

DEMIAN KOSTELNY

PHP

JUNIOR

KIT

2018

Предисловие от автора

Развитие языков программирования растёт с большой скоростью. С каждым годом появляются новые языки программирования, фреймворки и библиотеки для того чтобы облегчить жизнь программисту. Можно было бы уже сказать что языки программирования которым уже больше 20 лет являются уже устаревшими и непригодными для написания приложений в наше время. Но по-настоящему все обстоит по-другому.

Языки программирования которым уже больше 20 лет продолжают совершенствоваться и становиться все лучше и лучше. Или даже языки программирования которым где то 60 лет до сих пор используются, примером такого очень старого языка может служить FORTRAN который был создан компанией IBM в 50-х годах XX столетия.

Одним из таких немного старых языков программирования является PHP, который появился в 1995. Этот язык можно назвать первым динамическим языком программирования для создания динамических веб-сайтов. 80% веб-сайтов всего мира были написаны на этом легендарном языке программирования. Этот пользуется популярностью и по сегодняшний день многими программистами.

Каждый проводит множество конференций где встречаются множество разработчиков, программистов, студентов которые программируют на этом языке программирования. Я считаю что этот язык программирования проживёт ещё очень долго и он будет становиться все лучше и лучше.

Надеюсь что после того как читатель прочитает мою книгу и возьмет с неё знания, останется довольным и будет продолжать программировать и расти в этой отрасли. Все что надо, это имеет силу духа и желание чтобы изучать программирование ну и желательно также знать основы языка-разметки HTML. Поэтому желаю вам удачи в изучении программирования.

0 Добро пожаловать в PHP!

Изучение какого-то языка программирования всегда начинается с того что автор знакомит читателя с небольшой историей изучающего языка программирования, и рассказывает о том где его можно использовать. Именно в этой главе мы рассмотрим где и как можно использовать PHP, а также установим нужное нам программное обеспечение для программирования.

0.1 Что такое PHP?

В предисловии уже было сказано что PHP это язык программирования, но какой именно? Поскольку языки программирования также разделяются на какие то группы. Во первых это язык веб-программирования, то есть язык программирования для создания веб-приложения, веб-сайтов и т.д. Во вторых это язык Объектно-Ориентированного-Программирования, то есть PHP работает по принципу ООП (Объектно-Ориентированного-Программирования) и в этом языке все построено на объектах, классах и т.д., но об этом мы поговорим позже.

Об том что такое ООП пойдет ещё речь в этой книге, сейчас читателю рано знать что это такое. PHP был создан в 1995 году программистом Расмусом Лердоформ. Кстати Расмус уже давно пошёл с разработки и улучшения этого языка программирования.

Думаю вас заинтересовала как расшифровывается аббревиатура PHP. А расшифровывается она так: Hypertext Preprocessor -

“Препроцессор Гипертекста”. Проект имеет открытый код и распространяется по собственной лицензии. Сейчас над PHP работает группа энтузиастов с компании Zend Engine.

Думаю в некоторых читателей возник вопрос о том что такое динамические веб-приложения. Динамическое веб-приложение это как бы сказать “живое” приложение, само слово динамика значит движение. То есть динамическое веб-приложение как бы движется постоянно, к примеру веб-приложение может работать с БД(База Данных) и пользователем одновременно.

Я надеюсь что читатель ознакомился с основами языка разметки HTML, и знает что на HTML создаются статические веб-страницы на которых ничего не происходит, кроме отображения внешнего вида. То есть, примеру через HTML не возможно можно сделать так чтобы делались запросы к базе данных. Статика - это когда нет движения, если говорить на языке художников.

0.2 Как работает сервер

PHP не встроенный в браузер как HTML. Он работает отдельно на своем сервере. И перед тем как мы приступим к установке нужного для нас программного обеспечения, мы должны понять как именно работает сервер, поскольку мы будем устанавливать локальный сервер на котором и будет установлен PHP. Думаю некоторые читатели сразу же не поймут о чем я тут говорю, поскольку только что я сказал что мы будем устанавливать локальный сервер с PHP.

Некоторые возмущаются, поскольку на их мысль язык программирования не надо устанавливать, а его можно же использовать сразу. Нет, язык программирования надо устанавливать, к примеру для того чтобы мы могли программировать на языке программирования Ruby нам надо будет установить его в на свой ПК. И такое же самое надо будет сделать с языком Python, если мы конечно хотим его использовать. Ничто не откуда не берется. PHP не встроенный в браузер как HTML, он

работает на отдельном сервере. И поэтому мы должны установить локальный сервер, если мы хотим программировать на PHP.

Хорошо, а теперь давайте вернемся к установке локального сервера. Когда вы к примеру пишете адрес какого то сайта в адресной строке в браузере, и потом переходите на сайт, то вы посылаете запрос серверу на котором находится тот сайт адрес которого вы ввели в адресной строке. Сервер вам отвечает содержимым сайта, и после получения ответа вы можете спокойно лазить по сайту и просматривать его содержимое. То есть сервер имеет запрос и ответ.

То есть, запрос делает пользователь для того чтобы получить какие-то данные от сайта, а сервер в свою очередь отвечает пользователю данными. Существуют разные виды запросов, кроме тех запросов что используются для того чтобы получить содержимое сайта. К примеру, пользователь может сделать запрос на то чтобы внести какие-то данные в базу данных веб-сайта.

Об том какие существуют виды запросов и как их можно делать с помощью PHP мы поговорим в разделе «PHP и HTTP». Чтобы лучше вам запомнилось как работает сервер я вам нарисовал внизу небольшую простую схему, показанную на Рис. 0:



- *Рис. 0 – Схема показывающая работу сервера*

Теперь когда я вам объяснил поверхностно то как работает сервер, мы можем приступать к установке нужного для нас программного обеспечения и локального сервера.

0.3 Установка локального сервера

Выбор локального сервера - это довольно таки важный вопрос. Поскольку вам понадобится локальный сервер с поддержкой PHP 7.0 или выше. Кроме этого надо чтобы локальный сервер имел HTTP сервер Apache. Но это ещё не все, нам понадобится также сервер с поддержкой БД (Баз данных) и СУБД(Система-Управления-Баз-Данных) MySQL.

На счастье нам не надо будет все устанавливать по отдельности, поскольку уже существуют полноценные сборки в которых уже есть установлены все нужные для нас компоненты. Мы будем использовать локальный сервер XAMPP, он имеет все те компоненты для разработки. Можете заглянуть на официальный сайт по адресу:

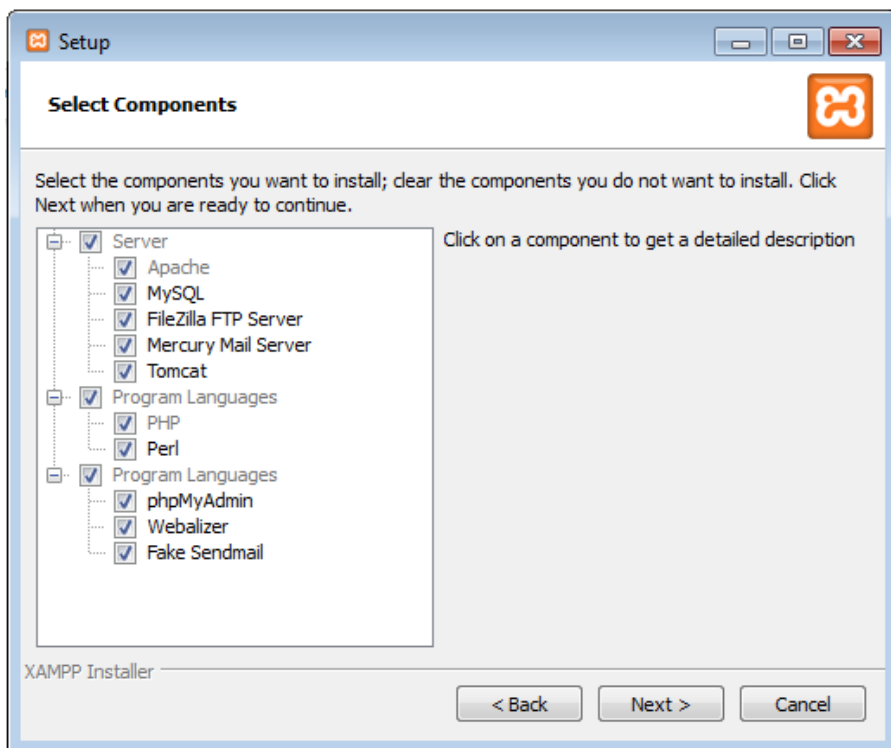
<http://apachefriends.org/ru/index.html>.



- *Рис. 1 - Главная страница официального сайта локального сервера XAMPP*

Выбирайте XAMPP для вашей операционной системы. Пока я пишу эту книгу, то я использую ОС Windows. Желательно чтобы был

ХАМРР с версией PHP 7.0 или выше. Скачайте установщик ХАМРР. После того как установщик загрузится откройте его. После того как установщик откроется вы увидите сначала сообщение о том что это установщик ХАМРР. Далее вам будет предложено выбрать компоненты которые вы хотите установить как это показано на Рис.1



- Рис.2 - Выбор установки компонентов в установщике ХАМРР

Поскольку мы будем устанавливать ХАМРР по умолчанию, то мы не будем ничего изменять в списке компонентов. Далее нам уже надо будет выбрать место куда мы хотим установить локальный сервер. Обычно по умолчанию ХАМРР устанавливается в корневую папку диска C:.

Вы конечно же можете сами выбрать место на вашем компьютере где вы хотите установить XAMPP. После того как вы выбрали место для «next», то начнется процесс установки XAMPP. Если все установилось и прошло успешно, то тогда мы можем продолжать. А теперь я объясню где и что находится. В главной папке находятся 3 приложения с префиксом xampp. Все они нужны для того чтобы запустить, остановить и контролировать локальный сервер XAMPP.

К примеру приложение `xampp_start.exe` нужно для того чтобы запустить локальный сервер, а `xampp_stop.exe` чтобы остановить локальный сервер. Запустите локальный сервер, это может занять немного времени. После этого откройте браузер и **перейдите по адресу** `localhost/`. В браузере должна будет появиться домашняя страница локального сервера.

Это страница локального сервера где мы можем посмотреть настройки локального сервера, узнать информацию о том что нам делать, посмотреть какие расширения установлены и т.д.. Если у вас будет отображаться содержимое и будет приветствие, то значит что у вас запустился локальный сервер и правильно работает.

0.4 Редактор кода

У нас есть локальный сервер, значит можно приступить к выбору редактора кода. Думаю некоторые читатели зададутся вопрос, а зачем нам редактор кода? Редактор кода, как вы уже смогли понять – это приложение которое позволяет редактировать код, но суть в том что это как обычный редактор текста, только есть подсветка синтаксиса языка программирования, нумерация строк и многое другое что позволяет эффективно писать код.

Вообще можно использовать встроенный редактор текста в вашей операционной системе. К примеру в Windows это программа “блокнот”, но будет намного лучше использовать редактор кода, поскольку у вас будет подсветка синтаксиса как я уже и говорил, и подсветка ошибок в

коде если они будут в коде. Я рекомендую использовать редактор кода Sublime Text 3, им пользуется много программистов, и я также. Он бесплатный и его очень легко использовать. Но кроме этого редактора кода также существует приложение Notepad++, которое имеет похожие функции на Sublime Text.

Если вы установили локальный сервер и редактор кода, то мы можем переходить к следующему разделу и начинать уже учиться программировать на PHP.

1 Первые шаги

Когда новичок в программировании начинает изучать какой то язык программирования, то есть такая традиция программистов как написать свою первую программу, которая бы выводила на экран одно сообщение “Hello World!”, т.е. «Привет мир!». Поэтому не будем нарушать традиций программистов и повторим эту же традицию. Заодно вы получите свой первый опыт с написанием простой программы. Но перед этим как мы начнём это делать, нам надо разобраться с тем как создать директорию для нашего небольшого приложения.

В папке `xampp`(папка локального сервера) есть директория `htdocs`, именно в этой папке и **хранятся приложения** которые мы будем запускать на локальном сервере. Создайте папку `app` в этой папке, это и будет папка с нашей первой программой. А теперь откройте редактор кода и скопируйте этот код:

```
<?php
    echo "Hello world!";
?>
```

После этого сохраните этот код в папке нашего первого приложения, назовите файл `index` и сохраните его в **формате** `*.php`. Запустите локальный сервер если он у вас ещё не был запущен, потом перейдите в браузер по адресу `localhost/app/index.php`. Вы видите надпись «Hello world!»? Если да то тогда это значит что вы сделали все правильно! Можно продолжать двигаться дальше, и следующим с чем надо будет разобраться это синтаксис языка программирования PHP.

Так званные PHP теги `<?php` и `?>` создают часть кода PHP который будет выполняться. То есть когда локальный сервер будет запускать файл то когда он прочитает эти теги то начнёт выполнять код который внутри этих тегов. Это если говорить на простом языке. Следующее что идет после наших тегов это команда `echo`, которая

служит для того чтобы выводить на экран текстовые данные которые внутри кавычек. Это вы может заметить в коде. К примеру, в нашем первой программе в кавычках было записано «Hello world!», и после того как мы запустили этот скрипт в браузере, то на странице вывелась та самая надпись которая была в кавычках. То есть, надпись «Hello world!».

В конце команды `echo` мы поставили точку с запятой, объясню зачем она ставится. Точка с запятой ставиться чтобы завершать выполнение какой то определенной команды в коде. Такое есть во многих языках программирования. Если мы не поставим точку с запятой, то произойдет ошибка и после того как мы перейдем на страницу с нашим приложением то мы увидим только сообщение от PHP что произошла синтаксическая ошибка и будет показан номер строки в которой не закрыли команду точкой с запятой.

Точки запятой надо всегда ставить, для новичка в программировании это будет смотреться немного дико, но в примерах этой книги вы увидите где и как ставятся точки с запятой в коде. Думаю я объяснил все довольно таки простым и популярным языком, и надеюсь вы смогли понять о чем именно я говорю. Дальше будут идти более сложные примеры чтобы вы могли усвоить материал. Поэтому переходим к следующему подразделу.

1.1 Комментарии в коде

Очень нужной вещью в программировании и в написании кода являются комментарии. Комментарии в PHP это строки которые можно увидеть только в коде, их суть заключается в том чтобы дать какие то подсказки в определенной части кода для разработчика. Когда программа начинает выполняться то она игнорирует комментарии. То есть вы можете писать все то что захотите в комментариях, и эти строки не будут прочитаны, поскольку они будут просто проигнорированы. Если к примеру в комментариях будет записана какая то команда то она проста не выполниться, а будет проигнорирована интерпретатором PHP.

Интерпретатор это программа которая выполняет определенный скрипт. Комментарии объявляются с помощью двоих слэшов:

```
<?php
    echo "Эта команда выполниться";
    // А эта строка будет проигнорирована
    // echo "Ну а эта команда будет
    // также проигнорирована";
    echo "Программирование это круто"; // Эта
строка кода будет выполнена
?>
```

Я думаю читатель задаст вопрос, в чем же смысл комментариев если они игнорируются интерпретатором? Поверьте, когда вы будете писать какое-нибудь большое приложение с тысячами строками кода, а также с кучей всяких функций то вам понадобится использовать комментарии в коде чтобы описать что делает та или иная функция чтобы не заблудиться.

То есть, к примеру, у вас будет большая функция которая имеет 300 строк кода, чтение этих 300-от строчек кода может занять время. И поэтому чтобы не читать все эти строчки кода, можно будет просто прочесть комментарий который вы создали чтобы на случай такой ситуации понять что делает данная функция.

Кроме двоих слэшов комментарии можно также объявлять так:

```
<?php
    # Это также комментарий

    /*
        А это вообще блок с комментариями.
        Здесь можно писать много чего
    */
?>
```

И вообще в книге автор будет много раз использовать комментарии чтобы описать некоторые части кода в примерах. Это чтобы читателю было легче понять что делает тот или иной скрипт.

1.2 Переменные в PHP

Я надеюсь что читатель учил алгебру и помнит о таком математическом понятии как переменные. Но не надо сразу же думать что если вы хотите научиться программировать на PHP, то надо сразу же учить алгебру. **Чтобы стать программистом математику не надо знать.**

Если вы не знаете что такое переменные, то давайте я вам приведу небольшую аналогию с ящиками. Переменные можно описать как ящик которому вы даете **название** и можете что то положить в него. К примеру мы можем назвать наш ящик `pizza` и спрятать внутри него бумажку с надписью “Паперони”.

А теперь если объяснить все это на техническом языке то выходит так, что переменная имеет свое **название** и **значение**. В нашем примере ящик был примером простой переменной, мы дали название ящику `pizza`, то есть название переменной, а также дали значение переменной, а точнее присвоили значение «Паперони». То значение которое мы присвоили нашей переменной имеет текстовый тип данных, что такое типы данных мы поговорим дальше. А сейчас давайте лучше поговорим о том как можно объявлять переменные в PHP.

Надеюсь что вы поняли что такое переменная и что теперь мы сможем начать реализовать переменные в PHP. Переменные в PHP объявляются с помощью **знака доллара \$**, после чего мы вводим название переменной(без пробела после знака доллара) и присваиваем ей значение с помощью оператора `=`. Пример создания переменной в PHP:

```
<?php
    $pizza = "Паперони"; // Здесь мы присвоили
текстовые данные и дали название pizza
?>
```

Попробуйте сейчас создать свою переменную с со своим названием на примере только что показанного примера. Если бы в PHP не было бы переменных то я очень сомневаюсь что смог бы вообще существовать этот язык программирования. Существуют также **правила по объявлению переменных** которых надо придерживаться если вы хотите чтобы ваш код работал правильно и не было никаких ошибок.

Во первых название переменной не должно начинаться с цифр. Во вторых название переменной должно быть только на **английском языке**. И во третьих, после знака доллара **не должно быть пробела**. Придерживайтесь этих правил написания и у вас все получится без ошибок.

1.3 Немного об типах данных

Переменная может иметь разные типы данных. К примеру в нашей программе где мы создали первую переменную, наша переменная имела текстовый тип данных то есть обычный кусочек текста. Но кроме текстовых данных есть также другие типы данных, к примеру числа:

```
<?php
    $int = 456;

?>
```

Вообще в PHP есть 9 типов данных, вы только что познакомились с числами и строками(текстовым типом данных). С числами можно делать **арифметические** операции. К примеру можно сделать сложение с помощью оператора +:

```
<?php
    $int1 = 5;
    $int2 = 34;
    echo $int1 + $int2;

?>
```

После того как вы сохраните файл с этим кодом и перейдете на вашу веб-страницу то вы должны увидеть число 39. В нашем коде мы объявили две переменные и дали им числовые данные после чего мы сложили их с помощью математического оператора `+`, и потом с помощью команды `echo` вывели результат на экран. Об математических операциях мы говорим немного позже а сейчас посмотрите на таблицу которая показана внизу, в ней приведены **основные типы данных**:

Тип данных	Значение	Пример
String	Текстовые данные	Hello world!
Intenger	Числовые данные	3848
Float	Число с плавающей точкой	6.02
Boolean	Логические данные	true
array	Массив	Mike, Alex, John

- Табл. 1 - Основные типы данных в PHP

С текстовым и числовым типом данных вы уже знакомы, но как на счёт других типов данных? Точка с плавающей точкой это почти то же самое что и простые числа, только число разделяет точки на десятки, тысячные и т.д. (просто вспомните немного математику).

Об логических данных будет отдельный подраздел поскольку это очень интересная и немного сложная тема. Ну и об массивах мы поговорим также немного позже, а сейчас мы будем только говорить об строках.

1.4 Строки в PHP

Вы уже познакомились со строками и знаете что их можно создавать с помощью **двойных кавычек**. Но строки можно также создавать и **одинарными кавычками**:

```
<?php
    echo 'Строка с одинарными кавычками';
?>
```

Довольно таки просто, следующая тема о которой надо поговорить это операторы для работы со строками.

1.4.1 Операторы для работы со строками

Операторов для работы со строками в PHP есть только 2, поэтому этот подраздел будет очень и очень кратким как предыдущий. Первый оператор нужен для того чтобы **объединить** две строки:

```
<?php
    echo "Это супер " . "строка";
    // Результат: Это супер строка
?>
```

Этот оператор можно будет использовать если надо будет к примеру вывести какую-нибудь переменную:

```
<?php
    $login = "SuperMan";
    echo "Логин: " . $login;
?>
```

Второй оператор используется для того чтобы присоединиться к верхнему аргументу и вывести все в одну строку:

```
<?php
    $top = "Hello ";
    $down .= "world";
    // Результат: Hello world
```


?>

1.4.2 Управление Последовательностью

Вы уже знаете что можно создавать строки с помощью одинарных и двойных кавычек. Но если мы к примеру попробуем отобразить одинарную кавычку внутри одинарных кавычек то получим ошибку. Или если к примеру попробуем отобразить двойную кавычку в двойных кавычках то также получим синтаксическую ошибку. В чем же проблема?

Проблема в том что РНР когда видит открывающуюся двойную кавычку думает что это начало строки и потом видит вторую двойную кавычку то есть ту кавычку которую мы и хотим отобразить в двойных кавычках, интерпретатор думает что эта кавычка закрывает строку и идет дальше но потом видит снова кавычку и думает что это начало строки.

То есть выходит так что РНР как бы нам говорит что мы не закрыли строку. Чтобы решить такую проблему надо всего лишь **поставить слэш** около кавычки, которую мы хотим отобразить в кавычках такого же вида:

```
<?php
    echo 'I\'m superman!';
    // Результат: I'm superman!
    echo "Book was named \"Superhero\"";
    // Результат: Books was named "Superhero"
```

?>

Это надо будет использовать если нам надо будет к примеру отобразить название нашей книги как показано в нашем примере. Хочу сказать очень важную вещь по поводу **одинарных кавычек**, есть очень большой минус. В одинарных кавычках **не работают последовательности** а также не отображаются переменные которые будут заключены внутри одинарных кавычек.

Поэтому использовать одинарные кавычки надо только в тех случаях если мы хотим вывести только строку, но можно и отобразить переменную присоединив её к строке с помощью оператора “.”. Стоп, а

что же такое последовательности? Автор все время говорит о каких то последовательностях, но так и не сказал что это такое.

Последовательности это как бы сказать маленькие команды со слэшем которые мы вставляем в строку, и с помощью таких последовательностей можно манипулировать строкой. То есть изменять что то в строке. Но вообще для манипулирования строки используются **специальные функции** о которых мы поговорим позже, а сейчас посмотрите на таблицу под номером 2 в которой показано какие именно есть последовательности в PHP.

Последовательность	Значение
\n	Новая строка
\r	Возвращение каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\e	Escape знак
\\$	Знак доллара

- Табл. 2 - Некоторые последовательности в PHP

А теперь после того как мы выяснили что это такое можно поговорить о том как и где можно использовать эти команды. Но хочу добавить что я не добавил в таблицу все последовательности которые есть в этом языке программирования. По-настоящему их намного больше, но я показал самые востребованные.

Последовательность \n - Новая строка

Как ясно из значения этой последовательности то она создает новую строку. Вместо того чтобы писать два раза `echo` можно просто воспользоваться данной последовательностью:

```
<?php
echo "С этого получится \ndве разные строки";
// Результат: С этого получится
//  две разные строки
echo "С этого получится";
echo "две разные строки";
// Результат: С этого получится две разные
строки
?>
```

С использованием этой последовательности ваш код будет выглядеть намного красивее и ещё будет использована намного меньше памяти о которой мы поговорим намного позже.

Последовательность \r - Возвращение каретки

Как вы поняли эта последовательность называется возвращение каретки. Такое понятие как возвращение каретки существовал ещё до появления программирования, ещё в те времена когда появились первые печатные машинки с клавиатурой.

Чтобы перейти в начало строки на такой машинке надо было вернуть каретку в начальное положение чтобы продолжить писать текст. Вот как это реализовано в PHP:

```
<?php
echo "Hello \rworld!";
// Результат: Hello
// world!
?>
```

Последовательность \t - Горизонтальная табуляция

Описать действие данной последовательности можно как **выравнивание** строк:

```
<?php
echo "Hello";
echo "\t world!";
// Результат: Hello   world!
?>
```

Аналогично работает также и вертикальная табуляция.

Последовательность \e - Escape знак

А эту табуляцию используют чтобы вывести этот знак ←, то есть escape знак:

```
<?php
echo "Знак escape: \e";
// Результат: Знак escape: ←
?>
```

Заключение об использовании последовательностей

В заключение можно сказать что последовательности можно использовать для того чтобы **манипулировать строками** но все таки лучше использовать специальные функции которые предназначены для работы со строками о которых мы поговорим немного позже.

А сейчас же я рекомендую читателю попрактиковаться с данными последовательностями. Ну а теперь настало время поговорить об синтаксисе определения строк heredoc и nowdoc.

1.4.3 Синтаксис heredoc и nowdoc

Ещё одним способом определения строки является синтаксис **heredoc** и **nowdoc**. Пример использования heredoc:

```
<?php
$label = <<<HEREDOC
    Вот здесь и находиться наша
    строка или текст.
HEREDOC;
?>
```

А вот пример использования nowdoc:

```
<?php
$label = <<<'NOWDOC'
    И здесь также может быть
    наша строка или текст.
NOWDOC;
?>
```

Вопросы и упражнения для самоконтроля

1. Что такое комментарии в PHP?
2. Как можно создать переменную?
3. Что такое типы данных?
4. Создайте несколько переменных с числовым и текстовым типом данных.
5. Какие есть правила создания названия для переменной?
6. Какие операторы можно использовать для строк?
7. Назовите основные последовательности и для чего их можно использовать?
8. Напишите небольшую программу которая использовала бы последовательности.

2 Числа

После ознакомления с одним типом данным можно переходить и к другому типу, то есть к числам. Синтаксис объявления чисел вы уже видели в первой главе. Числа в PHP могут иметь разные разряды, к примеру системой счисления которой мы каждый день пользуемся называется десятичной.

То есть когда мы считаем от 1 к 9 и потом у нас идет 10. Это и можно назвать десятичной системой счисления. Есть также восьмеричная система счисления это когда мы считаем от 1 к 7 и после 7 у нас уже идет не 8 а 10. И потом если в восьмеричной системе будет считать от 10 к 17 то у нас потом будет идти 20 и т.д. О разрядах чисел есть написано много книг, в нашем случае в этой книге мы будем пользоваться только десяти разрядной системой счисления. Пример создания чисел:

```
<?php
    $pizza = 8493;
    $nothing = -445; // Отрицательное число
?>
```

2.1 Операторы для работы с числами

Как и для строк так и для чисел существуют свои операторы для работы с ними. В таблице приведенной ниже показаны операторы которые можно использовать для работы с числами.

Оператор	Значение	Результат
+\$a	Идентичность	Перевод числа в целое

		число или число с плавающей точкой.
<code>-\$a</code>	Отрицание	Смена знака.
<code>\$a+\$b</code>	Сложение	Сумма двух чисел.
<code>\$a-\$b</code>	Вычитание	Разность двух чисел.
<code>\$a*\$b</code>	Умножение	Произведение двух чисел.
<code>\$a/\$b</code>	Деление	Остаток двух чисел.
<code>\$a**\$b</code>	Возведение в степень	Возведение числа <code>\$a</code> в степень числа <code>\$b</code> .
<code>\$a++</code>	Инкрементация	Увеличение числа на единицу.
<code>\$a--</code>	Декрементация	Уменьшение числа на единицу.

- Табл. 3 - Операторы для работы с числами в PHP

На первый взгляд некоторым новичкам может казаться что такие операторы не будут нужны для создания какого-нибудь веб-приложения. Но по-настоящему они очень нужны, к примеру оператор инкрементации используется для того чтобы вывести определенное количество пользователей.

Не надо сразу думать что если в PHP есть математические операторы то надо сразу начинать учить высшую математику или поступать в какой то математической университет. Нет, не надо так думать, чтобы программировать как я уже говорил знать математику необязательно. Внизу показан небольшой пример использования данных операторов:

```
<?php
echo 5 + 5; // * 10
```

```
echo 5 - 2; // * 3
echo 5 * 2; // * 10
echo 36 / 6; // * 6
echo 5 ** 2; // * 25
$a = 5;
echo ++$a; // * 6
echo --$a; // * 4
```

?>

Использование математических операторов очень нужная вещь при создании очень динамического приложения, к примеру полноценной соц. сети и т.п. Рекомендую читателю по практиковаться с данными операторами и написать какую-нибудь простую программу.

2.2 Числа с плавающей точкой

Помните школьной курс математики? Надеюсь вы вспомнили об таком виде чисел как числа с плавающей точкой. Это почти тоже самое что и простые числа только после того как мы указали нужное нам число мы ставим точку и продолжаем уже вводить не целое число а десятые, тысячные числа и т.д. Пример в PHP:

```
<?php
$a = 56.985;
$b = 6.02;
$c = 3.4;
echo $b + $c;
```

?>

Использовать числа с плавающей точкой надо будет тогда к примеру если вам надо будет показать курс какой-нибудь крипто валюты или вывести какое-нибудь научное число.

Заключение

В заключение можно сказать что данная глава вышла довольно таки короткой, но ясной и простой. Только не надо думать что мы уже не встретимся с числами в данной книге. В главе о функциях мы будем рассматривать функции которые можно использовать для того чтобы работать с числами. Сейчас я как и всегда попрошу читателя попрактиковаться с данным типом данных и ответить на вопросы которые показаны ниже.

Вопросы и упражнения для самоконтроля

1. Как создаются целые числа в PHP?
2. Какие есть операторы для работы с числами?
3. Напишите программу которая использовала бы числа и операторы для чисел.
4. Как можно представить числа с плавающей точкой в PHP?

3 Логические операции

Без логических операций веб-приложения не смогли бы просто существовать и сам язык программирования PHP также. Логические операции реализованы почти во всех языках программирования. Но что такое логическая операция? Думаю вы уже догадываетесь, но перед тем как начать разговор о понятии логической операции надо разобраться и понять что такое псевдокод.

3.1 Псевдокод

Псевдокод- это так званный код который используется для того чтобы описать работу **алгоритма** с помощью нашего человеческого языка без использования каких то технических терминов и языка программирования. Пример простого алгоритма описанного на псевдокоде:

```
Открыть коробку с пиццей;  
Взять специальный нож для пиццы;  
Разрезать пиццу на шесть частей;  
Положить нож на место;  
Съесть пиццу;
```

Вообще использовать псевдокод хорошо когда к примеру надо описать работу какого-нибудь сложного алгоритма вместо того чтобы сразу писать алгоритм на языке программирования. Вообще псевдокод хорошо использовать для проектирования программы. Но давайте немного усложним использовав логическую операцию “ЕСЛИ”:

```
Открыть коробку с пиццей;  
Если есть пицца тогда:  
    Взять специальной нож для пиццы;  
    Разрезать пиццу на 6 частей;  
    Взять одну часть;  
    Съесть пиццу;
```

Конец;

Ну вот мы и усложнили наш пример использовав логическую операцию “ЕСЛИ”. Пример стал немного интереснее и сложнее. Это будет первая логическая с которой мы познакомимся. Если вы не очень поняли как именно работает данная логическая операция то вот ещё один пример:

Если голоден тогда:

Пойти на кухню;

Поесть;

...

Конец;

Логическая операция “ЕСЛИ” будет работать только тогда когда условие данной логической операции будет равно истине, то есть если посмотреть в нашем предыдущем примере то человек который будет выполнять данный алгоритм, то он пойдет на кухню только тогда когда он будет голоден. Если условие логической операции будет равно ложь а не истине то оно никогда не выполниться.

3.2 Логическая операция if

Вы уже знаете как работает данная логическая операция(или ещё называют логической конструкцией) в псевдокоде, но теперь настало время попрактиковаться с этим в PHP. Сначала мы указываем **имя** логической операции, в нашем случае это будет логическая операция `if`, после этого указываем **условие** в круглых скобках и после чего мы открываем острые скобки и начинаем писать код который выполниться если условие логической операции будет равно истине. Вот как это выглядит на практике:

```
<?php
    if (3 > 1) {
        echo "Условие равно истине";
        // А все потому что 3 больше чем 1.
```

```
        // И в конечном счёте это истина,  
        // и именно это и будет истиной  
условия.  
    }  
    ?>
```

В нашем условии мы указали что если 3 больше чем 1 то тогда вывести на экран заданный нами текст в команде `echo`. Чтобы сделать это условие мы использовали оператор `>` больше чем, есть также оператор `<` меньше чем, с операторами которые можно использовать для работы с логическими операциями мы познакомимся в следующей под главе.

А что делать если нам надо написать код который выполнялся если бы логическая операция равнялась ложь? В таком случае можно использовать противоположность `if` то есть **else** что значит в переводе на русский - иначе. Пример работы данной логической операции на псевдокоде:

```
Если голоден тогда:  
    Пойти на кухню;  
    ...  
Иначе:  
    Продолжать учиться;  
Конец;
```

Использовать `else` можно к примеру в тех случаях если пользователь ввёл неправильные данные. Вот как это работает уже в самом языке программирования:

```
<?php  
if (3 > 5) {  
    echo "Эта часть кода никогда не  
выполниться поскольку условие равно ложь";  
} else {  
    echo "А эта часть кода выполниться";  
}  
?>
```

Рекомендую протестировать данный код чтобы вам было легче понять работу этой логической операции. Можете теперь дать волю своему творчеству и по пробовать написать какой-нибудь простой код с использованием данной логической операции.

Но давайте представим ситуацию что у нас кроме того что нужно написать код который выполнялся если условие равно ложь, то нужно также написать код который выполнялся если бы условие было бы не равно истине и ложь. То есть как бы сказать нейтральное условие.

Для этого в PHP была создана отдельная часть логической операции `if` которая называется `else if`. `else if` можно ставить столько раз в коде - сколько вам захочется, но важно чтобы были разные условия в каждого `else if`. `else if` можно описать как исключающее или. Но **очень важной вещью** которую я хочу сказать является то что данные конструкции не могут существовать без самой логической операции `if`.

То есть **вы не можете просто взять и поставить `else` если нету логической операции `if`**. А теперь давайте вернемся к нашему “исключающему или”. Алгоритм на псевдокоде который использует данную конструкцию:

Если голоден тогда:

...

Иначе если хочу сладкого:

Пойти на кухню;

Открыть холодильник;

Достать мороженное;

Иначе:

Продолжать учиться;

Конец;

Ну а теперь можем попробовать это на практике:

```
<?php
```

```
if (3 > 3) {
```

```
...
```

```
}  
else if (3 == 3) {  
    ...  
}  
else {  
    ...  
}  
?>
```

Думаю вы заметили логический оператор `==` который означает равно. В следующей подразделе мы как раз рассмотрим какие есть логические операторы и их работу.

3.3 Логические операторы

Поскольку вы уже поняли что такое логические операции и понимаете работу одного логического оператора, то можно двигаться дальше и начинать изучать логические операторы с помощью которых и можно строить условия для логических операций.

Оператор	Значение	Пример
<code>></code>	Больше чем	<code>\$a > \$b</code>
<code><</code>	Меньше чем	<code>\$a < \$b</code>
<code>==</code>	Равно	<code>\$a == \$b</code>
<code>!=</code>	Не равно	<code>\$a != \$b</code>
<code>===</code>	Тождественно равно	<code>\$a === \$b</code>
<code>!==</code>	Тождественно не равно	<code>\$a !== \$b</code>

>=	Больше чем или равно	\$a >= \$b
<=	Меньше чем или равно	\$a <= \$b
&&	Логическое "И"	\$a && \$b
And	Логическое "И"	\$a and \$b
Or	Логическое "ИЛИ"	\$a or \$b
	Логическое "ИЛИ"	\$a \$b

- Табл. 4 - Логические операторы в PHP

Ну а теперь мы пройдемся по каждому оператору разве что мы пропустим некоторые операторы которые повторяются. Если что то я не буду очень и очень рассматривать каждый, я буду говорить сразу же суть.

Логический оператор >

Вы уже знакомы с данным оператором ещё со школьного курса математики. Условие логической операции которая использует данный оператор будет равно истине только в том случае если первый аргумент будет численно равен больше чем второй аргумент:

```
<?php
$a = 6;
$b = 3;
if ($a > $b) {
    echo $a."Больше чем".$b;
}
?>
```

Логический оператор <

Условие будет равно истине только в том случае если первый аргумент численно меньше чем второй аргумент, пример использования:

```
<?php
$a = 1;
$b = 5;
if ($a > $b) {
    echo $a."Меньше чем ".$b;
}
?>
```

Логический оператор ==

Условие логической операции будет равно истине в том случае если первый аргумент будет равен второму аргументу:

```
<?php
$a = "Hello";
if ($a == "Hello") {
    echo "Переменная равна: " . $a;
} // Можно также записать $a == $a
// можно также использовать и другие
// типы данных к данному оператору
?>
```

Логический оператор !=

Условие логической операции которая использует данный оператор, будет равно истине если первый аргумент не будет равен второму аргументу:

```
<?php
$a = "Hello";
if ($a != "world") {
    echo "Переменная не равна world";
}
```



```
}  
?>
```

Логический оператор ===

Условие которое использует данный оператор будет равно истине если первый аргумент равен второму аргументу, но при этом у них одинаковые типы данных:

```
<?php  
if ("Hello" === "Hello") {  
    echo "Идентично";  
}  
?>
```

Логический оператор !==

Условие будет равно истине если первый аргумент не равен второму аргументу, но при этом два аргумента имеют одинаковый тип данных:

```
<?php  
$a = "Hello";  
$b = "world";  
if ($a !== $b) {  
    echo "$a не равно $b но имеют одинаковый  
тип данных";  
}  
?>
```

Логический оператор >=

Условие будет равно истине если первый аргумент будет численно больше чем/равен второму аргументу:

```
<?php
$a = 3;
$b = 3;
if ($a >= $b) {
    echo "Больше чем или равно";
    // Главное не плутать этот оператор с этим =>
    // оператором.
}
?>
```

Логический оператор <=

Условие будет равно истине если первый аргумент будет численно меньше чем/равен второму аргументу:

```
<?php
$a = 1;
$b = 5;
if ($a <= $b) {
    echo "Меньше чем или равно";
}
?>
```

Логический оператор &&

Это логическое “И”, его можно использовать к примеру если надо взять два аргумента и сравнить их с одним каким то аргументом. Или к примеру, можно взять два аргумента и сравнить их с двумя другими аргументами:

```
<?php
```

```
...
if ($name && $surname == $login) {
    echo "Ошибка!";
    ...
}
?>
```

Также можно использовать аналог and:

```
<?php
...
if ($password and $login != $secret) {
    echo "Ошибка!";
    ...
}
?>
```

Логический оператор or

Использование данного логического оператора надо тогда когда нам надо сравнить или первый аргумент или второй аргумент в логической конструкции. К примеру можно взять две переменные и задать условие если одна из переменных пустая то тогда... и так далее:

```
<?php
...
if (empty($login) or empty($name)) {
    echo "Ошибка!Переменные пустые!";
}
?>
```

В нашем примере мы использовали специальную функцию `empty()` которая предназначена для работы со строками и проверять пуста ли строка или нет. Об функциях для работы со строками мы поговорим в отдельной главе, а пока что вам хватит понимая этого. Можно также использовать аналог `||`:

```
<?php
```

```
...
if (empty($login) || empty($name)) {
    echo "Ошибка!Переменные пустые!";
}
?>
```

Заключение об логических операторах

Без этих логических операторов в написании веб-приложения не обойтись. С помощью них как раз можно создавать все важные детали в приложении. К примеру без логических операторов не можно обойтись при создании безопасной регистрации и т.п. Об написании своего приложения с регистрацией и аутентификацией мы поговорим в будущих разделах.

3.4 Логическая операция switch

Если у нас будет `if` в котором надо будет задать много вариантов условий, то в таком случае чтобы сократить написание кода надо использовать логическую операцию `switch` которая работает также как и `if` только немного по-другому создаются условия:

```
<?php
...
switch($a) {
    case 1:
        echo "А равно 1";
        break;
    case 2:
        echo "А равно 2";
        break;
}
?>
```

Только что вы увидели сравнение с числовым типом данных. Можно также сделать с текстовым типом данных:

```
<?php
switch ($super) {
    case "Cool":
        echo "Super равно cool";
        break;
    case "Nice":
        echo "Super равно nice";
        break;
    case "Bad":
        echo "Super равно bad";
        break;
}
```

??

Логическая операция `switch` выполняет команду за командой и очень важно понять работу этой логической операции если не хотите ошибок. То есть, `switch` просто ищет `case` в котором значение такое же самое с переменной которую мы хотим сравнить. Функция **`break`** поставленная в конце конструкции `case` останавливает выполнение данной конструкции.

В `switch` есть аналог `else`, это `default` который выполниться если значение переменной не равно ни одному из вариантов `case`:

```
<?php
...
switch ($a) {
    case 1:
        echo "A равно 1";
        break;
    case 2:
        echo "A равно 2";
        break;
```

```
        default:
            echo "А не равно 1 и 2";
    }
    ?>
```

Можно также реализовать к примеру такую конструкцию если к примеру переменная равнялась значению одной из конструкций и выводило одно и тоже сообщение:

```
<?php
switch ($a) {
    case 1:
    case 2:
    case 3:
        echo "1,2,3 один ответ";
        break;
    default:
        echo "Значение не равняется
значениям которые есть в case";
}
?>
```

Только что мы реализовали логическую операцию switch. Если описать на языке логики то это выглядело приблизительно где то так:

Если значение А равно 1 или 2 или же 3 то
тогда:

Вывести на экран надпись;

...

Но думаю возможно у кого-нибудь возник вопрос, как использовать логические операторы в данной логической конструкции. Такой вопрос возникает в некоторых новичков. Использование логических операторов в конструкции switch такое же самое как и в if, к примеру вот использование:

```
<?php
...
```

```
switch ($a) {  
    case ($a >= 10):  
        echo "А больше чем или равно 10";  
        break;  
    case ($a <= 10):  
        echo "А меньше чем или равно 10";  
        break;  
    ...  
}  
?>
```

Что ж, этого вам хватит знать об логической конструкции `switch`. Можем двигаться дальше и можно уже заканчивать главу. И закончим мы нашу главу небольшой интересной информацией об великом математике Джордже Буле - основоположника алгебры логики или как ещё её называют алгеброй Буля.

