны и одного типа, этот оператор возвращает `true`. В противном случае он возвращает `false`. Заметьте, что этот оператор не использует неявное приведение типов. Этот оператор полезен, когда вы не знаете, что сравниваемые значения являются значениями одного и того же типа. Например, строки `"0.0"` и `"0"` не равны. Оператор == сообщит, что они равны, но оператор === скажет, что они не равны.

Не равно (!= или <>)

Если операнды не эквивалентны, этот оператор возвратит `true` или `false` - в противном случае.

Тождественно не равны (!==)

Если операнды не равны или их тип отличается, этот оператор возвращает `true`, в противном случае он возвращает `false`

Больше чем (>)

Если операнд, указанный слева от оператора >, больше, чем операнд, указанный справа от оператора, этот оператор возвращает `true`. В противном случае вы получите `false`.

Больше или равно (>=)

Если левый операнд, *больше*, чем правый операнд, *или равен ему*, этот оператор возвращает `true`. В противном случае вы получите `false`.

Меньше чем (<)

Если левый операнд, меньше, чем правый операнд, этот оператор возвращает `true`. В противном случае вы получите `false`.

Меньше или равно (<=)

Если левый операнд, *меньше*, чем правый операнд, *или равен ему*, этот оператор возвращает `true`. В противном случае вы получите `false`.

Таблица 5.8. Сравнение различных типов

Первый операнд	Второй операнд	Результат
NULL или Строка	Строка	NULL преобразуется в "", числовое или лексикографическое сравнение
Логический или NULL	любой	Оба операнда приводятся к логическому типу, FALSE < TRUE
Объект	Объект	Два объекта равны, если они содержат одинаковые свойства и одинаковые значения, и являются экземплярами одного и того же класса. Встроенные классы могут определять свои собственные правила сравнения, объекты разных классов не сравниваются.

Число, Строка или Ресурс	Число, Строкаили Ресурс	Строки и ресурсы переводятся в числа, далее обычное числовое сравнение
Массив	Массив	Массивы с меньшим числом элементов считаются меньше, если ключ из первого операнда не найден во втором операнде - массивы не могут сравниваться, иначе идет сравнение соответствующих значений.
Объект	любой	Объект всегда больше
Массив	любой	Массив всегда больше

Поразрядные (побитовые) операторы

Поразрядные операторы работают с двоичным представлением своих операндов. Каждый операнд сначала переводится в свое двоичном представление, а потом уже выполняется определенный оператор. К побитовым операторам относят:

Побитовое отрицание (~)

Оператор побитового отрицания изменяет единицы в побитовом представлении на нули, а нули - на единицы. Перед выполнением отрицания числа с плавающей точкой будут преобразованы в целые числа. Если операнд - строка, то результатом также будет строка такой же длины, где каждый символ строки будет инвертирован (преобразован в двоичное преставление, в котором единицы заменены на нули, а нули - на единицы).

Побитовое И (&)

Поразрядный оператор *И* сравнивает каждый соответствующий бит в двоичном представлении операндов. Если биты установлены в 1, соответствующий бит в результате будет равен 1. В противном случае соответствующий бит в результате будет равен 0. Например, `0755 & 0671 = 0651`. Ничего не понятно? Давайте взглянем на двоичное представление. Преобразуем восьмеричное значение 0755 в двоичное 111101101. Восьмеричное число 0671 в двоичной системе будет выглядеть так: 110111001. Теперь мы можем просто сопоставить биты операндов и получить ответ:

```
  111101101
& 110111001
  ---------
  110101001
```

Двоичное число 110101001 в восьмеричной системе - 0651. Для преобразования чисел из одной системы счисления в другую вы можете использовать PHP-функции `bindec()`, `decbin()`, `octdec()` и `decoct()`.

Если оба операнда - строки, оператор возвращает строку, в которой каждый символ является результатом побитной операции *И* между двумя соответствующими символами в операндах. Длина результирующей строки будет равна длине более короткой строки из двух операндов. "Лишние" символы в более длинной строке будут проигнорированы. Например, `"wolf" & "cat"` = `"cad"`.

Поразрядное ИЛИ (|)

Поразрядный оператор ИЛИ сравнивает соответствующие биты в двоичном представлении операндов. Если оба бита равны 0, результирующий бит будет равен 0. В противном случае результирующий бит будет установлен в 1. Например: `0755 | 020 = 0775`.

Если оба операнда - строки, оператор возвращает строку, в которой каждый символ является результатом побитной операции ИЛИ между двумя соответствующими символами в операндах. Длина результирующей строки будет равна длине более длинной строки из двух операндов. Короткая строка будет дополнена недостающим (до длины более длиной строки) количеством двоичных нулей. Например, `"pussy" | "cat" = "suwsy"`.

Поразрядное исключающее ИЛИ (^)

Поразрядный оператор исключающее ИЛИ (XOR) сравнивает каждый бит в двоичном представлении операндов. Если любой бит в паре (но не оба!) равен 1, результат - 1. В противном случае результат - 0. Например, `0755 ^ 023 = 776`.

Если оба операнда - строки, оператор возвращает строку, в которой каждый символ является результатом побитной операции исключающее ИЛИ между двумя соответствующими символами в операндах. Длина результирующей строки будет равна длине более короткой строки из двух операндов. "Лишние" символы в более длинной строке будут проигнорированы. Например, `"big drink" ^ "AA" is "#(".`

Сдвиг влево (<<)

Оператор сдвига влево сдвигает влево биты в двоичном представлении левого операнда, на число мест, заданное в правом операнде. Оба операнда будут конвертированы в целые числа, если на момент вызова оператора они таковыми не являлись. При сдвиге влево на одно место все биты сдвигаются влево, а на образовавшееся справа место вставляется 0. Например, `3 << 1` (двоичное 11 будет сдвинуто на одно место влево) равно `6` (двоичное 110).

Каждый сдвиг влево (на одно место) равноценен умножению на 2. Результат сдвига влево - это умножение левого операнда на 2 в степени правого операнда.

Сдвиг вправо (>>)

Оператор сдвига вправо сдвигает вправо биты в двоичном представлении левого операнда, на число мест, заданное в правом операнде. Оба операнда будут конвертированы в целые числа, если на момент вызова оператора они таковыми не являлись. При сдвиге вправо на одно место все биты сдвигаются вправо, а на образовавшееся слева место вставляется 0. Если число отрицательно, что 1 будет вставленным как крайний левый бит числа. Самый правый (последний) бит отбрасывается. Например, `13 >> 1` (или двоичное 1101 сдвинутое на 1 бит вправо) даст в результате `6` (двоичное 110).

Логические операторы

Логические операторы предоставляют средства построения сложных логических выражений. Логические операторы расценивают свои операнды как логические значения, результатом операторов также являются логические значения. Вы можете использовать, как знаки пунктуации, соответствующие логическим операторам, так и названия операторов на английском языке.

К логическим операторам относятся:

Логическое И (`&&`, `and`)

Результатом логической операции *И* является `true`, если оба операнда истинны (равны `true`), в противном случае возвращается `false`. Если значение первого оператора - `false`, логический оператор *И* сразу возвращает значение `false`, даже не вычисляя значение второго оператора - в этом нет смысла. Вы можете использовать эту особенность в своих целях, напри-

мер, вы можете подключиться к базе данных, если некоторый флаг не равен false:

```
$result = $flag and mysql_connect();
```

Операторы && и and отличаются только своими приоритетами.

Логическое ИЛИ (||, or)

Результатом логической операции *ИЛИ* является true, если хотя бы один из операторов равен true. В противном случае результат будет false. Подобно оператору AND, оператор OR вычисляет сначала значение первого операнда и если оно равно true, то второй операнд не вычисляется. Вы можете использовать особенность этого оператора для обработки ошибки, если что-то пошло не так. Например:

```
$result = fopen($filename) or exit();
```

Операторы || и or отличаются только своими приоритетами.

Логическое исключающее ИЛИ (xor)

Результатом логического исключающего ИЛИ является true, если истинен какой-то один из операндов, но не оба. В противном случае возвращается false.

Логическое отрицание (!)

Оператор логического отрицания возвращает true, если операнд равен false, в противном случае (когда операнд равен true) возвращается false.

Операторы приведения типов

Хотя PHP является слабо типизированным языком, иногда полезно рассматривать значение как значение определенного типа. Операторы сведения типа (int), (float), (string), (bool), (array), (object) или (unset) позволяют вам принудительно свести значение к определенному типу. Чтобы использовать оператор приведения типа, поместите его слева от операнда. В таблице 5.9 приведены операторы приведения типа и их синонимы.

Таблица 5.9. Операторы приведения типа в PHP

Оператор	Синоним	Изменяет тип на
(int)	(integer)	Целый
(bool)	(boolean)	Логический
(float)	(double), (real)	Вещественный
(string)		Строка
(array)		Массив
(object)		Объект
(unset)		NULL

Операторы приведения типа не изменяют значение, но изменяют то, как будет это значение интерпретироваться. Рассмотрим следующий код:

```
$a = "5";
$b = (int) $a;
```

В результате переменной `$b` будет присвоено целое значение переменной `$a`, то есть число `5`, в то время как переменная `$a` останется строкой `"5"`. Чтобы изменить тип исходной переменной, просто присвойте результат обратно этой переменной:

```
$a = "5";      // строка "5"
$a = (int) $a;   // теперь $a хранит целое число 5
```

Не всегда приведение типа полезно. Приведение массива в числовой тип даст `1`, а приведение массива в строку даст строку `"Array"` (если массив пустой, то приведение даст 0).

Приведение объекта в массив создаст массив из свойств объекта, имена свойств будут отражены (маппированы) в их значения:

```
class Person
{
  var $name = "Фред";
  var $age = 35;
}

$o = new Person;
```

```
$a = (array) $o;

print_r($a);
Array (
[name] => Фред
[age] => 35
)
```

Вы можете привести массив к объекту, чтобы построить объект, свойства которого соответствуют ключам и значениям массива. Например:

```
$a = array('name' => "Фред", 'age' => 35, 'wife' =>
"Вильма");
$o = (object) $a;
echo $o->name;

Фред
```

Ключи, которые не являются допустимыми идентификаторами, будут недоступны при сведении массива к объекту, но будут восстановлены, когда объект вы обратно преобразуете в массив.

Оператор присваивания

Операторы присваивания сохраняют или обновляют значения в переменных. Операторы инкремента и декремента, рассмотренные ранее, также в некоторой степени являются операторами присваивания. Основной оператор присваивания - `=`, но в PHP есть много видов операторов присваивания, например, `+=` и `&=`.

Присваивание

Оператор присваивания (`=`) присваивает значение переменной. Левый операнд всегда переменная. Правый операнд может быть любым выражением - простым литералом, переменной или сложным выражением. Значение правого операнда сохраняется в переменной, заданной левым операндом.

Поскольку все операторы должны возвращать значение, оператор присваивания возвращает значение, которое будет присвоено переменной. Например, выражение `$a = 5` не только присваивает `5` переменной `$a`, но также возвращает значение `5` в случае использования в сложном выражении. Рассмотрим следующие выражения:

```
$a = 5;
$b = 10;
$c = ($a = $b);
```

Поскольку используются скобки, сначала вычисляется выражение `$a = $b`. Теперь и у `$a`, и у `$b` одно и то же значение - `10`. Это значение и будет результатом выражения `$a = $b`, оно же будет присвоено левому операнду (в этом случае `$c`). В результате у всех трех переменных будет одно и то же значение `10`.

Присваивание с операцией

В дополнение к основному оператору присваивания в PHP имеется несколько дополнительных операторов присваивания с операцией, которые экономят код. Эти операторы состоят из бинарного оператора, после которого следует знак равенства (=). Рассмотрим эти операторы:

Плюс-равно (+=)

Добавляет правый операнд к значению левого операнда, новое значение будет присвоено левому операнду. Следовательно, `$a += 5` эквивалентно `$a = $a + 5`.

Минус-равно (-=)

Вычитает правый операнд от значения левого операнда, новое значение будет присвоено левому операнду. Следовательно, `$a -= 5` эквивалентно `$a = $a - 5`.

Деление-равно (/=)

Делит значение левого операнда на значение правого операнда, результат присваивается левому операнду.

Умножение-равно (=)*

Умножает значение левого операнда на значение правого операнда, результат присваивается левому операнду.

Остаток-равно (%=)

Вычисляет `%` , то есть остаток от деления левого операнда на правый операнд, результат присваивается левому операнду.

Побитное-XOR-равно (^=)

Вычисляет значение побитной операции XOR между левым и правым операндами, результат присваивается левому операнду.

Побитное-AND-равно (&=)

Вычисляет значение побитной операции AND (И) между левым и правым операндами, результат присваивается левому операнду.

Побитное-OR-равно (|=)

Вычисляет значение побитной операции OR (ИЛИ) между левым и правым операндами, результат присваивается левому операнду.

Конкатенация-равно (.=)

Соединяет правый операнд со значением левого операнда, результат присваивается левому операнду. Выражение `$a .= $b` эквивалентно `$a = $a . $b`.

Разные операторы: подавление ошибок и другие

Осталось рассмотреть PHP-операторы, использующиеся для подавления ошибок, выполнения внешних команд и выбора значений:

Подавление ошибок (@)

Некоторые операторы или функции могут генерировать сообщения об ошибках. Оператор подавления ошибок позволяет подавить вывод этих сообщений.

Выполнение (`...`)

Оператор выполнения (два обратных апострофа, на клавиатуре находятся под тильдой) используется для запуска команд оболочки. Оператор возвращает вывод внешней команды. Например:

```
$listing = `ls -ls /tmp`;
echo $listing;
```

Оператор выбора (?:)

Условный оператор является единственным тернарным (троичным) оператором. Он вычисляет выражение перед `?`. Если выражение истинно (`true`), оператор возвращает значение выражения между `?` и `:`. В противном случае оператор возвращает выражение после `:`. Например:

```
<a href="<?= $url; ?>"><?= $linktext ? $linktext : $url;
?></a>
```

Если текст для ссылки `$url` присутствует в переменной `$linktext`, он используется для ссылки, в противном случае отображается URL, заданный переменной `$url`.

Тип (instanceof)

Оператор `instanceof` проверяет, является переменная объектом заданного класса, реализует ли она заданный интерфейс:

```
$a = new Foo;
$isAFoo = $a instanceof Foo;    // true
$isABar = $a instanceof Bar;    // false
```

5.1.5. Операторы управления выполнением

Язык PHP поддерживает несколько традиционных конструкций для управления потоком выполнения программы. К этим конструкциям относят условный оператор `if/else`, а также оператор `switch`, позволяющие выполнять (или не выполнять) различные участки кода в зависимости от условия. Также к этим конструкциям относятся циклы, например, `while` и `for`, использующиеся для повторного выполнения участков кода.

Оператор if

Оператор `if` проверяет истинность выражения и, если оно истинно, выполняет заданный программистом оператор:

if (выражение) оператор;

Чтобы задать альтернативный оператор, который будет выполнен, если выражение равно `false`, используется дополнительное ключевое слово `else`:

```
if (выражение)
        оператор1
 else
        оператор2
```

Например:

```
if ($user_validated)
      echo "Добро пожаловать!";
else
      echo "Доступ запрещен!";
```

Если вам нужно выполнить более одного оператора, используйте блок операторов - операторы, заключенные в фигурные скобки:

```
if ($user_validated) {
      echo "Добро пожаловать!";
      $greeted = 1;
}
else {
      echo "Доступ запрещен!";
      exit;
}
```

Язык PHP также предоставляет альтернативный синтаксис для этого оператора (как и для операторов циклов). Вместо заключения операторов блока в фигурные скобки, укажите после оператора `if` двоеточие (`:`), а в конце всей условной конструкции укажите ключевое слово `endif`. Следующий пример демонстрирует этот альтернативный синтаксис:

```
if ($user_validated):
      echo "Добро пожаловать!";
      $greeted = 1;
else:
      echo "Доступ запрещен!";
      exit;
endif;
```

Альтернативный синтаксис полезен, если у вас есть большие блоки HTML-кода внутри ваших операторов. Например:

```
<? if ($user_validated) :?>
<table>
      <tr>
            <td>Имя:</td><td>Sophia</td>
```

```
        </tr>
        <tr>
              <td>Фамилия:</td><td>Lee</td>
        </tr>
</table>
<? else: ?>
        Пожалуйста войдите.
<? endif ?>
```

Поскольку `if` - это тоже оператор, вы можете создавать цепочки, состоящие из операторов `if`. Рассмотрим еще один пример, демонстрирующий, как блоки помогают организовать мысли:

```
if ($good) {
        print("Хорошо!");
}
else {
        if ($error) {
              print("Ошибка!");
        }
        else {
              print("Плохо...");
        }
}
```

Вы можете использовать приведенный только что синтаксис, но PHP предлагает еще дополнительную возможность – ключевое слово `elseif`, позволяющее немного упростить код. С его использованием предыдущий пример может быть переписан так:

```
if ($good) {
        print("Хорошо!");
}
elseif ($error) {
        print("Ошибка!");
}
else {
        print("Плохо...");
}
```

Для простой проверки истинности/ложности можно также использовать тернарный оператор (`?:`). Обычно он используется в ситуациях, когда нужно что-то вывести, если заданная переменная истинна. Рассмотрим код, написанный с помощью оператора `if/else`:

```
<td><?php if($active) { echo "да"; } else { echo "нет"; }
?></td>
```

Этот же код можно переписать с использованием тернарного оператора:

```
<td><?php echo $active ? "да" : "нет"; ?></td>
```

Сравним синтаксис обоих операторов:

```
if (выражение) { true_оператор } else { false_оператор }
(выражение) ? true_выражение : false_выражение
```

Основная разница в том, что условный оператор не совсем оператор. Это означает, что он используется в выражениях, а результат тернарного выражения - самостоятельное выражение. В предыдущем примере оператор `echo` внутри условия `if`, а в случае с тернарным оператором `echo` предшествует выражению.

Оператор switch

Оператор *switch* используется тогда, когда вам надо задать различные действия в зависимости от значений одной переменной. Например, переменная содержит имя пользователя, а вы хотите выполнить определенные действия для каждого пользователя. Вот вы и сравниваете значение переменной с именами, задавая для каждого имени свое поведение. В этом смысле оператор *switch* аналогичен серии операторов *if* с условиями на одну и ту же переменную.

Итак, оператору `switch` передается выражение. Он сравнивает его значение со всеми значениями, указанными в ключевых словах `case`. Если найдено совпадение, будут выполнены все операторы, начиная с первого блока `case`, соответствующему выражению, и до первого ключевого слова `break`. Если совпадения не найдены, будут выполнены операторы, заданные ключевым словом `default`.

Предположим, что у вас есть следующий код:

```
if ($name == 'ktatroe') {
      // сделать что-то
}
else if ($name == 'dawn') {
      // сделать что-то
}
else if ($name == 'petermac') {
      // сделать что-то
}
else if ($name == 'bobk') {
      // сделать что-то
}
```

Используя оператор `switch`, этот код можно переписать так:

```
switch($name) {
      case 'ktatroe':
            // сделать что-то
            break;
      case 'dawn':
            // сделать что-то
            break;
      case 'petermac':
            // сделать что-то
            break;
      case 'bobk':
            // сделать что-то
            break;
}
```

Альтернативный синтаксис будет выглядеть так:

```
switch($name):
  case 'ktatroe':
    // сделать что-то
    break;
  case 'dawn':
    // сделать что-то
    break;
  case 'petermac':
    // сделать что-то
    break;
```

```
  case 'bobk':
    // сделать что-то
    break;
endswitch;
```

Поскольку операторы выполняются, начиная с первой метки `case`, соответствующей значению выражения и до следующего ключевого слова `break`, вы можете комбинировать несколько меток `case`. В следующем примере "да" будет выведено, если `$name` содержит "`sylvie`" или "`bruno`":

```
switch ($name) {
      case 'sylvie':
      case 'bruno':
            print("да");
            break;
      default:
            print("нет");
            break;
}
```

Желательно как-то комментировать тот факт, что вы не указываете оператор `break`, иначе вы можете когда-то забыть его указать и в результате будет выполнен код, который не должен быть выполнен.

Вы можете указать любое число уровней для ключевого слова `break` (делается это путем указания числа после оператора). Таким образом, `break` может разрывать несколько уровней вложенных операторов `switch`.

Оператор match

Оператор *match* пришел на смену оператору switch-case. Конструкция switch-case все еще работает и поддерживается в PHP 8, поэтому не придется переписывать сценарии, написанные с ее использованием. Новый оператор *match* представляет собой более компактную форму оператора *switch*:

```
$v = 1;
echo match ($v) {
    0 => 'Foo',
    1 => 'Bar',
    2 => 'Baz',
};  // Bar
```

Оператор while

Цикл с предусловием

Самая простая форма цикла - это оператор `while`:

while (выражение) оператор

Если выражение истинно, оператор будет выполнен. Затем будет заново вычислено значение выражения, если оно истинно, оператор снова будет выполнен и т.д. Другими словами, цикл `while` будет повторять выполнение оператора, пока истинно указанное выражение.

Рассмотрим код, который складывает числа от 1 до 10:

```
$total = 0;
$i = 1;

while ($i <= 10) {
      $total += $i;
      $i++;
}
```

Альтернативный синтаксис следующий:

while (выражение):
оператор;
еще операторы ;
endwhile;

Например:

```
$total = 0;
$i = 1;

while ($i <= 10):
      $total += $i;
      $i++;
endwhile;
```

Преждевременный выход из цикла возможен с помощью ключевого слова `break`. В следующем коде **`$i`** никогда не достигнет значения 6, поскольку цикл будет остановлен, как только `$i` достигнет `5`:

```
$total = 0;
$i = 1;

while ($i <= 10) {
  if ($i == 5) {
      break;              // прерывает цикл
  }

  $total += $i;
  $i++;
}
```

Вы можете указать в операторе `break` число уровней, которое нужно прервать. Представим, что у нас есть два цикла `while` - внутренний и внешний, мы можем прервать выполнение внешнего цикла из внутреннего (вложенного) цикла. Например:

```
$i = 0;
$j = 0;

while ($i < 10) {
  while ($j < 10) {
      if ($j == 5) {
            break 2; // прерывает выполнение двух циклов
      }
      $j++;
  }
 $i++;
}

echo "{$i}, {$j}";
0, 5
```

Оператор `continue` позволяет пропустить итерацию цикла, но не прервать его работу, а перейти снова к условию цикла. Как и в случае с `break`, вы можете указать число уровней цикла:

```
while ($i < 10) {
  $i++;
  while ($j < 10) {
```

```
    if ($j == 5) {
      continue 2;  // затрагивает два уровня
    }
    $j++;
  }
}
```

В этом коде `$j` никогда не достигнет значения больше `5`, но `$i` пройдет по всем значениям - от `0` до `9`.

Цикл с постусловием

Язык PHP также предлагает цикл с постусловием `do/while`, синтаксис которого выглядит так:

do
оператор
while (выражение)

Используйте цикл `do/while`, когда тело цикла должно быть выполнено хотя бы один раз, ведь в этом цикле сначала выполняется тело цикла, а потом уже проверяется условие:

```
$total = 0;
$i = 1;

do {
      $total += $i++;
} while ($i <= 10);
```

В цикле `do/while` вы также можете использовать операторы `break` и `continue`, как в случае с обычным оператором `while`.

Цикл `do/while` иногда полезно прервать, например, при возникновении ошибки. Пример:

```
do {
      // проверяем условие ошибки
      if ($errorCondition) {
            break;
      }
```

```
// выполняем другие операторы
} while (false);
```

Поскольку условие для цикла - `false`, тело цикла будет выполнено только один раз. Однако, если произойдет какая-то ошибка (о чем будет свидетельствовать переменная `$errorCondition`), код после `break` не будет выполнен.

Цикл for. Цикл со счетчиком

Цикл `for` во многом подобен оператору `while`, но обладает дополнительными качествами, которые могут быть полезны: вы можете добавить инициализацию счетчика, выполнять тело цикла заданное количество раз, задать действие (или действия – тело цикла), которое будет выполнено в рамках каждой итерации цикла.

Для начала рассмотрим цикл `while`, выводящий числа от `0` до `9`:

```
$counter = 0;

while ($counter < 10) {
      echo "Счетчик = {$counter}\n";
      $counter++;
}
```

Теперь рассмотрим, как это можно сделать с помощью цикла `for`:

```
for ($counter = 0; $counter < 10; $counter++) {
      echo "Счетчик = $counter\n";
}
```

Структура оператора `for` следующая:

for (старт; условие; инкремент) { оператор(ы); }

Выражение *старт* вычисляется один раз - в самом начале выполнения оператора `for`. При каждой итерации проверяется *условие*, если оно истинно, выполняется тело цикла. Если оно ложно, цикл прекращает свою работу. Выражение *инкремент* вычисляется после каждой итерации цикла.

Альтернативный синтаксис:

```
for (expr1; expr2; expr3):
```

```
  statement;
  ...;
endfor;
```

Следующая программа складывает числа от 1 до 10, используя цикл `for`:

```
$total = 0;
for ($i= 1; $i <= 10; $i++) {
      $total += $i;
}
```

Рассмотрим альтернативный синтаксис этого же цикла:

```
$total = 0;
for ($i = 1; $i <= 10; $i++):
  $total += $i;
endfor;
```

Вы можете указать несколько выражений в операторе `for`, разделив их запятыми, например:

```
$total = 0;

for ($i = 0, $j = 1; $i <= 10; $i++, $j *= 2) {
      $total += $j;
}
```

Также вместо выражения вы можете вообще ничего не указывать. В самой дегенеративной форме оператор `for` превращается в бесконечный цикл. Так, следующий цикл не может быть остановлен, он будет выводить фразу бесконечно[1]:

```
for (;;) {
      echo "Меня не остановить!<br />";
}
```

Как и с циклом `while`, в цикле `for` вы можете использовать операторы `break` и `continue` как для завершения цикла, так и для завершения текущей итерации.

1 На самом деле не бесконечно, а пока не будет превышено максимальное время выполнения PHP-сценария, заданное в настройках PHP. Прим. переводчика

Оператор foreach

Оператор `foreach` удобно использовать для прохода по элементам массива. Далее будут рассматриваться две формы оператора `foreach` - когда вы уже познакомитесь с массивами. Пока же просто рассмотрим общий синтаксис `foreach`, позволяющий получить каждое значение массива:

```
foreach ($массив as $значение) {
        // ...
}
```

Альтернативный синтаксис следующий:

```
foreach ($массив as $значение):
        // ...
endforeach;
```

Чтобы получить каждый ключ (индекс) и каждое значение массива, используйте следующий синтаксис:

```
foreach ($массив as $ключ => $значение) {
        // ...
}
```

Альтернативный синтаксис выглядит так:

```
foreach ($массив as $ключ => $значение):
        // ...
endforeach;
```

Конструкция try...catch

Конструкция *try...catch* является не совсем структурой управления выполнением программы, она используется для обработки системных ошибок. Например, если вы хотите убедиться, что вы установили соединение с базой данных прежде, чем продолжить выполнение программы, вы можете написать следующий код:

```
try {
$dbhandle = new PDO('mysql:host=localhost; dbname=library',
$username, $pwd);
```

```
doDB_Work($dbhandle);    // вызываем функцию обработки
данных
$dbhandle = null;        // освобождаем дескриптор, когда он
уже не нужен
}
catch (PDOException $error) {
      print "Ошибка!: " . $error->getMessage() . "<br/>";
      die();
}
```

Здесь будет предпринята попытка подключения к базе данных и обработки данных в блоке `try`. Если произойдет какая-то ошибка, выполнение будет передано блоку `catch`. Создается объект `$error` класса `PDOException`. Мы используем этот объект для вывода сообщения об ошибке. Блок `catch` можно использовать не только для вывода ошибок, но и для попытки подключения к альтернативной базе данных, а также для любой другой обработки сложившейся ситуации.

Операторы exit и return

Оператор `exit` немедленно завершает выполнение сценария. Оператор `return` возвращает управление программой из функции в вызывавший модуль. При этом выполнение возвращается в выражение, следующее после вызова текущего модуля.

В случае с функциями, если `return` вызван из функции, то он немедленно прекращает выполнение текущей функции и возвращает свой аргумент как значение данной функции. Оператор `return` может завершить работу сценария, если вызван за пределами функции (т.е. из глобальной области видимости).

Оператор `exit` принимает необязательное значение, которое используется в качестве состояния завершения процесса. Если это число, то оно означает статус завершения процесса. Если это строка, то значение будет выведено перед завершением процесса. Функция `die()` является псевдонимом для этой формы оператора `exit`:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD);

if (!$db) {
      die("Не могу подключиться к БД");
}
```

Наиболее часто используется следующая конструкция:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD)
or die("Не могу подключиться к БД");
```

Оператор goto

Оператор `goto` позволяет "перепрыгнуть" в другое место программы. Определить точки выполнения можно с помощью меток. Метка - это идентификатор, после которого следует двоеточие. После определения меток вы можете переходить по ним в разные места вашей программы.

```
for ($i = 0; $i < $count; $i++) {
      // упс, найдена ошибка!
      if ($error) {
            goto cleanup;
      }
}

cleanup:
// выполняем очистку
```

Оператор `goto` вы можете использовать только для перехода в пределах одной области видимости, также вы не можете "перепрыгнуть" в тело цикла или оператора `switch`. Вообще говоря, использование `goto` является плохим стилем. Вы, конечно, можете использовать `goto`, однако старайтесь переписать ваш код без `goto`, что сделает код более понятным.

5.1.6. Включение кода

Для загрузки кода и HTML из другого модуля в ваш текущий PHP-сценарий используются две конструкции: `include` и `require`. Обе подгружают файл при запуске вашего PHP-сценария, работают в условных операторах и циклах и сообщают, если загружаемый файл не найден. Основная разница в том, что при попытке загрузить несуществующий файл с помощью `require` произойдет фатальная ошибка и выполнение сценария будет прервано. А в случае с `include` вы лишь получите предупреждение, но выполнение сценария не будет остановлено.

Часто include используют для разделения специфичного содержимого веб-страницы в рамках дизайна сайта. Так, общие для всего сайта элементы, например, заголовок и нижняя часть страницы, сохраняются в отдельные HTML-файлы, которые подгружаются при открытии той или иной страницы. При этом каждая страница будет выглядеть так:

```
<?php include "header.html"; ?>

содержимое страницы

<?php include "footer.html"; ?>
```

Здесь мы используем include, потому что PHP позволяет продолжить обработку страницы, даже если произойдет ошибка при загрузке файлов дизайна. Конструкция require менее прощающая и больше подходит для загрузки библиотек кода, без которых невозможно продолжение выполнения сценария. Например:

```
require "codelib.php";
mysub(); // определена в codelib.php
```

Чуть более эффективный способ в работе с верхними и нижними колонтитулами (заголовками) состоит в том, чтобы подгрузить один файл, а затем вызвать функции для создания стандартизированных элементов сайта:

```
<?php require "design.php";
header(); ?>

content

<?php footer();
```

Если PHP не может обработать часть PHP-кода, добавленного с помощью include или require файла, будет выведено предупреждение, но выполнение сценария будет продолжено. Вы можете подавить вывод сообщений об ошибках, добавив оператор @ перед включением файла, например, @include.

Если в файле конфигурации PHP (*php.ini*) включена опция allow_url_fopen, вы можете включать файлы, находящиеся на удаленных узлах, указав URL вместо пути к файлу, например:

```
include "http://www.example.com/codelib.php";
```

Если имя файла начинается с *http://* или *ftp://*, файл будет получен с удаленного узла.

Имена подключаемых файлов могут быть произвольными. Обычно используются расширения *.php*, *.php5* и *.html*. Обратите внимание, что при удаленном получении файла с расширением *.php* с сервера, на котором включен PHP, вы получите вывод PHP-сценария, а не сам PHP-сценарий.

Если программа включает один и тот же файл дважды (без разницы - с помощью `include` или `require`), например, когда вы ошибочно вызвали инструкцию включения файла в цикле, этот файл будет включен еще раз. Если вы включаете HTML-код, то этот код будет выведен дважды. Чтобы предотвратить повторное включение файла и связанные с этим ошибки, используйте конструкции `include_once` и `require_once`. При первом включении файла они ведут себя, как `include` и `require`, но они игнорируют последующие попытки загрузить один и тот же файл. Например, множеству элементов страницы, которые хранятся в отдельных файлах, нужно знать текущие предпочтения пользователя. Библиотека элементов должна загрузить библиотеку пользовательских предпочтений с помощью `require_once`. Дизайнер страниц затем может подключить элемент страницы, не беспокоясь о том, был ли код пользовательских предпочтений уже загружен. Код загружаемого файла импортируется в область, в которой был найден оператор `include`, поэтому включаемый код увидит и сможет модифицировать все переменные вашего кода. Это может быть полезным, например, библиотека отслеживания пользователя может сохранить имя текущего пользователя в глобальной переменной `$user`:

```
// главная страница
include "userprefs.php";
echo "Привет, {$user}.";
```

Возможность библиотек видеть и изменять ваши переменные может также быть проблемой. Вам нужно точно знать, какие глобальные переменные используются библиотекой, чтобы убедиться, что вы не используете переменную с таким же именем в собственных целях, что может перезаписать значение библиотеки и нарушить ее работу.

Если конструкция `include` или `require` находится в функции, переменные в подключаемом файле попадают в область этой функции.

Поскольку include и require являются ключевыми словами, а не реальными операторами, вы должны всегда заключать их в фигурные скобки в условном операторе или циклах:

```
for ($i = 0; $i < 10; $i++) {
      include "repeated_element.html";
}
```

Используйте функцию get_included_files(), чтобы узнать, какие файлы подключает с помощью include или require ваша программа. Функция возвращает массив, состоящий из полных имен подключаемых файлов. Не обработанные на момент вызова функции файлы не будут включены в этот массив.

5.1.7. Внедрение PHP в Web-страницы

Хотя допускается написание и запуск полностью автономных PHP-сценариев, часто PHP-код просто встраивается в HTML- или XML-файлы. В конце концов, для этого PHP и создавался.

При обработке таких документов каждый блок PHP-кода заменяется выводом, который он порождает.

Поскольку один файл может содержать блоки как PHP-кода, так и не-PHP кода, нам нужно как-то идентифицировать PHP-части кода. Для этого в PHP предусмотрено четыре разных способа. Далее в этом разделе мы их все разберем.

Как вы увидите, первый и стандартный способ состоит в использовании тегов наподобие того, как это делается в XML. Второй метод напоминает SGML. Третий метод основан на ASP-тегах. Четвертый метод подразумевает использование стандартного HTML-тега <script>, благодаря чему можно редактировать страницы с внедренным PHP-кодом в обычном HTML-редакторе.

Стандартный (XML) стиль

Из-за появления языка разметки XML (eXtensible Markup Language) и миграции языка HTML на язык XML (XHTML), данный способ является

предпочтительным для встраивания PHP-кода. Данный способ заключается в использовании XML-совместимых тегов для обозначения инструкций PHP.

Чтобы использовать этот стиль, заключите PHP-код в теги `<?php` и `?>`. Все, что находится между этими маркерами будет интерпретироваться как PHP, а все, что находится за их пределами не будет считаться PHP-кодом. Добавление пробелов между маркерами и вложенным в них текстом не является необходимым, но повышает читабельность.

Например, чтобы PHP вывел строку "Привет, мир!", в вашу веб-страницу нужно вставить следующую строку:

```
<?php echo "Привет, мир!"; ?>
```

Точка с запятой перед маркером `?>` является необязательной, поскольку конец блока порождает и конец выражения. Полный HTML-файл может выглядеть так:

```
<!doctype html>
<html>
  <head>
    <title>Это моя первая PHP-программа!</title>
  </head>
  <body>
    <p>
      Мама, посмотри! Это моя первая PHP-программа:<br />
      <?php echo "Привет, мир!"; ?><br />
      Понравилось?
    </p>
  </body>
</html>
```

Конечно, данный пример не очень впечатляющий, поскольку этого же эффекта можно добиться и без PHP. Настоящая ценность PHP проявляется, когда мы можем поместить на веб-страницу динамическую информацию, полученную из разных источников, например, из баз данных или форм. Однако давайте все же вернемся к нашему примеру "Привет, мир!". Когда пользователь посетит эту страницу и просмотрит ее исходный код, он увидит это:

```
<!doctype html>
<html>
  <head>
    <title>Это моя первая PHP-программа!</title>
  </head>
  <body>
    <p>
      Мама, посмотри! Это моя первая PHP-программа:<br />
      Привет, мир!<br />
      Понравилось?
    </p>
  </body>
</html>
```

Обратите внимание, что пользователь видит только вывод PHP-кода, но не видит сам код.

Инструкции PHP могут быть помещены куда угодно, даже во внутрь допустимых HTML-тегов. Например:

```
<input type="text" name="first_name" value="<?php echo
"Питер"; ?>" />
```

Когда PHP обработает этот текст, он станет таким:

```
<input type="text" name="first_name" value="Питер" />
```

Код PHP внутри маркеров не обязательно должен находиться на одной строке. При необходимости вы можете добавить сколько угодно разрывов строки. Поэтому мы можем переписать PHP-код в нашем примере "Привет, мир!" так:

```
<?php
echo "Привет, мир"; ?>
<br />
```

В результирующем HTML-коде ничего не изменится.

Стиль SGML

Другой стиль внедрения PHP-кода приходит из SGML-тегов. Чтобы использовать этот стиль, просто заключите PHP-код в маркеры `<?` и `?>`. Рассмотрим снова наш пример:

```
<? echo "Привет, мир!"; ?>
```

Данный стиль также известен как короткие теги, официально поддерживается, но по умолчанию выключен. Чтобы включить поддержку коротких тегов нужно или собрать PHP с опцией `--enable-short-tags` или включить директиву `short_open_tag` в файле конфигурации PHP.

Использование короткого `echo`-тега `<?= ... ?>` включено независимо от доступности коротких тегов.

ASP-стиль

Поскольку ни SGML-, ни XML-стиль по сути не являются HTML-кодом, некоторые HTML-редакторы не распознают код, оформленный под них. В результате не работает цветная подсветка синтаксиса, контекстная справка и другие полезные функции. А некоторые редакторы даже "помогают" вам, удаляя "недопустимый" код.

Однако многие такие HTML-редакторы распознают другой механизм (не более допустимый, чем PHP) внедрения кода - Microsoft Active Server Pages (ASP). Подобно PHP, ASP - это метод внедрения сценариев стороны сервера в ваши документы. Если ваш редактор ничего не подозревает о существовании PHP, попробуйте использовать теги стиля ASP для внедрения участков PHP-кода в ваши страницы. ASP-стиль похож на SGML-стиль, но вместо `?` используется знак `%`:

```
<% echo "Привет, мир!"; %>
```

По умолчанию теги в стиле ASP выключены. Чтобы их использовать, вам нужно собрать PHP с опцией `--enable-asp-tags` или же включить директиву `asp_tags` в вашем конфигурационном файле PHP.

Использование <script>

Последний метод разделения PHP и HTML-кода заключается в использовании тега <script>. Обычно этот тег используется для сценариев, выполняющихся на стороне клиента, например, для внедрения JavaScript-кода. Поскольку при попадании веб-страницы в браузер PHP-код будет обработан, а в браузер попадет уже результат его выполнения, вы можете использовать тег <script> для включения PHP-кода. Для использования этого метода просто укажите php в качестве языка:

```
<script language="php">
echo "Привет, мир!";
</script>
```

Этот метод очень полезен с HTML-редакторами, которые работают только с HTML-кодом и не поддерживают команды обработки XML.

Использование echo для вывода контента

Наверное, самая распространенная операция в PHP-приложениях - вывод данных на экран пользователя. В контексте веб-приложения это означает вывод HTML-документа, который превратится в веб-страницу при просмотре пользователем.

Чтобы упростить эту операцию, PHP предоставляет специальные версии SGML- и ASP-тегов, которые принимают значение внутри и вставляют его в HTML-страницу. Чтобы использовать эту функцию, используйте знак = в открытом теге. Пример:

```
<input type="text" name="first_name" value="<?= "Питер";
?>">
```

Если вы используете ASP-теги, то же самое можно проделать и с ними:

```
<p>Это число (<%= 2 + 2 %>)<br />
и это число (<% echo (2 + 2); %>)<br />
одинаковые.</p>
```

После обработки вы получите следующий HTML-код:

```
<p>Это число (4)<br />
и это число (4)<br />
одинаковые.</p>
```

5.2. Функции

Функция - поименованный блок кода, выполняющий определенную задачу, возможно, реагирующий на ряд значений, переданных ей в виде параметров и, возможно, возвращающий единственное значение. Функции позволяют экономить на времени компиляции - независимо от того, сколько раз вы вызываете их, функции компилируются только один раз для страницы. Также они повышают надежность, позволяя вам исправлять ошибки в одном месте, а не по всей программе. Еще функции улучшают удобочитаемость, изолируя код, который выполняет определенные задачи.

Далее вы познакомитесь с синтаксисом вызова и определения функций, а также рассмотрите, как управлять переменными в функциях и как передавать значения функциям (мы рассмотрим, как параметры-значения, так и параметры-ссылки).

Вызов функции

Функции в PHP программе могут быть встроенными или определенными пользователем. Независимо от их происхождения, все функции вызываются одним и тем же способом:

$некотороеЗначение = имя_функции([parameter, ...]);

Число параметров, которые необходимо передать функции, отличается от функции к функции (и, как вы увидите далее, может быть разным для одной и той же функции). Параметры, переданные функции, могут быть любым допустимым выражением и должны следовать в определенном, ожидаемом порядке. Если вы передали параметры в другом порядке, то функция будет работать "наугад" и в большинстве случаев ее результатом будет "мусор". Документация по функции подскажет вам, какие параметры нужно передать функции и какие значения вам следует от нее ожидать.

Рассмотрим несколько примеров вызова функций:

```
// strlen() - встроенная функция, возвращающая длину строки
$length = strlen("PHP"); // $length = 3

// sin() и asin() - математические функции синус и арксинус
```

```
$result = sin(asin(1)); // $result = синус арксинуса 1 или 1.0

// unlink() удаляет файл
$result = unlink("functions.txt"); // false, если неуспешно
```

В первом примере мы передаем аргумент "PHP" функции `strlen()`, которая в ответ передает нам число символов в данной строке. В нашем примере она вернет значение `3`, которое будет присвоено переменной `$length`. Это простейший и наиболее частый способ использования функции.

Во втором примере мы передаем результат функции `asin(1)` функции `sin()`. Поскольку функции синус и арксинус обратные, мы должны всегда получить то же значение. Здесь мы видим, что функция может быть вызвана внутри другой функции. Возвращаемое значение внутреннего вызова будет отправлено внешней функции в качестве аргумента, а затем общий результат будет записан в переменную `$result`.

В последнем примере мы передаем имя файла функции `unlink()`, которая пытается удалить файл. Подобно многих функциям, она возвращает `false` в случае неудачи. Это позволяет вам использовать другую встроенную функцию, `die()`.

Поэтому данный пример может быть переписан так:

```
$result = unlink("functions.txt") or die("Не получилось
удалить файл!");
```

Функция `unlink()`, в отличие от двух других примеров, производит действие с параметром, переданным ей, за пределами самой функции. В данном случае она удаляет файл из файловой системы.

В PHP есть огромное число функций, которые вы можете использовать в ваших программах: от доступа к базам данных до создания графики, от чтения и записи XML-файлов до получения файлов из удаленных систем.

Определение функции

Для определения функции используется следующий синтаксис:

function [&] имя_функции([parameter[, ...]])
{
список операторов
}

Список операторов может содержать HTML-код. Вы можете определить PHP-функцию, которая вообще не содержит PHP-код. Например, функция `column()` просто присваивает удобное короткое имя HTML-коду, который нужно вывести множество раз при формировании страницы:

```
<?php function column()
{ ?>
      </td><td>
<?php }
```

Имя функции может начинаться с буквы или знака подчеркивания, после чего может следовать (или не следовать) больше букв, знаков подчеркивания и цифр. Имена функций не чувствительны к регистру символов, поэтому, вы можете вызвать функцию `sin()` как `sin(1)`, `SiN(1)`, `SIN(1)` и т.д., поскольку все эти имена обращаются к одной и той же функции. По соглашению, все встроенные PHP-функции принято вызывать в нижнем регистре символов.

Обычно функции возвращают некоторые значения. Для возврата значения из функции используйте оператор `return`. Поместите оператор `return <выражение>` в вашу функцию. Когда оператор `return` будет встречен во время выполнения функции, выполнение будет передано вызывающему функцию оператору, а результат вычисления `<выражение>` будет возвращен, как значение функции. Внутри функции вы можете определить любое число операторов `return` (например, вы можете использовать оператор `switch` для определения, какое из нескольких значений нужно вернуть).