3.5 Логический тип данных

Этот тип данных также называют `boolean` в честь английского математика Джорджа Буля, о котором мы поговорим немного позже. Логический тип данных может принимать только 2 значения это истинна и ложь. Пример создания двух переменных с логическими типами данных:

```
<?php  
$super = true; // Значение этой переменной  
равно истине  
$unhappy = false; // А значение этой переменной  
равно ложь  
?>
```

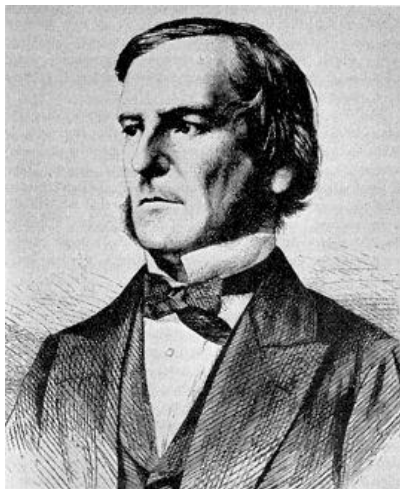
Мы уже говорили об том как к примеру условие операции может получать значение истинны и ложь. Некоторые функции в PHP могут отображать данные с помощью логического типа данных. К примеру

является такая-то переменная числом или нет, и если да то тогда мы получим ответ значением равным истине. Или наоборот, если какая-то переменная не является числом то мы получим ответ равный ложь.

3.6 Джордж Буль

Немного краткой биографии перед тем как я начну рассказывать о том что сделал этот большой математик. Джордж Буль - британский математик, логик и философ. Родился 2 ноября 1815 года в городе Линкольн в Англии и умер 8 декабря 1864 года. Один из основателей логики математики.

Джордж родился в не очень богатой семье ремесленника Джона Буля который увлекался математикой и логикой. Как раз отец Джорджа и дал ему первые уроки математики и заинтересовал сына в науке.

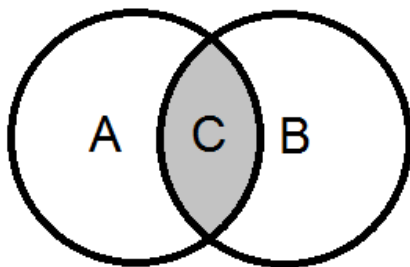


- Рис. 3 - Портрет Джорджа Буля

Если не было бы алгебры Буля то мы не смогли бы построить полноценные компьютеры и микроконтроллеры. Поскольку микропроцессоры строятся на логических вентилях которые исполняют логические операции которые описал Джордж Буль. Обо всем этом вы

можете почитать в книгах на тематику архитектуры компьютеров или как ещё это называют Архитектурой ЭВМ(Электрическая Вычислительная Машина).

Давайте я приведу несколько правил которые создал Джордж Буль в алгебре логики. К примеру мы можем описать логический оператор И с помощью двоих кружков Ейлера как это показано на рисунку под номером 4.



- Рис. 4 - Логическое И показанное на схеме кружков Ейлера

Тут есть только три переменные - это A, B и C. Теперь я объясню как все работает. Во первых давайте введем формулу для нашего логического “И”, которое в алгебре Буля обозначается +:

$$A + B = C$$

Наши переменные могут принимать только два значения это 1(истина) или 0(ложь). И в нашей формуле если две переменные будут иметь значения истины то переменная C будет равно истине. Лучше описать это очередной формулой:

$$1 + 0 = 0;$$

$$1 + 1 = 1;$$

$$0 + 1 = 0;$$

Надеюсь что вы поняли что даже если хотя бы один аргумент равен 1, а второй нет, то все равно на выходе будет ноль(значение равное ложь). Но если мы хотим чтобы была истина на выходе и при этом один из аргументов был равен истине, а второй был равен ложь то следует

использовать логическое “ИЛИ” которое обозначается умножением в формулах:

$$1 * 0 = 1;$$

$$1 * 1 = 1;$$

$$0 * 1 = 1;$$

Надеюсь вы заметили что конструкции которые существуют в алгебре Буля есть и в программировании но также оно используется и в компьютерах. Если вам интересна эта тематика, то вы можете найти много интересных книг на эту тему.

Заключение

В общем можно сказать что без логических операций вы не сможете построить своего веб-приложения. Они играют очень важную роль в программировании и нашей жизни. Поскольку когда мы делаем какой то выбор, то это уже и есть логическая операция. Без них не обойтись. Как и всегда я попрошу читателя попрактиковаться с материалом представленным в этой главе и ответить на вопросы и выполнить упражнения которые показаны ниже.

Вопросы и упражнения для самоконтроля

1. Что такое логическая операция?
2. Какие есть логические операции в PHP?
3. Что такое `if` и как он работает?
4. Можно ли поставить конструкцию `else` без логической операции?
5. Назовите логические операторы в PHP.
6. Напишите код который использовал логическую операцию `switch` и делал сравнение значений переменной `Z` с числами 10,30,20. Значение переменной давайте сами.

4 Циклы

Слово цикл походит от греческого слова *kuklos* - что значит круг. Думаю вы когда-нибудь встречались с этим словом. Цикл значит повторение каких то действий, пока не будет достигнута определенная цель. Циклы как и логические операции имеют свое условие и код который будет повторяться тогда когда условие цикла будет равно истине. Первый цикл с которым вы познакомитесь это будет цикл `while`.

4.1 Цикл `while`

Цикл будет работать только тогда когда условие цикла будет равно истине. Описание работы цикла `while` на псевдокоде:

```
Пока (голоден) :  
    Есть;  
    ...  
Конец;
```

В этом примере видим что если я буду голоден, я буду есть до тех пор пока я не буду сыт, это логически выходит из того что если я буду сыт то я не буду голоден и не будет потребности использовать данный цикл. А теперь сделаем похожую вещь только уже на практике:

```
<?php  
while (2 + 2 == 4) {  
    echo "Эта надпись будет выводиться все  
время на экран!";  
}  
?>
```

После того как вы запустите этот код то с очень большой скоростью на страницу будет выводиться заданная нами надпись в теле цикла. Я думаю вы заметили что в условии я записал `2 + 2 == 4`, в этом

есть небольшая программистская шутка о том пока мир не сойдет с ума то программа будет работать дальше. Условие равняется истине потому что $2 + 2$ будет равно 4.

Условие этого цикла можно сделать истинным просто поставив логический тип данных с истинным значением:

```
<?php
while (true) {
    echo "Это почти бесконечный цикл";
}
?>
```

Вы можете использовать в цикле логические операторы. Если вы хотите чтобы цикл работал, то вы просто должны сделать так чтобы условие цикла было равно истине.

4.2 Цикл do...while

После того как вы познакомились с одним циклом, то пора переходить к следующему циклу который работает почти также само как и while только истинность условия проверяется в конце самого цикла. Этот цикл хорошо использовать нам к примеру надо вывести определенное число записей к примеру:

```
<?php
$i = 0;
do {
    $i++;
} while ($i > 10);
?>
```

Запустите этот код и у вас будет выводиться 10 цифр: от 0 до 9. Думаю кто задастся вопросом почему именно от 0 к девяти а не от 1-го к 10-ти, а все потому что в программировании и вообще в компьютерах **счёт вводится от нуля**, то есть ноль это как бы сказать уже единица.

Также я думаю вы заметили, что мы в примере использовали математический оператор `++` чтобы увеличить нашу переменную на единицу. Надеюсь вы уже поняли работу этого цикла, то есть сначала действие а потом проверка истинности.

4.3 Цикл `for`

Цикл `for` является самым сложным в PHP. В цикл надо указать целых три выражения чтобы он мог работать. Как раз его используют для того чтобы вывести на экран определенное число чего-то. Синтаксис:

```
for (выраж1; выраж2; выраж3) {  
  
}
```

Первое выражение выполняется только один раз в начале условия, оно задает значение переменной которая будет использована в условии. А потом будет оцениваться второе выражение, если второе выражение будет равно истине то условие продолжит выполнять работу и будет выполнять вложенные в него операторы. В противном случае цикл остановит работу.

И в конце будет выполняться третье выражение. Если второе выражение будет отсутствовать, то цикл будет выполняться бесконечно. Выражения могут быть пустыми или содержать несколько выражений которые разделены точкой с запятой. Пример:

```
<?php  
for ($i = 0; $i <= 10; i++) {  
    echo $i;  
}  
?>
```

Код который мы только что написали будет выдавать тот же самый результат как и в примере цикла `do...while`. То есть выведется 10 цифр. Теперь давайте подробнее разберем наш пример, сначала мы задали значение переменной `i`, а потом использовали логический

оператор `<=` чтобы проверить равно ли наша переменная меньше чем 10 или равна 10.

Если второе выражение как я уже и говорил равно истине то цикл продолжит работу, и на конец мы используем оператор `++` чтобы увеличивать нашу переменную до 10. Цикл можно также записать:

```
<?php
for ($i = 0; ; i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}
?>
```

В этом примере мы уже пропустили второе выражение и тем самым мы указали что этот цикл будет работать бесконечно. Но поскольку мы поставили логическую операцию `if` которая указывает что если наша переменная будет равна больше чем 10 то остановить работу с помощью функции `break`. Есть также много других примеров как вы можете записать условие этого цикла, просто зайдите на официальную документацию разработчиков PHP.

Заключение

Раздел вышел довольно таки небольшим, но я хотел также сказать об `foreach` о котором вам ещё рано знать поскольку вы ещё не изучили такой тип данных как массивы. Этот цикл служит для того чтобы перебирать элемента массива о которых вы узнаете немного позже. А сейчас же дайте ответы на вопросы для самоконтроля и сделайте упражнения.

Вопросы и упражнения для самоконтроля

1. Что такое цикл?
2. Какие есть циклы в PHP?
3. Цикл `while` будет работать если его условие будет равно ложу.
4. Как работает цикл `do...while`?
5. Напишите программу с циклом `while` для проверки значения переменной `$a`.

5 Массивы

Когда мы слышим слово массив то сразу представляем себе что-нибудь большое. Или кто то даже подумает, что массив значит масса. Нет, это все очень разные вещи. Массив - это набор элементов из разных типов данных. Если вы не очень поняли что это такое то можете себе представить куча шкафчиков, которые объединены вместе. То есть вы помните пример переменной на шкафчике, то себе теперь представьте что один массив это несколько таких шкафчиков.

А внутри шкафчиков вы положили листки бумаги, на которых написали какие то данные и тем самым установив им тип данных. Надеюсь что вы уже поняли что это такое и можем продолжать.

5.1 Создание Массивов

Вот как выглядел бы простенький массив на псевдокоде:

```
Коробка = "Ключ", "Яблоко", "Фото";
```

Как я уже говорил массивы могут иметь **любые типы данных** и могут быть иметь несколько типов данных подряд, и также в массивы можно вместить переменные. Вот как это делается на практике:

```
<?php
$arr = [85994, "Prodigy", "Experience", 556];
$string_arr = ["Hl.exe", "user", "vinyl"];
$arr2 = [$box, $pizza1, $tape];
// Массив можно записать также вот так:
$arr3 = [
    "Element1",
    "Element2",
    "Element3"
];
?>
```

Вот так выглядит массивы на практике, а теперь пришло время обратиться к элементам массива. Поскольку счёт в программировании ведется от нуля то и обращаться к элементам массива мы будем от нуля:

```
<?php
    $arr[0]; // Мы обращаемся к первому
элементу массива $arr
    echo $arr[1]; // А это уже второй элемент
?>
```

Способ который мы только что просмотрели называется способом обращения к элементам массива через индекс, но также можно обратиться и через само название элемента, к примеру:

```
<?php
    $arr["Prodigy"];
?>
```

Есть также ещё один способ создания массивов:

```
<?php
    $arr[] = "Their";
    $arr[] = "Law";
```



```
$arr[] = "The singles";  
?>
```

Теперь когда вы знаете как создаются массивы и как ним можно обращаться то можно изучить операторы которые используются для работы с массивами.

5.2 Операторы для массивов

В табл. 6 показаны операторы для работы с массивами.

Оператор	Действие	Пример
+	Объединение	<code>\$a + \$b</code>
==	Равно	<code>\$a == \$b</code>
===	Идентично	<code>\$a === \$b</code>
!=	Не равно	<code>\$a != \$b</code>
<>	Не равно	<code>\$a <> \$b</code>
!==	Не идентично	<code>\$a !== \$b</code>

- Табл. 5 - Операторы для массивов в PHP

Многие здесь вам операторы знакомы, но думаю что вам не совсем понятно как они используются в массивах. Поэтому начнём разбор операторов.

Оператор для массивов +

Используя этот оператор вы объедините два массива в один целый. Пример:

```
<?php  
...
```

```
$c = $a + $b;  
?>
```

Оператор массивов ==

Данный оператор будет проверять имеют ли два массива одинаковые элементы или хотя бы несколько из них. Будет выдавать значение `true` если будет равно истине и `false` если равно ложь:

```
<?php  
...  
echo $a == $b;  
?>
```

Оператор для массивов ===

Как и оператор `==` будет сравнивать содержимое массива, в этом случае наш оператор будет проверять не только содержимое массива на наличие одинаковых элементов но также будет ли иметь массивы одинаковый тип данных. Пример:

```
<?php  
$a = ["Name", "Login", "Password"];  
$b = ["Login", 456, 59.04];  
  
echo $a === $b; // Будет равно false  
поскольку массивы имеют разные элементы  
// а также разные типы данных  
?>
```

Заключение об операторах для массивов

Я не упомянул такие операторы как `!=`, `!==` и `<>`, поскольку их работа вам уже понятна и очень проста. А теперь можем двигаться дальше, и сейчас мы поговорим об многомерных массивах.

5.3 Многомерные массивы

Само название этих массивов уже вам говорит что эти массивы будут большими. Это как бы сказать массив в массиве. Или можно снова таки воспользоваться примером со шкафчиками, то есть шкафчик в шкафчике. Вот к примеру небольшой пример:

```
<?php
    $cont = [ ["Verilog", "Masm", "Wasm"] ];
?>
```

Обращение к таким массивам также выглядит по-другому. Вы должны сначала указать какой к **какому массиву** хотите обратиться а потом выбрать элемент этого массива:

```
<?php
    echo $cont[0][1]; // Будет выведено Masm
?>
```

Только что мы создали двухмерный массив но также есть и трёхмерный массив:

```
<?php
    $super3 = [ [ ["1", "2", "3"] ], [ [1, 2, 3] ] ];
    echo $super3[0][1][2]; // Мы выбираем
первый массив и второй массив который внутри первого
массива
// И в конце мы выбираем третий элемент
массива
?>
```

5.4 Массив ключ-значение

Данный вид массива реализован во многих других языках программирования, к примеру в языке программирования Ruby такой массив называется хэшем. Элементы такого массива это **ключи** которые имеют свое **значение**. То есть ключ это как бы сказать идентификатор.

Создание таких массивов выглядит вот так:

```
<?php
$developer = [
    "Jim" => "Admin",
    "Mike" => "Designer",
    "John" => "Creator"
];
// Или можно записать вот так:
$team = ["Alex" => "QA", "George" =>
"Manager"]];
?>
```

=> - этот оператор называется **хэш-рокет**, он используется для присвоения значения ключу. Ключ-значение массив хорошо использовать если нам к примеру надо сделать систему пользователей. К примеру ключ может служить название поля а значение ключа быть переменной в которую пользователь и вводит данные:

```
<?php
$user = [
    "login" => $login,
    "name" => $name,
    "info" => $info,
    "password" => $password
];
?>
```

Но можно также и реализовать систему записей и вообще можно реализовать все, что вы захотите, все зависит только от вашего желания. Об том как создать приложение с регистрацией и входом в свой аккаунт мы поговорим позже в отдельной главе.

Обращение к элементам таких массивов осуществляется с помощью ключа:

```
<?php
$developer["Jim"]; // Обращение к ключу Jim и
его содержимого
```

```
$team[0]; // Можно и в такой способ  
обращаться  
>
```

А также можно добавлять элементы в массив в такой способ:

```
<?php  
$arr[0] = "User";  
>
```

После этого в массиве добавиться элемент с ключом 0 и значением “User”. На этом я могу завершить раздел, в следующем разделе мы поговорим об очень важном элементе который нам облегчает в несколько раз работу вместо того чтобы “изобретать велосипед”.

Ну и как всегда я попрошу читателя попрактиковаться с материалом который был показан в этом разделе, а также ответить на вопросы для самоконтроля и выполнить упражнения внизу.

Вопросы и упражнения для самоконтроля

1. Что такое массив и как его можно создать в PHP?
2. Какие есть виды массивов?
3. Могут ли массивы иметь разные типы данных?(Да/Нет)
4. Напишите код в котором был бы двухмерный массив с числами.

6 Функции

Если вы будете писать какое-то большое веб-приложение и у вас будет часть кода которую вы будете много где использовать, то намного лучше будет создать функцию с этой частью кода, вместо того чтобы “изобретать велосипед”. Поскольку если вы будете все время вставлять код который вам надо будет много где использовать то это займет очень много времени, а также будет очень трудно читать код.

Вообще о том что “изобретать велосипед” не всегда не хорошо написано во многих книгах по проектированию своего приложения или проекта. К примеру в одной из самых лучших книг по проектированию “Совершенный Код” которую написал Стив Макконнел, написано что лучше использовать тот инструмент который уже был создан чем писать новый. Но если вы все таки хотите написать свой инструмент, то объясните чем он лучше за тот что уже был создан.

Давайте вернемся к нашей теме о функциях, в PHP уже есть множество **встроенных функций** которые предназначены для разной работы. К примеру есть специальные функции для работы с текстовым типом данных и т.д. Но сейчас мы поговорим об общем понятии функций в PHP.

6.1 Создание функций

Как я уже и говорил, функция позволяет нам не изобретать велосипед снова и снова. Сейчас же я приведу вам пример функции на псевдокоде:

```
Функция Приветствие() :  
    сказать "Привет!";  
Конец;
```

Объявили мы функцию с помощью аргумента Функция после чего мы указали название нашей функции и это Приветствие. В круглых скобках мы указываем локальные переменные нашей функции о которых

поговорим немного позже. Код который мы написали внутри функции будет выполняться тогда когда мы где то будем вызывать.

Вот как выглядело бы использование своей функции в псевдокоде:

Если увижу знакомого:

Приветствие();

Конец;

Попробуем теперь создать функцию в PHP:

```
<?php
function Hello() {
    echo "Hello world!";
}
?>
```

Аргумент `function` используется как раз для того чтобы объявлять функции. Как и в переменных так и у функций есть также свои правила создания названия. Название функции не должно начинаться с цифр или с русского алфавита. Не обязательно называть функцию с большой буквы можно и маленькой. После создания функции её можно использовать в нужных для вас местах:

```
<?php
...
Hello();
?>
```

Очень **важно** чтобы функция которую вы создали **находилась** в том же коде, где и вы хотите её использовать, но лучше сохранять функции в отдельных файлах. После чего подключать эти файлы между собой, но об этом поговорим позже.

А сейчас же мы поговорим об локальных переменных. Локальные переменные это переменные которые видит только функция внутри себя. К примеру мы не сможем обратиться к локальной переменной функции если она имеет локальную область видимости, если же локальная переменная будет иметь глобальную область видимости то тогда уже к

этой переменной смогут обратиться другие переменные, функции и т.д. и т.п.

Пример функции с локальной переменной:

```
<?php
...
function Hello($words) {
    echo $words;
}
?>
```

Теперь я объясню работу функции которую мы только что создали. \$word это **локальная переменная** которой мы будем задавать значение когда будем вызывать где-то нашу функцию. Нельзя использовать функцию не задав локальным переменным этой функции значения. Вот как будет выглядеть использование нашей функции:

```
<?php
...
Hello("Hello World!");
?>
```

Мы видим что мы задали значение локальной переменной "Hello World!". Также локальные переменные могут иметь заданные значения по умолчанию:

```
<?php
function Default($a = 3) {
    ...
}
?>
```

Давайте усложним немного наш пример и напомним функцию которая делала сложение двух переменных:

```
<?php
function sum($a,$b) {
    return echo $a + $b;
}
```


?>

После того как мы введем локальные переменные в коде где мы будем использовать эту функцию то функция возвратит нам результат с помощью аргумента `return`. Аргумент `return` просто возвращает управление конструкции где была вызвана функция. Но вообще если аргумент `return` в функции, то аргумент прекратить работу функции и если надо может возвратить значение функции, к примеру может возвратить значение каких то переменных.

Также в функции можно вставлять логические операции и циклы, но существовать они будут только тогда когда функция будет вызвана. А также могут быть функции в функциях, но эти функции которые были созданы в функции будут существовать только тогда когда функция в которую была вложена функция будет где-то вызвана. Такие функции называются вложенными.

Ну а теперь как я и обещал в предыдущих главах я покажу функции которые предназначены для работы с разными типами данных. Я не буду показывать все функции которые созданы для того чтобы работать с типами данных, поскольку их очень и очень много.

6.2 Функции для работы со строками

Покажу только некоторые функции которые предназначены для работы с этим типом данных. Посмотрите на таблицу 8:

Название функции	Действие
<code>echo</code>	Выводит на экран задданую строку с <code>enter</code> 'ом.
<code>print</code>	Работает также как и <code>echo</code> только без <code>enter</code> 'а
<code>rtrim</code>	Удаляет пробелы с конца и начала строки.

substr	Возвращает подстроку заданной строки.
ord	Возвращает ASCII-код символа
implode	Объединяет элементы массива в строку.
explode	Разбивает строку на подстроки с помощью разделителя.
htmlspecialchars	Преобразовывает HTML-объекты в специальные коды.

- *Табл. 6 - Некоторые функции для работы со строками в PHP*

Ну а теперь мы быстро пройдемся по этим функциям и объясню их работу. Мы разве что не будем разглядывать работу `print` и `echo` поскольку уже ясно что они делают.

Функция `rtrim`

Как было описано то эта функция удаляет пробелы в начале и конце строки. Пример:

```
<?php
$name = " Mike ";
echo rtrim($name); // Выводит:Mike
?>
```

Есть также функция `trim` которая работает по такому же самому принципу.

Функция `substr`

Данная функция имеет три аргумента которые мы должны указать: `substr(string, start, end)`. Первый аргумент это строка с которой мы хотим достать подстроки, второй это индекс от

которого мы будем доставать подстроку, и третий аргумент это индекс на котором мы хотим закончить извлечение подстроки. Пример:

```
<?php
echo substr("apple", 0, 3); // Выведет: appl
echo substr("video", 1, 4); // Выведет: ideo
?>
```

Функция ord

Как и было описано данная функция возвращает ASCII коды символов строки. Пример:

```
<?php
echo ord("a"); //Результат: 97
echo ord("on"); //Результат: 111
?>
```

Функция implode

Собирает все элементы массива в одну строку. Пример работы:

```
<?php
$case = ["Files", "Papers", "Pen"];
echo implode($case); //Выведет: Files PapersPen
?>
```

Функция explode

Разбивает строку на элементы массива при помощи разделителя:

```
<?php
$mail = "Journal Book Paper";
$messanges = explode(" ", $mail);

echo $mail[0]; //Journal
echo $mail[1]; //Book
echo $mail[2]; //Paper
?>
```

Как вы заметили? то данная функция имеет два аргумента, в первом мы указываем разделитель, а во втором уже строку которую хотим разбить на массив. В нашем случае разделитель это пробел, а строка это переменная `$mail`. Можно также сделать и наоборот, превратить элементы массива в строку.

Функция `htmlentities`

Преобразовывает все HTML-объекты в соответствующие коды:

```
<?php
$objects = "<div>ННН</div>";
echo htmlentities($objects);
//Результат: &lt;div&gt;ННН&lt;/div&gt;
?>
```

Это, конечно же не все функции для работы со строками, по настоящему их очень и очень много, если вам интересно то можете зайти на официальный сайт PHP: `php.net` там есть официальная документация по функциям для работы со строками и не только. Документация языка программирования это первое место куда должен обратиться программист если ему что то неясно. В документации PHP вы сможете найти очень и много разных статей, справочников по функциям и т.д. и т.п. Поэтому пользуйтесь документацией поскольку это может очень и очень сильно помочь разработчику.

6.3 Функции для работы с числами

После того как мы рассмотрели какие есть функции для работы со строками и вы поняли как вообще выглядят встроенные функции, то можем рассмотреть некоторые функции которые предназначены для работы с числами:

Функция	Действие
Sqrt	Находит квадратный корень.
Rand	Генерирует случайное число.
Pow	Возводит в степень.
Pi	Число Пи.
Max	Возвращает наибольшее значение.
Min	Возвращает наименьшее значение.

- Табл. 7 - Некоторые функции для работы с числами в PHP

Я хотел также вставить в этот список тригонометрические функции, но все таки решил что лучше не вставлять поскольку читатель который не очень хорошо понимает математику может не очень хорошо понять что вообще происходит при использовании данных функций.

Функция sqrt

Функция просто находит квадратный корень заданного числа:

```
<?php
echo sqrt(9); //3
echo sqrt(25); //5
?>
```

Функция rand

Эта функция есть во многих языках программирования. Мы задаем от которого числа начинать считать и на котором остановить отсчёт:

```
<?php
echo rand(0,9); //Будет считать от 0 до 9 и
потом выведет случайное число
```

```
?>
```

Функция pow

Функция имеет два аргумента первый это степень в которую мы хотим возвести число, и второй аргумент это число который мы хотим возвести в заданную нами степень:

```
<?php
echo pow(2, 5); //25
echo pow(3, 5); //125
?>
```

Функция pi

Функция не делает ничего супер уникально просто выводит число пи которое очень большое:

```
<?php
echo pi(); // 3.14... и т.д.
?>
```

Функция max

В функцию мы вводим числа или можем вводить также строки, и в результате функция возвратит нам какой элемент имеет большее числовое значение чем другие:

```
<?php
echo max(1, 2, 3, 4, 6, 9); // 9
?>
```

Функция min

Работает также как и max только наоборот:

```
<?php
echo min(3, 6, 1, 5, 9); // 1
?>
```

Это не все функции которые предназначены для работы с числами, если хотите увидеть больше то тогда обращайтесь в официальную документацию по числовым функциям. Эти функции я показал как пример, для того чтобы вы могли потом с ними поработать чтобы лучше усвоить материал.

6.4 Функции для работы с массивами

Ну и напоследок об встроенных функциях для работы с типами данных я расскажу вам об функциях для работы с массивами. Посмотрите на таблицу 8:

Функция	Действие
Array	Создает массив.
in_array	Проверяет есть ли в массиве заданный элемент.
array_sum	Вычисляет сколько элементов есть в массиве.
array_unique	Удаляет повторные элементы в массиве.
array_values	Извлекает все элементы массива.
array_rand	Выбирает случайный элемент в массиве.
array_pop	Извлекает последний элемент массива.

- Табл. 8 - Некоторые функции для работы с массивами в PHP

Функция array

Функция используется для создания массива и его указания его элементов:

```
<?php
$groups = array (
    "Kraftwerk",
    "The prodigy",
    "Eisenfunk"
);
//Можно также создавать массивы с ключами:
$team = array(
    "Mike" => "Teamlead",
    "Jack" => "Developer",
    "Lorens" => "Engineer"
);
?>
```

Функция in_array

Используется чтобы проверить есть ли заданный нами элемент в заданном нами массиве. Функция имеет три параметра которые мы можем указать. Первый это элемент/значение которое мы ищем в массиве, второй это массив в котором мы будем искать заданный нами элемент и третий аргумент это режим со строгой проверкой:

```
<?php
$group = ["Front-end", "Back-end", 134];

if (in_array(134, $group)) {
    echo "Поиск без строгой проверки";
}

if (in_array("134", $group, true)) {
    echo "'134' найдено со строгой проверкой";
}
```



```
?>
```

Функция array_sum

Суммирует все элементы массива в один:

```
<?php
$values = [3, 5, 7, 8];
echo array_sum($values);
// Результат: 24
?>
```

Функция array_unique

Удаляет повторяющиеся элементы в массиве:

```
<?php
$app = ["Code", "Plan", "Team", "Code"];
$new_app = array_unique($app);
echo $new_app;
?>
```

Функция array_values

Извлекает все элементы массива:

```
<?php
$array = ["Paperoni", "Cheese", "Tomatoes"];
echo array_values($array);
?>
```

Функция array_rand

Выбирает случайный элемент массива:

```
<?php
$a = [2, 3, 5, 7, 9];
echo array_rand($a);
?>
```

Функция array_pop

Извлекает последний элемент массива:

```
<?php
$val = [1, 2, 3];
echo array_pop($val);
?>
```

На этом я завершаю разговор об встроенных функциях в PHP, хотя мы ещё с встретимся в будущих главах с многими функциями которые к примеры предназначены для работы с базой данных. А сейчас же мы поговорим об так званых анонимных функциях.

6.5 Анонимные функции

Анонимные функции - известны также как функции которое без названия. К примеру мы можем присвоить какой то переменной значение функции:

```
<?php
$my_func = function($word) {
    echo $word;
}

$my_func($word);
?>
```

Заключение

Об функциях ещё много чего можно сказать, я дал вам основные знания об этом. С помощью функций нам не приходится изобретать велосипед снова и снова. Следующий раздел будет посвящен Объектно-Ориентированному-Программированию — принцип, который намного делает легче разработку и не надо изобретать велосипеды.

Когда мы будем изучать классы, то мы познакомимся с публичными, приватными и защищенными функциями. Но об этом немного позже а сейчас же попрошу читателя ответить на вопросы для самоконтроля и выполнить упражнения которые внизу чтобы лучше закрепить материал.

Вопросы и упражнения для самоконтроля

1. Что такое функция и как её определить?
2. Как можно использовать локальные переменные?
3. Назовите хотя бы 3 функции которые предназначены для работы со строками.
4. Назовите хотя бы 3 функции для работы с числами.
5. Создайте функцию в которую мы бы задавали три числа А, В и С. Сначала функция должна умножить А на В а потом добавить С и найти квадратный корень всего этого.

7 PHP и ООП

В этом разделе мы будем изучать материал, который не всегда можно понять с первого раза, поэтому будьте внимательными. ООП - это аббревиатура, которая расшифровывается как Объектно-Ориентированное-Программирование. В ООП все строится на классах, методах и объектах. Класс - это как бы сказать шаблон, который мы создаем, чтобы потом с этого шаблона сделать объект. Классы имеют свои функции и методы которым мы можем объявлять область видимости, но об этом немного позже.

Если вы не очень поняли что значит класс то представьте что классы это формочки для выпекания печенья, а цвет печенья, тесто с которого оно сделано и т.д. - это уже функции класса. Вот так можно себе научно-популярно представить, что это такое. ООП основано на: полиморфизме, инкапсуляции и наследовании. Давайте я объясню все эти понятия.

Полиморфизм - это концепция где общий интерфейс обрабатывает разные типы данных.

Инкапсуляция - это концепция объединяющая данные и код и обрабатывая их как одно целое.

Наследование - это когда дочерний класс наследует методы от родительского класса.

Или если вы не очень хорошо поняли, то наследование это когда какой-то класс берет методы от другого класса. Класс который взял методы это дочерний класс а класс в которого взяли методы родительский класс. Этих понятий вам хватит для начала, а теперь поговорим об создании классов.

7.1 Классы

Классы объявляются с помощью аргумента `class` и после чего мы вводим название класса и наконец раскрываем фигурные скобки и начинаем писать код внутри класса. Хочу заметить что класс это не просто шаблон который мы создаем, это даже отдельный класс данных.

Пример создания:

```
<?php
class Group {
    ...
}
?>
```

После создания класса мы можем создать объект на шаблоне этого класса с помощью директива `new`:

```
<?php
$group1 = new Group();
?>
```

Помните об общих правилах создания названия, никаких цифр сначала и символов на русском алфавите. В классе мы можем указывать свои функции, они могут иметь глобальную область видимости и локальную:

```
<?php
class Group {
    public function Create() {
```

```
... // Эту функцию можно будет вызывать
за пределами класса
    }
    private function Delete() {
        ... // Эта функция не будет доступна
за пределами класса
    }
    protected function Update() {
        ... // А эта функция будет доступна
только классу
    }
}
?>
```

В эти функции можно будет указать свой код и также локальные переменные, а чтобы вызвать функцию класса надо будет указать имя класса, поставить :: и указать название функции с локальными переменными если они конечно есть:

```
<?php
Groups::Create();
?>
```

В классе могут быть свои переменные которым мы устанавливаем область видимости и также можем задать значение которое будет по умолчанию:

```
<?php
class Groups {
    public $name = "Kiss";
    public $genre = "Rock";
    public $since = "1974";
}
?>
```

К переменным которые внутри класса можно обращаться в такой способ:

```
<?php
$group1_name = Group::name;
$group = new Group();
$test = $group->name;
?>
```

Внутри функции обращение к какой-то переменной класса осуществляется через супер-переменную `$this`:

```
<?php
class Group {
    public $name = "Kiss";
    ...
    public Create($value) {
        $this->name = $value;
        return $value;
    }
}
?>
```

Ну а теперь когда вы поняли как работают переменные в функциях класса и вообще в классе то можно поговорить об константах. Константы - это почти тоже самое что и переменные только они не объявляются знаком доллара и всегда имеют постоянное значение. Константы не должны быть свойством или вызовом функции. Объявляются они с помощью директива `const`. К примеру:

```
<?php
class Album {
    const Price = 50;
    const Constant = 50 + 5 * 2;
    ...
}
```

?>

Теперь вы имеете чёткое представление об классах и функциях которые внутри их. А теперь поговорим об конструкторах и деструкторах.

7.2 Конструкторы и Деструкторы

Конструкторы это такие функции которые мы создаем внутри класса и они вызываются только один раз при создании нового объекта этого класса. Конструкторы и деструкторы могут иметь свои локальные переменные так как и простые функции. Конструктор создается создавая функцию и давая ей имя `__construct`:

```
<?php
class Album {
    public $name = "";
    function __construct($name = "default") {
        return $name;
    }
}
?>
```

Заметьте что при создании конструкции не указывается область видимости. Если вы теперь создадите новый объект то вы получаете надпись default:

```
<?php
$album1 = new Album();
?>
```

Мы выяснили что такое конструктор, но что такое деструктор? Деструктор это также функция, вызывается она тогда когда освобождаются все переменные или завершение скрипта:

```
<?php
class Group {
    ...
    function __destruct() {
```



```
        echo "Скрипт завершен";  
    }  
}  
?>
```

Конструкторы и деструкторы хорошо использовать если к примеру в начале надо сделать проверку значений или удалить не нужные данные после окончания работы скрипта. Теперь поговорим об наследовании и что это такое. Как я уже говорил наследование это когда дочерний класс принимает функции от родительского класса, реализовать это можно с помощью аргумента `extends`:

```
<?php  
include("group.php");  
class Album extends Group {  
    ...  
}  
?>
```

Как вы видите то класс альбом принял функции от класса `Group` который находится в отдельном файле в том же месте где и находится файл с классом `Album` и подключили мы этих два файла с помощью функции `include`. Теперь мы сможем использовать функции класса `Group` с помощью класса `Album`:

```
<?php  
...  
Album::Create();  
$new_obj = new Album();  
?>
```

Но важно что дочерний класс не сможет принять классы с **защищенной** (`protected`) областью видимости. Это ещё одна способность языка PHP с которой нам не придется *изобретать велосипед*.

7.3 Абстрактные классы

Перед тем как я расскажу вам об абстрактных данных, я для начала расскажу об том что такое абстракция в общем. Представьте себе компьютер, вы чётко знаете что это компьютер а не материнская плата, видеокарты, процессор, микроконтроллеры вместе с транзисторами и т.д. и т.п. То есть абстракция это понимание какой то группы объектов как единое целое.

Ну а теперь вернемся к абстрактным классам, если класс содержит хотя бы одну абстрактную функцию то мы должны определить данный класс как абстрактный. Абстрактные классы как раз и реализующих один из принципов ООП в PHP - полиморфизм. Как было сказано, полиморфизм - это концепция где есть общий интерфейс для обработки разных типов данных и т.д.

То есть обработка данных как одно целое:

```
<?php
abstract class AbstractClass {

    public function AbstFunc() {
        // Публичная функция
    }

}
?>
```

Чтобы сделать статическую функцию надо использовать аргумент `static`. Аргумент `static` создает статическую переменную. К статическим переменным можно обращаться без создания экземпляра класса, к примеру:

```
<?php
class MyClass {
    public static $some = "value";
}
```

```
        public function PubFunc() {
            return self::$some;
        }
    }

    class MyClass2 extends MyClass {
        public function ParFunc() {
            return parent::$some;
        }
    }

    MyClass::$some;
?>
```

Надеюсь вы заметили что мы в первом классе в функции указываем чтобы функция возвращала от себя статический метод. Ну а теперь посмотрим на пример с статической функцией:

```
<?php
class MyClass {
    public static function MyFunc() {
        ...
    }
}

MyClass::MyFunc();
?>
```

Теперь когда вы поняли как можно реализовать абстракцию данных в классах, можно поговорить об интерфейсах.

7.4 Интерфейсы

Интерфейсом можно описать функции которые должны быть в классе, только мы не описываем их работу. Функции которые создаются в интерфейсе должны быть публичными, а также внутри функций ничего

не должно быть. Интерфейс объявляется с помощью аргумента `interface`:

```
<?php
interface Group {
    public function create();
    public function update();
    public function delete();
}
?>
```

После создания интерфейса его можно реализовать в классе с помощью использования аргумента `implements`:

```
<?php
class Mettalica implements Group {
    function create() {
        ...
    }
    function update() {
        ...
    }
    ...
}
?>
```

То есть это также что то типо шаблонов. Ну а теперь поговорим об том что такое перегрузка и магические методы.

7.5 Перегрузка и магические методы

С помощью перегрузки можно динамическим способом создавать функции которые обрабатываются магическими методами, которые создаются в классе. Чтобы перегрузить свойства(локальные переменные) надо использовать функции `__get` и `__set`. Это сработает тогда когда

объект к которому мы обращаемся не содержит свойства к которым можно осуществить доступ:

```
<?php
public void __set(string $name, mixed $value);
public mixed __get(mixed $name);
?>
```

Вот пример использования:

```
<?php
class Group {
    public $since = "1990";
    public function __set($name, $value) {
        echo " {$name} - значение не равно
name";
    }
    public function __get($name) {
        echo "Значение не равно value";
    }
}
$obj = new Group;
echo $obj->a; // Свойства не существует и мы
получим сообщение которое было задана в функции
echo $obj->b = 3; // Свойства не существует
echo $obj->since; // Свойство существует
?>
```

Ну а теперь в общем поговорим об том как перегрузить функции, это делается с использованием метода `__call`. Это сработает тогда если в классе нету функций которым можно осуществить доступ:

```
<?php
class Album {
    public function __call($name, $arguments)
{
```

```
        return "Задданая вами функция не
существует $name";
    }
```

```
        public function Create() {
            ...
        }
    }
```

```
$obj = new Album;
$obj->Create(); // Функция существует
$obj->Get(); // Функция не существует
?>
```

Мы рассмотрели перегрузку а теперь посмотрим на магические методы. Первым магическим методом будет метод `__toString()`, метод будет срабатывать тогда когда мы попытаемся превратить класс в строку:

```
<?php
class MyClass {
    public function __toString() {
        return "Это будет строка класса";
    }
}
```

```
$obj = new MyClass;
echo $obj; // Результат:Это будет строка класса
?>
```

Следующий магический метод это `__invoke()`. Метод будет вызываться тогда когда объект будут пытаться вызвать как функцию:

```
<?php
class MyClass {
    public function __invoke($value) {
```

```
        return $value;
    }
}
```

```
$obj = new MyClass;
echo $obj("Hello World");
?>
```

Напоследок это все что я хотел сказать об перегрузке и магических методов. Далее мы рассмотрим трейты, после этого завершается этот раздел который был посвящен ООП.

7.6 Трейты

Трейты немного похожие на классы но используются они для того чтобы структурировать группировать функционал. Сам по себе трейт не сможет существовать, это один из элементов наследования, это использования функционала класса без необходимости:

```
<?php
trait NewTrait {
    public function test() {
        return "Hello world!";
    }
}
?>
```

После этого можно немного отдохнуть и попрактиковаться с материалом который был представлен в этом подразделе.

Заключение

После всех этих сложных тем которые для новичка очень тяжело запомнить можно спокойно расслабиться и попрактиковаться с ООП. Перед нами ещё долгий и сложный путь по базах данных, сессиях, безопасности и т.д. Как и в каждом конце главы я прошу читателя

ответить на вопросы и выполнить вопросы для самоконтроля которые внизу.

Вопросы и упражнения для самоконтроля

1. Что такое ООП? На чем оно основано?
2. Объясните что такое: полиморфизм, наследование и инкапсуляция.
3. Что такое класс? Как его можно создать?
4. Как можно реализовать наследование?
5. Напишите программу с использованием класса которая использовался для того чтобы создавать группу. В классе должны быть такие параметры как: название группы, год основания, краткое описание и основатель.

8 Дата и Файлы

Через язык программирования РНР можно работать с файловой системой вашего приложения. Например, с помощью РНР можно создавать разные файлы/директории. И это ещё не все функции которые мы с вами можем делать с файловой с помощью языка программирования РНР. Файлы можно редактировать прямо с приложения, или читать их содержимое.

В общем РНР функции для работы с файловой системой позволяют реализовать полноценную работу с файлами. Работа с файлами через РНР делиться на 3 стадии, это:

1. Открытие файла(если он существует)
2. Работа с файлом. Например, манипуляция содержимым файла.
3. Закрытие файла и завершение работы с ним.

Именно по таким 3 стадиям мы и будем работать с файловой системой. Безопасность файловой системы в этом разделе рассматриваться не будет, об этом речь пойдет в разделе об безопасности.

8.1 Работа с файлами

Первая стадия работы с файлами как уже упоминалось это открытие самого файла если он существует. В РНР есть функция `fopen()` для того чтобы открывать файлы. Функция имеет 3 параметра, два из которых являются не обязательным и один обязательный. Первый параметр это **файл** который мы хотим открыть, а второй параметр это **режим** через который мы будем работать с файлом:

```
<?php
```

```
$opening = fopen("openit.txt", "r");  
?>
```

Теперь надо задаться вопросом что это за **режим** для работы с файлами я поставил? Режим работы с файлами `r` предназначен для того чтобы открыть файл только для чтения. Внизу я сделал таблицу, в которой показаны все режимы для работы с файлами. Эти режимы можно использовать в любой функции для работы с файловой системой, если в функции есть параметр с помощью которого можно задать режим работы с файлом.

Режим работы	Действие
<code>r</code>	Открыть файл только для чтения.
<code>r+</code>	Открыть файл для чтения и записи.
<code>w</code>	Создание нового файла. Если будет вызвано на существующий файл то файл будет удален.
<code>w+</code>	Создание нового файла. Если будет вызван существующий файл содержимое файла будет удалено.
<code>a</code>	Открытие существующего файла в режиме записи.
<code>a+</code>	Открытие существующего файла в режиме записи и чтения.