Давайте посмотрим на простую функцию (листинг 5.2). Она принимает две строки, выполняет их конкатенацию (соединение) и возвращает результат (в нашем случае мы создали более медленный вариант оператора конкатенации, но это только ради примера):

Листинг 5.2. Конкатенация строки

```
function strcat($left, $right)
{
      $combinedString = $left . $right;
      return $combinedString;
}
```

Функция принимает два параметра, `$left` и `$right`. Используя оператор конкатенации, функция создает объединенную строку в переменной `$combinedString`. Далее функция возвращает в качестве результата значение переменной **`$combinedString`**.

Поскольку оператор `return` может принимать любой тип выражений, даже самые сложные, мы можем упростить программу так:

```
function strcat($left, $right)
{
      return $left . $right;
}
```

Если мы поместим функцию в нашу PHP-страницу, мы сможем ее вызвать из любого места страницы. Давайте рассмотрим листинг 5.3.

Листинг 5.3. Использование нашей функции конкатенации

```
<?php
function strcat($left, $right)
{
return $left . $right;
}
$first = "Это - ";
$second = "полное предложение!";
echo strcat($first, $second);
```

При отображении страницы будет показано полное предложение.

В следующем примере функция принимает целое значение и дублирует его с помощью побитного сдвига и возвращает результат:

```
function doubler($value)
{
      return $value << 1;
}
```

Как только функция будет определена, ее можно будет использовать везде на странице, например:

```
<?= "Пара 13 =  " . doubler(13); ?>
```

Вы можете использовать вложенные объявления функции, но при этом необходимо учитывать некоторые ограничения. Вложенные объявления не ограничивают видимость внутренней функции, которая может быть вызвана в любом месте вашей программы. Внутренняя функция автоматически не получает параметры внешней функции. И, наконец, внутренняя функция не может быть вызвана, пока не была вызвана внешняя функция, также ее нельзя вызвать из кода, обработанного после внешней функции:

```
function outer ($a)
{
      function inner ($b)
      {
            echo "$b";
      }
echo "$a 2 ";
}
// выведет "1 2 3"
outer("1");
inner("3");
```

Область действия переменной

Если вы не используете функции, любая созданная вами переменная может использоваться в любом месте страницы. Когда у вас есть функции, это не всегда так. Функции хранят собственные наборы переменных, которые отличаются от тех, которые используются на странице и в других функциях.

Переменные, определенные в функции, включая ее параметры, недоступны за пределами функции и, по умолчанию, переменные, определенные за пределами функции, недоступны внутри функции. Мы уже говорили об этом ранее и следующий пример иллюстрирует это:

```
$a = 3;
function foo()
{
$a += 2;
}
```

```
foo();
echo $a;
```

Переменная `$a` объявлена внутри функции `foo()` отличается от переменной `$a`, объявленной за пределами функции. Даже если `foo()` использует оператор добавить и присвоить, значение внешней переменной `$a` останется `3` на протяжении всей жизни страницы. Внутри функции у переменной `$a` будет значение `2`.

Как было показано ранее, степень видимости переменной в программе, называют областью видимости переменной. Переменные, создаваемые в функции, видимы только внутри функции. Переменные, созданные за пределами функций и объектов, имеют глобальную область видимости и существуют везде за пределами этих функций и объектов. В PHP существует несколько предопределенных переменных, обладающих и областью видимости уровня функции, и глобальной областью (они часто называются суперглобальными переменными).

На первый взгляд, даже опытный программист может подумать, что в предыдущем примере `$a` будет равна `5` при достижении оператора `echo`, поэтому помните об этом, когда выбираете имена для ваших переменных.

Глобальные переменные

Если вы хотите использовать глобальные переменные в ваших функциях, вы можете использовать ключевое слово `global`. Синтаксис следующий:

```
global var1, var2, ...
```

Изменим предыдущий пример, добавив ключевое слово `global`:

```
$a = 3;
function foo()
{
      global $a;
      $a += 2;
}
foo();
echo $a;
```

Вместо создания новой переменной с именем `$a` и областью уровня функции, PHP использует глобальную переменную `$a` в этой функции. Теперь, когда новое значение `$a` отображено, мы увидим `5`.

Вы должны использовать ключевое слово `global` перед первым использованием глобальной переменной или глобальных переменных, к которым вы хотите получить доступ. Поскольку они определены до тела функции, параметры функции никогда не будут глобальными переменными.

Использование `global` эквивалентно созданию ссылки на переменную в переменной `$GLOBALS`. Другими словами оба следующих объявления создают переменную с областью видимости уровня функции, которая является ссылкой на то же значение глобальной переменной `$var`:

```
global $var;
$var = $GLOBALS['var'];
```

Статические переменные

Подобно C, язык PHP поддерживает статические переменные функций. Значение статической переменной сохраняется между всеми вызовами функции, а ее инициализация происходит только при первом вызове функции. Короче говоря, все вызовы функции работают только с одной и той же областью памяти, в которой хранится значение статической переменной (в случае с обычной переменной каждый раз под нее выделяется новая память, то есть переменная инициализируется при каждом вызове функции).

Для объявления статической переменной используется ключевое слово `static`. Обычно при первом использовании статической переменной ей присваивается начальное значение:

```
static var [= value][, ... ];
```

В лист. 5.4 переменная `$count` увеличивается при каждом вызове функции.

Листинг 5.4. Статическая переменная-счетчик

```
<?php
function counter()
{
      static $count = 0;
      return $count++;
}
for ($i = 1; $i <= 5; $i++) {
      print counter();
}
```

Когда функция вызывается в первый раз, переменной `$count` присваивается значение `0`. Это значение возвращается, и переменная `$count` увеличивается. Когда функция завершает свою работу, переменная `$count` не уничтожается подобно нестатическим переменным, а ее значение остается таким же до следующего вызова переменной `counter()`. Цикл `for` отображает числа от 0 до 4.

Параметры функции

Благодаря объявлению аргументов в определении функции, функция может ожидать определенное (или неопределенное) число аргументов (параметров), требуемых ей для работы. Существует два разных способа передачи параметров функции. Первый, наиболее часто используемый, передача параметра по значению (такие параметры называются параметрами-значениями). Второй - по ссылке (эти параметры называются параметрами-ссылками).

Передача параметров-значений

В большинстве случаев вы передаете параметры по значению. В качестве аргумента может быть любое допустимое выражение. Это выражение вычисляется, а его результат передается соответствующей переменной в функции. Во всех примерах ранее мы передавали аргументы по значению.

Передача параметров-ссылок

Передача по ссылке позволяет вам обойти обычные правила видимости переменных путем непосредственного предоставления функции прямого доступа к переменной. Чтобы быть переданным по ссылке, аргумент должен быть переменной (а не выражением). Вы сообщаете функции, что передаете параметр-ссылку, указывая амперсанд (`&`) перед именем переменной. В примере 5.2. представлена измененная функция `doubler()`, написанная с использованием параметров-ссылок.

Пример 5.2. Измененная версия функции `doubler()`

```
<?php
function doubler(&$value)
{
      $value = $value << 1;
```

```
}
$a = 3;
doubler($a);
echo $a;
```

Поскольку параметр `$value` функции передан как ссылка на фактическое значение переменной `$a` вместо копирования этого значения, переменная `$a` будет модифицирована функцией. Ранее нам нужно было возвратить удвоенное значение, но теперь мы просто модифицируем значение переменной (обратите внимание, оператора `return` уже нет). Поскольку мы передали переменную `$a` по ссылке, она находится в полной власти функции. В нашем случае `doubler()` просто присваивает ей новое значение.

В качестве параметров-ссылок могут выступать только переменные, а не константы или выражения. Если мы будем использовать оператор `<?= doubler(7); ?>` из предыдущего примера, мы получим ошибку. Однако, вы можете назначить стандартные значения параметрам, переданных по ссылке (так же, как и стандартные значения для параметров-значений).

Даже в случаях, когда вы не планируете изменять переданное значение, вы все равно можете использовать параметры-ссылки. Когда вы используете параметры-значения, PHP копирует значения параметров. Для больших строк и объектов это может быть очень "дорогой" в плане использования системных ресурсов операцией. Передача параметров по ссылке устраняет необходимость копирования значения.

Параметры по умолчанию

Допустим, ваша функция при вызове должна принять определенный параметр. Например, вы вызываете функцию для получения предпочтений для сайта, и этой функции должны передать параметр с именем "предпочтения". Теперь допустим, что вы хотите предусмотреть возможность получения всех предпочтений. Вместо определения какого-то специального ключевого слова, означающего, что вы хотите получить все предпочтения, вы можете просто не передавать аргументы функции. Такое решение возможно, если вы установите значение аргумента функции по умолчанию. В нашем примере этим значением и будет то ключевое слово, которое будет обрабатываться функцией в случае неполучения другого значения аргумента.

Чтобы определить значение по умолчанию, присвойте параметру значение при объявлении функции. Значение по умолчанию не может быть сложным выражением, оно может быть только скалярным значением:

```
function getPreferences($whichPreference = 'all')
{
// если $whichPreference = "all", будут возвращены
// все предпочтения;
// в противном случае будет возвращены указанные
// предпочтения ...
}
```

Когда вы вызываете `getPreferences()`, вы можете указать параметр. Если вы сделаете это, то получите предпочтение, соответствующее переданной строке. В противном случае будут возвращены все предпочтения.

У функции может быть любое число параметров со значениями по умолчанию. Однако они должны быть перечислены после всех параметров, у которых нет значений по умолчанию.

Переменное число параметров

Функция может содержать переменное число параметров. Например, функция `getPreferences()` из предыдущего размера может возвращать предпочтения для любого числа имен, а не только для одного. Чтобы объявить функцию с переменным числом параметров, просто не указывайте параметры при определении функции:

```
function getPreferences()
{
      // некоторый код
}
```

PHP предоставляет три функции, которые вы можете использовать в функции для получения параметров, переданных ей. Функция `func_get_args()` возвращает массив всех переданных параметров функции. Функция `func_num_args()` возвращает количество параметров, переданных функции. Функция `func_get_arg` возвращает параметр с определенным номером. Например:

```
$array = func_get_args();
$count = func_num_args();
$value = func_get_arg(номер_аргумента);
```

В примере 5.3 функция `count_list()` принимает любое число аргументов. В цикле вычисляется сумма всех переданных значений. Если параметры не заданы, возвращается `false`.

Пример 5.3. Сумма всех аргументов

```
<?php
function countList()
{
      if (func_num_args() == 0) {
      return false;
      }
else {
      $count = 0;
      for ($i = 0; $i < func_num_args(); $i++) {
            $count += func_get_arg($i);
      }
      return $count;
      }
}
echo countList(1, 5, 9); // выведет "15"
```

Результат любой из этих функций не может быть использован непосредственно как параметр другой функции. Вместо этого вы должны сначала передать результат в переменную, а затем уже использовать ее при вызове функции. Следующее выражение не будет работать:

```
foo(func_num_args());
```

Вместо этого используйте следующий код:

```
$count = func_num_args();
foo($count);
```

Отсутствующие параметры

PHP позволяет вам быть достаточно ленивым - когда вы вызываете функцию, вы можете передать любое число аргументов функции. Параметры, ожидаемые функцией, но не переданные вами, останутся неустановленными, и для каждого из них будет выведено предупреждение.

```
function takesTwo($a, $b)
{
      if (isset($a)) {
            echo " a установлен\n";
      }
      if (isset($b)) {
            echo " b установлен\n";
      }
}

echo "Вызываем функцию с двумя аргументами:\n";
takesTwo(1, 2);
echo "Вызываем функцию с одним аргументом:\n";
takesTwo(1);
```

Вывод:

```
Вызываем функцию с двумя аргументами:
a установлен
b установлен
Вызываем функцию с одним аргументом:

Warning: Missing argument 2 for takes_two()
in /path/to/script.php on line 6

a установлен
```

Контроль типа

При определении функции вы можете потребовать, чтобы передаваемый ей параметр был определенного типа, то есть экземпляром определенного класса (включая экземпляры классов, которые расширяют этот класс), или экземпляром класса, реализующего определенный интерфейс, или масси-

вом, или колбеком с типом `callable`. Для добавления контроля типа к параметру, перед названием переменной в списке параметров функции установите имя класса, служебное слово `array` или `callable`. Например:

```
class Entertainment {}

class Clown extends Entertainment {}

class Job {}

function handleEntertainment(Entertainment $a, callable
$callback = NULL)
{
      echo "Обработка " . get_class($a) . " успешна\n";

      if ($callback !== NULL) {
            $callback();
      }
}

$callback = function()
{
// делаем что-то
};

handleEntertainment(new Clown); // работает
handleEntertainment(new Job, $callback); // ошибка времени выполнения
```

Параметр с контролем типа должен быть или `NULL`, или экземпляром указанного класса `Entertainment`, экземпляром подкласса `Clown` (расширения класса `Entertainment`), массивом или типом `callable`. В противном случае будет ошибка времени выполнения.

Контроль типа не работает со скалярными типами, то есть нельзя установить требования о том, чтобы параметры был, скажем, типа `interger` или `string`.

Возвращаемые значения

Функции в PHP могут возвращать только одно значение и делают это с помощью ключевого слова `return`:

```
function returnOne()
```

```
{
      return 42;
}
```

Чтобы вернуть несколько значений, просто в качестве возвращаемого значения используйте массив (`array`):

```
function returnTwo()
{
      return array("Фред", 35);
}
```

Если вы не указали оператор `return`, функция вернет значение `NULL`.

По умолчанию, значения копируются за пределы функции. Для возврата значения по ссылке, объявляют функцию с амперсандом (`&`) перед ее именем и при присвоении возвращаемого значения переменной:

```
$names = array("Фред", "Барни", "Вильма", "Бетти");

function &findOne($n) {
      global $names;
      return $names[$n];

}
$person =& findOne(1);    // Барни
$person = "Вася";         // изменяет $names[1]
```

В этом коде функция `findOne()` возвращает ссылку на `$names[1]` вместо копии его значения. Поскольку мы присваиваем по ссылке, `$person` является ссылкой для `$names[1]` и второе присваивание изменяет значение в `$names[1]`.

Эта техника иногда используется для возвращения больших строк или массивов. Однако помните, что возврат ссылки на значение медленнее, чем возврат самого значения.

Переменные функции. Обращение к функциям через переменные

PHP поддерживает переменные функции. Это означает, что если к имени переменной присоединены круглые скобки, PHP ищет функцию с тем же именем, что и результат вычисления переменной и пытается ее выполнить. Рассмотрим ситуацию, где переменная используется для определения, какую из трех функций нужно запустить:

```
switch ($which) {
  case 'first':
            first();
            break;
  case 'second':
            second();
            break;
  case 'third':
            third();
            break;
}
```

В этом случае мы можем использовать переменные функции для вызова надлежащей функции. Чтобы вызвать переменную функцию, добавьте после имени переменной круглые скобки. Мы можем перезаписать предыдущий пример так:

```
$which();  // если $which = "first", будет вызвана функция
first() и т.д.
```

Если для переменной не существует функции, будет сгенерирована ошибка времени выполнении при запуске кода. Чтобы предотвратить это, вы можете использовать функцию `function_exists()` для определения существования функции перед ее вызовом:

```
$yesOrNo = function_exists(function_name);
```

Например:

```
if (function_exists($which)) {
```

```
    $which();   // если $which = "first", будет вызвана
функция first() и т.д.

}
```

Конструкции языка вроде `echo()` и `isset()` не могут быть вызваны таким образом:

```
$which = "echo";
$which("hello, world"); // не работает
```

5.3. Строки

Большинство данных, с которыми вы встречаетесь при написании программ, являются последовательностями символов или строками. Строки содержат имена людей, пароли, адреса, номера кредитных карточек, истории покупки и многое другое. По этой причине PHP содержит широкий выбор функций для работы со строками.

В этой книге строки уже рассматривались. Сейчас же мы рассмотрим две очень важных темы – кодирование и экранирование строк, а также сравнение строк. Как показывает практика, иногда результаты сравнения бывают неожиданными.

5.3.1. Кодирование и экранирование

Поскольку PHP-программы часто взаимодействуют с HTML-страницами, веб-адресами (URL), базами данными, сейчас мы рассмотрим функции, которые помогут вам в работе с этими типами данных. HTML, адреса веб-страниц и команды базы данных - это строки, но каждая из них требует своего способа экранирования (итогового вывода) символов. Например, пробелы в URL должны быть записаны как `%20`, а знак меньше (<) в HTML-документе должен быть заменен как `<`. В PHP предусмотрено множество встроенных функций для преобразования между этими кодировками.

В формат HTML

Специальные символы в HTML представляются *сущностями*, например, `&` и `<`. В PHP есть две функции, которые позволяют преобразовать специальные символы строки в их HTML-сущности: одна для удаления HTML-тегов, а другая только для извлечения мета-тегов.

Экранирование всех специальных символов

Функция `htmlentities()` заменяет все специальные символы (за исключением пробелов) на HTML-сущности. В результате будут заменены знаки меньше (<), больше (>), амперсанд (&) и другие символы, для которых существуют HTML-сущности.

Например:

```
$string = htmlentities("Einstürzende Neubauten");
echo $string;
```

Вывод:

```
Einstürzende Neubauten
```

В результате символ ü будет заменен на HTML-сущность `&uuml;` и будет корректно отображен при просмотре веб-страницы. Как вы можете видеть, пробел не был преобразован в ` `.

Функция `htmlentities()` на самом деле принимает три параметра:

```
$output = htmlentities(input, quote_style, charset);
```

Необязательный параметр *charset* задает кодировку. По умолчанию используется кодировка ISO-8859-1. Параметр *quote_style* определяет, будут ли двойные или одинарные кавычки преобразованы в HTML-сущности. По умолчанию используется значение параметра `ENT_COMPAT`, при этом конвертируются только двойные кавычки, при значении `ENT_QUOTES` оба типа кавычек будут преобразованы, а если указать `ENT_NOQUOTES` кавычки будут оставлены как есть. Нет опции, конвертирующей только одинарные кавычки. Например:

```
// текст с двойными и одинарными кавычками
$input = <<< End
"Stop pulling my hair!" Jane's eyes flashed.<p>
End;

// преобразуем двойные кавычки
$double = htmlentities($input);
// &quot;Stop pulling my hair!&quot; Jane's eyes
flashed.&lt;p&gt;

// и двойные, и одинарные
$both = htmlentities($input, ENT_QUOTES);
// &quot;Stop pulling my hair!&quot; Jane&#039;s eyes
flashed.&lt;p&gt;

// оставляем кавычки как есть
$neither = htmlentities($input, ENT_NOQUOTES);
// "Stop pulling my hair!" Jane's eyes flashed.&lt;p&gt;
```

Экранирование только символов синтаксиса HTML

Функция `htmlspecialchars()` конвертирует наименьший набор сущностей, необходимых для генерации корректного HTML-кода. Будут конвертированы следующие сущности:

- Амперсанд (`&`) преобразуется в `&`
- Двойная кавычка (`"`) преобразуется в `&quot;`
- Одинарная кавычка (`'`) преобразуется в `&#039;` (только в режиме `ENT_QUOTES`).
- Знак "меньше чем" (<) преобразуется в `<`
- Знак "больше чем" (>) преобразуется в `>`'

Если у вас есть приложение, отображающее данные, введенные пользователем в форме, вам нужно пропустить их через `htmlspecialchars()` перед отображением или сохранением. Если вы это не сделаете, а пользователь введет строку вроде "`угол < 30`", браузер будет считать специальные символы HTML-кодом и в результате отобразит искаженную страницу.

Как и `htmlentities()`, функция `htmlspecialchars()` может принимать три аргумента:

```
$output = htmlspecialchars(input, [quote_style, [charset]]);
```

Параметры *quote_style* и *charset* имеют то же значение, что и для `htmlentities()`.

Не существует функций для обратного преобразования сущностей в оригинальный текст, поскольку это редко когда нужно. Однако существует относительно простой способ сделать это. Используйте функцию `get_html_translation_table()` для получения таблицы перевода, используемой этими функциями в заданном стиле кавычек. Например, получить таблицу перевода, которую использует `htmlentities()`, можно так:

```
$table = get_html_translation_table(HTML_ENTITIES);
```

Получить таблицу перевода, которую использует `htmlspecialchars()` в режиме `ENT_NOQUOTES`, можно так:

```
$table = get_html_translation_table(HTML_SPECIALCHARS, ENT_
NOQUOTES);
```

После этого мы можем зеркально отразить таблицу перевода (поменять местами ключи и значения), используя array_flip(), а затем выполнить обратное преобразование:

```
$str = htmlentities("Einstürzende Neubauten"); // закодировано
$table = get_html_translation_table(HTML_ENTITIES);
$revTrans = array_flip($table);
echo strtr($str, $revTrans); // обратно в обычный текст
```

Вывод:

```
Einstürzende Neubauten
```

Конечно, вы можете также получить таблицу перевода, добавить в нее другие преобразования, которые вы хотите осуществить, а затем вызвать `strstr()`. Например, если вы хотите, чтобы `htmlentities()` также кодировала пробелы в ` `, сделайте следующее:

```
$table = get_html_translation_table(HTML_ENTITIES);
$table[' '] = ' ';
$encoded = strtr($original, $table);
```

Удаление HTML-тегов

Функция `strip_tags()` удаляет HTML-теги из строки:

```
$input = '<p>Привет, &quot;ковбой&quot;</p>';
$output = strip_tags($input);
// $output = 'Привет, &quot;ковбой&quot;'
```

Функция может принимать второй аргумент, указывающий, какие теги нужно оставить в строке. В списке нужно указать только открывающиеся теги. Закрывающиеся теги указывать не нужно.

```
$input = 'В строке останутся <b>жирные</b> <i>теги</i><p>';
$output = strip_tags($input, '<b>');
// $output = 'В строке останутся <b>жирные</b> теги'
```

Атрибуты сохраненных тегов не изменяются функцией `strip_tags()`. Поскольку атрибуты (например, `style` и `onmouseover`) могут влиять на внешний вид и поведение веб-страницы, сохранение некоторых тегов с помощью `strip_tags()` является потенциально опасной затеей.

Извлечение META-тегов

Функция `get_meta_tags()` возвращает массив META-тегов HTML-страницы, указанный в виде URL или локального файла. Имена META-тегов (`keywords`, `author`, `description` и т.д.) становятся ключами в массиве, а содержимое META-тега становится соответствующим значением:

```
$metaTags = get_meta_tags('http://www.example.com/');
echo "Разработчик страницы {$metaTags['author']}";
```

Вывод:

```
Разработчик страницы: Иван Иванов
```

Синтаксис этой функции следующий:

```
$array = get_meta_tags(filename [, use_include_path]);
```

Передайте `true` в качестве *use_include_path*, чтобы позволить PHP открывать файл, используя стандартный include-путь.

Конвертирование в URL

В языке PHP предусмотрены функции для конвертирования в URL-кодировку и обратно, что позволяет вам создавать и декодировать URL. Есть два типа URL-кодирования, которые отличаются способом обработки пробелов. Первый способ (определен в RFC 3986) заменяет пробелы в URL последовательностью символов `%20`. Второй способ (реализован системой `application/x-www-form-urlencoded`) заменяет пробелы символом + и используется при построении строк запросов.

Заметьте, что вам не нужно использовать эти функции на полном URL, таком как *http://www.example.com/hello*, поскольку они экранируют двоеточия и слеши, и результат будет таким:

```
http%3A%2F%2Fwww.example.com%2Fhello
```

Вам нужно кодировать только часть URL (все, что после http://www.example.com/), а затем добавить имя протокола и домена.

Кодирование и декодирование по RFC 3986

Для кодирования строки согласно URL-соглашению RFC 3986 используйте функцию `rawurlencode()`:

```
$output = rawurlencode(input);
```

Функция принимает строку и возвращает ее копию, в которой недопустимые для применения в URL символы закодированы в формате `%dd`.

Если вы динамически генерируете гипертекст для ссылок на странице, вам нужно конвертировать их с помощью `rawurlencode()`:

```
$name = "Programming PHP";
$output = rawurlencode($name);
echo "http://localhost/{$output}";
```

Вывод:

```
http://localhost/Programming%20PHP
```

Обратное преобразование можно выполнить функцией `rawurldecode()`:

```
$encoded = 'Programming%20PHP';
echo rawurldecode($encoded);
```

Вывод:

```
Programming PHP
```

Кодирование строки параметров

Функции `urlencode()` и `urldecode()` отличаются от их raw-версий только тем, что они заменяют пробелы знаком плюс (+), а не последовательностью `%20`. Этот формат используется для построения строки параметров и значений Cookies. Интерпретатор PHP автоматически декодирует строки параметров и значения Cookie, поэтому вам не нужно использовать эти функции для обработки данных значений.

Функции полезны для создания строки запроса:

```
$baseUrl = 'http://www.google.com/q=';
$query = 'PHP sessions -cookies';
$url = $baseUrl . urlencode($query);
echo $url;
```

Вывод:

```
http://www.google.com/q=PHP+sessions+-cookies
```

В формат SQL

Большинство систем управления базами данных требуют, чтобы строковые литералы в ваших SQL-запросах были экранированы. Схема кодирования предельно проста - одинарные кавычки, двойные кавычки, NUL-байты и обратные слеши должны экранироваться с помощью обратного слеша. Функция `addslashes()` добавляет эти слеши, а `stripslashes()` - удаляет их:

```
$string = <<< EOF
"Это никогда не будет работать," закричала она,
а затем нажала клавишу (\).
EOF;
$string = addslashes($string);
echo $string;
echo stripslashes($string);
```

Вывод:

```
\"Это никогда не будет работать\" закричала она,
а затем нажала клавишу (\\) key.
"Это никогда не будет работать" закричала она,
а затем нажала клавишу (\).
```

Примечание. Некоторые базы данных (например, Sybase) требует, чтобы одинарные кавычки были экранированы другими одинарными кавычками вместо обратного слеша. Для этих баз данных включите `magic_quotes_sybase` **в вашем файле php.ini.**

Кодирование С-строк

Функция `addcslashes()` экранирует строку слешами в стиле языка С. Все символы, за исключением представленных в таблице 5.10 и с ASCII-кодами от 32 до 126 будут преобразованы в восьмеричное представление (например, `"\002"`). Функции `addcslashes()` и `stripcslashes()` используются с нестандартными системами баз данных, у которых есть собственное представление об экранировании символов.

Таблица 5.10. Символы-исключения для `addcslashes()` и `stripcslashes()`

ASCII-значение	Кодирование
7	\a
8	\b
9	\t
10	\t
11	\v
12	\f
13	\r

Функция `addcslashes()` вызывается с двумя аргументами - строка, которую нужно закодировать и экранируемые символы:

```
$escaped = addcslashes(string, charset);
```

Укажите диапазон символов, используя конструкцию "..":

```
echo addcslashes("hello\tworld\n", "\x00..\x1fz..\xff");
```

Вывод:

```
hello\tworld\n
```

Будьте внимательны при экранировании символов `0`, `a`, `b`, `f`, `n`, `r`, `t` и `v`. Они будут преобразованы в `\0`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t` и `\v`. В языке C все они являются предопределенными escape-последовательностями, а в PHP это может послужить причиной путаницы.

Функция `stripcslashes()` принимает строку и удаляет экранирование, произведенное функцией `addcslashes()`:

```
$string = stripcslashes(escaped);
```

Например:

```
$string = stripcslashes('привет\tмир\n');
// $string = "привет\tмир\n"
```

5.3.2. Сравнение строк

В PHP есть два оператора и шесть функций для сравнения строк.

Точные сравнения. Операторы == и ===

Операторы `==` и `===` позволяют определить, являются ли две строки эквивалентными. Разница между ними заключается в обработке операндов, которые не являются строками. Оператор `==` преобразует операнды в строки, поэтому он сообщит, что число `3` равно строке `"3"`. Оператор `===` вернет `false`, если типы данных аргументов отличаются:

```
$o1 = 3;
$o2 = "3";
if ($o1 == $o2) {
```

```
      echo("== вернул true<br>");
}
if ($o1 === $o2) {
      echo("=== вернул true<br>");
}
```

Вывод:

```
      == вернул true
```

Операторы сравнения (<, <=, >, >=) также работают со строками:

```
$him = "Антон";
$her = "Юлия";

if ($him < $her) {
      print "{$him} лексикографически находится перед
{$her}\n";
}
```

Вывод:

```
      Антон лексикографически находится перед Юлия
```

Однако при сравнении строк и чисел результаты могут быть неожиданными:

```
$string = "PHP";
$number = 5;

if ($string < $number) {
      echo("{$string} < {$number}");
}
```

Вывод:

```
      PHP < 5
```

Когда один аргумент оператора сравнения - число, другой аргумент сводится к числу. При сведении к числу строки "PHP" она превратится в 0. Поскольку 0 меньше 5, будет выведена строка "PHP < 5".

Чтобы явно сравнить две строки, используйте функцию `strcmp()`:

```
$relationship = strcmp(string_1, string_2);
```

Функция возвращает *-1*, если строка *string_1* лексикографически меньше строки *string_2* (то есть строка *string_1* при сортировке в алфавитном порядке находится перед строкой *string_2*). Значение *1* возвращается, если *string_2* лексикографически находится перед *string_1*, а *0* возвращает, если строки одинаковые:

```
$n = strcmp("PHP Rocks", 5);
echo($n);
```

Вывод:

```
1
```

Функция `strcasecmp()` является вариацией `strcmp()`. Разница в том, что `strcasecmp()` конвертирует строки в нижний регистр перед сравнением. Аргументы и возвращаемые значения такие же, как и для `strcmp()`:

```
$n = strcasecmp("Федор", "феДОр");  // $n = 0
```

Другая модификация `strcmp()` предназначена для сравнения первых нескольких символов строки. Функция `strncasecmp()` принимает дополнительный аргумент - число символов, которые будут использоваться для сравнения:

```
$relationship = strncmp(string_1, string_2, len);
$relationship = strncasecmp(string_1, string_2, len);
```

Есть еще две функции, использующиеся для сравнения строк в *натуральном порядке* – функции `strnatcmp()` и `strnatcasecmp()`, которые принимают те же аргументы, что и `strcmp()` и возвращают те же значения. При сортировке в натуральном порядке числовые части сортируются отдельно от строчных частей. Таблица 5.11 показывает разницу между натуральным порядком и ASCII-порядком.

Таблица 5.11. Натуральный порядок vs ASCII-порядка

Натуральный порядок	ASCII-порядок
pic1.jpg	pic1.jpg
pic5.jpg	pic10.jpg
pic10.jpg	pic5.jpg
pic50.jpg	pic50.jpg

Приблизительное сравнение

PHP предоставляет несколько функций, позволяющих вам выяснить, являются ли строки приблизительно равными: `soundex()`, `metaphone()`, `similar_text()`, and `levenshtein()`:

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($string_1, $string_2 [, $percentage
]);
$similarity = levenshtein($string_1, $string_2);
$similarity = levenshtein($string_1, $string_2 [, $cost_ins,
$cost_rep, $cost_del ]);
```

Алгоритмы Soundex и Metaphone возвращают одинаковые значение для слов, имеющих сходное произношение на английском языке. Алгоритм Metaphone более точек, что демонстрирует следующий пример:

```
$known = "Fred";
$query = "Phred";

if (soundex($known) == soundex($query)) {
      print "soundex: {$known} звучит как {$query}<br>";
}
else {
      print "soundex: {$known} не звучит как {$query}<br>";
}
if (metaphone($known) == metaphone($query)) {
      print "metaphone: {$known} звучит как {$query}<br>";
}
else {
      print "metaphone: {$known} не звучит как {$query}<br>";
}
```

Вывод:

```
soundex: Fred не звучит как Phred
metaphone: Fred звучит как Phred
```

Функция `similar_text()` вычисляет степень похожести двух строк, возвращает число одинаковых символов, которые есть в двух переданных строках. В третий параметр, если он задан, заносится степень похожести двух строк, выраженная в процентах:

```
$string1 = "Rasmus Lerdorf";
$string2 = "Razmus Lehrdorf";
$common = similar_text($string1, $string2, $percent);
printf("У переданных строк %d общих символов (%.2f%%).",
$common, $percent);
```

Вывод:

```
У переданных строк 13 общих символов (89.66%)
```

Алгоритм Левенштейна вычисляет подобность двух строк на основании того, сколько символов нужно добавить, заменить или удалить, чтобы сделать строки одинаковыми. Например, у слов "cat" и "cot" расстояние Левенштейна равно 1, поскольку нужно заменить всего один символ ("a" на "o"), чтобы сделать их одинаковыми:

```
$similarity = levenshtein("cat", "cot"); // $similarity = 1
```

Данная мера подобия вычисляется быстрее, чем используемая функцией `similar_text()`. Вы можете передать функции `levenshetin()` три параметра, чтобы вычислить вес вставок, удалений и замен:

```
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```

5.4. Массивы

Как было упомянуто ранее, язык PHP поддерживает и скалярные, и составные типы данных. Мы рассмотрим один из составных типов: массивы.

Массив - это упорядоченный набор данных, представленных в виде пар ключ-значение.

Чтобы представить, что такое массив, подумайте о нем, как о прямоугольном лотке для яиц с несколькими рядами. Каждый отсек лотка может содержать яйцо, но мы покупаем, перемещаем весь лоток сразу. Лоток для яиц может содержать не только яйца. В его отсеки можно положить болты, гайки, камешки и множество других мелких предметов. Таким образом, массив не ограничен одним типом данных. Он может содержать строки, целые числа, логические переменные и т.д. К тому же элементы массива могут, в свою очередь, также являться массивами, но об этом мы поговорим позже.

Мы поговорим о создании массива, добавлении и удалении элементов из массива, а также о переборе содержимого массива. Поскольку массивы очень и очень полезны, множество встроенных функций PHP работает с ними. Например, если вы хотите отправить электронное письмо более чем одному адресату, вы можете хранить адреса e-mail в массиве и в цикле итерировать по нему, отправляя сообщение по текущему адресу. Также у вас может быть форма, в которой пользователь может выбрать несколько элементов и выбранные пользователем элементы будут переданы вам в виде массива.

5.4.1. Индексированные и ассоциативные массивы

В PHP существует два типа массивов: индексированные и ассоциативные. Ключи индексированного массива – целые числа, начинающиеся с 0. Индексируемые массивы используются, когда вы идентифицируете вещи непосредственно по их позиции (индексу). Ассоциативные массивы в качестве ключей используют строки и ведут себя больше как таблицы на два столбца. Первый столбец - это ключ, который используется для доступа к значению требуемого элемента массива (второй столбец).

Во внутреннем представлении PHP хранит все массивы как ассоциативные. Единственная разница между ассоциативными и индексированными

массивами – то, чем является ключ. Некоторые функции обработки массивов предназначены только для работы с индексированными массивами, поскольку они предполагают, что вы работаете с ключами, которые являются последовательными целыми числами, начинающимися с 0. Тем не менее, в обоих случаях ключи уникальны. Другими словами, у вас не может быть двух элементов с тем же ключом, независимо от того, является ли ключ строкой или целым числом.

У массивов PHP есть внутренний порядок, который независим от ключей, значений и функций, которые вы можете использовать для обхода массива. Обход массива (последовательный перебор элементов массива) основан на этом внутреннем порядке. Обычно элементы массива хранятся в том порядке, в котором они были вставлены в массив, однако функции сортировки, описанные далее, позволяют вам изменять порядок на основе ключей, значений или чего-либо еще.

5.4.2. Идентификация элементов массива

Перед тем, как перейти к созданию массивов давайте рассмотрим структуру уже существующего массива. Вы можете получить доступ к определенным значениям существующего массива, используя имя переменной массива, за которым следует ключ (или индекс), заключенный в квадратные скобки:

```
$age['fred']
$shows[2]
```

Ключи могут быть или строкой или целым числом. Строковые значения, эквивалентные целым числам (без ведущих нулей) считаются целыми значениями. Поэтому `$array[3]` и `$array['3']` ссылаются на один и тот же элемент. Но `$array['03']` ссылается уже на другой элемент. Отрицательные числа тоже допустимы, но они указывают позицию, начиная с конца массива, как в Perl.

Строки, состоящие из одного слова, вы можете не заключать в кавычки. Например, `$age['fred']` аналогично `$age[fred]`. Однако в PHP считается хорошим стилем всегда использовать кавычки, потому что ключи без кавычек неотличимы от констант. Когда вы используете константы как незаключенный в кавычки индекс, PHP использует значение констант в качестве индекса и выводит предупреждение:

```
define('index', 5);
echo $array[index];      // получает $array[5], а не
$array['index'];
```

Вы должны использовать кавычки, если вы используете интерполяцию для построения индекса массива:

```
$age["Clone{$number}"]
```

В то же время, иногда вы должны заключить ключ в кавычки, чтобы убедиться, что получаете то значение, которое ожидаете:

```
// это неправильно
print "Hello, {$person['name']}";
print "Hello, {$person["name"]}";
```

5.4.3. Хранение данных в массивах

Сохранение значения в массиве создаст массив, если он еще не существовал, однако попытка получить значение массива, который не был определен, не приведет к созданию массива. Например:

```
// До этого $addresses не был определен
echo $addresses[0];       // ничего не выведет
echo $addresses;    // ничего не выведет
$addresses[0] = "spam@cyberpromo.net";
echo $addresses;    // выведет "Array"
```

Инициализировать массив в вашей программе можно с помощью простого присваивания:

```
$addresses[0] = "spam@cyberpromo.net";
$addresses[1] = "abuse@example.com";
$addresses[2] = "root@example.com";
```

Это индексируемый массив, с целочисленными индексами, начинающимися с 0. А теперь рассмотрим ассоциативный массив:

```
$price['прокладка'] = 15.29;
$price['диск'] = 75.25;
$price['шина'] = 50.00;
```

Простейший способ инициализировать массив - использовать конструкцию `array()`, которая строит массив из своих аргументов. Конструкция строит индексируемый массив, где значения индекса (начиная с 0) создаются автоматически:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com",
"root@example.com");
```

С помощью `array()` можно также создать и ассоциативный массив, используя символ => для разделения ключей (индексов) и значений:

```
$price = array(
      'прокладка' => 15.29,
      'диск' => 75.25,
      'шина' => 50.00
);
```

Обратите внимание на использование пробелов и выравнивание. Мы сгруппировали код, благодаря чему его будет проще читать, в него будет проще добавить новые значения и удалить ненужные элементы, однако он полностью эквивалентен следующему примеру:

```
$price = array('прокладка' => 15.29, 'диск' => 75.25,
'шина' => 50.00);
```

Вы также можете определить массив, используя более короткий, альтернативный, синтаксис (с использованием квадратных скобок):

```
$days = ['gasket' => 15.29, 'wheel' => 75.25, 'tire' =>
50.0];
```

Чтобы создать пустой массив, вызовите `array()` без параметров:

```
$addresses = array();
```

Вы можете указать начальный ключ, а затем список значений. Значения будут вставлены в массив, начиная с указанного ключа, индексы следующим элементам будут присвоены последовательно:

```
$days = array(1 => "Пн", "Вт", "Ср", "Чт", "Пт", "Сб",
"Вс");
// 2 = Вт, 3 = Ср и т.д.
```

Если начальный индекс – нечисловая строка, индексы всех последующих элементов будут целыми числами, причем нумерация будет начинаться с **0**:

```
$whoops = array('Пт' => "Черный", "Коричневый", "Зеленый");
// аналогично
$whoops = array('Пт' => "Черный", 0 => "Коричневый", 1 =>
"Зеленый");
```

5.4.4. Добавление значений в конец массива

Для вставки значений в конец существующего массива, используйте синтаксис `[]`:

```
$family = array("Фред", "Вильма");
$family[] = "Павел";          // $family[2] = "Павел"
```

Эта конструкция подразумевает, что индексами массива являются числа и добавляет элементы в массив со следующего числового индекса, начиная с **0**. Попытка использования данной конструкции для добавления элемента в ассоциативный массив без указания соответствующего ключа почти всегда является ошибкой программиста, при этом PHP просто создаст для элементов числовые индексы без всякого предупреждения:

```
$person = array('name' => "Фред");
$person[] = "Вильма"; // $person[0] теперь - "Вильма"
```

5.4.5. Присваивание диапазона значений

Функция `range()` создает массив последовательных чисел или символов между двумя заданными значениями, которые вы передадите ей в качестве аргументов. Например:

```
$numbers = range(2, 5);           // $numbers = array(2, 3, 4, 5);
$letters = range('a', 'z');       // $letters содержит англ. алфавит
$reversedNumbers = range(5, 2);  // $reversedNumbers = array(5, 4, 3, 2);
```

При этом для построения диапазона используется только первая буква строкового аргумента:

```
range("aaa", "zzz"); // то же, что и('a','z')
```

5.4.6. Получение размера массива

Использование и результаты функций `count()` и `sizeof()` идентичны. Они возвращают число элементов в массиве. Между этими функциями нет никакой разницы, равно как и каких-либо стилистических предпочтений, какую из функций использовать. Пример:

```
$family = array("Федор", "Виталий", "Павел");
$size = count($family);                    // $size = 3
```

Эта функция подсчитывает только, сколько *значений* фактически есть в массиве:

```
$confusion = array( 10 => "десять", 11 => "одиннадцать", 12 =>
"двенадцать");
$size = count($confusion); // $size = 3
```

5.4.7. Заполнение массива

Для создания массива и его инициализации одним и тем же значением, используйте `array_pad()`. Первый параметр этой функции – массив, второй – минимальное число элементов, которое вы хотите добавить в массив, а третий параметр – это значение созданных аргументов. Функция `array_pad()` возвращает новый заполненный массив, оставляя исходный массив (источник) нетронутым.

Рассмотрим `array_pad()` в действии:

```
$scores = array(5, 10);
$padded = array_pad($scores, 5, 0); // $padded = array(5, 10, 0, 0, 0)
```

Обратите внимание, что новые значения добавляются в конец массива. Если вы хотите, чтобы новые элементы были добавлены в начало массива, используйте отрицательный второй аргумент:

```
$padded = array_pad($scores, -5, 0); // $padded = array(0, 0, 0, 5, 10);
```

Если вы дополняете ассоциативный массив, существующие ключи будут сохранены, а новым элементам будут добавлены числовые ключи, начиная с 0.

5.4.8. Многомерные массивы

Значения массива сами могут быть массивами. В PHP вы можете легко создавать многомерные массивы:

```
$row0 = array(1, 2, 3);
$row1 = array(4, 5, 6);
$row2 = array(7, 8, 9);
$multi = array($row0, $row1, $row2);
```

Вы можете обращаться к элементам многомерного массива, используя несколько наборов квадратных скобок `[]`:

```
$value = $multi[2][0];    // колонка 2, столбец 0. $value = 7
```

Для интерполяции многомерного массива, вы должны заключить весь массив в фигурные скобки:

```
echo("Значение в строке 2, столбце 0 = {$multi[2][0]}\n");
```

Ошибка при использовании фигурных скобок приведет к примерно такому выводу:

```
Значение в строке 2, столбце 0 = Array[0]
```

5.4.9. Извлечение нескольких значений

Чтобы скопировать все элементы массива в переменные, используйте конструкцию `list()`:

```
list ($variable, ...) = $array;
```

Значения массива копируются в перечисленные переменные во внутреннем порядке массива. По умолчанию это порядок, в котором были добавлены элементы, но, используя функции сортировки, вы можете изменить данный порядок. Рассмотрим пример:

```
$person = array("Фред", 35, "Бетти");
list($name, $age, $wife) = $person;
// $name = "Фред", $age = 35, $wife = "Бетти"
```

Примечание. Использование функции `list()` - установившаяся практика для получения значений из базы данных, где возвращается один рядок. Это автоматически загрузить данные, полученные из простого запроса, в серию локальных переменных. Рассмотрим небольшой пример по выборке данных о двух противоборствующих командах и спортивной базы данных:

```
$sql = "SELECT HomeTeam, AwayTeam FROM schedule WHERE Ident =
7";
$result = mysql_query($sql);
list($hometeam, $awayteam) = mysql_fetch_assoc($result);
```

Если в массиве есть больше значений, чем вы указали в `list()`, оставшиеся значения будут проигнорированы:

```
$person = array("Фред", 35, "Бетти");
list($name, $age) = $person; // $name = "Фред", $age = 35
```

Если, наоборот, в массиве меньше значений, чем указано в `list()`, дополнительные переменные будут установлены в `NULL`:

```
$values = array("hello", "world");
list($a, $b, $c) = $values; // $a = "hello", $b = "world", $c = NULL
```

Две или более последовательных запятых в `list()` позволяют пропустить значения в массиве:

```
$values = range('a', 'e'); // используем rand() для заполнения массива
list($m, , $n, , $o) = $values; // $m = "a", $n = "c", $o = "e"
```

5.4.10. "Вырезка" из массива

Для извлечения некоторого подмножества из массива, используйте функцию `array_slice()`:

```
$subset = array_slice(array, offset, length);
```

Функция `array_slice()` возвращает новый массив, состоящий из последовательных элементов исходного массива. Параметр *offset* задает начальный элемент, который будет скопирован (**0** представляет первый элемент массива), а *length* – определяет число элементов, которое должно быть скопировано. У нового массива будут последовательные числовые индексы, начинающиеся с **0**. Например:

```
$people = array("Том", "Дик", "Хэри", "Бренда", "Джо");
$middle = array_slice($people, 2, 2); // $middle = array("Хэри", "Бренда");
```

Как правило, функцию `array_slice()` имеет смысл использовать на индексируемых массивах (то есть на тех, индексы которых являются последовательными целыми числами, начинающимися с **0**):

```
// здесь использовать array_slice() нет смысла
$person = array('name' => "Фред", 'age' => 35, 'wife' => "Бетти");
$subset = array_slice($person, 1, 2); // $subset = array(0 => 35, 1 => "Бетти")
```

Комбинируйте `array_slice()` с `list()` для извлечения некоторых значений в переменные:

```
$order = array("Том", "Дик", "Харриет", "Бренда", "Джо");
list($second, $third) = array_slice($order, 1, 2);
// $second = "Дик", $third = "Харриет"
```

5.4.11. Разделение массива на несколько массивов

Чтобы разделить массив на несколько меньших массивов, используйте функцию `array_chunk()`:

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

Функция возвращает массив, состоящий из меньших массивов. Третий аргумент, *preserve_keys*, является логическим значением и определяет, нужно ли, чтобы элементы новых массивов имели такие же ключи, как и исходный массив (что полезно для ассоциативных массивов) или нужно использовать новые числовые значения, которые будут начинаться с **0** (полезно для индексируемых массивов). По умолчанию будут назначены новые ключи, как показано ниже:

```
$nums = range(1, 7);
$rows = array_chunk($nums, 3);
print_r($rows);
```

Результат:

```
Array (
  [0] => Array (
      [0] => 1
      [1] => 2
      [2] => 3
)
  [1] => Array (
      [0] => 4
      [1] => 5
      [2] => 6
)
  [2] => Array (
      [0] => 7
)
)
```

5.4.12. Ключи и значения

Функция `array_keys()` возвращает массив, состоящий только из ключей, следующих во внутреннем порядке массива:

```
$arrayOfKeys = array_keys(array);
```

Пример:

```
$person = array('name' => "Фред", 'age' => 35, 'wife' =>
"Вильма");
$keys = array_keys($person); // $keys = array("name", "age", "wife")
```

Также PHP предоставляет функцию для получения значений массива - `array_values()`:

```
$arrayOfValues = array_values(array);
```

Как и с функцией `array_keys()`, значения возвращаются во внутреннем порядке массива:

```
$values = array_values($person); // $values = array("Фред", 35, "Вильма");
```

5.4.13. Проверка существования элемента массива

Чтобы узнать, существует ли элемент в массиве, можно использовать функцию `array_key_exists()`:

```
if (array_key_exists(key, array)) { ... }
```

Функция возвращает логическое значение, показывающее есть элемент с ключом, заданным первым параметром `key`, в массиве, заданным вторым параметром `array`.

Недостаточно просто сказать:

```
if ($person['name']) { ... } // это может вводить в заблуждение
```

Даже если в массиве есть элемент с ключом `'name'`, соответствующее ему значение может быть `false` (то есть `0`, `NULL` или пустая строка). Вместо этого используйте `array_key_exists()`, как показано ниже:

```
$person['age'] = 0; // не рожден?
if ($person['age']) {
echo "true!\n";
}
if (array_key_exists('age', $person)) {
echo "существует!\n";
}
```

Вывод:

```
Существует!
```

Вместо этого многие используют функцию `isset()`, которая возвращает `true`, если элемент существует и не равен `NULL`:

```
$a = array(0, NULL, '');

function tf($v)
{
      return $v ? 'T' : 'F';
}

for ($i=0; $i < 4; $i++) {
      printf("%d: %s %s\n", $i, tf(isset($a[$i])),
tf(array_key_exists($i, $a)));
}
```

Вывод:

```
0: T T
1: F T
2: T T
3: F F
```

5.4.14. Удаление и вставка элементов в массив

Функция `array_splice()` может удалить или вставить элементы в массив и, опционально, создать другой массив из удаленных элементов:

```
$removed = array_splice(array, start [, length [,
replacement ] ]);
```

Давайте посмотрим на массив:

```
$subjects = array("физика", "химия", "математика",
"биология", "информатика", "драма", "классика");
```

Мы можем удалить элементы `"математика"`, `"биология"` и `"информатика"`, вызвав `array_splice()` так, чтобы она начала работу с элемента 2 и удалила 3 элемента:

```
$removed = array_splice($subjects, 2, 3);
// $removed = array("математика", "биология",
"информатика")
```

```
// $subjects = array("физика", "химия", "драма",
"классика")
```

Если опустить длину, `array_splice()` удалит элементы до конца массива:

```
$removed = array_splice($subjects, 2);
// $removed is array("математика", "биология",
// "информатика", "драма", "классика")
// $subjects is array("физика", "химия")
```

Если вы просто хотите удалить элементы из исходного массива и не хотите сохранять их значения, вам не нужно сохранять результаты `array_splice()`:

```
array_splice($subjects, 2);
// $subjects is array("физика", "химия");
```

Для вставки элементов на место удаленных, используйте четвертый аргумент:

```
$new = array("право", "финансы", "география");
array_splice($subjects, 4, 3, $new);
// $subjects = array("физика", "химия", "математика",
// "биология", "право", "финансы", "география")
```

Размер заменяемого массива не должен быть равным числу удаляемых элементов. Массив может "сужаться" или "расширяться" при необходимости:

```
$new = array("право", "финансы", "география");
array_splice($subjects, 3, 4, $new);
// $subjects is array("физика", "химия", "математика",
// "право", "финансы", "география")
```

Для вставки новых элементов в массив с продвижением существующих элементов вправо, удалите 0 элементов:

```
$subjects = array("физика", "химия", "математика');
$new = array("право", "финансы");
array_splice($subjects, 2, 0, $new);
// $subjects = array("физика", "химия", "право", "финансы",
// "математика")
```

Хотя все наши примеры для простоты были приведены на основе индексируемых массивов, `array_splice()` также работает и ассоциативными массивами:

```
$capitals = array(
      'США' => "Вашингтон",
      'Великобритания' => "Лондом",
      'Новая Зеландия' => "Веллингтон",
      'Австралия' => "Канберра",
      'Италия' => "Рим"
      'Канада' => "Оттава"
);
$downUnder = array_splice($capitals, 2, 2); // удаляет Нов. Зел. и Австр.
$france = array('Франция' => "Париж");

array_splice($capitals, 1, 0, $france); // вставляет
                                        // Францию между США и ВБ
```

5.4.15. Преобразование между массивами и переменными

PHP предоставляет две функции `extract()` и `compact()`, которые выполняют преобразование массива в переменные и обратно. Имена переменных соответствуют ключам в массиве, а значения переменных становятся значениями в массиве. Рассмотрим этот пример:

```
$person = array('name' => "Фред", 'age' => 35, 'wife' =>
"Вильма");
```

Он может быть преобразован в следующие переменные:

```
$name = "Фред";
$age = 35;
$wife = "Вильма";
```

5.4.16. Создание переменных из массива

Функция `extract()` автоматически создает локальные переменные из массива. Индексы элементов массива становятся именами переменных:

```
extract($person); // $name, $age и $wife теперь будут установлены
```

Если при извлечении массива окажется, что создаваемая переменная уже существует, то ее значение будет перезаписано извлекаемым из массива значением.

Вы можете изменить поведение `extract()` путем передачи второго параметра. В приложении вы найдете описание возможных значений для этого второго аргумента. Наиболее полезное значение - это `EXTR_PREFIX_ALL`, которая указывает, что для всех извлекаемых переменных будет создан префикс. Это гарантирует уникальность созданных имен переменных при использовании `extract()`. Рекомендуется всегда использовать `EXTR_PREFIX_ALL`, как показано здесь:

```
$shape = "round";
$array = array('cover' => "bird", 'shape' => "rectangular");

extract($array, EXTR_PREFIX_ALL, "book");
echo "Cover: {$book_cover}, Book Shape: {$book_shape}, Shape:
{$shape}";
```

Вывод:

```
Cover: bird, Book Shape: rectangular, Shape: round
```

5.4.17. Создание массива из переменных

Функция `compact()` – обратная функция для `extract()`. Передайте ей имена переменных, которые будут помещены в массив. Функция `compact()` создает ассоциативный массив, где в качестве ключей будут выступать имена переменных, а в качестве значений элементов - значения переданных переменных. Любые неустановленные строки будут просто пропущены.

Рассмотрим пример `compact()` в действии:

```
$color = "индиго";
$shape = "кривая";
$floppy = "нет";
$a = compact("color", "shape", "floppy");

// или
$names = array("color", "shape", "floppy");
$a = compact($names);
```

5.4.18. Обход массивов

В большинстве задач с массивами нужно что-то сделать с каждым элементом, например, отправить *e-mail* каждому элементу массива адресов, обновить каждый файл в массиве имен файлов, или сложить каждый элемент массива цен. Существует несколько способов обхода массивов в PHP и вы можете выбрать нужный, основываясь на ваших данных и на решаемой задаче.

Конструкция foreach

Наиболее частая практика обхода всех элементов массива – использование конструкции `foreach`:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");

foreach ($addresses as $value) {
      echo "Обработка {$value}\n";
}
```

Результат:

```
Обработка spam@cyberpromo.net
Обработка abuse@example.com
```

PHP выполняет тело цикла (оператор `echo`) один раз для каждого элемента массива `$addresses`, при этом переменная `$value` содержит значение текущего элемента. Элементы обрабатываются во внутреннем порядке массива.

Альтернативная форма `foreach` предоставляет доступ к текущему ключу:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' =>
"Wilma");
```

```
foreach ($person as $key => $value) {
      echo "Fred's {$key} is {$value}\n";
}
```

Вывод:

```
Fred's name is Fred
Fred's age is 35
Fred's wife is Wilma
```

В этом случае ключ для каждого элемента помещается в переменную `$key`, а соответствующее ему значение помещается в `$value`.

Конструкция `foreach` работает не с самим массивом, а с его копией. Вы можете безопасно вставить или удалить элементы в теле цикла `foreach`, зная, что цикл не будет пытаться вставить или удалить элементы исходного массива.

Функции-итераторы

Каждый PHP-массив отслеживает текущий элемент, с которым вы работаете. Указатель на текущий элемент называется итератором. В PHP есть функции для установки, перемещения и сброса итератора:

- `current()` – возвращает элемент, на который в данный момент указывает итератор;
- `reset()` – перемещает итератор на первый элемент в массиве и возвращает его;
- `next()` – перемещает итератор на следующий элемент в массиве и возвращает его;
- `prev()` – перемещает итератор на предыдущий элемент в массиве и возвращает его;
- `end()` – перемещает итератор на последний элемент массива и возвращает его;
- `each()` – возвращает ключ и значение текущего элемента в виде массива и перемещает итератор на следующий элемент в массиве;
- `key()` – возвращает ключ текущего элемента.

Функция `each()` используется для цикла по элементам массива. Она обрабатывает элементы в соответствии с их внутренним порядком:

```
reset($addresses);

while (list($key, $value) = each($addresses)) {
      echo "{$key} = {$value}<br />\n";
}
```

Вывод:

```
0 = spam@cyberpromo.net
1 = abuse@example.com
```

В данном случае не создается копия массива, как в случае с `foreach`. Это полезно для обработки очень больших массивов для экономии памяти.

Функции-итераторы полезны, когда вы нуждаетесь рассмотреть некоторые части массива отдельно от других. Пример 5.5 демонстрирует код, который строит таблицу, обрабатывая первый индекс и значение ассоциативного массива как заголовок таблицы.

Пример 5.5. Построение таблицы с помощью функций итераторов

```
$ages = array(
      'Имя' => "Возраст",
      'Фред' => 35,
      'Барни' => 30,
      'Тигр' => 8,
      'Пух' => 40

);

// начинает таблицу и печатает заголовок
reset($ages);

list($c1, $c2) = each($ages);
echo("<table>\n<tr><th>{$c1}</th><th>{$c2}</th></tr>\n");

// выводит оставшиеся значения
```

```
while (list($c1, $c2) = each($ages)) {
echo("<tr><td>{$c1}</td><td>{$c2}</td></tr>\n");
}

// завершает таблицу
echo("</table>");
```

Использование цикла for

Если вы знаете, что имеете дело с индексируемым массивом, где ключи – последовательные целые числа, начинающиеся с 0, вы можете использовать цикл `for` для прохода по индексам массива. Оператор `for` оперирует непосредственно с массивом, а не с его копией (как это делает `foreach`) и обрабатывает элементы во внутреннем порядке следования.

Рассмотрим, как вывести массив, используя цикл `for`:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");

$addressCount = count($addresses);

for ($i = 0; $i < $addressCount; $i++) {
      $value = $addresses[$i];
      echo "{$value}\n";
}
```

Вывод:

```
spam@cyberpromo.net
abuse@example.com
```

5.4.19. Вызов функции для каждого элемента массива

PHP предоставляет механизм, функцию `array_walk()` для вызова пользовательской функции для каждого элемента массива:

```
array_walk(array, callable);
```

Функция, которую вы определяете, принимает два или, опционально, три аргумента: первый – это значение элемента, второй – ключ элемента, а третий – будет передан в качестве третьего параметра в функцию обратного вызова. Рассмотрим другой способ вывода столбцов таблицы, созданной из значений массива:

```
$callback = function printRow($value, $key)
{
      print("<tr><td>{$value}</td><td>{$key}</td></tr>\n");
};

$person = array('name' => "Фред", 'age' => 35, 'wife' =>
"Вильма");

array_walk($person, $callback);
```

Изменим этот пример так, чтобы определять цвет фона, используя дополнительный третий параметр `array_walk ()`. Этот параметр предоставляет нам гибкость, необходимую для вывода множества таблиц с разными фоновыми цветами:

```
function printRow($value, $key, $color)
{
      echo "<tr>\n<td bgcolor=\"{$color}\">{$value}</td>";
echo "<td bgcolor=\"{$color}\">{$key}</td>\n</tr>\n");
}

$person = array('name' => "Фред", 'age' => 35, 'wife' =>
"Вильма");

echo "<table border=\"1\">";

array_walk($person, "printRow", "lightblue");

echo "</table>";
```

Если у вас есть несколько параметров, которые вы хотите передать в вызываемую функцию, просто передайте массив в качестве третьего параметра.

```
$extraData = array('border' => 2, 'color' => "красный");
```

```
$baseArray = array("Ford", "Chrysler", "Volkswagen",
"Honda", "Toyota");

array_walk($baseArray, "walkFunction", $extraData);

function walkFunction($item, $index, $data)
{
echo "{$item} <- элемент, затем граница: {$data['border']}";
echo " цвет->{$data['color']}<br />" ;
}
```

Результат:

```
Ford <- элемент, затем граница: 2 цвет->красный
Crysler <- элемент, затем граница: 2 цвет->красный
VW <- элемент, затем граница: 2 цвет->красный
Honda <- элемент, затем граница: 2 цвет->красный
Toyota <- элемент, затем граница: 2 цвет->красный
```

Функция `array_walk()` обрабатывает элементы в их внутреннем порядке.

5.4.20. Сокращение массива

Функция `array_reduce()` (дальний родственник `array_walk()`) применяет пользовательскую функцию к каждому элементу массива, чтобы в итоге свести массив к единственному значению:

```
$result = array_reduce(array, callable [, default ]);
```

Функция принимает два аргумента: общее значение и текущее обрабатываемое значение. Функция должна вернуть новое общее значение. Например, чтобы вычислить сумму квадратов значений массива, используется следующий код:

```
$callback = function addItUp($runningTotal, $currentValue)
{
      $runningTotal += $currentValue * $currentValue;
      return $runningTotal;
};

$numbers = array(2, 3, 5, 7);
```

```
$total = array_reduce($numbers, $callback);

echo $total;
```

Результат:

```
87
```

Функция `array_reduce()` делает следующие вызовы функций:

```
addItUp(0, 2);
addItUp(4, 3);
addItUp(13, 5);
addItUp(38, 7);
```

Аргумент *default*, если он предоставлен, задает начальное общее значение. Например, если мы изменим вызов `array_reduce()` в предыдущем примере на:

```
$total = array_reduce($numbers, "addItUp", 11);
```

То получим следующие вызовы функций:

```
addItUp(11, 2);
addItUp(15, 3);
addItUp(24, 5);
addItUp(49, 7);
```

Если массив пуст, `array_reduce()` возвращает значение по умолчанию. Если значение по умолчанию не указано и массив пуст, `array_reduce()` возвращает `NULL`.

5.4.21. Поиск значений

Функция `in_array()` возвращает `true` или `false`, в зависимости является ли первый аргумент элементом массива, заданного во втором аргументе:

```
if (in_array(to_find, array [, strict])) { ... }
```

Если задан третий аргумент и он равен `true`, типы данных аргумента *to_find* и найденного значения в массиве должны совпадать. По умолчанию проверка типов данных не производится.

Рассмотрим простой пример:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com",
"root@example.com");
$gotSpam = in_array("spam@cyberpromo.net", $addresses);
// $gotSpam = true
$gotMilk = in_array("milk@tucows.com", $addresses);
// $gotMilk= false
```

PHP автоматически индексирует значения в массиве, поэтому `in_array()` гораздо быстрее, чем проверка каждого значения в цикле, чтобы найти то, что вам нужно.

Пример 5.6 проверяет, ввел ли пользователь информацию во всех обязательных поля формы.

Пример 5.6. Поиск в массиве

```
<?php
function hasRequired($array, $requiredFields)
{
        $keys = array_keys($array);

        foreach ($requiredFields as $fieldName) {
                if (!in_array($fieldName, $keys)) {
                        return false;
                }
                if (strlen($array[$fieldName])===0)
                        return false;
        }

        return true;
}

if ($_POST['submitted']) {
        echo "<p>Вы ";
        echo hasRequired($_POST, array('name', 'email_address')) ? "" : "не";
        echo " заполнили все обязательные поля формы.</p>";
        }
?>
```

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<p>
       Имя: <input type="text" name="name" /><br />
       Email address: <input type="text" name="email_address" /><br />
       Возраст (optional): <input type="text" name="age" /></p>
<p align="center"><input type="submit" value="Передать
name="submitted" /></p>
</form>
```

Вариантом функции `in_array()` является функция `array_search()`. Если `in_array()` возвращает `true`, если значение найдено, `array_search()` возвращает ключ элемента, если он найден:

```
$person = array('имя' => "Фред", 'возраст' => 35, 'жена' =>
"Вильма");

$k = array_search("Вильма", $person);

echo("{$k} Фреда - Вильма\n");
```

Вывод:

```
жена Фреда - Вильма
```

Функция `array_search()` также принимает необязательный третий аргумент, который требует, чтобы типы искомого значения и значения в массиве соответствовали.

5.4.22. Сортировка

Сортировка изменяет внутренний порядок элементов в массиве и опционально перезаписывает ключи для отображения этого нового порядка. Например, вы можете использовать сортировку для упорядочивания списка баллов от большего к меньшему, вывода в алфавитном порядке списка имен, или для вывода списка пользователей на основании того, кто, сколько сообщений отправил.

Язык PHP предоставляет три способа сортировки массивов – по ключам, по значениям без изменения ключей или по значениям с изменением ключей. Каждый вид сортировки может быть выполнен в порядке возрастания, убывания или в порядке, определенном пользовательской функцией.

Сортировка одного массива за один раз

Функции, предоставленные PHP для сортировки массива, показаны в таблице 5.12.

Таблица 5.12. Функции PHP для сортировки массивов

Эффект	По возрастанию	По убыванию	Пользовательская сортировка
Сортирует по значениям, а затем заново назначает индексы, начиная с 0	sort()	rsort()	usort()
Сортирует массив по значениям	asort()	arsort()	uasort()
Сортирует массив по ключам	ksort()	krsort()	uksort()

Функции `sort()`, `rsort()` и `usort()` предназначены для работы с индексируемыми массивами, поскольку они присваивают новые числовые ключи и таким образом реализуют сортировку. Они полезны, когда вы должны ответить на вопросы вроде: "Какие 10 самых высоких баллов?", "Кто третий по алфавитному порядку" и т.д. Другие функции сортировки могут быть использованы на индексируемых массивах, но получить доступ к отсортированному порядку вы сможете, только используя функции обхода, например, `foraech` и `next`.

Для сортировки имен по алфавиту, выполните следующие действия:

```
$names = array("Катя", "Анжела", "Вика", "Марина");
sort($names); // $names = array("Анжела", "Вика", "Катя", "Марина")
```

Чтобы получить список имен в обратном алфавитном порядке, просто используйте `rsort()` вместо `sort()`.

Если у вас есть ассоциативный массив, содержащий имена пользователей и время сессии каждого из них, вы можете использовать функцию `arsort()` для отображения трех пользователей, находящихся на сайте больше всего времени:

```
$logins = array(
  'njt' => 415,
  'kt' => 492,
  'rl' => 652,
  'jht' => 441,
  'jj' => 441,
  'wt' => 402,
  'hut' => 309,
);

arsort($logins);

$numPrinted = 0;

echo "<table>\n";

foreach ($logins as $user => $time) {
      echo("<tr><td>{$user}</td><td>{$time}</td></tr>\n");

      if (++$numPrinted == 3) {
                  break; // выводим только 3 пользователя
      }
}
echo "</table>";
```

Если вы хотите отобразить таблицу в убывающем порядке по имени пользователя, используйте функцию `ksort()`.

Пользовательская сортировка требует от вас предоставления функции, которая принимает два значения и возвращает значение, которое определяет порядок двух переданных значений в отсортированном массиве. Функция должна вернуть `1`, если первое значение больше, чем второе, `-1`, если первое значение меньше, чем второе и `0`, если значения равны с точки зрения вашей функции сортировки.

Натуральный порядок сортировки

Встроенные функции PHP корректно сортируют строки и числа, но не корректно работают со строками, содержащими числа. Например, если у вас есть имена файлов *ex10.php*, *ex5.php* и *ex1.php*, обычные функции сортировки переупорядочат их в следующем порядке: *ex1.php*, *ex10.php* и *ex5.php*. Для корректной сортировки строк, содержащих числа, используйте функции `natsort()` и `natcasesort()`:

```
$output = natsort(input);
$output = natcasesort(input);
```

Сортировка нескольких массивов за один раз

Функция `array_multisort()` сортирует несколько массивов за один раз:

```
array_multisort(array1 [, array2, ... ]);
```

Передайте ей серию массивов и порядков сортировки (заданных константами `SORT_ASC` и `SORT_DESC`) и она переупорядочит элементы этих массивов, назначив новые индексы. Она подобна операции `join` в реляционной базе данных.

Представьте, что у нас есть много людей и несколько частей данных по каждой персоне:

```
$names = array("Том", "Дик", "Херит", "Бренда", "Джо");
$ages = array(25, 35, 29, 35, 35);
$zips = array(80522, '02140', 90210, 64141, 80522);
```

Первый элемент каждого массива представляет одну запись – вся информация о Томе. Аналогично, второй элемент представляет вторую запись – всю информацию о Дике. Функция `array_multisort()` переупорядочивает элементы массивов, сохраняя записи. Поэтому если Дик должен быть в массиве `$names` перед Томом, то остальная информация в остальных двух массивах также должна предшествовать информации о Томе. (Обратите внимание, что нам нужно заключить почтовый индекс Дика в кавычки, чтобы PHP не интерпретировал его как восьмеричное значение).

Здесь мы сортируем записи сначала по возрасту (в порядке возрастания), а затем в порядке убывания по почтовому индексу:

```
array_multisort($ages, SORT_ASC, $zips, SORT_DESC, $names, SORT_ASC);
```

Нам нужно включить `$names` в функцию, чтобы убедиться, что имя Дика останется с его возрастом и индексом. Выведем результат сортировки:

```
for ($i = 0; $i < count($names); $i++) {
      echo "{$names[$i]}, {$ages[$i]}, {$zips[$i]}\n";
}
```

Вывод:

```
Том, 25, 80522
Херит, 29, 90210
Джо, 35, 80522
Бренда, 35, 64141
Дик, 35, 02140
```

Инвертирование массивов

Функция `array_reverse()` инвертирует внутренний порядок элементов в массиве:

```
$reversed = array_reverse(array);
```

Числовые ключи будут перенумерованы, начиная с **0**, строковые индексы останутся нетронутыми. В общем, намного лучше использовать функции сортировки в обратном порядке вместо сортировки массива с последующим его инвертированием.

Функция `array_flip()` возвращает массив наоборот, то есть ключи массива становятся значениями, а значения массива – ключами.

```
$flipped = array_flip(array);
```

Например, если у вас есть массив, отражающий имена пользователей в их домашние каталоги, вы можете использовать `array_flip()` для создания массива, отражающего домашние каталоги в имена пользователей:

```
$u2h = array(
      'gnat' => "/home/staff/nathan",
      'frank' => "/home/action/frank",
      'petermac' => "/home/staff/petermac",
      'ktatroe' => "/home/staff/kevin"
);

$h2u = array_flip($u2h);
$user = $h2u["/home/staff/kevin"]; // $user = 'ktatroe'
```

Элементы исходного массива, не являющиеся ни строками, ни целыми числами, останутся нетронутыми в результирующем массиве. Новый массив поможет вам найти ключи в оригинальном массиве по их значениям, но данный метод эффективен только тогда, когда у исходного массива значения уникальны.

Сортировка в случайном порядке

Чтобы перемешать элементы в случайном порядке, используйте функцию `suffle()`. Она заменяет все существующие ключи – строки или числа – последовательными целыми числами, начинающимися с 0. Рассмотрим, как рандомизировать порядок дней недели:

```
$weekdays = array("Пн", "Вт", "Ср", "Чт", "Пт");
shuffle($weekdays);
print_r($days);
```

Результат:

```
Array(
      [0] => Вт
      [1] => Чт
      [2] => Пн
      [3] => Пт
      [4] => Ср
      )
```

Очевидно, порядок после `shuffle()` может не совпасть с демонстрационным выводом здесь из-за случайной природы функции. Если вам нужно просто выбрать случайный элемент из массива, используйте функцию `rand()` для выбора случайного индекса элемента, что более эффективно.

5.4.23. Работа со всем массивом

В PHP есть несколько функций, которые применяются сразу ко всем элементам массива. Вы можете выполнить слияние массивов, найти разницу между двумя массивами, вычислить сумму всех элементов массива и т.д.

Вычисление суммы всех элементов массива

Функция `array_sum()` вычисляет сумму значений индексированного или ассоциативного массива:

```
$sum = array_sum(array);
```

Например:

```
$scores = array(98, 76, 56, 80);
$total = array_sum($scores);          // $total = 310
```

Соединение двух массивов

Функция array_merge() интеллектуально объединяет два или более массива:

```
$merged = array_merge(array1, array2 [, array ... ])
```

Если числовой ключ из более раннего массива повторяется, значению из более позднего массива присваивается новый числовой ключ:

```
$first = array("привет", "мир");   // 0 => "привет", 1 => "мир"
$second = array("exit", "here"); // 0 => "exit", 1 => "здесь"
$merged = array_merge($first, $second);
// $merged = array("привет", "мир", "выход", "здесь")
```

Если строковой ключ из раннего массива повторяется, ранее значение будет заменено поздним значением:

```
$first = array('билл' => "клинтон", 'тони' => "данза");
$second = array('билл' => "гейтс", 'адам' => "вест");
$merged = array_merge($first, $second);
// $merged = array('билл' => "гейтс", 'тони' => "данза", 'адам' => "вест")
```

Вычисление разницы между двумя массивами

Другая часто используемая функция принимает набор массивов и возвращает разницу, то есть значения одного массива, которые отсутствуют в другом массиве. Функция `array_diff()` вычисляет разницу, возвращает массив со значениями из первого массива, которых нет во втором массиве:

```
$diff = array_diff(array1, array2 [, array ... ]);
```

Например:

```
$a1 = array("билл", "клара", "элла", "саймон", "юлия");
$a2 = array("джек", "клара", "тони");
$a3 = array("элла", "саймон", "галина");

// найти значения $a1, которых нет в $a2 или $a3
$difference = array_diff($a1, $a2, $a3);

print_r($difference);
```

Результат:

```
Array(
      [0] => "билл",
      [4] => "юлия"
);
```

Значения сравниваются с использованием оператора `===`, поэтому `1` и `"1"` – разные значения. Ключи первого массива сохраняются, поэтому в `$diff` ключ значения `"билл"` равен `0`, а ключ значения `"юлия"` равен `4`.

В другом примере следующий код вычисляет разницу двух массивов:

```
$first = array(1, "два", 3);

$second = array("два", "три", "четыре");

$difference = array_diff($first, $second);

print_r($difference);
```

Результат:

```
Array(
   [0] => 1
   [2] => 3
)
```

Фильтрация элементов массива

Для выделения подмножества массива на основании его значений, можно использовать функцию `array_filter()`:

```
$filtered = array_filter(array, callback);
```

Каждое значение массива *array* передается функции `callback`. Возвращаемый массив содержит только те элементы исходного массива, для которых функция вернула `true`. Например:

```
// нечетный
$callback = function isOdd ($element)
{
      return $element % 2;
};

$numbers = array(9, 23, 24, 27);

$odds = array_filter($numbers, $callback);
// $odds = array(0 => 9, 1 => 23, 3 => 27)
```

Как видите, ключи массива сохраняются. Эта функция наиболее полезна с ассоциативными массивами.

5.5. Обработка форм

В PHP очень просто обрабатывать формы, поскольку параметры формы доступны в массивах `$_GET` и `$_POST`. В этом разделе мы рассмотрим много трюков и методов работы с формами.

5.5.1. Методы

Существует два HTTP-метода, которые клиент может использовать для передачи данных формы на сервер: GET и POST. Метод, который будет использовать конкретная форма, указывается атрибутом `method` тега `form`. Теоретически, название методов не чувствительны к регистру, но некоторые браузеры могут требовать, чтобы название метода было указано прописными буквами.

Запрос GET кодирует параметры формы в URL, который называется строкой запроса. Текст, который следует после `?` и есть строка запроса:

```
/path/to/chunkify.php?word=despicable&length=3
```

Метод POST требует передачи параметров формы в теле HTTP-запроса, оставляя нетронутым URL.

Основная видимая разница между методами GET и POST - это строка URL. Поскольку при использовании метода GET все параметры формы кодируются в URL, пользователи могут добавить в закладки все GET-запросы. Однако, они не могут добавить в закладки POST-запросы.

Наибольшая разница между запросами GET и POST, однако, намного более тонкая. Спецификация HTTP говорит, что GET-запросы идемпотентны, то есть один GET-запрос для определенного URL, включая параметры формы, такие же, как два или более запроса для этого URL. Поэтому веб-браузеры могут кэширования страницы ответов для GET-запросов, поскольку страница ответа не изменяется, независимо от того, сколько раз страница была загружена. Из-за этого GET-запросы должны использоваться только для запросов, ответы на которые никогда не будут изменяться, например, разделение строки на меньшие блоки или умножение чисел.

POST-запросы не являются идемпотентными. Это означает, что они не могут быть кэшированы и при каждом отображении страницы нужно свя-

заться с сервером. Вероятно, ваш браузер спросит вас "Повторить отправку данных формы?" перед отображением или перезагрузкой определенных страниц. Из-за этого POST-запросы нужно использовать для запросов, ответы на которые могут изменяться со временем, например, отображение содержимого корзины или текущие сообщения доски объявлений.

Однако, идемпотентность часто игнорируется в реальном мире. Кэш браузеров обычно так плохо реализованы, да и кнопку *Обновить* так просто нажать, что программисты часто используют запросы GET или POST только на основании того, хотят ли они видеть параметры запроса в URL или нет. Важно помнить, что GET-запросы не должны использоваться для любых действий, которые вызывают изменения на сервере, например, размещение заказа или обновление базы данных.

Тип метода, используемый при запросе PHP-страницы доступен через `$_SERVER['REQUEST_METHOD']`. Например:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
// обрабатываем GET-запрос
}
else {
die("Вы должны использовать только GET-запрос!");
}
```

5.5.2. Параметры

Используйте массивы `$_POST`, `$_GET` и `$_FILES`, чтобы получить доступ к параметрам формы. Ключи массива - это имена параметров, а значения массива - это значения тех параметров. Поскольку в именах HTML можно использовать точки, но они запрещены в PHP-имена, точки будут преобразованы в знаки подчеркивания (_) в массиве.

Листинг 5.5 показывает HTML-форму, используемую для разбивки слова на блоки. Форма содержит два поля: одно для строки (имя параметра `word`), а одно - для длины блоков (имя параметра `number`).

Листинг 5.5. Форма chunkify.html

```
<html>
<head><title>Форма разбивки строки на части</title></head>
  <body>
      <form action="chunkify.php" method="POST">
```

```
        Введите слово: <input type="text" name="word" /><br
/>
          Длина одной части слова? <input type="text"
      name="number" /><br />
        <input type="submit" value="Разбить!">
      </form>
  </body>
</html>
```

Пример 5.7 содержит PHP-сценарий *chunkify.php*, которому передаются параметры формы, приведенной в примере 5.5. Сценарий копирует параметры в переменные, а затем использует их.

Пример 5.7. Сценарий chunkify.php

```
<?php
$word = $_POST['word'];
$number = $_POST['number'];
$chunks = ceil(strlen($word) / $number);

echo  "Блоки размером {$number} символов слова '{$word}':<br
/>\n";

for ($i = 0; $i < $chunks; $i++) {
      $chunk = substr($word, $i * $number, $number);
      printf("%d: %s<br />\n", $i + 1, $chunk);
}
?>
```

5.5.3. Самообработка страниц

PHP страница может использоваться не только для обработки формы, но и для ее создания. Если страница, показанная в примере 5.8, будет запрошена методом GET, она выведет форму, принимающую температуру в Фаренгейтах. Если страница вызвана методом POST, она вычисляет и отображает соответствующую температуру в Цельсиях.

Пример 5.8. Само-обработка страниц (*temp.php*)

```
<html>
  <head><title>Преобразование температуры</title></head>
  <body>
      <?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
```

```
        <form action="<?php echo $_SERVER['PHP_SELF']
    ?>" method="POST">
        Температура в Фаренгейтах:
        <input type="text" name="fahrenheit" /><br />
        <input type="submit" value="Конвертировать в
    Цельсии!" />
        </form>
    <?php }
    else if ($_SERVER['REQUEST_METHOD'] == 'POST') {

        $fahrenheit = $_POST['fahrenheit'];
        $celsius = ($fahrenheit - 32) * 5 / 9;
        printf("%.2fF = %.2fC", $fahrenheit, $celsius);

    }
    else {
    die("Этот сценарий работает только с методами GET и
POST");
    } ?>
  </body>
</html>
```

Рисунок 5.2. показывает страницу преобразования температуры и результат преобразования.

Рис. 5.2. Страница преобразования температуры и ее вывод

Другой способ решить, что делать - показать форму или обработать ее, проверить, был ли установлен один из параметров формы. Это позволяет написать страницу самообработки, использующую метод GET для передачи значений. Пример показывает новую версию нашей страницы самообработки, которая передает параметры, используя запрос GET. Для определения, что делать, эта страница использует присутствие или отсутствие параметров формы.

5.5.4. Липкие формы

Много веб-сайтов используют технику, известную как "липкие" формы, в которой результаты запроса сопровождаются поисковой формой, использующей в качестве значений по умолчанию значения из предыдущего запроса.

Например, если вы ищите в Google "Программирование на PHP", поле поиска наверняка уже содержит эту строку. Чтобы изменить строку запроса на "Программирование на PHP от O'Reily", вам нужно просто добавить дополнительные ключевые слова.

Подобную липкую форму достаточно просто реализовать. Пример 5.9 показывает наш сценарий преобразования температуры (см. лист. 5.8) с так называемой липкой формой. Основной прием - использовать форму со значениями по умолчанию при создании HTML-поля.

Пример 5.9. Преобразование температуры с липкой формой (sticky_form.php)

```
<html>
  <head><title>Преобразование температуры</title></head>
  <body>
      <?php $fahrenheit = $_GET['fahrenheit']; ?>
        <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
        Температура в Фаренгейтах:
      <input type="text" name="fahrenheit" value="<?php
echo $fahrenheit; ?>" />
        <br />
        <input type="submit" value="Преобразовать в
Цельсии!" />
      </form>
      <?php if (!is_null($fahrenheit)) {
            $celsius = ($fahrenheit - 32) * 5 / 9;
            printf("%.2fF = %.2fC", $fahrenheit, $celsius);
```

```
      } ?>
  </body>
</html>
```

5.5.5. Многозначные параметры

Списки HTML, созданные тегом `SELECT` позволяют выбирать несколько значений. Чтобы убедиться, что PHP распознает несколько значений, которые браузер передаст сценарию обработки формы, вам нужно снабдить имя поле в HTML-форме квадратными скобками `[]`. Например:

```
<select name="languages[]">
      <option name="c">C</input>
      <option name="c++">C++</input>
      <option name="php">PHP</input>
      <option name="perl">Perl</input>
</select>
```

Теперь, когда пользователь нажмет кнопку *Отправить*, `$_GET['languages']` содержит массив вместо простой строки. Этот массив содержит значения, которые были выбраны пользователем.

Пример показывает форму, позволяющую выбрать несколько значений из списка, а также сценарий ее обработки. Форма предоставляет пользователю выбрать несколько характеристик его личности.

Листинг 5.9. Выбор нескольких значений списка (select_array.php)

```
<html>
  <head><title>Личность</title></head>
  <body>
      <form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
      Охарактеризуйте себя:<br />
      <select name="attributes[]" multiple>
            <option value="веселый">Веселый</option>
            <option value="угрюмый">Угрюмый</option>
            <option value="умный">Умный</option>
            <option value="чувствительный">Чувствительный</
      option>
            <option value="расточительный">Расточительный</
```

```
      option>
            <option value="шоппер">Шоппер</option>
      </select><br />
      <input type="submit" name="s" value="Записать!" />
      </form>

<?php if (array_key_exists('s', $_GET)) {
        $description = join(' ', $_GET['attributes']);
        echo "Вы - {$description}";
      } ?>
  </body>
</html>
```

В примере 5.9 кнопка `submit` называется "s". Мы проверяем наличие этого параметра, чтобы понять, нужно ли нам обрабатывать форму. Рисунок 5.3 показывает форму и результат ее обработки.

Подобная техника может использоваться для любой формы, которая может возвращать несколько значений. Пример 5.10 показывает измененную версию нашей формы выбора характеристик личности, переписанную с использованием переключателей (checkbox) вместо списка (select). Обратите внимание: изменился только HTML-код, код обработки остался неизменный, независимо от того, что мы используем - переключатели или список.

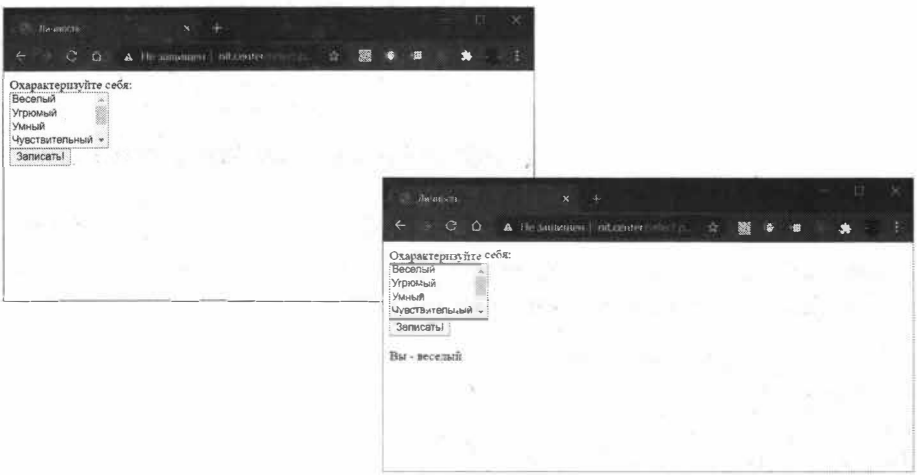


Рис. 5.3. Выбор нескольких значений списка и их обработка

Пример 5.10. Обработка переключателей (checkbox_array.php)

```
<html>
<head><title>Личность</title></head>
<body>
  <form action="<?php $_SERVER['PHP_SELF']; ?>" method="GET">
  Охарактеризуйте себя:<br />
  <input type="checkbox" name="attributes[]" value="веселый"
/>Веселый<br />
  <input type="checkbox" name="attributes[]" value="угрюмый"
/>Угрюмый<br />
  <input type="checkbox" name="attributes[]" value="умный"
/>Умный<br />
  <input type="checkbox" name="attributes[]" value="чувствительный
"/>Чувствительный<br/>
  <input type="checkbox" name="attributes[]" value="расточительный
"/>Расточительный<br/>
  <input type="checkbox" name="attributes[]" value="шоппер" />
Шоппер<br />
  <br />
  <input type="submit" name="s" value="Записать!" />
  </form>

  <?php if (array_key_exists('s', $_GET)) {
            $description = join (' ', $_GET['attributes']);
            echo "Вы - {$description}";
      } ?>
</body>
</html>
```

5.5.6. Липкие многозначные параметры

Теперь вы зададитесь вопросом, можно ли сделать многозначные параметры липкими? Да, можно, но это не просто. Вам нужно проверить каждое значение - было ли оно установлено в форме. Например:

```
Умный: <input type="checkbox" name="attributes[]"
value="умный"
<?php
if (is_array($_GET['attributes']) && in_array('умный', $_
GET['attributes'])) {
      echo "включен";
} ?> /><br />
```

Вы можете использовать эту технику для каждого переключателя, но она подвержена ошибкам. Намного проще написать функцию, которая будет генерировать HTML для возможных значений и работать с копией отправленных параметров. Пример 5.11 показывает новую версию формы из примера 5.10. Хотя форма подобна той, которая была представлена в примере 5.10, за сценой находится способ генерирования формы.

Пример 5.11. Липкие многозначные переключатели (checkbox_array2.php).

```
<html>
<head><title>Личность</title></head>
<body>

<?php // получаем значения формы, если они есть
$attrs = $_GET['attributes'];

if (!is_array($attrs)) {
      $attrs = array();
}

// создаем HTML для переключателей с такими же именами
function makeCheckboxes($name, $query, $options)
{
      foreach ($options as $value => $label) {

      $checked = in_array($value, $query) ? "checked" : '';

      echo "<input type=\"checkbox\" name=\"{$name}\"
      value=\"{$value}\" {$checked} />";
      echo "{$label}<br />\n";
      }
}
// список значений и меток для переключателей
$personalityAttributes = array(
  'веселый' => "Веселый",
  'угрюмый' => "Угрюмый",
  'умный' => "Умный",
  'чувствительный' => "Чувствительный",
  'расточительный' => "Расточительный",
  'шоппер' => "Шоппер"
); ?>
```

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="GET">
Охарактеризуйте себя:<br />
<?php makeCheckboxes('attributes[]', $attrs,
$personalityAttributes); ?><br />
<input type="submit" name="s" value="Записать!" />
</form>

<?php if (array_key_exists('s', $_GET)) {
      $description = join (' ', $_GET['attributes']);
      echo "Вы - {$description} personality.";
} ?>
</body>
</html>
```

Сердце этого кода - функция `makeCheckboxes()`. Она принимает три аргумента: имя группы переключателей, массив значений по умолчанию и массив, отображающий значения параметров в метки переключателей. Список параметров для переключателей содержится в массиве `$personalityAttributes`.

5.5.7. Загрузка файлов

Для управления загрузкой файлов (она поддерживается большинством современных браузеров) используется массив `$_FILES`. Используя различную аутентификацию и функции загрузки файлов, вы можете контролировать, кому разрешено загружать файлы и что может сделать пользователь с этими файлами в вашей системе. Следующий код отображает форму, которая позволяет загружать файлы на этой же странице:

```
<form enctype="multipart/form-data"
  action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="10240">
Имя файла: <input name="toProcess" type="file" />
<input type="submit" value="Загрузить" />
</form>
```

Самая большая проблема при загрузке файлов - риск получения слишком большого файла. У PHP есть два способа предотвращения этого: жесткое ограничение и мягкое ограничение. Опция `upload_max_filesize` в *php.ini*

позволяет задать жесткое ограничение размера загружаемых файлов (по умолчанию 2 Мб). Если ваша форма передаст параметр, названный MAX_FILE_SIZE, PHP будет использовать его в качестве мягкого ограничения. Например, в предыдущем примере установлено ограничение в 10 Кб. PHP игнорирует значение MAX_FILE_SIZE, если оно превышает значение upload_max_filesize. Также обратите ваше внимание, что у формы загрузки файлов должен быть атрибут enctype со значением "multipart/form-data".

Каждый элемент в массиве $_FILES предоставляет информацию о загруженном файле:

- name - имя загруженного файла, переданное браузером. Довольно сложно сделать с этим что-то полезное, поскольку на клиентской машине используются несколько другие имена, чем на веб-сервере. Например, если клиентская машина работает под управлением Windows и сообщит вам, что файл называется *D:\PHOTOS\ME.JPG*, а веб-сервер работает под управлением Unix, данное имя файла будет бессмысленным для него.
- type - MIME-тип загруженного файла, предложенный клиентом.
- size - размер загруженного файла (в байтах). Если пользователь пытается загрузить слишком большой файл, размер будет установлен в 0.
- tmp_name - имя временного файла на сервере после его загрузки. Если пользователь попытается загрузить слишком большой файл, это имя будет содержать значение "none".

Правильный способ проверить, был ли успешно загружен файл - использовать функцию is_uploaded_file(), как показано ниже:

```
if (is_uploaded_file($_FILES['toProcess']['tmp_name'])) {
      // файл успешно загружен
}
```

На сервере файлы хранятся во временном каталоге сервера по умолчанию, который задан в файле *php.ini* опцией upload_tmp_dir. Чтобы переместить файл в другой каталог, используйте функцию move_uploaded_file():

```
move_uploaded_file($_FILES['toProcess']['tmp_name'], "path/
to/put/file/{$file}");
```

Вызов `move_uploaded_file()` автоматически проверяет, был ли загружен файл. По завершению работы сценария все загруженные ним файлы будут удалены из временного каталога.

5.5.8. Проверка формы

Перед использованием или сохранением для последующего использования данных, введенных пользователем, вам нужно сначала проверить их. Доступно несколько стратегий проверки данных. Первая заключается в использовании JavaScript на стороне клиента. Однако поскольку пользователь может выключить JavaScript или если браузер не поддерживает его, эта проверка не может быть единственной.

Более безопасный способ - использовать PHP для проверки данных. Пример 5.12 показывает страницу с формой. Страница позволяет пользователю ввести медиа-элемент. Все три элемента формы - имя, тип медиа-элемента и имя файла - являются обязательными. Если пользователь забудет ввести какое-либо из значений, страница сообщит ему об этом. Любые ранее заполненные пользователем поля будут установлены в ранее введенные значения. А кнопка "Создать" будет заменена кнопкой "Продолжить", когда пользователь модифицирует форму.

Листинг 5.12. Проверка формы (data_validation.php)

```
<?php
$name = $_POST['name'];
$mediaType = $_POST['media_type'];
$filename = $_POST['filename'];
$caption = $_POST['caption'];
$status = $_POST['status'];

$tried = ($_POST['tried'] == 'yes');

if ($tried) {
  $validated = (!empty($name) && !empty($mediaType) &&
!empty($filename));

  if (!$validated) { ?>
      <p>Имя, тип файла и имя файла - обязательные поля.
      Заполните их для продолжения</p>
<?php }
}
```

```
if ($tried && $validated) {
      echo "<p>Элемент был создан.</p>";
}

// был ли выбран тип файла? выведет "selected" если так
function mediaSelected($type)
{
  global $mediaType;

  if ($mediaType == $type) {
      echo "selected"; }
} ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
method="POST">
  Имя: <input type="text" name="name" value="<?= $name; ?>"
/><br />

  Статус: <input type="checkbox" name="status"
value="active"
  <?php if ($status == "active") { echo "checked"; } ?> />
Активен<br />

  Тип: <select name="media_type">
    <option value="">Выберите:</option>
    <option value="picture" <?php mediaSelected("picture");
?> />Картинка</option>
    <option value="audio" <?php mediaSelected("audio"); ?>
/>Аудио</option>
    <option value="movie" <?php mediaSelected("movie"); ?>
/>Фильм</option>
  </select><br />

  Файл: <input type="text" name="filename" value="<?=
$filename; ?>" /><br />

  Заголовок: <textarea name="caption"><?= $caption; ?></
textarea><br />
  <input type="hidden" name="tried" value="yes" />
  <input type="submit" value="<?php echo $tried ?
"Продолжить" : "Создать"; ?>" />
</form>
```

В этом случае проверка допустимости - это проверка предоставленного значения. Мы устанавливаем `$validated` в `true`, когда все три переменные - `$name`, `$type` и `$filename` - содержат информацию. Другие возможные проверки могут проверять допустимость адреса электронной почты, существование локального файла и т.д.

Например, чтобы проверить поле возраста и убедиться, что оно содержит неотрицательное целое число, можно использовать этот код:

```
$age = $_POST['age'];
$validAge = strspn($age, "1234567890") == strlen($age);
```

Вызов `strspn()` находит число цифр в начале строки. Если пользователь указал положительное целое число, то вся строка будет состоять из цифр. Можно было бы также использовать регулярное выражение для осуществления этой проверки:

```
$validAge = preg_match('/^\d+$/', $age);
```

Проверка допустимости адресов электронной почты - почти невыполнимая задача. Нет никакого способа определить, соответствует ли строка допустимому адресу электронной почты. Однако, вы можете найти опечатки, требуя, чтобы пользователь ввел адрес электронной почты дважды (в двух разных полях). Вы также можете предотвратить введение электронных адресов вида "*me*" или "*me@aol*", требуя наличия знака `@` и хотя бы одной точки после него. В дополнение можно проверить на соответствие домена, на которые вы не хотите отправлять почту, например, whitehouse.gov или fsb.ru.

Например:

```
$email1 = strtolower($_POST['email1']);
$email2 = strtolower($_POST['email2']);

if ($email1 !== $email2) {
      die("Адреса e-mail не совпадают");
}

if (!preg_match('/@.+\..+$/', $email1)) {
      die("Некорректный адрес e-mail");
}
```

```
if (strpos($email1, "whitehouse.gov")) {
      die("Я не буду отправлять почту в Белый дом");
}
```

Проверка допустимости полей обычно является операцией со строками. В этом примере мы использовали регулярные выражения и функции обработки строк, чтобы убедиться, что строки, введенные пользователем - это то, что мы ожидаем.

5.6. Базы данных

PHP поддерживает более 20 баз данных, в том числе большинство популярных коммерческих и открытых разновидностей. Реляционные системы баз данных (MySQL, PostgreSQL и Oracle) являются магистралью большинства современных веб-сайтов. В них хранится информация корзины, история заказов, обзоры продуктов, информация о пользователе, номера кредитных карточек и иногда даже сами веб-страницы.

В этой главе будет показано, как получить доступ к базам данных из PHP. Мы сконцентрируемся на встроенном расширении для работы с СУБД MySQL – mysqli (это новая версия расширения, в отличие от устаревшего расширения mysql). Если вам нужно работать с другими СУБД, обратите внимание на расширение PDO, рассмотрение которого выходит за рамки этой книги.

5.6.1. Последовательность работы с базой данных

Работа с базой данных осуществляется так:

1. Устанавливаем подключение.
2. Выполняем запрос.
3. Читаем результат выполнения.
4. При необходимости повторяем пункты 2 и 3, если нужно выполнить несколько запросов к базе данных.
5. Закрываем соединение.

5.6.2. Подключение к базе данных

Сначала рассмотрим, как подключиться к СУБД MySQL. Для этого нужно создать новый объект **mysqli**:

```
$mysqli = new mysqli($DBHOST, $DBUSER, $DBPASSWD, $DBNAME);

if ($mysqli->connect_errno) {
    echo "SQL errno: " . $mysqli->connect_errno . "<br>\n";
    echo "Error: " . $mysqli->connect_error . "\n";
    die();
}
```

При подключении к базе данных нужно указать узел ($DBHOST), имя пользователя ($DBUSER), пароль ($DBPASSWD), имя базы данных ($DBNAME). После попытки установки соединения неплохо было бы проверить наличие ошибки подключения (свойство connect_errno). Если ошибка есть, мы выводим информацию о ней и завершаем работу сценария функцией die().

Чтобы подавить вывод информации об ошибке в момент самого подключения, нужно использовать знак @:

```
@$mysqli = new mysqli($host, $dbuser, $dbpass, $db);
```

После того, как соединение успешно установлено не будет лишним выбрать кодировку соединения для нормальной работы с символами национальных алфавитов:

```
$mysqli->set_charset("utf8");
```

В нашем реальном проекте, который будет рассмотрен в последней части этой книги, процесс подключения к базе данных мы вынесли в файл connect.php, который инклудится каждым сценарием, которому необходим доступ к БД:

```
<?php

@$mysqli = new mysqli($host, $dbuser, $dbpass, $db);
```

```
if ($mysqli->connect_errno) {
    echo "SQL errno: " . $mysqli->connect_errno . "<br>\n";
    echo "Error: " . $mysqli->connect_error . "\n";
    die();
}

$mysqli->set_charset("utf8");

?>
```

Переменные $host, $dbuser, $dbpass и $db определены в другом include-файле – config.php:

```
<?php
// Параметры для доступа к БД
$host = "localhost";
$db = "cq96281_db";
$dbuser = "cq96281_db";
$dbpass = "topsecretpassword";
```

5.6.3. Выполнение запроса к базе данных

Выполнить запрос к базе данных очень просто: для этого нужно передать сформированную строку с SQL-запросом передать в метод query() объекта mysqli. В результате будет получен другой объект – объект результата, через который можно получить доступ к результатам выполнения запроса.

Пример:

```
$q = "select * from price where id=$k";
$result = $mysqli->query($q);
```

Строку запроса лучше всего хранить в отдельной переменной, особенно, если она формируется динамически, как в нашем примере. Тогда вы в любой момент можете вывести ее, чтобы убедиться, что передаете в БД именно то, что хотели:

```
$q = "select * from price where id=$k";
echo $q;
$result = $mysqli->query($q);
```

Сам объект результата содержит некоторые полезные данные (кроме самого результата). Так, в его свойстве num_rows содержится количество строк результата, а свойство *filed_count* содержит количество полей в результате.

Вот вывод объекта результата функцией print_r():

```
mysqli_result Object
(
    [current_field] => 0
    [field_count] => 8
    [lengths] =>
    [num_rows] => 1
    [type] => 0
)
```

5.6.4. Получение данных

Итак, мы выполнили запрос к базе данных. Теперь нам нужно прочитать ответ. Разумеется, читать ответ нужно только в том случае, если наш запрос подразумевает какой-либо ответ от базы данных. Как правило, это SELECT-запрос. Для запросов INSERT (вставка), UPDATE (обновление), DELETE/TRUNCATE/DROP (удаление), CREATE (создание) читать ответы не нужно. Можно лишь прочитать свойство **errno** (номер ошибки) или **error** (строка с описанием ошибки) и этого будет достаточно – ведь нужно знать, выполнилась ли операция или нет. А еще проще – проверять на истинность результат выполнения запроса

```
/* Создание таблицы не возвращает результирующего набора */
if ($mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City")
=== TRUE) {
      printf("Таблица myCity успешно создана.");
}
else {
      die("Невозможно продолжить работу. Таблица myCity не
создана");
```

Прочитать ответ проще всего методом fetch_assoc() результирующего объекта. В результате мы получим ассоциативный массив (или *false*, если в результате больше нет записей), представляющий собой одну запись таблицы-результата.

Рассмотрим пример:

```
$sql = "select * from p_invoices";
$result = $mysqli->query($sql);

$sum = 0;

while ($row = $result->fetch_assoc()) {
   echo "<p>$row[id] $row[dt] $row[title] $row[sum]";
   $sum = $sum + $row['sum'];
}
echo "Итого: $sum";
```

Сначала мы делаем запрос, выбирающий все счета. Далее мы в цикле while(), который удобно использовать в нашем случае, итерируем по всем записям результата (если таковые есть) и выводим ID счета, дату, заголовок и сумму. В цикле мы также считаем итоговую сумму.

Ключи массива $row соответствуют названиям колонок таблицы MySQL.

Узнать количество строк в результате можно с помощью свойства *num_rows*:

```
if ($result->num_rows == 0) {
      echo "Таблица пуста";
else {
      // обработка результата
      ...
```

Если вы производили вставку в таблицу с полем AUTO_INCREMENT, вы можете использовать свойство *insert_id*, чтобы узнать идентификатор последней вставленной записи. Например, если есть таблица:

```
ID    Username
1     Den
2     Mark
```

и вы добавляете в нее новую запись, а затем читаете свойство *insert_id*, то оно будет равно 3:

```
echo "ID нового пользователя " . $result->insert_id;
```

5.6.5. Закрытие соединения

По большому счету, закрывать соединения не обязательно, поскольку они закрываются автоматически, когда завершается работа сценария. Однако это правило не касается постоянных соединений. В расширении mysqli есть метод close():

```
$mysqli->close();
```

5.6.6. Получение суммы колонки

Ранее был показан пример, где мы вычисляли итоговую сумму в цикле при выводе списка счетов. Когда вы получаете все записи таблицы и планируете их как-то обрабатывать в цикле, например, выводить в браузер, то такой вариант вполне приемлемый. Но когда нужно просто вычислить сумму, нет особого смысла получать все записи, а затем в цикле производить подсчет суммы. Достаточно использовать встроенную SQL-функцию *sum*:

```
SELECT SUM(колонка) FROM таблица;
```

Такой запрос нужно передать базе данных, получить ответ и вывести его. Рассмотрим пример:

```
$q = "select sum(summa) from p_cart";
$resq = $mysqli->query($q);
$row = $mysqli->fetch_row();
$totalSum = $row[0]; // итого
```

Нужный нам результат будет храниться в массиве $row под индексом **0**.

На этом все. В следующей части книги мы рассмотрим основы языка SQL. Вы научитесь производить выборку, вставку, обновление и удаление записей.

Глава 6.

Основы MySQL

6.1. Знакомство с MySQL

6.1.1. Основная информация о MySQL

MySQL – это быстрый, надежный, многопоточный, многопользовательский сервер баз данных. Сервер MySQL предназначен для критически важных производственных систем с большой нагрузкой, а также для встраивания в массово развертываемое программное обеспечение.

Для таких программных продуктов часто используется (и мы его будем использовать в этой книге) термин СУБД – система управления базой данных (в англоязычной литературе – DBMS, Database Management System).

Нужно отметить, что MySQL – самая популярная СУБД с открытым исходным кодом. В данный момент распространяется и поддерживается корпорацией Oracle. Все мы знаем, что такое СУБД Oracle. Правильно, это СУБД для крупных корпораций. Для всех остальных – MySQL. Именно этим и объясняется популярность данной СУБД. MySQL установлена на всех (!) хостингах. Другими словами, когда вы покупаете хостинг, в 100% случаев на нем в качестве СУБД будет использоваться именно MySQL. Поэтому полученные в этой части книги являются универсальными и пригодятся при разработке сайта любой сложности.

База данных MySQL является реляционной. В реляционной базе данных данные хранятся в отдельных таблицах, а не в одном большом хранилище. Структуры базы данных хранятся в физических файлах, оптимизирован-

ных по скорости. Реляционная база данных – это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных. В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке – значение атрибута. Каждая стока таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей. К этим данным можно получить доступ многими способами, и при этом реорганизовывать таблицы БД не требуется.

Как вы заметили, в названии MySQL есть слово "SQL". SQL (Structured Query Language) – структурированный язык запросов. SQL - это наиболее распространенный стандартизированный язык, используемый для доступа к базам данных. В зависимости от среды программирования вы можете вводить SQL напрямую (например, для создания отчетов), встраивать операторы SQL в код, написанный на другом языке (например, на PHP), или использовать специфичный для языка API, скрывающий синтаксис SQL.

SQL определяется стандартом ANSI / ISO SQL. Стандарт SQL развивается с 1986 года, и существует несколько версий. Самая первая версия стандарта SQL появилась в 1986 году и получила название SQL-86. Далее стандарты выпускались в 1989, 1992, 1999, 2003, 2006, 2008, 2011 и 2016 годах. Начиная с 1999 года, стандарты SQL называются так: SQL:год_выпуска. Так, последняя версия называется SQL:2016.

6.1.2. Терминология

Перед тем, как приступить к формированию собственных запросов, рассмотрим некоторые термины:

- **Сущность** (entity) – то, что представляет интерес для пользователей базы данных, часть бизнес-процесса, например, клиент, запчасть, заказ и т.д.
- **Поле/Столбец** (field/column) — это базовый элемент данных в БД. Как уже было отмечено, у поля есть тип. Например, поле id может быть целым, а поле address - строкой.

- **Запись** (record) — это набор полей, содержащих связанную информацию. Пусть в таблице будут два поля **id** и **address**. Первое поле - это номер пользователя, а второе - адрес пользователя с номером **id**.

- **Таблица** (table) — это набор записей одной структуры полей. Все записи в таблице имеют одинаковую структуру.

- **База данных** (database) — это совокупность связанных таблиц. Впрочем, таблицы могут быть не всегда связаны, никто вам не мешает создать две абсолютно несвязанные таблицы в одной базе данных, поэтому будем считать, что база данных – это просто совокупность таблиц. Конечно, на практике все таблицы базы данных так или иначе связаны друг с другом.

- **Индекс** (index) — используется для быстрого поиска нужной записи в базе данных. Поиск может производиться по значению одного поля или по значению нескольких полей.

- **Первичный индекс/ключ** (index) — управляет порядком отображения записей в таблице. Поле первичного индекса должно быть уникальным - таблице не может быть два поля с одним и тем же значением.

- **Вторичный индекс/ключ** (secondary index) — в отличие от первичного индекса, может строиться по нескольким полям и необязательно должен быть уникальным. Вторичные индексы используются для связывания таблиц.

- **Представления** (views) - предоставляют единственное представление о данных, полученных из одной или нескольких таблиц или представлений. Представление является альтернативным интерфейсом к данным, которые хранятся в основной таблице (ах), по которым было создано представление.

- **Хранимая процедура** (stored procedure) - объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

Индексы часто называются *ключами:* первичный индекс называется первичным ключом, а вторичный — вторичным ключом.

6.1.3. Что такое SQL

SQL (Structured Query Language) – язык структурированных запросов к базе данных. Здесь все просто – пользователь вводит запрос, база данных его выполняет и возвращает какие-то данные или результат выполнения запроса, например, количество затронутых запросом строк.

Основное преимущество SQL — простота его использования. Предположим, что SQL нет. Но нам необходима база данных. Что делать? Сначала нужно разработать формат двоичного файла (с двоичным файлом операции поиска выполняются быстрее, чем с текстовым): продумать заголовок файла, разделители полей и индексы. На все это уйдет много времени. Еще больше времени займет разработка набора функций, которые будут работать с нашим двоичным форматом. Спустя какое-то время этот набор функций мы создадим. Затем напишем приложение с их использованием. Допустим, нам потребуется просмотреть весь файл данных и выбрать из него записи с определенным значением поля, например, чтобы поле "Количество" было больше нуля. Алгоритм обработки будет следующим: нужно в цикле пересмотреть все записи и вывести только те, которые не содержат нуль в поле "Количество". Не очень удобно, но все же выполнимо. Но вот незадача: а что делать, если возникнет необходимость радикально изменить формат файла? Ведь при разработке мы могли допустить ошибку, точнее недоработку, которая "всплыла" несколько месяцев спустя. Тогда нам придется заново переделывать весь наш набор функций. Причем делать все так, чтобы это не затронуло работу уже существующих приложений, иначе нужно будет переписывать и эти приложения.

В случае с SQL все гораздо проще. Вам не нужно изобретать велосипед заново: забудьте о формате данных и наборе функций — для вас этих проблем просто не существует. Все, что вам необходимо знать — это то, что есть база данных, которая выполнит любой ваш "приказ", сформулированный по правилам языка SQL. Программировать вам тоже ничего не придется — зачем циклы, если можно попросить SQL: "Просмотри все записи, и выбери те, у которых поле "Количество" равно нулю". Даже если возникнет необходимость, можно будет перейти на другую систему управления базами данных, SQL-запросы останутся прежними.

Таким образом, преимущества SQL следующие:

- Независимость от системы управления базами данных — благодаря тому, что приняты стандарты SQL, его запросы будут оди-

наково выполняться, что в Oracle, что в MySQL. Конечно, есть определенные отличия синтаксиса некоторых СУБД, но они, как правило, незначительны.

- Полноценность SQL как языка управления данными — с помощью SQL можно не только производить выборку данных, но и полноценно управлять ними: добавлять новые данные, изменять и удалять уже имеющиеся, создавать базы данных. Выходит, что кроме знания SQL, вам не понадобятся никакие другие дополнительные сведения для работы с СУБД.

Язык SQL, согласно стандарту ANSI, состоит из нескольких подразделов:

- Data Definition Language (DDL) или Schema Definition Language (SDL) - язык определения данных или язык определения схемы (такое определение используется в ANSI). Состоит из команд, создающих объекты (таблицы, индексы, представления) в базе данных.

- Data Manipulation Language (DML) - язык манипулирования данными. Команды, определяющие, какие данные содержатся в таблицах в любой момент времени.

- Data Control Language (DCL) - язык управления данными. Содержит команды, определяющие может ли пользователь выполнить отдельное действие.

6.1.4. Как выглядят запросы и как их передать базе данных

Поскольку SQL является структурированным языком запросов, все запросы в нем имеют четкую структуру, которой рекомендуется придерживаться, если вы хотите, чтобы ваш код был удобен для чтения. Вот пример запроса SELECT (выборка данных):

```
SELECT *
FROM users
WHERE active = 0;
```

Как видите, не зря SQL когда-то назывался SEQL: уж очень он похож на обычный английский язык. Понять суть этого запроса можно, даже не имея представления об SQL. Очевидно, этот запрос выбирает информацию обо всех клиентах со статусом 0 (что означает этот статус, зависит от проекта базы данных).

Для передачи запроса нужно использовать клиент для работы с базой данных. В случае с СУБД MySQL, у вас есть три основных варианта (есть и другие, но для работы с этой книгой вам будет достаточно знать о трех вариантах):

1. Консольная программа **mysql**. Использовать ее не очень удобно, как и все программы такого рода. Да и в случае с хостингом использовать ее вам вряд ли получится, поскольку на хостинге запускать программы нельзя, а подключаться удаленно к серверу БД (когда соединение инициируется с удаленного узла) – тоже нельзя – это запрещено политикой безопасности во многих случаях. Когда же вы пишете PHP-скрипт, то он будет выполняться на том же сервере, что и MySQL, поэтому соединение устанавливается с localhost – с локальным по отношению к PHP-скрипту узлу.

2. **MySQL Workbench** – полноценная среда для работы с базой данных. Входит в состав СУБД MySQL. В Windows устанавливается автоматически при установке MySQL-сервера, в Linux нужно устанавливать отдельно. Необходимые инструкции описаны по адресу https://dev.mysql.com/doc/workbench/en/wb-installing-linux.html. Замечания относительно этой программы такие же, как и относительно консольной версии – с ее помощью вы сможете подключиться только к тому серверу, который вы контролируете лично сами и где разрешили удаленное подключение к БД.

3. Приложение **phpMyAdmin** – самый удобный вариант для новичка. Приложение работает через браузер, все видно и понятно. Вы можете просматривать таблицы, создавать новые, изменять структуру таблиц, вставлять, обновлять и удалять записи – все это без знания SQL – с помощью интерфейса программы. Более того, программа будет отображать SQL-запросы, так что вы с ее помощью сможете не только управлять базой

данных, но и учить SQL. Конечно, программа позволяет ввести SQL-запрос пользователю, что позволит попрактиковаться или даже проверить запрос перед его помещением в сценарий. phpMyAdmin установлена на очень многих хостингах, поэтому в большинстве случаев ссылка на нее будет в панели управления хостингом.

4. Язык программирования с поддержкой **SQL** – поддержка SQL и подключения к БД имеется во многих языках программирования. Вы можете использовать предоставляемый API для подключения к БД, передачи запроса и обработки результата выполнения SQL-запроса.

Все способы равнозначны и вы можете использовать любой из них. Понятно, что в нашем случае для отладки запросов вы будете использовать phpMyAdmin, как и для начального создания базы данных. А вот обычные запросы вроде выборки, вставки вы будете передавать через PHP.

Принцип в этом случае такой:

1. С помощью PHP-сценария вы передаете запрос, например, SELECT * FROM library;
2. Получаете результат запроса – список записей таблицы **library**, если в ней есть записи.
3. Обрабатываете и отображаете результат пользователю, например, оформляете в виде визуальной таблицы полученные записи базы данных.

Создание базы данных

Когда у вас есть права на создание базы данных, то создать новую базу данных можно запросом CREATE DATABASE <название новой БД>. Но будем реалистами. В большинстве случаев у вас будет хостинг, а не виртуальный/физический сервер, а на хостинге либо база данных создается автоматически при регистрации хостинга (и эта одна база данных представляется в аренду пользователю) или же вам предоставляется возможность создать ограниченное количество баз данных (3-5) через панель управле-

ния хостингом. Но в любом случае ввести запрос CREATE DATABASE вам никто не позволит.

Как правило, в панели управления хостингом есть раздел Базы данных (рис. 6.1). В этом разделе есть команда Создание новой базы данных (рис. 6.2).

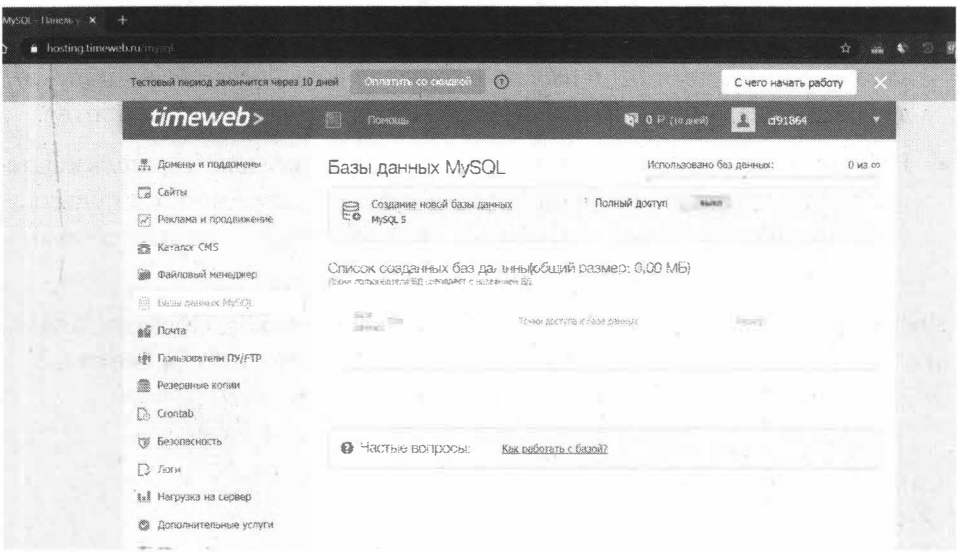


Рис. 6.1. Список баз данных пуст

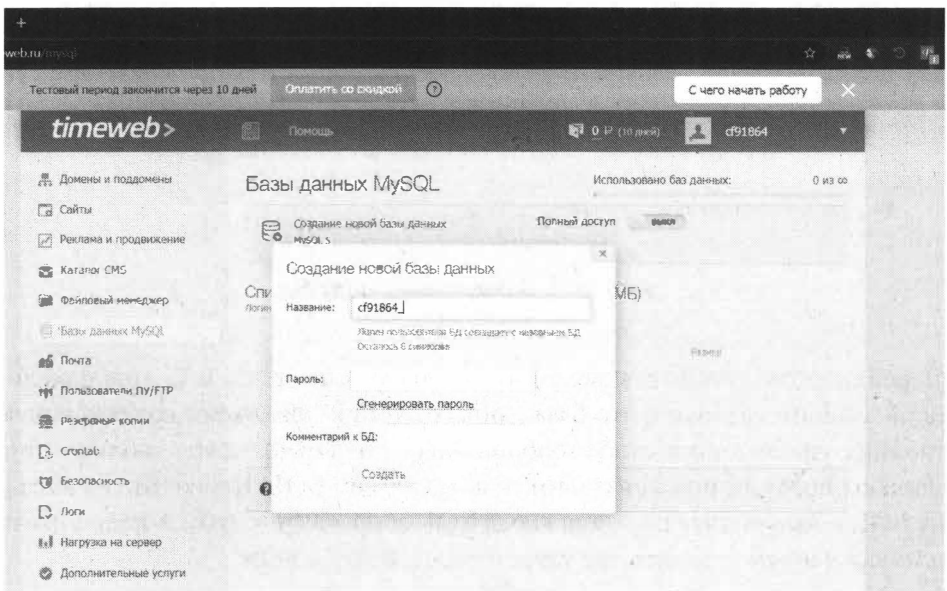


Рис. 6.2. Создание новой базы данных

Чтобы установить подключение с сервером баз данных в PHP-сценарии вам нужно указать следующую информацию:

- Имя сервера – в 99% случаев это будет localhost. В оставшемся 1% имя сервера нужно уточнить в службе поддержки или вы его должны знать сами, если вы сами настраивали MySQL-сервер.
- Имя базы данных и имя пользователя – в некоторых случаях, как показано на рис. 6.2, имя БД и имя пользователя совпадают. Но иногда – это два разных параметра. Все зависит от панели управления хостингом.
- Пароль пользователя – задается при создании базы данных/пользователя. Большинство панелей управления создают базу данных и пользователя, предназначенного для работы с этой БД. При создании пользователя задается и пароль.

После создания база данных появится в списке (рис. 6.3). Обратите внимание: под каждой БД есть ссылка на phpMyAdmin – по этой ссылке откроется phpMyAdmin для работы с выбранной базой данных.

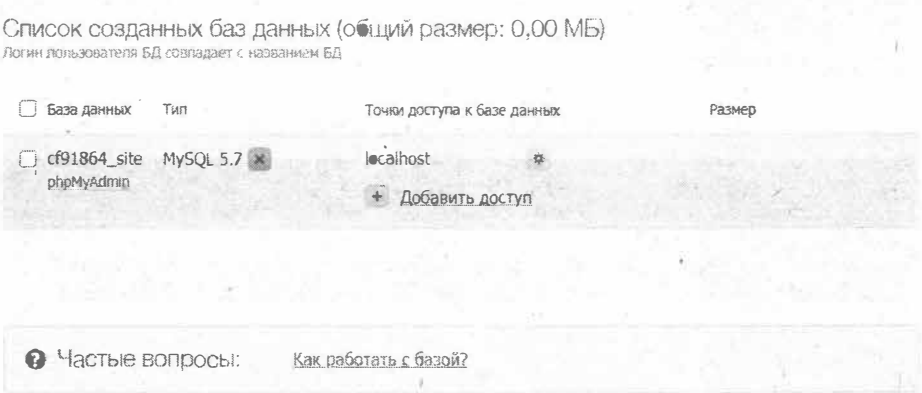


Рис. 6.3. База данных создана

Перейдите по этой ссылке. Как показано на рис. 6.4, приложение phpMyAdmin сообщает, что база данных пуста и предлагает создать новую таблицу, указав количество столбцов. Мы не будем создавать таблицу с помощью phpMyAdmin, а создадим ее посредством SQL. Перейдите на вкладку SQL – вы увидите поле для ввода SQL-запроса (рис. 6.5). А какой именно запрос нужно вводить, вы узнаете далее в этой книге.

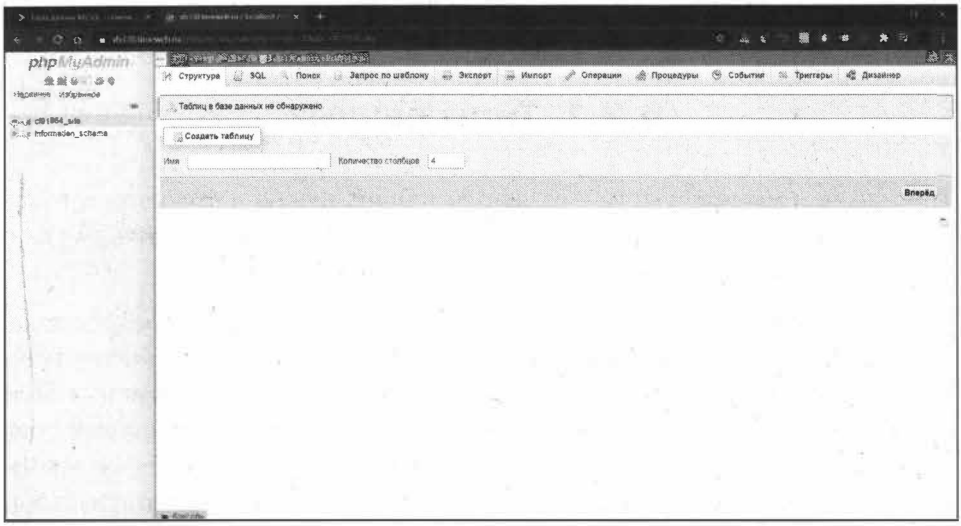


Рис. 6.4. Приложение phpMyAdmin

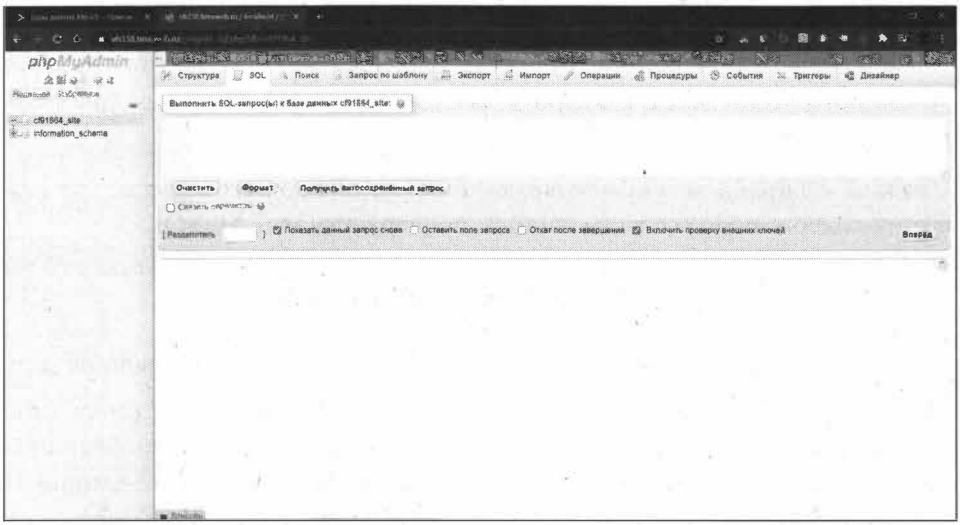


Рис. 6.5. Вкладка SQL

6.2. Создание таблиц

6.2.1. Типы данных

Прежде, чем мы приступим к созданию таблиц, нужно поговорить о типах данных – ведь при создании таблицы вам нужно будет определить тип данных для каждого поля таблицы.

У каждого поля таблицы есть свой тип данных. Тип данных определяет, какие данные могут храниться в этом поле. Все значения в этом поле будут одинакового типа. Например, если взять нашу таблицу *customers* и ее поле *cid*, то, скорее всего, оно должно быть целого типа (INT), поскольку оно содержит числовые идентификаторы клиентов. Хотя в последнее время, как уже было отмечено, часто используются строковые идентификаторы UUID, позволяющие более точно идентифицировать клиента. Поле **email**, наоборот, должно быть строкой, поскольку электронный адрес нельзя представить в виде числа.

Стандарт ANSI поддерживает только текстовый и числовой типы, но многие коммерческие (и не только) СУБД поддерживают другие типы данных. Кроме того, типы DATE (дата) и TIME (время) также являются практически стандартными.

Сначала мы пройдемся по основным типам данных, а потом рассмотрим таблицу, содержащую все типы данных, поддерживаемые MySQL.

Символьные данные

Символьные данные хранят строки фиксированной или переменной длины. Разница между этими строками заключается в том, что строки фиксированной длины дополняются справа дополнительными пробелами, а строки произвольной длины – нет. Например, вы создали поле длиной 10 символов и добавили в него значение Hello (5 символов). При использовании фиксированного типа данных реальное значение будет "Hello " (дополнительные 5 пробелов справа).

Для описания символьных данных используются следующие типы:

```
CHAR(20)  /* строка фиксированной длины */
VARCHAR(20)  /* строка переменной длины */
```

Чаще всего используется тип VARCHAR. В отличие от CHAR хранимая строка будет занимать именно столько места, сколько необходимо. Например, если определена длина в 10 символов, но в столбец сохраняется строка в 6 символов, то хранимая строка так и будет занимать 6 символов плюс дополнительный байт, который хранит длину строки.

Начиная с MySQL 5.6, типы CHAR и VARCHAR по умолчанию используют кодировку UTF-8, которая позволяет использовать до 3 байт для хранения символа в зависимости от языка (для многих европейских языков по 1 байту на символ, для ряда восточно-европейских и ближневосточных - 2 байта, а для китайского, японского, корейского - по 3 байта на символ).

Стандартом предусмотрено, что максимальная длина строки для типов CHAR и VARCHAR составляет 255 символов, но в MySQL все немного не так:

- Максимальная длина типа CHAR – 255 символов.
- Максимальная длина типа VARCHAR – 65535 символов.

Однако рекомендуется все же для хранения длинных строк использовать не тип VARCHAR, а другие типы, специально предназначенные для этого:

- TEXT - текст длиной до 65 КБ.
- MEDIUMTEXT- позволяет хранить текст длиной до 16 МБ.
- LARGETEXT - текст длиной до 4 ГБ.

Да, ничто не помешает вам создать поле VARCHAR с длиной строки 60000, но такая длина строки может не поддерживаться другой СУБД. Если вам понадобится перенести БД в другую СУБД, могут возникнуть проблемы. Поэтому для такого текста лучше использовать тип TEXT.

Выбирая тот или иной текстовый тип, необходимо помнить следующее:

- Если размер данных, загружаемых в текстовый столбец, превышает максимальный размер для этого типа, не поместившиеся данные отсекаются.
- В отличие от столбца типа *varchar*, при загрузке данных в такой столбец пробелы в конце строки не удаляются.
- При использовании столбцов типа *text* для сортировки или группировки используются только первые 1024 байта, хотя при

необходимости это ограничивающее значение можно увеличить.

- Разные текстовые типы присущи исключительно MySQL. У MS SQL Server для больших символьных данных есть только один тип text, а в DB2 и Oracle применяется тип данных под названием clob (Character Large Object, большой символьный объект).

При создании столбца для данных произвольного формата, например, столбца comment (комментарий к заказу) нужно выбрать тип TEXT, чтобы не ограничивать пользователя в его мемуарах.

> **Примечание. В Oracle Database допускаются столбцы char до 2000 байт и varchar до 4000 байт. SQL Server может оперировать данными типа char и varchar размером до 8000 байт. Поэтому при работе с Oracle или SQL Server потребность в текстовых типах данных меньше, чем при работе с MySQL.**

В языках, использующих латинский алфавит, например в английском, довольно мало символов, то есть каждый символ хранится как один байт. В других языках, таких как японский и корейский, много символов. Таким образом, в них для хранения одного символа требуется несколько байт. Поэтому такие наборы символов называют многобайтовыми наборами символов (multibyte character sets) или попросту кодировками. Узнать, сколько байтов занимает один символ в той или иной кодировке, можно с помощью SQL-запроса SHOW CHARACTER SET.

Вы можете задать кодировку отдельно для каждого столбца при создании таблицы, например:

```
VARCHAR(200) CHARACTER SET utf8
```

Но это не очень удобно, поэтому в MySQL позволяется задавать кодировку для всей базы данных сразу:

```
CREATE DATABASE test CHARACTER SET utf8;
```

Если кодировка не объявлена, то по умолчанию используется кодировка, указанная в конфигурационном файле MySQL-сервера.

Числовые данные

Числовые данные очень часто используются в базах данных – номер клиента, номер позиции товара в корзине, количество товара и т.д. Все это примеры использования числовых данных. Наиболее часто используются следующие числовые типы данных:

- INT – позволяет хранить значения от -2 147 483 648 до 2 147 483 647 или от 0 до 4 294 967 295, если он объявлен как беззначный UNSIGNED.
- SMALLINT – хранит традиционное для 32-битных систем значение целого типа в диапазоне от -32 768 до 32 767 или от 0 до 65535 для UNSIGNED.
- FLOAT – хранит очень большое вещественное значение. Для подсчета заработанных миллиардов его вполне хватит.

Тип FLOAT позволяет указать точность (**precision**, общее допустимое число разрядов, как справа, так и слева от десятичной точки) и масштаб (**scale**, допустимое число разрядов справа от десятичной точки), но эти параметры не являются обязательными. Задавая точность и масштаб для столбца, имеющего тип с плавающей точкой, необходимо помнить, что сохраняемые в нем данные будут округляться, если число разрядов в них превысит заданный масштаб и/или точность. Например, столбец, определенный как float(4,2), будет сохранять всего четыре разряда, два слева и два справа от десятичной точки. Поэтому с такими числами, как 27,44 и 8,19, будет все в порядке, а вот число 17,8675 будет округлено до 17,87, а число 178,5 будет округлено (грубо) до 99,99 – самое большое число, которое может быть сохранено в этом столбце.

Временные данные

Третий часто используемый тип данных после строк и чисел – дата и время:

- Дата оформления заказа.
- Желаемая дата и время доставки заказа.

- Планируемая дата отгрузки товара.
- Дата рождения клиента или сотрудника.

Примеров для применения типов DATE и TIME можно привести очень много. В MySQL есть типы данных на все случаи жизни:

- DATE – хранит дату в формате YYYY-MM-DD.
- DATETIME – хранит дату и время в формате YYYY-MM-DD HH:MI:SS
- TIMESTAMP – формат такой же, как и у DATETIME, но принимаемые значения отличаются – от 1970-01-01 00:00:00 до 2037-12-31 23:59:59, а DATETIME практически не ограничивает эти значения – от 1000-01-01 00:00:00 до 9999-12-31 23:59:59.
- YEAR – используется для хранения года в формате YYYY, принимает значения от 1901 до 2155.
- TIME – время в формате HHH:MI:SS, допустимые значения от -838:59:59 до 838:59:59.

Вот как правильно выбрать тип временных данных:

Для хранения предполагаемой даты доставки заказа покупателю и даты рождения сотрудника использовались бы столбцы типа **date**, поскольку знать точное время рождения человека необязательно, а спланировать будущую доставку с точностью до секунды нереально.

- Для хранения информации о фактической доставке заказа покупателю использовался бы тип **datetime** (дата и время), поскольку важно отследить не только дату, но и точное время доставки.
- Столбец, отслеживающий время последнего изменения пользователем определенной строки таблицы, использовал бы тип **timestamp** (временная метка). Этот тип содержит ту же информацию, что и тип **datetime** (год, месяц, день, час, минуту, секунду), но при добавлении или изменении строки таблицы сервер MySQL автоматически заполнит столбец **timestamp** текущими значениями даты/времени.
- Столбец для хранения только данных о годе использовал бы тип **year** (год).

- Столбцы, содержащие данные о временном интервале, необходимом для выполнения задачи, использовали бы тип **time** (время). Этому типу данных не нужно хранить компонент даты – это сбивало бы с толку, поскольку интерес представляет только количество часов/минут/секунд, необходимое для выполнения задания. Эту информацию можно было бы получить, найдя разность значений из двух столбцов типа **datetime** (первый хранит дату/время начала выполнения задания, а второй – дату/время его завершения). Но проще использовать один столбец **time**.

Сводная таблица типов данных

В таблице 6.1 представлены типы полей, поддерживаемые популярной СУБД MySQL. В описаниях используются следующие обозначения:

- M - указывает максимальный размер вывода. Максимально допустимый размер вывода составляет 255 символов.
- D - употребляется для типов данных с плавающей точкой и указывает количество разрядов, следующих за десятичной точкой. Максимально возможная величина составляет 30 разрядов, но не может быть больше, чем M-2.

Квадратные скобки ('[' и ']') указывают для типа данных группы необязательных признаков.

При включенном атрибуте ZEROFILL включает заполнение нулями до определенного количества мест. Например, представим, что у нас есть поле, объявленное как INT (4) ZEROFILL и в нем есть значение 5. При выводе это значение будет отображаться как 0005.

Заметьте, что если для столбца указать параметр ZEROFILL, то MySQL будет автоматически добавлять в этот столбец атрибут UNSIGNED. Атрибут UNSIGNED означает, что у числовых значений не будет знака (то есть все значения будут положительными).

Предупреждение: следует помнить, что при выполнении вычитания между числовыми величинами, одна из которых относится к типу UNSIGNED, результат будет беззнаковым!