- Табл. 9 - Режимы для работы с файлами в PHP

Надеюсь вы прочитали таблицу и поняли что за что отвечает, а теперь я покажу несколько примеров:

```
<?php
$files = fopen("counter.txt", "rt"); //
Текстовый режим чтения указанного файла
$files2 = fopen("http://google.com", "r"); //
Открытие HTTP соединения для чтения страницы
?>
```

Об работе HTTP в PHP мы ещё поговорим, а сейчас мы должны подкрепить ещё много информации для того чтобы двигаться дальше. Окей, мы знаем как делать первую стадию для работы с файлами в PHP а как насчёт второй стадии с помощью которой мы взаимодействием с данными файла? Сделать это можно с помощью функции `fwrite()`, эта функция имеет 3 параметра, первый и второй обязательный, а третий по требованию:

```
<?php
$files = fopen("counter.txt", "a");
$text = "Эта строка будет записана в файл";
$record = fwrite($files, $text);
?>
```

Из этого можно сказать что первом параметре функции мы указываем имя файла в который хотим внести изменения, а второй аргумент это уже строка которую мы хотим внести в файл. А третий параметр мы вносим если нам надо, он отвечает за длину строки которую мы хотим внести в файл.

Если мы хотим вывести весь файл по строкам то в этом случае нам надо использовать функцию `fgets()`:

```
<?php
$myfile = fopen("count.txt", "r");

if ($myfile) {
    while (!feof($myfile)) {
```

```
        $text = fgets($myfile, 999);  
        echo $text;  
    }  
}  
else {  
    exit ('Error with file');  
}  
?>
```

Функция имеет два параметра которые мы должны обязательно указать. Первый это название файла к которому мы хотим обратиться и второй это сколько строк мы хотим взять с файла. Посмотрите на условие в цикле которое мы создали, здесь мы использовали функцию `feof` которая проверяет наступил ли конец файла.

В нашем мы поставили оператор `!` которая является логическим не, то есть работу нашего с вами цикла можно описать так, пока не наступил конец файла то не переставать выводить строки. Ну в конечном результате мы закрываем наш файл с помощью функции `fclose()` :

```
<?php  
$myfile = fopen("count.txt", "r");  
fclose($myfile);  
?>
```

В следующем под-разделе мы посмотрим какие есть функции для работы с файлами, я выведу лишь некоторые функции.

8.2 Смена прав доступа

Смена прав доступа к файлу это нужная вещь. К примеру если вы будете создавать приложение где есть группы пользователей и надо будет кто именно может иметь доступ к определенному файлу. Сменить права доступа можно с помощью функции `chmod()` .

Функция имеет два параметра которые надо указать это `filename` - то есть название файла к которому мы будем обращаться, и

параметр `mode` в котором мы и указываем какие права доступа мы хотим поставить на файл с помощью 8-ми разрядных чисел.

В разделе об числах я вам уже говорил об числах с разными разрядами. 8-ми разрядные числа это набор цифр в котором есть только 8 символов: 0,1, 2...7. Цифра 8 не указывается в 8-ми разрядной системе счисления.

В 8-разрядной системе счисления счёт как и в десятичной системе счисления начинается от единицы и идет дальше. Но когда будет цифра 7 то после её не будет 8 а будет 10. А потом если будет 17 т снова таки не будет 18 а будет 20.

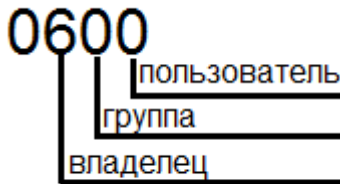
После того как мы выяснили как именно работает 8-ми разрядная система счисления можно поговорить об том как именно можно задавать права доступа. Параметр `mode` имеет три 8-ми разрядные числа с помощью которых мы можем задать права доступа для файла. Есть 3 цифры которыми задаются права, но если суммировать эти числа то можно создавать разные типа прав доступа. Вот 3 цифры и их значения:

- 1 - доступ на выполнение;
- 2 - доступ для записи;
- 4 - доступ на чтение;

Чтобы лучше понять как именно это работает посмотрите на пример который показан внизу:

```
<?php
    chmod("count.txt", 0600); // В этом
примере владелец файла имеет права для записи и
чтения, другие не имеют доступа к файлу
?>
```

Из только что показанного примера можно вытянуть себе такую схему которая объясняет какой именно разряд чисел имеет ответственность:



- *Рис. 5 - Схема объясняющая то как устроены права доступа в PHP*

Где нули там нету вообще никаких прав доступа. Владелец в нашем примере имеет почти полные права для доступа. Это чётко понятно из значения старшего разряда. Что ж я надеюсь что вы хотя бы немного поняли как именно это работает. Теперь рассмотрим ещё несколько примеров:

```
<?php
    chmod("text.txt", 0604); // Владелец может
    читать и записывать данные в файл, а пользователь
    может читать и выполнять файл.
    chmod("count.txt", 0644); // Почти то же самое
    что и в предыдущем примере только здесь уже группа
    может выполнять и читать файл
    chmod("doc1.txt", 0755); // Владелец получает
    полный доступ к файлу, а группа и пользователь
    получают доступ на чтение и запись
    chmod("tetris.txt", 0777); // Все получают
    полный доступ к файлу
?>
```

Как я уже и говорил смена прав доступа это важная вещь. Вот что можно сказать в заключение данного под раздела.

8.3 Проверка файлов

Затроним тему об безопасности файлов, в данном подразделе мы рассмотрим функции и способы которыми можно делать проверку файлов. К примеру проверку файлов на формат, размер и т.д. Первой функцией с которой мы познакомимся это `is_dir()` которая проверяет является указанное нами имя файла директорией.

Функция `is_dir`

Функция имеет только один параметр это `filename` и данные функция возвращает в двух логических значениях истины и лжи. Пример:

```
<?php
var_dump( is_dir("php.txt") ); // false
var_dump( is_dir("files") ); // true
?>
```

Поскольку простые команды для вывода данных не способны отобразить логическое значение то мы использовали в этом примере функцию `var_dump()` которая используется для того чтобы отобразить полные данные об переменной или об других значениях. Чтобы понять в каких целях можно использовать данную функцию вот пример:

```
<?php
$direct = "default";

if (is_dir($direct) {
    echo "Данные проверены, продолжаем
работу";
    ...
} else {
    echo "Простите но это не файл.";
}
?>
```

Из только что показанного примера можно понять что с помощью данной функции можно реализовать полноценную проверку данных, этот вопрос безопасности очень важен. А теперь посмотрим на функцию `is_file()` которая определяет является ли заданный объект файлом.

Функция `is_file()`

Как и функция `is_dir()` имеет только один параметр, т.е. название файла который мы хотим проверить. И также эта функция возвращает данные в виде логических двоих логических значений. Пример:

```
<?php
var_dump( is_file("folder/user.txt") ); //true
var_dump( is_file("folder3/info/") ); //false
?>
```

Думаю что делать дополнительного примера для этой функции уже не надо. Вы уже познали две такие функции который используются для проверки в файловой системе. Двигаемся дальше, следующей функцией будет `is_readable()` проверяющая существование файла и доступен ли он для чтения.

Функция `is_readable()`

Какие и две предыдущие функции имеет один параметр и выводит данные в виде двоих логических значений:

```
<?php
var_dump( is_readable("counter.txt") ); //
true, но вообще значение зависит от прав доступа
?>
```


Функция `is_writable()`

Функция имеет один параметр и работает также как и предыдущие функции. Вообще функция проверяет доступен ли файл для записи:

```
<?php
var_dump( is_writable("counter.txt") ); //
```

Значение зависит от прав доступа к файлу

```
?>
```

Функция `file_exists()`

Функция имеет один параметр и возвращает данные в виде двоих логических значений истине и ложь. Функция проверяет существует ли файл или нет:

```
<?php
var_dump( file_exists("test.txt") ); // Будет
равно истине если файл существует, или ложь если нет
?>
```

На этом я хочу закончить данный подраздел, но об безопасности с файлами мы ещё поговорим. А также в отдельном подразделе рассмотрим некоторые функции для работы с файлами.

8.4 Функции для работы с файлами

Как и для каждого типа данных, для работы с файлами также есть свои функции с помощью которых вы можете манипулировать файлами и данными которые содержат эти файлы. Вы уже познакомились с несколькими такими функциями, но если вы захотите сделать какие то более совершенные вещи с файлами, то вам конечно же понадобится больше функций который расширят круг возможностей для работы с файлами. В таблице 10 приведены некоторые функции для работы с файлами:

Функция	Действие
Copy	Копирует файл.
pathinfo	Возвращает данные пути к файлу.
Mkdir	Создаёт директорию.
Rmdir	Удаляет директорию.
Stat	Возвращает информацию об файле.
filetype	Возвращает тип файла.
fileowner	Возвращает данные о владельце файла.
fileperms	Возвращает данные о правах доступа.
filegroup	Возвращает данные о группе файла.
filesize	Возвращает данные о размере файла.

- *Табл. 10 - Некоторые функции для работы с файлами в PHP*

Конечно здесь вы можете не найти функций описание которых отвечает той функции, которую вы хотите найти. В таком случае воспользуйтесь документацией функций по работе с файлами.

Хорошо, вы увидели довольно таки небольшой список функций которые предназначены для работы с файлами. Здесь я указал функции которые как я считаю, могут вам понадобиться в разработке. Поэтому давайте быстро пройдемся по каждой функции которая указана в таблице.

Функция `copy()`

Как было сказано функция может копировать файлы. Имеет 3 параметра, 2 обязательных и 3-ий не обязательный. Параметры: `filename` – файл который мы хотим скопировать и путь к этому

файлу, `dest` – имя файла который мы скопируем, а также путь к этому файлу. Пример использования функции:

```
<?php
copy("doc.txt", "doc-copy.txt");
?>
```

Функция `pathinfo()`

Функция имеет два параметра, `path` – путь к файлу который мы хотим проанализировать и `options` – массив в котором есть много элементов с помощью которых можно получить много информации о нахождении файла.

Массив имеет такие элементы:

`dir` – путь к папке, внутри которой находится файл.

`basename` – полное имя файла.

`extension` – формат файла.

`filename` – название файла.

Массив работает по принципу ключ-значение. Пример:

```
<?php
echo "<pre>";
print_r(pathinfo("user/challenge.txt"));
echo "</pre>";
?>
```

Результат:

```
Array
(
    [dirname] => user/
    [basename] => challenge.txt
    [extension] => txt
    [filename] => challenge
)
```

Используя функцию `print_r` которая предназначена для того чтобы вывести данные об переменной или других объектах мы смогли увидеть что данные мы получили в виде массива. Заметьте, что здесь с помощью команды `echo` я открыл тег `<pre>` чтобы вывод результата был красивым и легким для чтения.

Функция `mkdir()`

Как было указано в таблице, то эта функция используется для того чтобы создавать директорию. Функция имеет целых 4 параметра:

`pathname` – путь к директории.

`mode` – режим прав доступа.

`recursive` – разрешает создавать вложенные директории, которые указаны в первом параметре.

`context` – описание контекста.

Мы рассмотрим только два параметра. Пример создания директории с помощью этой функции:

```
<?php
mkdir("/user/mydir");
mkdir("/user/secret/" , 0700);
?>
```

Поскольку вы уже знакомы с такой функцией как `chmod()` то вы уже знаете как можно задавать права доступа. В нашем примере сначала мы просто создали директорию не указывая прав доступа, а уже потом создали директорию с полными правами доступа только для владельца файла.

Функция `rmdir()`

Функция имеет только два параметра: `dirname` и `context`. Как было сказано в таблице, то данная функция удаляет директорию:

```
<?php
rmdir("user/delete_it");
```

?>

Можно также использовать аналог этой функции - `unlink()`.

Функция `stat()`

Функция имеет только один параметр это название файла. Но при этом данные об файле возвращаются в виде массива, который имеет 12 элементов, массив работает по принципу ключ-значение. Пример вызова функции:

```
<?php
echo "<pre>";
print_r(stat("main/hello.jpg"));
echo "</pre>";
?>
```

Хорошо, а теперь поговорим о том какие именно есть элементы массива и за какие функции они отвечают. Внизу приведена таблица, в которой показываются все элементы массива, с помощью которых можно получить разную информацию:

Элемент	Значение
Dev	Имя устройства.
Ino	Номер inode.
Mode	Режим защиты inode.
Nlink	Количество ссылок.
Uid	Идентификатор владельца.
Gid	Идентификатор группы.
Rdev	Тип устройства, если устройств inode.

Size	Размер файла в байтах.
Atime	Время последнего обращения к файлу.
Mtime	Время последней модификации файла.
Ctime	Время последнего изменения inode.
blksize	Размер блока ввода-вывода файловой системы.
Blocks	Количество используемых 512-байтовых блоков.

- Табл. 11 - Элементы массива функции `stat()` в PHP

Мы не будем рассматривать каждый элемент, особенно мы не нуждаемся в том чтобы рассмотреть последних два элемента. Чтобы вывести какую-нибудь информацию об каком то заданном файле используя эту функцию, то мы должны задать один из ключей который показан в табл. 11. Например:

```
<?php
$info_file = stat("info.doc");
echo $info_file["mtime"]; // В результате
получим время модификации файла
?>
```

Данную функцию хорошо использовать тогда когда мы хотим достать какие то дополнительные данные об файле кроме его названия.

Функция `filetype()`

Функция имеет только один параметр, это имя файла или путь к нему. Как было уже сказано, то эта функция проверяет тип файла, т. е. это типичная функция для проверки файлов. Значения будут возвращаться в виде названия типа файла, но главное это не путать с форматом файла:

```
<?php
echo filetype('info/logo.jpg'); // Значение
будет file
echo filetype('user/data/'); // Если это будет
папка то получим dir
?>
```

Функция fileowner()

Имеет только один параметр - filename. Функция находит идентификатор владельца файла. Чтобы бы не было ошибки используйте функцию posix_getpwuid() чтобы получить имя владельца в виде строки:

```
<?php
print_r(posix_getpwuid(fileowner('user/info.txt')));
?>
```

На этой ноте я хочу завершить написание этого подраздела об функциях для работы с файлами. Да, у нас ещё осталось несколько функций, которые мы можем рассмотреть, но их работа и вообще сами они очень просты. Поэтому я завершу этот подраздел и будем уже рассматривать работу с датой.

8.5 Дата

Представьте что вам придется создать приложение в котором будут записи и когда пользователь будет создавать новую запись должна будет записываться дата создания записи. Чтобы это реализовать в PHP

есть отдельный класс и функции для этого. То есть класс для реализации даты и функции для работы с датой.

Самой простой функцией для вывода времени является функция `time()`, если мы вызовем эту функцию то получим значение в секундах после появления UNIX эпохи. Окей, хорошо мы только что увидели просто функцию для развлечения, но должна существовать функция которая могла бы быть полезной для нас во время разработки.

И да, такая функция есть. Функцией этой является `getdate()` которая вычитывает информацию о времени и дате. Внутри функции есть массив в котором как раз и содержится информация о дате и времени которую мы можем увидеть. Массив работает по принципу ключ-значение, внизу показаны названия ключей и значения которые они могут принимать:

Ключ	Описание
<code>seconds</code>	Секунды (от 0 до 59)
<code>minutes</code>	Минуты (от 0 до 59)
<code>hours</code>	Часы (от 0 до 23)
<code>mday</code>	День месяца (от 1 до 31)
<code>wday</code>	День недели (от 1 до 6)
<code>mon</code>	Номер месяца (от 1 до 12)
<code>year</code>	Год
<code>yday</code>	День года (от 0 до 365)
<code>weekday</code>	Название дня недели (от Sunday до Saturday)

month	Название месяца (от January до December)
0	Количество секунд прошедших после создания эпохи UNIX.

- Табл. 12 - Элементы массива функции `time()`

Ну и вот мы наконец-то нашли функцию которая может нам быть полезна в разработке, к примеру, дата создания пользователем записи.

Пример использования:

```
<?php
$day = getdate();
print_r($day);
?>
```

Чтобы не было ошибки вывода данных, мы использовали функцию `print_r` которая используется для того чтобы выводить информацию об переменной в удобочитаемом виде. Я говорю об ошибке которая появилась бы если бы мы попробовали вывести данные даты с помощью простой команды `echo`.

Проблема заключается в том что если мы пробуем вывести данные через `echo` то мы сразу получим ошибку - а все через то что мы пытаемся передать команде целый отдельный объект даты, который даже не конвертирован в строку. И именно через это мы получаем ошибку. Как выводит данные объекта даты, мы об этом поговорим позже. Давайте вернемся к нашему примеру. В результате вы должны увидеть названия ключей и их значения. Хорошо, ну а что же дальше? Может ещё есть функции которые позволяют нам работать с датой?

Конечно что да! Их очень много также как и других функций, которые предназначены для работы с определенным типом данных. И одной из таких функций является функция `date()` которая форматирует вывод системной даты, то есть мы сами задаем дату и время которое нам надо, и при этом мы можем конвертировать данные так чтобы не

получать ошибку, при выводе с помощью команды `echo` или любой другой команды для вывода текстовых данных.

Функция имеет два обязательных параметра. Первый это шаблон, по которому мы хотим отформатировать дату и второй это есть метка времени в которой может быть задано нами время или системное время.

Пример использования функции:

```
<?php
echo date("Y m d");
?>
```

Окей, и что мы только что создали? Как мы видим, то мы указали только первый параметр. Если вы запустите этот код в браузере то увидите текущий год, номер месяца и день месяца. Ну и как мы видим то данная функция также является очень полезной, поскольку мы можем нормально выводить данные даты и времени через `echo`.

Данная функция имеет массив в котором содержится вся информация о дате и времени в ключах. В таблице 13 показаны почти все ключи которые есть внутри массива:

Ключ	Описание
D	День месяца с ведущим нулем(от 01 до 31)
D	Название дня недели 3-мя символами(от Mon до Sun)
J	День месяца без ведущего нуля(от 1 до 31)
L	Полное название недели(от Sunday до Saturday)
N	Порядковый день недели в соответствии со

	стандартом ISO-8601 (от 1 до 7)
S	Английский суффикс порядкового числительно дня месяца в двух символах (st, nd, rd или th)
W	Номер дня недели (от 0 до 6)
Z	День в году (от 0 до 365)
W	Порядковый номер недели года в соответствии со стандартом ISO-8601, начиная с понедельника
F	Полное название месяца (например May)
M	Номер месяца с ведущим нулем (от 01 до 12)
M	Сокращенное название месяца 3-мя символами (от Jan до Dec)
N	Номер месяца без ведущего нуля (от 1 до 12)
T	Количество дней в заданном месяце (от 28 до 31)
L	Високосный год или нет (1 если да или 0 если н високосный)
O	Номер года в соответствии со стандартом ISO-8601 (Пример 2018)
Y	Номер года (Пример 2018)
Y	Номер года в двух цифрах (пример 03)

B	Время в формате интернет-времени(от 000 до 999)
G	Часы в 12-часовом формате без ведущего нуля (от 1 до 12)
G	Часы в 24-часовом формате без ведущего нуля (от 1 до 23)
H	Часы в 12-часовом формате с ведущим нулем (от 01 до 12)
H	Часы в 24-часовом формате с ведущим нулем (от 01 до 23)
I	Минуты с ведущим нулем (от 00 до 59)
S	Секунды с ведущим нулем (от 00 до 59)
U	Микросекунды, функция date() всегда будет возвращать 000000 поскольку данная функция принимает только целые числа
V	Миллисекунды. Принимает также только целые числа как и date()
E	Идентификатор временной зоны (пример GTC)
I	Признак летнего времени(1 если да или 0 если нет)

- Табл. 13 - Форматы времени для функций времени

С этого можно сделать вывод - что ключей есть очень много и с помощью их мы можем сделать отображение любого вида даты и времени. Это кстати ещё не все ключи, которые доступны, есть также и другие ключи которые вы сможете посмотреть в официальной документации этой функции.

Что ж теперь точно можно сказать что данная функция очень полезна, поскольку теперь можно точно сказать что мы можем манипулировать временем. Для того чтобы мы могли задавать своё время то во втором аргументе функции `date()` надо вызвать функцию `mktime()`, с помощью которой мы и зададим желаемое нами **время**.

Конечно можно создать переменную и внутри этой переменной указать функцию, и уже потом отформатировать заданное время через функцию `date()`. Функция `mktime()` имеет целых 6 параметров которые вы можете указать. Внизу в небольшой таблице поданы все 6 параметров и описание за что они отвечают:

Параметр	Описание
hour	Количество часов которые прошли с начала указания параметров month, day и year.
minute	Количество минут прошедших после указания параметра hour.
second	Количество секунд прошедших после указания параметре minute.
month	Количество месяцев прошедших с конца предыдущего года.
day	Количество дней прошедших с конца предыдущего месяца.
year	Номер года.

- Табл. 14 - Параметры функции `mktime()`

Хорошо, а теперь давайте немного по практикуемся:

```
<?php
```

```
echo date("M-d-Y", mktime(0, 0, 0, 4, 25, 2018));  
?>
```

Окей, ну и что мы только что сделали? Во-первых, заметьте то в какой способ мы отформатировали указанное время. Если мы запустим этот скрипт в браузере то увидим название месяца, день месяца и год который был указан с помощью функции `mktime()`.

Конечно-же вы можете указать дату так как вы этого хотите и отформатировать в свой способ воспользовавшись таблицей с ключами для функции `date()` или официальной документацией разработчиков.

Давайте рассмотрим ещё несколько примеров:

```
<?php  
    $mydate = mktime(20, 45, 0, 4, 25, 2018);  
    echo date("G i - d Y" , $mydate);  
?>
```

Этот пример намного интересен, кроме этого в датах можно также ставить противоположные знаки:

```
<?php  
    $mydate = mktime(0, 0, 0, 3, -24, 2018);  
    echo date("d Y", $mydate);  
?>
```

Только что мы рассмотрели один способ создания даты и времени в PHP, но кроме этого есть также и другой способ созданий даты напрямую через встроенный класс `DateTime`. Создание объекта даты напрямую:

```
<?php  
    $mydate = new DateTime("18-04-25");  
    echo $mydate->format("Y-m-d");  
?>
```

Как можно увидеть из примера то мы создали новый объект класса `DateTime`, после чего мы отформатировали объект функцией

которая встроена внутри этого класса - `format`. А чтобы изменить метку времени есть функция `modify`:

```
<?php
$mydate = new DateTime("18-05-11");
$mydate->modify("+1 day");
$mydate->modify("+1 month");
echo $mydate->date("Y-m-d")
?>
```

В этом примере, мы сделали модификацию нашей временной метки добавив, 1 день и 1 месяц в значение этого объекта. На этом я хочу завершить этот раздел об работе с датой. Если вы хотите узнать по больше об функциях для работы датой можете заглянуть в официальную документацию.

9 PHP и HTTP

После того как мы изучили типы данных, логические операции, циклы, массивы, функции и принципы ООП можно начинать изучать более сложные вещи в PHP. У вас уже достаточно знаний чтобы начать двигаться дальше. От этого раздела мы уже начнём создавать небольшие простые мини веб-приложения в которых уже пользователь сможет взаимодействовать с самим веб-приложением. Для начала мы должны изучить как можно работать с формами.

Я надеюсь что читатель знаком уже с основами языка разметки HTML и понимает что такое теги, атрибуты и как с ними правильно работать. Но перед тем как мы начнём я хочу пояснить два понятия. Первое это что такое входные данные и второе это что такое выходные данные. Не каждый новичок в программировании сразу поймет что это такое. Но вообще это очень просто. Когда мы вводим в программу какие то данные на обработку, то эти данные называются входными.

После этого когда программа возвращает обработанные данные то эти данные уже называются выходными. А сейчас начнём изучать то о чем я говорил раньше. Внизу показана схема которая как раз и показывает принцип работы входных и выходных данных.

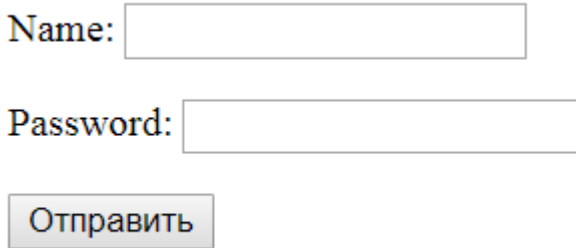


- *Рис. 6 - Схема которая демонстрирует работу входных и выходных данных*

9.1 Работа с формами

Сейчас мы создадим небольшую программу которая будет демонстрировать работу с формами в PHP. Но что же из себя представляет форма? Форма это объект внутри с помощью которого пользователь сможет передавать какие то данные на сервер приложения в котором находится данная форма.

После того как сервер получит данные от формы он сможет их обработать с помощью заданного скрипта для обработки формы. Внизу показан простой пример формы которая была создана на языке разметки HTML:



The image shows a simple web form. It consists of two text input fields. The first field is preceded by the label "Name:" and the second by "Password:". Below these fields is a button with the text "Отправить" (Submit) in Russian. The form is styled with a light gray border and a simple layout.

- *Рис. 7 - Пример простой формы для того чтобы авторизоваться*

Как вы можете заметить то в нашей форме есть два поля, в которые пользователь и будет заносить какие то данные, а также есть специальная кнопка чтобы отправить данные скрипту который и будет обрабатывать форму. Сейчас мы с вами создадим свою форму которая также будет иметь свои поля и скрипт который будет обрабатывать данные.

Из сказанного выше можно понять что у нас будет два файла которые будут работать между собой. Один будет хранить в себе HTML-

код формы, а второй будет срабатывать тогда когда форма будет отправлена и обрабатывать полученные данные от формы.

Пусть наша форма будет иметь только 2 поля, первое поле будет называться name т.е. это будет поле для имени как вы смогли понять, а второе поле это уже будет поле в которое пользователь может внести любую информацию и назовём это поле мы info. Хорошо, а теперь давайте как раз и создадим наш с вами скрипт, назвав его form.html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My form</title>
  </head>
  <body>
    <form action="process.php"
method="POST">
      <label>Name: </label><br />
      <input type="text" name="name"
/>
      <label>Info: </label><br />
      <input type="text" name="info"
/>
      <input type="submit" />
    </form>
  </body>
</html>
```

Теперь я объясню что за что отвечает, во-первых у нас есть тег <form> который как раз и создаёт данную форму. Если посмотреть на атрибуты которые внутри этого тега то здесь можно заметить атрибут action внутри которого указывается название скрипта который будет получать данные от формы и обрабатывать их.

Дальше идет атрибут method с помощью которого мы задаем HTTP метод с помощью которого будет работать форма. Поскольку мы хотим сделать отправку данных то мы будем использовать метод POST который используется для отправки данных. Об HTTP-методах речь

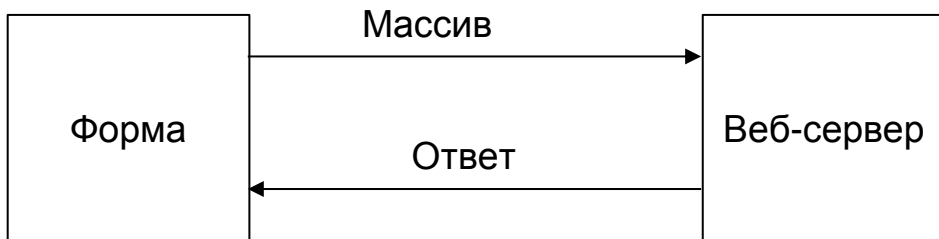
пойдет в следующих подразделах этого раздела. Давайте поговорим об созданных нами полях. Создали мы поля с помощью тега `<input>`, после этого мы задали тип для поля с помощью атрибута `type`, в нашем случае наши поля имеют текстовый тип то есть `text`.

Поля могут иметь разные типы. Например, есть специальный для загрузки файлов, или например тип для большого текста. Чтобы наша форма выглядела более менее понятной для простого пользователя то для этого мы использовали два тега `<label>` с помощью которых мы просто создали текстовые названия наших полей. Эти теги не мешают работать нашей форме, они просто делают её более понятной для пользователя.

Заметьте, что в конце формы мы как будто создаём ещё одно поле, но по-настоящему это кнопка, которая используется для обработки формы и отправки данных в заданный скрипт. **Важно помнить**, что если такая кнопка будет создана на простой странице где нету формы, то при нажатии на эту кнопку никакой отправки данных не будет и в общем ничего не произойдет. А теперь давайте перейдем к разговору об том как мы будем получать и обрабатывать полученные данные.

Как я и говорил то у нас будет скрипт, который и будет обрабатывать полученные данные от формы. Скрипт ничего такого сложного делать не будет, просто будет выводить полученные данные на экран. PHP устроен так что после того как мы сделали отправку данных то отправленные данные попадают в супер-переменную `$_POST` где эти данные получают свои индексы с названием поля, а значение ключа это уже те данные которые были введены в поле.

Если мы захотим обратиться к каким то отправленным данным из формы то нам надо будет просто указать в массиве `$_POST` индекс с названием поля, после этого мы получим отправленные данные. Но об этом мы поговорим скоро, а пока что посмотрите на схему, которая показана внизу, которая легко объясняет что и как работает:



- Рис. 8 - Демонстрация работы формы с веб-сервером

Из схемы чётко видно, что данные отправляются от формы в массив POST, после чего они отправляются в скрипт который и обрабатывает полученные данные а также обращается к ним, и делать он это может не один раз. Думаю некоторые зададутся вопросом, что же значит понятие супер-переменная в PHP? Супер-переменные это специальные переменные в PHP которые имеют глобальную область видимости и могут быть использованы везде в любой части кода и в любом файле.

Каждая супер-переменная отвечает за какую то вещь. Например, супер-переменная `$_POST` отвечает за получение и хранение данных отправленных через форму. Мы ещё познакомимся с другими супер-переменными, а пока что нас с вами интересует только супер-переменная `$_POST`. Давайте создадим скрипт который и будет обрабатывать данные отправленные из формы и назовём мы его `process.php`:

```
<?php
echo $_POST["name"] . "<br />";
echo $_POST["info"]; // Как вы можете здесь
заметить, то здесь мы делаем вывод данных которые
были введены в поля name и info.
```

```
// Т.е. как я уже и говорил, что если мы хотим
обратиться к данным которые были отправлены через
форму, мы должны указать названия полей в массиве
POST
```

```
?>
```

Теперь сохраните файл назвав его `process.php`, в результате у вас должно получиться два файла, это `form.html` то есть файл внутри которого храниться сама форма и `process.php` который обрабатывает полученные данные от формы. Проверьте, правильны ли названия ваших файлов, а также проверьте атрибут `action` в файле с формой. Главное чтобы в этом атрибуте было указано имя файла, который обрабатывает форму.

Ну а теперь настало время протестировать вашу форму, перейдите на адрес вашего приложения на файл с формой. После этого введите какие-то данные в форму и отправьте их. В результате вас должно перенаправить на скрипт, который обрабатывает отправленные данные. Если вы все делали так как я говорил то у вас все должно получиться, если что то не получается то пересмотрите все то о чем я говорил выше и повторите данные шаги снова.

В завершение этого подраздела скажу что форму бывают не только для того чтобы вводить какие то текстовые данные, но также для того чтобы отправлять какие-то файлы и т.п. Поэтому двигаемся дальше, и следующее что на нас с вами ждет это поговорить о том какие существуют HTTP-методы.

9.2 HTTP-методы

В предыдущем подразделе мы научились как можно работать с формами, а также познакомились с одним из HTTP-методов т.е. с методом POST. Я уже говорил, что кроме этого метода существуют и другие методы для работы с HTTP которые намного больше позволяют расширить возможности работы с HTTP.

Знать какие существуют HTTP-методы это важно. Поскольку с помощью них как раз и осуществляется отправка данных, удаление данных и т.п. в сети. Если кто то не знает то HTTP это стандартный протокол передачи данных гипертекста по сети. Надеюсь вы помните как в начале книги мы рассматривали как именно работает сервер, поскольку

сейчас это нам понадобится. Внизу показана схема работы сервера чтобы напомнить читателю как все работает:



- *Рис. 9 - Работа клиента с сервером*

Как мы видим из схемы то работа протокола HTTP осуществляется по принципу **клиент-сервер**. О том как вообще работает весь протокол я говорить не буду, поскольку знать о том что сервер работает по принципу клиент-сервер уже достаточно, поэтому переходим к основной теме этого подраздела т.е. к методам которые используются в HTTP. Но перед тем как перейдем к самому изучению методов мы должны понять как именно строятся HTTP-запросы. Как не странно то когда мы вводим в адресной строке браузера адрес сайта и нажимаем `enter` то браузер автоматически создаёт запрос.

Знать как выглядит запрос это важно, поскольку можно будет понять что именно происходит. Вот пример запроса на содержимое страницы `/about` веб-сайта `example.com` который используется в качестве примера:

```
GET /about/ HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1;
rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html
Connection: close
```

Объясню, что именно происходит, во-первых в самой первой строке мы делаем запрос содержимого страницы /about на сайте example.com. Посмотрите внимательно, для этого запроса мы используем метод GET, а также после того как мы указали название страницы мы указали версию протокола которую мы хотим использовать HTTP/1.1. Эта версия сейчас используется наибольшее, конечно же есть и версия HTTP/2.2.

В следующей строке кода запроса указан адрес веб-сайта к которому мы делаем запрос. В третьей строке уже содержится информация об том какой браузер используется для запроса страницы, кроме информации об браузере есть также информация операционной системе.

В четвертой строке содержится информация об том в каком виде возвращать нам содержимое страницы. В нашем случае это в виде статической HTML веб-страницы. В виде статической веб-страницы содержимое веб-сайта возвращается наиболее часто, конечно же содержимое может возвращаться и в других видах, к примеру в виде файлов. И в последней строке указано, что у нас закрытое соединение. Вы уже понимаете как выглядит содержимое запроса. Можно продолжать разговор дальше и сейчас мы поговорим об методах которые используется в запросах и ответах.

Метод GET

Данный метод используется для того чтобы доставать какие то данные из веб-сервера. Простым примером того как работает этот метод может служить тот момент когда мы с вами делаем запрос на получение содержимого веб-сайта. То есть когда мы посылаем запрос серверу на получение содержимого сайта, мы используем метод GET.

И если сервер подтвердит запрос, то мы в свою очередь получим содержимое веб-сайта в виде статической страницы. Конечно же ответ можно также получить в виде динамической страницы, но об этом речь в

этой книге пойдет намного позже. Заметьте также что для того чтобы получить данные мы используем метод GET, а для того чтобы отправить мы используем метод POST.

Метод PUT

Этот метод уже используется для того чтобы загрузить указанные данные на указанный URI адрес. Долго объяснять не буду, вы просто загружаете данные с помощью этого метода на URI адрес.

Метод PATCH

Метод аналогичен методу PUT, только используется к целому ресурсу, а к её небольшой части.

Метод DELETE

Тут уже с названия ясно для чего применяться этот метод. То есть для удаления определенного ресурса.

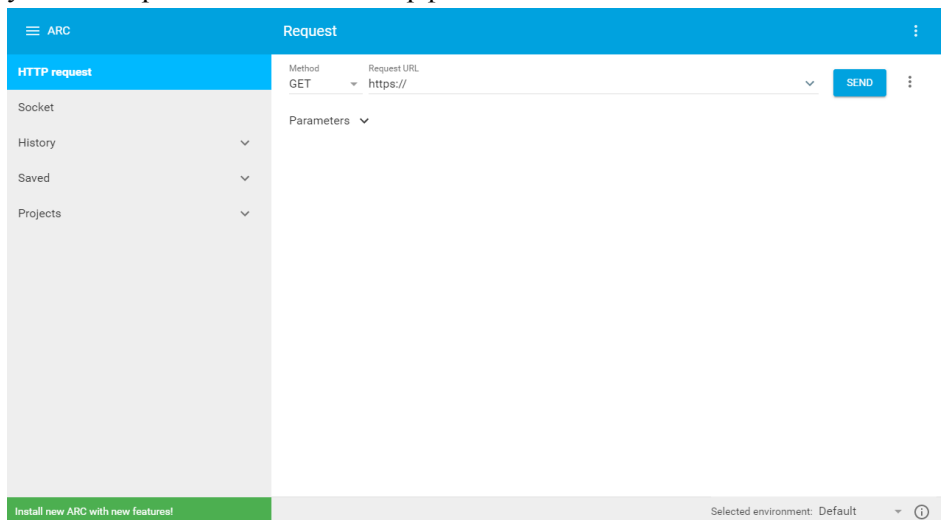
Только что мы с вами рассмотрели неполный список HTTP-методов, конечно же, я мог указать все методы, но я знаю что люди хотят услышать саму суть, поэтому указал только те с которыми вы можете частенько встретиться во время разработки.

Если нам надо будет рассмотреть тело запроса, то это можно сделать с помощью разных приложений для браузера Google Chrome или Mozilla. Это уже конечно же зависит какой браузер вы используется, главное чтобы вы могли с помощью установленного приложения рассмотреть что находится внутри тела HTTP запроса.

Одним из таких приложений которые позволяют это делать является ARC или *Advanced REST Client*. Данное приложение позволяет не только рассматривать тело запросов и данные, которые отправляются через эти запросы. Но также и создавать сами эти запросы. То есть мы можем задавать методы и данные которые будут в этом запросе.

Кроме *ARC* есть также и другое приложение – *Postman*, его также хорошо использовать для того чтобы тестировать запросы и создавать их, но интерфейс и управление немного сложнее чем у *ARC*. Но в этой книге и в этом разделе мы будем использовать именно *ARC*, поэтому установите данное приложение и включите его. Если кто то не знает то найти все установленные приложения в Google Chrome можно по адресу `chrome://apps`.

А теперь после того как вы открыли приложение, то вы должны увидеть перед собой такой интерфейс:




- Рис. 11 - Интерфейс приложения *ARC* в Google Chrome

На первый взгляд выглядит сложно и немного не понятно. Но если посмотреть внимательно то можно увидеть сверху надпись *Request*, т.е. если перевести с английского то – запрос. На этой странице вы создаете запросы, думаю, вы заметили здесь поле *Method* с помощью которого мы можем задать метод для нашего запроса.

Дальше идет URL адрес к которому мы хотим сделать запрос. А ещё дальше идет кнопка для того чтобы отправить запрос. Ниже мы уже задаем параметры запроса, а также редактируем тело запроса.

Parameters ^

Headers	Variables
 <> Toggle source mode + Insert headers set	
Header name	Header value
Content-Type	application/json
× ✎ ?	
ADD HEADER	
✓ Headers are valid Headers size: 30 bytes	

- Рис. 12 - Поля для создания запроса в ARC

Как мы видим, то здесь мы можем задать заголовки нашего запроса, кроме этого добавить переменные в заголовок запроса. В нашем случае то что вы сейчас вы видите то это те параметры которые доступны для метода GET. Если я выберу что у моего запроса будет метод POST то здесь я уже как раз и смогу редактировать тело запроса:

Headers	Body	Variables
Body content type	Editor view	
application/json	Raw input	
FORMAT JSON MINIFY JSON		
<pre>{ }</pre>		

- Рис. 13 - Задание дополнительных параметров для запроса

Здесь мы указываем данные которые будут передаваться через запрос, данные будут в формате языка нотации JSON. Мы можем также выбрать

и другой формат для передачи данных через запрос, но лучше использовать JSON поскольку он простой и понятный.

Но сейчас речь не будет идти об JSON. Сейчас же мы продолжим разбираться что и как устроено в нашем интерфейсе. Думаю, вы заметили, что у нас с левой стороны есть меню. Через это меню можно просмотреть вашу историю запросов, подключить сокет, просмотреть сохраненные запросы или проекты.

Хорошо, а теперь пора протестировать какой-нибудь запрос с помощью PHP и этого приложения. Давайте создадим файл `request.php` к которому мы будем делать запросы через ARC после чего будем получать результаты на странице с нашим скриптом.

Одной из супер-переменных, которая может служить для того чтобы взаимодействовать с HTTP является `$_SERVER`:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    echo "Метод запроса: POST";
} else if ($_SERVER["REQUEST_METHOD"] == "GET")
{
    echo "Метод запроса: GET";
} else {
    echo "Произошла ошибка;
}
?>
```

Здесь в нашей логической конструкции мы хотим сделать проверку того какой метод используется в запросе. Для этого через супер-переменную `$_SERVER` мы обращаемся к индексу `REQUEST_METHOD` после чего указываем название HTTP-метода который мы хотим с вами проверить после запроса.

Думаю я не очень понятно объяснил, поэтому я просто напишу на псевдокоде алгоритм только что написанного PHP-кода:

```
Если метода запроса == POST тогда:
    Вывести надпись "Метод запроса: POST";
```

```
Иначе если метода запроса == GET тогда:
    Вывести надпись "Метод запроса: GET";
Иначе:
    Вывести надпись "Произошла ошибка";
```

Надеюсь, что алгоритм написанный на псевдокоде смог вам помочь в том что мы только что написали. А теперь сохраните этот файл под названием `request.php` и перейдите в приложение ARC и сделайте запрос с методом GET на этот файл. У меня формирование запроса выглядит вот так:



- P

ис. 14 - Поле для создания запроса в ARC

Главное чтобы вы делали запрос на адрес файла, который как раз и будет проверять тип запроса. Задайте метод для запроса GET после чего укажите точный адрес файла на который вы хотите сделать запрос, в нашем случае мы хотим увидеть результат после отправки запрос поэтому мы делаем запрос на файл `request.php`.

После того отправки запроса перейдите на адрес этого файла в браузере. И в результате вы должны увидеть надпись "Метод запроса: GET". Если мы поменяем HTTP-метод то получим другой результат. Например если мы с вами будем использовать метод POST то получим другую надпись, но если мы выберем метод который не заданный в логической конструкции то получим заданную надпись с ошибкой. Конечно же мы можем с вами добавить другие методы в логическую конструкцию.

Но все таки я сделал её очень простой и наглядной, чтобы показать как работать с запросами и как они вообще работают. На этом я завершу этот подраздел ну а дальше мы с вами поговорим об использовании технологии Cookies.

9.3 PHP и Cookies

Если кто то подумал, что сейчас вместо того чтобы учиться программировать мы будем выпекать печенье. Нет, сейчас мы будем изучать такую важную вещь как Cookies. Cookies - это данные полученные от веб-сайт на веб-сервере которые сохраняются на клиенте. Если пользователь снова посетит сайт который был посещен, то куки-данные будут повторно использованы.

Простым примером работы куки(cookies) будет отображение на веб-сайте числа сколько раз был посещен сайт определенным пользователем. Данные которые были записаны в cookies, как уже говорилось сохраняются на клиенте, и сохраняются они в специальных cookies-файлах. Но кроме файлов куки могут храниться и в самом браузере.

В основном куки используют для того чтобы хранить такую информацию как время посещения веб-сайта, информация об местонахождении и т.п. Конечно же в куки файлах можно хранить любые данные, только главное чтобы все это было в безопасности и от этого была какая то польза. Куки можно отключить в браузере, но это по желанию пользователя который посещает определенный веб-сайт.

Хорошо, а теперь после того как я рассказал вам теорию пора вам рассказать об том как можно использовать куки на практике. Для создания куки-данных в PHP есть функция `setcookie()`, которая имеет 6 параметров:

- `name` - имя нашего куки файла(обязательный параметр)
- `value` - значение куки(переменной)
- `expire` - срок “годности” куки, то есть время сколько будет существовать ещё “годная” куки до того как закроется браузер.
- `path` - путь к куки-файлу
- `domain` - домен с которого был установлен куки.