Таблица 6.1. Типы данных

Тип данных	Описание
Целые типы	
TINYINT[(M)] [UNSIGNED] [ZEROFILL]	Диапазон -128...+127. Диапазон без знака 0..255. Подходит для очень малых чисел. Занимает 1 байт.
BIT, BOOL, BOOLEAN	Синоним для TINYINT(1) и может хранить два значения 0 и 1. Данный тип может принимать встроенные константы TRUE (значение 1) и FALSE (значение 0).
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]	Диапазон -32768...+32767. Без знака - 0..65535. Занимает 2 байта.
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]	Диапазон -8 388 608... +8 3888 607. Без знака 0..16 777 215. Занимает 3 байта
INT INT[(M)] [UNSIGNED] [ZEROFILL]	Диапазон -2 147 483 648... - 2 147 483 647 Без знака 0..4 294 967 295. Занимает 4 байта.
INTEGER	Синоним для INT
BIGINT[(M)] [UNSIGNED] [ZEROFILL]	Диапазон -9 223 372 036 854 775 808... ±9 223 372 036 854 775 807 Без знака - 0 до 18446744073709551615 Все арифметические операции выполняются с использованием значений BIGINT или DOUBLE со знаком, так что не следует использовать беззнаковые целые числа больше чем 9223372036854775807 (63 бита), кроме операций, выполняемых логическими функциями. Размер – 8 байтов.

Вещественные типы	
FLOAT (точность) [UNSIGNED] [ZEROFILL]	Число с плавающей точкой. Атрибут точности может иметь значение <=24 для числа с плавающей точкой обычной (одинарной) точности и между 25 и 53 - для числа с плавающей точкой удвоенной точности Обеспечивает небольшую точность. Занимает 4 байта.
FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]	Малое число с плавающей точкой обычной точности. Допустимые значения: от -3,402823466E+38 до -1,175494351E-38, 0, и от 1,175494351E-38 до 3,402823466E+38. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D - количество разрядов, следующих за десятичной точкой. Обозначение FLOAT без указания аргументов или запись вида FLOAT(X), где X <=24 справедливы для числа с плавающей точкой обычной точности. Занимает 4 байта.
DOUBLE [(M,D)] [UNSIGNED] [ZEROFILL]	Обеспечивает двойную точность по сравнению с FLOAT. Хранит дробные числа с плавающей точкой двойной точности от -1.7976 * 10308 до 1.7976 * 10308, занимает 8 байт. Также может принимать форму DOUBLE(M,D), где M - общее количество цифр, а D - количество цифр после запятой. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D - количество разрядов, следующих за десятичной точкой. Обозначение DOUBLE без указания аргументов или запись вида FLOAT(X), где 25 <= X <= 53 справедливы для числа с плавающей точкой двойной точности.
DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL] , REAL[(M,D)] [UNSIGNED] [ZEROFILL]	Тоже, что и DOUBLE
DECIMAL [(M[,D])] [UNSIGNED] [ZEROFILL]	Дробное число, которое хранится в виде строки
DEC, NUMERIC	Синоним для DECIMAL

Строки	
[NATIONAL] CHAR(M) [BINARY]	Строка фиксированной длины, при хранении всегда дополняется пробелами в конце строки до заданного размера. Диапазон аргумента M составляет от 0 до 255 символов. Концевые пробелы удаляются при выводе значения. Если не задан атрибут чувствительности к регистру BINARY, то величины CHAR сортируются и сравниваются как независимые от регистра в соответствии с установленным по умолчанию алфавитом. Атрибут NATIONAL CHAR (или его эквивалентная краткая форма NCHAR) представляет собой принятый в ANSI SQL способ указания, что в столбце CHAR должен использоваться установленный по умолчанию набор символов (CHARACTER). В MySQL это принято по умолчанию. CHAR является сокращением от CHARACTER. MySQL позволяет создавать столбец типа CHAR(0).
CHAR	Синоним для CHAR(1)
[NATIONAL] VARCHAR(M) [BINARY]	Строка переменной длины. Примечание: концевые пробелы удаляются при сохранении значения (в этом заключается отличие от спецификации ANSI SQL). Диапазон аргумента M составляет от 0 до 255 символов. Если не задан атрибут чувствительности к регистру BINARY, то величины VARCHAR сортируются и сравниваются как независимые от регистра.
TINYTEXT	Максимальная длина строки — 255 символов
TEXT	Максимальная длина строки — 65535 символов (64 Кб)
MEDIUMTEXT	Максимальная длина строки — 16 777 215 символов
LONGTEXT	Максимальная длина строки — 4 294 967 295 символов

Бинарные типы	
TINYBLOB	Максимум — 255 байтов. Значение типа BLOB (бинарные данные).
BLOB	Максимум — 65535 байтов
MEDIUMBLOB	Максимум — 16 777 215 байтов
LONGBLOB	Максимум — 4 294 967 295 байтов
Дата и время	
DATE	Дата в формате ГГГГ-ММ-ДД. Поддерживается интервал от '1000-01-01' до '9999-12-31'
TIME	Время в формате ЧЧ:ММ:СС
TIMESTAMP	Дата и время в формате timestamp, однако при отображении оно выводится в виде ГГГГММДДЧЧММСС. Интервал от '1970-01-01 00:00:00' до некоторого значения времени в 2037 году. Столбец TIMESTAMP полезен для записи даты и времени при выполнении операций INSERT или UPDATE, так как при этом автоматически вносятся значения даты и времени самой последней операции, если эти величины не введены программой.
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС. Поддерживается интервал от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'.
YEAR[(2|4)]	Год в двухзначном или четырехзначном форматах (по умолчанию формат четырехзначный). Допустимы следующие значения: с 1901 по 2155, 0000 для четырехзначного формата года и 1970-2069 при использовании двухзначного формата (70-69). MySQL выводит значения YEAR в формате YYYY, но можно задавать значения в столбце YEAR, используя как строки, так и числа

Другие типы данных	
ENUM('значение1','значение2',...)	Перечисление. Перечисляемый тип данных. Объект строки может иметь только одно значение, выбранное из заданного списка величин 'значение1', 'значение2', ..., NULL или специальная величина ошибки "". Список ENUM может содержать максимум 65535 различных величин.
SET('значение1','значение2',...)	Набор. Объект строки может иметь ноль или более значений, каждое из которых должно быть выбрано из заданного списка величин 'значение1', 'значение2', ... Список SET может содержать максимум 64 элемента.

6.2.2. Оператор CREATE

Оператор CREATE может использоваться для создания различных объектов. Мы же будем использовать его пока только для создания таблиц:

```
CREATE TABLE ИМЯ_ТАБЛИЦЫ
(
  ПОЛЕ1       ТИП [<РАЗМЕР>]       ПАРАМЕТРЫ,
  . . .
  ПОЛЕN       ТИП [<РАЗМЕР>]       ПАРАМЕТРЫ
)
```

Поскольку пробелы используются для разделения отдельных частей команд в SQL, их нельзя использовать как часть имени таблицы (либо как часть какого-либо другого объекта). Если нужен пробел, тогда обычно используют символ подчеркивания (_).

Значение аргумента размера зависит от типа данных. Если он не указывается, то система установит значение автоматически. Тип данных, для которого обязательно следует указывать размер, - это CHAR (он же VARCHAR в некоторых СУБД). В данном случае аргумент размера - целое число, задающее максимальное число символов, которые могут содержаться в поле.

Помните, что таблицы принадлежат создавшему их пользователю. Работать с таблицей может или пользователь *root* (у которого есть максимальные

права) или пользователь, которому разрешено работать с таблицей. Далее в этой книге мы поговорим еще о правах пользователей.

Рассмотрим команду создания таблицы *library_books*, содержащую информацию о книгах:

```
CREATE TABLE IF NOT EXISTS library_books (
  id VARCHAR(40) NOT NULL,
  title VARCHAR(200) NOT NULL,
  cover VARCHAR(100) DEFAULT NULL,
  author VARCHAR(100) DEFAULT NULL,
  active INT DEFAULT 0,
  folder VARCHAR(255) DEFAULT NULL,
  url VARCHAR(255) NOT NULL,
  seokeys VARCHAR(255) NOT NULL,
  seodescr VARCHAR(255) NOT NULL,
  PRIMARY KEY (id)
) DEFAULT CHARSET=utf8mb4;
```

Таблица 6.2. Описание полей таблицы library_books

Поле	Назначение
id	UUID книги, строка переменной длины максимального размера в 40 символов, не может быть равна NULL
title	Название книги, максимальный размер – 200 символов.
cover	Файл обложки книги (обложка хранится в папке книги), максимальный размер – 100 символов, по умолчанию – NULL (пустое значение)
author	Автор книги, максимальный размер – 100 символов, по умолчанию – NULL (пустое значение)
active	Флаг видимости книги (1 – видима, 0 – не отображается во фронте, при попытке открыть по старому адресу – 404). Значение по умолчанию – 0.
folder	Папка с картинками книги
url	URL книги (источник)
seokeys	Ключевые слова для SEO
seodescr	META-описание для SEO

Порядок столбцов в определении таблицы существенен, он определяет порядок, в котором задаются значения элементов строк. Определения столбцов не должны задаваться в отдельных строках, но они должны разделяться запятыми.

Последняя запись PRIMARY KEY(id) означает, что поле **id** будет служить первичным ключом и все его значения должны быть уникальны, что вполне логично – артикул должен однозначно идентифицировать товар.

Обратите внимание: мы используем символьный идентификатор (поле **id**) для книг. В некоторых случаях можно использовать числовые идентификаторы. Для таких идентификаторов можно использовать модификатор AUTO_INCREMENT, например:

```
id INT AUTO_INCREMENT
```

В данном случае создается поле-счетчик. При вставке записи следующий идентификатор будет на 1 больше, чем идентификатор в предыдущей строке таблицы. AUTO_INCREMENT позволит нам не заботиться о числовых идентификаторах – при вставке записи мы можем просто указывать 0, а база данных уже сама вычислит следующий номер в этой таблице.

6.3. Основные SQL-запросы

Сейчас мы вкратце рассмотрим четыре основных запроса – SELECT, INSERT, UPDATE и DELETE. Запрос SELECT позволяет выбрать информацию из базы данных, INSERT – добавить запись, UPDATE – обновить запись/записи, а DELETE – удалить запись/записи. Цель этой главы – не полное рассмотрение данных четырех запросов, а всего лишь краткое знакомство с ними.

Начнем мы с оператора INSERT, поскольку на данном этапе наша база данных пуста и нам нужно добавить данные в таблицы.

Примечание. Данные заносятся и удаляются с помощью трех команд языка манипулирования данными (Data Manipulation Language - DML): INSERT (вставить), UPDATE (обновить) и DELETE (удалить). В SQL их часто называют командами обновления (update commands).

6.3.1. Оператор INSERT – вставка записей в таблицу

Сокращенная форма оператора INSERT выглядит так:

```
INSERT INTO <название таблицы> [(список полей)]
VALUES (список_значений);
```

Значения при использовании этой формы должны точно соответствовать структуре таблицы, например, если первое поле в таблице **products** типа INT, то и первое значение должно быть типа INT. Строковые и временные значения заключаются в кавычки, числовые можно использовать без кавычек.

Добавим несколько значений в таблицу *library_books*:

```
INSERT INTO library_books VALUES
('2be8d998-0c82-11eb-93fc-ac1f6bbd4340',
'Основы Slackware Linux. Официальный учебник.', '../books/
osnovi_slack_are_linux__ofitsial_nii_uchebnik_/cover.jpg',
'Дэвид Кэнтрелл, Логэн Джонсон, Крис Люменс', 1, 'osnovi_
slack_are_linux__ofitsial_nii_uchebnik_', 'osnovi_slack_are_
linux__ofitsial_nii_uchebnik_', 'Вашему вниманию предлагается
русскоязычный перевод руководства', 'Вашему вниманию
предлагается русскоязычный перевод руководства ')
```

Мы использовали сокращенную форму оператора INSERT – мы не указывали список полей. Если же мы будем указывать список полей, то оператор примет вид:

```
INSERT INTO library_books (id, title, cover, author,
active, folder, url, seokeys, seodescr)…
```

Полная форма позволяет изменить порядок следования значений в блоке VALUES, а также не указывать некоторые значения. Если поле не указано в списке полей, то его значение тоже не указывается. База данных вместо этого использует значение по умолчанию. Если таковое не задано, при выполнении запроса вы получите сообщение об ошибке.

6.3.2. Оператор SELECT – выбор данных из таблицы

Хотя в названии этого раздела и есть фраза "выбор данных из таблицы", вы должны понимать, что SELECT можно использовать для выбора любых других значений из базы данных, например, для выбора результатов встроенных функций. Если вы забыли, который час и вообще какое сегодня число, вы можете ввести оператор:

```
SELECT now();
```

Данный запрос отобразит текущую дату и время.

Впрочем, мы отклонились от темы. Мы только что добавили записи в таблицы и хотим их просмотреть.

Оператор SELECT выбирает записи, которые соответствуют определенным условиям. Синтаксис оператора следующий:

```
SELECT [DISTINCT|ALL] {*| [полеl AS псевдоним]  [,...,
полеN AS
псевдоним]}
FROM Имя_таблицы1 [,..., Имя_таблицыN]
[WHERE условие]
[GROUP BY список полей]  [HAVING условие]
[ORDER BY список полей]
[LIMIT к-во]
```

Параметр WHERE – используется для определения, какие строки должны быть выбраны или включены в GROUP BY.

GROUP BY – необязательный параметр оператора SELECT, используется для группировки строк по результатам агрегатных функций (MAX, SUM, AVG и др.).

> **Внимание! Нужно, чтобы в SELECT были заданы только требуемые в выходном потоке столбцы, перечисленные в GROUP BY и/или агрегированные значения. Распространенная ошибка – указание в SELECT столбца, пропущенного в GROUP BY.**

HAVING — необязательный параметр оператора SELECT для указания условия на результат агрегатных функций.

Параметр HAVING <условия> аналогичен WHERE <условия> за исключением того, что строки отбираются не по значениям столбцов, а строятся из значений столбцов, указанных в GROUP BY, и значений агрегатных функций, вычисленных для каждой группы, образованной GROUP BY.

Если параметр GROUP BY в SELECT не задан, HAVING применяется ко всем строкам таблицы, полностью дублируя WHERE (правда, это допускается не во всех реализациях стандарта SQL).

На первый взгляд синтаксис слишком сложный, но на практике все намного проще. В большинстве случаев оператор SELECT можно сократить до вида:

```
SELECT [список полей|*] FROM имя_таблицы
[WHERE условие]
[ORDER BY поля]
```

Разберемся, что отброшено, а что осталось. Все лишнее в самом начале оператора мы отбросили и оставили список полей или же *, если нужно выбрать все поля в порядке, определенном при создании таблицы. Если же задан список полей, то вы можете переопределить их порядок при выводе таблицы – далее будет показано, как это сделать.

Предложение WHERE используется для определения условия, а ORDER BY - для сортировки.

Давайте рассмотрим несколько примеров. Попробуем выбрать названия всех статей:

```
SELECT title FROM library;
```

Результат выполнения этого запроса показан на рис. 6.6. Видно, что мы выбрали столбец title из таблицы library. Поскольку никакие условия не были заданы, то были отображены все записи из таблицы.

В операторе SELECT вы можете указывать столбцы в произвольном порядке. По умолчанию у нас столбцы определены в следующем порядке – id, catid, url, status, title, folder, image, author, link, content, shortdesrc, dt, tags, seokeys, seodescr, popular. Например, если вас интересуют только названия статей и их авторы, введите запрос:

```
SELECT title, author FROM library;
```

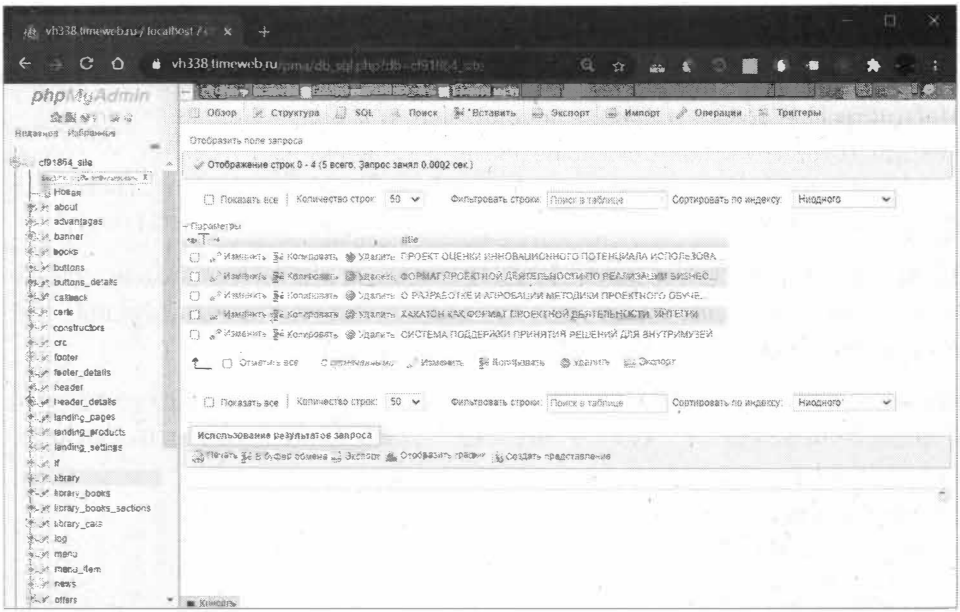


Рис. 6.6. Названия всех статей

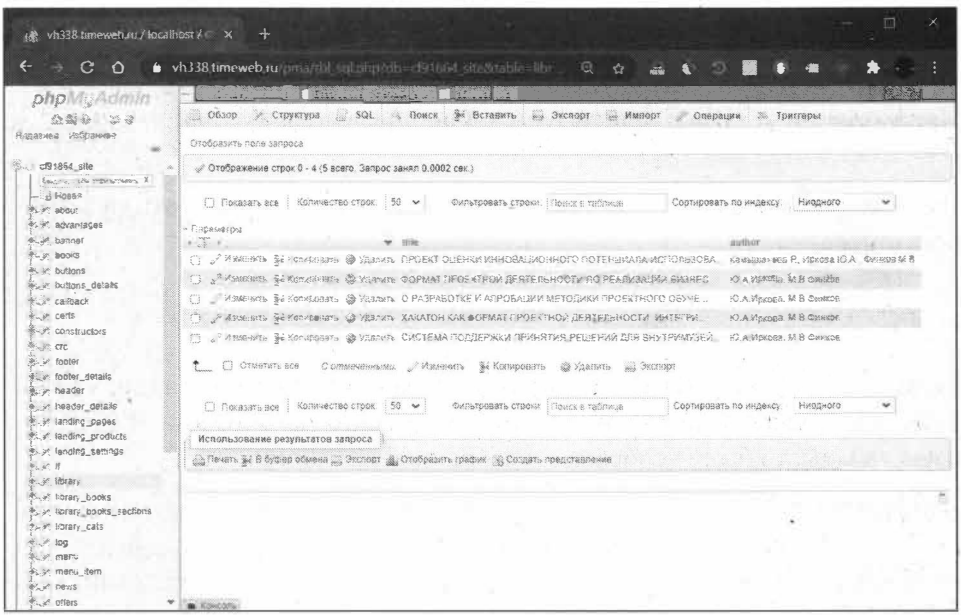


Рис. 6.7. Выборка полей в произвольном порядке

При желании можно вывести все поля сразу (рис. 6.8): для этого вместо списка полей укажите звездочку.

```
SELECT * FROM library;
```

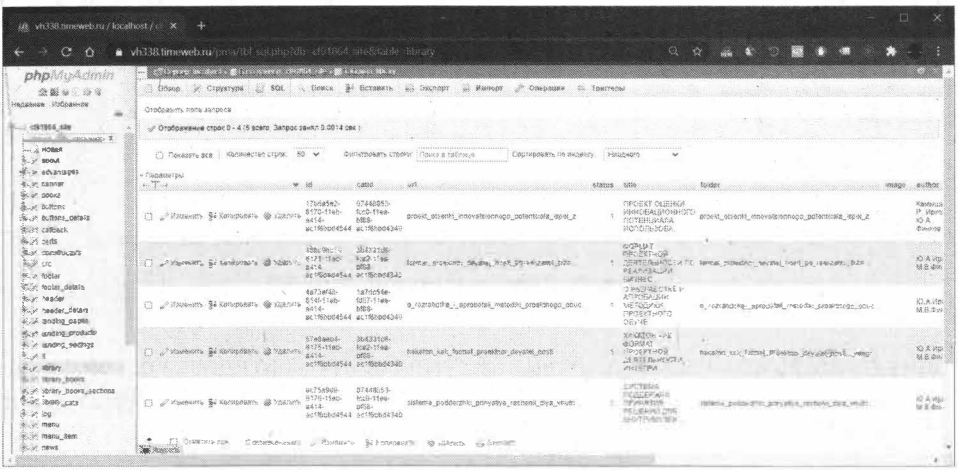


Рис. 6.8. Выборка всех полей и всех записей: вместо списка полей указан *, а какое-либо условие (WHERE) не определено, поэтому выводятся все записи

Параметр WHERE позволяет указать условие, определяющее, какие записи нужно выбрать. То есть будут выбраны все записи, удовлетворяющие заданному условию:

```
SELECT *
  FROM library
  WHERE url like '%format%'
```

На рис. 6.9 отображены все записи, где **url** содержит значение "format" (кстати, кавычки вы можете использовать, как двойные, так и одинарные – разницы никакой нет, главное, чтобы везде были одинаковыми).

По умолчанию, если задан первичный ключ, то сортировка записей осуществляется по этому полю в порядке возрастания (если это строка – в прямом алфавитном порядке). Изменить порядок сортировки можно с помощью ORDER BY. Параметр ORDER BY позволяет указать поле, по кото-

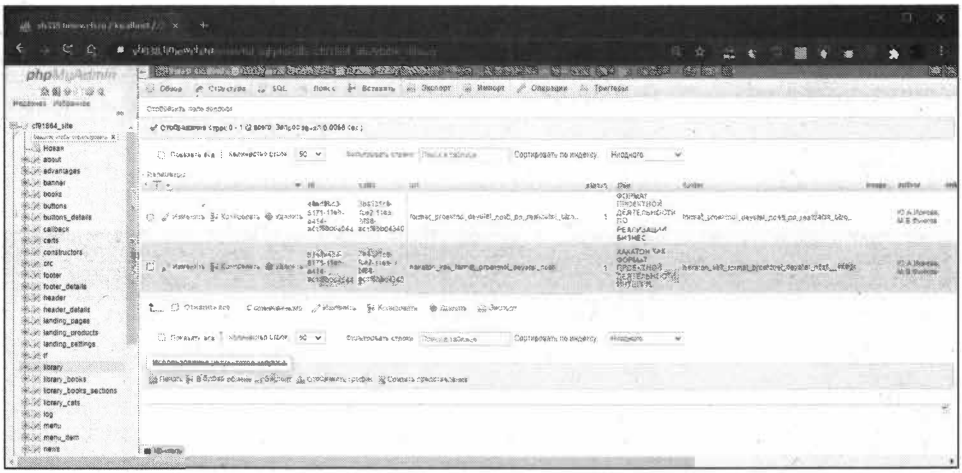


Рис. 6.9. Задано условие WHERE

рому будет выполнена сортировка. По умолчанию сортировка осуществляется по возрастанию (ASC), но вы можете указать и порядок по убыванию (DESC). Пример:

```
SELECT title
  FROM library
  ORDER BY dt DESC;
```

Здесь мы сортируем записи от дорогих к дешевым – типичная задача при разработке Интернет-магазина (рис. 6.10).

Попробуем задать условие и отсортировать:

```
SELECT title, url, dt
  FROM library
  WHERE url like '%format%;
  ORDER BY dt DESC;
```

Здесь мы выводим статьи, в URL которых есть слово 'format' и сортируем результат по дате.

Мы рассмотрели далеко не все возможности оператора SELECT. Но для первого знакомства вполне достаточно.

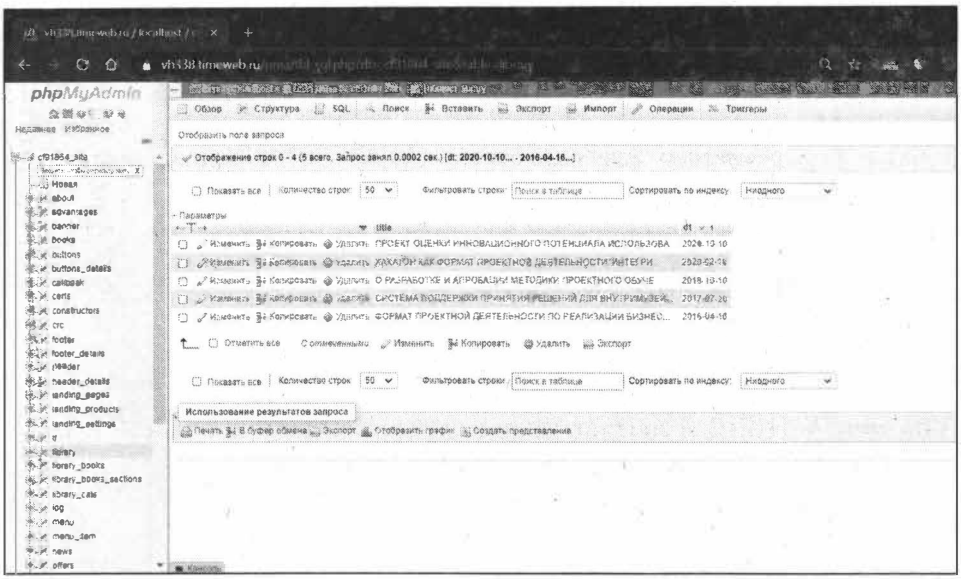


Рис. 6.10. Продукты отсортированы от дорогих к дешевым

6.3.3. Оператор UPDATE – обновление записи

Представим, что нам нужно обновить запись, допустим, товар с артикулом CIPH12-01 подорожал и теперь его цена составляет 17, а не 15 выдуманных фантиков. Для изменения записи используется оператор UPDATE. Синтаксис этого оператора выглядит так:

```
UPDATE <имя таблицы>
SET {<имя столбца> = {<выражение для вычисления значения
столбца>
| NULL
| DEFAULT},...}
[ {WHERE <предикат>}]
```

Итак, сначала нужно задать имя таблицы. Затем нужно задать список обновляемых столбцов. Список задается в формате:

```
имя_столбца = значение
```

Вместо конкретного значения допускаются два предопределенных значения:

- NULL – указывает, что поле не содержит значения (его можно использовать только при условии, что поле не определено как NOT NULL, то есть как поле которое не может принимать NULL-значения).

- DEFAULT – устанавливает заданное при создании таблицы значение по умолчанию, если, конечно, оно задано.

Примечание. Если поле определено, как NOT NULL, например:

CREATE TABLE user (

...

email varchar(100) NOT NULL

...)

то присвоить ему значение NULL не получится

Параметр WHERE позволяет, как и в случае с оператором SELECT, задать условие. Попробуем изменить цену товара CIPH12-01:

```
UPDATE products
SET price = 17
WHERE sku = "CIPH12-01";
```

Просмотрим содержимое таблицы **products** после выполнения этого оператора.

Редко когда приходится изменять цену одного товара. Скорее всего, приходится изменять цену сразу всей группы товаров. Согласитесь, вряд ли производитель поднял цену на черный чехол, а цену на зеленый оставил бы без изменений. Поэтому мы можем установить сразу цену для всей группы товаров, например:

```
UPDATE products
SET price = 20
WHERE vendor = "NoName";
```

При желании, например, на время акции, мы можем изменить цену сразу всех товаров. Представим, что сегодня – черная пятница, и мы хотим сделать скидку в размере 5% на все товары, которые у нас есть. Для этого нам нужно ввести запрос:

```
UPDATE products
SET price = price * 0.95;
```

Поскольку мы не задали условие WHERE, то столбец **price** будет обновлен у всех записей таблицы. Путем умножения на 0.95 мы снижаем цену на 5% (это обычная математика и никакого волшебства).

6.3.4. Оператор DELETE – удаление записей

Оператор DELETE удаляет строки из временных или постоянных базовых таблиц, представлений или курсоров, причем в двух последних случаях действие оператора распространяется на те базовые таблицы, из которых извлекались данные в эти представления или курсоры. Оператор удаления имеет простой синтаксис:

```
DELETE FROM <имя таблицы >
[WHERE <предикат>];
```

Если предложение WHERE отсутствует, удаляются все строки из таблицы или представления (представление должно быть обновляемым).

С оператором DELETE нужно быть осторожными. Если вы хотите удалить только одну запись, лучше использовать ограничитель LIMIT 1, даже если запись, как вам кажется, идентифицируется по уникальному ключу. Представим ситуацию, что кто-то изменил структуру таблицы и теперь ключ построен по другому полю, а поле sku, которое мы ранее использовали, как ключ, теперь самое обычное поле. Затем кто-то в него добавил несколько записей и для всех ошибочно указал артикул APPLE01. Если вы выполните запрос:

```
DELETE FROM products
WHERE sku = "APPLE01";
```

то вы удалите все записи, где столбец sku равен "APPLE01". Если вы зададите ограничение LIMIT 1, то вы удалите только одну запись, затем, при просмотре содержимого таблицы у вас будет возможность заметить, что что-то пошло не так:

```
DELETE FROM products
WHERE sku = "APPLE01"
LIMIT 1;
```

Удалить все записи из таблицы можно, не задавая параметра WHERE:

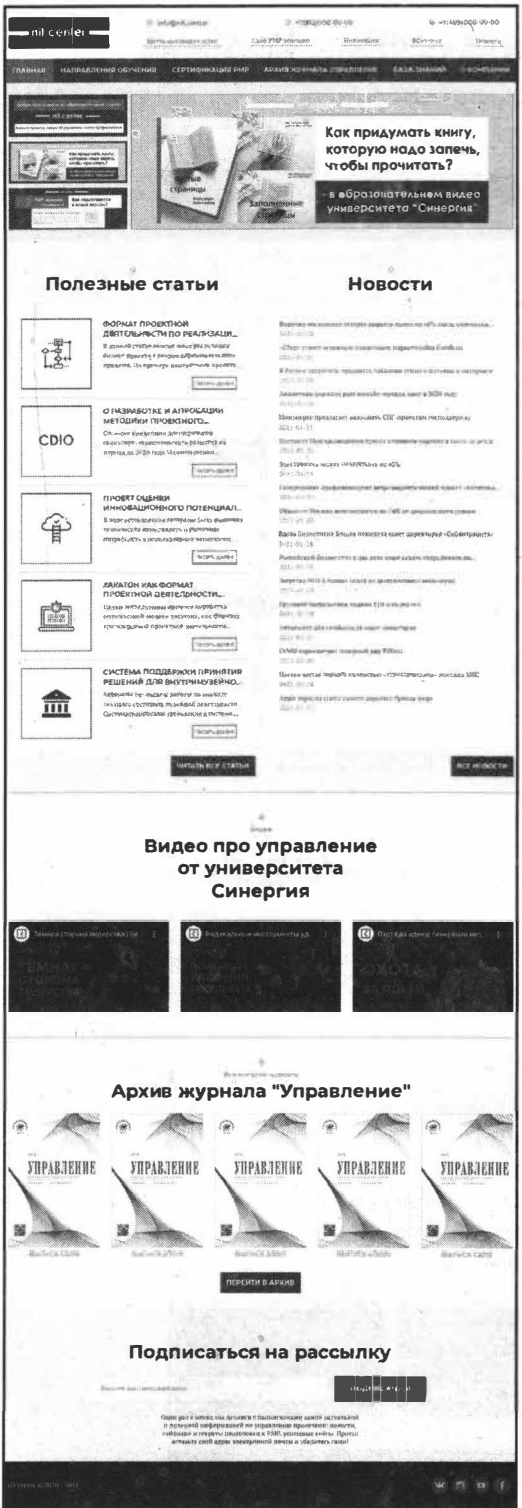
```
DELETE products;
```

В данном случае все продукты будут удалены.

Глава 7.

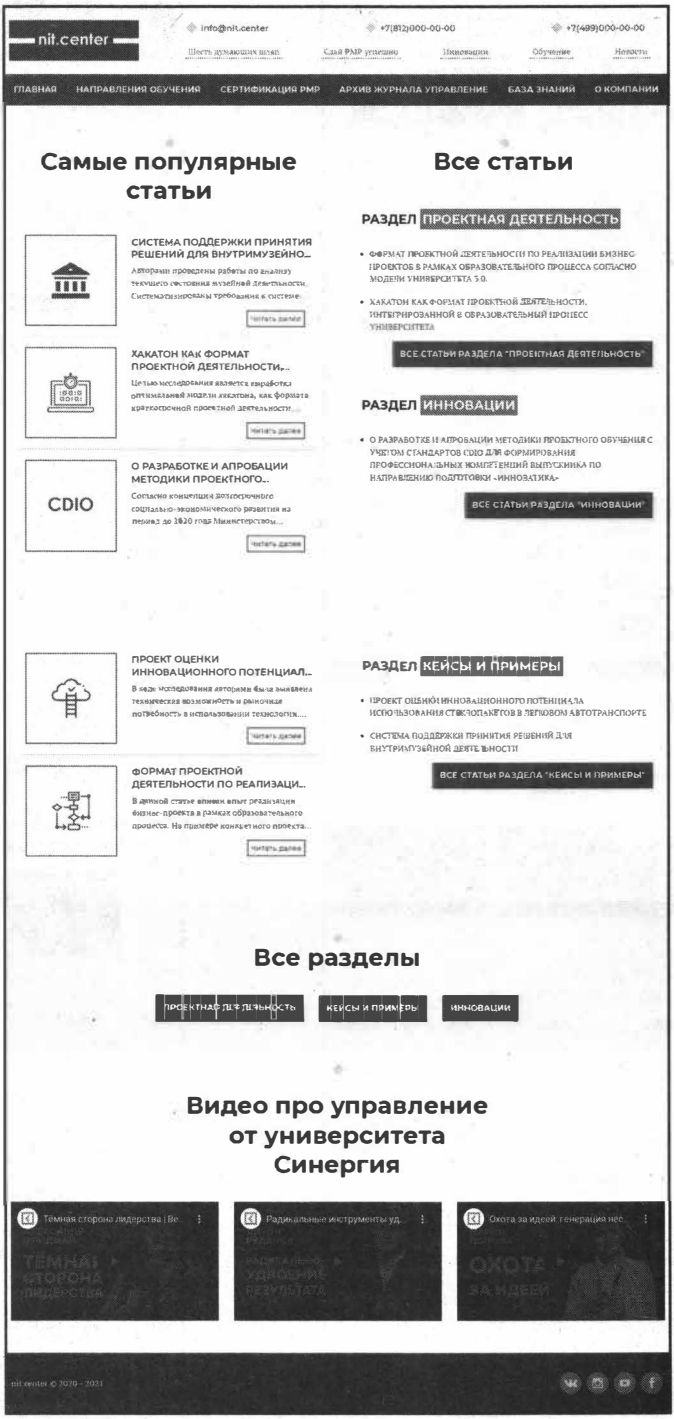
Практический пример. Разработка FrontEnd'а

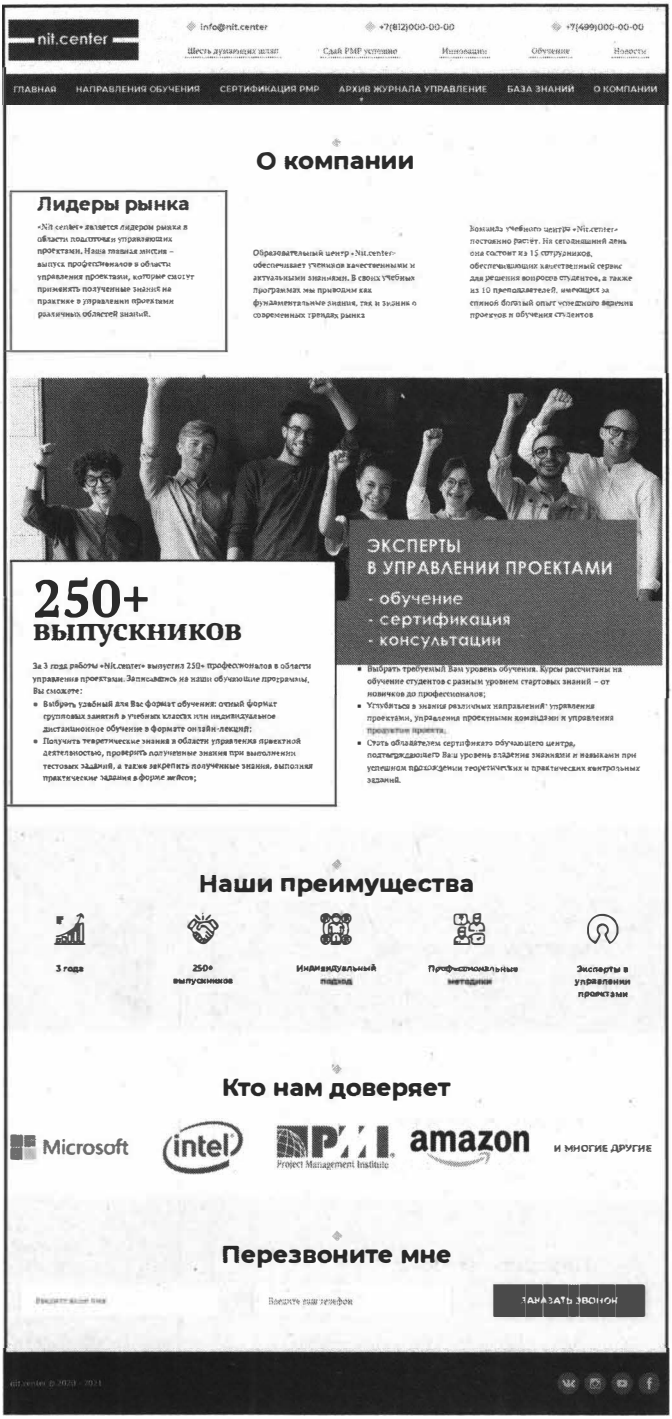
В данной и следующей главах будет рассмотрен пример образовательного интернет-портала, который условно размещен на домене nit.center. Все рассуждения будут даваться как-будто мы работаем на домене nit.center. Возможно, на этом (или другом) домене авторы разместят реальную демонстрацию примера. Уточнить информацию об этом вы сможете на сайте издательства nit.com.ru , на странице данной книги. Там же будет находиться программный код примера. В главе 9 рассказано, как его размещать на хостинге.



Главная страница

Главная страница (адаптировано под смартфон)


nit.center
info@nit.center
+7(812)000-00-00
+7(499)000-00-00
Шесть думающих шляп
Клей PMP успешно
Инновации
Обучение
Новости
ГЛАВНАЯ
НАПРАВЛЕНИЯ ОБУЧЕНИЯ
СЕРТИФИКАЦИЯ PMP
АРХИВ ЖУРНАЛА УПРАВЛЕНИЕ
БАЗА ЗНАНИЙ
О КОМПАНИИ
Самые популярные статьи
СИСТЕМА ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ ДЛЯ ВНУТРИМУЗЕЙНО...
Авторами проведены работы по анализу текущего состояния музейной деятельности. Систематизированы требования к системе.
читать далее
ХАКАТОН КАК ФОРМАТ ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ,...
Целью исследования является разработка оптимальной модели хакатона, как формата краткосрочной проектной деятельности....
читать далее
CDIO
О РАЗРАБОТКЕ И АПРОБАЦИИ МЕТОДИКИ ПРОЕКТНОГО...
Согласно концепции долгосрочного социально-экономического развития на период до 2020 года Министерство...
читать далее
ПРОЕКТ ОЦЕНКИ ИННОВАЦИОННОГО ПОТЕНЦИАЛ...
В ходе исследования авторами была выявлена техническая возможность и рыночная потребность в использовании технологии....
читать далее
ФОРМАТ ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ ПО РЕАЛИЗАЦИ...
В данной статье описан опыт реализации бизнес-проекта в рамках образовательного процесса. На примере конкретного проекта...
читать далее
Все статьи
РАЗДЕЛ ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ
ФОРМАТ ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ ПО РЕАЛИЗАЦИИ БИЗНЕС-ПРОЕКТОВ В РАМКАХ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА СОГЛАСНО МОДЕЛИ УНИВЕРСИТЕТА 5.0.
ХАКАТОН КАК ФОРМАТ ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ, ИНТЕГРИРОВАННОЙ В ОБРАЗОВАТЕЛЬНЫЙ ПРОЦЕСС УНИВЕРСИТЕТА
ВСЕ СТАТЬИ РАЗДЕЛА "ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ"
РАЗДЕЛ ИННОВАЦИИ
О РАЗРАБОТКЕ И АПРОБАЦИИ МЕТОДИКИ ПРОЕКТНОГО ОБУЧЕНИЯ С УЧЕТОМ СТАНДАРТОВ CDIO ДЛЯ ФОРМИРОВАНИЯ ПРОФЕССИОНАЛЬНЫХ КОМПЕТЕНЦИЙ ВЫПУСКНИКА ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ «ИННОВАТИКА»
ВСЕ СТАТЬИ РАЗДЕЛА "ИННОВАЦИИ"
РАЗДЕЛ КЕЙСЫ И ПРИМЕРЫ
ПРОЕКТ ОЦЕНКИ ИННОВАЦИОННОГО ПОТЕНЦИАЛА ИСПОЛЬЗОВАНИЯ СУПЕРКОНДЕНСАТОРОВ В ЛЕГКОВОМ АВТОТРАНСПОРТЕ
СИСТЕМА ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ ДЛЯ ВНУТРИМУЗЕЙНОЙ ДЕЯТЕЛЬНОСТИ
ВСЕ СТАТЬИ РАЗДЕЛА "КЕЙСЫ И ПРИМЕРЫ"
Все разделы
ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ
КЕЙСЫ И ПРИМЕРЫ
ИННОВАЦИИ
Видео про управление от университета Синергия
Тёмная сторона лидерства | Ве
Радикальные инструменты уд...
Охота за идеей: генерация нес...
nit.center © 2020 - 2021



Страница "О компании"

Архив журнала

Страница Направления обучения

Лендинг Сертификация PMP

7.1. Создание файловой структуры

Создание сайта начинается с организации файловой структуры. На начальном этапе не обязательно использовать большое количество каталогов, достаточно определить хранилища для основных типов файлов:

- Файлы верстки
- Файлы стилей
- Файлы скриптов
- Дополнительные файлы:
  - Картинки
  - Иконки
  - Шрифты
  - Текстовые файлы
  - и т.д.

Конечно, такую структуру можно сделать через стандартный проводник, но гораздо удобнее использовать IDE – интегрированную среду разработки. Она же будет использоваться и для написания кода. В нашем примере представлена IDE от компании Microsoft – Visual Studio Code, которая является наиболее распространенной средой, в особенности, для начинающих разработчиков.

Базовая структура итогового проекта представлена на рис. 7.1:

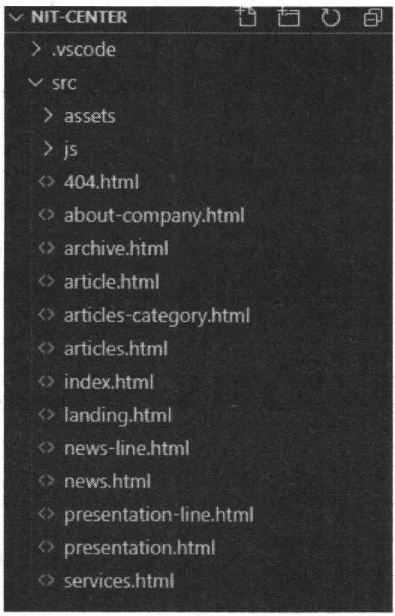


Рис. 7.1. Файловая структура итогового проекта

Корневая папка нашего проекта называется *nit-center*, далее располагаются две папки: .vscode, где находятся локальные настройки IDE для текущего проекта, и папка src, содержащая непосредственно файлы сайта.

В папке src располагаются документы в формате HTML – сверстанные страницы нашего проекта-сайта. Соответствие файлов страницам нашего сайта приведено в таблице 7.1

Таблица 7.1. Соответствие файлов вертки страницам сайта

Имя файла	Страница сайта	Является пунктом основного меню
index.html	Главная страница сайта "НИТ – учебный центр по управлению проектами"	Да
news.html	Страница новости	Нет
news-line.html	Страница с перечнем всех новостей	Нет
services.html	Страница с направлениями обучения	Да
landing.html	Страница "Сертификация PMP"	Да
archive.html	Страница с архивом журнала "Управление"	Да
articles.html	Страница со статьями	Да (подпункт главного меню)
article.html	Страница статьи	Нет
articles-category.html	Страница со статьями определенной категории	Нет
presentation-line.html	Страница с презентациями	Да (подпункт главного меню)
presentation.html	Страница презентации	Нет
about-company.html	Страница с информацией о компании	Да

В папках js и assets расположены дополнительные файлы проекта. В папке js располагается основной скриптовой файл, который подключается ко всем страницам сайта – main.js. В папке assests имеется своя организация хранения файлов, в зависимости от их формата (рис.7.2):

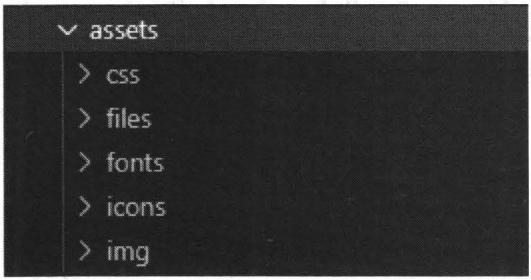


Рис. 7.2. Структура хранения дополнительных файлов проекта

Назначение каждой папки расписано в таблице 7.2.

Таблица 7.2

Назначение подпапок каталога assets

Папка	Назначение	Хранимые форматы
css	Хранятся основной и вспомогательные стилевые файлы. Также, при подключении дополнительных стилевых библиотек (например, Bootsrtap) их файлы также располагаются в данной папке	.css
files	Хранятся дополнительные файлы, используемые в проекта, например, презентации и текстовые файлы	.pdf, .doc и др.
fonts	Хранятся файлы, связанные с локально подключаемыми шрифтами	.woff, .woff2 и др.
icons	Хранятся иконки, в основном в векторном формате, либо с прозрачным фоном	.svg, .png и др.
img	Хранятся любые картинки, используемые в проекте	.jpeg, .png, .webp и др.

Для каждого проекта возможна своя организация файловой структуры, в зависимости от поставленных задач, а главное, от вкусов самого разра-

ботчика. Представленная схема не является каким-то стандартом и может быть, как и полностью скопирована при реализации собственных проектов, так и быть изменена с учетом персональных требований.

7.2. Верстка главной страницы

Перед началом верстки, разработчик всегда получает первоначальный макет страницы/сайта/лендинга и мысленно обозначает основные структурные элементы, на которые разбивается макет. Макет может быть представлен, как в формате .psd, используемом программой Adobe Photoshop, так и в формате популярного инструмента для прототипирования интерфейсов Figma, или в любом другом формате (например, как изображение).

Рис. 7.3. Фрагмент макета главной страницы, выполненный в программе Figma

Для нашей главной страницы в проекте, структура с учетом основных элементов будет выглядеть следующим образом (рис.7.4):

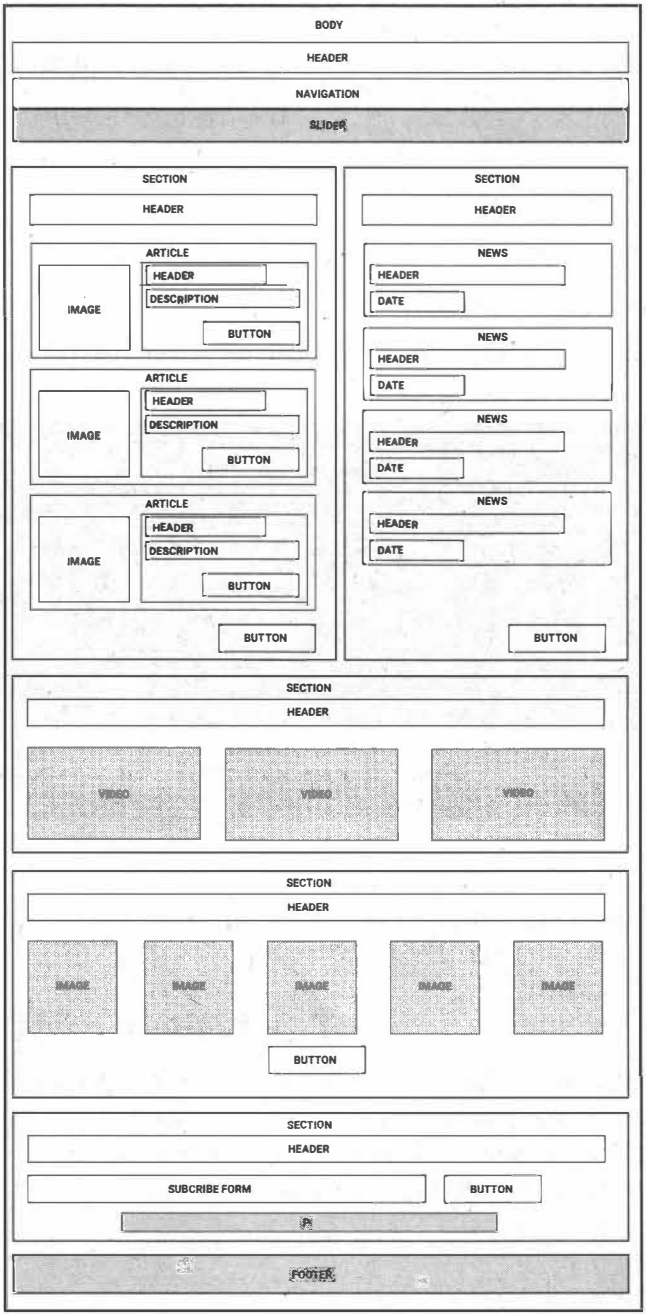


Рис. 7.4. Структура главной страницы проекта (см. стр. 319)

7.3. Верстка основных блоков главной страницы:

Далее разберем верстку главной страницы, при этом подробно рассмотрим наиболее крупные структурные блоки:

- блок HEAD
- блок HEADER
- блоки SECTION
- блок FOOTER

В общем виде, структура главной страницы имеет следующий вид:

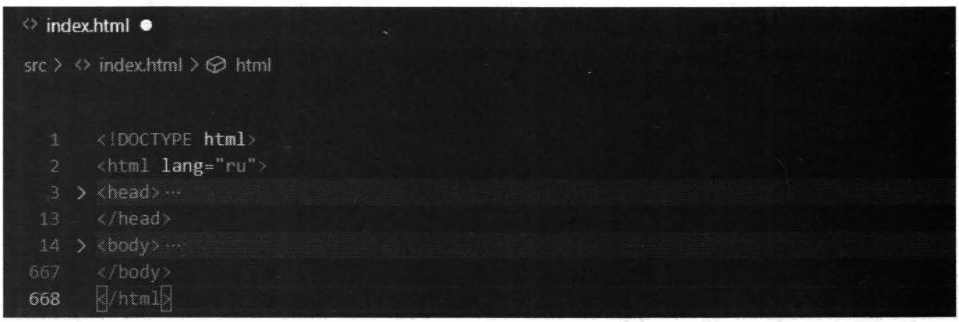


Рис. 7.5. Базовая структура главной страницы.

Основное содержимое страницы, отображающееся в браузере, расположено внутри тега BODY.

7.3.1. Блок HEAD

В данном блоке описывается основная информация о странице, указывается ее заголовок, перечисляются подключаемые стилевые файлы, шрифты, скрипты, прописываются дополнительные теги META, используемые для SEO-оптимизации страницы в поисковых системах.

Код данного блок приведен в листинге 7.1.

Листинг 7.1. Код блока HEAD

```
<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>НИТ -учебный центр по управлению проектами</title>

    <link href="https://fonts.googleapis.com/css2?family=Mont
serrat:wght@100;200;300;400;500;600;700;800;900&display=swap"
rel="stylesheet">

    <link href="https://fonts.googleapis.com/css2?family=
PT+Serif:ital,wght@0,400;0,700;1,400;1,700&display=swap"
rel="stylesheet">

    <link rel="stylesheet" href="assets/css/style.css">

    <link rel="stylesheet" href="assets/css/normalize.css">

    <link rel="stylesheet" href="assets/css/plugins/font-
awesome.min.css">

    <link rel="stylesheet" href="assets/css/plugins/swiper-
bundle.css">

</head>
```

Разберемся, какие внутренние и сторонние ресурсы подключаются к странице нашего сайта, а также, какие настройки мы указываем для работы. Разбор блока приведен в таблице 7.3.

Таблица 7.3
Разбор блока HEAD

Тег	Описание, Назначение
<meta charset="UTF-8">	Указывает на используемую на странице кодировку текста UTF-8
<meta name="viewport" content="width=device-width, initial-scale=1.0">	Определяет, как страница будет отображаться на дисплеях разного разрешения – адаптировать свое содержимое, либо сохранять пропорции и размеры. В данном случае фрагмент content="width=device-width, initial-scale=1.0" указывает на то, что ширина страницы будет адаптироваться под ширину устройства
<title>НИТ -учебный центр по управлению проектами</title>	В данном теге определяется название страницы, которое будет указываться в адресной строке браузера
<link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@100;200;300;400;500;600;700;800;900&display=swap" rel="stylesheet">	В данном теге подключается шрифт Montserrat. Ссылку для подключения можно сгенерировать на странице fonts.google.com. Данный шрифт относится к категории Sans Serif, т.е. шрифт без засечек. На нашем сайте, данный шрифт используется в основном для заголовков разного уровня, кнопок, пунктов меню и т.п.
<link href="https://fonts.googleapis.com/css2?family=PT+Serif:ital,wght@0,400;0,700;1,400;1,700&display=swap" rel="stylesheet">	В данном теге подключается шрифт PT Serif. Ссылку для подключения можно сгенерировать на странице fonts.google.com. Данный шрифт относится к категории Serif, т.е. шрифт с засечками. На нашем сайте, данный шрифт используется в основном для больших текстовых блоков, например, содержании статьи, описание презентации и т.п.

<link rel="stylesheet" href="assets/css/style.css">	В данном теге подключается основной стилевой файл style.css. Если упустить данный тег, то написанные нами стили попросту не применятся к странице
<link rel="stylesheet" href="assets/css/normalize.css">	В данном теге подключается дополнительный стилевой файл, который называется normalize.css. В данном файле содержаться "нормализующие" стили для многих базовых тегов, улучшающих их отображение в разных браузерах. Скачать файл можно по ссылке: https://necolas.github.io/normalize.css/
<link rel="stylesheet" href="assets/css/plugins/font-awesome.min.css">	В данном теге подключается дополнительный стилевой файл иконочного шрифта Font Awesome. Подключение данных таблиц стиля позволяет нам упростить использование на нашем сайте иконок из популярной библиотеки. Узнать подробнее о Font Awesome и способах его подключения к сайту можно по ссылке: https://fontawesome.ru/
<link rel="stylesheet" href="assets/css/plugins/swiper-bundle.css">	Данный тег указывается только для страниц, где используется подключаемый сторонний слайдер Swiper. В данном теге мы подключаем таблицу стилей для этого слайдера. Подробнее о подключении слайдера Swiper к нашему сайту будет рассказано в главе про JavaScript.

7.3.2. Блок HEADER

В данном блоке содержится структура главного меню страницы и всего сайта в целом. Как и на многих сайтах, для разных страниц меню остается единообразным, меняются только стили активных пунктов меню. Т.к. меню – это видимая часть сайта, которая отображается пользователю, то данные

и последующие блоки располагаются внутри тега BODY. Код данного блока представлен в листинге 7.2:

Листинг 7.2. Код блока HEADER:

```
<header class="main-header">
    <div class="container">
        <div class="header-top">
            <a href="http://nit.center/" class="main-logo-link">
                <img src="http://nit.center/assets/icons/main-logo.
png" width="250" height="100">
            </a>
            <div class="header-top-content col-lg-9">
                <ul class="contacts-list">
                    <li class="contacts-item">
             <a href="mailto:info@nit.center" class="contacts-link">
                            info@nit.center
                        </a>
                    </li>
                    <li class="contacts-item">
                     <a href="tel:+78123090686" class="contacts-link">
                            +7(812)000-00-00
                        </a>
                    </li>
                    <li class="contacts-item">
                      <a href="tel:+74992838606" class="contacts-link">
                            +7(499)000-00-00
                        </a>
                    </li>
                </ul>
                <ul class="tags-list">
                    <li class="tags-item">
    <a href="http://nit.center/presentation/6hats" class="tag-link">
                            Шесть думающих шляп
                        </a>
                    </li>
                    <li class="tags-item">
        <a href="http://nit.center/landing.php" class="tag-link">
```

```
                        Сдай PMP успешно
                    </a>
                </li>
                <li class="tags-item">
    <a href="http://nit.center/articles/kernel" class="tag-link">
                        Инновации
                    </a>
                </li>
                <li class="tags-item">
     <a href="http://nit.center/education.php" class="tag-link">
                        Обучение
                    </a>
                </li>
                <li class="tags-item">
         <a href="http://nit.center/news.php" class="tag-link">
                        Новости
                    </a>
                </li>
            </ul>
        </div>
        <button class="navbar-toggler">
            <span class="toggler-icon"></span>
            <span class="toggler-icon"></span>
            <span class="toggler-icon"></span>
        </button>
    </div>
</div>
<div class="navigation-wrapper">
    <div class="container">
        <nav class="main-navigation">
            <ul class="main-navigation-list">
            <li class="main-navigation-item">
    <a href="http://nit.center" class="main-navigation-link"
active>Главная</a></li><li class="main-navigation-item">
    <a href="http://nit.center/education.php" class="main-
navigation-link" >Направления обучения</a></li><li class="main-
navigation-item">
```

```
        <a href="http://nit.center/landing.php" class="main-
navigation-link" >Сертификация PMP</a></li><li class="main-
navigation-item">
        <a href="http://nit.center/archive.php" class="main-
navigation-link" >Архив журнала Управление</a></li><li class="main-
navigation-item">
        <a href="http://nit.center/articles.php" class="main-
navigation-link" >База знаний</a>
          <ul class="submenu-list"><li class="main-navigation-item">
                <a href="http://nit.center/articles.php"
class="main-navigation-link">Статьи</a></li><li class="main-
navigation-item">
              <a href="http://nit.center/presentations.php"
class="main-navigation-link">Презентации</a></li></ul>
        </li><li class="main-navigation-item">
        <a href="http://nit.center/about.php" class="main-
navigation-link" >О компании</a></li>

                </ul>
            </nav>
        </div>
    </div>
</header>
```

7.3.3. Блок SECTION – слайдер

В данном блоке содержится код слайдера для главной страницы. Особенность слайдера заключается в наличии миниатюр слайдов в боковой части, с помощью которых также доступна навигация между слайдами. Для "оживления" слайдера используется библиотека одного из популярных слайдеров Swiper. Если внимательно изучить код верстки данного блока, то можно увидеть, что у некоторых тегов присутствует класс, имеющий в названии составную "swiper". Это значит, что данные блоки структуры слайдера, соответствуют документации Swiper. Листинг данного блока:

Листинг 7.3. Блок SECTION – слайдер

```
<section class="slider-section">
    <h1 class="visually-hidden"></h1>
    <div class="container">
        <div class="slider">
            <div class="slider-col-1 main-slider-thumbs-
container">
                <ul class="preview-list swiper-wrapper">
                <li class="preview-item swiper-slide" active
data-slide-to="1">
                <img src="slider/10.jpg" width="98px" alt="">
            </li><li class="preview-item swiper-slide"  data-
slide-to="2">
                <img src="slider/8.jpg" width="98px" alt="">
            </li><li class="preview-item swiper-slide"  data-
slide-to="3">
                <img src="slider/9.jpg" width="98px" alt="">
            </li>

                </ul>
            </div>
            <div class="slider-col-2">
                <div class="slider-item-field main-slider-container">
                  <div class="slider-item-wrapper swiper-wrapper">
                  <div class="slider-item swiper-slide">
                  <a target=_blank href="http://nit.center/">
                        <img src="slider/10.jpg"
width="1030px" height="360px" alt="">
                    </a>
                </div><div class="slider-item swiper-slide">
                    <a target=_blank href="http://nit.center/">
                        <img src="slider/8.jpg" width="1030px"
height="360px" alt="">
                    </a>
                </div><div class="slider-item swiper-slide">
               <a target=_blank href="http://nit.center/landing.php">
```

```
                        <img src="slider/9.jpg" width="1030px"
height="360px" alt="">
                    </a>
                </div>

                    </div>
                    <div class="slider-navigation">
                    </div>
                </div>
            </div>
        </div>
    </div>
</section>
```

7.3.4. Блок SECTION – Статьи и новости

Данный блок содержит в себе две колонки, в которых представлены списки: в левом – список со статьями, в правом – список новостей. В листинге данного блока указано по одному элементу для статей и новостей, чтобы не перегружать листинг однотипными элементами.

Листинг 7.4.

```
<div class="articles-news-books-area row-section-field">
    <div class="row-sections-wrapper container">
        <div class="row">
            <section class="useful-articles double-block">
                <div class="articles-row row">
                    <div class="section-title">
                        <span class="sub-title">Статьи</span>
                        <h2>Полезные статьи</h2>
                    </div>
                </div>
                <ul class="articles-list">
                 <li class="articles-item">
```

```
                    <div class="article-image">
                        <img src="http://nit.center/content/
format_proektnoi_deyatel_nosti_po_realizatsii_biznes_
proektov_v_ramkah_obrazovatel_nogo_protsessa_soglasno_
modeli_universiteta_3_0_/cover.jpg" width="150" height="150"
alt="Обложка статьи ФОРМАТ ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ ПО
РЕАЛИЗАЦИИ БИЗНЕС-ПРОЕКТОВ В РАМКАХ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА
СОГЛАСНО МОДЕЛИ УНИВЕРСИТЕТА 3.0.">
                    </div>
                    <div class="article-description">
                        <h3>ФОРМАТ ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ ПО
РЕАЛИЗАЦИИ БИЗНЕС-ПРОЕКТОВ В РАМКАХ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА
СОГЛАСНО МОДЕЛИ УНИВЕРСИТЕТА 3.0.</h3>
                        <p>
                            В данной статье описан опыт
реализации бизнес-проекта в рамках образовательного процесса.
На примере конкретного проекта разобрана среднесрочная форма
проектной деятельности, рекомендованная авторами к внедрению в
образовательный процесс в ВУЗах РФ. Приве
                        </p>
                        <a href="http://nit.center/article/
format_proektnoi_deyatel_nosti_po_realizatsii_biznes_proektov"
class="button">Читать далее</a>
                    </div>
                </li>
                ...

                </ul>
                <div class="show-more-wrapper">
                    <a href="articles.php" class="colored-
btn">Читать все статьи</a>
                </div>
            </section>
            <div class="double-block">
                <section class="news">
                    <div class="news-row row">
                        <div class="section-title">
                            <span class="sub-title">Новости
мира СПО</span>
                            <h2>Новости</h2>
```

```
                        </div>
                    </div>
                    <ul class="news-list">
                    <li class="news-item">
                        <a href="http://nit.center/
news/53f57dcd-616b-11eb-a414-ac1f6bbd4544" class="news-
header">
                            Выручка московских театров выросла
почти на 60% после смягчения ограничений
                        </a>
                        <span class="news-date">2021-01-28</
span>
                    </li>
                    ...

                    </ul>
                    <div class="show-more-wrapper">
                        <a href="news.php" class="colored-
btn">Все новости</a>
                    </div>
                </section>
            </div>
        </div>
    </div>
</div>
```

7.3.5. Блок SECTION – видео

В данном блоке расположены видео в три колонки. Из нового, для главной страницы, можно отметить использование тегов IFRAM. Код блока представлен в листинге 7.5:

Листинг 7.5.

```
<section class="video-section">
    <div class="container">
        <div class="video-wrapper">
            <div class="video-row row">
                <div class="section-title col-lg-6">
                    <span class="sub-title">Видео</span>
                    <h2>Видео про управление от университета
Синергия</h2>
                </div>
            </div>
            <ul class="video-content-row row">
            <li class="tetra-block col-lg-4">
                    <iframe height="228" src="https://
www.youtube.com/embed/AwfkCcdldKw" frameborder="0"
allow="accelerometer; encrypted-media; gyroscope; picture-in-
picture" allowfullscreen></iframe>
                </li><li class="tetra-block col-lg-4">
                    <iframe height="228" src="https://
www.youtube.com/embed/6Aw5hRKZg4g" frameborder="0"
allow="accelerometer; encrypted-media; gyroscope; picture-in-
picture" allowfullscreen></iframe>
                </li><li class="tetra-block col-lg-4">
                    <iframe height="228" src="https://
www.youtube.com/embed/leZJps7wwYw" frameborder="0"
allow="accelerometer; encrypted-media; gyroscope; picture-in-
picture" allowfullscreen></iframe>
                </li>
            </ul>
        </div>
    </div>
</section>
```

7.3.6. Блок SECTION – Архив журнала

Данный блок представляет собой пятиколончатую секцию, каждый элемент-журнал представляет собой картинку с текстовым блоком-подписью.

В данном листинге также не указаны повторяющиеся элементы списка, обернутые в тег <li>. Листинг данной секции:

Листинг 7.6.

```
<section class="management-archive">
    <div class="management-archive-shape">
        <img src="./assets/img/archive-management/shape-1.png"
aria-hidden="true">
    </div>
    <div class="container">
        <div class="management-archive-row row">
            <div class="section-title">
                <span class="sub-title">Все выпуски журнала</
span>
                <h2>Архив журнала "Управление"</h2>
            </div>
        </div>
        <ul class="penta-wrapper row">
            <li class="penta-block management-archive-item">
                <div class="management-archive-item-wrapper">
                    <img src="http://nit.center/lf/images/11.
jpg" width="232" alt="Выпуск 3/2018">
                    <span class="archive-number">Выпуск
3/2018</span>
                    <a target=_blank href="http://nit.center/
lf/pdf/2018-3.pdf" class="download-issue-btn">
                        Скачать в PDF
                    </a>
                </div>
            </li>
            ...
        </ul>
        <div class="show-more-wrapper">
            <a href="archive.php" class="colored-btn">Перейти
в архив</a>
        </div>
    </div>
</section>
```

7.3.7. Блок SECTION – подписка

Наличие на сайте формы обратной связи для клиентов – это правило хорошо тона для любого разработчика. Т.к. большинство сайтов или интернет-порталов нацелены на привлечение клиента и побуждение его на совершение покупки, то такой элемент, как форма обратной связи просто необходим. Для создания форм используется тег FORM, в котором располагаются поля для ввода пользователем данных, создаваемые тегом INPUT. Листинг данного блока:

Листинг 7.7:

```
<section class="subscribe">
    <div class="container">
        <div class="subscribe-wrapper">
            <div class="subscribe-row row">
                <div class="section-title">
                    <h2>Подписаться на рассылку</h2>
                </div>
            </div>
            <div class="subscribe-row row">
                <div class="col-lg-8">
                    <form class="subscribe-form"
name="subscribe-form" method="POST">
                        <label>
                            <input type="email"
name="subscribe-email" placeholder="Введите ваш почтовый ящик"
maxlength="40" required>
                        </label>
                        <div class="subscribe-button-wrapper">
                            <button class="subscribe-button
colored-btn" type="submit">Подписаться</button>
                        </div>
                    </form>
                    <p class="subscribe-description">
                        Один раз в месяц мы делимся с
подписчиками самой актуальной и полезной информацией по
управлению проектами: новости, лайфхаки и секреты подготовки
```

```
к PMP, успешные кейсы. Просто оставьте свой адрес электронной
почты и убедитесь сами!
                    </p>
                </div>
            </div>
        </div>
    </div>
</section>
```

7.3.8. Блок FOOTER

Последний блок на данной странице – это футер. В футерах сайтов обычно указывают контактную информацию, ссылки на профили в социальных сетях, иногда дублируют основное меню, либо делают отдельное меню для футера. В нашем проекте в футере содержится название сайта, а также набор ссылок на профили в социальной сети. Как и другие блоки, данный блок обернут в семантическмй тег FOOTER, который позволяет понять разработчикам и браузерам его назначение. Листинг данного блока:

Листинг 7.7:

```
<footer class="main-footer">
    <div class="container">
        <div class="footer-wrapper">
         <p class="footer-creditals">nit.center © 2020 - 2021</p>
            <ul class="social-list">
                <li class="social-item">
                    <a href="" aria-label="Вконтакте">
                        <svg xmlns="http://www.w3.org/2000/
svg" width="27" height="15" viewBox="0 0 26.14 14.91">

  </svg>
                    </a>
                </li>
                <li class="social-item">
                    <a href="" aria-label="Instagram">
```

```
                        <svg xmlns="http://www.w3.org/2000/
svg" width="20" height="20" viewBox="0 0 20 20">

  </svg>
                    </a>
                </li>
                <li class="social-item">
                    <a href="" aria-label="YouTube">
                        <svg xmlns="http://www.w3.org/2000/
svg" width="20" height="20" viewBox="0 0 512 352">

                        </svg> </a>
                </li>
                <li class="social-item">
                    <a href="" aria-label="Facebook">
                        <svg xmlns="http://www.w3.org/2000/
svg" width="19" height="22" viewBox="0 0 10.15 21.74">

   </svg>
                    </a>
                </li>

            </ul>
        </div>
    </div>
</footer>
```

В данном блоке, иконки социальных сетей созданы с помощью интеграции кода SVG-изображений непосредственно в верстку страниц. Данная реализация является одним из способов добавления на страницу векторных изображений, которые можно легко редактировать с помощью стилевых файлов. Однако из-за объемистости кода SVG изображений в листинге 2.7 он опущен.

7.4. Верстка основных блоков страницы "Направления обучения"

В качестве дополнительного примера верстки страницы нашего сайта, рассмотрим блоки страницы "Направления обучения". Макет данной страницы представлен на рис. 7.6. Внешний вид страницы можно посмотреть на стр. 322.

Как видно из макета данной страницы, у всех страниц сайта будет 2 общих элемента – блоки HEADER и FOOTER. И т.к. мы рассмотрели структуру данных блоков для главной страницы ранее, в данном разделе остановимся только на новых блоках. На странице "Направление обучения" можно выделить **4** крупных новых блока:

1. Блок SECTION – Услуги.
2. Блок SECTION – Наши преимущества.
3. Блок SECTION – Кто нам доверяет (партнеры).
4. Блок SECTION – Форма обратной связи.

Рассмотрим каждый из этих блоков более подробно.

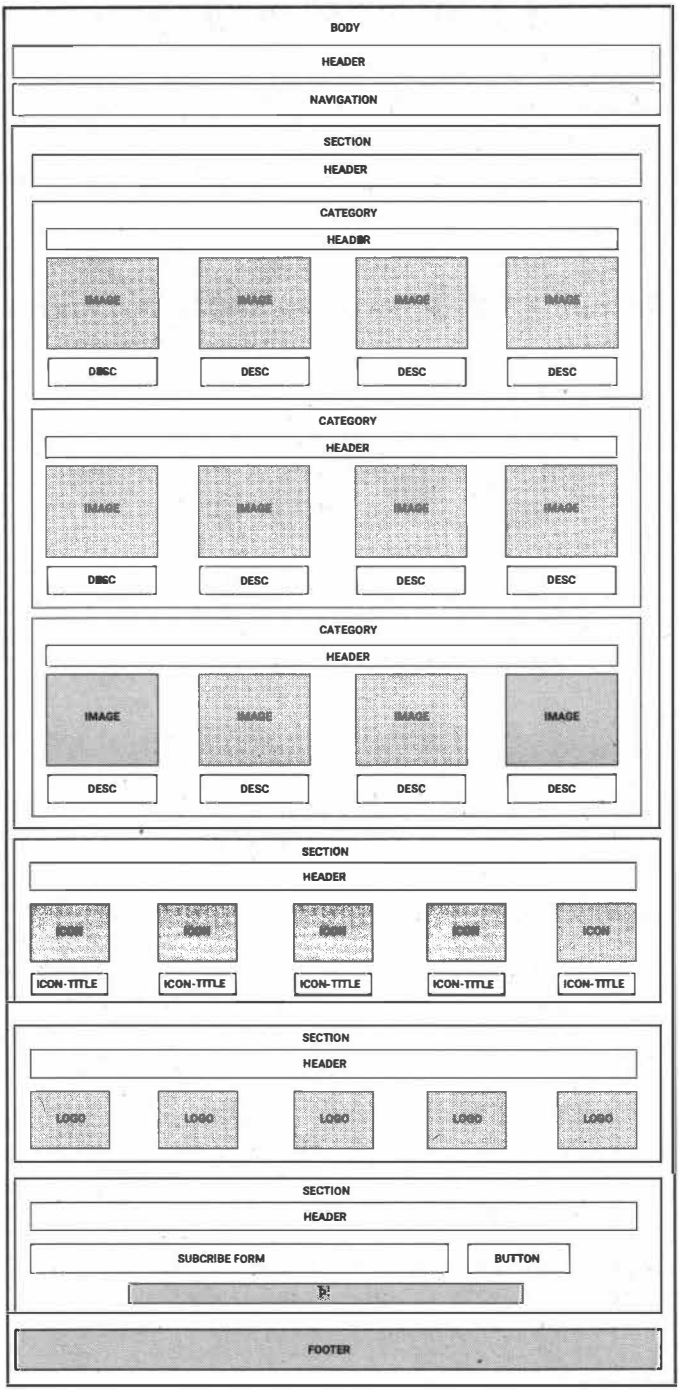


Рис. 7.6. Макет страницы "Направления обучения"

7.4.1. Верстка блока SECTION – Услуги

В общем виде, данный блок можно представить как 3 элемента списка, в каждом из которых находится по 4 колонки. Именно по этому принципу и выполнена верстка данного блока. Код блока можно посмотреть в листинге 7.8.

Листинг 7.8. Верстка блока SECTION – Услуги

```
<section class="services-block">
    <div class="container">
        <div class="header-row row">
            <div class="section-title">
                <span class="sub-title"></span>
                <h1>Услуги</h1>
            </div>
        </div>
        <div class="quatro-field">
            <ul class="services-categories-list">
                <li class="services-categories-item">
                    <h3 class="services-categories-
name">Управление проектами</h3>
                    <ul class="services-list quatro-wrapper">
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/1.png> </div> <span class="services-item-
description">Управление содержанием</span>
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/2.png> </div> <span class="services-item-
description">Управление качеством</span>
                        </li>
```

```
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/3.png> </div> <span class="services-item-
description">Управление заинтересованными сторонами</span>
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/4.png> </div> <span class="services-item-
description">Управление рисками</span>
                        </li>
                    </ul>
                </li>
                <li class="services-categories-item">
                    <h3 class="services-categories-
name">Управление персоналом</h3>
                    <ul class="services-list quatro-wrapper">
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/5.png> </div> <span class="services-item-
description">Управление проектными командами</span>
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/06.png> </div> <span class="services-item-
description">Управление мотивацией</span>
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/07.png> </div> <span class="services-item-
description">Лидерство</span>
```

```
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/08.png> </div> <span class="services-item-
description">Контроль исполнения поручений</span>
                        </li>
                    </ul>
                </li>
                <li class="services-categories-item">
                    <h3 class="services-categories-
name">Управление продуктом</h3>
                    <ul class="services-list quatro-wrapper">
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/09.png> </div> <span class="services-item-
description">Управление жизненным циклом продукта</span>
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/10.png> </div> <span class="services-item-
description">Бизнес-анализ</span>
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
                                <img src=/assets/img/
education/11.png> </div> <span class="services-item-
description">Юнит-экономика</span>
                        </li>
                        <li class="services-item quatro-
block">
                            <div class="services-item-icon">
```

```
                                    <img src=/assets/img/
education/12.png> </div> <span class="services-item-
description">Продвижение продукта в SMM</span>
                            </li>
                        </ul>
                    </li>
                </ul>
            </div>
        </div>
</section>
```

7.4.2. Верстка блока SECTION – Наши преимущества

С точки зрения верстки данный блок имеет довольно очевидную пятиколончатую структуру. Из особенностей верстки данного блока можно выделить использование встроенного SVG-кода для изображения иконок. Для большей читаемости листинга, данные SVG фрагменты исключены из него. Листинг данного блока:

Листинг 7.9. Верстка блока SECTION – Наши преимущества

```
<section class="benefits">
    <div class="container">
        <div class="header-row row">
            <div class="section-title">
                <h2>Наши преимущества</h2>
            </div>
        </div>
        <ul class="benefits-list penta-wrapper">
            <li class="benefits-item penta-block">
                <div class="benefit-icon">
                    <svg xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 480.006 480.006">
                    ...
                    </svg>
                </div>
```

```
                <h3 class="benefit-title">3 года</h3>
            </li>
            <li class="benefits-item penta-block">
                <div class="benefit-icon">
                    <svg height="511pt" viewBox="0 -3
511.99992 511" width="511pt" xmlns="http://www.w3.org/2000/
svg">
                    ...
                    </svg>
                </div>
                <h3 class="benefit-title">250+ выпускников</h3>
            </li>
            <li class="benefits-item penta-block">
                <div class="benefit-icon">
                    <svg xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 496 496">
                    ...
                    </svg>
                </div>
                <h3 class="benefit-title">Индивидуальный
подход</h3>
            </li>
            <li class="benefits-item penta-block">
                <div class="benefit-icon">
                 <svg xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 512 512">
                        ...
                    </svg>
                </div>
                <h3 class="benefit-title">Профессиональные
методики</h3>
            </li>
            <li class="benefits-item penta-block">
                <div class="benefit-icon">
                    <?xml version="1.0" ?>
                    <svg id="Layer_1" style="enable-
background:new 0 0 24 24;" version="1.1" viewBox="0 0 24
24" xml:space="preserve" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
```

```
                    ...
                </svg>
            </div>
            <h3 class="benefit-title">Эксперты в управлении
проектами</h3>
        </li>
    </ul>
</div>
</section>
```

7.4.3. Верстка блока SECTION – Кто нам доверяет (партнеры)

Данный блок имеет такую же структуру верстки, как и предыдущий блок с преимуществами. Единственно отличие заключается в том, что в данном блоке изображение добавлены через тег IMG. Листинг данного блока:

Листинг 7.10. Верстка блока SECTION – Кто нам доверяет (партнеры)

```
<section class="partners">
    <div class="container">
        <div class="header-row row">
            <div class="section-title">
                <span class="sub-title">Статьи</span>
                <h2>Кто нам доверяет</h2>
            </div>
        </div>
        <ul class="penta-wrapper">
            <li class="penta-block">
                <a target=_blank href="" class="partner-link">
                    <img src="http://nit.center/assets/img/
parnters/logo1.png" alt="Логотип партнера Microsoft">
                </a>
            </li>
            <li class="penta-block">
```

```
                <a target=_blank href="" class="partner-link">
                    <img src="http://nit.center/assets/img/
parnters/logo2.png" alt="Логотип партнера Intel">
                </a>
            </li>
            <li class="penta-block">
                <a target=_blank href="" class="partner-link">
                    <img src="http://nit.center/assets/img/
parnters/logo3.png" alt="Логотип партнера PMI">
                </a>
            </li>
            <li class="penta-block">
                <a target=_blank href="" class="partner-link">
                    <img src="http://nit.center/assets/img/
parnters/logo4.png" alt="Логотип партнера Amazon">
                </a>
            </li>
            <li class="penta-block">
                <a class="partner-link more-partners">
                    И многие другие
                </a>
            </li>
        </ul>
    </div>
</section>
```

7.4.4. Верстка блока SECTION – Кто нам доверяет (партнеры)

Данный блок имеет такую же структуру, как и блок с формой на главной странице (см раздел. 7.3.7). В отличие от блока на главной странице, у данной формы два поля для ввода, заданных тегами INPUT. Листинг данного блока:

Листинг 7.11. Верстка блока SECTION – Кто нам доверяет (партнеры)

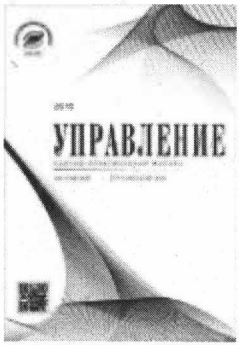
```
<section class="feedback-contact">
    <div class="container">
        <div class="header-row row">
            <div class="section-title">
                <span class="sub-title">Статьи</span>
                <h2>Перезвоните мне</h2>
            </div>
        </div>
        <form class="feedback-form" name="feedback-form"
method="POST">
            <label>
                <input type="text" name="feedback-name"
placeholder="Введите ваше имя" maxlength="40" required>
            </label>
            <label>
                <input type="tel" name="feedback-phone"
placeholder="Введите ваш телефон" maxlength="40" required>
            </label>
            <div class="feedback-button-wrapper">
                <button class="feedback-button colored-btn"
type="submit">Заказать звонок</button>
            </div>
        </form>
    </div>
</section>
```

7.5. Стилизация страниц сайта

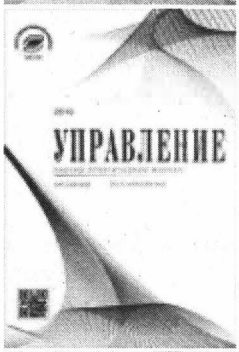
Для стилизации сайта использовались каскадные таблицы стилей, или попростому – язык стилей CSS3.

В качестве примера стилизации, рассмотрим секцию с архивом журналов "Управление" (см раздел 7.3.6). Если открыть этот блок в браузере без применения стилей, то мы увидим следующую картину:

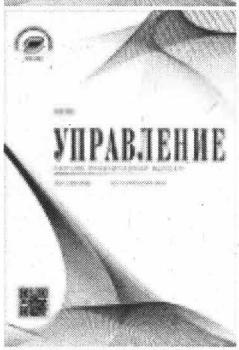
Архив журнала "Управление"



- Выпуск 4/2020 Скачать в PDF



- Выпуск 1/2018 Скачать в PDF



- Выпуск 3/2017 Скачать в PDF

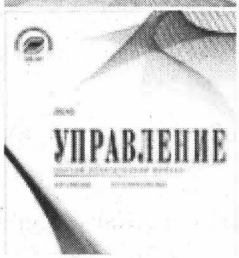


Рис. 7.8. Блок "Архив журнала Управление" без установленных стилей

Итоговый вариант применения стилей к нашему файлу верстки представлен на рисунке 7.9.:

Рис. 7.9. Блок "Архив журнала Управление" с установленными стилями

Чтобы осуществить подобную трансформацию нам нужно корректно составить стилевой файл для данного блока. Стилизация обектов (тегов) данного блока будет осуществляться с помощью использования классов. Чтобы вспомнить, какие классы указаны в верстке данного блока, можно обратиться к разделу 7.3.6. Стили для страницы записываются в документ style.css, который располагается в каталоге nit-center/src/assets/css, специально созданном для файлов данного типа. Примененные к данному блока свойства стилей представлены в таблице 7.4:

Таблица 7.4. Стили блока "Архив журнала Управление"

Тег	Стили	Назначение стилей
<section class="management-archive">	.management-archive { position: relative; padding: 80px 0 40px; overflow: hidden; }	Данные стили задают верхний и нижний отступы всего блока, не позволяя соседним блокам "прилипать" к нему

Тег	Стили	Назначение стилей

	management-archive-shape {
	position: absolute;
	top: 10%;
	right: 5%;
	animation: rotated 15s infinite linear;
	
	opacity: .3;
	}
		Данные стили используются для настройки отображения декоративного фонового элемента. Из особенностей – придание объекту вращения с помощью CSS3 анимации

<img src="./assets/img/archive-management/shape-1.png" aria-hidden="true">	img {
	max-width: 100%;
	height: auto;
	}
		Данные стили позволяют занимать изображению 100% своего блока-родителя, при этом автоматически подгоняя высоту, для сохранения пропорций

	.container {
	margin: 0 auto;
	padding: 0px;
	padding: 0 10px;
	max-width: 1280px;
	}
		Данный свойства ограничивают весь контент блока. Также с их помощью контент располагается ровно по центру страницы

	.row {
	list-style: none;
	display: flex;
	flex-wrap: wrap;
	margin-right: -15px;
	margin-left: -15px;
	padding: 0;
	
	box-sizing: border-box;
	}
	
	.management-archive-row {
	justify-content: center;
	text-align: center;
	}
		Первая группа стилей задает контейнер, где будет располагаться заголовок нашей секции. Т.к. стили являются универсальными, то могут быть применены к разным блокам.
		
		Вторая группа стилей дополняет первую, переписывая свойства конкретно под наш блок.

	.section-title {
	margin-bottom: 30px;
	}
		Свойство задает нижний отступ для заголовка.

Тег	Стили	Назначение стилей

	.section-title .sub-title {
	display: none;
	position: relative;
	margin: 0;
	padding: 0;
	min-height: 10px;
	
	font-family: var(--title-family);
	font-size: 16px;
	line-height: 24px;
	font-weight: 500;
	color: #666e82;
	}
		Свойства отвечают за стилизацию подзаголовка. Но т.к. в данных блоках подзаголовок не отображается, то он скрыт с помощью свойства display: none;

<h2>	.section-title h2,
	.section-title h1{
	position: relative;
	margin: 0;
	padding: 0;
	
	font-family: var(--title-family);
	font-size: 46px;
	line-height: 56px;
	font-weight: 700;
	letter-spacing: 0px;
	color: #222;
	}
		Данные свойства стилизуют все заголовки второго и первого уровня, которые находятся внутри заголовочного блока секции < section-title>

<ul class="penta-wrapper row">	.penta-wrapper {
	display: flex;
	flex-wrap: wrap;
	list-style: none;
	margin: 0;
	padding: 0;
	margin-right: -15px;
	margin-left: -15px;
	
	box-sizing: border-box;
	}
		Первая группа стилей задает контейнер, в котором будут располагаться колонки – картинки журналов.
		
		Вторая группа стилей дополняет первую с помощью универсальных правил селектора .row

Тег	Стили	Назначение стилей

<li class="penta-block management-archive-item">	.penta-block {
	flex: 0 0 20%;
	max-width: 20%;
	position: relative;
	width: 100%;
	padding-right: 15px;
	padding-left: 15px;
	box-sizing: border-box;
	}
	
	.management-archive-item {
	position: relative;
	margin-bottom: 10px;
	}
		Первая группа стилей является универсальной для всех блоков, где используется пяти колончатая структура отображения контента. Как видно из свойсв, для каждого блока отдается 20% или 1/5 ширины родительского блока.
		
		Вторая группа свойств дополняет первую с помощью специфических для нашего блока значений.

	.management-archive-item-wrapper {
	display: flex;
	flex-direction: column;
	align-items: center;
	
	height: 100%;
	
	cursor: default;
	}
		Данный блок стилизует "оболочку" элемента – журнала

	.archive-number {
	display: inline-block;
	vertical-align: top;
	height: 48px;
	overflow: hidden;
	
	font-family: var(--title-family);
	font-size: 16px;
	line-height: 24px;
	font-weight: 500;
	text-transform: uppercase;
	color: #666e82;
	}
		Данный стиль отвечает за отображение подписи к журналу

Тег	Стили	Назначение стилей

<a target=_blank href="http://nit.center/lf/pdf/2018-3.pdf" class="download-issue-btn">	.download-issue-btn {
	cursor: pointer;
	display: inline-block;
	align-self: flex-end;
	padding: 13px 5px 13px 40px;
	max-width: 190px;
	width: 190px;
	box-sizing: border-box;
	
	font-family: var(--title-family);
	font-size: 18px;
	line-height: 20px;
	font-weight: 500;
	text-transform: none;
	color: #4e256f;
	background-color: #fff;
	border: 2px solid #4e256f;
	
	transition: all .2s ease-out;
	}
		Данные стили отвечают за отображение кнопки "Скачать в PDF", которая появляется при наведение на обложку журнала

	.show-more-wrapper {
	display: flex;
	justify-content: flex-end;
	}
	
	.management-archive .show-more-wrapper {
	justify-content: center;
	}
		Данные стили отвечают за создание оболочки-контейнера в котором будет находится кнопка "Перейти в архив"

Тег	Стили	Назначение стилей

<a href="archive.
php" class="colored-
btn">Перейти в архив</
a>
	.colored-btn,
	.show-more,
	.back-to-list-btn {
	position: relative;
	display: inline-block;
	padding: 15px 15px;
	z-index: 5;
	overflow: hidden;
	
	font-family: var(--title-
	family);
	font-size: 16px;
	line-height: 20px;
	font-weight: 700;
	text-align: center;
	color: #fff;
	vertical-align: top;
	text-transform:
	uppercase;
	
	background-color:
	#4e256f;
	box-shadow: 5px 5px
	10px rgba(0, 0, 0, .1);
	
	transition: all .15s ease-
	out;
	}
		И последняя группа стилей отвечает за непосредственную стилизацию кнопки "Перейти в архив"

Помимо указанных стилей, для данного блока существуют дополнительные стили, реагирующие на взаимодействие с пользователем. Для задания подобных стилей используются селекторы псевдоклассов :hover и :active. Дополнительные стили указаны в таблице 7.5.