- `secure` – защищенная передача данных через HTTPS протокол.

Пример создания куки:

```
<?php
setcookie("Test", "cookie");
setcookie("cookie_time", "value", time()+3600);
// Эта куки "проживет" на протяжении часа, это можно
заметить посмотрев на функцию time() и на то какое
значение оно получает
?>
```

Если нам надо будет узнать значение определенной куки, то мы должны будем обратиться к индексам супер-переменной `$_COOKIE`, указав в названии индекса название куки к которому мы хотим обратиться:

```
<?php
setcookie("Text", "supercookie");
echo $_COOKIE["Text"]; // Здесь мы обращаемся к
куки в которого название Text, и в результате мы
получим значение supercookie
?>
```

А чтобы удалить куки, достаточно указать название куки и очистить её значение:

```
<?php
setcookie("Text", "");
?>
```

Конечно же это не все функции которые существуют для того чтобы работать с куки. А также я не показал что можно ещё делать с данными куки-файлов. Раньше в начале 2000-х годов некоторые веб-сайты для реализации системы регистрации и авторизации использовали куки-файлы для сохранения в них логина и пароля пользователя. Это конечно же было очень не надежной системой и тем более небезопасной.

Например, если кто то использовал такой веб-сайт, и он зашёл в свой профайл а потом вышел из него, но не удалил куки-файлы, то после этого любой мог проникнуть в куки-файлы которые хранились на данном компьютере и мог найти логин и пароль пользователя. Кроме этого злоумышленник мог взять и кучу других данных которые он мог использовать в своих целях.

Сейчас, конечно же, для создания системы регистрации и авторизации используют другую технологию – сессии. Об этой технологии мы поговорим позже, но скажу то что она намного надежнее куки-файлов если веб-сайт имеет системы регистрации и авторизации.

Я вам уже показал как можно работать с куки-данными, на этом я завершу этот подраздел и будем двигаться дальше изучая глубже материал и делая больше практических примеров.

9.4 \$_SERVER

`$_SERVER` – это одна из супер-переменных в PHP, вы уже встречались с ней и знаете что её можно использовать для того чтобы узнать какие-то данные об запросе. Вообще это большой массив внутри которого хранятся данные о HTTP путях, заголовках запросов и размещение файлов. Внизу показан список параметров в массиве:

- `GATEWAY_INTERFACE` – информация об версии CGI, которую использует веб-сервер.
- `SERVER_NAME` – имя сервера, под которым выполняется текущий скрипт.
- `SERVER_SOFTWARE` – идентификатор сервера, который используется при передаче запросов и ответов.
- `SERVER_PROTOCOL` – протокол который использует веб-сервер, например HTTP 1.1.
- `REQUEST_METHOD` – метод который использует веб-сервер для запроса веб-страницы(GET,POST и т.д.)

- `REQUEST_TIME` - время начала запроса на веб-странице.
- `QUERY_STRING` - строка запроса страницы.
- `DOCUMENT_ROOT` - корневая директория внутри которой выполняется текущий скрипт.
- `HTTP_ACCEPT` - содержание заголовка Accept внутри HTTP.
- `HTTP_ACCEPT_CHARSET` - выбор кодировки для веб-страницы(например utf-8).
- `HTTP_ACCEPT_ENCODING` - содержание заголовка Accept-Encoding, если он есть.
- `HTTP_ACCEPT_LANGUAGE` - содержание заголовка accept-language, если он есть.
- `HTTP_CONNECTION` - содержание заголовка connection.
- `HTTP_HOST` - Хост в HTTP заголовке.
- `HTTP_REFERER` - адрес страницы которая ссылается на текущий скрипт.
- `HTTP_USER_AGENT` - строка содержащая информацию об клиенте(браузер, операционная система и т.д.).
- `REMOTE_ADDR` - IP адрес удаленного пользователя который обращается к данной странице.
- `REMOTE_HOST` - Хост с которого обращается удаленный пользователь.
- `REMOTE_PORT` - Порт, который используется для соединения с веб-сервером.
- `SCRIPT_FILENAME` - полный путь к текущему скрипту.
- `SERVER_ADMIN` - значение `SERVER_ADMIN` заданное в веб-сервере Apache.
- `SERVER_PORT` - текущий порт для передачи данных веб-сервера по HTTP.

- `SERVER_SIGNATURE` – информация об версии сервера.
- `PATH_TRANSLATED` – основной путь к текущему скрипту.
- `SCRIPT_NAME` – информация об имени скрипта и пути к нему.
- `REQUEST_URI` – URI адрес текущей веб-страницы.
- `PHP_AUTH_DIGESTS` – используется для аутентификации по HTTP.
- `PHP_AUTH_USER` – содержит информацию об имени пользователя авторизованного по аутентификации HTTP.
- `PHP_AUTH_PW` – содержит информацию об пароле пользователя авторизованного по аутентификации HTTP.
- `AUTH_TYPE` – содержит информацию об типе аутентификации по HTTP.

Как видим то здесь есть много параметров которые вы можете использовать во время разработки если вам надо будет узнать какую-то информацию об веб-сервере или HTTP протоколе. На этом я хочу завершить данный подраздел, но это ещё не значит что работа с HTTP и PHP окончена. В этой книге мы ещё не раз будем говорить об том как PHP может работать в сети.

Вопросы и упражнения для самоконтроля

1. Какой протокол используется для передачи данных?
2. Как осуществляется передача данных?
3. Какие есть методы в HTTP?
4. Для чего нужен метод POST и PUT?
5. Что такое COOKIES?
6. Каким способом можно проверить тип запроса в PHP?

10 PHP и Базы данных

После того как вы изучили основы PHP мы можем продолжать двигаться дальше поскольку у вас уже достаточно знаний чтобы понимать что то большее, чем просто разбираться что такое типы данных и т.п. Поэтому с этого раздела мы начинаем разрабатывать свои небольшие веб-приложения которые демонстрируют реальные примеры разработки, и делаем мы это для того чтобы лучше закрепить материал и понять как все работает. А также мы создадим своё родное приложение которое будет иметь функционал как у самой примитивной соц. сети.

Как вы смогли понять, то в этом разделе мы поговорим об том как PHP работает с базами данных. Я не буду вам рассказывать весь этот материал на сухом и скучном техническом языке, а лучше покажу это на простых примерах разработки мини-приложений. Хорошо, а теперь давайте подумаем, я сказал что у мы будем с вами делать приложение которое имеет функционал похожий на самую примитивную соц. сеть.

А значит если у нас будет что то на подобии соц. сети, то значит у нас будет большое количество данных которые нам придется где то хранить. Кроме этого у нас должны будут созданы структуры в базе данных по которым должны будут правильно храниться данные.

Но что же такое база данных? База данных как вы уже смогли понять, это место где можно хранить какие то данные. Давайте я приведу вам простую аналогию в реальной жизни.

Аналогию с базой данных можно провести на примере большого шкафа в котором есть много полок. На полках могут быть разные предметы, полки в свою очередь можно поделить на категории. Например, на одних полках можно хранить книги с жанром приключений, а на других полках можно хранить предметы которые имеют красный цвет. Или можно просто сделать так чтобы на одних полках хранились просто футболки, а на других джинсы.

Надеюсь что такая картина смогла дать вам понять как именно могут храниться данные в базе данных. Но вообще база данных это представление данных в виде таблицы, со столбцами которые в свою очередь имеют свои значения. Базы данных имеют свои таблицы, и именно в таблицах хранятся данные в виде записей. В таблицах есть столбцы которые задают параметры для записей, которые будут добавлены в таблицу.

Например, у нас может столбец с названием записи. То есть, например названием столбца у нас будет `name`. Дальше можем добавить столбец который будет хранить какой-нибудь текст который будет принадлежать записи. И назовём мы этот столбец `data`. В результате после того как мы захотим добавить в новую запись в эту таблицу, то запись будет иметь два поля: `name` и `data`. Более наглядно это можно продемонстрировать на примере простой таблицы:

Name	Data
Название записи будет здесь.	А здесь уже будут данные которые были добавлены в запись.

Програмирование - это круто?	Да! Это очень круто!
------------------------------	----------------------

- *Табл. 15 - Пример таблицы в базе данных*

Как мы видим то тут у нас есть таблица, которая на данный момент хранит в себе две записи имеющая два столбца. То есть когда мы добавляем какую-нибудь запись в таблицу базы данных, запись будет иметь поля которые будут зависят от того какие есть столбцы в таблице.

Но на практике все выглядит по-другому. Во-первых, столбцы могут иметь свои типы данных и такие параметры, как например длина определенного поля в записи. В нашем примере в таблице которую мы просто себе представили как типичную таблицу, которая есть в Microsoft Office Exel, наши столбцы имели текстовые типы данных.

Запись, которая добавляется в таблицу зависит от столбцов таблицы. А вы заметили, что я не дал названия таблице? На практике мы должны будем давать обязательно названия для наших таблиц, если мы хотим к ним обращаться в базе данных. Это логично, поскольку как можно обратиться к человеку, если мы не знаем его имени?

Кроме того что мы должны будем давать названия для таблиц, мы также должны будем дать название и для нашей базы данных внутри которой будут храниться таблицы. Хорошо, а теперь после того как я рассказал вам, как все устроено на простом понятном человеческом языке пора переходить к тому что я должен рассказать какие средства мы будет использовать для создания баз данных и их редактирования.

В начале книги я вам говорил об таком типе баз данных как MySQL. Этот тип БД устанавливается автоматически вместе с локальным сервером XAMPP. Кроме того что мы будем использовать этот тип баз данных, мы будем также использовать Систему-Управления-Баз-Данных(СУБД) - phpMyAdmin.

Приложение phpMyAdmin можно найти по адресу `localhost/phpmyadmin`. Данное приложение дает полноценный

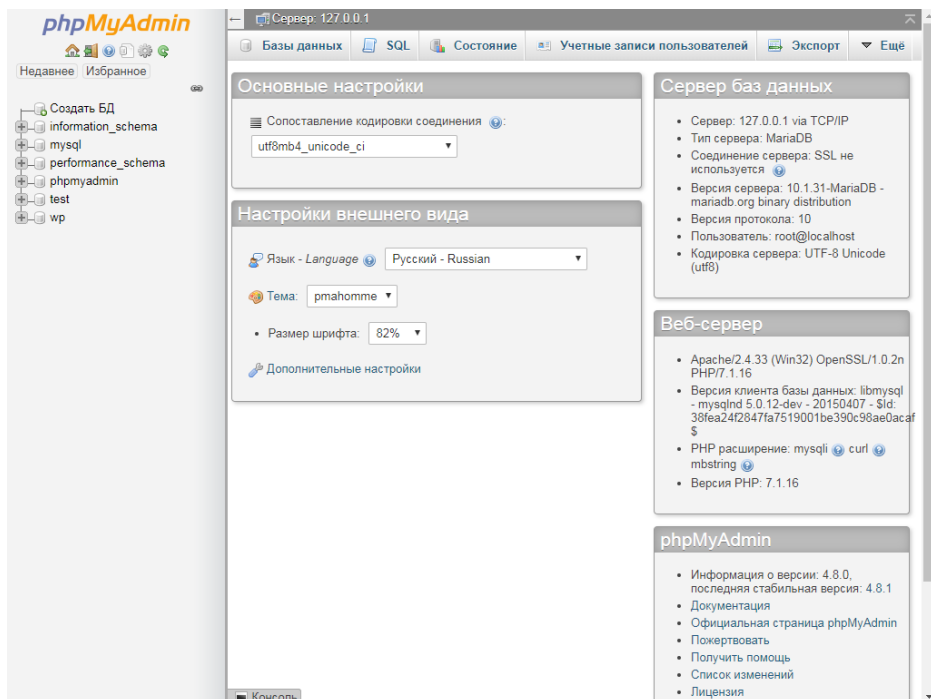
комфортный интерфейс для работы с базами данных, а также с таблицами. Создавать таблицы мы будем с помощью языка программирования SQL, который предназначен для работы с БД и таблицами. Начнется этот раздел с того что мы ознакомимся с MySQL, потом мы больше углубимся в мир программирования баз данных.

10.1 MySQL

MySQL был создан в 1995 годом гигантом корпорацией Oracle, которая на данный момент создала очень много полезного и много пользовательского программного обеспечения, которое используются во всем мире. Но кроме Oracle над этим проектом работала также компания Sun Microsystems. Изначально MySQL это была попытка повышения производительности работы БД(баз данных). И должен сказать, что эта попытка была довольно таки успешной.

MySQL используют во всем мире, многие ИТ-компании используют эту систему. Поскольку она очень эффективная и продуктивная а также легка в использовании. Хорошо, а теперь давайте поговорим об том какую СУБД(Система-Управления-Баз-Данных) мы будем использовать для того чтобы работать с базами данных.

Когда мы устанавливали XAMPP то мы автоматически установили вместе с MySQL одну из самых популярных СУБД – phpMyAdmin. Эта система создана на двух языках программирования PHP и JavaScript. Но суть заключается в том что данная СУБД позволяет легко через графический интерфейс управлять базами данных, а также данными которые внутри их. Перейти в phpMyAdmin можно по адресу в браузере: `localhost/phpmyadmin`. На изображении которое показано внизу можете увидеть интерфейс phpMyAdmin:



- Рис. 16 - Главная страница администратора phpMyAdmin

На первый взгляд может показаться, что это очень сложный интерфейс, который имеет много кнопок и также много всяких данных об наших базах данных. Но по-настоящему все очень просто.

В центре страницы вы можете увидеть настройки для phpMyAdmin. В этих настройках можете выбрать язык для приложения, а также дизайн, если вам не очень то и нравится такой серый интерфейс. С левой стороны вы можете увидеть список с базами данных, которые есть на данном веб-сервере. На изображении, которое показано выше, есть мои базы данных, поэтому если у вас их нет, не надо думать, что что-то не так.

С помощью этого меню вы можете перейти на страницу для создания баз данных, или же на страницу с данными одной из выбранных баз данных. Сверху страницы есть также меню, которое предназначено для того, чтобы переходить на другие страницы phpMyAdmin.

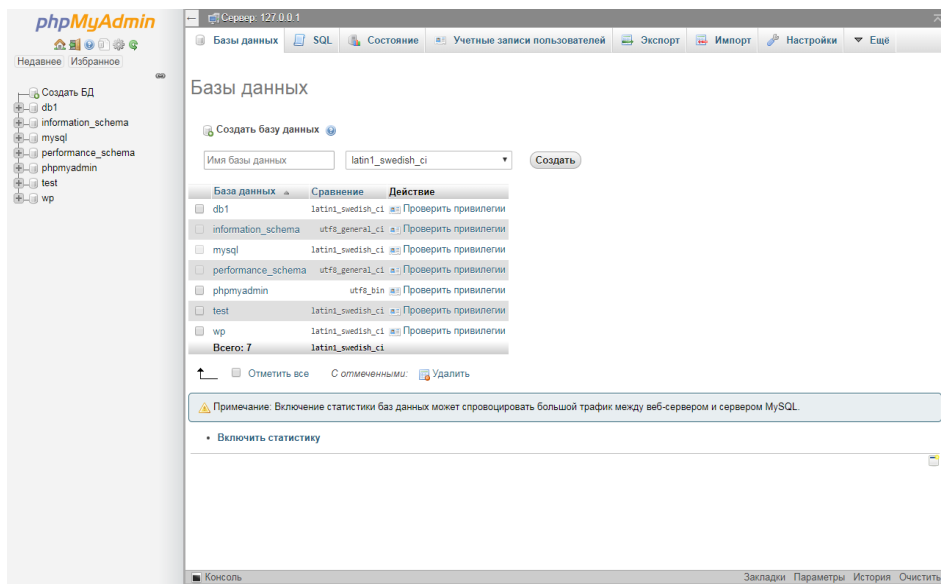
Программировать таблицы, а точнее создавать таблицы для наших баз данных можно с помощью языка программирования запросов SQL. Об этом я уже говорил. Можете заметить, что в верхнем меню есть пункт который как раз и называется SQL, нажав на эту кнопку, вас перенаправит на страницу для создания SQL запросов.

С помощью SQL вы можете создавать базы данных, таблицы баз данных, редактировать базы данных, а также таблицы и сами данные которые внутри их. На данный момент полностью рассматривать данное меню нету смысла, нам для начала хватит каких-то поверхностных знаний для того чтобы понимать данную систему. Двигаемся дальше. Вы наверное уже заметили что справа страницы есть поля в которых показаны данные об том какой сервер для баз данных мы используем, название сервера, хост и т.д.

Это небольшое поле хорошо использовать для того если вам надо будет узнать какие-то данные об сервере. Например, когда мы будем подключаться к базе данных через PHP, то нам надо будет знать хост сервера к которому мы хотим с вами подключиться. В самом внизу страницы, есть окно консоли для того чтобы работать с MySQL через консоль.

Об этом способе работе с MySQL я ещё расскажу. А сейчас же давайте создадим нашу первую базу данных для нашего небольшого приложения. Для этого перейдите в меню которое слева, давайте перейдем по ссылке “Создать БД”.

После этого нас должно перенаправить на страницу для создания БД:



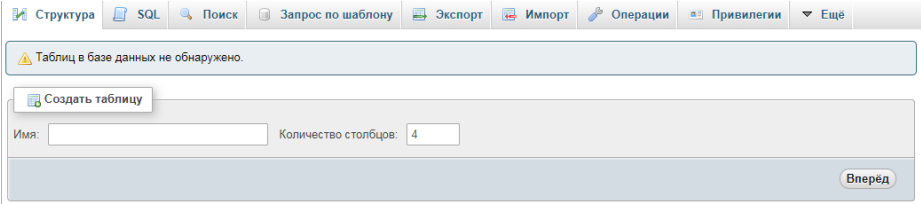
- Рис. 17 - Страница создания и просмотра характеристик баз данных

На этой странице кроме того что вы можете создавать и удалять базы данных вы можете также узнать данные о них. Также здесь есть список баз данных, которое есть на данном веб-сервере. Ну а теперь давайте создадим нашу первую с вами базу данных. Для этого в поле названия базы данных введите желаемое вами имя для вашей базы данных.

Я назову свою базу test, вы конечно же можете подобрать другое название. Но название БД не должно состоять из символов русского алфавита, и в начале не должно быть пробелов или каких-то других символов как например: @!-+;,'" и т.п.

После создания базы данных вы сможете начать работать с ней, то есть редактировать таблицы и данные которые внутри таблиц. А теперь давайте создадим нашу первую таблицу. Для этого перейдите в

созданную вами базу данных в левом меню. После этого вы попадете в интерфейс для просмотра структуры базы данных и доступных таблиц. Поскольку в вашей базе данных нету ещё таблиц, то вы должны будете увидеть такое сообщение:



- *Рис. 18 - Интерфейс создания таблицы для БД в phpMyAdmin*

Есть несколько способов создать таблицы. Например, таблицы можно создать через окно которое показано на изображении выше. Но можно также и через SQL запрос. Поскольку мы изучаем с вами сейчас язык программирования PHP, то знания о том как строить запросы через SQL понадобятся. А все потому что когда мы будем с вами работать с базами данных через PHP, то у нас не будет какого то интерфейса для создания записей в таблицах и т.д. И поэтому единственным способом работы с базами данных через PHP будет использование SQL-запросов.

Поэтому, для начала мы создадим нашу первую таблицу программным путем. Нам надо понять синтаксис SQL, без понимания синтаксиса мы просто не сможем понять для чего нужны все эти символы, которые мы используем во время создания запроса. Наша первая таблица будет называться `test`, на данный момент мы создадим только один столбец:

```
CREATE TABLE test (
    name varchar(20) NOT NULL
);
```

Хорошо, а теперь давайте рассмотрим что и как работает. Во-первых, для создания таблицы мы использовали команду `CREATE TABLE`. После этого мы указали имя таблицы `test`. Но команда

CREATE не только предназначена для создания таблицы, но также для создания базы данных. Для этого надо просто заменить аргумент TABLE на DATABASE:

```
CREATE DATABASE test;
```

Но давайте все-таки вернемся к теме о том как мы с вами создали нашу таблицу. После команды создания таблицы и названия таблицы мы открыли дужки внутри которых мы и начали прописывать параметры таблицы. То есть создавать столбцы для нашей таблицы. Чтобы создать столбец сначала мы указываем имя столбца, после этого мы указываем тип данных для столбца и в конце мы указываем также параметры заполнения столбца.

В нашем примере мы назвали наш первый столбец name, для столбца мы указали строковый(текстовый) тип данных varchar, а внутри дужек мы указали максимальную длину символов которые могут быть в созданной записи столбца. И в конце дали параметр NOT NULL, что значит что данный столбец не может быть пустым. То есть значение столбца не должно равняться NULL. Теперь вы уже немного понимаете синтаксис и как выглядит создание таблицы. Поэтому давайте сделаем таблицу для списка пользователей на веб-сайте:

```
CREATE TABLE my_posts(  
    id int(11) AUTO_INCREMENT,  
    username varchar(20) NOT NULL,  
    password varchar(20) NOT NULL,  
    PRIMARY KEY(id)  
);
```

Мы сделали маленькую таблицу с тремя столбцами, и на этот раз она выглядит уже намного интереснее и сложнее. И это значит что можно будет уже больше взаимодействовать с такой таблицей. Нашими тремя столбцами стали такие столбца как идентификатор пользователя, логин пользователя и пароль пользователя.

Выглядит просто, но по-настоящему таблицы для хранения данных пользователей выглядят по-другому, и имеет другие параметры. А также для таких таблиц в которых хранятся данные об пользователях используется шифрование. Но об безопасности у нас будет отдельный раздел. А теперь посмотрите на первый столбец. Здесь мы поставили числовой тип данных `INT` с максимальной длиной в 11 символов. Заметьте параметр `AUTO_INCREMENT`, здесь вы можете увидеть знакомое вам слово инкремент.

Объясню вам что это за параметр который мы только что с вами поставили. Как можно понять из названия это автоматический инкремент. То есть вместо того чтобы самому вручную добавлять снова и снова идентификатор пользователя увеличивая его на единицу, мы просто воспользуемся автоматическим инкрементом. То есть теперь, когда в эту таблицу будет добавляться новая запись, то её идентификатор будет увеличиваться на единицу.

Следующее два столбцы рассматривать нету смысла, поскольку вам уже и так понятно какой в них тип данных и параметры. Переходим к последнему элементу в запросе. Это команда(`PRIMARY KEY`) для указания ключа для таблицы, в нашем случае это идентификатор пользователя `id`.

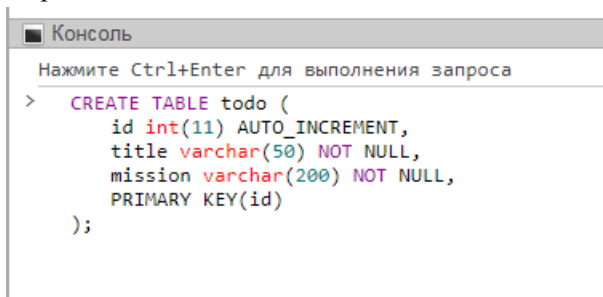
А теперь выполните этот запрос и теперь вы сможете увидеть таблицу в БД. На этом я завершу этот подраздел, у вас есть теперь основы понимания того что такое таблица и как её можно создавать в базе данных. Следующий подраздел будет посвящен более глубокому изучению SQL и таблиц в базах данных.

10.2 Таблицы в SQL

В предыдущем разделе я как раз и сказал что данный раздел будет посвящен более глубокому изучению таблиц в SQL. Сейчас мы будем работать с вами с SQL через встроенную консоль в `phpMyAdmin`. Разницы между запросами которые создаются через страницу создания

запросов, и через консоль - нету. Все так же само, просто методы создания другие.

В встроенной консоли phpMyAdmin вы можете нажимать спокойно кнопку enter не беспокоясь о то что вы запустите ту или иную команду. А если нам надо будет выполнить запрос, то надо нажать на две клавиши `ctrl+enter`. Сейчас я перешел через меню в базу данных `test`, и сейчас я сделаю запрос на создание таблицы для списка заданий:



```
Консоль
Нажмите Ctrl+Enter для выполнения запроса
> CREATE TABLE todo (
    id int(11) AUTO_INCREMENT,
    title varchar(50) NOT NULL,
    mission varchar(200) NOT NULL,
    PRIMARY KEY(id)
);
```

- *Рис. 20 - Создание таблицы в phpMyAdmin с помощью встроенной консоли*

Использовать эту таблицу мы конечно же не будем, но использовать для примера это хорошая идея. Как мы видим то в этой таблице нету никаких новых параметров. Это больше напоминает нам ту таблицу, которая была показана в предыдущем примере.

Выполните данный запрос, и в результате вы увидите то же самое сообщение, что вы видели когда создавали таблицу через страницу создания таблиц на SQL. Мы уже умеем создавать сами таблицы, но как насчёт того чтобы создать какие то записи в таблице? Для этого есть команда `INSERT`, и давайте рассмотрим пример, который показанный ниже:

```
INSERT INTO todo (title, mission) VALUES
('Learn PHP', 'I need to learn.');
```

Как вы видите, то здесь мы добавляем новую запись в таблицу `todo`. Чтобы лучше понять что мы сейчас делаем, выполните данный

запрос и перейдите в эту таблицу в базе данных. В результате вы сможете увидеть, что у нас теперь есть одна запись в таблице.

Ну а теперь вернемся к коде запроса который мы только что выполнили. Сначала мы указываем, в какую именно таблицу мы хотим добавить запись, и делаем мы это с помощью команды `INSERT` и предложения `INTO`, после чего указываем имя таблицы, к которой мы хотим применить операцию.

Важно понимать, что такое предложение в SQL. Предложение это не команда, предложением можно больше назвать дополнением для команды которое дает больше возможностей для создания более динамического запроса. Ну а что идет дальше? Дальше в дужках мы указываем какие поля будет иметь запись. В нашем запросе это `title` и `learn`. После этого идет предложение `VALUES` которое указывает какие именно данные мы хотим занести в указанные нами поля. В дужках мы указали краткие строковые данные.

Можете попробовать сами создать какую запись в таблице. Только главное чтобы столбцы и параметры таблицы были указаны правильно. А как насчёт того чтобы попробовать извлечь данные из какой то записи из таблицы? Для этого есть команда `SELECT`, после команды мы указываем с каких полей записей хотим взять данные, и после этого указываем предложение `FROM` для того чтобы указать из какой таблицы мы хотим взять эти данные. Давайте рассмотрим небольшой пример:

```
SELECT * FROM todo;
```

Только что мы сделали запрос на то чтобы взять **все** данные из таблицы `todo`. А указанием того что мы хотим взять данные со всех полей столбцов стал оператор `*`. Если мы хотим с вами взять какой то определенный столбец то достаточно ввести после команды `SELECT` название нужного нам столбца:

```
SELECT title FROM todo;
```

Но снова таки, здесь уже будут взяты все поля со столбца `title`. Если мы хотим только один чётко определенный объект то надо задать

условие поиска с помощью предложения WHERE, задав условие для поиска:

```
SELECT title FROM todo WHERE id = 1;
```

Выполнив этот запрос, мы получим поле title, id которого имеет значение равное 1. Или можно взять все элементы по id, задав такое условие поиска:

```
SELECT * FROM todo WHERE id = 1;
```

Как вы смогли понять то команда WHERE нужна для того чтобы задать условие поиска. Задав условие поиска, мы можем динамично искать любые данные в таблице. Можно брать не только один элемент, но и несколько, как показано это в этом примере:

```
SELECT id,title FROM todo WHERE title = "Hello";
```

Я решил вам показать таблицу в которой показаны самые востребованные команды в SQL:

Команда	Действие
INSERT	Создание записи в таблице.
CREATE	Создание БД/таблицы.
SELECT	Извлечение данных в таблице.
UPDATE	Обновить данные в записи.
DELETE	Удалить данные в записи.
RENAME	Переименовать таблицу.
ALTER	Модификация столбца.

DESCRIBE	Описание таблицы.
----------	-------------------

- *Табл. 16 - Самые востребованные команды в SQL*

Команда INSERT

Простым примером манипуляции записями таблиц может быть команда INSERT, с помощью которой вы можете добавлять данные в БД и в таблицу:

```
INSERT INTO my_posts (title, data, author)
VALUES ('PHP Junior Kit', 'It's nice book dude.',
'Demian Kostelny');
```

После аргумента INTO идут столбцы, в которые мы хотим добавить данные, после чего идет ещё предложение VARIABLES, которое указывает какие именно данные надо добавить. Можете сделать тестовый запрос в таблице, не обязательно указывать те же самые данные как у меня. Можете написать что то своё, только главное чтобы все было совместно со столбцами таблицы.

Команда DELETE

Хорошо, мы умеем создавать и извлекать данные, а как насчёт удаления данных? Для этого есть команда DELETE которая используется для удаления указанных нами данных. Например, вот как выглядит запрос удаления записи в таблице где значение столбца author равняется Mike Freeman:

```
DELETE * FROM my_posts WHERE author = 'Mike
Freeman';
```

Заметьте, что в нашем примере мы удаляем все записи в таблице, где автором записей является Mike Freeman.

Команда UPDATE

А что делать если нам надо обновить столбцы в таблице? Для этого в SQL есть команда UPDATE:

```
UPDATE [таблица] SET [что именно изменить]
WHERE [условие поиска];
```

Небольшой пример, где показана таблица в которой сохраняются контактные данные, и где производится изменение данных номер телефона пользователя Gordon Freeman:

```
UPDATE contacts SET phone = '85983921', email =
'gordon.freeman@gordonfree.com' WHERE name =
'Gordon', last_name = 'Freeman';
```

Обновление понадобится тогда когда пользователь захочет изменить какие-то данные об себе. У нас также будет отдельный PHP пример с формой которая используется для изменения данных пользователем.

Команда ALTER

Если нам понадобится сделать какую-нибудь модификацию столбцов в определенной таблице, то воспользуйтесь командой ALTER. Команда может добавлять/изменять/удалить столбец. Пример добавления нового столбца в таблицу:

```
ALTER my_table ADD note varchar(200);
```

В примере в таблицу `my_table` мы добавили столбец с названием `note` и текстовым типом данных `varchar`. Или ещё вот пример переименования столбца:

```
ALTER my_table RENAME COLUMN note TO
super_note;
```

Или вот пример удаления столбца:

```
ALTER my_table DROP note;
```

Ну или если нам надо будет заменить тип данных есть такая команда:

```
ALTER my_table MODIFY super_note text(500);
```

Но это ещё не все то, на что способна команда ALTER. Можно выключить PRIMARY KEY таблицы:


```
ALTER my_table DISABLE PRIMARY KEY;
```

Ну и конечно же включить:

```
ALTER my_table ENABLE PRIMARY KEY;
```

На этой ноте я завершаю этот подраздел, и мы углубляемся дальше в мир программирования баз данных.

10.3 Типы данных в SQL

Я показал вам только 3 типа данных SQL, но по-настоящему их намного больше. Все типы данных делятся на разные группы. Например, есть группа с строковыми данными т.е. текстовыми. Внизу показана таблица с группами типов данных:

Название типа данных	Типы данных в группе
Целые числа	int; tinyint; bigint; smallint; bit; numeric; decimal; money; smallmoney;
Приблизительные числа	float; real;
Дата и время	date; time; datetime; smalldatetime;datetimeof fset;
Строковые	char; text; varchar;

- *Табл. 17 - Числовая группа типов данных в SQL*

Дальше мы будем рассматривать каждую группу по отдельности и типы данных которое есть в определенной группе. Сначала мы рассмотрим группы с целыми числами. Также я хотел сказать что в таблице я указал не все типы данных, я сделал это для того чтобы облегчить материал, поскольку их много, а знать все не обязательно. Главное знать самые востребованные для работы.

Целые числа

Вы уже знаете как оперировать с числами в PHP, в SQL работают также само как и те числа с которыми мы встречаемся каждый день. В нижней таблице показано описание типов данных в этой группе:

Тип данных	Длина	Память
Bigint	от -2^{63} (-9 223 372 036 854 775 808) до $2^{63}-1$ (9 223 372 036 854 775 807)	8 байт
Int	от -2^{31} (-2 147 483 648) до $2^{31}-1$ (2 147 483 647)	4 байта
smallint	от -2^{15} (-32 768) до $2^{15}-1$ (32 767)	2 байта
tinyint	от 0 до 255	1 байт

- Табл. 17 - Группа целых чисел в SQL

Какой-то большой разницы между этими типами данных нету. Только они имеют разную максимальную длину и разный размер памяти. Ну и вот небольшой пример с числовым типом данных:

```
CREATE TABLE data (  
    ...  
    cool bigint(1000) NOT NULL  
);
```

Чтобы указать в таблице, какой именно тип будет использоваться со знаком или без знака есть команда метод UNSIGNED:

```
CREATE TABLE data (  
    cool bigint(600) UNSIGNED
```

```
);
```

После выполнения этого запроса создаться таблица, в котором будет поле `cool` с большим числовым типом данных `bigint`, поле сможет иметь положительный или отрицательный знак. В заключение о группе целых чисел скажу что использование этих типов данных зависит от ситуации когда нам надо будет использовать их. Например, если нам надо будет создать поле в котором будет очень большая цифра которая не поддерживается другими числовыми типами данных.

Приблизительные числа

Об приблизительных числах мало что можно сказать. Но использование приблизительных чисел нужно для того чтобы показать числа которые имеют очень большую длину после точки. Об этом я уже говорил в разделе об числах в PHP. Пример в котором создается таблица с столбцом `avog` с числом Авогардо:

```
CREATE TABLE science ( avog float(6) NOT NULL );
```

Дата и время

Мы уже проходили с вами тему о дате и времени только в языке программирования в PHP. Общее значение времени в SQL и PHP почти такое же самое. В SQL дата и время, как и PHP имеет свой формат и диапазон. Посмотрите на таблицу показанную ниже:

Тип данных	Численность	Диапазон
Time	чч:мм:сс [.nnnnnn]	От 00:00:00.000000 до 23:59:59
Date	ГГГГ-ММ-ДД	От 0001-01-01 до 31.12.99

smalldate	ГГГГ-ММ-ДД чч:мм:сс	От 01.01.1900 до 06.06.2079
datetime	ГГГГ-ММ-ДД чч:мм:сс [.ннн]	От 01.0.1753 до 31.12.1999
datetime2	ГГГГ-ММ-ДД чч:мм:сс [.ннннннн]	От 0001-01-01 00:00:00.000000 до 9999-12.31 23:59:59.999999
Datetimeoffset	ГГГГ-ММ-ДД чч:мм:сс [.ннннннн] (в офрмате UTC)	От 0001-01-01 00:00:00.000000 до 9999-12.31 23:59:59.999999 (в фомате UTC)

- Табл. 18 - Группа типов данных даты и времени в SQL

Внимательно рассмотрев таблицу, можно заметить какая есть разница между этими двумя типами данных. Но кроме все этого есть куча функций, которые могут иметь эти типы данных. Вот пример представления данных даты с помощью типа данных date:

```
CREATE TABLE dates (
    example_date DATE NOT NULL
);
```

А сейчас давайте сделаем модификацию нашей таблицы с постами, добавив столбец с типом данных date:

```
ALTER my_posts ADD creating_date date;
```

Ну и небольшой пример как можно доставать данные из таблицы в зависимости от времени даты:

```
SELECT * FROM my_posts WHERE creating_date =
"2018-06-06";
```

Строковые данные

Вы уже знаете два типа данных из этой группы типов данных, это `text` и `varchar`. Посмотрите на таблицу под номером 19 чтобы лучше понять характеристику этих типов данных:

Тип данных	Длина
Char	от 1 до 8000
Varchar	от 1 до 8000
Text	от 1 до 2 147 483 647

- Табл. 19 - Группа строковых типов данных в *SQL*

Примеры приводить не буду, в разделе о таблицах вы уже насмотрелись примеров с `varchar` и `text`. Разве что скажу, для чего нужен тип данных `char`. `char` - может иметь в себе один символ, и так наиболее часто бывает, что этот тип данных используют только для одного символа. Например, в языке программирования *C#* есть этот тип данных, и вместить в себе он может только один символ.

10.4 Операторы

Для создания более гибких и динамических запросов к БД надо использовать операторы. Например, вот пример того как буду извлекаться данные из таблицы где идентификатор равен больше чем 5:

```
SELECT * FROM todo WHERE id > 5;
```

С этим оператором вы уже встречались в подразделе об числовых операторах. Посмотрите на таблицу ниже, в которой я привел математические операторы для SQL-запросов:

Оператор	Значение
=	Равно
>	Больше чем
<	Меньше чем
>=	Больше или равно
<=	Меньше или равно
<>	Не равно

- Табл. 20 - Математические операторы в SQL

Работа этих операторов такая же самая как и в PHP. Долго задерживаться не будем, поэтому перейдем к булевым операторам.

Булевы операции

А если нам надо будет извлечь не только данные где идентификатор больше за число 5 но и title равен строке "Hello world!". Вот пример:

```
SELECT * FROM todo WHERE id > 5 AND title = "Hello world!";
```

Так уже больше интересно. Можно сделать запросы динамичнее добавив ещё операторов:

Оператор	Значение
AND	Логическое "И"
OR	Логическое "ИЛИ"

NOT	Логическое "НЕ"
-----	-----------------

- Табл. 21 - Логические операторы в SQL

Давайте для лучшего закрепления материала рассмотрим такой пример:

```
SELECT * FROM todo WHERE id > 5 OR NOT title = "Super todolist";
```

Специальные операторы

Кроме тех операторов, которые были рассмотрены выше есть и специальные операторы. Это IN, BETWEEN, LIKE и IS NULL. Как и логические операторы, эти специальные операторы делают запросы более динамическими и расширенными.

Первый оператор, который мы рассмотрим это будет оператор IN. Так что же из себя представляет этот оператор. Оператор IN это набор со значениями которые мы указываем, он проверяет есть ли в таблице заданные значения того или иного объекта.

Вот пример простого запроса с категориями:

```
SELECT * FROM todo WHERE category IN ('Science', 'Blockchain', 'Technologies');
```

Описать этот запрос на простом понятном для человека языке можно так: запрос берёт все элементы в которых значение категории может быть или Science, или Blockchain, или Technologies. Надеюсь, вы поняли работу этого оператора. Двигаемся дальше и сейчас мы узнаем об операторе BETWEEN. Этот оператор работает очень похоже на оператор IN, но только уже здесь устанавливается диапазон значений. Например:

```
SELECT * FROM todo WHERE id BETWEEN 10 AND 20;
```

Запрос извлечет элементы, в которых идентификатор в диапазоне от 10 к 20. Или вот ещё пример с буквами:

```
SELECT * FROM todo WHERE category BETWEEN "A"
AND "C";
```

Следующий оператор это LIKE. Он ищет в указанном столбце указанные в условии подстроки. Рассмотрим простой пример:

```
SELECT * FROM todo WHERE title LIKE "A";
```

После выполнения этого запроса из таблицы todo будут извлечены данные которые в которых title начинается на букву А. Но важно помнить что данный оператор работает только со **строковыми данными**. Если нам понадобится найти значения в которых будут значения NULL. То надо использовать оператор IS NULL:

```
SELECT * FROM todo WHERE id IS NULL;
```

Мы не получим никаких данных из поля id поскольку они будут пусты. На этом я завершаю этот подраздел об операторах в SQL.

Заключение об операторах

В заключение об операторах в SQL скажу, что они позволяют создавать более гибкие и динамические запросы, что позволяет расширить круг возможностей.

10.5 Агрегатные функции

Для того чтобы взаимодействовать с группой полей в таблице надо использовать агрегатные функции. В таблице, которая находится ниже можно посмотреть список агрегатных функций:

Функция	Действие
COUNT	Выдаёт количество полей в таблице.
SUM	Выдаёт сумму выбранных полей.

AVG	Выдаёт средние значение выбранных полей.
MAX	Выдаёт наибольшее значение выбранных полей.
MIN	Выдаёт наименьшее значение выбранных полей.

- *Табл. 22 - Агрегатные функции*

Вопрос, как использовать агрегатные функции в SQL? Например, вот запрос с командой `SELECT` где после команды `SELECT` я использую агрегатную функцию:

```
SELECT SUM(id) FROM todo;
```

Это была функция `SUM`. И если выполнить такой запрос, то мы получим сумму всех `id` в таблице `todo`. Можно уже считать, что мы рассмотрели функцию `SUM`, а как насчёт функции `COUNT`?

Как было сказано то данная функция возвращает значение количества выбранных полей в таблице:

```
SELECT COUNT(title) FROM todo;
```

Результатом этого запроса будет значение количества полей столбца `title`. Эту команду полезно использовать, если нам надо получить количество пользователей.

Функция GROUP BY

Функция `GROUP BY` используется в команде `SELECT` для агрегации записей выбранных в таблице. Например можно сделать такой запрос с такой конструкцией:

```
SELECT title, year FROM posts GROUP BY year;
```

После выполнения этого запроса результаты будут погруппированы по столбцу `year`. То есть результат будет показан по

возрастанию значения года. Само название команды переводиться как - группировать по.

Говорить об других агрегатных функциях нету смысла. Почему? Поскольку я уже вам показал, как можно использовать агрегатные функции, а также с некоторыми вы уже встречались в PHP, только они использовались немного по-другому. Например есть агрегатная функция MAX и в разделе о функциях в под разделе об функциях для работы с числовыми данными вы встречались с функцией max () которая работает также само как и та самая агрегатная функция.

На этом я завершаю этот подраздел об агрегатных функциях.

10.6 Транзакции

Транзакция это единица работы БД. Если транзакция была выполнена успешно, то модификация данных, сделанная во время транзакции становится частью базы данных. Но если транзакция прошла не успешно и получилась ошибка, то будет сделан откат данных.

Простым примером транзакции может быть то как мы делаем запрос на создание таблицы и мы получаем успешный результат что таблица создана. Это успешная транзакция как вы смогли уже понять. Для того чтобы можно было работать с транзакциями в SQL есть функции которые я привел ниже:

Функция	Действие
BEGIN	Начало транзакции.
COMMIT	Внесение изменений.
ROLLBACK	"Откат" внесенных изменений.

- Табл. 23 - Команды для транзакций

Команды транзакций в действии

Как уже было сказано, эта команда задает начало транзакции. Вот пример запроса который отправляет транзакцию MySQL:

```
BEGIN;  
INSERT INTO todo (title,author) VALUES ('Super  
title', 'Mr. Freeman');  
COMMIT;  
SELECT * FROM todo;
```

После выполнения такого запроса вы получите результат вывода всех записей с помощью команды COMMIT. Команда COMMIT отмечает успешное завершение транзакции, и если надо можно ввести команду которая выполниться после завершения транзакции, как и было показано в нашем примере.