Таблица 7.5. Дополнительные стили блока "Архив журнала Управление"

Тег	Стили	Назначение стилей

<a target=_blank href="http://nit.center/lf/pdf/2018-3.pdf" class="download-issue-btn">	.download-issue-btn:hover,
	.download-issue-btn:focus {
	color: #fff;
	background-color: #4e256f;
	}
	
	.download-issue-btn::before {
	position: absolute;
	content: "";
	
	top: calc(50% - 15px);
	left: 8px;
	width: 30px;
	height: 30px;
	
	background-color: transparent;
	background-image: url("../icons/download-icon-yellow.svg");
	background-position: 0 0;
	background-repeat: no-repeat;
	background-size: contain;
	}
	
	.download-issue-btn:hover::before,
	.download-issue-btn:focus::before {
	background-image: url("../icons/download-icon-white.svg");
	}
		Данные группы стилей отвечают за изменение кнопки:
		А) при наведении курсора на обложку;
		Б) при наведении курсора на саму кнопку

Тег	Стили	Назначение стилей

Перейти в архив	.colored-btn:focus,
	.colored-btn:hover,
	color: #c4c4c4;
	}
	
	.colored-btn::before {
	position: absolute;
	content: "";
	top: 0;
	left: 0;
	width: 5px;
	height: 100%;
	z-index: -1;
	background-color: #ac3193;
	
	/* transition: all 0.4s linear; */
	transition: all .15s ease-out;
	}
	
	.subscribe-button.colored-btn::before,
	.feedback-button.colored-btn::before {
	background-color: #4e256f;
	}
		Данные группы стилей отвечают за изменение кнопки при наведении курсора на саму кнопку

7.6. Адаптивная верста сайтов

На сегодняшний момент недостаточно сделать сайт, который будет работать только в браузерах на компьютерах и ноутбуках. Сейчас от верстки требуется одинаково сбалансированного и аккуратного отображения на различные устройства с различными разрешениями дисплеев: планшетах, смартфонах, экранах телевизоров и т.д. Верстка, которая соответствует данному требованию, называется адаптивной.

Для реализации адаптивной вёрстки, необходимо задать дополнительные стили для определенных тегов, чтобы они могли иметь разное отображение, на экранах с разным разрешением. Например, у нас есть блок, в котором 5 колонок. На больших дисплеях компьютеров эти колонки занимают один ряд, но если мы будем уменьшать дисплей, при этом сохраняя все колонки в одной ряду, то они в определенный момент станут нечитаемыми. Для этого, если на больших дисплеях каждая колонка занимает 20% (1/5) ряда, то на экранах планшетов, например, будет занимать 50%. Таким образом, колонки выстроятся в 3 ряда по 2+1 колонки. Для экранов мобильных телефонов, мы зададим им ширину 100%, таким образом получим 5 рядов по одной колонке.

Также, в зависимости от разрешения экрана, мы можем скрывать определённые элементы, меня позиции элементов относительно друг друга, управлять размерами шрифтов и цветами.

7.6.1. Определение ключевых точек

На первом этапе создания адаптивной верстки необходимо определиться со значениями ключевых точек – разрешениями экрана, в которых будут применяться новые стили. Для нашего проекта были выбраны ключевые точки, обозначенные в таблице 7.6.

Таблица 7.6. Ключевые точки адаптивной верстки

Разрешение экрана	Устройства
≥1280px	Мониторы, ноутбуки, телевизоры, ...
≥1000px (1000px – 1279px)	Планшеты в горизонтальной ориентации
≥768px (768px – 999px)	Планшеты в вертикальной ориентации, мобильные устройства в горизонтальной ориентации
≥480px (480px – 767px)	Мобильные устройства в вертикальной ориентации
≤479px	Дисплеи низкого разрешения

7.6.2. CSS стили адаптивной верстки

После того, как мы определились с ключевыми точками, необходимо написать дополнительные свойства элементов, для отображения в указанных в таблице 7.5 промежутках. Для указания CSS свойств для определенных экранном используется специальное ключевое слово медиа-запроса @media.

Все стили прописываются в основном файле стилей style.css. Рассмотрим на примере нескольких блоков, реализацию адаптивных свойств CSS3. Внешний вид главной страницы портала в стандартном и адаптированном виде можно увидеть на стр. 319.

7.6.3. Примеры адаптивных селекторов

Селектор ".container". В первую очередь, настроим ширину контента наших страниц, в зависимости от разрешения экрана. Нам не нужен эффект "резиновой" верстки, когда при изменении разрешения экрана, контент растягивается пропорционально. Данный эффект крайне непредсказуемый и трудноуправляемый. Вместо этого, для каждого промежутка между ключевыми точками мы задаем фиксированные параметры ширины области контента. Обновленные стили для данного селектора представлены в Листинге 7.12

Листинг 7.12. Адаптивные правила селектора ".container"

```
.container {
    margin: 0 auto;
    padding: 0px;
    padding: 0 10px;
    max-width: 1280px;
}
    @media only screen and (min-width: 1280px) {
        .container {
            max-width: 1280px;
        }
    }
    @media only screen and (min-width: 1000px) and (max-width:
1279px) {
        .container {
```

```
            max-width: 1000px;
        }
    }
    @media only screen and (min-width: 768px) and (max-width:
999px) {
        .container {
            max-width: 768px;
        }
    }
    @media only screen and (min-width: 480px) and (max-width:
767px) {
        .container {
            max-width: 480px;
        }
    }
    @media only screen and (max-width: 479px) {
        .container {
            max-width: 320px;
        }
    }
```

Селектор ".management-archive-item". В качестве следующего примера рассмотрим блок "Архив журнала Управления" на главной странице. Для большего разрешения – это пятиколончатая структура, которая должна изменяться для других разрешений. Согласно нашим правилам, для разрешений между 768px и 999px блок принимает трехколончатый вид, причем все элементы после третьего скрываются. А для экранов с разрешением ниже 768px, скрываются все элементы кроме первого (при этом элемент автоматически занимает 100% ширины родителя). Код стилей для данного селектора представлен в листинге 7.13.

Листинг 7.13. Адаптивные правила селектора ".management-archive-item"

```
.management-archive-item {
    position: relative;
    margin-bottom: 10px;
}
```

```
@media only screen and (min-width: 768px) and (max-width: 999px) {
    .management-archive-item {
        flex: 0 0 33.333333%;
        max-width: 33.333333%;
    }
    .management-archive .management-archive-item:nth-child(n+4) {
        display: none;
    }
}
@media only screen and (max-width: 767px) {
    .management-archive .management-archive-item:nth-child(n+2) {
        display: none;
    }
}
```

Селектор ".main-navigation-link". И в качестве последнего примера адаптивных стилей рассмотрим ссылку-пункт главного меню. В зависимости от разрешения устройств у пунктов меню изменяются такие свойства стилей, как размер шрифта, отступы и высота строки. Код стилей для элементов меню представлен в листинге 7.14:

Листинг 7.14. Адаптивные правила селектора

```
.main-navigation-link {
    position: relative;
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 14px 2px;
    height: 100%;
    box-sizing: border-box;
    z-index: 5;
    overflow: hidden;

    font: inherit;

    font-family: var(--title-family);
    font-size: 16px;
```

```
    font-weight: 700;
    text-align: center;
    color: #fff;
    vertical-align: top;
    text-transform: uppercase;

    background-color: #4e256f;
    border: 4px solid #4e256f;

    transition: all 0.1s linear;
}
    @media only screen and (min-width: 1000px) and (max-width: 1279px) {
        .main-navigation-link {
            font-size: 12px;
            line-height: 20px;
        }
    }
    @media only screen and (min-width: 768px) and (max-width: 999px) {
        .main-navigation-link {
            justify-content: flex-start;
            padding: 5px 0 5px 300px;
        }
    }
    @media only screen and (max-width: 767px) {
        .main-navigation-link {
            justify-content: flex-start;
            padding: 5px 0 5px 10px;
            font-size: 14px;
            line-height: 20px;
        }
    }
```

Глава 8.

Практический пример. Разработка BackEnd'а

В данной части будет реализован бэкенд полноценного портала (см. стр. 319-322), условный домен которого http://nit.center (программный код и ссылка на возможное демо — на сайте издательства nit.com.ru). Затягивать введение к этой части книги не будем, поскольку она и так будет немаленькой. Приступим.

8.1. Планирование

8.1.1. Основные функции сайта

Разработка любого сайта начинается с планирования. Не нужно считать авторов данной книги занудами. Мы не зануды, мы практики и реалисты. Сначала нужно определить функции сайта хотя бы по той причине, что далее вам придется создавать таблицы базы данных для хранения контента, следовательно, вам нужно знать, какой контент будет храниться и какие функции над ним будут доступны.

Чтобы проще было спланировать функционал сайта, пройдемся по его основным страницам:

1. **Главная страница портала** (см. рис. на стр. 319-322). Она будет содержать графический слайдер, блок полезных статей, последних новостей, несколько случайно выбранных журналов "Управление" и форму подписки. Следовательно, уже можно выделить следующий функционал и следующие таблицы:
   - Таблица **slider** для хранения информации о слайдах.
   - Таблица **library** для хранения информации о статьях. Блок отображения списка статей, соответственно, нам еще понадобится страница отображения самой статьи.
   - Таблица **news** для хранения информации о новостях. Кроме этого нам понадобится страница отображения текста новости и сценарий импорта новостей из RSS.

» Сценарий обработки формы подписки – он должен принять введенный пользователем **email** и добавить в специально отведенную для этого таблицу.

» Таблица **lf** для хранения архива журналов. Также понадобится страница отображения списка журналов.

2. **Страница Направления обучения (см. рис. на стр. 319-322)**. Наш портал подразумевает не только предоставление информации (статей), но и привлекает пользователей на курсы. На данной странице будут отображаться список курсов, преимущества нашего центра и логотипы наших партнеров. Для каждой из этих сущностей будет использоваться отдельная таблица в БД.
3. **Сертификация PMP (см. рис. на стр. 319-322)** – страница, содержащая сведения о сертификации.
4. **Архив журналов Управление. (см. рис. на стр. 319-322).** Это и есть та самая страница, о которой шла речь в пункте 1е.
5. **База знаний (см. рис. на стр. 319-322** – это не страница, а целый раздел сайта. В нем будут отображаться статьи и презентации. Для каждой из этих сущностей у нас будет отдельная таблица – **library** и **presentations** соответственно.
6. **О компании (см. рис. на стр. 319-322)** — Информационная страница О компании, информация для этой страницы будет храниться в таблице about.

Принцип работы движка

Рассмотрим несколько особенностей реализации движка сайта. Все они будут разбиты на следующие группы:

1. Генерирование страниц сайта по шаблону.
2. Понятие профиля сайта
3. Управление меню
4. Управление заголовками и футерами сайта
5. Управление наборами кнопок
6. Управление наборами SEO-данных

Генерирование страниц сайта

Генерирование страниц сайта осуществляется с помощью шаблонизатора, находящегося в файле internal/template.php, по шаблонам, находящимся в папке **skin**. При этом мы используем принцип 1 страница = 1 шаблон. Другими словами, если на сайте есть страницы статьи, списка статей, новости, списка новостей, то для каждой из этих страниц будет свой собственный шаблон.

Шаблон страницы – это HTML-файл, внутри которого содержатся переменные, заключенные в фигурные скобки, например, {sitetitle}. Задача PHP-сценария, генерирующего контент страницы – заполнить шаблон страницы (назначить значения переменным, заключенным в фигурные скобки) и вывести получившийся HTML-код в браузер.

Подробно о шаблонах мы поговорим в разделе *Шаблоны сайта*.

Понятие профиля сайта

Профиль сайта – это реестр, содержащий сведения об активных элементах сайта:

- Заголовке
- Футере
- Меню
- Наборе кнопок
- SEO-наборе.

Для большего комфорта нашим движком предусмотрено, что администратор сайта может формировать несколько заголовков, несколько футеров, меню, наборов кнопок и SEO-наборов. Затем с помощью профиля сайта осуществляется выбор нужного набора в зависимости от ситуации. Например, на период новогодних праздников можно разработать новое меню для сайта, содержащее раздел *Акции*, в котором будут различные пункты, например, *Новогодние скидки*, *Рождественские подарки* и пр. Затем с помощью профиля сайта просто выбирается новое меню из списка ранее созданных. Такой подход позволяет быстро переключать различные элементы сайта в зависимости от ситуации. Подход не нов и не является нашим "ноу хау": подобный функционал есть в панели управления WordPress, когда можно

выбрать разные меню. Мы лишь расширили эту концепцию до концепции профиля сайта, позволяющего выбрать не только другое меню, но и другие элементы сайта.

Технически профиль сайта реализован в виде таблицы site_profile. В разделе **Модель данных** будет подробно описана структура этой таблицы, и назначение ее полей.

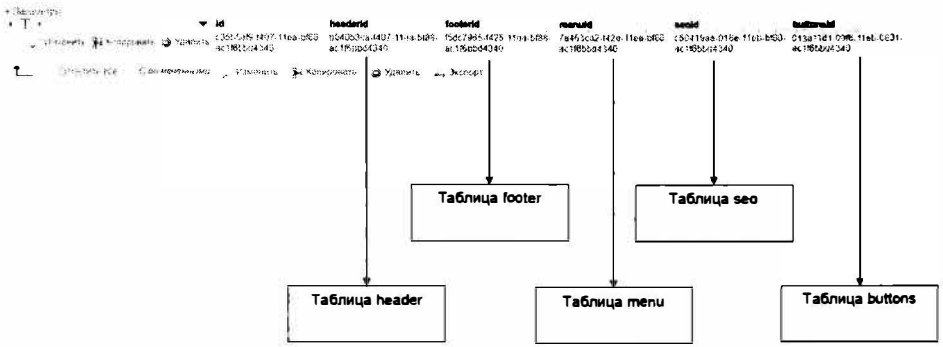


Рис. 8.1. Таблица site_profile

Управление меню

Список созданных меню хранится в таблице **menu** (рис. 8.2). Вы можете создать неограниченное количество разных меню. При создании меню указывается лишь ее заголовок, а UUID можно сгенерировать с помощью SQL-функции uuid(). SQL-запрос, позволяющий добавить новое меню выглядит так:

```
insert into menu values(uuid(), "Мое меню");
```

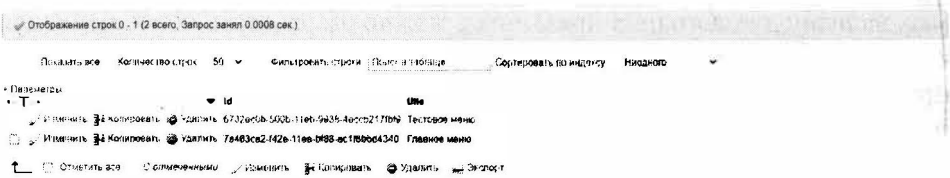


Рис. 8.2. Таблица menu

Таблица *menu_details* содержит пункты каждого меню (рис. 8.3). При этом поле menuid содержит ID меню из таблицы menu.

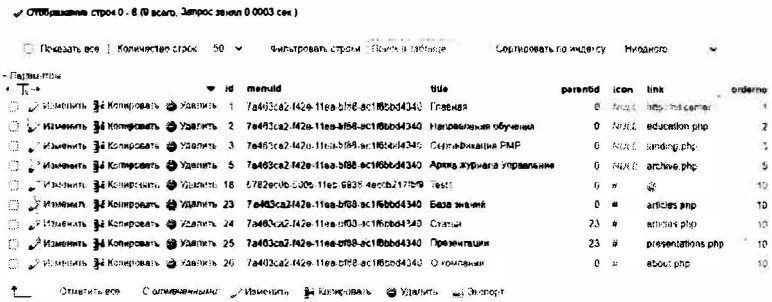


Рис. 8.3. Таблица menu_details

Когда вы создали меню, его ID нужно добавить в таблицу *site_profile*, в поле menuid, иначе меню не будет отображаться!

Управление заголовками и футерами

Аналогично меню, вы можете создать несколько заголовков и несколько футеров данных. Сведения о созданных заголовках хранятся в таблице header, о футерах – footer. Очень важным в этих таблицах является поле code (рис. 8.4).

Рис. 8.4. Таблица header

Данное поле содержит произвольный пользовательский HTML/JavaScript-код, который будет выведен:

- в конце тега <head> для кода заголовка;
- перед закрывающимся тегом </body> для кода футера.

Таблицы header_details и footer_details содержат переменные заголовка/футера соответственно. На рис. 8.5 приведен фрагмент таблицы header_details. Поле **header_id** содержит UUID заголовка, которому принадлежит переменная (из таблицы **header**). Поле **hvar** – это название переменной для использования в шаблоне сайта, а *hvalue* – значение переменной.

Рис. 8.5. Фрагмент таблицы header_details

Установка переменных осуществляется в сценарии set_vars.php следующим образом:

```
 // Заполняем переменные заголовка
$sql = "select * from header_details where header_id =
\"$header\"";
$result = $mysqli->query($sql) or die($mysqli->error);
while ($row = $result->fetch_assoc()) {
    $tpl->set_value($row['hvar'],$row['hvalue']);
}
```

Как видно из кода, сначала мы получаем все записи, относящиеся к выбранному заголовку (where header_id = \"$header\"). Далее мы в цикле проходимся по всем записям и устанавливаем переменные шаблона посредством вызова метода *set_value*. В шаблоне должны быть объявлены переменные, указанные в таблице, иначе подстановки значений не произойдет, например:

```
<!–фрагмент шаблона index.html –->
<div class="header-top-content col-lg-9">
                    <ul class="contacts-list">
                        <li class="contacts-item">
```

```
<a href="{adr1link}" class="contacts-link">
                {adr1}
            </a>
        </li>
        <li class="contacts-item">
<a href="{adr2link}" class="contacts-link">
                {adr2}
            </a>
        </li>
        <li class="contacts-item">
<a href="{adr3link}" class="contacts-link">
                {adr3}
            </a>
        </li>
    </ul>
    <ul class="tags-list">
        <li class="tags-item">
<a href="{link1url}" class="tag-link">
                {link1text}
            </a>
        </li>
```

Управление наборами кнопок и SEO-данных

Как и в случае с другими элементами сайта, вы можете сформировать несколько наборов кнопок. Работает все аналогично:

- Наборы кнопок описываются в таблице buttons, содержимое конкретного набора – в таблице **buttons_details**.
- Наборы SEO-данных описываются в таблице seo, содержимое набора – в таблице **seo_details**.
- Структура таблиц buttons, seo, buttons_details, seo_details подобна структуре таблиц для наборов заголовка и футера (есть небольшие отличия, описанные в разделе Модель данных).
- Подстановка значений осуществляется в том же сценарии set_vars.php.

Чтобы то или иное название кнопки появилось на сайте, нужно выполнить следующие действия:

1. Создать набор кнопок и добавить его UUID в таблицу **site_profile** (в поле buttonsid)

2. В таблице **buttons_details** определить название кнопки, например, subscribe и ее значение – *Подписаться*.

3. В шаблоне сайта определить кнопку так:

```
<div class="subscribe-button-wrapper">
      <button class="subscribe-button colored-btn"
  type="submit">{subscribe}</button>
</div>
```

Название кнопки заключается в фигурные скобки, как и в случае с другими переменными.

Изначально данный функционал разрабатывался для кнопок. Но, учитывая особенности его реализации, вы можете использовать его для реализации управления любым набором переменных шаблона, в том числе заголовков сайта. Никто не мешает вам написать {title} между двух тегов <h1>:

```
<h1>{title}</h1>
```

Затем определить эту переменную в наборе кнопок и ее значение будет подставлено в заголовок. Другими словами, конечная реализация дарит гибкость, позволяя администратору полностью управлять заголовками через базу данных.

8.1.2. Модель данных

Когда мы определились с основными страницами, нам нужно продумать структуру таблиц сайта. В таблице 8.1 приводится список и описание каждой таблицы сайта, чтобы вы понимали, где и что находится.

Таблица 8.1.

Название	Описание
about	Информация для страницы *О компании*. Содержит текстовые блоки, выводящиеся на этой странице.
advantages	Содержит данные блока Наши преимущества. Блок выводится на страницах *Услуги* и *О компании*.
buttons, buttons_details	Информация о наборе кнопок

callback	Данные о заказе обратного звонка
footer, footer_details	Содержат описание футера (нижней части) сайта. Первая таблица содержит список футеров и пользовательский код, выводимый в нижней части страницы (до тега </body>). Вторая содержит элементы (переменные) каждого футера, описанного в таблице *footers*. Редактирование таблиц осуществляется через панель управления.
header, header_details	Аналогично предыдущим двум таблицам, но эти две таблицы описывают заголовок сайта.
lf	Таблица архива журнала *Управление*. Содержит информация о выпусках журналов.
library	Содержит статьи, представленные на сайте.
library_cats	Содержит категории (разделы) статей. Это не контент, как в случае с library_books_sections, а категории статей – *Дистрибутивы*, *Программирование* и т.д.
menu, menu_item	Первая таблица содержит различные меню, которые могут быть на сайте, а вторая – элементы этих меню. Таблицы редактируются через панель управления.
news	Сюда вносятся новости, полученные посредством парсинга XML-фида, заданного параметром *rss* (хранится в параметрах сайта). Импорт осуществляется сценарием news_parse.php.
popular_cats	Содержит ID популярных разделов статей. Статьи из этих разделов выводятся на странице Все статьи (блок справа), сценарий articles.php
presentations	Список презентаций, отображаемый на соответствующей странице.
projects	Наши проекты. Содержит логотипы проектов/партнеров.

seo, seo_details	По аналогии с футерами и заголовками. В первой таблице описываются наборы SEO-данных, во второй – переменные каждого набора. Здесь содержатся ключевые слова и META-описания для различных страниц сайта – лендинга, услуги, о компании и т.д.
services, services_cats	Категории курсов и сами курсы
settings	Таблица хранит различные параметры сайта – заголовок, URL и др.
site_profile	Профиль сайта. В этой таблице хранятся выбранные заголовок, футер, меню, SEO-набор.
slider	Слайдер. Здесь хранится информация о слайдах, ссылках со слайдов.
subscribe	Данные о подписчиках. Заполняется посредством формы подписки.
trust	Информация для блока *Нам доверяют*.

Далее мы пройдемся по каждой из таблиц и опишем ее структуру. Вы узнаете, что означает то или иное поле каждой таблицы.

Таблица about

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `about` (
  `sname` varchar(100) NOT NULL,
  `svalue` text,
  `scomment` varchar(255) NOT NULL,
  PRIMARY KEY (`sname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.2. Описание полей таблицы about

Поле	Назначение
sname	Имя параметра
svalue	Значение параметра
scomment	Комментарий, описывающий назначение параметра

Таблица advantages

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `advantages` (
  `hvar` varchar(40) NOT NULL,
  `hvalue` varchar(250) DEFAULT NULL,
  PRIMARY KEY (`hvar`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.3. Описание полей таблицы advantages

Поле	Назначение
hvar	Имя параметра
hvalue	Значение параметра

Таблицы buttons и buttons_details

Первая таблица содержит созданные ID наборов кнопок, вторая – информацию о надписях на кнопках. Структура таблиц следующая:

```
CREATE TABLE IF NOT EXISTS `buttons` (
  `id` varchar(40) NOT NULL,
  `title` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS `buttons_details` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `buttons_id` varchar(40) DEFAULT NULL,
  `bvar` varchar(50) DEFAULT NULL,
  `bvalue` varchar(255) DEFAULT NULL,
  `bcomment` varchar(200) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Назначение полей такое же, как у таблиц header, header_details.

Таблица callback

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `callback` (
  `id` varchar(40) NOT NULL,
  `uname` varchar(100) DEFAULT NULL,
  `phone` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.4. Описание полей таблицы callback

Поле	Назначение
id	UUID-идентификатор заказа обратного звонка
uname	Имя пользователя, заказавшего звонок
phone	Номер телефона для обратного звонка

Таблица footer

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `footer` (
  `id` varchar(40) NOT NULL,
  `title` varchar(40) DEFAULT NULL,
  `code` text,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.5. Описание полей таблицы footer

Поле	Назначение
id	UUID-идентификатор футера
title	Заголовок футера (отображается только в модуле *Профиль сайта*)
code	Текст, который будет вставлен до тега </body>

Таблица footer_details

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `footer_details` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `footer_id` varchar(40) DEFAULT NULL,
  `fvar` varchar(50) DEFAULT NULL,
  `fvalue` varchar(255) DEFAULT NULL,
  `fcomment` varchar(200) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.6. Описание полей таблицы footer_details

Поле	Назначение
id	Числовой идентификатор строки, используется просто как индекс для более удобной работы с таблицей в phpMyAdmin
footer_id	Идентификатор футера из таблицы footer
fvar	Переменная футера
fvalue	Значение переменной футера
fcomment	Описание переменной футера

Таблица header

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `header` (
  `id` varchar(40) NOT NULL,
  `title` varchar(100) DEFAULT NULL,
  `slogan` varchar(255) DEFAULT NULL,
  `code` text,
  `menuid` varchar(40) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.7. Описание полей таблицы header

Поле	Назначение
id	UUID-идентификатор созданного заголовка, используется для получения элементов заголовка из таблицы header_details
title	Название заголовка, отображается только в панели управления в разделе *Профиль сайта*
slogan	Общий слоган сайта, отображается на главной странице
code	Произвольный код, вставляемый в секцию <header>
menuid	Оставлено для обратной совместимости с ТЗ БД. Не используется. ID меню указывается в профиле сайта (таблица site_profile).

Таблица header_details

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `header_details` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `header_id` varchar(40) DEFAULT NULL,
  `hvar` varchar(50) DEFAULT NULL,
  `hvalue` varchar(255) DEFAULT NULL,
  `hcomment` varchar(200) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.8. Описание полей таблицы header_details

Поле	Назначение
id	Числовой идентификатор строки, используется просто как индекс для более удобной работы с таблицей в phpMyAdmin
header_id	Идентификатор футера из таблицы **header**
hvar	Переменная заголовка
hvalue	Значение переменной заголовка
hcomment	Описание переменной заголовка

Для импорта продуктов нажмите кнопку **Импорт**. XML-фид для импорта продуктов задается в параметрах лендинга (см. след. таблицу). Кнопка **Очистить** удаляет ранее импортированные продукты.

Таблица lf

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `lf` (
  `id` varchar(40) NOT NULL,
  `year` int(11) DEFAULT NULL,
  `title` varchar(100) DEFAULT NULL,
  `totalnum` int(11) DEFAULT NULL,
  `pdf` varchar(200) DEFAULT NULL,
  `buylink` varchar(255) DEFAULT NULL,
  `content1` text,
  `content2` text NOT NULL,
  `content3` text NOT NULL,
  `content4` text NOT NULL,
  `active1` int(11) NOT NULL DEFAULT '1',
  `active2` int(11) NOT NULL DEFAULT '1',
  `active3` int(11) NOT NULL DEFAULT '1',
  `active4` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.9. Описание полей таблицы lf

Поле	Назначение
id	UUID журнала
year	Год выпуска журнала
title	Заголовок
totalnum	Общий номер журнала
pdf	PDF-файл журнала, файлы хранятся в lf/pdf
buylink	Ссылка на URL, где можно купить журнал
content1 – content4	Поля контента 1-4
active1 – active4	Флаги видимости контента 1-4 во фронте

Таблица library

Структура:

```
CREATE TABLE IF NOT EXISTS `library` (
  `id` varchar(255) NOT NULL,
  `catid` varchar(255) DEFAULT NULL,
  `url` varchar(255) DEFAULT NULL,
  `status` int(11) DEFAULT NULL,
  `title` varchar(255) DEFAULT NULL,
  `folder` varchar(255) DEFAULT NULL,
  `image` varchar(255) DEFAULT NULL,
  `author` varchar(255) DEFAULT NULL,
  `link` varchar(255) DEFAULT NULL,
  `content` mediumtext,
  `shortdescr` varchar(255) DEFAULT NULL,
  `dt` date DEFAULT NULL,
  `tags` varchar(300) DEFAULT NULL,
  `seokeys` varchar(255) DEFAULT NULL,
  `seodescr` varchar(255) DEFAULT NULL,
  `popular` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.10. Описание полей таблицы library

Поле	Назначение
id	UUID статьи
catid	UUID раздела
url	URL статьи во фронте
status	Состояние статьи, пока обрабатывается 1 – статья видима во фронте, остальное – невидима, планировались разные значения для разных состояний (модерация и т.д.), но не реализовались в этой версии
title	Заголовок статьи
folder	Папка, в которую будут загружаться картинки статьи

image	Обложка статьи (обычно называется cover.jpg и находится в папке статьи)
author	Автор
link	Ссылка на источник
content	Контент (текст статьи)
shortdescr	Аннотация
dt	Дата добавления статьи
tags	Теги
seokeys	Ключевые слова для SEO
seodescr	META-описание для SEO
popular	Флаг популярности статьи, непопулярные статьи – 0, чем выше значение тем популярнее статья

Таблица library_cats

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `library_cats` (
  `id` varchar(255) NOT NULL,
  `title` varchar(255) DEFAULT NULL,
  `descr` text,
  `parent_id` varchar(255) DEFAULT NULL,
  `active` int(11) DEFAULT NULL,
  `url` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `url` (`url`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.11. Описание полей таблицы library_cats

Поле	Назначение
id	UUID раздела (категории статей)
title	Заголовок раздела, например, *Дистрибутивы*

descry	Короткое описание раздела
parent_id	UUID родительского раздела
active	Флаг видимости раздела во фронте
url	SEF URL (articles/<url>). Поле является уникальным

Таблица menu

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `menu` (
  `id` varchar(40) NOT NULL,
  `title` varchar(70) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.12. Описание полей таблицы menu

Поле	Назначение
id	UUID идентификатор меню
title	Название меню (для внутреннего использования)

Таблица menu_item

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `menu_item` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `menuid` varchar(40) DEFAULT NULL,
  `title` varchar(40) DEFAULT NULL,
  `parentid` int(11) DEFAULT NULL,
  `icon` varchar(100) DEFAULT NULL,
  `link` varchar(255) DEFAULT NULL,
  `orderno` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.13. Описание полей таблицы menu_item

Поле	Назначение
id	Числовой идентификатор записи, для более простой работы с phpMyAdmin
menuid	UUID-идентификатор меню, к которому относится элемент
title	Заголовок (выводится во фронте)
parented	UUID родительского элемента меню
icon	Не используется (не реализовано в дизайне)
link	Ссылка элемента меню
orderno	Используется для упорядочивания пунктов меню

Таблица news

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `news` (
  `id` varchar(255) NOT NULL,
  `title` varchar(255) DEFAULT NULL,
  `dt` date DEFAULT NULL,
  `active` int(11) DEFAULT NULL,
  `content` text,
  `link` varchar(255) DEFAULT NULL,
  `tags` varchar(300) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.14. Описание полей таблицы news

Поле	Назначение
id	UUID новости
title	Заголовок новости
dt	Дата импорта новости
active	Флаг видимости новости
content	Содержимое новости

link	Ссылка на источник
tags	Тег новости

Таблица popular_cats

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `popular_cats` (
  `id` varchar(40) NOT NULL,
  `orderno` int(11) DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.15. Описание полей таблицы popular_cats

Поле	Назначение
id	UUID популярной категории
orderno	Порядок вывода категории

Таблица presentations

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `presentations` (
  `id` varchar(40) NOT NULL,
  `title` varchar(100) DEFAULT NULL,
  `author` varchar(100) DEFAULT NULL,
  `dt` date DEFAULT NULL,
  `pdf` varchar(255) DEFAULT NULL,
  `active` int(11) DEFAULT NULL,
  `url` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.16. Описание полей таблицы presentations

Поле	Назначение
id	UUID презентации
title	Заголовок презентации
author	Автор
dt	Дата презентации
pdf	PDF-файл презентации (хранятся в /presentations)
active	Флаг видимости
url	SEF URL (часть после /presentation/)

Таблица projects

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `projects` (
  `id` varchar(40) NOT NULL,
  `title` varchar(100) DEFAULT NULL,
  `url` varchar(100) DEFAULT NULL,
  `image` varchar(100) DEFAULT NULL,
  `active` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.17. Описание полей таблицы projects

Поле	Назначение
id	UUID проекта
title	Заголовок проекта (в данный момент во фронте не отображается)
url	URL проекта (ссылка)
image	Логотип проекта
active	Флаг видимости

Таблица seo

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `seo` (
  `id` varchar(40) NOT NULL,
  `title` varchar(70) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.18. Описание полей таблицы seo

Поле	Назначение
id	UUID набора SEO
title	Заголовок (используется только в панели управления)

Таблица seo_details

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `seo_details` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `seo_id` varchar(40) DEFAULT NULL,
  `hvar` varchar(50) DEFAULT NULL,
  `hvalue` varchar(255) DEFAULT NULL,
  `hcomment` varchar(200) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8mb4;
```

Таблица 8.19. Описание полей таблицы seo_details

Поле	Назначение
id	Числовой ID для более удобной работы в phpMyAdmin
seo_id	UUID набора данных
hvar	Переменная SEO-набора
hvalue	Значение переменной
hcomment	Описание переменной (для отображения в панели управления)

Таблица services

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `services` (
  `id` varchar(40) NOT NULL,
  `catid` varchar(40) NOT NULL,
  `title` varchar(100) DEFAULT NULL,
  `image` text,
  `active` int(11) DEFAULT '1',
  `orderno` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.20. Описание полей таблицы services

Поле	Назначение
id	UUID услуги/курса
catid	Категория услуги/курса
title	Название
image	Изображение курса
active	Флаг видимости
orderno	Используется для сортировки при выводе

Таблица services_cats

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `services_cats` (
  `id` varchar(40) NOT NULL,
  `title` varchar(100) DEFAULT NULL,
  `orderno` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.21. Описание полей таблицы services_cats

Поле	Назначение
id	UUID категории услуги/курса

title	Название категории
orderno	Используется для сортировки при выводе

Таблица settings

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `settings` (
  `sname` varchar(100) NOT NULL,
  `svalue` varchar(255) DEFAULT NULL,
  `scomment` varchar(255) NOT NULL,
  PRIMARY KEY (`sname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.22. Описание полей таблицы settings

Поле	Назначение
sname	Название параметра
svalue	Значение параметра
scomment	Описание назначения параметра

Таблица site_profile

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `site_profile` (
  `id` varchar(40) NOT NULL,
  `headerid` varchar(40) DEFAULT NULL,
  `footerid` varchar(40) DEFAULT NULL,
  `menuid` varchar(40) NOT NULL,
  `seoid` varchar(40) DEFAULT NULL,
  `buttonsid` varchar(40) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.23. Описание полей таблицы site_profile

Поле	Назначение
id	UUID профиля

headerid	UUID активного заголовка
footerid	UUID активного футера
menuid	UUID активного меню
seoid	UUID активного набора SEO-данных
buttonsid	Не используется

Таблица slider

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `slider` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `url` varchar(200) DEFAULT NULL,
  `alt` varchar(150) DEFAULT NULL,
  `active` int(11) DEFAULT NULL,
  `slideor` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4;
```

Таблица 8.24. Описание полей таблицы slider

Поле	Назначение
id	ID слайда
url	Ссылка со слайда
alt	Текст **alt** для слайда
active	Флаг видимости слайда
slideor	Порядок отображения слайда

Таблица subscribe

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `subscribe` (
  `id` varchar(40) NOT NULL,
  `email` varchar(100) DEFAULT NULL,
```

```
  `dt` date DEFAULT NULL,
  `active` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.25. Описание полей таблицы subscribe

Поле	Назначение
id	UUID подписчика
email	E-mail подписчика
dt	Дата подписки
active	Флаг видимости

Таблица trust

Структура таблицы:

```
CREATE TABLE IF NOT EXISTS `trust` (
  `id` varchar(40) NOT NULL,
  `title` varchar(100) DEFAULT NULL,
  `url` varchar(100) DEFAULT NULL,
  `image` varchar(100) DEFAULT NULL,
  `active` int(11) DEFAULT '1',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Таблица 8.26. Описание полей таблицы trust

Поле	Назначение
id	UUID записи в таблице *Нам доверяют*
title	Заголовок
url	Ссылка с логотипа
image	Картинка (хранятся в assets/img/parnters/)

Параметры сайта

В таблице **settings** хранятся следующие параметры сайта.

Таблица 8.27. Описание параметров сайта

Параметр	Описание
callback	E-mail менеджера, на который будет отправлено уведомление о заказе обратного звонка
rss	Источник для RSS-ленты новостей
rsstag	Когда новости будут добавляться в таблицу news, будет добавлен данный тег
site_url	**Базовый URL сайта. Указание правильного адреса важно для корректной работы CMS**
main_news_count	К-во новостей на главной
main_articles_count	К-во статей на главной
main_lf_count	К-во журналов на главной
articles_art_count	К-во статей справа на странице статьи, по умолчанию 7

8.1.3. Структура проекта

Разработав структуру базы данных, нужно позаботиться о структуре папок проекта. Все следующие папки находятся в корневом каталоге сайта – в каталоге public_html:

1. **assets** – вспомогательные ресурсы сайта, например, стили, некоторые картинки, шрифты, пиктограммы (icons) и некоторые другие файлы.
2. **js** – содержит JS-код сайта.
3. **skin** – здесь находятся шаблоны каждой страницы сайта. PHP-код с помощью шаблонизатора будет заполнять эти шаблоны данными, полученными из базы данных.
4. **content** – здесь содержатся картинки статей сайта.
5. **books** – картинки для книг сайта.
6. **presentations** – презентации в формате PDF.
7. **lf** – архив журнала Управление, картинки и PDF-файлы.

Вспомогательные ресурсы сайта во многом зависят от верстальщика и они будут отличаться для разных проектов – разные шрифты, картинки и т.д.

Каталог **js** содержит JS-код слайдера, файл main.js – основной скрипт интерфейса пользователя и некоторые дополнительные JS-скрипты.

Каталог **skin** содержит шаблоны каждой страницы, которая есть на сайте. Рассмотрим содержимое этого каталога подробнее.

8.1.4. Шаблоны сайта

Принцип генерирования HTML-кода

Принцип работы движка сайта следующий:

1. Сценарии получают необходимую информацию из БД
2. Загружаются шаблоны. Переменные в шаблонах заменяются полученной информацией из БД
3. Сгенерированный HTML-код отправляется в браузер.

Внешний вид сайта полностью зависит от шаблонов, хранящихся в папке **skin**. Назначение шаблонов по умолчанию приведено в таблице 8.28. Кэш не используется, поэтому любое изменение в шаблонах сразу должно отображаться на сайте. Если этого не происходит, то нужно очистить кэш браузера или искать причину в кэше на стороне сервера, если таковой имеется.

Список шаблонов

Таблица 8.28. Назначение шаблонов по умолчанию

Шаблон	Описание	Используется в скрипте
about-company.html	Страница *О компании*	about.php
article.html	Шаблон, выводящий контент одной статьи	article.php

articles.html	Шаблон страницы *Все статьи*	articles.php
articles-category.html	Отображение контента категории (раздела) со статьями	articles.php
index.html	Шаблон главной страницы	index.php
archive.html	Архив журнала	archive.php
news-line.html	Страница со всеми новостями	news.php
news.html	Страница с новостью	news.php
presentation.html	Страница с одной презентацией	presentations.php
presentation-line.html	Страница со всеми презентациями	presentations.php
landing.html	Лендинг PMP	landing.php
services.html	Курсы	education.php

Переменные шаблонов

Переменные в шаблонах заключаются в фигурные скобки {}. Внутри скобок может быть практически любой текст, но лучше придерживаться правил именования переменных – латинские буквы, цифры, знак подчеркивания. Пример:

```
<html>
<head>
   <title>{sitetitle}</title>
</head>
...
```

Здесь используется переменная {sitetitle}. Она будет заменена значением, сформированным сценарием.

Общие переменные заголовка

К общим переменным заголовка относят следующие переменные (они встречаются во всех шаблонах):

- {sitetitle} – заголовок, заполняется сценарием путем получения одноименной переменной из таблицы header_details
- {slogan} – слоган сайта, получается сценарием из таблицы header_details
- {k_*} – ключевые слова для данной страницы, ключевые слова берутся из активного SEO-набора
- {d_*} – описание для данной страницы, берется из активного SEO-набора.
- {k} – ключевые слова, генерируемые скриптом по выдаваемому контенту, например, для статей заполняются по полю seokeys
- {d} – описание, генерируемое скриптом, для статей берется по полю descr.
- {adrN} – адресная строка, заданная в активном заголовке (header_details)
- {adrNlink} – ссылка для адресной строки, берется из header_details.
- {linkNtext} – текст для ссылки над меню, хранится в header_details
- {link1url} – ссылка над меню (header_details)
- {MAINMENU} – переменная полностью генерируется скриптом по содержимому таблиц menu, menu_item
- {HEADERCODE} – пользовательский код, вставляемый в заголовок, хранится в таблице header

Общие переменные футера

К общим переменным футера относят следующие переменные (они встречаются во всех шаблонах):

- {copyright} – информация об авторских правах, хранится в footer_details
- {vkpage}, {instagram}, {youtubechannel}, {fbpage} – ссылки на социальные сети и Youtube-канал.
- {FOOTERCODE} – код, добавляемый в конце документа.

Первые три типа ссылок выводятся из таблицы **footer_details**, последняя ссылка – из таблицы **footer**. Такое разделение из-за типа поля таблицы. Для кода нужно поле типа *text*, нет смысла для *fvalue* назначать этот тип поля – достаточно и varchar().

Переменные главной страницы (скин index.html)

Главная страница использует следующие переменные:

- {SLIDERPREVIEW} – предварительный просмотр слайдера, формируется функцией get_slider_preview();
- {SLIDER} – код слайдера, генерируется функцией get_slider();
- {ARTICLES} – список статей, генерируется функцией get_articles();
- {NEWS} – список статей, функция get_news();

Переменные скина about-company

Данный скин содержит следующие переменные:

- {textN} – текстовые блоки, редактируемые через панель управления;
- {advN} – текстовые надписи из таблицы advantages, редактируются через панель управления;
- {PROJECTS} – список проектов, get_projects();
- {TRUSTS} – информация для блока *Нам доверяют*, функция get_trusts().

Переменные скина article

Список переменных:

- {BREAD} – хлебные крошки, генерируются автоматически;
- {FEATURED} – адрес картинки (обложка);
- {AUTHOR} – автор статьи;
- {POSTTITLE} – заголовок статьи;

- {CROPPED_TITLE} – сокращенный заголовок статьи (для SEO);
- {CONTENT} – контент статьи;
- {SIMPLELIST} – простой список статей;
- {ARTICLES} – список статей в колонке справа;
- {CAT} – раздел статей.

Переменные скина articles

Список переменных:

- {ARTICLES1} – верхний список статей;
- {ARTICLES2} – нижний список статей;
- {UP} – верхний список категорий;
- {DOWN} – нижний список категорий;
- {SECTIONS} – список доступных категорий (выводятся только видимые active = 1) категории;
- {VIDEO} – видеоролики.

Переменные скина articles-category

Список переменных:

- {SECTITLE} – название раздела;
- {LIST} – список статей раздела, основной контент;
- {ARTICLES} – список статей справа;

Переменные скина archive

Список переменных:

- {YEARS} – годы, формируется автоматически;
- {LF} – содержимое – список выпусков за выбранный год или за последний, если год не выбран.

Переменные скина news-line

Список переменных:

- {NEWS} – список новостей;
- {POSTS} – список статей.

Переменные скина news

Список переменных:

- {POSTTITLE} – заголовок новости;
- {DT} – дата;
- {CONTENT} – контент новости;
- {NEWS} – список других новостей.

Переменные скина presentation

Список переменных:

- {TITLE} – заголовок презентации;
- {DT} – дата презентации;
- {AUTHOR} – автор.

Переменные скина presentation-line

Список переменных:

- {LIST} – список презентаций, основной контент.

Переменные скина services

Список переменных:

- {advN} – текст преимуществ, редактируется через админку;
- {SERVICESBLOCK} – блок *Услуги*;
- {OFFERS} – список предложений;
- {PROJECTS} – проекты/партнеры;
- {TRUST} – содержимое блока *Нам доверяют*.

8.2. Разработка сайта

8.2.1. Назначение основных сценариев

В корневом каталоге public_html будут находиться следующие файлы:

- **about.php** – выводит страницу *О компании (см. стр. 321)*
- **article.php** – выводит страницу с контентом статьи.
- **articles.php** – выводит страницу *Все статьи* или страницу с категорией статьи, если она задана в качестве параметра (см. стр. 320).
- **autoloader.php** – важный файл, подключаемый в начале каждого сценария. Обеспечивает автоматическую загрузку всех нужных PHP-скриптов.
- **callback.php** – обработчик формы обратного звонка.
- **index.php** – основной файл (см. стр. 319).
- **library.php** – очень важный файл, содержащий библиотечные функции, которые вызываются другими скриптами
- **news.php** – отображает страницу с контентом новости или страницу со списком новостей, если конкретная новость не задана параметром.
- **presentations.php** – страница со списком презентаций.
- **education.php** – выводит информацию о курсах портала.
- **set_vars.php** – данный файл устанавливает переменные заголовка, футера, кнопок, меню.
- **internal/connect.php** – файл, использующийся для подключения к базе данных.
- **internal/config.php** – файл, содержащий параметры подключения к базе данных.
- **index.php** – основной файл (см. стр. 319).

Весь код в файлах закомментирован. Поэтому мы не будем описывать все файлы в книге, а разберемся только с основными файлами.

8.2.2. Шаблонизатор template.php

Наш проект не использует серьезный шаблонизатор вроде Blade или Twig. Вместо этого используется простенький шаблонизатор, позволяющий заменить в HTML-коде скина (шаблона) переменные вида {имя} на заданное значение. Метод get_tpl() загружает файл шаблона, метод set_value() устанавливает значение переменной, а метод tpl_parse() выводит HTML-код.

Листинг 8.1. Файл template.php - шаблонизатор

```
<?php
// класс шаблонизатора
class template_class
    {
    var $values      = array();  // переменные шаблона
    var $html;                   // HTML-код

// метод загрузки шаблона
    function get_tpl($tpl_name)
      {
      if(empty($tpl_name) || !file_exists($tpl_name))
        {
        return false;
        }
      else
        {
        $this->html  = join('',file($tpl_name));
        }
      }

// метод установки значения
    function set_value($key,$var)
      {
      $key = '{' . $key . '}';
      $this->values[$key] = $var;
      }

// парсинг шаблона
    function tpl_parse()
      {
      foreach($this->values as $find => $replace)
            {
            $this->html = str_replace($find, $replace, $this-
>html);
            }
      }
    }
```

```
// экземпляр класса
$tpl = new template_class;
?>
```

8.2.3. Файл autoloader.php – подключение необходимых файлов

Чтобы наш сайт работал, как нужно, в каждый сценарий нужно добавить строчку:

```
include('autoloader.php');
```

Содержимое этого файла очень простое:

Листинг 8.2. Файл autoloader.php

```
<?php
include('internal/config.php');
include('internal/connect.php');
include('library.php');
include('internal/template.php');
?>
```

Он ничего не делает, просто подключает нужные сценарии.

8.2.4. Основной сценарий index.php

Благодаря шаблонизатору и include-файлам, код нашего главного файла **index.php** очень компактный (лист. 8.3).

Листинг 8.3. Код файла index.php

```
<?php
  // Подключаем необходимые файлы
  include('autoloader.php');
  // Загружаем слайдер
  $slider_preview = get_slider_priview(3);
  $slider = get_slider(3);

  // Загружаем основной шаблон
  $tpl->get_tpl('skin/index.html');
  // Устанавливаем переменные шаблона
  // Предварительный просмотр и сам слайдер
  $tpl->set_value('SLIDERPREVIEW',$slider_preview);
```

```
  $tpl->set_value('SLIDER',$slider);
  // Новости
  $tpl->set_value('NEWS', get_news(get_settings('main_news_
count')));
   // Статьи
  $tpl->set_value('ARTICLES',   get_articles(get_
settings('main_articles_count')));
  $tpl->set_value('LF', get_lf(get_settings('main_lf_
count')));
  // Видео
  $tpl->set_value('VIDEO', get_video_block());

  // Главное меню
  $menu = get_active_menu();
  $mainmenu = get_menu($menu, 1);
  $tpl->set_value('MAINMENU', $mainmenu);
  // Устанавливаем переменные заголовка и футера
  include('set_vars.php');

  $tpl->tpl_parse();
  echo $tpl->html;

?>
```

Принцип работы сценария прост – сначала он получает данные из БД, например, функция get_menu() возвращает HTML-код меню сайта. Сама функция определена в **library.php**. Затем наш скрипт, получив HTML-код, устанавливает его в качестве значения для переменной шаблона MAINMENU. После чего методом tpl_parse() скрипт преобразует внутреннее представление шаблона в HTML-код, который выводится оператором *echo*.

Обратите внимание на следующие вызовы:

```
  $tpl->set_value('NEWS', get_news(get_settings('main_news_
count')));
```

Мы не просто получаем новости из базы данных. Мы сначала получаем значение переменной main_news_count из таблицы **settings** с помощью функции get_settings() (она также описана в library.php). После этого мы получаем новости из БД, передав в функцию get_news() количество новостей из настроек сайта.

8.2.5. Сценарий about.php

Сценарий **about.php** (стр. О компании — стр. 321) похож по своему характеру на index.php. Он также получает необходимые данные с помощью функций, определенных в library.php.:

```
// Получаем необходимые данные
$menu = get_active_menu();                  // Выбранное меню
$mainmenu = get_menu($menu, 7);             // Активный пункт меню
7
$trust = get_trust();                       // Нам доверяют
$projects = get_projects();                 // Проекты


// Устанавливаем переменные шаблона
$tpl->set_value('MAINMENU', $mainmenu);
$tpl->set_value('TRUST', $trust);
$tpl->set_value('PROJECTS', $projects);
```

8.2.6. Сценарий article.php – отображение статьи

Данный сценарий не очень простой, поэтому мы не будем приводить сразу весь его исходный код, а рассмотрим частями.

Первым делом мы получаем URL статьи и пытаемся найти URL с таким адресом:

```
$id = filter_var($_GET['id'], FILTER_SANITIZE_MAGIC_QUOTES);
// Выбираем только из активных статей, чтобы никто не мог
// просмотреть скрытую статью
// по старой ссылке
$sql = "select * from library where url = \"$id\" and
status = 1 limit 1";
$result = $mysqli->query($sql) or die($mysqli->error);
```

Если количество строк в результате > 0, то мы что-то делаем. Если же 0, то это означает, что статьи нет, и мы должны перенаправить пользователя на страницу **404**:

```
if ($result->num_rows > 0) {
...
}
else {
    // перенаправляем на 404
    header('HTTP/1.1 404 Not Found');
    die();
}
```

Если же статья существует, мы выполняем следующие действия:

```
// В массив post получаем запись со всей информацией о
// статье
    $post = $result->fetch_assoc();
// Заполняем переменные для большего удобства
    $post_title = $post['title'];
    $post_author = $post['author'];
    $post_content = $post['content'];
// Формируем обложку статьи
    $image = str_replace('../','', $post['image']);
```

```
    if (strlen($image) == 0) $image = $site_url . "assets/
img/articles/article2.jpg";
    else $image = $site_url . $image . '?' .
rand(1111,9999);
// Получаем название категории, в которой находится статья
    $cat = get_cat_name($post['catid']);
// Получаем список из 10 статей этой же категории для
//формирования списка
// Другие статьи
    $sl = get_articles_simple_list(10, $post['catid']);
    $catid = $post['catid'];  // Понадобится нам далее
```

Затем все эти переменные будут использоваться для формирования значений переменных шаблона:

```
$tpl->set_value('CONTENT', $post_content);
$tpl->set_value('AUTHOR', $post_author);
$tpl->set_value('FEATURED', $image);
$tpl->set_value('CAT', $cat);
$tpl->set_value('ROOTCAT', 'Библиотека');
$tpl->set_value('SIMPLELIST', $sl);
$tpl->set_value('BANNER', get_banner());
$tpl->set_value('FULLPATH', $site_url);
```

Нам также нужно сформировать список хлебных крошек для нашей статьи. Сначала мы формируем массив категорий:

```
$cats = array();
$cats[] = $catid;  // добавляем в массив ИД категории статьи
$current = $catid; // текущий элемент = номер ИД категории статьи
while ($current !== 'root') {
    $current = get_parent_cat($current);
    if ($current !== 'root')
        $cats[] = $current;
}
```

Чтобы иерархия была правильной, нужно инвертировать наш массив, иначе хлебные крошки будут выводиться в обратном порядке:

```
$cats = array_reverse($cats, true); // обратная сортировка по ключам
```

Осталось сформировать HTML-код, отображающий эти самые крошки:

```
$breads = '';
foreach ($cats as $v) {
    $url = get_cat_url($v);
    if ($url == '#')
    $breads .= "<li><a href=\"$site_url" .
"articles.php\">" . get_cat_name($v) . "</a></li>";
    else
    $breads .= "<li><a href=$site_url" .
"articles/$url>" . get_cat_name($v) . "</a></li>";
}
```

Нужно только не забыть добавить значение этой переменной в шаблон:

```
$tpl->set_value('BREAD', $breads);
```

Вот теперь вы точно знаете, какая строчка и за что отвечает в файле article.php. Полный код не публикуем, вы его найдете в прилагаемом к книге архиве.

8.2.7. Вывод страниц – articles.php

Сценарий **articles.php** отвечает за отображение сразу двух страниц:

1. Страница со списком статей категории – если задан параметр **id** (URL категории).
2. Страница *Все статьи*, отображающая популярные статьи в избранных категориях.

Рассмотрим его первую часть. Так как сценарий довольно большой, весь его код мы рассматривать не будем. В прилагаемом к книге архиве вы найдете полный код и сможете его прочитать (код прокомментирован).

Итак, первым делом нам нужно получить URL категории и посмотреть, если такая категория. Если такой категории нет, мы перенаправляем браузер на страницу 404 – генерируем ответ 404:

```
// Выводим список статей категории
$id = filter_var($_GET['id'], FILTER_SANITIZE_MAGIC_QUOTES);

// Получили url нужно найти id категории
```

```
    $sql = "select * from library_cats where url = \"$id\"
and active = 1 limit 1";
    $result = $mysqli->query($sql) or die('Error');
    if ($result->num_rows == 0) {
        header('HTTP/1.1 404 Not Found');
        die();
    }
```

Учитывая инструкцию die(), если категория статей не найдена, то мы "убиваем" сценарий и его выполнение прекращается.

Далее мы получаем ID статьи (UUID идентификатор):

```
$row = $result->fetch_assoc();
$id = $row['id'];
```

Используя этот идентификатор, мы делаем вот что. Мы получаем из таблицы **library** список всех статей, которые принадлежат категории с идентификатором $id:

```
    $sql = "select * from library where catid = \"$id\" and
status = 1 order by popular desc";
    $result = $mysqli->query($sql) or die('Error');
    $list = '';          // HTML-код
    $articles_count = $result->num_rows;  // счетчик статей
```

Опять-таки наши действия зависят от того, были ли найдены в этом разделе статьи:

```
  if ($result->num_rows > 0) {
  // Отображаем список статей
  }
  else {
   // Статей нет, пытаемся вывести список подкатегорий, если
они есть
  }
```

Список статей мы отображаем так:

```
while ($row = $result->fetch_assoc()) {
        $image = str_replace('../','', $row['image']);

        if (strlen($image) == 0)
```

```
$image = $site_url . "assets/img/articles/article2.jpg";
        else
$image = $site_url . $image . '?' . rand(1111,9999);

        $list .= "<li class=\"articles-item\">
          <div class=\"article-image\">
            <img src=\"$image\" width=\"150\" height=\"150\"
alt=\"Обложка статьи\">
          </div>
          <div class=\"article-description\">
            <h3>$row[title]</h3>
            <p>
                $row[shortdescr]
            </p>
            <a href=\"$site_url"
. "article/$row[url]\" class=\"button\">$read_more_text</a>
        </div>
          </li>";
    }
```

А вот так мы выводим список подкатегорий. Кроме него мы также и выводим список статей в каждой из подкатегории, чтобы пользователь сразу мог выбрать понравившуюся ему статью:

```
$list = "<P align=center>Выберите подраздел:
        <ul class=\"article-tags-list\">" . get_section_
list($id) . "</ul>";
        // Ищем статьи в подразделах
        $sql = "select * from library_cats where parent_id
= \"$id\" and active=1";
        $res3 = $mysqli->query($sql) or die($mysqli-
>error);

        while ($row3 = $res3->fetch_assoc()) {
     // Ищем только активные статьи, где status = 1
            $sql = "select * from library where catid =
\"$row3[id]\" and status = 1";
            $res4 = $mysqli->query($sql) or die($mysqli-
>error);

            while ($row4 = $res4->fetch_assoc()) {
                $image = str_replace('../','',
$row4['image']);
```

```
                $articles_count++;

                if (strlen($image) == 0)
$image = $site_url . "assets/img/articles/article2.jpg";
                else
$image = $site_url . $image;

                $list .= "<li class=\"articles-item\">
                <div class=\"article-image\">
                    <img src=\"$image\" width=\"150\"
height=\"150\" alt=\"Обложка статьи\">
                </div>
                <div class=\"article-description\">
                    <h3>$row4[title]</h3>
                    <p>
                        $row4[shortdescr]
                    </p>
                    <a href=\"$site_url"
. "article/$row4[url]\" class=\"button\">$read_more_text</a>
                </div>
                </li>";

            }
        }
```

Зачем нам переменная $articles_count. Список статей может быть разным – все зависит от категории. В одной категории может быть 2-3 статьи, в другой – 10 статей. Посредством $articles_count мы регулируем другой список – список статей, который выводится в колонке справа:

```
if ($articles_count < 6) $n = 2;
if ($articles_count >= 6 && $articles_count < 10) $n = 5;
if ($articles_count > 10) $n = 10;

$tpl->set_value('ARTICLES', get_articles($n));
```

Так мы достигаем примерной визуальной длины двух списков – основного – и в sidebar. В принципе, можно было бы реализовать пагинацию, но для статей на нашем портале не очень много и пагинация, мы посчитали, лишняя.

8.2.8. Сценарий callback.php

Данный сценарий не генерирует никакой HTML-код, а просто принимает значение – номер телефона и вносит его в базу данных.

```
include('autoloader.php');

$email = get_settings('callback');

$name = filter_input(INPUT_POST,
            'feedback-name', FILTER_SANITIZE_MAGIC_QUOTES);
$phone = filter_input(INPUT_POST,
            'feedback-phone', FILTER_SANITIZE_MAGIC_QUOTES);

if (strlen($name) > 0 && strlen($phone) > 0) {

$sql = "insert into callback values(uuid(), \"$name\",
\"$phone\")";
$result = $mysqli->query($sql) or die('error');

mail($email,
      "Callback request",
      "Заказ обратного звонка от $name, номер телефона
$phone");

}
```

Обратите внимание, как мы получаем значения, которые вносим в базу. Мы все их фильтруем, используя фильтр FILTER_SANITIZE_MAGIC_QUOTES, который применяет функцию addslashes() к переменной из POST-запроса. Без этого, если переменная содержит кавычки, у нас не получится добавить ее в базу данных, поскольку ее значение попросту "разорвет" запрос. После добавления запроса на обратный звонок в БД, мы отправляем сообщение по e-mail менеджеру.

8.2.9. Сценарий news.php

Сценарий **news.php** отображает все новости или же конкретную новость, если задан ее ID в качестве параметра. Сценарий довольно простой (лист. 8.4).

Листинг 8.4. Фрагмент сценария news.php

```
 if (!isset($_GET['id'])) {
   // Выводим новостную ленту, так как параметр id не задан
   $tpl->get_tpl('skin/news-line.html');
   // Выводим 50 последних новостей
   $tpl->set_value('NEWS', get_news(50));
   $tpl->set_value('POSTS', get_articles(5));

   $menu = get_active_menu();
   $mainmenu = get_menu($menu);

    $tpl->set_value('MAINMENU', $mainmenu);

   include('set_vars.php');

   $tpl->tpl_parse();
   echo $tpl->html;
 }
 else {
     // Задан параметр id, отображаем новость
     $id = filter_var($_GET['id'], FILTER_SANITIZE_MAGIC_
QUOTES);
     $sql = "select * from news where id = \"$id\" limit 1";
     $result = $mysqli->query($sql) or die($mysqli->error);
     $post = $result->fetch_assoc();

     $tpl->get_tpl('skin/news.html');
     $tpl->set_value('NEWS', get_news(5));

     $menu = get_active_menu();
     $mainmenu = get_menu($menu);

     $tpl->set_value('MAINMENU', $mainmenu);

     $tpl->set_value('POSTTITLE', $post['title']);
     $tpl->set_value('DT', $post['dt']);
     $tpl->set_value('CONTENT', $post['content']);
     $tpl->set_value('TAG', $post['tags']);
     $tpl->set_value('BANNER', get_banner());

   include('set_vars.php');

   $tpl->tpl_parse();
   echo $tpl->html;

 }
```

8.2.10. Сценарий set_vars.php

Это очень важный сценарий, который вызывается практически всеми сценариями фронтэнда. Он устанавливает переменные заголовка, футера, наборов кнопок, SEO-набор. Если не вызвать данный сценарий, то переменные заголовка, футера, названия кнопок и SEO-данные не будут установлены.

Листинг 8.5. Фрагмент сценария set_vars.php

```
// Получаем активные заголовок, футер, SEO-набор, набор кнопок
$header = get_active_header();
$footer = get_active_footer();
$seo = get_active_seo();
$buttons = get_active_buttons();
// Получаем пользовательский код заголовка и футера
$tpl->set_value('HEADERCODE',get_header_code($header));
$tpl->set_value('FOOTERCODE',get_footer_code($footer));

// Заполняем переменные заголовка
$sql = "select * from header_details where header_id =
\"$header\"";
$result = $mysqli->query($sql) or die($mysqli->error);
while ($row = $result->fetch_assoc()) {
    $tpl->set_value($row['hvar'],$row['hvalue']);
}
// То же самое для набора кнопок
$sql = "select * from buttons_details where buttons_id =
\"$buttons\"";
$result = $mysqli->query($sql) or die($mysqli->error);
while ($row = $result->fetch_assoc()) {
  $tpl->set_value($row['bvar'],$row['bvalue']);
}
// Для SEO-деталей
$sql = "select * from seo_details where seo_id = \"$seo\"";
$result = $mysqli->query($sql) or die($mysqli->error);
  while ($row = $result->fetch_assoc()) {
    $tpl->set_value($row['hvar'],$row['hvalue']);
  }

// Заполняем переменные футера
$sql = "select * from footer_details where footer_id =
\"$footer\"";
$result = $mysqli->query($sql) or die($mysqli->error);
while ($row = $result->fetch_assoc()) {
    $tpl->set_value($row['fvar'],$row['fvalue']);
}
```

8.2.11. Сценарий sub.php

Данный сценарий используется для обработки формы подписки. Он получает, подобно callback.php, e-mail пользователя и добавляет его в таблицу:

```
$email = filter_input(INPUT_POST,
            'subscribe-email',
            FILTER_SANITIZE_MAGIC_QUOTES);

$sql = "insert into subscribe values(uuid(), \"$email\",
now(), 1)";
$result = $mysqli->query($sql) or die($mysqli->error);
```

8.2.12. Библиотека функций library.php

Самый большой файл, содержащий много всевозможных библиотечных функций. Мы не будем рассматривать каждую функцию, вместо этого рассмотрим таблицу, содержащую пояснение каждой используемой функции. Зная параметры и описание функции, вам будет проще читать код сценариев сайта.

Таблица 8.29. Библиотечные функции, определенные в library.php

Функция	Параметры	Описание
get_settings()	$sname – имя параметра	Возвращает значение параметра из таблицы **settings**
get_lf()	$n – количество, $year - год	Возвращает n журналов из архива, year – нужный код, по умолчанию год = 0, в этом случае выводится последний доступный год
get_cat_name()	$id – ID категории	Возвращает название категории статей по ее ID

get_articles_simple_list()	$n – количество, $id – категория	Возвращает простой список статей из категории ID. Возвращаются только включенные статьи (status=1), количество ограничивается параметром n. По умолчанию id не указывается, в этом случае статьи не ограничиваются по категории, а только по количеству
get_slider_priview()	$n – количество	Выводит $n превью слайдеров
get_slider()	$n – количество	Выводит $n слайдов в баннере на главной
get_news()	$n – количество	Выводит 10 (по умолчанию) включенных новостей
get_books()	$n – количество, $class – класс	Выводит 6 (по умолчанию) статей, при этом используется класс CSS, заданный параметром $class
get_menu()	$id – ID меню $active – ID активного раздела	Выводит меню, ID меню задается параметром $id, элемент, заданный вторым параметром, делается активным
get_articles()	$n – количество	Выводит $n статей, сортировка идет по полю popular по убыванию. Ограничение popular > 0 и status = 1
get_active_header(). get_active_footer(), get_active_seo(), get_active_menu()	-	Возвращают активные (выбранные в профиле сайта) заголовок, футер, SEO-набор и меню соответственно
get_header_code(), get_footer_code	$id – ID заголовка/футера	Возвращает код заголовка или футера соответственно

get_trust()	-	Возвращает содержимое блока *Нам доверяют*
get_projects()	-	Содержимое блока *Наши проекты*
get_banner()	-	Выводит заполненный блок баннера (см. раздел 7)
isAbsoluteUrl	$url	Возвращает *true*, если указан абсолютный путь, *false* – относительный
get_button_name()	$button_name – ID кнопки	Возвращает название кнопки из активного набора кнопок
get_active_buttons()	-	Возвращает ID активного набора кнопок
get_cat_url()	$id – ID категории	Возвращает URL категории
isActiveSection()	$id – ID категории	true, если ID – активная категория, false – во всех остальных случаях

8.2.13. Парсинг новостей RSS

Парсинг новостей осуществляется сценарием news_parse.php (лист. 8.6).

Листинг 8.6. Фрагмент сценария news_parse.php

```
// Получаем адреса RSS-источника и тега из настроек сайта
$rss = get_settings('rss');
$tag = get_settings('rsstag');

$url = $rss;
// Получаем XML-файл RSS-источника
$content = file_get_contents($url);
// Парсинг файла с помощью класса SimpleXmlElement
$items = new SimpleXmlElement($content);

$count = 0;

// Добавляем каждую найденную новость в базу данных
foreach($items -> channel -> item as $item) {

      // Проверяем, есть ли такой title в БД
      $sql = "select * from news where title = \"$item-
      >title\"";
      $result = $mysqli->query($sql) or die($mysqli->error);
      if ($result->num_rows > 0) continue; // пропускаем

      $title = filter_var($item->title, FILTER_SANITIZE_
      MAGIC_QUOTES);
      $descr = filter_var($item->description, FILTER_SANITIZE_
      MAGIC_QUOTES);
      $link = filter_var($item->guid, FILTER_SANITIZE_MAGIC_
      QUOTES);

      echo "<p>Importing news: $title";

      $sql = "insert into news values(uuid(), \"$title\",
      now(), 1, \"$descr\", \"$link\", \"$tag\")";
      $result = $mysqli->query($sql) or die($mysqli->error);
      $count++;

}
```

Мы не просто добавляем новости в базу данных, мы проверяем существование новости. Если новость существует, мы пропускаем ее.

На этом все. В следующей части мы поговорим, как развернуть прилагаемый к книге архив.

Глава 9.

Разворачивание архива на хостинге

9.1. Подготовительные мероприятия

9.1.1. Выбор хостинга

Прежде, чем мы разместим наш сайт, нужно выбрать хостера – иначе не будет, где его размещать. Хостинг – это место для размещения вашего сайта. Физически он представляет собой сервер, на котором будут храниться ваши файлы. Но хостинг хостингу - рознь и от правильно выбранного хостинга зависит многое. Главное – это удовлетворение пользователей вашего сайта. Если хостинг будет медленным или будет часто не работать, то ваша аудитория уйдет к конкурентам.

Хостинг бывает платным и бесплатным. Сразу нужно отбросить всякие иллюзии по поводу бесплатного хостинга. Бесплатным бывает лишь сыр в мышеловке – "бесплатность" владелец хостинга компенсирует размещением рекламы на каждом сайте, место под которое было предоставлено. Другими словами, когда ваш сайт будет открываться, будет загружаться реклама от хостера. Мало кому это понравится.

Поэтому нужно выбирать платный хостинг, тем более что тарифы не очень высокие. На рис. 9.1 показаны средние тарифы от одного из хостеров. В среднем в месяц вам придется платить около 200 рублей – не так уж и много.

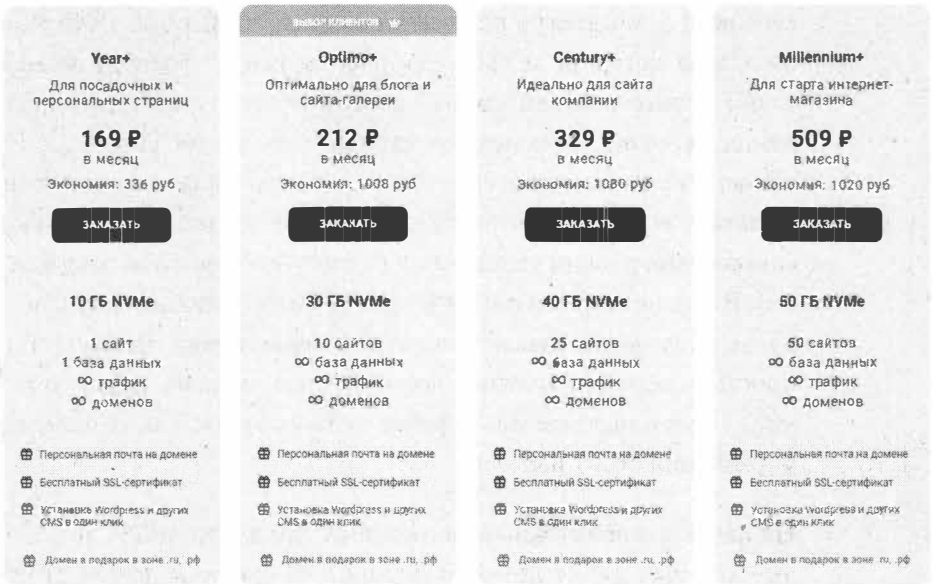


Рис. 9.1. Тарифы

При выборе хостинга нужно обратить внимание на следующие моменты:

- **Тарифы** – тариф не должен быть слишком высоким, но и не должен быть очень дешевым. Очень дешевый тариф говорит зачастую о низком качестве предоставляемых услуг. Правда, некоторые хостеры умудряются предоставлять низкокачественные услуги по средней или даже высокой цене. Тариф должен вас устраивать. Также обратите внимание на способы оплаты – вам должно быть удобно платить за хостинг.

- **Наличие поддержки PHP и MySQL** – это ключевое требование для работы нашего сайта. Если хостинг использует другой стек технологий, он нам не подойдет.

- **Служба поддержки 24/7** – круглосуточная служба поддержки означает, что если ночью произойдет авария, будем кому заняться устранением последствий и время простоя сайта будет сокращено до минимально возможного.

- **Наличие удобной панели управления** – часто хостеры предоставляют демо-доступ на несколько дней, чтобы вы могли оценить ваш комфорт и скорость работы сайта. Воспользуйтесь такой возможностью, чтобы оценить простоту панели управления. Хорошо, если панель стандартная вроде DirectAdmin, cPanel. Такие панели всем известны и даже если вы с ними не знакомы, есть документация. Некоторые хостеры используют самописные панели управления (панели собственной разработки). В них нет ничего плохого, просто одни – удобные, другие – очень неудобные. Бывает так, что на выполнение привычной и простой операции тратишь полчаса, пока найдешь нужную команду в непонятном интерфейсе пользователя. Таких панелей управления стоит избегать.

- **Наличие всевозможных акционных предложений** – некоторые хостеры дарят домен в подарок. В среднем домен стоит 800-1000 рублей, поэтому такой подарок означает, что вы сэкономите эту сумму при выборе данного хостинга. Также очень часто делается скидка при оплате сразу за год, причем скидка весьма существенная – 10-20%. Так что используйте все возможности сэкономить.

- **Наличие бесплатного SSL-сертификата** – все современные сайты уже давно перешли на SSL. Поисковые машины снижают рейтинги сайтов, работающих по протоколу HTTP, а не HTTPS. Многие хостеры предоставляют бесплатные SSL-сертификаты. Это означает, что вы сэкономите 1000 – 1500 рублей в год на покупке SSL-сертификата. Этот момент тоже нужно учитывать!

- **Отзывы о хостере** – когда выбрали несколько вариантов, поищите в Интернете отзывы о каждом из них. Отзывам слепо верить нельзя (их могут написать конкуренты), но для понимания общей картины с ними нужно ознакомиться.

Когда вы выбрали подходящий вариант, нужно поговорить о выборе тарифа. Самый дешевый тариф, как правило, подразумевает размещение толь-

ко одного сайта. Может, сегодня вам нужен только один сайт, а завтра вам захочется создать еще один и тогда придется менять тариф или покупать еще один аккаунт. Посмотрите на рис. 9.1. Если сравнивать тарифы Year+ и Optimo+, то второй значительно лучше – только дискового пространства у него в три раза больше, он позволяет создавать 10 сайтов, а количество баз данных – не ограничивается. Первый же тариф позволяет разметить только 1 сайт и 1 базу данных. Поверьте, этого будет мало.

9.1.2. Во сколько обойдется хостинг?

Нужно считать затраты не за месяц, а за год. Например, на рис. 9.1 оптимальный тариф стоит 212 рублей. Стоимость за год – 2544. К этой стоимости нужно добавить еще стоимость домена и SSL-сертификата. Домены в зонах .ru и .рф стоят дешево – порядка 179 рублей в год, именно поэтому некоторые хостеры дарят их при покупке хостинга. Домен в зоне .com стоит значительно дороже – 920 рублей.

Здесь нужно понимать, какой домен вам нужен и какие цели вы преследуете. Если целевая аудитория – только пользователи Российской Федерации, нет смысла переплачивать за домен .com, вполне достаточно домена .ru. В остальных случаях есть смысл присмотреться к международным доменам.

Сертификат в зависимости от вендора стоит от 1000 рублей. Если посчитать все вместе и по максимуму, то в год получится около 5000 рублей. Но вы можете выбрать домен .ru и получить его бесплатно при покупке хостинга, можете использовать бесплатный сертификат Let's Encrypt. В итоге получится сумма только за хостинг – 2544 рубля. Неплохая экономия почти в два раза, правда?

9.1.3. Установка дополнительного программного обеспечения

Для подключения к хостингу вам понадобится, как минимум, FTP-клиент. Мы рекомендуем FileZilla – это бесплатный FTP-клиент, который можно скачать по адресу **https://filezilla.ru/**.

Также вам понадобится удобный текстовый редактор для правки некоторых файлов перед их загрузкой на хостинг. Мы рекомендуем бесплатный редактор MS Visual Studio Code – лучший выбор для Windows-платформы:

https://code.visualstudio.com/.

Если у вас нет еще этих программ, установите их прямо сейчас.

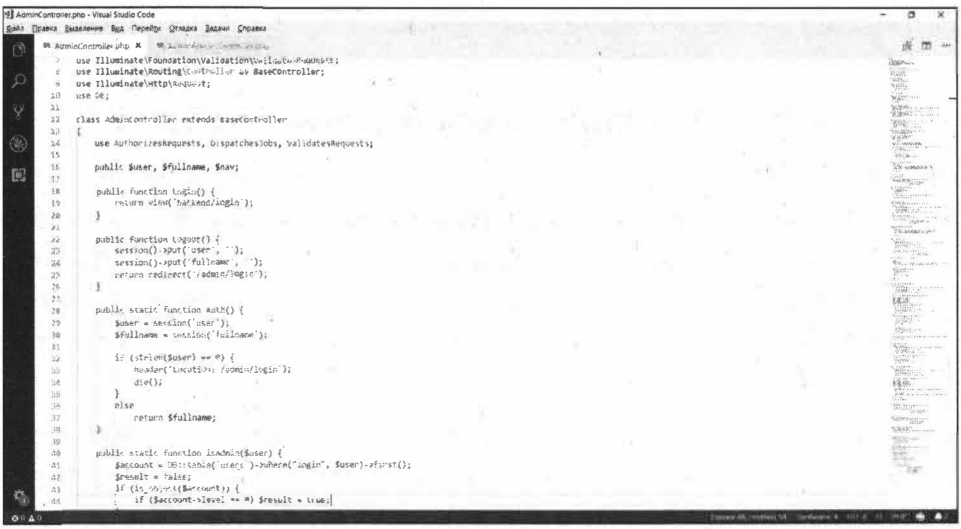


Рис. 9.2. Редактор MS Visual Studio Code

9.2. Подробная инструкция

Шаг 1: Создание базы данных

Первым делом нужно создать базу данных. Конкретные инструкции зависят от используемой панели управления. В панели управления Timeweb необходимо зайти в раздел **Базы данных** и нажать ссылку **Создание новой базы данных** (рис. 9.3).

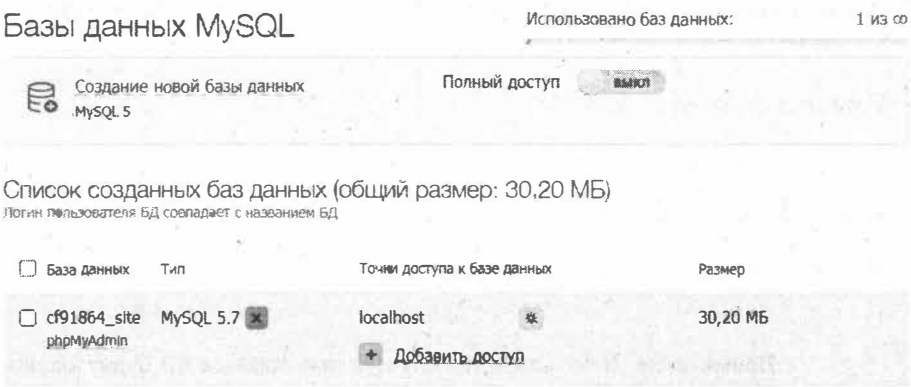


Рис. 8.3. Раздел Базы данных в панели управления хостингом Timeweb

В появившемся окне нужно ввести название базы данных и пароль. Пользователь базы данных будет создан автоматически, и его имя будет совпадать с названием базы данных (рис. 9.4).

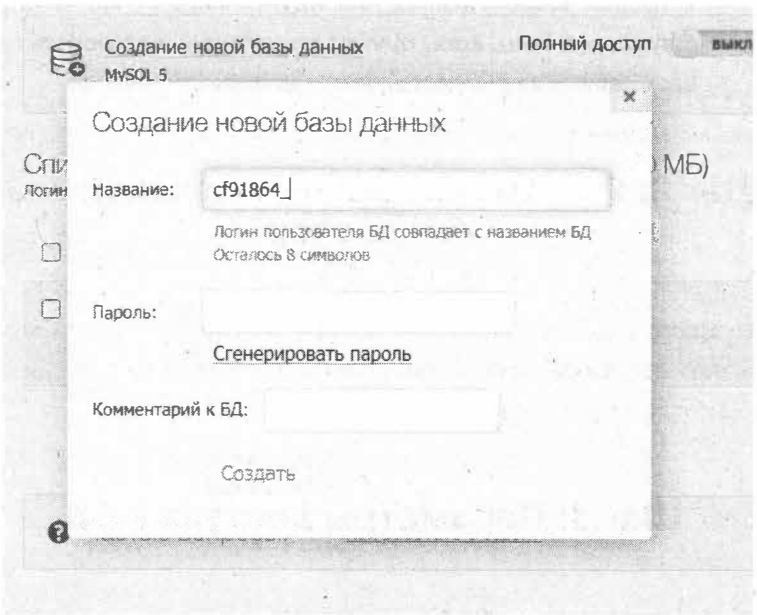


Рис. 9.4. Создание новой БД

Если у вас локальный сервер вроде XAMPP, то параметры для доступа к БД будут такими:

1. Имя пользователя – *root*
2. Пароль – пустая строка (не задан)
3. Имя базы данных – **test**
4. Имя сервера – localhost

Примечание. В большинстве случаев имя сервера БД будет localhost, если иное не указано в панели управления базами данных.

Примечание. Некоторые хостеры ограничивают количество баз данных. Если вы не можете создать новую базу данных, можете использовать существующую, только убедитесь, что названия таблиц (см. часть 7, раздел "Модель данных") не будут совпадать с именами уже существующих таблиц. Хотя, конечно же, лучше использовать отдельную базу данных во избежание непредвиденных ситуаций.

Шаг 2: Распаковка архива на локальный компьютер

Скачайте архив с сайта nit.com.ru и распакуйте на локальный компьютер. Для распаковки можете использовать любой архиватор, поддерживающий формат ZIP, в том числе встроенный архиватор Windows.

Шаг 3: Параметры доступа к БД

Откройте файл internal/config.php и укажите собственные параметры для доступа к базе данных.

```
<?php
// Параметры для доступа к БД
$host = "localhost";
$db = "название вашей БД";
$dbuser = "имя пользователя";
$dbpass = "пароль";
?>
```

Шаг 4: Загрузка файлов на хостинг

Далее следует подключить к хостингу по FTP или SSH и закачать все распакованные файлы (кроме файла с дампом БД *.sql) в каталог public_html. Не спешите обращаться к сайту – пока у вас ничего не выйдет. Для загрузки файлов по FTP мы рекомендуем использовать приложение FileZilla, а по SSH – Bitvise SSH Client. Оба приложения – бесплатные. Параметры доступа по FTP и SSH вы можете узнать в панели управления хостинга или уточнить в службе поддержки хостингом.

Примечание. Иногда папка, в которую нужно загрузить файлы, называется htdocs или просто html. Если вы не видите папку public_html, вероятнее всего, она называется htdocs или html. Если не можете разобраться, обратитесь в службу поддержки.

Шаг 5: Импорт базы данных

В архиве есть файл с дампом базы данных (*.sql). Откройте приложение phpMyAdmin через панель управления хостингом и перейдите на вкладку **Импорт**. Выберите прилагающийся SQL-файл и нажмите кнопку **Вперед** (рис. 9.5).

При импорте базы данных будут загружены демо-статьи и прочие материалы, чтобы сайт не выглядел пустым при запуске, а также будут созданы необходимые для его работы таблицы.

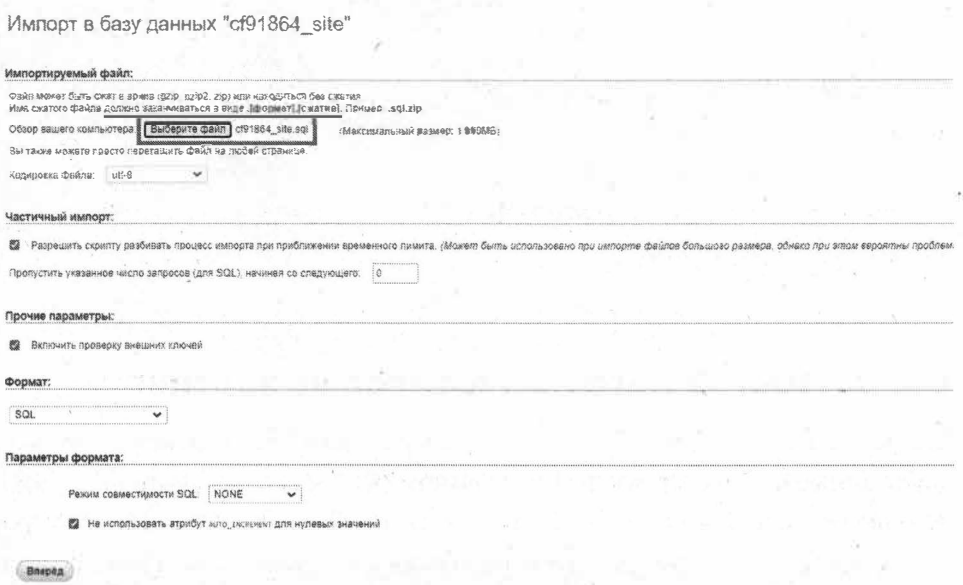


Рис. 8.5. Импорт дампа базы данных

Шаг 6: Правим адрес сайта

С помощью phpMyAdmin откройте таблицу settings и измените параметр site_url, в качестве его значения нужно указать доменное имя вашего сайта, например, http://my.site.ru.

После этого можно обратиться по только что указанному адресу и вы увидите главную страницу вашего нового сайта (рис. 9.7).

На этом все..

Показать все | Количество строк: 50 | Фильтровать строки: Поиск в таблице | Сортировать по индексу: Ниодного

Параметры

	sname	svalue	scomment
Изменить Копировать Удалить	articles_art_count	7	К-во статей в блоке справа (на странице статьи)
Изменить Копировать Удалить	banner_image	/assets/img/article-p/banner-img-1.jpg	Изображение баннера-заглушки (отображается, если н...
Изменить Копировать Удалить	banner_options	target=_blank	Параметры ссылки баннера
Изменить Копировать Удалить	banner_url	#	Ссылка с баннера-заглушки
Изменить Копировать Удалить	banner_width	300	Ширина баннера
Изменить Копировать Удалить	callback	admin@localhost	E-mail для отправки уведомлений об обратном звонке
Изменить Копировать Удалить	main_articles_count	6	К-во статей на главной
Изменить Копировать Удалить	main_lf_count	5	К-во журналов на главной
Изменить Копировать Удалить	main_news_count	17	К-во новостей на главной
Изменить Копировать Удалить	rss	https://www.vedomosti.ru/rss/rubric/business	Источник для RSS-ленты новостей
Изменить Копировать Удалить	rsstag	business	Тэг, добавляемый при импорте RSS
Изменить Копировать Удалить	site_url	http://rd.center/	Базовый URL сайта

Отметить все | С отмеченными: Изменить Копировать Удалить Экспорт

Рис. 9.6. Изменение адреса сайта

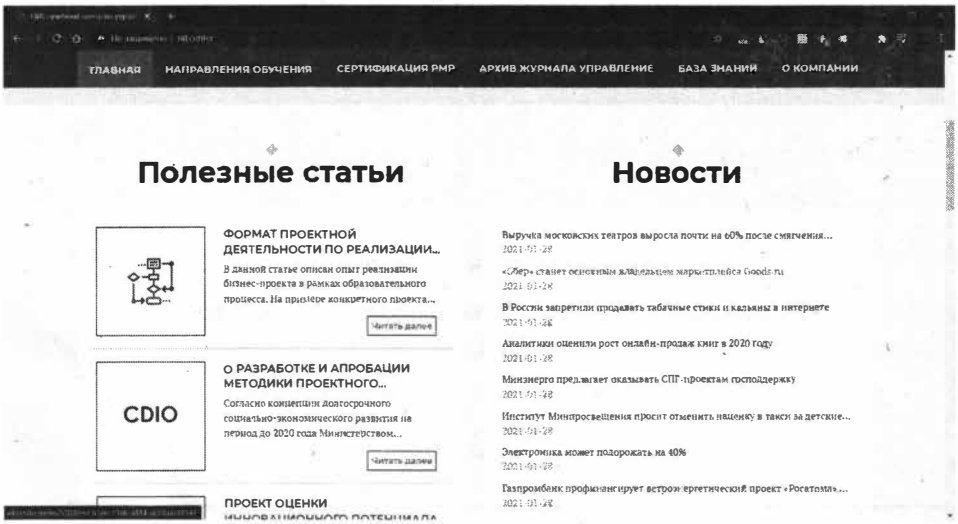


Рис. 9.7. Сайт готов к работе

Кириченко А.В., Никольский А.П., Дубовик Е.В.

Web на практике

CSS, HTML, JavaScript MySQL, PHP

для fullstack-разработчиков

Группа подготовки издания:

Зав. редакцией компьютерной литературы: *М. В. Финков*

Редактор: *Е. В. Финков*

Корректор: *А. В. Громова*

12+

ООО «Наука и Техника»

Лицензия №000350 от 23 декабря 1999 года.

192029, г. Санкт-Петербург, пр.Обуховской обороны, д. 107.

Подписано в печать 01.02.2021. Формат 70x100 1/16.

Бумага газетная. Печать офсетная. Объем 27 п. л.

Тираж 1400. Заказ 1023

Отпечатано с готовых файлов заказчика
в АО «Первая Образцовая типография»,
филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ»
432980, Россия, г. Ульяновск, ул. Гончарова, 14

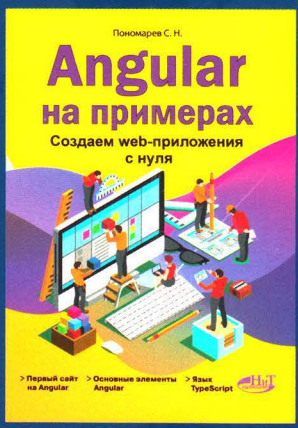
Кириченко А.В., Никольский А.П., Дубовик Е.В.

WEB НА ПРАКТИКЕ

CSS, HTML, JavaScript, MySQL, PHP для fullstack-разработчиков

В данной уникальной практической книге описаны все основные технологии, представляющие собой полный стек (full stack) web-технологий, предназначенный для полноценной разработки современных сайтов и интернет-порталов. Рассмотрен пример создания многофункционального интернет-портала с описанием разработки фронтенда и бэкенда.

Издательство «Наука и Техника» рекомендует:



www.nit.com.ru

ISBN 978-5-94387-271-6

9 78- 5- 94387- 271- 6

Издательство "Наука и Техника"
г. Санкт-Петербург

Для заказа книг:
(812) 412-70-26
e-mail: nitmail@nit.com.ru
www.nit.com.ru