Дальше у нас идет последняя команда и это команда ROLLBACK которая как было описано выше используется для того чтобы сделать “откат” внесенных изменений транзакцией. Команда может делать любой “откат” к любым изменениям транзакций. Важно также сказать, что транзакциям можно давать названия. Вот пример запроса с использованием ROLLBACK:

```
BEGIN TRAN @my_tran:  
INSERT INTO users (username) VALUES  
( 'Superuser' );  
COMMIT;  
SELECT * FROM users;  
ROLLBACK TRAN @my_tran;
```

Как мы видим, то в этом примере мы задали имя для нашей транзакции после чего мы с помощью команды ROLLBACK сделали откат изменений.

10.7 Резервное копирование базы данных

Данные, которые будут храниться в вашей базе данных могут становить какую то ценность. И поэтому чтобы не потерять их если могут случиться какие то события которые могут навредить данным в вашей БД, надо делать резервные копии БД. Для этого в MySQL есть команда `mysqldump`:

```
mysqldump -u пользователь -pпароль база_данных
```

Как вы смогли понять, то здесь мы вводим имя и пароль пользователя `mysql`. Например, если у нас локальный сервер то запрос с этой командой выглядел бы вот так:

```
mysqldump -u root -p test
```

После выполнения такого запроса в консоли мы получим вывод данных на экран, которые есть в базе данных. Но, а если мы хотим выгрузить данные в файл то нам уже надо выполнить такую команду:

```
mysqldump -u пользователь -pпароль test > test.sql
```

Выполнив эту команду, данные из БД `test` будут выгружены в файл `todo.sql`. Если заглянуть в этот файл то можно будет увидеть код запроса для создания таблиц. Но если нам понадобится создать копию структуры отдельной таблицы то перед именем базы данных надо будет ввести название таблицы:

```
mysqldump -u пользователь -pпароль test todo > todo.sql
```

А чтобы сделать копию структуры всех баз данных надо ввести вместо названия базы данных команду `--all-databases`:

```
mysqldump -u пользователь -pпароль --all-databases > all_db.sql
```

Хорошо, ну а что делать, если у нас есть копия БД и мы хотим её загрузить на сервер баз данных? Для этого нам просто надо будет

воспользоваться командой `mysql`, например загрузка всех баз данных на сервер:

```
mysql -u пользователь -ппароль < all_db.sql
```

Но если нам надо будет выгрузить только одну базу данных то тогда нам надо будет воспользоваться ключом `D`:

```
mysql -u пользователь -ппароль -D test < test.sql
```

Или вот ещё пример где показано как можно выгрузить отдельную таблицу:

```
mysql -u пользователь -ппароль test < todo.sql
```

На этом я завершаю раздел об работе с `mysql` в общем, следующий раздел будет об том как можно работать с MySQL через PHP, и конечно же мы будем делать реальные примеры. А сейчас же, как и в каждом разделе я закончу этот запрос для вас вопросами и упражнениями для самоконтроля.

Вопросы и упражнения для самоконтроля

1. Что такое `mysql`?
2. Что такое база данных?
3. Какими способами можно создать БД?
4. Для чего используется SQL?
5. Какие есть способы создания таблицы в БД?
6. Зачем нужна команда `SELECT`?
7. Для чего нужно предложение `AUTO_INCREMENT`?
8. Создайте таблицу `products` которая бы использовалась для хранения покупок. В таблице должны быть столбцы с именем продукта, описанием и его ценой.
9. Как можно использовать команду `INSERT`? Сделайте запрос в свою таблицу с этой командой.

10. Для чего нужно предложение `PRIMARY KEY`?
11. Какие вы знаете типы данных в SQL?
12. Какие есть группы типов данных в SQL?
13. К какой группе принадлежит тип данных `money`?
14. Какие есть типы данных в группе даты и времени?
15. Что такое агрегатные функции?
16. Зачем нужна функция `COUNT`?
17. Создайте таблицу в которой один столбец будет иметь числовые данные. После этого воспользуйтесь командой `INSERT` и создайте 5 полей записей в этом столбце. После чего воспользуйтесь функцией `SUM` что сложить значения всех полей.
18. Что такое транзакции и для чего они нужны?
19. Как задается начал транзакции?
20. Опишите работу команды `COMMIT`?
21. Как можно сделать откат данных?
22. Какая команда используется для резервного копирования данных?
23. Как можно сделать копирование данных только одной таблицы в базе данных?
24. Как можно сделать копирование всех баз данных?
25. Как делается выгрузка данных в базу данных?
26. Как выгрузить одну отдельную таблицу в БД?

11 Доступ к MySQL через PHP

Как вы смогли понять из названия раздела, то этот раздел будет посвящен тому как получить доступ и как работать с MySQL через PHP. В php есть отдельный раздел функций, в котором есть функции с помощью которых можно реализовать доступ к MySQL, а также делать запросы к базам данных и таблицам и многое другое с помощью чего можно работать с MySQL.

В этом разделе будут показаны реальные примеры использования MySQL в PHP, но кроме этого также и где можно использовать то что мы выучили о PHP. Первое что мы должны сделать, это подключиться к самой базе данных. Эти знания нам очень понадобятся когда мы уже будем программировать само наше приложение.

Конечно же, самым простым способом показать все это на реальном примере создания очень простого блога. Блог не будет иметь регистрации, но он сможет прекрасно продемонстрировать то, как будет выглядеть работа с базой данных.

Хорошо, тогда начинаем создавать наш примитивный блог.

11.1 Доступ к MySQL через PHP

Мы не сможем просто взять и осуществить доступ к базе данных через PHP если мы не будем подключены к ней. Первое что мы должны сделать - это подключиться к базе данных. Для этого в PHP есть функция `mysqli_connect`, внутри функции мы должны указать локальный сервер, пользователя, пароль пользователя и имя базы данных к которой мы хотим подключиться.

Например, вот как будет выглядеть подключение к базе данных если мы используем локальный сервер:

```
<?php
    $connect = mysqli_connect("localhost", "root",
"", "test");
?>
```

Как вы смогли заметить, то мы оставили пустым поле с паролем, а все потому что мы используем локальный сервер. Но если у вас другая операционная система, например Linux то может быть другое название пользователя и пароль. А все через то, что при установке MySQL на Linux надо обязательно указать пароль и имя пользователя. Но это конечно же все зависит от того какую вы используете операционную систему.

Вернемся к подключению к базе данных, лучше указывать имя сервера, имя пользователя и т.д. в отдельных переменных, так будет легче читать код:

```
<?php
    $host = "localhost";
    $username = "root";
    $password = "";
    $db = "test";

    $connect = mysqli_connect($host, $username,
$password, $db);
?>
```


Поскольку мы с вами в этом разделе будем создавать блог, то у нас будет отдельный файл внутри которого и будет воспроизводиться подключение к базе данных. Сделаем наш файл более динамическим, добавим проверку на успешное подключение к нашей базе данных:

```
<?php
$host = "localhost";
$username = "root";
$password = "";
$db = "test";

$connect = mysqli_connect($host, $username,
$password, $db);

if ( !$connect ) {
    echo "Подключение не удалось";
    echo "Произошла ошибка:" .
mysqli_connect_errno;
    exit;
}
?>
```

Здесь функция `mysqli_connect_errno()` выводит ошибку если она произошла. Это будет полезно использовать, если мы не сможем подключиться к самой базе данных. Теперь немного представим структуру нашего блога:

```
|
|- classes/
|---db.php
```

Как видно из схемы, то у нас будет папка которая называется `classes`. Внутри папки будут находиться классы, которые мы будем использовать в нашем приложении. Мы рассмотрели способ подключения к базе данных. Но как насчёт построения запросов? Об этом будет в следующем подразделе.

11.2 Создание запросов к MySQL в PHP

Создание запросов через PHP позволяет получать данные с базы данных и обрабатывать их уже в PHP скрипте. После чего эти данные можно отображать на страницах своего приложения. В предыдущем подразделе мы с вами подключились к базе данных. Теперь надо сделать таблицу для постов и тестовый запрос для того чтобы посмотреть работает ли все, вот код нашей таблицы на SQL:

```
CREATE TABLE posts (  
    id int(11) AUTO_INCREMENT,  
    title varchar(20) NOT NULL,  
    data text(350) NOT NULL,  
    author varchar(20) NOT NULL,  
    PRIMARY(id)  
);
```

Создайте в нашей базе данных запрос с этим кодом, и переходим к следующему зданию. Нам надо создать тестовый запрос, и этот запрос будет создавать запись в базе данных с помощью команда INSERT. Для этого нам для начала надо подключить файл с подключением к базе данных, после чего мы обратимся к переменной, с помощью которой мы подключаемся к базе данных. Вот как это выглядит на практике:

```
<?php  
require("classes/db.php");
```

```
$request = $connect->query("INSERT INTO posts  
(title, data, author) VALUES ('Super Test', 'Here is  
data for request', 'Anonymous') ");
```

```
if ($request) {  
    echo "ALL IS WORKING";  
} else {
```

```

        echo "ERROR WITH REQUEST";
    }
    ?>

```

Как вы смогли заметить то здесь мы создаём файл не в папке `classes`, а папке нашего приложения в общем. Назовём этот файл `index.php` после чего перейдём по адресу нашего приложения в браузере и в результате если мы правильно все подключили, мы должны получить ответ об том что запрос был отправлен и всё прошло успешно.

Сообщением которое сообщает об успешном запросе есть строка `"ALL IS WORK"`. Если вы видите тоже самое то тогда запрос был успешным и значит вы можете перейти в БД и посмотреть в таблицу, после чего вы увидите что данные которые были указаны в запросе который создался через PHP-скрипт были получены.

Хорошо, а давайте теперь попробуем извлечь эти данные? Тут уже все будет немного сложнее. После того как мы сделаем запрос, мы должны будем воспользоваться специальной функцией `mysqli_fetch_assoc()` чтобы извлечь данные:

```

<?php
require ("classes/db.php");

$query = "SELECT * FROM posts";
$request = connect->query($query);

if ($request) {
    while ($row =
mysqli_fetch_assoc($request)) {
        echo $row["id"] . $row["title"] .
$row["data"] . $row["author"];
    }
}
?>

```

Как вы смогли здесь заметить, я использую цикл `while` для того чтобы вывести все записи из таблицы. Конечно, можно было сделать так чтобы взять из таблицы только одну запись. Именно переменная `row` содержит массив из данными, и как вы смогли заметить то индексы элементов массива это названия столбцов в таблице.

А функция `mysqli_fetch_assoc()` как раз и используется для того чтобы извлечь данные из заданного запроса. Теперь если вы перейдете на страницу вашего приложения, т.е. `index.php` то вы сможете увидеть вывод записей из вашей таблицы, если вы сделали все правильно.

Конечно же, здесь не будет дизайна. Но зато мы сможем увидеть все записи, которые были в таблице. Кроме цикла `while` можно было также воспользоваться циклом `for`, но нам надо будет подсчитать число записей с помощью функции `num_rows`:

```
<?php
// ...
$rows = $request->num_rows;
for ($i = 0; $i < $rows; $i++) {
    echo "Title:" . $request-
>fetch_assoc()["title"];
    echo "Data:" . $request-
>fetch_assoc()["data"];
    echo "Author:" . $request-
>fetch_assoc()["author"];
}
?>
```

Думаю, вы уже смогли здесь заметить что я использую другую функцию `fetch_assoc` а также что я вызываю её так как метод какого то класса. Это повязано с тем, что переменная `request` получает значение от класса `mysqli` и именно через это мы можем вызвать функции для MySQL через эту переменную с помощью этой стрелочки `->`.

Я просто вам показал, как можно создавать запросы к базе данных с помощью РНР, но когда мы завершим создание запросов нам надо будет закрыть соединение с базой данных, для этого есть функция `close()` :

```
// Конец файла index.php
$request->close();
$connect->close();
?>
```

Как вы можете увидеть, то сначала мы закрыли соединение для запроса, и после чего мы закрыли соединение с базой данных в нашем скрипте. Есть также и второй способ закрыть соединение с БД, это с помощью функции `mysqli_close()` :

```
// Конец файла index.php
mysqli_close($connect);
?>
```

На этом я завершу этот подраздел, и мы перейдем к более практическим задачам и созданию своего примитивного блога с использованием MySQL.

11.3 Создание блога

У нас уже есть таблица для записей блога. А как насчёт других элементов? Как например страницы для создания записи, редактирования и т.д. Для начала мы должны будем создать форму для создания записей после чего сделать скрипт который обрабатывал эту форму.

Начнём с формы, в нашей таблице есть 3 столбца(это если не считать `id`), значит в нашей форме должно быть 3 поля для каждого столбца:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My App</title>
    </head>
```

```

        <body>
            <form action="create.php"
method="post">

                <p>
                    <label>Title: </label>
                    <input type="text" name="title"

/>

                </p>
                <p>
                    <label>Data: </label>
                    <textarea
name="data"></textarea>
                </p>
                <p>
                    <label>Author: </label>
                    <input type="text"
name="author" />
                </p>
                <p>
                    <input type="submit"
value="Создать пост" />
                </p>
            </form>
        </body>
    </html>

```

Вы заметили какие имена мы дали нашим полям с помощью атрибута name, мы уже с вами изучали обработку форм в отдельном подразделе. Двигаемся дальше, и сейчас нам надо будет сделать скрипт для обработки формы. Создадим в корневой папке этот файл, назвав его create-form.php.

Все данные мы будем получать в массиве POST после чего мы их будем распределять по отдельным переменным и уже тогда делать запрос с ними. Код скрипта показан ниже:

```
<?php

require ("classes/db.php");

if (isset($_POST["title"])) { $title =
$_POST["title"]; }
if (isset($_POST["data"])) { $data =
$_POST["data"]; }
if (isset($_POST["author"])) { $author =
$_POST["author"]; } // Делаем проверку если ли в
массиве элементы с такими индексами

if (empty($title)) { exit("Заполните все
поля"); }
else if (empty($data)) { exit("Заполните все
поля"); }
else if (empty($author)) { exit("Заполните все
поля"); }
// Делаем проверку переменных на пустоту.
// Если будут пустыми то тогда вывести
сообщение об ошибке.

$query = "INSERT INTO posts (title, data,
author) VALUES ($title, $data, $author)";
$request = $connect->query($query);

if ($request) {
    echo "Все данные были успешно отправлены";
} else {
```

```
        echo "Простите, но случилась ошибка";  
    }  
  
    ?>
```

Как вы смогли заметить, то сначала мы сделали проверку данных на их существование, после этого мы создали переменные и проверили их на пустоту. И в конце мы сделали запрос, заметьте то как мы сделали передачу переменных в таблицу. Сохраните его в корневой папке назвав его `create.php`.

А теперь это можно протестировать. Перейдите по адресу вашего приложения на файл с формой. У меня например адрес выглядит будет приблизительно вот так `localhost/app/create-form.php`. Заполнив форму, отправьте форму и потом проверьте базу данных сменилось ли что то.

Если вы увидите что в таблице постов появился новый пост и он содержит внутри себя данные, которые вы ввели, то тогда я вас поздравляю! Значит все работает, и вы можете создавать теперь посты в своем маленьком приложении. Сразу скажу, что по настоящему разработка сейчас выглядит намного по-другому, но об этом пойдет речь в конце книги.

Хорошо, двигаемся дальше. Дальше на нас ждет создание функции на редактирование записи. Здесь уже скрипт и форма будут храниться в одном и том же файле. Начнём мы со скрипта, мы должны сделать так чтобы сначала мы получили данные об конкретной записи.

После этого получив данные записи эти данные заносятся в форму и пользователь мог сам уже под редактировать что то сам. И в конце типичный алгоритм с запросом. Метод который будет использоваться в запросе как вы уже смогли догадаться это будет `UPDATE`.

Начинаем, для начала мы должны сделать так чтобы происходил поиск поста по id которое будет введено в адресной строке после вопросительного знака.

Для этого мы воспользуемся супер-переменной `$_GET`:

```
<?php
require ("classes/db.php");

$id = $_GET["id"]; // Достаем id поста из
адресной строки
$query = "SELECT * FROM posts WHERE id = '$id'
";
$request1 = $connect->query($query);

if ($request1) {
    $row = mysqli_fetch_assoc($request1);

    $title = $row["title"];
    $data = $row["data"];
    $author = $row["author"];
?>

<!-- Здесь будет форма -->
<!-- Она появиться только тогда если будет
успешный запрос -->
<!-- И именно внутри формы в значения поля
мы будем вставлять только что созданные переменные --
>

<?php
    } else {
        exit("Простите но случилась
ошибка");
    }
?>
```

Как вы смогли заметить, то здесь есть разделение на часть кода для PHP и на часть код для HTML. Хорошо, а теперь создадим саму форму:

```
<?php
// ...
// Код для получения переменных и т.д....
?>
<form action="update.php" method="post">
    <p>
        <label>Title: </label>
        <input type="text" name="title"
value="<?php echo $title; ?>" />
    </p>
    <p>
        <label>Data: </label>
        <textarea name="data">
            <?php echo $data; ?>
        </textarea>
    </p>
    <p>
        <label>Author: </label>
        <input type="text" name="author"
value="<?php echo $author; ?>"
    </p>
</form>
<?php
    //...
    // Код ошибки в else
?>
```

Вот и сама форма, думаю вы заметили, что с помощью атрибута value будут выводиться значения переменных в полях. И угадайте как будет называться файл если он выполняет функцию формы и

обработчика? `update.php`, вот так он будет называться. Это хорошо заметно по атрибуту `action` в теге формы.

Мы создали только часть кода для того чтобы достать данные об посте из таблицы с помощью указания `id` через адресную строку. Об том как это будет работать поговорим после того как напомним часть кода для обработки формы. В этой части кода мы сначала должны будем сделать проверку на наличие каких то данных, которые мы можем обработать. После этого нам останется только создать запрос к таблице и вывести результат. Вот код:

```
<?php
// ...
// Если что, то это конец нашего файла где есть
код ошибки
if (isset($_POST["title"]) or
isset($_POST["data"])) {
    $title = $_POST["title"];
    $data = $_POST["data"];
    $author = $_POST["author"];
    $query = "UPDATE posts SET title =
'$title', data = '$data', author = '$author' WHERE id
= '$id'";

    $request2 = $connect->query($query);

    if ($request2) {
        echo "Операция прошла успешно, пост
был обновлен";
    } else {
        echo "Простите, но произошла
ошибка";
    }
}
```

?>

Прекрасно, у нас есть теперь код для обновления записи. Сохраните этот файл в корневой папке назвав его `update.php`. После этого перейдите по адресу этого скрипта, **но важно** чтобы вы добавили в конце вопросительный знак и написали переменную `id` которой вы присвоили значение 1.

У меня это выглядит приблизительно вот так `localhost/app/update.php?id=1`. Главное чтобы в таблице был пост с идентификатором, который был равен 1. С помощью супер-переменной `$_GET` мы как раз и получаем значение `id` через адресную строку. Если у вас поля заполнены тем что храниться в таблице в базе данных с таким же идентификатором как и в адресной строке, то тогда это значит что вы сделали все правильно. Главное теперь протестировать работает ли обновление данных. Если запросы работают успешно и данные обновляются, то тогда мы можем двигаться дальше.

Мы должны создать отдельный скрипт, с помощью которого можно было бы просматривать отдельную запись по `id`. То есть делать это с помощью супер-переменной `$_GET` как мы это делали в скрипте для редактирования поста. Это будет конечно же намного легче, поскольку нам надо будет достать только данные какой то одной записи и не надо будет делать запросов на обновление данных. И также нам не придется создавать какие то формы и мы сможем поместить все в один файл. Вот код:

```
<?php
require("classes/db.php");

$id = $_GET["id"];

$query = "SELECT * FROM posts WHERE id =
'$id'";
$request = $connect->query($query);
```

```
if ($request) {  
    while($row = mysqli_fetch_assoc($request))  
    {  
        echo "Title: " . $row["title"];  
        echo "Data: " . $row["data"];  
        echo "Author: " . $row["author"];  
    }  
}  
?>
```

Работать такой код будет только тогда когда будет значение идентификатора, а также пост с соответствующими id будет в базе данных. Последние что нам осталось добавить, так это скрипт удаления записи. Он будет по такому же алгоритму как скрипт для просмотра отдельной записи, то есть находить по идентификатору. Код соответствующего скрипта показан ниже:

```
<?php  
require("classes/db.php");  
  
$id = $_GET["id"];  
$query = "DELETE FROM posts WHERE id = '$id'";  
$request = $connect->query($query);  
  
if ($request) {  
    echo "Запись была успешно удалена";  
    // Можно было также сделать так чтобы  
    // если был успешный запрос, то с  
    // помощью функции require или include  
    // загружался файл внутри которого  
    // было сообщение об успешном запросе  
} else {
```

```
        exit("Произошла ошибка, запрос прошёл  
        неуспешно");  
    }  
  
    ?>
```

Сохраните файл в корневой папке, назвав его `delete.php`, после чего перейдите по адресу данного файла, не забыв при этом добавить в конце вопросительный знак с переменной `id` и желаемым значением идентификатора. Если вы увидите что запрос прошёл успешно, то проверьте таблицу в базе данных и посмотрите была ли удалена запись с заданным вами значением идентификатора.

Если у вас все работает, то тогда это значит что вам удалось сделать уже свой примитивный блог. Да, здесь нету регистрации, нету какого-то большого и сложного функционала. Но за то это дало вам понять как можно работать с MySQL через PHP на практике. Конечно же мы будем делать систему регистрации и логина, но об этом речь пойдет в следующем разделе.

А сейчас же мы будем продолжать делать практические примеры с MySQL и PHP. В этом разделе вы познакомились на практике как можно создавать запросы к MySQL разного типа. Кроме всего этого вы создали свой простой очень примитивный блог. Конечно же сейчас есть намного быстрее и функциональные технологии для создания своего блога. Но создание своего блога помогло вам понять больше об создании запросов. На практике сейчас все выглядит по-другому. Как я уже и говорил об этом речь пойдет в конце книги.

11.4 Загрузка файлов и MySQL

Вообще сам файл загрузить в базу данных нельзя. Но можно же хранить файлы на определенном сервере, а адреса к этим файлам хранить в базе данных? Именно в такой способ сейчас работает загрузка файлов, после чего в базе данных есть их адрес и в нужный момент в приложении

можно сделать так чтобы PHP просто искал адрес файла, и после этого по адресу файла находил сам файл и загружал его.

Для загрузки файла нам конечно же понадобится форма с помощью которой пользователь будет выбирать какой именно файл загружать. Загруженные файлы будут храниться в нашем приложении в папке `files`, после создания формы добавим таблицу в базу данных и в конце скрипт для обработки формы. Начнём с формы:

```
<!DOCTYPE html>
<html>
    <head>
        <title>File upload</title>
    </head>
    <body>
        <form action="upload.php"
method="post">
            <p>
                <input type="hidden"
name="MAX_FILE_SIZE" value="300000" />
                <!-- Поле MAX_FILE_SIZE
служит для указания максимального размера для
загружаемого файла. Поле должно быть указано с самого
начало перед самым полем загрузки. Заметьте также что
данное поле спрятано с помощью значения hidden в
атрибуте type -->
            </p>
            <p>
                <input type="file"
name="my_file" />
            </p>
            <p>
                <input type="submit"
value="Загрузить файл" />
            </p>
        </form>
    </body>
</html>
```

```

        </p>
    </form>
</body>
</html>

```

У нас есть форма, назовём её `file_upload.php` и сохраним в корневой папке. А теперь нам нужна таблица для базы данных, поэтому давайте создадим её:

```

CREATE TABLE files (
    id int(11) AUTO_INCREMENT,
    file_name varchar(255) NOT NULL,
    PRIMARY KEY(id)
);

```

Все что нам надо в таблице это идентификатор файла и адрес к файлу. И давайте уже создадим скрипт, который будет обрабатывать саму форму:

```

<?php
require("classes/db.php");

$dir = "files/"; // Здесь мы указываем путь к
папке в которую мы хотим загружать файлы
$uploaded_file = $dir .
basename($_FILES["my_file"]["name"]); // тут уже мы
формируем адрес к файлу, а также достаём его название
из массива $_FILES с помощью элемента name

if ($_FILES["my_file"]["tmp_name"],
$uploaded_file)
{ // Здесь уже в условии логической операции
происходит загрузка файла в указанный нами путь
в переменной dir.

```



```
//      Если операция прошла успешно, то тогда
можно делать уже дальнейшее действий внутри логической
конструкции

        $query = "INSERT INTO files (file_name)
VALUES ('$uploaded_file')";
        $request = $connect->query($query);

        if ( $request ) {
            echo "Файл был загружен а также
адрес был добавлен в базу данных";
        } else {
            exit("Простите но произошла
ошибка");
        }
    }
    ?>
```

Сохраните скрипт, назвав его upload.php. Давайте теперь протестируем работает ли все у нас. Загрузите какой-то файл в форму, после чего отправьте его. Если вы видите перед собой сообщение о том что все прошло успешно то не спешите радоваться. Для начала проверьте папку в которой хранятся загружаемые файлы, ну и таблицу чтобы посмотреть добавился ли адрес файла.

Если все есть то это значит что все работает. Ну а как же можно доставать эти адреса и пользоваться ними? То как извлечь какие то данные из таблицы это мы знаем, но вот то как надо использовать адреса файлов не совсем ясно. Я покажу вам небольшой пример с файлом изображения.

У меня есть изображение cat.jpg которое уже храниться в папке файлов, а также есть адрес в таблице базы данных. Адрес в таблице базы данных выглядит вот так files/cat.jpg. Представим, что мы хотим сделать вывод всех изображений, поэтому для этого напишем такой скрипт:

```
<?php
require("classes/db.php");

$query = "SELECT * FROM files";
$request = $connect->query($query);

if ($request) {
    echo "Операция прошла успешно";
    while ($row =
mysqli_fetch_assoc($request)) {
        echo "<img src='" .
$row["file_name"] . "' />";
    }
}

?>
```

Сначала мы делаем типичный запрос на извлечение данных. После этого если запрос прошёл успешно мы извлекаем данные в переменную `$row` и в результате выводим изображения с помощью команды `echo` и HTML тега `` и с помощью атрибута `src` указываем путь к файлу через переменную `$row` где храниться адрес к файлу.

Похожие вещи можно делать с любыми файлами. Главное правильно указать путь к файлу. Думаю некоторые новички кодеры спросят, а как же делается обрезка изображения? Это уже не дело PHP, это можно сделать с помощью языка программирования JavaScript и вообще лучше выбрать для такой задачи библиотеки которые были специально созданы для этого языка программирования.

Пока что рекомендую читателю дальше читать эту книгу и обязательно пробовать программировать что то своё. То есть создавать какие то свои небольшие программки для того чтобы лучше закрепить материал. Но кроме того, уже начинать придумывать то что они сами захотят создать. К примеру свою небольшую соц. сеть и т.п.

Это больше поможет понять программирование на PHP. Но не надо полностью рассчитывать на книгу. В любой сфере какой то профессии надо пользоваться всеми ресурсами для обучения которые есть под рукой. То есть, например я настоятельно рекомендую читать документацию этого языка программирования.

В этом разделе я вам показал, как можно делать запросы к базе данных через PHP. А также показал несколько практических примеров использования этого. На этом моменте я хочу завершить этот раздел и перейти к другой очень важной вещи, такой как сессии, а также мы более подробнее поговорим об cookie-файлах в следующем разделе. А сейчас как и в конце каждого раздела я попрошу читателя ответить на запросы и выполнить упражнения которые показаны ниже.

Вопросы и упражнения для самоконтроля

1. Какая есть функция для подключения к базе данных через PHP?
2. Зачем нужна функция `mysqli_errno()`?
3. Создайте запрос к созданной вами таблице на извлечение всех данных. Также заполните таблицу какими то данными чтобы она не была пустой.
4. Для чего служить функция `mysqli_fetch_assoc()`?
5. Создайте таблицу внутри которой хранились названия книг и их авторы. После этого напишите скрипт который выводил бы названия всех книг и их авторов.
6. Как можно извлечь данные по отдельным характеристикам? (Например, когда `id` равно 4).
7. Напишите скрипт, который обновлял данные определенного поля в таблице.

8. Напишите скрипт для загрузки и хранения изображений в файловой системе приложения. Но при этом чтобы в таблице хранились адреса к изображениям.

12 Сессии

Мы уже с вами встречались с куками в этой книге, в разделе об PHP и HTTP. В этой же главе мы поговорим об сессиях которые реализовывают функционал такой же как и в куки только немного по-другому. Сессии позволяют сохранять данные об пользователю который зашёл на веб-сайт даже тогда когда он закрыл браузер. Или например можно сделать так чтобы выводилось точное число того сколько раз пользователь заходил на ваш веб-сайт.

С помощью сессий как раз можно реализовать систему регистрации и аутентификации в свой аккаунт в приложении. В этом разделе у нас будет пример с созданием регистрации и аутентификации в веб-приложении с помощью PHP. Начну я этот раздел из того что мы разберемся что из себя представляют сессии а также рассмотрим как я уже и говорил более подробно куки.

12.1 Сессии

Сессии это массив переменных, которые хранятся на сервере и относятся только к указанному пользователю. Чтобы начать работу сессий в скриптах вашего приложения вы должны воспользоваться функцией `session_start()`, если мы не начнём работу сессий то мы не сможем обращаться к переменным которые хранятся в сессиях.

В PHP сессии представляют собой массив супер-переменной `$_SESSION` внутри которой можно хранить куки. Например:

```
<?php
session_start();
```

```
$_SESSION["name"] = "My name"; // Здесь мы
создали переменную в массиве сессий с именем name и
присвоили ей значение My name
?>
```

После создания переменной в сессии к этой переменной можно будет обратиться через любой другой скрипт, где была начата работа сессий. Например, у нас есть два файл `page.php` и `ruby.php`. В скрипте `page.php` была создана переменная в сессии с именем `info` со значением “Testing session”. Пользователь сначала запустил скрипт `page.php` где была начата работа сессий, т.е. была функция `session_start()` после этого он перешёл на скрипт `ruby.php` где также была начата работа сессий и где выводилось значение переменной `info` на странице, скрипт выглядел вот так:

```
<?php
session_start();

echo $_SESSION["info"];
?>
```

То есть если было задано значение определенной переменной сессии. А теперь давайте создадим систему регистрации и аутентификации.

12.2 Создание регистрации

Как и всегда для каждого элемента в приложении в котором будет редактирование или добавление каких то данных в базу данных понадобится форма. Для начала мы должны создать форму регистрации, поскольку какой смысл создания аутентификации в приложении если нету регистрации?

Форма будет иметь в себе 4 поля, первое это логин пользователя, второе это e-mail пользователя, четвёртое это пароль пользователя и пятое это пароль пользователя который надо повторить. Вот код формы:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Register Form</title>
    </head>
    <body>
        <form action="register.php"
method="post">
            <p>
                <label>Username</label>
                <input type="text"
name="username" />
            </p>
            <p>
                <label>E-mail: </label>
                <input type="text"
name="email" />
            </p>
```

```
        <p>
            <label>Password: </label>
            <input type="text"
name="password" />
        </p>
        <p>
            <label>Repeat password:
</label>
            <input type="text"
name="repeat_password" />
        </p>
        <input type="submit"
value="Register" />
    </form>
</body>
</html>
```

Хорошо у нас есть форма. Дальше нам надо написать скрипт регистрации, он будет проверять совпадают ли пароли, об безопасности как я уже и говорил у нас будет отдельный раздел. Вот скрипт:

```
<?php
require("classes/db.php");

    if (isset($_POST["username"])) { $username =
$_POST["username"]; }
    if (isset($_POST["email"])) { $email =
$_POST["email"]; }
    if (isset($_POST["password"])) { $password =
$_POST["password"]; }
    if (isset($_POST["repeat_password"])) {
$repeat_password = $_POST["repeat_password"]; }
    // Делаем проверку на наличие значений в наших
полях
```

```
if (empty($username) or empty($email)) {
    exit("Заполните все поля");
} else if (empty($password) or
empty($repeated_password)) {
    exit("Заполните все поля");
}
// Делаем проверку являются ли наши поля
пустыми

if ($password == $repeated_password) //
Проверяем ли совпадают наши пароли, если да то тогда
продолжаем работу приложения
{
    $query = "INSERT INTO users
(username, email, password, repeated_password) VALUES
('$username', '$email', '$password',
'$repeated_password')";
    $request = $connect->query($query);

    if ($request) {
        echo "Регистрация прошла
успешно!";
    } else {
        exit("Простите, но произошла
ошибка");
    }
} else {
    exit("Введенные вами пароли не
совпадают");
}

?>
```


Как вы видите, то здесь есть проверка на совпадение введенных паролей. Если они совпадают, то приложение продолжит свою работу. Но думаю некоторые заметили, что здесь я делаю запрос к таблице, которой не существует. Таблица эта называется `users` и именно в ней мы будем хранить данные об пользователях, давайте же создадим её в базе данных:

```
CREATE TABLES users (  
    id int(11) AUTO_INCREMENT,  
    username varchar(20) NOT NULL,  
    email varchar(20) NOT NULL,  
    password varchar(20) NOT NULL,  
    repeated_password varchar(20) NOT NULL,  
    PRIMARY KEY(id)  
);
```

У нас есть два файла, в нашем приложении которые служат для регистрации. Это `register.php` - форма для регистрации и `complete.php` - обработчик формы. Теперь можете протестировать два этих файла. На первый взгляд код мог вам показаться сложным, но по-настоящему все очень просто. Сначала идет проверка значений, после этого идет проверка на совпадение паролей, и если они совпадают то только тогда будет создан запрос к базе данных на создание нового пользователя.

Можно было конечно же добавить фильтрацию данных. То есть сделать регистрации более безопасной, чтобы какой-то хакер не мог взять и занести вредный код в поля который мог повредить базу данных. Об безопасности как я уже и говорил будет отдельный раздел, а также будет раздел об криптографии. Криптография - это наука которая занимается защитой данных с помощью шифров. То есть, это наука об шифровании. У нас есть простая система регистрации а теперь давайте создадим систему авторизации чтобы пользователь мог авторизоваться в своем аккаунте.

12.3 Создание аутентификации

Если у нас уже есть система регистрации, то можно добавить и систему аутентификации чтобы пользователь мог войти в свой аккаунт. Также мы создадим простой скрипт для просмотра своего аккаунта. Тут уже на помощь приходят сессии. Будет проверка по двум полям. Если поле с логином пользователя совпадает с логином которое введено на базе данных и пароль совпадает с паролем который введен в базе данных то тогда будет происходить авторизация.

Создадим форму для авторизации:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Login</title>
    </head>
    <body>
        <form action="login.php"
method="post">

            <p>

                <label>Username: </label>
                <input type="text"
name="username" />

            </p>
            <p>

                <label>Password: </label>
                <input type="text"
name="password" />

            </p>
            <input type="submit"
value="Sign In" />
        </form>
```

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

У нас есть форма, а теперь давайте создадим скрипт для аутентификации:

```
<?php
require ("classes/db.php");
session_start(); // Мы обязательно должны
запустить работу сессии, если мы хотим чтобы
переменные сессии получали значения и могли их
использовать.

if (isset($_POST["username"])) { $username =
$_POST["username"]; }
if (isset($_POST["password"])) { $password =
$_POST["password"]; }

if (empty($username) or empty($password)) {
    exit("Введите все поля");
}

$query = "SELECT * FROM users WHERE username =
'$username'";
$request = $connect->query($query);

if ($request) {
    $row = mysqli_fetch_assoc($request);
    if ($username == $row["username"] &&
$password == $_POST["password"]) {
        $_SESSION["username"] = $username;
        $_SESSION["email"] = $row["email"];
        $_SESSION["password"] = $password;
```

```
        echo "Вы были успешно  
авторизированы";  
    } else {  
        exit("Введенные вами данные  
неправильные");  
    }  
} else {  
    exit("Простите но пользователь с  
таки логином не найден");  
}  
  
?>
```

Вот так и выглядит авторизация, то есть у нас есть логин и пароль которые проверяются по данным из таблицы. И если они совпадают, то авторизация проходит успешно. Конечно же сейчас все выглядит немного по-другому, но об этом мы поговорим в разделе безопасности где будет идти речь об шифровании паролей. Если у нас вами есть система аутентификации то надо сделать также способ выхода из своего аккаунта, сделать страницу просмотра своего аккаунта.

12.4 Просмотр аккаунта

Сейчас мы будем делать систему просмотра аккаунтов, которая будет работать с помощью поиска id пользователя в базе данных, после чего будут извлекаться данные и выводиться на странице. Это будет довольно таки просто сделать с полученными вами знаниями, поэтому начнём писать скрипт:

```
<?php  
require("classes/db.php");  
  
$id = $_GET["id"];  
$query = "SELECT * FROM users WHERE id =  
'$id'";
```

```

$request = $connect->query($query);

if ($request) {
    $row = mysqli_fetch_assoc($request);

    $id = $row["id"];
    $username = $row["username"];
    $email = $row["email"];
}

?>

<div class="profile">
    <h2><?php echo $username; ?></h2>
    <h3><?php echo $email; ?></h3>
</div>
<?php
    }
?>

```

Мы сделали типичный поиск данных с помощью супер-переменной `$_GET`. У вас уже достаточно знаний чтобы понять выше написанный код и это хорошо. Поэтому я не буду объяснять как все работает. Завершу этот раздел тем, что покажу вам внизу код который можно использовать для того чтобы выйти из своего аккаунта, т.е. завершить сессию:

```

<?php //logout.php
unset($_SESSION["username"]);
unset($_SESSION["email"]);
unset($_SESSION["password"]);
unset($_SESSION["repeat_password"]);
exit("Сессия была успешно завершена");
?>

```

Как и в конце каждого раздела, я попрошу уважаемого читателя ответить на вопросы и выполнить упражнения которые показаны ниже.

Вопросы и упражнения для самоконтроля

1. Что такое сессия и как она работает?
2. Как можно задать значение переменной сессии?
3. Для чего нужно запускать работу сессий?
4. Как бы вы создали систему регистрации в своем приложении?
5. Как завершать работу сессий?

13 Безопасность

На безопасности никогда нельзя экономить. Безопасность пользователя или потребителя всегда должна быть на высоком уровне. Перед тем как мы перейдем к самой теме безопасности в PHP я для начала расскажу об том какие существуют методы взлома приложений. Одним из самых распространенных способов является использование так званых инъекций. Инъекции как раз и называли инъекциями через то что злоумышленник как будто делает инъекцию нашему приложению чтобы "заразить" его вредным кодом, который нарушит работу приложения.

Очень часто используются SQL-инъекции, которые содержат в себе вредный SQL код, инъекции часто внедряют в формы для введения каких-то данных. Чтобы избежать инъекций используется фильтрация данных которая запрещает использование вредных SQL-инъекций.

Сейчас было создано очень много способов для того чтобы избежать всего этого. От использования самого простого кода до использования многих паттернов и библиотек которые предназначены для того чтобы фильтровать вредные данные. Кроме SQL-инъекций, вредный код злоумышленник может также внедрить с помощью сессий.

Или ещё злоумышленник может использовать сессию чтобы украсть данные другого пользователя. Существует много способов взлома приложения и пользователей. Иногда пользователя сами идут на шаг к злоумышленнику веря что они получили сообщение от разработчиков и т.п. и т.д.

Конечно же сейчас с использованием многих фреймворков и библиотек разработка выглядит по-другому. И разработчику не приходится очень долго мучаться над созданием фильтрованием данных, когда за него это уже сделал фреймворк. Если кто то не знает, то фреймворк это набор инструментов и библиотек для облегчения процесса разработки приложения для программиста. И начнём мы с вами с фильтрации данных строк, которые были введены в формулу пользователем.

13.1 Фильтрация данных

Представим себе ситуацию, что пользователь вводит данные которые содержат в себе вредные символы с помощью которых можно сделать команду SQL. Чтобы избежать вредных символов, а точнее символов с помощью которых можно построить инъекции, нужно воспользоваться функцией `str_replace()` :

```
<?php
```

```
$bad_symbols = array("!", "#", ".", ";", ":",
"/", "$", "%", "@");
$plainText = "Hello w@rld!"; // Это будут
данные которые мы с вами обработаем
$clear_data = str_replace($bad_symbols, "",
$plainText); // Результат: Hello wrld
?>
```

Как вы видите, то эта функция имеет целых три параметра. Первый параметр это символ(ы), которые мы хотим заменить в определенной строке, второй параметр это то на что мы хотим заменить эти символы и третий это уже строка в которой мы хотим сделать данную операцию. Я использовал массив символов которые надо заменить. Конечно же, можно было сделать и так:

```
<?php
$bad_symbol = "$";
$plainText = "$var";
$plainText = str_replace($bad_symbol, "",
$plainText); // Результат: var
?>
```

Вообще для фильтрации данных в PHP были созданы отдельные функции. Например, есть функция `filter_has_var()` которая проверяет существование переменной с помощью заданного типа. Ниже показан таблица функций назначенных для фильтрации данных:

Функция	Действие
<code>filter_has_var()</code>	Проверяет существование переменной с указанным типом.
<code>filter_id()</code>	Возвращает

	идентификатор принадлежащий фильтру.
<code>filter_input_array()</code>	Получает массив переменных и при необходимости фильтрует их.
<code>filter_input()</code>	Получает переменную и при необходимости фильтрует её.
<code>filter_list()</code>	Возвращает список всех поддерживающих фильтров.
<code>filter_var_array()</code>	Принимает массив переменных и при необходимости фильтрует её.
<code>filter_var()</code>	Фильтрует переменную с помощью заданного фильтра.

- Табл. 24 - Функции для фильтрации данных в PHP

Функция `filter_has_var()`

Как уже было сказано функция предназначена для того чтобы проверить существование определенной переменной с заданным типом данных. Функция имеет два параметра, первый это один из этих типов: `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, `INPUT_SERVER`, `INPUT_ENV`.

Вам уже немного становится понятно из названий этих типов что это такое. Второй параметр функции это название переменной, которую мы хотим проверить. Функция возвращает ответ только в двух логических значениях, TRUE и FALSE. Если операция прошло успешно то значение будет равно истине (TRUE), если наоборот то тогда ложь(FALSE). Пример:

```
<?php
```

```
$_GET["value"] = "test";
```

```
print_r(filter_has_var(INPUT_GET, "value")); //
```

Значение будет равно истине то есть TRUE

// Здесь мы делаем проверку значения переменной в массив `$_GET`, на тип `INPUT_GET` похожую вещь можно также сделать с типом `INPUT_POST`

```
$_POST["test"] = "PHP Book";
```

```
print_r(filter_has_var(INPUT_POST, "test")); //
```

Здесь также ответ получим значения истины

```
?>
```

Проверка здесь осуществляется на такие простые типы как GET и POST.

Функция `filter_id()`

Функция возвращает ответ в числом виде идентификатора определенного фильтра, например:

```
<?php
```

```
print_r(filter_id("validate_email"));
```

```
?>
```

В этом примере мы достаем значение идентификатора фильтра `validate_email`.

Функция `filter_input_array()`

Функция имеет три параметра первый это один из типов `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, `INPUT_SERVER`, `INPUT_ENV`. Второй параметр это массив в котором определяются атрибуты для фильтра, и третий параметр добавляет в результат ключи со значением `NULL`. Пример фильтрации:

```
<?php
$filters = array(
    "title" => array(
        "filter" => FILTER_CALLBACK,
        "flags" => FILTER_FORCE_ARRAY,
        "options" => "ucwords"
    ),
    "email" => FILTER_VALIDATE_EMAIL,
);
print_r(filter_input_array(INPUT_POST,
$filters));
?>
```

Пример выглядит немного сложно, если посмотреть внимательно то у нас здесь есть 2 переменные это `title` и `email` которым мы задаем параметры на фильтрацию данных. Можно заметить, что где первая переменная мы задаем параметры с помощью флагов, то есть с помощью индекса `flags`.

Функция `filter_input()`

Работает почти также само, как и `filter_input_array` только здесь уже у нас не массив а одна переменная. Функция имеет четыре параметра, первый это типы для переменной которые вам уже были показаны несколько раз, т.е. `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, `INPUT_SERVER` и `INPUT_ENV`.

Второй параметр это уже имя переменной, которой мы хотим сделать фильтрацию. Третий это идентификатор фильтра, который мы хотим применить к данной переменной, и четвертый это массив в котором мы задаем дополнительные параметры для фильтрации.

Сделаем такой пример:

```
<?php
$GET["id"] = 1;
print_r(filter_input(INPUT_GET, "id",
FILTER_VALIDATE_INT)); // Как вы видите, то здесь мы
применили фильтр FILTER_VALIDATE_INT. Фильтры
вызываются как константы
?>
```

Полный список фильтров я пока что показывать не буду. Разве что в конце этого раздела покажу таблицу, где есть их описание. Переходим к следующей функции.

Функция `filter_list()`

Функция не имеет параметров, её надо вызывать в цикле `foreach()` чтобы посмотреть список всех поддерживаемых фильтров. Вот как это выглядит:

```
<?php
foreach(filter_list() as $id=>$filter) {
    echo "<b>" . $filter . "</b>" .
filter_id($filter);
}
?>
```

Если функцию не вызвать в этом цикле то мы не получим никаких результатов. Далее у нас идет ещё две функции `filter_var_array()` и `filter_var()`, они работают так же само как и `filter_input_array()` и `filter_input()` только имеют разные названия.

Давайте сделаем несколько реальных примеров с фильтрацией данных. Конечно же, для реализации фильтрации данных в PHP можно использовать не только функции показаны выше но и любые функции в общем. Например, вот пример фильтрации входных данных на пробелы и запретные символы:

```
<?php
    $symbols = array("!", "~", ",", ".", "<", ">",
"/", "\"", "?", "{", "}", "[", "]", "=", "+", "-",
"_", "*", "&", "^", ":");
    $plaint_text = "[Hello!]";
    $plaint_text = str_replace($symbols, "",
$plaint_text); // Удаляем символы которые были
указаны в массиве symbols
    $plaint_text = str_replace(" ", "",
$plaint_text); // Удаляем пробелы
?>
```

Здесь уже намного больший массив из символами. Если мы хотим сделать фильтрование HTML-тегов то надо воспользоваться функцией `strip_tags()`, функция имеет два параметра, в первом мы указываем строку для фильтрации, а во втором указываем теги которые не надо удалять, пример:

```
<?php
    $text = "<h2>Hello world!</h2><p>Here is
another HTML tag</p>";
    echo strip_tags($text); // Результат: Hello
world!Here is another HTML

    echo strip_tags($text, "<h2>"); // Результат:
Hello world!<h2>Here is another HTML tag</h2>
?>
```

Это полезно использовать там где пользователь создаёт посты, например если найдется пользователь который захочет создавать сам теги

в своем посте а мы этого не хотим, то тогда использовать эту функцию. А если мы хотим сохранить эти теги в базе данных но при этом, зашифровав их в специальные сущности надо воспользоваться функцией `htmlspecialchars()`:

```
<?php
    $to_encrypt = "<p>Это просто пример</p>";
    echo htmlspecialchars($to_encrypt);
    // Результат: &lt;p/a>Это просто
пример&lt;p/a>
?>
```

Если вы хотите избежать того чтобы пользователь вводил пробелы в начале или в конце строки воспользуйтесь функцией `trim()`:

```
<?php
    echo trim(" Hello world "); // Результат:Hello
world
?>
```

Хорошо, а теперь давайте создадим сами функцию которая могла фильтровать данные зарегистрированного пользователя, это уже более реальный пример:

```
<?php
function filter_data($text) {
    $bad_symbols = array("\x27", "\x22",
"\x60", "\t", "\r", "\n", "+", "=", "-", "*", "%",
"?", "<", ">", "!");
    $text = trim($text);
    $text = strip_tags($text);
    $text = str_replace($bad_symbols, "",
$text);
    $text = str_replace(" ", "", $text);
}
?>
```

Эту функцию вы можете использовать для фильтрации полей. Такая функция сделает ваше приложение почти неуязвимым к SQL-инъекциям, но снова таки повторю. На практике все выглядит намного по-другому и сейчас применяются другие методы фильтрования данных, но то что я вам показываю даёт вам понять как все работает.

13.2 Безопасность файловой системы

Кроме какой-то фильтрации данных надо также реализовать безопасность файловой системы перед вредными файлами, которые могут быть загружены в файловую систему базы данных. Для этого надо будет сделать проверку форматов файлов чтобы в систему не могли попасть например php-файлы с помощью которых можно внедрить червей в приложение.

Как вы смогли понять то с помощью файлов загруженных в приложение можно делать атаки на само приложение. Давайте себе представим небольшую ситуацию когда в нас имеется приложение с пользователями и каждый пользователь, например, может удалять свою папку. Вот код который занимается удалением папки:

```
<?php
    session_start(); // Запускаем работу сессий
чтобы мы могли обратиться к их переменным
    $username = $_SESSION["username"];

    $file_delete = $_POST["file"];

    unlink($file_delete);

    echo "Your directory has been deleted";
?>
```

Думаю уже некоторые читатели смогли заметить в чем здесь заключается проблема. Как вы можете здесь заметить то здесь нету

проверки на пользователя, во-вторых вы можете заметить, что файл который надо удалить вводить пользователь и эти данные мы получаем через массив \$_POST. А это значит, что любой злоумышленник сможет сделать атаку на файловую систему. То есть нам надо сделать так чтобы была проверка файлов, а также пользователя на авторизацию.

И кроме этого проверку на существование файла чтобы мы не получали ошибок если файл не будет существовать. Вот код нашего скрипта:

```
<?php
    if (empty($_SESSION["username"]) or
empty($_SESSION["password"])) { // Делаем проверку на
авторизацию, т.е. если сессия будет пустой то тогда
это значит что пользователь не авторизирован
        exit("Эта страница для авторизованных
пользователей");
    } else {
        $username = $_SESSION["username"];
        $file = basename($_POST["userfile"]);
        $home = "/users/$username"; //
Устанавливаем папку пользователя
        $path_to_file = "$home/$file"; //
Устанавливаем путь к указанному файлу пользователем

        if (file_exists($path_to_file) &&
unlink($path_to_file)) {
            echo "Файл был успешно удален.";
        } else {
            exit("Файл не удалось удалить");
        }
    }
?>
```


Теперь у нас есть проверка на авторизацию а также проверка пути файла, а точнее мы задаем путь к файлу и папке пользователя чтобы не было атак. Этот код стал более безопаснее. Но если бы у нас была дополнительная система, которая проверяла бы существование файла в файловой система но также и существование в базе данных – то это сделало бы система ещё более безопасной.

И при этом с помощью таблицы в базе данных можно было бы проверять авторство файлов по пользователю который авторизированный. Давайте сейчас сделаем пример со скриптом для загрузки файлов и при этом была также проверка формата файла который загружается на сервер.

Вот код данного скрипта:

```
<?php
...
$file_to_check =
mime_content_type($_FILES["file"]["tmp_name"]); // С
помощью этой переменной мы как раз и будем делать
проверку формата файла, можете заметить какая здесь
функция используется для этого
if (in_array($file_to_check, array() )) { // В
массиве которое указано в условии мы указали все
форматы которые будут позволены для загрузки
move_uploaded_file($_FILES["file"]["tmp_name"],
"home/uploads/" . $_FILES["file"]["name"]);
echo "Файл был успешно загружен";
} else {
exit("Произошла ошибка, или вы пытаетесь
загрузить файл с другим форматом или ошибка в
сети.");
}
?>
```

Как видно из примера то сначала мы использовали функцию которая проверяет форматы, т.е. `mime_content_type()`. После этого в нашей логической конструкции мы с вами указали массив доступных форматов для файлов, а также сделали их проверку.

Если файл имеет один из разрешенных форматов которые мы указали, то тогда файл будет загружен в **указанную директорию** в функции `move_uploaded_file()`. В такой способ мы сделали довольно таки примитивную проверку форматов файлов.

Но что делать, если название файла будет иметь в себе какие-то запрещенные символы? В таком случае надо сделать поиск заданной подстроки в названии файла с помощью функции `strstr()`. В основном главное это чтобы загруженные файлы на сервер не могли навредить веб-приложению или украсть данные из БД.

На этом я завершу этот подраздел, и в следующем подразделе мы с вами поговорим об такой науке как – Криптография. Сейчас каждый разработчик должен знать хотя бы основы использования этой науки, поскольку шифрование и защита данных сейчас очень актуальная. Поэтому переходим к следующему подразделу.

13.3 Криптография

Я уже упоминал как то об этой науке в этой книге. Повторю снова что это за наука. Криптография – это наука, которая занимается созданием шифров и защитой данных с помощью использования математических методов. То есть это как бы сказать наука об шифрах и шифровании.

В этом подразделе я вам расскажу об самом понятии шифров, какие есть их виды и для чего вообще они нужны. Начнём пожалуй с того что я расскажу немного истории об этой науке. Как не странно но начало этой науке дали великие древнегреческие ученые. Ну и вообще очень много современных наук появились благодаря этим талантливым и мудрым ученым.

Первым очень простым инструментом который, использовался для того чтобы зашифровать сообщение являлась небольшая палочка которая называлась Скиталой. Скитала имела определенное число кутов и определенную длину. На палочку наматывалась лента как это показано на данном изображении:

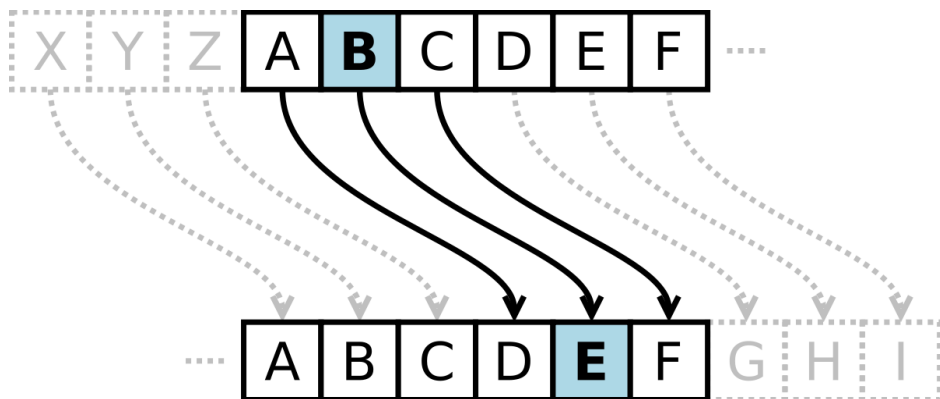


- *Рис. 21 - Инструмент Скитала. Изображение было взято с открытой энциклопедии Википедии.*

Лента могла быть из пергамента или кожи, т.е. сделано с такого материала на котором можно было нормально писать сообщения. Намотав ленту человек писал своё сообщение после чего передавал такую ленту человеку который хотел прочитав это сообщение. Человек который хотел прочитав сообщение также имел копию той же самой скиталы на которой было написано сообщение.

Если человек не имел копии той скиталы на которой было сообщение то он просто не мог прочитав сообщение. Это было только начало криптографии, дальше все становилось намного интереснее. Следующей технологией, которая использовалась для шифрования был шифр Юлия Цезаря.

Внизу вы можете схему на которой показано как он работает:



- Рис. 22 - Алгоритм работы шифра Цезаря. Изображение было взято с открытой энциклопедии Википедии.

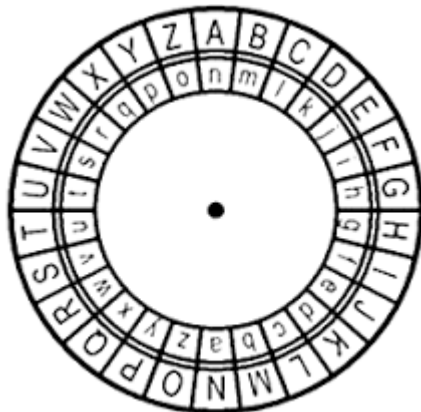
Как вы видите, то здесь есть основной алфавит и алфавит с зашифрованными буквами. А точнее просто с измененной позицией букв в алфавите. Например, если мы хотим с вами зашифровать букву А то нам надо перейти на три символа вперед и тогда выбрать этот символ.

И именно тот выбранный символ это будет наша зашифрованная буква А. В нашем примере это буква D. Если у нас будет буква В то тогда это уже будет Е. А если это С то буквой шифра будет F. То есть суть шифра заключается в том что если мы хотим с вами ч зашифровать букву то мы должны с вами двигаться на три символа вперед после чего выбирать этот символ.

В такой способом можно шифровать сообщения. Но шифр Цезаря довольно таки прост и его просто взломать. Двигаемся вперед в истории криптографии, и дальше на нас ждет Киевская Русь. Во время этой эпохи был созданный шифр который использовал алфавит древнерусского языка и алфавит древнегреческого языка.

Сообщение шифровалось по той схеме, что некоторые буквы просто заменялись на буквы древнегреческого алфавита. Если была древнерусская буква З – то есть земля, то ее могли заменить на греческую букву ζ – дзета.

Приблизительно вот так происходило шифрование в то время. Конечно же сейчас я не буду полностью вам рассказывать историю всей криптографии потому что это может занять очень и очень много времени и страниц этой книги. Поэтому я рассказываю только ключевые и интересные моменты чтобы читатель мог немного отдохнуть от программирования. В эпоху Возрождения был созданный шифровальный диск Альберти. Вот как он выглядел:



- *Рис. 23 - Шифровальный диск Альберти. Данное изображение было взято из портала ipro.spb.ru*

Данный шифр очень сильно напоминает шифр Цезаря поскольку здесь так же используется схема с заменой символов в шифре. Только здесь уже нам надо перемещаться на 3 буквы вперед. Суть заключается в том что шифр мог иметь заданные правила шифрования, а точнее уже установленную последовательность букв.

Как вы можете заметить, то на изображении есть два диска. На первом диска были буквы алфавита которые мы хотели использовали когда нам надо было зашифровать определенный символ. А внизу располагался уже зашифрованный вариант этих букв. Первый диск служил указателем для шифрованных букв. То есть например, если мы хотим зашифровать слово FAKE, то мы просто должны были бы по очереди смотреть буквы на двух дисках после чего делать шифрование.

Вот как бы выглядело это слово в зашифрованном виде - indj. Конечно же, данный шифр мог взломать только тот у которого был такой же самый диск с помощью которого и шифровалось сообщение. Такая же ситуация как и Скиталой. Такой шифр по сути невозможно было взломать. Двигаемся дальше, и следующей эпохой в нас была Эпоха Нового Времени. В это время был создан шифр Виженера, это была большая матрица с ключом который использовался для того чтобы зашифровать либо расшифровать сообщение.

Внизу есть большая схема с шифром Виженера:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

- Рис. 24 - Демонстрация работы шифра Виженера

Матрица выглядит довольно таки сложной. С первого раза вы не сможете понять все сразу, но попробую все объяснить. Как я и говорил у нас есть ключ который себе создаем для того чтобы шифровать и дешифровать данные.

Ключом может быть определенное слово, например, у меня это будет слово cool. И пусть слово которое мы хотим с вами зашифровать это будет погма. И теперь смотрите на матрицу, с левой стороны вы видите столбец алфавита - это столбец для букв ключа. А сверху находится ещё один столбец и это столбец для букв данных которые мы хотим зашифровать.

Теперь если мы с вами хотим зашифровать слово погма то тогда нам надо для начала искать букву С в столбце для букв ключа и букву N в столбце для букв исходных данных. И в результате мы должны с вами найти клетку с буквой Р. То есть вы делаете шифрование по координатах как это показано на изображении выше с буквой F.

Это была эпоха Нового времени. Дальше через несколько сотен лет во время второй мировой войны данная наука очень сильно понадобилось в вооружение многих стран для защиты своих военных данных и документов.

Великим толчком для развития компьютеров или как на то время их называли ЭВМ(Электрическая-Вычислительная-Машина) как раз и стала шифровальная машина Енигма, которая была создана немецкими учеными для немецкой армии на то время. Енигму вы можете увидеть на изображении которое показано ниже:



- *Рис. 25 - Внешний вид шифровальной машина “Енигма”. Данное изображение было взято с веб-сайта rtr.md*

Объяснять то как она работает не буду, но скажу очень важный факт что благодаря тому что удалось взломать шифр данной машины то войну удалось сократить на целых 3 года. Ну а что было дальше с этой великой наукой после второй мировой войны?

Начался появляться новый раздел криптографии – Цифровая криптография. То есть криптография, которая используется в компьютерных системах для цифрового шифрования данных. Цифровая криптография позволяет автоматизировать шифрование данных, а также саму скорость шифрования и дешифрования. То есть ученому теперь не приходится крутить какие вентели для того чтобы зашифровать часть данных, все автоматизировано и нажав одну кнопку программа сама шифрует данные а ученому остается только подобрать параметры шифрования.

Сейчас знание криптографии и информационной безопасности это очень важно для любого разработчика. Давайте с вами в общем поговорим об шифрах и об их видах. Шифр это набор определенных правил и параметров для того чтобы защитить данные. Если кратко, то это просто алгоритм для защиты данных. Шифры имеют ключи которые используются для того чтобы зашифровать или расшифровать данные. Ключи имеют выбранный алгоритм шифрования, а также имеют длину в битах. Шифры бывают трех видов:

- Симметричные – это шифры, которые используют только один ключ для шифровки и дешифровки данных. Длина таких шифров может быть от 16 до 256 бит.
- Асимметричные – это шифры, которые имеют два ключа для шифрования и дешифрования данных. Первый ключ называется публичным и используется для того чтобы зашифровать данные. Второй ключ называется приватным ключом который используется для того чтобы расшифровать данные.
- Хэш-функции – это уже цифровые подписи которые используются для того чтобы подтвердить свою личность.

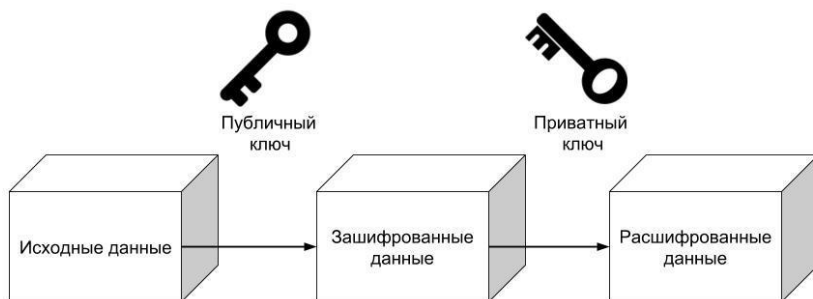
Внизу показана схема работы симметрического шифра:



- Рис. 26 - Демонстрация работы симметричного шифрования

Как вы можете заметить, то на данной схеме есть два клиента. Первый клиент делает шифрование сообщения с помощью ключа который вы можете увидеть сверху схемы. Данный ключ используется для того чтобы зашифровать и расшифровать сообщение.

После шифрования сообщения клиент передает сообщение по сети второму клиенту, который получив его делает дешифрование сообщения с помощью того же самого ключа с помощью которого и была сделана шифровка сообщения.



- Рис. 27 - Демонстрация работы асимметричного шифрования

Здесь уже схема работы асимметричного шифрования. Я упростил схему забрав клиентов а показав только основные моменты работы этого типа шифров. Как я и говорил, публичный ключ используется для того чтобы зашифровать данные, а приватный чтобы их расшифровать. То есть если бы у нас было бы два клиента то это выглядело вот так. Первый клиент используя свой публичный ключ и имея публичный ключ второго клиента делает шифрацию сообщения, после этого он передает это самое сообщение второму клиенту.

Который в свою очередь делает дешифрацию этого сообщения. И в результате он может его прочитать и ответить первому клиенту. Думаю вы уже поняли в чем смысл работы? На всякий случай повторю, публичный ключ - для того чтобы зашифровать сообщение, а приватный ключ - для того чтобы дешифровать сообщение.

Теперь, давайте перейдем к последнему виду шифров, т.е. к хэш-функциям. Как я уже и говорил то хэш-функции это цифровые подписи для того чтобы подтвердить свою личность. Некоторые могут не знать что такое хэш-функция, поэтому скажу сразу, что это преобразование массива входных данных любой длины в битовую строку определенной длины, который выполняется определенным алгоритмом.

Простым примером работы хэш-функции является схема, которая показана ниже:



- Рис. 28 - Демонстрация работы хэш-функции

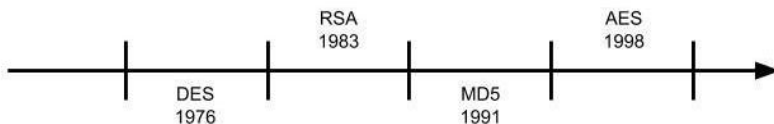
Мы просто даём входные данные, после чего получаем выходные данные. После того как мы с вами получаем цифровую подпись - мы можем ее прикрепить к нашему с вами зашифрованному сообщению. Мы могли конечно же рассматривать все очень подробно. К примеру, то как работает симметричное шифрование на примере какого-то определенного шифра, но мы этого не делали по одной простой причине. Работа алгоритмов шифрования очень сложная изнутри.

Например, чтобы понять как работает алгоритм AES(а об этом мы алгоритме мы поговорим позже) вам бы понадобились для этого знания математического анализа и статистики. А все потому что криптография решает проблемы шифрования с помощью математических методов.

В следующем подразделе мы с вами поговорим об том какие есть популярные шифры для защиты данных. Кратко рассмотрим их работу, а также познакомимся с блочными и потоковыми шифрами.

13.4 Шифры

Мы уже с вами знаем, какие существуют виды шифров. Теперь можно поговорить об том какие именно шифры используются на данный момент для шифрования данных. Я сделал небольшой time-line чтобы вам показать как появлялись шифры на этот свет:



- *Рис. 29 - Временная линия с демонстрацией появления популярных шифров*

Здесь я вывел одни из самых популярных шифров которые сейчас используются для того чтобы шифровать данные. Первый алгоритм, который начинается этот time-line является симметричный шифр DES. Без аббревиатуры это звучит вот так Data-Encryption-Standart.

То есть если перевести это на русский язык, то это будет звучать как Стандарт-Шифрования-Данных. И да, это реально стандарт шифрования данных который используется. Переходим к следующему шифру, и дальше у нас идет популярный ассиметричный шифр RSA.

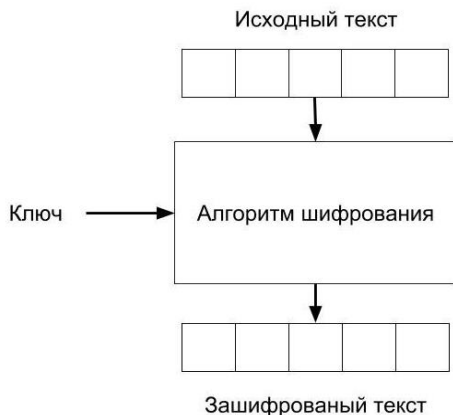
Название данного шифра походит от фамилий разработчиков данного шифра, т.е. Rivest, Shamir и Adleman. Сейчас это один из самых популярных шифров, он используется в приложении Telegram для шифрования сообщений. Но Telegram использует не только шифр RSA, оно использует также другие шифры которые показаны в time-line. Дальше в нашем time-line идет цифровая подпись MD5, аббревиатура которой расшифровывается как Message-Digest-5 – Дайджест Сообщения 5.

Данный алгоритм можно использовать не только как хэш-функцию для цифровых подписей, но также как простой генератор имен для загружаемых файлов в веб-приложение. Это сделает систему файловую систему безопаснее. И последний шифр, который у нас остался это симметричный шифр AES или Advanced - Encryption - Standart – Расширенный-Стандарт-Шифрования. Ну и как не странно это также стандарт шифрования, это понятно также из названия шифра, но кроме этого данный шифр также используется в приложении Telegram.

Хорошо, я вас познакомил с популярными шифрами. Об их работе мы поговорим позже, а сейчас же чтобы двигаться нам дальше нам надо узнать что такое блочное и потоковое шифрование.

Блочное шифрование это шифрование, которое используется в симметричных шифрах для более безопасного шифрования. Существует много разных режимов блочного шифрования, и выбирать стоит той который более подходит под решаемую задачу. Но вопрос, что же такое блочное шифрование?

Внизу показана схема, которая как раз и демонстрирует работу блочного шифрования:



- *Рис. 30 - Схема демонстрирующая работу блочного шифрования на примере режима блочного шифрования ECB*

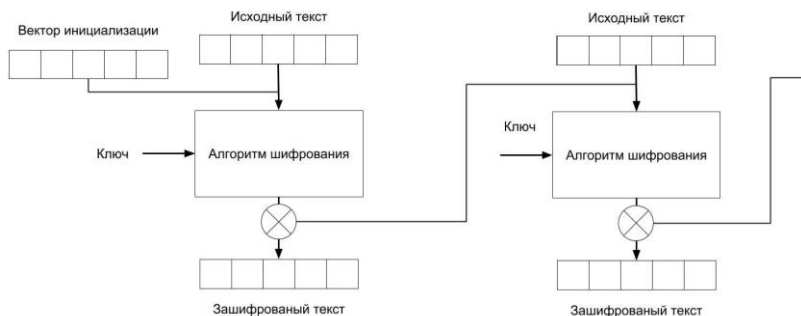
Во-первых, блочное шифрование это когда исходный текст разбивается на блоки которые имеют длину ключа, которым будет шифроваться сообщение. Блоки в свою очередь потом берутся и шифруются по отдельности ключом.

В результате вектор инициализации задает случайную последовательность для этих блоков. И в конце все эти блоки собираются

в одно целое и у нас получается шифротекст. Т.е. зашифрованные данные.

Теперь посмотрите на схему и сверху вы как раз и сможете увидеть блоки исходного текста. И дальше все идет так как я вам объяснял только что, т.е. дальше идет обработка этих блоков ключом и алгоритмом шифрования. И как показано в конце схемы – то все это становится одним целым и мы получаем зашифрованные данные. Довольно таки просто. Мы только что с вами рассмотрели режим блочного шифрования ECB(Electronic-Code-Book) или Электронная Книга-Кодов.

Существует много других режимов блочного шифрования. Например, CBC который мы сейчас с вами рассмотрим.

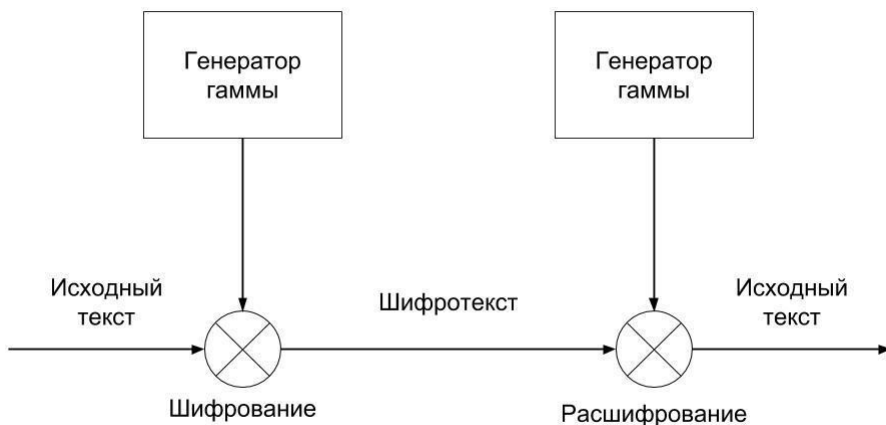


- Рис. 31 - Схема демонстрирующая работу режима блочного шифрования CBC

На схеме есть вектор инициализации, который задает случайную последовательность блоков. Если посмотреть направо, то там дальше идут другие блоки в которых используется тот самый вектор инициализации который использовался в самом начале для шифрации первого блока.

И так вектор инициализации переходит к каждому следующему блоку и так пока не будет полностью зашифровано сообщение. Хорошо, только что мы с вами рассмотрели, что такое блочные шифры, а теперь давайте рассмотрим что такое потоковые шифры.

Потоковый шифр – это симметричный шифр, в котором каждый символ исходного текста преобразуется в символ шифротекста в зависимости от ключа и его расположения в потоке исходного текста. Посмотрите на схему которая показана ниже:



- Рис. 32 - Схема демонстрирующая работу потоковых шифров

Генератор гаммы генерирует потоковые шифры для того чтобы зашифровать исходный текст. После этого происходит процесс шифрования и мы получаем шифротекст. И дальше на схеме показывается процесс дешифрации шифротекста.

Потоковые шифры делятся на два типа:

- Синхронизирующие – это потоковые шифры, в которых поток ключей генерируется независимо от исходного текста и шифротекста.
- Самосинхронизирующее – это потоковые шифры, в которых поток шифров создается функцией ключа и зависит от определенного числа символов шифротекста.

Каждый из этих двух типов потоковых шифров имеют свои плюсы и минусы. Например, в синхронизирующих потоковых шифрах или СПШ есть плюс в том что есть отсутствие эффекта распространения

ошибок. То есть, только один бит который был искаженным будет зашифрован неверно.

Внизу я вывел другие плюсы и минусы СПШ:

+ предохраняет от любых вставок и удаления шифротекста, если они будут то они будут обнаружены и это приведет к потере синхронизации.

- СПШ уязвимы к изменению отдельных битов зашифрованного текста. Если злоумышленнику известен исходный текст - то он сможет изменить биты так чтобы расшифровать шифротекст.

Внизу показаны плюсы и минусы АПШ или Самосинхронизирующих Поточковых Шифров:

+ Размешивание статистики исходного текста. То есть поскольку каждый символ исходного текста влияет на шифротекст, то статистические свойства исходного текста распространяются на весь шифротекст. И через это АПШ может быть устойчивым к атак на основе избыточности исходного текста, чем СПШ.

- распространенные ошибки, то есть каждом неправильному биту соответствует N ошибок в исходном тексте.

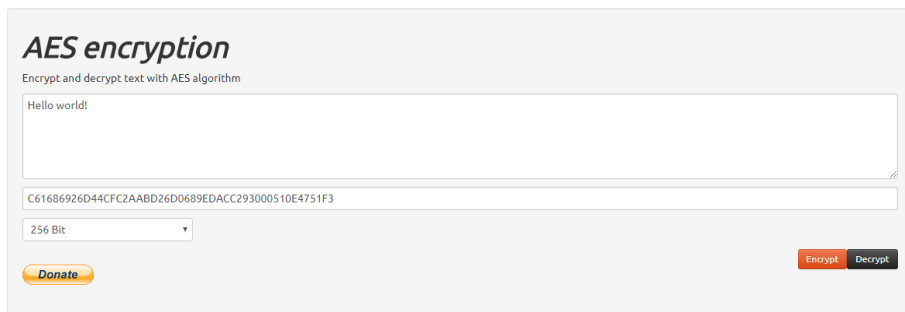
- чувствительность к вскрытию повторной передачей. То есть, при многих передачах шифрования и дешифрования того самого шифротекста могут появиться ошибки в битах.

Мы с вами рассмотрели, какие существуют шифры и их виды. А что же можно сказать об тех технологиях которые существуют для того чтобы реализовать способы шифрования в своем приложении. Если вы хотите попробовать что то зашифровать то можно воспользоваться проектом AES Encryption.

AES Encryption как понятно из названия то это веб-сайт который позволяет тестировать шифр AES, все что вам надо это только симметричный ключ данного шифр, длиной от 128 до 256 бит. Ну и конечно же данные которые вы хотите зашифровать.

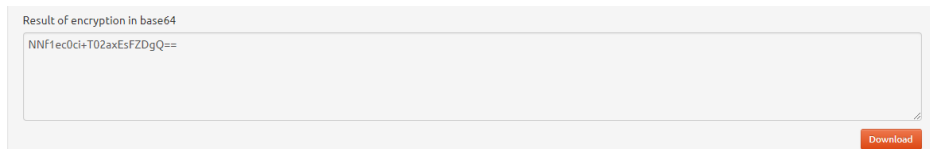
Ключ можно достать из веб-сайта AES Security <https://asecuritysite.com/encryption/keygen>. На данном веб-сайте вы сможете подобрать все нужные параметры для вашего симметричного ключа, главное подобрать длину ключа и кодовое слово которое будет зашифровано в ключе.

Вы получите все данные об вашем ключе, также вы сможете просмотреть вектор инициализации который используется в вашем ключе на данном сервисе. Но нас интересует именно ключ, поэтому скопируйте ключ после переменной key. После генерации ключа можете отправиться на AES Encryption и вы можете заполнить поля так как я заполнил на изображении показанном ниже:



- *Рис. 33 - Форма для ввода данных чтобы зашифровать произвольные данные*

После того как вы ввели все данные в поля нажмите на кнопку “Encrypt”, и в результате вы должны получить шифротекст тех данных которые вы ввели в поле для исходного текста. У меня это выглядит вот так:



- *Рис. 34 - Вывод результата шифрования на AESEncryption*

То есть я зашифровал своим ключом фразу “Hello world!”. У вас конечно же с вашим ключом получится совсем другой результат. Думаю некоторые читатели которые сразу зашли на данный веб-сайт заметили пункт в меню под название «PHP». И если вы перейдете по данной ссылке то вы попадете на страницу где вы сможете взять и скопировать код класса который сможет реализовать функции шифрования с помощью алгоритма AES.

Класс не представляет из себя ниченого сложного, это просто набор функций для шифрования и дешифрования данных с помощью алгоритма AES. Скопируйте данный код и сохраните в отдельном файле назвав его при этом `aes.php`. После этого создайте другой файл который должен называться `cipher.php`, сейчас мы попрактикуемся с шифрованием в PHP, вставьте в файл `cipher.php` этот код:

```
<?php
    include("aes.php"); // подключаем класс к
нашему скрипту

    $data_to_encrypt = "Encrypt me!";
    $pass_for_key = "Cool key";
    $length = 256;

    $aes = new AES($data_to_encrypt,
$pass_for_key, $length);
    $encrypted = $aes->encrypt();
    $aes->setData($encrypted);
    $decrypted = $aes->decrypt();

    echo "Результат после шифрования: " .
$encrypted . "<br />";
    echo "Результат после дешифрования: " .
$decrypted . "<br />";
?>
```

Теперь я объясню что как и к чему. Сначала мы подключаем класс к нашему скрипту, чтобы мы могли воспользоваться его функциями. После этого мы создаём 3 переменные, которые нам понадобятся когда мы будем создавать объект нашего класса. Этими переменными являются `$data_to_encrypt`, `$pass_for_key` и `$length`, Первая переменная содержит данные, которые мы хотим с вами зашифровать. Вторая же хранит в себе кодовую фразу для нашего ключа. И третья содержит данные об длине блока.

После того как мы создали с вами эти переменные мы создали объект класса алгоритма AES, указав при этом 3 созданные нами переменные как параметры класса. Далее мы сделали шифрование данных и создали переменную `$encrypted` который содержит в себе зашифрованные данные. А после этого мы создали переменную `$decrypted`, которая уже хранит в себе дешифрованные данные данных которые мы зашифровали. И в конце нашей программы мы сделали вывод сначала зашифрованных данных, а потом уже дешифрованных данных.

В такой способ мы смогли увидеть как работает шифрование с использованием данного класса. Рекомендую читателю сейчас по пробовать запрограммировать что то свое. Например, задать свою длину для блока или указать другие данные которые вы хотите зашифровать. Теперь я скажу такую вещь, только что мы с вами воспользовались библиотекой. Стоп, не надо думать что мы только что брали и использовали какую то библиотеку для того чтобы получить какую то книгу и т.п.

Нет, мы с вами воспользовались библиотекой которая в немного расширила наш с вами функционал. То есть в нашем случае добавила возможность шифрования данных с помощью алгоритма AES. Библиотека в программировании это как бы сказать свое родное расширение для определенного языка программирования. Библиотеки облегчают работу программисту, что позволяет более эффективно разрабатывать приложения. На данный момент библиотек очень много и

очень часто программисты пользуются именно библиотеками, а не функциями самого языка программирования. И правильно, зачем же “изобретать велосипед”? Конечно же, «изобретать велосипед» хорошо только тогда, когда ваш «изобретенный велосипед» будет лучше за уже созданные «велосипеды».

Поэтому я хочу сказать читателю, что если он будет разрабатывать что то в будущем, то пусть обязательно использует библиотеки и разные расширения чтобы тратить меньше времени на разработку. Это стандарт использовать библиотеки и любые другие инструменты, которые позволяют облегчить разработку. Мы с вами установили и воспользовались библиотекой, которая добавила функционал шифрования с алгоритмом AES. Класс был довольно таки маленьким, но он все равно выполнял те математические операции которые выполняются во время шифрования с алгоритмом AES.

Существует очень много других криптографических библиотек для PHP, которые добавляют алгоритмы шифрования популярных шифров. И сейчас мы с вами познакомимся с ещё одной такой библиотекой. А точнее с расширением. В официальной документации вы можете найти статью об том, какие существуют расширения на PHP для реализации криптографических операций. Первым таким расширением является расширение PECL.

PECL имеет множество пакетов которые могут установиться вместе с этим расширением для того чтобы расширить ещё больше функционал в сторону криптографии в PHP. Официальной документацией является веб-сайт <https://pecl.php.net>. Именно с этого веб-сайта вы можете установить данное расширение. Самым простым способом установки PECL это будет скачивание zip-архива. Посмотреть все доступные версии данного расширения, а также все доступные zip-архивы можно просмотреть за адресом: <https://pecl.php.net/package/zip>.

Я выбрал версию 1.15.0 данного расширения. В zip-архиве данного расширения вы сможете найти файлы данного расширения, но также и просмотреть примеры с использованием данного расширения. В официальной документации PHP есть отдельная статья об том как установить PECL и как его использовать. А теперь давайте уже перейдем к установке данного расширения. Во-первых, для того чтобы установить PECL нам понадобится система PEAR(это также расширение если что). Веб-сайтом данной системы является <http://pear.php.net>.

Для установки PEAR нам понадобится просто взять и установить файл установки `go-pear.phar`. Поэтому перейдите в раздел установки(download), и следует дальнейшим инструкциям. Ссылка на установку файла показана на изображении ниже:

Getting and installing the PEAR package manager

Windows

After you have downloaded and installed PHP, you have to manually execute the batch file located in e.g. `c:\php\go-pear.bat`. Alternatively, download <https://pear.php.net/go-pear.phar> with your browser and save the output to a local file named `go-pear.phar`. You can then run

```
php go-pear.phar
```

- *Рис. 35 - Ссылка на установку PEAR на официальном сайте проекта PEAR*

Скачайте этот файл и переместите его в папку `php` которая находится в папке локального сервера XAMPP. После этого откройте командную строку и перейдите в папку `php` нашего локального сервера. Если вы не знаете, как это сделать, то это можно сделать с помощью команды `cd` которая используется для перехода в папки. Когда вы откроете командную строку, то вы сразу заметите что вы будете находиться на диске C: в папке пользователя. Но это ещё, если вы пользуетесь операционной системой Windows.

У меня хатрр установлен в корневой папке на диску C: и сейчас мы перейдем в папку локального сервера. Чтобы перейти в корневую папку диска C: надо прописать такую команду:

```
C:\Users\User>cd /
```

```
C:\>
```

Как вы можете заметить то в такой способ мы перешли в корневую папку диска C:. Внизу показано то как я перешёл в папку с локальным сервером:

```
C:\>cd хатрр
```

```
C:\хатрр>cd php
```

```
C:\хатрр\php>
```

Или конечно же, можно это сделать вот так:

```
C:\>cd хатрр/php
```

Но если вы установили XAMPP на диск D:, то чтобы перейти на диск D: выполните следующую команду:

```
C:\>cd /d D:\
```

```
D:\>
```

Теперь после того как вы перешли в папку из PHP нашего локального сервера то вы можете начать установку PEAR. Скачайте и переместите файл go-pear.phar в папку php. В документации по установке PEAR была описана команда, которая начинает установку PEAR, установка делается с помощью этой команды:

```
C:\хатрр\php>php go-pear.phar
```

В этой команде вы можете заметить что мы обращаемся к приложению php.exe которое есть в данной папке. Именно это приложение отвечает за работу PHP на локальном сервере. После запуска этой команды, PHP спросит устанавливаем ли мы системную или локальную копию PEAR. Мы конечно же устанавливаем системную копию поэтому просто жмем кнопку enter. После этого PHP выведет список папок, в которые он установит файлы, мы также на это не обращаем внимания а просто жмем кнопку enter, поскольку мы хотим установить PEAR по умолчанию.

Дальше начнется установка файлов данного расширения. Но на этом установка расширения не заканчивается, теперь нам надо взять и отредактировать системный файл PHP - `php.ini`. Данный системный файл находится в папке PHP. Откройте данный файл в редакторе кода после чего добавьте в конец файла такую строку:

```
include_path = ".;c:\php\pear"
```

Сохраните файл. Добавив эту строку кода, мы как раз и подключили данное расширение к PHP. И мы можем теперь пользоваться его функционалом. Дальше нам надо будет запустить файл регистра данного расширения. Просто откройте его, файл регистра имеет название `PEAR_ENV.reg`. Дальше перезапустите ваш локальный сервер, если он у вас был включен. Перезапустите также и командную строку. Это чтобы расширение запустилось, и было зарегистрировано в системе.

И теперь попробуйте протестировать расширение с помощью такой простой команды как проверка версии расширения:

```
pear version
```

После этого если у вас выведутся данные об версии расширения, то это будет значить что вы установили успешно расширение PEAR. Теперь вы умеете устанавливать расширения на PHP, и частенько это может понадобиться, когда вы будете что то разрабатывать. И главное что вы это пробовали на практике. Давайте вернемся к нашей с вами теме об установке PECL. Но перед этим хочу сказать, что только что установленное расширение используется для повышения безопасности, с помощью дополнительного функционала.

А теперь давайте поговорим об установке в PECL. Но перед этим скажу, что PECL это не одно целое и единое, а это отдельный сборник расширений для PHP работающих через PEAR которые мы с вами потом можем выбрать для своих целей. На официальном сайте мы можем найти страницу на которой мы можем просмотреть все доступные для нас расширения которые разделены по разных категориях. И именно там можно наткнуться на категорию Encryption что переводиться как шифрование.

На данный момент есть только 6 PECL расширений, которые позволяют реализовать алгоритмы шифрования. Сейчас мы с вами установим расширение для PECL, которое имеет название `mcrypt`. Как раз с ним мы сейчас и поработаем.

Установить данное расширение можно за адресом <https://pecl.php.net/package/mcrypt>. Скачайте `.tgz`-архив расширения и разархивируйте его в папку `pear`, которая находится в папке `php`. Далее вам надо подключить данное расширение через системный файл `php.ini`:

```
include_path = ".;c:\php\pear\mcrypt-1.0.1"
```

После этого вы сможете воспользоваться функциями данного расширения. Теперь давайте протестируем данное расширение, для этого мы создадим небольшой тестовый скрипт. Создайте новый скрипт под название `mcrypt.php` и добавьте в него такой код:

```
<?php
$key = hash("sha256", "my key", true);
print_r($key);
?>
```

Теперь я вам объясню вам, что мы только что сделали. Это была всего лишь одна строка, но ей нам хватит для того чтобы понять работает ли наше расширение. Мы создали хэш, а точнее цифровую подпись с алгоритмом SHA256, после этого мы вывели данные ключа на страницу чтобы можно было увидеть какой-то результат того что мы только что сделали.

Хэш мы создали с помощью функции `hash()`, сначала мы указали алгоритм, который хотим использовать для создания хэша. Далее мы указали секретное кодовое слово для хэша. Если в последнем параметре нашей функции будет поставлено значение равное истине, то мы получим результат в виде необработанных двоичных данных. А если же будет значение равное ложь - то получим данные в шестнадцатерибричной кодировке.

Давайте теперь усложним наш пример и сделаем полноценное шифрование определенных данных. Вот код:

```
<?php
$key = hash("sha256", "this is a secret key");
$input = "Let us meet at 9 o'clock at the
secret place.";

$td = mcrypt_module_open("rijndael-128", "",
"cbc", "");
$iv =
mcrypt_create_iv(mcrypt_enc_get_iv_size($td),
MCRYPT_DEV_URANDOM);
crypt_generic_init($td, $key, $iv);
$encrypted_data = mcrypt_generic($td, $input);
mcrypt_generic_deinit($td);
mcrypt_module_close($td);
?>
```

Вы можете заметить переменную `$input`, эта переменная это данные, которые мы хотим с вами зашифровать. Дальше у нас идет переменная, `$td` внутри которой мы выбираем алгоритм шифрования, а также режим блочного шифрования для данного алгоритма. В нашем примере мы выбрали алгоритм `rijndael-128`, ключ имеет длину 128 битов, это можно понять из названия самого алгоритма. А также я думаю, вы заметили здесь, что мы выбрали режим блочного шифрования CBC.

После этого идет вектор инициализации который генерируется с помощью функции `mcrypt_create_iv()`. И только после того как мы сгенерировали все нужные для нас компоненты мы начинаем шифрование с помощью команды `mcrypt_generic_init()`. Внутри этой функции мы как раз и указали сгенерированные нами компоненты для шифрования. После этого мы создали переменную, `$encrypted_data` – внутри которой содержаться зашифрованные данные.

И в конце скрипта мы удаляем компоненты, которые использовались для шифрования. Конечно же, когда вы перейдете на страницу с данным скриптом то вы не увидите никакого результата, поскольку мы не воспользовались ни одной из команд для вывода данных. Вы можете сделать вывод данных с помощью одной из команд для вывода данных, а точнее команд для вывода данных об переменных. Что ж, какой мы можем сделать вывод из того что мы с вами сделали?

Во-первых, мы научились, как можно устанавливать расширения. Во-вторых, мы научились использовать шифры, а также можем объяснить их работу и классификацию. Во-третьих, мы можем реализовать шифрование данных с помощью РНР. Мы очень много чего с вами сделали в данном подразделе, и я считаю, что настало время переходить к следующему подразделу. Я также рекомендую читателю заглянуть в официальную документацию по расширению Mcrypt.

Сделайте несколько примеров, и рассмотрите какие есть функции в данном расширении и обязательно сделайте практику с ними. Следующий подраздел будет об шифровании паролей в РНР. А также об хранении паролей в базе данных в зашифрованном виде. Криптография – это очень сложная тема для обсуждения на конференциях и не только. Криптографию сложно усвоить, поскольку криптография это дело нашей с вами безопасности и поэтому все надо очень четко рассмотреть и создать чтобы система была безопасной.

13.5 Хэширование паролей

Простым примером того для чего нужно хэширование/шифрование паролей в вашем приложении является ситуация которая случилась в 2016 году с приложением Твиттер. В этом 2016 году с базы данных Твиттера было украдено 30 миллионов аккаунтов, но проблема заключается в том что пароли хранились в незашифрованном виде.

Если бы пароли хэшировались то такой ситуации не было бы. И компания не понесла бы убытки. Популярными алгоритмами для хэширования паролей являются md5 и sha256. Но вообще в PHP 5.5 для этого были добавлены отдельные функции, которые позволяют хэшировать пароли. Давайте рассмотрим с вами простой пример хэширования пароля с использованием функции `password_hash()` :

```
<?php
$password = "Hello";
$hash = password_hash($password,
PASSWORD_DEFAULT);
print_r($hash);
?>
```

Сохраните данный скрипт и протестируйте его в браузере. Вы сможете увидеть вывод данных хэшированного пароля. Некоторые читатели смогли здесь заметить, что в функции мы указали вторым аргументом функции константу `PASSWORD_DEFAULT`, которая делает хэширование по умолчанию. Когда идет хэширование пароля, то к паролю добавляются фальшивые данные которые называются `salt`(соль). Это подвышает безопасность и позволяет обмануть злоумышленника.

Если мы хотим проверить был ли введенный пароль пользователем верно, то надо просто взять хэш и пароль из базы после чего просто проверить этих два элемента с помощью функции `password_verify()`:

```
<?php
if (password_verify($password, $hash)) { //
Если получим значение которое равно истине то значит
пароль введен верно
    ...
} else {
    ...
}
?>
```

Если у вас будет такая ситуация что вам понадобится узнать какие-то данные об хэшированном пароле то вам надо будет воспользоваться функцией `password_get_info()`, которая выводит данные в виде массиве с тремя элементами:

- `algo` – алгоритм который использовался для хэширования.
- `algoName` – название алгоритма который использовался для хэширования.
- `options` – опции которые использовались при хэшировании.

Если нам понадобится изменить параметры `salt`, то мы должны воспользоваться функцией `password_needs_rehash()`:

```
<?php
if (password_needs_rehash($hash, PASSWORD_DEFAULT,
["cost"] => 12)) {
    $hash = password_hash($password,
PASSWORD_DEFAULT, ["cost"] => 12);
    ...
} else {
    ...
}
?>
```

В этом коде вы можете увидеть то, как мы изменением параметры `salt`, меняя длину данного параметра. Это был подраздел об хэшировании паролей, это важно сейчас. Тема криптографии сейчас очень актуальна и поэтому криптографию стоит изучать разработчику. Я уже об этом говорил. Сейчас используются множество библиотек и фреймворков которые в несколько раз облегчают жизнь программисту.

Сейчас приложения не пишутся с нуля, у них уже есть готовый «каркас» и программисту остается только строить своё приложение на данном каркасе.

Вопросы и упражнения для самоконтроля

1. Какие существуют методы безопасности в PHP?
2. Какие есть виды атак на приложение?
3. В какой способ злоумышленник может навредить базе данных?
4. Для чего нужна фильтрация данных?
5. Какие есть методы/функции для фильтрации данных?
6. Напишите скрипт который не допускал присутствия цифр в поле формы которая имела бы два поля.
7. Что такое криптография и для чего она нужна?
8. Расскажите немного об истории криптографии. (Необязательно)
9. Какие существуют виды шифров а также расскажите об их разнице и об их работе.
10. Что такое блочное шифрование и потоковое шифрование? А также в чем разница этих двух видов?
11. Расскажите об часто используемых шифрах.
12. Для чего нужны ключи шифрам?
13. Напишите код с который шифровал такие данные «Hello world!».
14. Какие есть способы хэширования паролей?
15. Какие алгоритмы применяются для хэширования паролей?
16. Зачем нужно хэшировать пароли?
17. Какие есть функции для хэширования паролей?

14 Работа с другими веб-службами

В этом разделе речь пойдет уже о том как можно работать с другими веб-службами с помощью Прикладного-Программного-Интерфейса или более правильное название API(Application-Program-Interface). Именно с помощью API мы можем авторизоваться с помощью любой соц. сети в любом приложении, где это программным путем поддерживается. Например, может быть веб-приложение, где можно авторизоваться с помощью своего аккаунта Твиттера.

В этом случае, для авторизации будет использована API Твиттера, которое было создано разработчиками самого Твиттера. Или если, например, мы захотим с вами авторизовать в каком то приложении с

помощью аккаунта в Facebook, то здесь уже будет использовано Facebook API.

То есть многие сервисы сами создают свою API для того чтобы им могли пользоваться другие разработчики чтобы ввести какие то дополнительные сервисы, чтобы взаимодействовать с сервисом который и дает данное API. Перед тем как мы реально начнём работать с API какого-то сервиса, то мы должны понять, что такое JSON, а потом уже научиться делать запросы на получение JSON данных с помощью PHP. Поэтому первый подраздел этого раздела будет об JSON.

14.1 Что такое JSON?

Аббревиатура JSON расшифровывается как JavaScript-Object-Notation. Т.е. нотация JavaScript объектов, это не язык программирования. JSON это текстовый формат обмена данными между компьютерами, а также в сети. JSON очень легко позволяет передавать определенные данные в сети. Кроме JSON'а данные можно также передавать в формате XML. Но мы с вами будем говорить именно об JSON'е. В PHP есть специальные функции для того чтобы кодировать и декодировать JSON информацию и потом получать её в виде массива.

Давайте с вами рассмотрим небольшой пример массива JSON-данных:

```
{
    "firstName": "Mike",
    "lastName": "Freeman",
    "city": "Amsterdam",
    "job": "Web-developer",
    "contactData": [
        "phoneNumber": "300231134",
        "e-mail": "mikefree@cool-app.com",
    ]
}
```

Теперь давайте я вам объясню, что из себя представляют из себя эти данные. Во-первых, вы видите, что мы как бы сказать объявляем массив внутри которого как раз и находятся JSON-данные. Эти JSON-данные мы можем передать или просто хранить. Здесь вы можете увидеть первые элементы это `firstName`, `lastName`, `city`, `job` и массив `contactData`. Думаю вы уже заметили что здесь есть ключ и значение к этому ключу. Заметьте то как создаются эти элементы, а также с помощью какого оператора делается присвоение данных для этих ключей. (Подсказка - с помощью оператора :).

Элемент `contactData` это массив который содержит элементы из данными для контакта. Для примера мы будем работать с Star Wars API чтобы понять то как все работает. Попробуйте создать какие то свои JSON данные. Т.е. создайте свой массив, в котором вы сами будете добавлять свои объекты и сами выбирать что там должно быть. Вот ещё один пример, чтобы больше закрепить материал:

```
{
    "name": "PHP Junior Kit",
    "description": "Book about programming
with PHP",
    "author": "Demian Kostelny"
    "tags": {
        "Programming",
        "PHP",
        "Web-development"
    }
}
```

Здесь вы уже видите массив `tags` который имеет в себе просто элементы, но не элементы которые имеют ключ и значение. Сделайте несколько простых файлов с JSON данными, чтобы большие закрепить материал. А теперь мы переходим к следующему подразделу, в котором мы будем работать с JSON и PHP.

14.2 JSON и PHP

Для того чтобы работать в PHP с JSON как я уже и говорил есть специальные функции. Давайте представим ситуацию, когда нам надо передать определенные данные по сети и нам надо создать массив с JSON данными. Чтобы создать массив с JSON данными в PHP нам для начала надо просто создать массив в PHP который, содержал бы в себе элементы JSON массива:

```
<?php
$json_data = array(
    "firstName" => "Mike",
    "lastName" => "Freeman",
    "job" => "Web-developer",
    "contactData" => array(
        "phoneNumber" => "3001300",
        "e-mail" => "mikefree@cool-app.com"
    )
);
?>
```

Как вы можете увидеть, то здесь мы просто перевели JSON данные в простой PHP массив. Чтобы закодировать этот массив в JSON данные надо воспользоваться функцией `json_encode()` :

```
<?php
$json_data = array(
    "firstName" => "Mike",
    "lastName" => "Freeman",
    "job" => "Web-developer",
    "contactData" => array(
        "phoneNumber" => "3001300",
        "e-mail" => "mikefree@cool-app.com"
    )
);
```

```
$encoded = json_encode($json_data);
print_r($encoded); // Заметьте разницу между
ЭТИМ ВЫВОДОМ ДАННЫХ
print_r($json_data); // И ЭТИМ ВЫВОДОМ ДАННЫХ
?>
```

Результатом этого кода будет такая строка в виде JSON-данных:

```
{"firstName": "Mike", "lastName": "Freeman",
"job": "Web-developer", "contactData":
{"phoneNumber": "3001300", "e-mail": "mikefree@cool-
app.com"}}
```

В такой способ можно брать кодировать простые массивы в JSON данные, после чего брать их и передавать по сети. А если нам надо будет декодировать JSON данные, то для этого надо воспользоваться функцией `json_decode()`:

```
<?php
...
print_r(json_decode($encoded, true)); // Здесь
мы декорируем закодированные JSON-данные.
```

// Второй параметр этой функции надо ставить если декодированные данные были возвращены в виде массиве.

```
?>
```

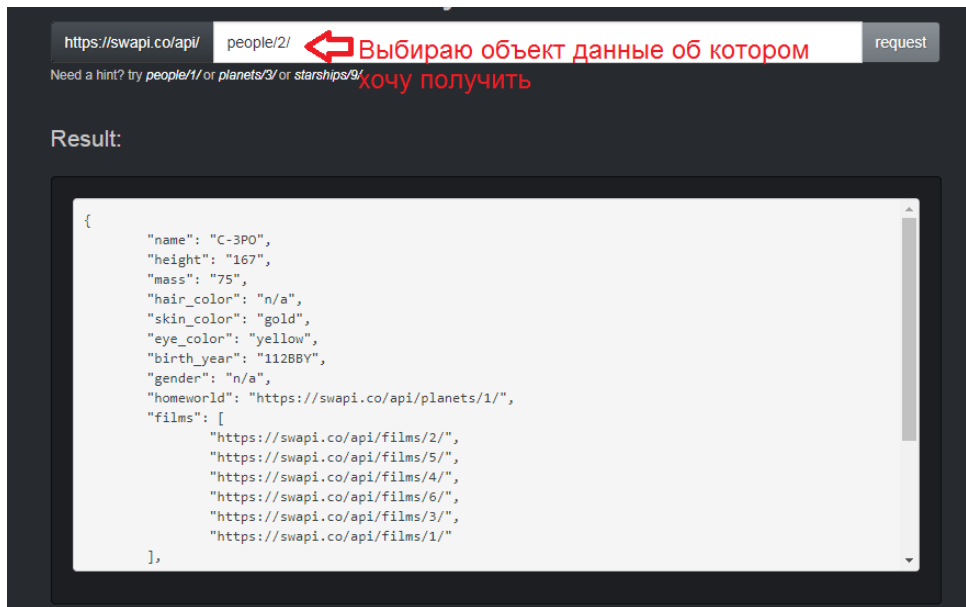
Вывод будет в виде данных об массиве:

```
Array
(
    [firstName] => Mike
    [lastName] => Freeman
    [job] => Web-developer
    [contactData] => Array
        (
            [phoneNumber] => 3001300
```

```
[e-mail] => mikefree@cool-app.com  
    )  
)
```

Это всего лишь данные об массиве. Только что вы выучили две функции которые позволяют кодировать и декодировать JSON данные. Эти функции играют важную роль, когда мы будем отправлять или получать данные в формате JSON. А теперь давайте поработаем с Star Wars API. Star Wars API - это веб-сайт, который позволяет обучиться работать с JSON на простых практических примерах. Данные которые вы будете получать от Star Wars API это например информация об героях фильма, планетах которые были показаны в фильме и т.п.

То есть Star Wars API это по сути просто база данных об этом чудесном фильме, и суть в том чтобы на примере данных которые там есть научиться работать с JSON, а также немного привыкнуть к работе с API. Вы можете прямо сейчас перейти на веб-сайт Star Wars API по адресу: <https://swapi.co>. И прямо на веб-сайте вы сможете уже сделать запросы на чтобы протестировать API. Вот, как я получил данные С-ЗРО на swapi.io:



- Рис. 36 - Пример вывода результата после запроса на swapi.io

Сначала я ввел в адресной строке объект который я хочу получить. У меня это была группа `people`, после чего я указал номер героя об котором я хочу получить информацию. Далее я нажал на кнопку “request”, после чего получил данные в формате JSON. Если, например, вы хотите получить данные об четвертом герое фильма, то тогда надо просто указать его цифру, например вот так `people/4/`.

Можете также зайти в документацию SWAPI, и посмотреть какой элемент в массиве данных за что отвечает. Сразу ясно что когда мы делаем запрос на получение данных об каком то персонаже, то например элемент `name` будет отвечать за имя персонажа. А теперь давайте поговорим о том, как можно делать запросы на получение данных с помощью PHP. Есть функция имеющая название `file_get_contents()`, данная функция, позволяет получить содержимое страницы определенного сайта и отобразить через

написанный PHP скрипт. Вот пример получения содержимого страницы SWAPI с помощью использования функции `file_get_contents()`:

```
<?php
$url = file_get_contents("https://swapi.co");
echo $url;
?>
```

После выполнения данного скрипта вы получите содержимое главной страницы SWAPI через ваш скрипт. Функция вообще имеет 5 параметров. Но нас будет интересовать именно те параметры которые позволяют доставать данные.

С помощью третьего параметра функции мы можем определить параметры запроса и сделать тем самым запрос более динамическим. Например, вот скрипт где есть переменная `$context`, с помощью которой мы как раз и задаем больше параметров для запроса, что позволяет его сделать более динамичным как я уже и говорил:

```
<?php
$options = array(
    "http" => array(
        "method" => "GET",
        "header" => "Accept-language: en\r\n"
    )
);

$context = stream_context_create($options);
$file = file_get_contents("https://swapi.co",
false, $context);
?>
```

Здесь, в этом примере мы указали параметры заголовка для HTTP запроса. Об том что это такое мы уже с вами говорили. В переменной `$context` мы создаем поток данных, после этого с помощью переменной `$file` мы достаём содержимое главной страницы SWAPI только уже с заданными нами параметрами для заголовков.

Второй параметр функции `file_get_contents()` может иметь только одно из двух логических значений. Второй параметр позволяет делать поиск файла, в нашем случае мы ничего не ищем, и поэтому ставим значение, которое равно ложь. Это как раз один из способов получения данных. А теперь давайте получим данные об одном из героев в формате JSON. Поэтому для этого мы возьмем такие параметры для запроса:

```
<?php
$url = "https://swapi.co/people/4/";
$options = array(
    "header" => "Content-Type:
application/json");
);
$context = stream_context_create("http" =>
$options);
echo file_get_contents($url, false, $context);
?>
```

Обратите внимание на то что мы указали что мы хотим получить **ответ** именно в **формате JSON-данных**. В такой способ мы задали заголовок, после чего выполнили запрос и получили данные в формате JSON. Конечно же можно было присвоить переменную полученным данным после чего взять и декодировать их для использования. Но для того чтобы получить какие-то JSON данные через сеть можно также воспользоваться расширением cURL.

cURL имеет свой набор функций которые мы можем использовать для того чтобы работать с запросами. Например, вот как можно получать данные об герое по номер 5 на SWAPI с помощью cURL:

```
<?php
$curl = curl_init(); // Запускаем переменную
для которой будет использован cURL

$url = "https://swapi.co/people/5";
```

```
curl_setopt_array($curl, array(
    CURLOPT_URL => $url,
    CURLOPT_HTTP_VERSION =>
CURL_HTTP_VERSION_1_1,
    CURLOPT_CUSTOMREQUEST => "GET",
    CURLOPT_HTTPHEADER => array( "content-
type: application/json", ),
));

$response = curl_exec($curl); // Получаем ответ
после запроса в виде JSON данных
$error = curl_error($curl); // Если будет ошибка
то данная переменная получит её значение
curl_close($curl); // Завершаем запрос

if ($error) {
    exit($error);
} else {
    echo $response;
}

?>
```

Теперь я пожалуй объясню как работает этот немного сложный код. Во-первых, мы создали переменную `$curl` которая использовалась для создания запроса. Дальше мы задали все параметры для запроса с помощью функции `curl_setopt_array()`.

Внутри функции указаны такие параметры как:

1. URL к которому мы делаем запрос.
2. Версия HTTP протокола использующая в запросе.
3. HTTP-метод использующий в запросе.
4. Параметры для заголовка HTTP-запроса.

Дальше после указания всех этих параметров мы получаем ответ функции в переменную `$response` с помощью функции `curl_exec()`. Точнее мы импортировали ответ в эту переменную с помощью этой функции. Если у нас будет ошибка, то переменная `$err` получит её значение с помощью вызова функции `curl_error()`. В конце мы завершаем наш запрос с помощью функции `curl_close()`.

И уже дальше идет логическая конструкция суть которой заключается в том что мы просто выводим либо ошибку(если она у нас есть), либо мы выводим ответ полученный после успешного запроса.

Это также способ, с помощью которого можно получать данные. Но важно также сказать, что данные можно не только получать с помощью всех этих функций, но также и отправлять их. Если запрос прошёл успешно и вы получили ответ в переменную `$response`, то тогда можно декодировать информацию которая находится в этой переменной, а все потому что ответ от SWAPI мы получили в виде JSON-данных.

Когда мы декодируем ответ, то мы сможем обращаться к этим данным, только они будут представлены в виде простого массива. Конечно же можно и отправлять данные с помощью cURL. Только тогда уже надо поменять HTTP-метод который используется в запросе, а также указать в параметрах для запроса какие именно данные надо отправить в формате JSON. Вот код-пример, в котором идет отправка данных на локальный хост:

```
<?php
$curl = curl_init();

$url = "localhost/test/index.php";
// Конечно же данный код работал бы если
реально существовала такая папка которая имела
бы этот файл.
```


// И кроме этого могла также обрабатывать все эти JSON запросы.

```
$arr = array(
    "title" => "Super Post",
    "info" => "It's super nice and interesting
post",
    "authors" => array(
        "SuperUser1234",
        "Gordon",
        "RabbitMan",
    )
);

curl_setopt_array($curl, array(
    CURLOPT_URL => $url,
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTP_VERSION =>
    CURL_HTTP_VERSION_1_1,
    CURLOPT_CUSTOMREQUEST => "POST", //
Заметьте что здесь уже другой метод HTTP
запроса
    CURLOPT_POSTFIELDS => json_encode($arr),
// Передаем массив из данными которые хотим
передать в формате JSON
    CURLOPT_HTTPHEADER => array("content-type:
application/json"),
));

$response = curl_exec($curl); // Получаем ответ
$error = curl_error($curl);
curl_close($curl); // Завершаем запрос
```

```
if ($err) {
    exit($err);
} else {
    echo $response;
}
?>
```

Вы можете здесь заметить новый параметр в массиве для параметров cURL, это `CURLOPT_POSTFIELDS`. В него мы передали массив данных, который мы сразу же кодируем в JSON данные. Сразу хочу сказать что это все просто пример того как можно передавать данные по сети в формате JSON. Поэтому не пробуйте тестировать код, поскольку будет ошибка об том что такой то и такой то файл не найден. Помните как мы с вами работали с HTTP через ARC? Так вот, теперь вы можете попробовать протестировать SWAPI внутри этого приложения и добавить даже что то свое в HTTP запрос.

А давайте сейчас вернемся к нашей функции `file_get_contents()`. Кроме того чтобы с помощью этой функции можно доставать данные, то можно также и отправлять данные прямо как в cURL. Это даже логически выходит, поскольку остается только заменить метод и добавить поле с данными, которые мы хотим отправить.

Вот например код в котором показано отправление данных с помощью `file_get_contents()`:

```
<?php
$data = array(
    "book" => "PHP Junior Kit",
    "readed" => "27.07.2018",
    "user" => "CodeMan"
);

$options = array(
```

```
"method" => "POST", // Выбираем метод для
запроса
"header" => "Content-Type:
application/json",
"content" => json_encode($data) //
Указываем данные которые хотим отправить
);

$context = stream_context_create(array("http"
=> $options));

$url = "localhost/test/test.php";

echo file_get_contents($url, false, $context);
?>
```

Объяснять весь этот код не буду поскольку вы уже встречались с похожим в начале этого подраздела. Все важные моменты закомментированы и можете их прочитать, если вы что то не поняли. Кроме того что с помощью таких запросов вы можете получать содержимое страницы, вы можете также получать Cookies.

На этом я завершу этот краткий раздел и мы сделаем заключение.

Заключение

В этом разделе мы с вами изучили, что такое JSON и как можно делать запросы. Обмениваться данными с помощью JSON легко, но надо не забывать об безопасности и шифровании если вы будете передавать какие-то важные данные. Хотя конечно же если это какие-то очень секретные данные то JSON лучше не использовать, а использовать другие способы для того чтобы отправлять и шифровать безопасно данные.

Я завершу этот раздел, и конечно же, как и в каждом конце раздела попрошу читателя попрактиковаться с материалом который был

показан в этом разделе. А лучше ответить на вопросы и выполнить упражнения которые показаны ниже.

Вопросы и упражнения для самоконтроля

1. Что такое JSON и для чего он нужен?
2. Какие есть способы передачи данных с помощью JSON?
3. Найдите другие ресурсы на которых вы сможете попрактиковаться работать с JSON. То есть такие как например Star Wars API.
4. Зачем нужны первых три параметра функции `file_get_contents()`?
5. Как задаются параметры для HTTP-запроса в функции `file_get_contents()`?
6. Как выглядит массив в массив в JSON'е?
7. Как инициализировать объект для cURL?
8. Для чего нужен параметр `CURLOPT_POSTFIELDS` в функции `curl_setopt_array()`?
9. Как вы сделаете так, чтобы в HTTP запросе передавались именно JSON данные?

15 Ошибки в коде

Часто когда вы пишете программы и потом их запускаете, то может вылететь ошибка. Когда вы будете работать с какими то большими проектами, то часто вы будете получать не одну ошибку. И анализировать данные по отдельности не очень эффективно, поэтому надо научиться управлять ошибками в коде чтобы выводились все нужные данные для анализа, вместо того чтобы делать это вручную.

Поэтому речь в этом разделе пойдет об том как управлять ошибками, а также выводит нужные данные об коде если у нас будет определенная ошибка.

15.1 Виды ошибок

Думаю, что читатель на протяжении данной книги уже получал какие-нибудь ошибки во время изучения программирования на PHP. Кроме того что мы можем с вами наблюдать за тем как выводятся ошибки, мы можем также с ними работать. Простым примером работы с ошибками являться возможность поменять место для вывода ошибок. Или например можно сохранять логи с ошибками которые выводятся. Это полезно тогда когда нам надо хранить историю об ошибках. Можно также отключить показ ошибок, если будет потребность в этом.

Вывод ошибок можно настроить через файлы конфигурации браузера. И запись логов также можно сразу же включить через файлы конфигурации браузера. Но давайте поговорим, о том, какие существуют виды ошибок, поскольку на протяжении этой книги я не говорил какие именно существуют ошибки.

- **Синтаксическая ошибка.** Ошибка такого типа часто возникает когда вы например забыли поставить точку с запятой в конце команды. Или например вы забыли поставить фигурную скобку в логической конструкции.
- **Фатальная ошибка.** Очень серьезные ошибки которые могут возникнуть, например, при вызове несуществующей функции. Программа конечно же от такой ошибки останавливает свою работу.
- **Предупреждение.** Это уже не ошибка, а что то в роде небольших рекомендаций, которые мы можем получить от PHP. При предупреждении программа не останавливает свою работу. Простым примером предупреждения может быть ситуация когда мы забыли добавить все параметры в вызываемую функцию.
- **Замечание.** Это также свое родная рекомендация, которая возникает, например, когда мы хотим вывести переменную которая ещё не была создана.
- **Строгое предупреждение.** Возникает, тогда когда мы хотим воспользоваться устаревшими средствами. Например, устаревшей функция, которая уже не используется.

Кроме этих видов ошибок, есть также несколько констант ошибок определенных в PHP:

- `E_ERROR` - тип фатальных ошибок. При такой ошибке работа программы сразу прекращается.
- `E_WARNING` - предупреждение для программиста, выполнение программы не останавливается.
- `E_PARSE` - ошибка возникающая во время компиляции программы.
- `E_USER_WARNING` - замечание для пользователя. Не представляет из себя какую то критичную ошибку.

- `E_COMPILE_WARNING` - предупреждение которое создается при компиляции.

Вот такие есть виды ошибок, которые вам надо знать. Переходим к самому управлению ошибками в PHP. Но перед этим скажу, что самым простым способом устранения синтаксических ошибок это использовать редакторы кода. Как например Sublime Text о котором я говорил в начале книги. Конечно же есть намного мощнее инструменты которые в несколько раз расширяют возможности. Например, можно также воспользоваться очень популярным и хорошим, но платным IDE - Web Storm.

15.2 Управление ошибками

Для начала нам надо включить отслеживание ошибок с помощью функции `error_reporting()`, для того чтобы мы могли видеть любые ошибки:

```
<?php
    error_reporting(E_ALL); // Как вы видите,
то здесь мы указали что мы хотим отслеживать все
ошибки
?>
```

Теперь мы сможем видеть любые ошибки. Хорошо, а теперь переходим к самому управлению. Первая функция с которой мы с вами познакомимся это будет `set_error_handler`, которая устанавливает пользовательский обработчик ошибок.

Функция имеет 5 параметров. Внизу показан список этих параметров:

- `errno` - тип ошибки, указывается в виде числа.
- `errstr` - содержит сообщение об ошибке.
- `errfile` - необязательный параметр, который содержит в себе имя файла, в котором произошла ошибка.

- `errline` - необязательный параметр, содержащий номер строки в котором произошла ошибка.
- `errcontext` - также необязательный параметр, содержащий переменные в области видимости где произошла ошибка.

Самым простым примером использования этой функции, является функция, которая использует данную функцию для создания обработчика ошибок, вот похожий код с примером который можно найти также в документации разработчиков:

```
<?php
    error_reporting(E_ALL); // Включаем отображение
всех ошибок
    ini_set('display_errors', 1);

    // функция для создания обработчика ошибок
    function myHandler($level, $message, $file,
$line, $context) {
        // в логической конструкции определяем вывод
данных в зависимости от вида ошибки
        switch ($level) {
            case E_WARNING:
                $type = 'Warning';
                break;
            case E_NOTICE:
                $type = 'Notice';
                break;
            default;
                // Если уже нету никакой ошибки, то
передам работу самому PHP
                return false;
        }
        // выводим содержание ошибка
```



```

        echo "<h2>$type: $message</h2>";
        echo "<p><strong>File</strong>:"
$file:$line</p>";
        echo "<p><strong>Context</strong>: $".
join(',', $', array_keys($context))."</p>";
        return true;
    }

```

```

// регистрируем наш обработчик ошибок в систему с
помощью функции set_error_handle
set_error_handler('myHandler', E_ALL);
?>

```

То есть, как вы смогли понять, то данная функция нужна для того чтобы создать свой обработчик ошибок. Есть небольшая проблема с данной функцией. Заключается она в том, что функция не отображает фатальной ошибки. И поэтому чтобы мы могли отобразить свой текст при фатальной ошибке надо воспользоваться функцией `register_shutdown_function()`:

```

<?php
function shutdown_message()
{
    echo "Данный текст всегда будет выводиться
при ошибках";
}

register_shutdown_function("shutdown_messa
ge");
// Добавляем название функции которая будет
выводит данные
?>

```

Хорошо, но что же будет дальше? Дальше я хочу вам продемонстрировать функцию `error_get_last()`, которая позволяет

вывести последнюю полученную ошибку. Пример использования функции:

```
<?php
function shutdown_message()
{
    $err = error_get_last();
    echo "Предыдущая ошибка: " . $err;
}

register_shutdown_function("shutdown_message");
?>
```

Но разве это все функции которые позволяют обрабатывать ошибки? Конечно же нет, вы можете зайти на официальную документацию и посмотреть какие ещё есть функции для обработки ошибок. Все это можно найти по адресу <http://php.net/manual/ru/book.errorfunc.php>. Вы научились как можно сделать свой обработчик ошибок. Это может облегчить работу с кодом, когда вы будете разрабатывать свое большое приложение.

15.3 Отладка кода

В этом подразделе мы уже с вами поговорим то как можно отлаживать код и просматривать отладочную информацию. Отладка кода позволяет более детально описать работу программы в логах, или в выводах консоли. Существует много способов представления отладки кода. Отладка кода не только позволяет с легкостью определять ошибки, но также и искать баги в коде. Так что давайте приступим!

Что же может нам дать данные об переменной? Мы с вами знаем целых две функции, которые могут вывести такую информацию, это `var_dump` и `print_r`. Например, если у нас с вами будет какой-то

массив то можно сделать вывод информации об этом массиве, тем самым узнав больше об его элементах:

```
<?php
$some_data = array(
    "Band" => "Kraftwerk",
    "Since" => "1970",
    "Genre" => "Electronic music",
    "Members" => array(
        "Ralf Hutter",
        "Henning Schmitz",
        "Fritz Hilbert",
        "Falk Grieffenhagen"
    )
);

print_r($some_data);
?>
```

Вот результат выполнения программы:

```
Array
(
    [Band] => Kraftwerk
    [Since] => 1970
    [Genre] => Electronic music
    [Members] => Array
        (
            [0] => Ralf Hutter
            [1] => Henning Schmitz
            [2] => Fritz Hilbert
            [3] => Falk Grieffenhagen
        )
)
```

Как вы можете увидеть, то мы не только получаем данные об элементах массива, но также и структуру массива. Но думаю некоторые смогли здесь заметить небольшую проблему. И проблемой является то что не выводятся типы данных строк, и мы не можем понять точно что здесь является строкой, а что имеет числовой тип данных.

И для решения этой проблемы, самым простым способом будет использовать функцию `var_dump()`, поскольку она выводит все данные об элементах, их типы данных и длину символов. Вот как будет выглядеть результат вывода с использованием функции `var_dump()` в предыдущем примере:

```
array(4) {  
    ["Band"]=> string(9) "Kraftwerk"  
    ["Since"]=> string(4) "1970"  
    ["Genre"]=> string(16) "Electronic music"  
    ["Members"]=> array(4) {  
        [0]=> string(11) "Ralf Hutter"  
        [1]=> string(15) "Henning Schmitz"  
        [2]=> string(13) "Fritz Hilbert"  
        [3]=> string(18) "Falk  
Grieffenhagen"  
    }  
}
```

Сразу же видно, какая разницу между этими двумя выводами результатов. Точнее разницы между выводом данных функций `print_r()` и функцией `var_dump()`. Что ж проблема решена, что же будет идти дальше? С версии PHP 5.6 теперь вместе с самым языком программирования поставляется также отладчик кода `phpdbg`. Он сразу же установлен в PHP как расширение. Конечно же можно использовать также другой отладчик, который создан также самими разработчиками Zend. Но для того чтобы работать с отладчиком очень рекомендовано использовать IDE - PHP Storm. Поскольку он хорошо позволяет отслеживать работу программы.

Но хочу сказать **важную информацию**, о том что phpdbg уже не используют, и вообще данное расширение умерло, т.е. его уже не используют. Зато появилось намного эффективное расширение - xDebug. И сейчас мы об нем поговорим. Для начала нам надо будет установить xDebug, и сделать это можно только через расширение Pecl. Для этого введите в командную строку такую команду:

```
pecl install xdebug
```

После этого у вас должны установиться файлы данного расширения. Следующим шагом будет подключение расширения. В разделе об криптографии мы уже с вами рассматривали то как можно подключить расширение к PHP через файл `php.ini`. Если кто то не помнит, как это делать, то вы просто указываете путь к расширению с помощью функции `include_path`.

Конечно же есть и другой способ установки данного расширения. Для этого надо перейти на страницу установки xDebug по адресу <https://xdebug.org/download.php>, после чего просто скачать `.dll` файл и подключить его через функцию `extension()` в файлу конфигурации PHP - `php.ini`. Дальше после всего этого давайте сделаем тестирование нашего расширения, для этого создадим новый скрипт с таким кодом:

```
<?php
$arr = array(
    "Name" => "Nice Book",
    "Author" => "Super Author",
    "Year" => "2018"
);

var_dump($arr);
?>
```

Думаю читатель сразу же задастся вопросом “Что? Мы же делали уже подобную вещь не раз, зачем это?”. А вот что я скажу, просто

запустите этот код и посмотрите на результат. И что? Вы видите здесь разницу с выводом данных?

Вот первое что сделало данное расширение. У нас теперь улучшенный вывод данных с помощью функции `var_dump()`. Теперь вы можете увидеть информацию об области видимости и также много чего другого. Но кроме того что теперь у вас улучшенный вывод данных, с помощью функций для отладки кода - то теперь вывод ошибок также улучшен. И конечно вы можете получить намного больше информации об работе скрипта.

Например, вот какой будет вывод ошибки, если мы пытаемся вывести на экран переменную которая не была объявлена:

(!) Notice: Undefined variable: foo in /home/vagrant/Code/xdebug2/index.php on line 3				
Call Stack				
#	Time	Memory	Function	Location
1	0.0006	357664	{main}()	.../index.php:0

Изображение было взято с веб-сайта habr.

Кроме того что ошибки выводятся с более детальной информацией. Можно также получить информации об том, сколько РНР использует оперативную память. Данное расширение это намного лучше использования разных функций для того чтобы выводить информацию об коде. На этом моменте я хочу завершить данный раздел и переходить к следующему. Но перед этим мы должны сделать небольшое заключение об том что мы выучили в данном разделе.

Заключение

Написать программу безошибочно невозможно. Каждый имеет право на ошибку. И во время разработки своего веб-приложения вы возможно частенько будете делать ошибки, и это нормально. На ошибках которые вы получаете во время написания программы - вы учитесь. Конечно же намного лучше учиться на чужих ошибках.

Уметь управлять ошибками в коде хорошо, и это можно использовать для того чтобы было легко выводить данные об коде. И с помощью таких расширений как xDebug, с легкостью можно найти ошибки в коде. Как и в конце каждого раздела, я попрошу читателя чтобы он ответил на вопросы и выполнил упражнения которые находятся в конце данного раздела.

Вопросы и упражнения для самоконтроля

1. Какие могут быть причины возникновения ошибок в коде?
2. Какие есть виды ошибок?
3. Зачем нужен вывод ошибок в PHP?
4. Какие есть функции для управлением ошибок?
5. Как можно создать свой обработчик ошибок? Создайте свой обработчик ошибок который выводил бы отдельный текст к каждой ошибке.
6. Напишите программу которая выводила бы определенный текст при любой ошибке.

16 Пакетный менеджер

В этом разделе мы с вами поговорим об том как можно управлять пакетами. А если более ясно, то как можно с легкостью и с небольшой тратой времени устанавливать разные библиотеки и фреймворки. Это позволяет разработчику сэкономить время также на выполнение таких рутинных задач как настройка веб-приложения созданного на фреймворке с помощью пакетного менеджера.

Мы с вами будем использовать самый часто использующий пакетный менеджер для PHP - Composer. Первый подраздел будет об том как установить данный пакетный менеджер, дальше мы с вами на простых примерах будем устанавливать библиотеки для того чтобы показать простоту использования пакетного менеджера.

16.1 Установка

Для установки нашего пакетного менеджера давайте перейдем на официальный веб-сайт `composer'a`, по такому адресу: <https://getcomposer.org/>. После того как вы перейдете вы сможете увидеть красивый логотип данного приложения. Перейдите на страницу установки: <https://getcomposer.org/download/>, на странице вы можете увидеть способы установки которые зависят от вашей операционной системы. Я использую Windows, поэтому сейчас

речь пойдет об том как устанавливать пакетный менеджер под эту операционную систему.

Если у вас какая-то другая операционная система, то просто установите по инструкции которая предназначена для вашей операционной системы. Для того чтобы установить Composer под Windows, вам не придется брать и устанавливать это через командную строку. Достаточно будет просто скачать установщик по этой ссылке:

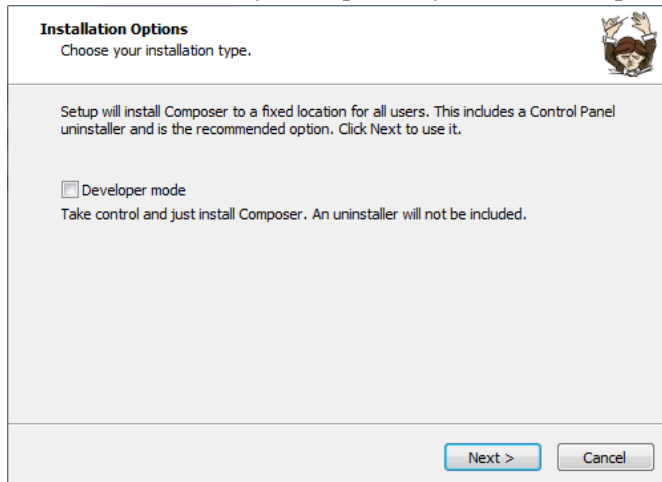
Windows Installer

The installer will download composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

Download and run [Composer-Setup.exe](#) - it will install the latest composer version whenever it is executed.

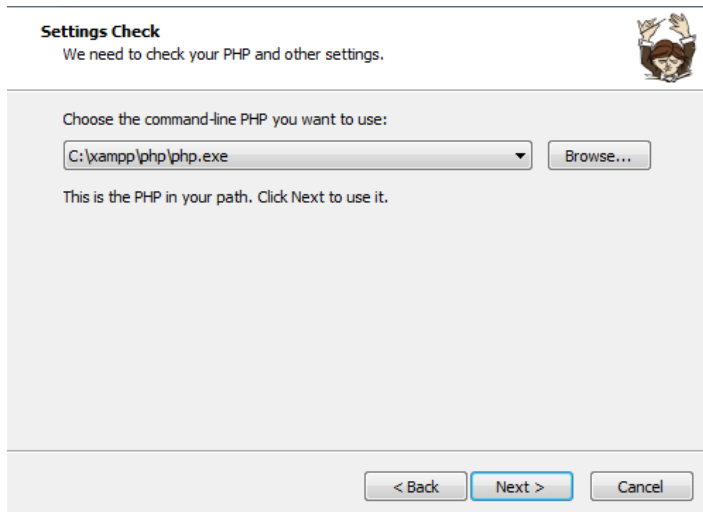
- *Рис. 37 - Ссылка на скачивание установщика пакетного менеджера Composer на официальном сайте пакетного менеджера.*

Скачайте установщик, после чего запустите его. После запуска установщика, вам будет предложено выбрать режим установки. Единственный доступный режим установки это режим разработчика:



- *Рис. 38 - Выбор режима установки пакетного менеджера в установщике Composer.*

Выбираем этот режим установки и двигаемся дальше. После этого установщику надо будет проверить версию РНР которую мы используем на локальном сервере. И поэтому мы должны будем указать путь к приложению `php.exe`:



- Рис. 39 - Указание пути приложения `php.exe` в установщике пакетного менеджера *Composer*.

После проверки РНР версии, установщик начнёт установку пакетного менеджера. И после того как вы установили сам `composer`, то надо будет его протестировать. Самым простым способом это сделать будет ввести команду для проверки версии пакетного менеджера:

```
composer -v
```

После этого выведётся большая надпись *Composer* сделанная с разных символов, и после нее версия самого пакетного менеджера. Ну и список с кучей разных команд которые можно использовать в *Composer*.

Сразу хочу сказать что когда вы будете устанавливать в своё приложение какие-нибудь библиотеки, то `composer` будет создавать отдельную папку `vendor` внутри которой будут находиться все папки и файлы библиотек. Каждая установленная библиотека/фреймворк будет

иметь отдельную папку, в которой и будут размещаться файлы библиотеки/фреймворка.

Кроме этого в корневой папке будет создан файл `composer.json`, внутри которого можно будет указать информацию об приложении. Также можно будет программным способом указать какие библиотеки должны будут установлены в приложении.

Вот как выглядит приблизительно файл `composer.json`:

```
{
    "name": "authername/my_app",
    "description": "Here is description",
    "authors": [
        {
            "name": "George Butaki",
            "email":
"georgel23@example.com"
        }
    ],
    "require": {
        "monolog/monolog": "1.12.0"
    }
}
```

Сначала мы указали имя приложения к которому мы хотим установить нужные для нас библиотеки. Оцените то, как мы с вами указывали название, во-первых сначала мы указали никнейм автора приложения, после слэша мы указываем само название нашего приложения. Дальше мы указали описание приложения в поле `description`. А дальше после этого элемента идет элемент `authors` внутри которого записаны данные об авторе приложения. Здесь вы можете увидеть имя автора, а также его e-mail.

Конечно же, можно указать какие-то дополнительные данные, как например веб-сайт автора, или какие-то страницы в соц. сетях которые принадлежат автору приложения.

В конце файла конфигурации вы можете увидеть массив `require` внутри которого записаны данные об тех библиотеках, которые должны быть подключены. В нашем случае здесь только одна библиотека и это `monolog`. В следующем подразделе речь пойдет о том, как правильно использовать данный пакетный менеджер.

16.2 Использование

Как было сказано, то в данном подразделе речь пойдет именно об использовании пакетного менеджера `composer`. Хорошо, представим себе ситуацию что нам надо установить библиотеку, которая бы с легкостью расширило бы функционал отправки электронных писем через PHP. Одной из самых использующих библиотек является библиотека `SwiftMailer`. Давайте её установим с помощью такой команды:

```
composer require swiftmailer/swiftmailer
```

С помощью этой команды как раз и установиться данная библиотека. Создаться папка `vendor` внутри которой как раз и будут храниться данные об библиотеках. Но важным моментом является то, что если вы например используете `Linux`, то данная команда может у вас не сработать. И в таком случае вам придется установить файл `composer`'а - `composer.phar`.

К которому вы можете обращаться чтобы установить какую - нибудь библиотеку. Обращаться к этому файлу вы будете через PHP, т.е. команда инсталляции `swiftmailer` в через такой способ будет выглядит вот так:

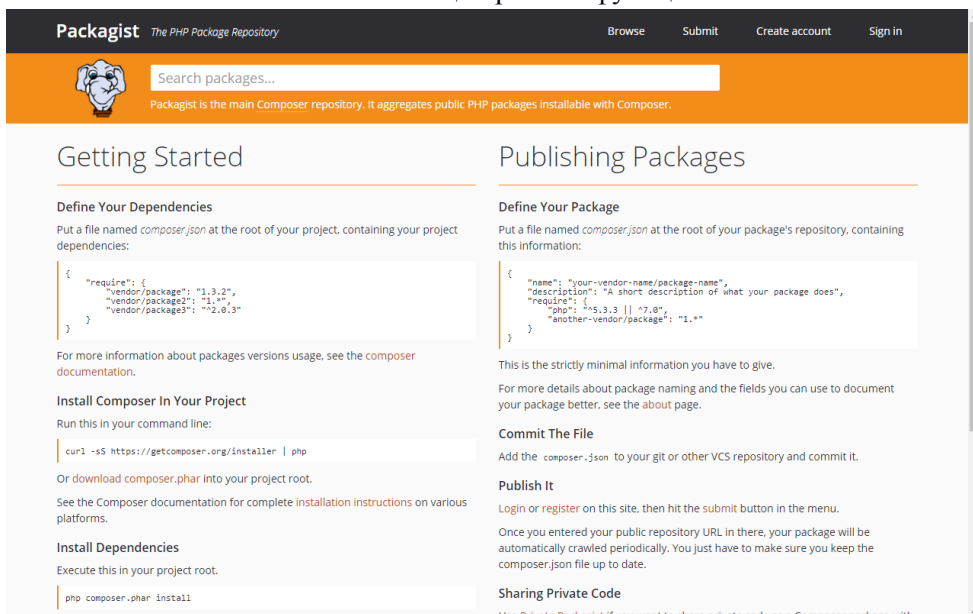
```
php composer.phar require  
swiftmailer/swifmailer
```

После этого, как я уже и говорил, будет создана папка с библиотекой, а также файлы с конфигурацией приложения. Хорошо, а что же дальше делать? Как например подключить данную библиотеку в PHP файл?

Все очень просто, надо просто подключить один файл `autoload.php`, который находится в папке `vendor`:

```
<?php
require "vendor/autoload.php";
// ...
?>
```

В результате вы подключите все доступные библиотеки которые есть в данной папке. Файл `autoload.php` отвечает за подключение всех библиотек, которые находятся в папке `vendor`. Хорошо, мы научились подключать библиотеки, а также устанавливать их с помощью пакетного менеджера. Если нам надо будет найти определенный пакет то можно воспользоваться веб-сайтом <https://packagist.org>, на котором можно найти много пакетов имеющих разный функционал.



Packagist The PHP Package Repository

Search packages...

Packagist is the main [Composer](#) repository. It aggregates public PHP packages installable with [Composer](#).

Getting Started

Define Your Dependencies

Put a file named `composer.json` at the root of your project, containing your project dependencies:

```
{
  "require": {
    "vendor/package": "1.3.2",
    "vendor/package2": "1.*",
    "vendor/package3": "~2.0.3"
  }
}
```

For more information about packages versions usage, see the [composer documentation](#).

Install Composer In Your Project

Run this in your command line:

```
curl -s https://getcomposer.org/installer | php
```

Or download `composer.phar` into your project root.

See the [Composer documentation](#) for complete installation instructions on various platforms.

Install Dependencies

Execute this in your project root.

```
php composer.phar install
```

Publishing Packages

Define Your Package

Put a file named `composer.json` at the root of your package's repository, containing this information:

```
{
  "name": "your-vendor-name/package-name",
  "description": "A short description of what your package does",
  "require": {
    "php": "^5.3.3 || ^7.0",
    "another-vendor/package": "1.*"
  }
}
```

This is the strictly minimal information you have to give.

For more details about package naming and the fields you can use to document your package better, see the [about](#) page.

Commit The File

Add the `composer.json` to your git or other VCS repository and commit it.

Publish It

[Login](#) or [register](#) on this site, then hit the [submit](#) button in the menu.

Once you entered your public repository URL in there, your package will be automatically crawled periodically. You just have to make sure you keep the `composer.json` file up to date.

Sharing Private Code

Use [Private Packagist](#) if you want to share private code as a [Composer](#) package with

- *Рис. 40 - Главная страница сайта пакетов для PHP - Packagist*

Кроме того что вы можете найти нужные для вас пакеты, вы можете также создать свой пакет и опубликовать его на данном сайте.

Конечно же для того чтобы создать свой пакет вам придется иметь очень много знаний в веб-разработке, но также вы должны иметь чётко определенную проблему которую будет решать данный пакет.

Вот представим себе, что мы нашли пакет и нам надо его установить:

phpseclib/phpseclib	PHP	↓ 29 344 912
PHP Secure Communications Library - Pure-PHP implementations of RSA, AES, SSH2, SFTP, X.509 etc.		★ 2 338
defuse/php-encryption	PHP	↓ 3 581 680
Secure PHP Encryption Library		★ 1 941

- *Рис. 41 - Пример вывода результатов поиска пакетов на сайте packagist.*

Давайте просто установим пакет под именем `php-encryption` для того чтобы повторить материал. Для этого давайте выполним команду:

```
composer require defuse/php-encryption
```

Что ж, на этом я хочу завершить данный подраздел. В следующем разделе мы будем говорить об том как можно посылать электронные письма с помощью PHP, а также используя библиотеку `swiftmailer`.

Заключение

Использование пакетного менеджера в несколько раз облегчает разработку приложения. Сейчас нельзя себе представить разработку приложений без использования каких-то расширений и библиотек. Библиотеки и расширения позволяют сократить время на разработку, а также предоставляют полноценный каркас для создания своего приложения. Поэтому используйте библиотеки, поскольку это облегчит вам разработку.

Я завершаю этот раздел и мы переходим к следующему. Но перед этим я попрошу читателя ответить на вопросы которые ниже, а также выполнить упражнения которые также находятся ниже.

Вопросы и упражнения для самоконтроля

1. В чем смысл использования библиотек?
2. Как можно установить пакеты?
3. Как подключить все библиотеки к одному PHP файлу?
4. Какие есть поля в файле `composer.json`?
5. Где можно найти нужные пакеты?

17 Отправка электронных писем

PHP можно использовать не только для того чтобы создавать какие-то хорошие динамические приложения которые могли делать все что мы захотим. Также можно реализовать в своем приложении отправку электронных писем. Например это можно использовать для того чтобы сделать автоматическое отправление писем для рассылки своим клиентам. В общем можно много чего реализовать. Конечно же, мы не будем использовать PHP средства, которые предназначены для этого.

Мы будем использовать библиотеку Swiftmailer которую мы устанавливали в предыдущем разделе. Если вы пропустили предыдущий раздел, то я очень и очень рекомендую читателю вернуться к этому разделу. Или если вы просто не установили данную библиотеку, то просто введите эту команду:

```
composer require swiftmailer/swiftmailer
```

Установите в папке отдельного проекта и переходите к следующему подразделу.

17.1 Основы

Для начала нам надо будет подключить библиотеку. Но перед этим я немного расскажу вам об структуре папки. Во-первых, я создал отдельный проект, точнее папку для хампр как отдельный проект и внутри установил как раз swiftmailer. Дальше мы создадим файл `index.php`:

```
<?php  
require "vendor/autoload.php";  
?>
```


А теперь мы должны подключиться к нашему e-mail серверу. Но перед этим немного расскажу на базовом уровне об том какие есть протоколы в сети. Когда мы хотим мы с вами например, загрузить на сервер файл то мы используем протокол FTP. К примеру, когда мы загружаем в своё приложение новый скрипт. Когда мы с вами загружаем какой-то файл через браузер, то здесь уже работает протокол HTTP. А если мы хотим послать какое-то сообщение, то уже здесь мы используем протокол SMTP.

SMTP - как можно понять используется для того чтобы посылать электронные письма на сервер электронной почты. Поскольку мы с вами используем локальный сервер, то надо будет настроить PHP под локальный SMTP сервер. Но мы можем этого не делать! А все потому что с помощью swiftmailer можно с легкостью подключаться к любым другим сервисам. У нас уже есть все нужные средства. Мы будем с вами подключаться к e-mail сервису от компании Google, т.е. к Gmail.

Поэтому давайте с вами сделаем подключение, сразу говорю что вам понадобится ваша учётная запись Google а также почтовый ящик для теста. В примере я буду использовать аккаунт, которого не существует, вот код с примером использования Swiftmailer:

```
<?php
require "vendor/autoload.php";

$transporter = (new
Swift_SmtpTransport("smtp.gmail.com", 465, "ssl"))
    ->setUsername("example@gmail.com")
    ->setPassword("my_password");

?>
```

Смотрите, мы создаём объект SMTP транспортера для того чтобы подключиться к сервису Gmail. Когда мы с вами создали новый объект, то мы указали три параметра. Первый параметр - это сервер, к которому мы хотим подключиться, второй это порт и третий это протокол который мы используем для подключения. Заметьте что мы потом указали логин и

пароль пользователя учетной записи сервиса gmail. Туда вы можете ввести данные своей учетной записи. Далее мы можем с вами взять и попробовать отослать тестовое сообщение:

```
<?php
require "vendor/autoload.php";

$transporter = (new
Swift_SmtpTransport("smtp.gmail.com", 465, "ssl"))
    ->setUsername("example@gmail.com")
    ->setPassword("my_password");

$mailer = new Swift_Mailer($transporter); //
Создаем наш майлер для того чтобы отсылать сообщения

$message = (new Swift_Message("Мое первое
сообщение"))
    ->setFrom(["example@gmail.com" => "Mike
Jordan"])
    ->setTo(["receiver@gmail.com" => "Name of
receiver"])
    ->setBody("Here is text of message");

$result = $mailer->send($message);
?>
```

В этом коде мы уже создаем сообщение которое мы потом отсылаем с помощью нашего майлера. То что вы сейчас видите, это самый простой способ использования Swiftmailer. Переходим к следующему подразделу.

17.2 Загрузка файлов

Хорошо, мы можем отсылать простые сообщения с текстом. Ну а как насчёт того чтобы сделать так чтобы можно было отсылать файл? Для этого нам понадобится создать объект класса, который отвечает за файлы в Swiftmailer:

```
<?php
require "vendor/autoload.php";

// ...
// Здесь мы как бы создаем наш транспортер
// в переменной $transporter
// ...

$mailer = new Swift_Mailer($transporter);

$message = (new Swift_Message("Мое первое
сообщение"))
    ->setFrom(["example@gmail.com" => "Mike
Jordan"])
    ->setTo(["receiver@gmail.com" => "Name of
receiver"])
    ->setBody("Here is text of message");
    $attachment =
Swift_Attachment::fromPath("path/to/my/file.pdf", "application/pdf"); // Указываем какой файл
мы хотим прикрепить к нашему письму

$message->attach($attachment); // Прикрепляем
сам файл к нашему письму

$result = $mailer->send($message);
```

?>

Заметьте, что для того чтобы прикрепить к письму файл мы воспользовались функцией `formPath`, с помощью которой мы указали путь к файлу, а также его формат с помощью второго параметра указали формат для нашего файла. А дальше уже с помощью функции `attach` мы прикрепили сам файл к письму. Но если у вас случилась такая ситуация что вам надо поменять имя файла который вы хотите вложить, то просто воспользуйтесь функцией `setFilename()` :

```
<?php
//...
$attachment =
Swift_Attachment::formPath("path/to/my/file.pdf",
"application/pdf")
    ->setFilename("super-file.pdf"); //
```

Изменяем имя файла

```
//...

?>
```

Конечно, есть и много других способов как можно прикрепить файлы к письму. Также хочу подметить, что в письме можно использовать какой-то HTML код:

```
<?php
//...
$message->setBody("
    <div class='container'>
        <h3>My beauty title</h3>
        <p>
            And here some text.
        </p>
    </div>
");
//...

?>
```

Конечно же, файлы можно прикреплять не по указанной отдельной переменной, но также сразу же создав объект прямо в функции, или по url адресу:

```
<?php
//...
$message->attach(
    Swift_Attachment::fromPath("path_to/image.
jpg", "image/jpg");
); // Прикрепляем изображение прямо в функции

$message->attach(
    Swift_Attachment::fromPath("https://exampl
e.com/image.jpg");
); // Прикрепляем изображение по URL-адресу
//...
?>
```

Ну что же ещё можно сказать об контенте добавляемым пользователем? Мы знаем, как можно прикреплять файлы, добавлять HTML-код в сообщение. Что же можно ещё сделать? Надо немного поговорить об безопасности и об том как создавать рассылку данных.

17.3 Создание рассылки

А что будем делать если нам надо будет разослать какие-то письма с информацией нескольким людям а не только одному человеку? Для этого мы должны будем передать массив из адресами почтовых ящиков в функцию для тех кто получает сообщение:

```
<?php
// ...
$message->sendTo([
    "email1@example.com" => "Super Recipient",
    "address@example.com",
```

```
        "email2@example.com" => "Nice Recipient"
    ]);
    // ...
    ?>
```

Заметьте что второй элемент массива который мы передаем - не имеет значения, то есть имени которое мы можем дать. В такой способ можно разработать систему рассылки в своем приложении. То есть просто брать из базы данных целый массив с адресами электронных адресов и передавать их на управление функции. Такое же самое можно сделать и с электронными адресами для отправителя.

То есть электронные письма можно посылать прямо из нескольких электронных адресов. Но а что же можно сказать об безопасности в Swiftmailer? Есть ли средства которые смогут защитить посылаемые сообщения? Конечно же, что есть! Например, можно с легкостью создавать сертификаты прямо к сообщениям:

```
<?php
$message = new Swift_Message();

$message->setFrom('example@example.com', 'Example Mailer');
$message->addRecipient('path/to/recipient@example.com', 'Example Recipient');

$message->attachSigner($smimeSigner);

$smimeSigner = new Swift_Signers_SMimeSigner();
// Создаем объект генератора сертификатов
$smimeSigner->setSignCertificate
("path/to/certificate.pem", "path/to/private-
key.pem"); // Указываем путь к сертификату и
приватному ключу

$message->attachSigner($smimeSigner); //
Прикрепляем сертификат
?>
```

Как вы видите то здесь надо обязательно указать сертификат и приватный ключ если мы хотим прикрепить наш сертификат/электронную подпись. Если вы хотите рассмотреть больше

функционала Swiftmailer то можете заглянуть в официальную документацию по адресу:

<https://swiftmailer.symfony.com/docs/index.html>

А сейчас же мы поговорим об том как можно реализовать отправку электронных писем с помощью PHP средств. На этом мы долго застревать не будем.

17.4 PHP Средства

Для того чтобы мы могли использовать PHP средства для отправки электронных писем то для начала нам надо будет настроить файл конфигурации PHP - `php.ini`. Нам надо будет добавить несколько строчек кода конфигурации. Чтобы мы могли настроить наш PHP под SMTP нам надо найти такие строчки кода в файле конфигурации:

```
[mail function]
; For Win32 only.
; http://php.net/smtp
SMTP=localhost
; http://php.net/smtp-port
smtp_port=25

; For Win32 only.
; http://php.net/sendmail-from
;sendmail_from = me@example.com

; For Unix only. You may supply arguments as
well (default: "sendmail -t -i").
; http://php.net/sendmail-path
;sendmail_path =
```

В моем файле конфигурации PHP эти строчки кода находятся на таких адресах 1056-1069. вы видите здесь переменную SMTP которая имеет значение localhost.

В этой переменной как вы смогли понять то оно хранит название SMTP сервера, который отвечает за функции электронной почты. Дальше идет переменная `smtp_port` которая хранит в себе имя порта сервера. Здесь в этих переменных мы можем указать также параметры для сервиса Gmail. Дальше вы можете заметить переменную `sendmail_from` которая хранит в себе значение имя почтового ящика, с которого будут отправляться электронные письма.

Хорошо, после того как мы настроили наш файл конфигурации то мы можем переходить к тому чтобы начинать писать первый небольшой скрипт для того чтобы можно отправлять электронные письма с помощью PHP средств. То есть будет форма, через которую мы будем вводить данные и отсылать их потом. Параметры эти настроены, если мы будем использовать PHP без настраивания сервера через PHP. Конечно же, подключиться к SMTP серверу можно и через PHP скрипт. Подключение к SMTP серверу через PHP-скрипт будет вам немного напоминать скрипт подключения к БД.

Вот сам скрипт:

```
<?php
$send_to = "user12345678@example.com";
$subject = "Здесь тема сообщения";
$message = "Привет! Как дела?";
$headers = array(
    "From" => "supermike@example.com",
    "Reply-to" => "supermike@example.com",
    "X-Mailer" => «яPHP/" . phpversion()
);

mail($send_to, $subject, $message, $headers);
?>
```

Для того чтобы отправить сообщение мы воспользовались функцией `mail()`. Но кроме этого мы задали несколько параметров, первый это кому мы хотим отослать сообщение, второй параметр это тема

сообщения, третий параметр само сообщение и четвертый это массив внутри которого содержатся заголовки с параметрами для запроса. Это довольно таки простой способ использования PHP средств, для того чтобы отсылать сообщения. Функция `mail()` использовала те данные для отсылки сообщения, которые были указаны в файле конфигурации PHP.

Думаю некоторые вспомнят что я говорил об том, что можно подключаться к SMTP серверу на чистом PHP, и при этом не редактируя файла конфигурации PHP. Скажу правду, в предыдущем разделе я вам не показывал то, как можно подключаться к SMTP серверу через чистый PHP, а все потому что написание скрипта который подключался к SMTP серверу - это очень сложно.

Поскольку вообще для того чтобы подключиться к SMTP серверу надо указать **все** параметры заголовков, а также уйму других параметров и для этого надо много времени. А также новичку будет довольно таки сложно понять код для подключения. Но если кому-то интересно, то может найти информацию об этом.

Но все-таки, если вы хотите использовать PHP средства для того чтобы отсылать электронные письма - то все таки лучше настраивать файл конфигурации чтобы было легче работать. На этом я завершу этот раздел, и дальше мы перейдем к более серьезной теме. А пока что давайте сделаем заключение об то что мы изучили в данном разделе.

Заключение

Использование PHP для того чтобы отправлять электронные письма, позволяет сделать автоматизацию рассылки каких-то сообщений, например рассылку пользователям приложения. Если использовать Swiftmailer то работа сократится в несколько раз. Кроме этой библиотеки вы можете найти кучу других библиотек, которые вы можете использовать для облегчения отправки электронных писем с помощью использования PHP.

Конечно же существует большая куча веб-приложений которые можно использовать с легкостью для того чтобы создавать рассылку, даже не умея программировать. Но возможно все равно кому-то понадобится сделать в своем приложении функцию, повязанную с отправкой электронных писем. На этом я завершаю данный подраздел, и как всегда я попрошу читателя ответить на вопросы и выполнить упражнения которые находятся ниже.

Вопросы и упражнения для самоконтроля

1. Как установить библиотеку Swiftmailer?
2. Какие есть параметры для того чтобы подключиться к SMTP серверу?
3. Наведите пример сервисов к которым можно подключаться через SMTP?
4. Напишите программу с использованием Swiftmailer, которая разослала бы нескольким людям письмо с темой “Hello world!”, и с таким же содержанием.
5. Какие есть функции для того чтобы присвоить определенны данные для сообщения? Например, заголовок или содержание письма.
6. Как можно прикреплять файлы к сообщению с помощью Swiftmailer?
7. Как можно сделать отправку электронных писем на чистом PHP?
8. Какие есть переменные в файле конфигурации PHP для того чтобы настроить SMTP для PHP?

18 PHP и тестирование

Качество кода это важно. От этого может зависеть очень много факторов. К примеру, скорость работы скрипта. В этом разделе мы поговорим об том как можно тестировать свое приложение написанное на php. Есть множество библиотек, которые также позволяют делать тесты для своего кода.

Для реализации тестов мы будем использовать фреймворк PHPUnit. PHPUnit - это один из самых популярных PHP фреймворков для того чтобы тестировать свое приложение. Если кто то забыл что такое фреймворк, или просто не знает, то это набор библиотек и инструментов для того чтобы разрабатывать своё приложение. Фреймворк это как бы сказать готовый каркас на котором разрабатывается приложение.

18.1 Установка

Для начала давайте установим PHPUnit. Вы можете перейти на официальный сайт по ссылке: <https://phpunit.de/>. Думаю, вы смогли догадаться что устанавливать данный фреймворк мы будем с помощью пакетного менеджера composer.

Ссылка на страницу с инструкцией установки:
<https://phpunit.de/getting-started/phpunit-7.html>.

Давайте установим этот фреймворк, воспользовавшись данной командой в пакетном менеджере:

```
composer require --dev phpunit/phpunit ^7
```

Хочу заметить, что в конце данной команды мы указали то что мы хотим использовать версию которая выше 7. После установки мы можем начинать его использовать, главное только не забыть подключить фреймворк к приложению.

18.2 Первый тест

У нас будет класс Book, который мы будем тестировать. Вот содержание кода класса, он имеет только одну функцию, которую мы протестируем с помощью PHPUnit:

```
<?php
require "vendor/autoload.php";

class Book {
    public function sumData($a, $b)
    {
        return $a + $b;
    }
}
?>
```

Класс вышел очень примитивным, но суть заключается в том чтобы просто его протестировать и понять как работают тесты в PHPUnit. Как вы видите то здесь мы создали функцию, которая выполняет операцию сложения двух чисел a и b. Вот как создается тест для данного класса:

```
<?php
require "vendor/autoload.php";
require "book.php"; // Обязательно подключаем
то что хотим протестировать
```

```
class TestBook extends
PHPUnit_Framework_TestCase {
    private $book;

    protected function setUp()
    {
        $this->book = new Book();
    }

    protected function tearDown()
    {
        $this->book = NULL;
    }

    public function testData() // Функция
которая будет тестировать функцию в нашем классе
    {
        $test = $this->book->sumData(5, 4);
        $this->assertEquals(9, $test);
        // Проверяем правильность вывода
    }
}

?>
```

Теперь смотрите, здесь есть 2 функции которые отвечают за включение/выключение теста. Первой функцией является функция

`setUp()` которая создает новый объект класса, который мы хотим протестировать.

Дальше идет функция `tearDown()`, которая уже выключает тест, при этом давая значение `NULL` локальной переменной `$book`, которая содержит в себе объект тестируемого класса. Последняя функция в нашем тесте, это функция, которая как раз и тестирует функцию в нашем классе. Вы можете заметить что здесь есть переменная `$test` которая как раз и тестирует функцию `sumData()`. Мы передаем определенные параметры нашей функции для того чтобы её протестировать, после этого мы обращаемся к классу `PHPUnit_Framework_TestCase()`, используя при этом функцию `assertEquals()`.

Функция `assertEquals()`, проверяет, вернул ли правильное значение наша функция `sumData()` в переменной `$test`. В нашем случае вы можете заметить, что мы проверяем, правильно ли значение которое должно быть равно числу 9. А теперь что бы запустить тест нам надо ввести данную команду:

```
phpunit BookTest.php
```

Здесь мы указываем имя файла, который является тестом, написанным на PHPUnit. После этого вы сможете увидеть результат теста, который выведется в консоль. Лог который выведет консоль будет содержать информацию об скорости выполнения скрипта. Кроме того что мы получаем информацию об скорости выполнения скрипта, мы также получаем информацию об том сколько было использовано оперативной памяти PHP для выполнения данного скрипта.

Вот какой у меня был вывод результат:

```
PHPUnit 7.3 by Sebastian Bergmann and contributors.
```

```
.
```

```
Time: 114 ms, Memory: 12.0Mb
```

```
OK (1 test, 1 assertion)
```

Думаю, вы заметили, сообщение которое в конце результата которое сообщает об том, что был проведен 1 тест и 1 проверка данных, а также что все прошло успешно. А давайте рассмотрим с вами тест, который не был пройден успешно? Т.е. посмотрим, как выглядит ошибка при не успешном тесте. Для этого просто достаточно добавить неправильное значение в новой функции при проверке данных:

```
<?php
...
class BookTest extends
PHPUnit_Framework_TestCase {
    ...
    public function badTest() {
        $result = $book->sumData(5, 5);
        $this->assertEquals(9, $result);
    }
}

?>
```

После того как мы попытаемся выполнить тест то получим в результате сообщение что в тесте было неправильное значение и что тест был проведен не успешно:

```
PHPUnit 7.3 by Sebastian Bergmann and
contributors.
```

```
.F
```

```
Time: 130 ms, Memory: 12Mb
```

```
There was 1 failure:
```

```
1) BookTest::badTest
Failed asserting that 9 matches expected 10.
```

```
BookTest.php: 30
```

```
FAILURES!
```

```
Tests: 2, Assertions: 2, Failures: 1.
```

Можете заметить, что в конце сообщения показано что произошёл один неудачный тест. Вы можете прямо сейчас попробовать написать собственную функцию, которая будет выполнять определенную операцию, и при этом, когда вы будете писать собственный тест, то вы можете добавить в него собственную проверку работы теста.

18.3 Data Provider

Представьте, что нам надо будет проверять много значений и делать много тестов. Чтобы не создавать много функций для тестов надо воспользоваться data provider'ом, например, вот как выглядит типичный код, в котором много тестов с проверкой значений:

```
<?php
...
class BookTest extends
PHPUnit_Framework_TestCase {
    ...
    public function test1()
    {
        $result = $this->book->sumData(2,
4);
        $this->assertEquals(6, $result);
    }
    public function test2()
    {
        $result = $this->book->sumData(3,
6);
```



```

        $this->assertEquals(9, $result);
    }
    public function test3()
    {
        $result = $this->book->sumData(4,
10);
        $this->assertEquals(14, $result);
    }
}
?>

```

Вы видите, что здесь данные функции занимают много места, а также они могут использовать больше оперативной памяти. Поэтому для того чтобы не “изобретать велосипед”, давайте создадим Data Provider - т.е. провайдер данных, который как раз и предоставит все эти значения одному тесту чтобы не приходилось использовать много места. И при этом будут сделаны тесты с проверкой значений, которые были указаны в провайдере данных.

Вот код класса уже с нашим провайдером данных:

```

<?php
...
class BookTest extends
PHPUnit_Framework_TestCase {
    ...
    public function addProvider()
    {
        return array(
            [3, 4, 7],
            [4, 15, 19],
            [6, 10, 16],
        );
    }
}

```

```
/**
 * @dataProvider addProvider
 */
// Заметьте то в какой способ мы добавляем
// провайдер данных к тестовой функции,
public function testData($a, $b,
$expected)
{
    $result = $this->book->sumData($a,
$b);
    $this->assertEquals($expected,
$result);
}
?>
```

Заметьте что в функции, которая создаёт тест, имеет 3 локальные переменные. Две первые переменные это переменные, которые имеет функция, которую мы хотим с вами протестировать. Последней локальной переменной, является переменная, которая проверяет значение в функции `assertEquals()`.

Как вы уже смогли понять, то просто передаем данной функции массив из данными которые содержатся в нашем провайдере данных. После этого с помощью локальных переменных, которые используются в функции для создания теста, мы выполняем работу функции которую хотим протестировать, после чего проверяем на правильность значения.

Как вы видите то использовать провайдеры данных хорошо, если мы хотим сделать большое количество тестов, а также не нагружать работу скрипта.

18.4 Больше методов для написания тестов

Кроме того что мы можем проверять правильность значения в тестах, можно также и проверять наличие значения с помощью функции `assertNotEmpty()`:

```
<?php
...
class MyTest extends PHPUnit_Framework_TestCase
{
    ...
    public function newTest()
    {
        $this->assertEquals("TRUE", $value);
        $this->assertNotEmpty($value);
    }
}

?>
```

Можно также проверить проверку значения на то равно ли оно истине или нет с помощью функции `assertTrue()`:

```
<?php
...
class MyTest extends PHPUnit_Framework_TestCase
{
    public function test()
    {
        $this->assertTrue(false);
        // Если мы попробуем сделать тест то
        получим
        // конечно же ошибку, поскольку мы
        даём
```

```

        // значение которое равно ложь, а не
        истине
    }
}

```

?>

Или вот ещё пример, вот функция для теста, которая проверяет существование определенной папки - `assertDirectoryExists()` :

```

<?php
...
class MyTest extends PHPUnit_Framework_TestCase
{
    ...
    public function mytest()
    {
        $this->assertDirectoryExists
("/path/to/directory");
    }
}
?>

```

Я могу описать ещё здесь просто кучу методов которые вы можете использовать для того чтобы писать собственные тесты, но лучше зайти на официальный сайт PHPUnit с документацией, и самому найти метод который вас интересует.

Ссылка на документацию:

<https://phpunit.de/documentation.html>.

На этом моменте мы завершаем данный раздел, и делаем заключение об том что мы выучили за этот раздел.

Заключение

Тестирование приложения важный процесс, который отвечает за качество вашего продукта. Тестировать можно по многим характеристикам, как например время выполнения скрипта или то сколько использует программа оперативной памяти для выполнения определенной операции. В этом разделе вы научились то как можно делать простые тесты, если вы хотите больше узнать об том как можно тестировать приложения то можете заглянуть в официальную документацию или даже посмотреть книгу, которая есть на сайте разработчиков.

Там вы сможете найти больше информации об том как писать тесты. Вообще созданием тестов и тестированием программы занимается человек, которого называют тестировщиком или QA. Это также ещё одна отрасль в IT сфере, которая может заинтересовать читателя. Моим заданием было показать вам основы написания тестов, а не нагружать вас большим объемом информации об том, как можно создать тест для какого-нибудь большого класса внутри которого около 20 функций, и код данного класса занимает 1000 строчек.

Для вас сейчас главное это получить знания Junior'a, т.е младшего разработчика. А дальше уже вы сможете ещё больше развиваться и создавать более динамичные, и сложные приложения. Самым лучшим способом получить больше знаний программированию это будет работать над каким-то реальным проектом. А также посещать какие-нибудь конференции, которые на тему программирования. Никакая книга в мире не сможет просто взять и сделать из человека супер-разработчика. Есть много вещей, которые можно научиться только с опытом. Чтобы стать хорошим разработчиком надо делать реальные проекты и иметь чёткую цель создания чего-то, а также развития навыка разработчика.

Кроме того что в данном разделе вы научились то как можно писать простые тесты, вы также имели первый опыт с фреймворками. В следующих разделах мы также поговорим об том какие есть PHP

фреймворки для создания приложений с уже готовым набором библиотек и инструментов. Как и в каждом конце раздела, я попрошу читателя ответить на вопросы и выполнить упражнения, которые находятся в конце раздела.

Вопросы и упражнения для самоконтроля

1. PHPUnit фреймворк или библиотека?
2. Что надо для того чтобы написать тест?
3. Какая функция служит для того чтобы проверить правильность значения выполненной другой функцией?
4. Какие данные мы получаем после завершения теста?
5. Что может привести к ошибке в тестировании?
6. Напишите тест который проверял бы файл по таким характеристикам как: читабельность (readable) и возможность редактирования(writeable). Для того чтобы найти нужные вам функции в документации по функциям тестов PHPUnit. Ссылка на документацию:
`http://phpunit.readthedocs.io/en/7.1/assertions.html`
7. Напишите программу, которая бы делала тест на строки. Проверку можете сделать по желаемым качествам, главное чтобы вы воспользовались документацией для того чтобы найти нужные вам функции.

19 Фреймворки

Я уже как то упомянул в этой книге об фреймворках. Если кто не помнить что это такое, то я напомним, фреймворк - это набор библиотек и инструментов, которые позволяют создавать приложение не “изобретая велосипед”, а также во многом облегчают разработку приложения.

В этом разделе мы поговорим с вами об том какие существуют PHP фреймворки, как выбрать себе подходящий, а также поработает с одним из фреймворков - Symfony. Symfony - это простой фреймворк, который годиться для того чтобы изучать его как первый фреймворк.

19.1 Какие существуют PHP фреймворки?

Существует много PHP фреймворков которые можно использовать для разработки собственного приложения. На данный момент самыми популярными фреймворками являются: Laravel, Phalcon, Symfony, CakePHP, CodeIgniter и Yii2. Это целых 6 популярных фреймворков, это круто, но это не значит что вам придется учить каждый. Хотя это уже зависит от вашего желания. Вы можете учить столько фреймворков сколько вы хотите. Но какой же стоит выбрать? Самым популярным фреймворком считается Laravel, на рис. 42 вы можете увидеть логотип данного фреймворка.



- Рис. 42 - Логотип фреймворка Laravel

Вообще выбор фреймворка зависит от того какую задачу нам надо решить. Так же само и с библиотеками. Но в общем можно использовать любой фреймворк для создания приложения, главное чтобы данный фреймворк имел все те нужные инструменты для создания приложения.

Для новичка вообще надо для начала понять, как вообще выглядит создание приложения с помощью фреймворка. Первое что надо будет вообще понять любому новичку который изучает свой первый фреймворк - это понять что такое MVC паттерн, и как он работает в фреймворке. А также хочу добавить, что кроме понимания данного паттерна разработчику надо также будет знать об разных шаблонах проектирования.

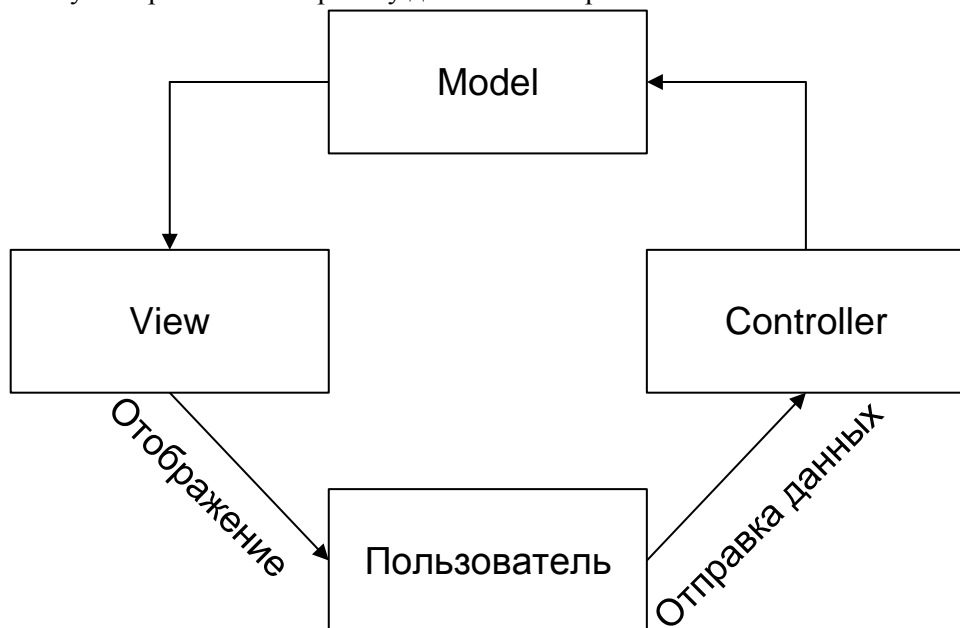
Но сейчас мы говорим об фреймворках в общем. Т.е. новичку надо понять общую картину того как создаются приложения на фреймворках. Поэтому в следующем подразделе мы обговорим общее понятия об фреймворках, а также установим фреймворк Symfony.

19.2 Общее понятия

Многие фреймворки используют паттерн об котором я говорил выше. То есть MVC паттерн, аббревиатура расшифровывается как Model-View-Controller(Модель Вид Контроллер). Объясню, как эта работает система. У нас есть 3 компонента, которых я назвал выше. Каждый из них отвечает за определенную часть в приложении. Внизу показан список, в котором указано что и за что отвечает каждый компонент паттерна:

- Model - этот компонент который отвечает за запросы к базе данных, а также за таблицы. То есть отвечает за структуры таблиц.
- View - это компонент который отвечает за внешний вид приложения. То есть это файлы внешнего вида.
- Controller - этот компонент который уже отвечает за работу страниц. Например контроллер может маршрутизацию страниц и за то что будет происходит на странице.

Все эти элементы работают между собой, и внизу я показал вам схему которая объясняет работу данного паттерна:



- Рис. 43 - Схема поясняющая работу MVC паттерна

По схеме вы видите что пользователь посылает данные контроллеру, который потом передает указания модели, после чего модель передает указания компоненту отвечающему за отображение внешнего вида(интерфейса). Но вообще контроллер дает указания компоненту, отвечающему за отображение интерфейса - о том что именно надо отображать. С этого можно понять, что благодаря тому что пользователь взаимодействует с элементами внешнего вида, он может посылать данные контроллеру который потом обрабатывает эти данные и передает другим компонентам.

Объясню теперь больше о том как это будет выглядеть во время того как мы будем создавать приложение на фреймворке. Мы будем

отдельно создавать контроллер, который будет отвечать за определенный функционал в нашем приложении. Например, пусть будет отвечать за создание постов.

Кроме контроллера, нам надо будет создать модель, которая будет отвечать за посты в базе данных. А дальше уже мы создадим отдельные файлы, которые будут хранить в себе код отображения определенного внешнего интерфейса. Это, например, могут быть веб-страницы. В вашем приложении может быть столько контроллеров, моделей и шаблонов, сколько вам будет нужно и столько, сколько вы захотите. Количество контроллеров, моделей и шаблонов зависит уже от функционала приложения. Как я уже и говорил, то самым лучшим способом показать это будет на примере фреймворка. Поэтому в следующем разделе речь будет о фреймворке Symfony.

19.3 Фреймворк Symfony

На данный момент новейшей версий Symfony является версия 4.1. Именно её мы будем использовать на наших примерах. Фреймворк Symfony поддерживает компания SensioLabs из Франции. А теперь давайте перейдем к установке фреймворка, для этого просто введем команду с помощью пакетного менеджера:

```
composer create-project symfony/website-skeleton my-app
```

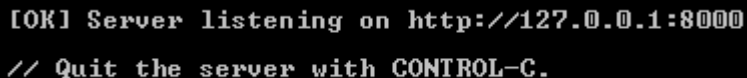
В конце команды вы можете заметить что мы вводим название нашего приложения, в нашем случае это my-app. После того как Symfony установиться вы сможете перейти в папку со своим приложением и начать создавать его на готовой основе. Для примера чтобы читатель мог понять, как работает MVC паттерн на практике, мы будем создавать простой примитивный блог для создания/редактирования/просмотра и удаления постов.

Давайте перейдем в командой строке в папку с нашим приложением, и для того чтобы протестировать работает ли все у нас, давайте запустим локальный сервер Symfony для нашего приложения:

```
php bin/console server:run
```

Немного объясню, как работает данная команда. С помощью PHP мы обращаемся к папке `bin` которая находится в папке нашего приложения, дальше идет обращение к консоли Symfony через файл `console` который как раз и находится в папке `bin`. И только потом после того как мы обратились к консоли мы можем набрать команду которую хотим выполнить. В нашем примере мы запускаем локальный сервер с помощью команды `server:run`. Давайте вернемся к нашей командной строке.

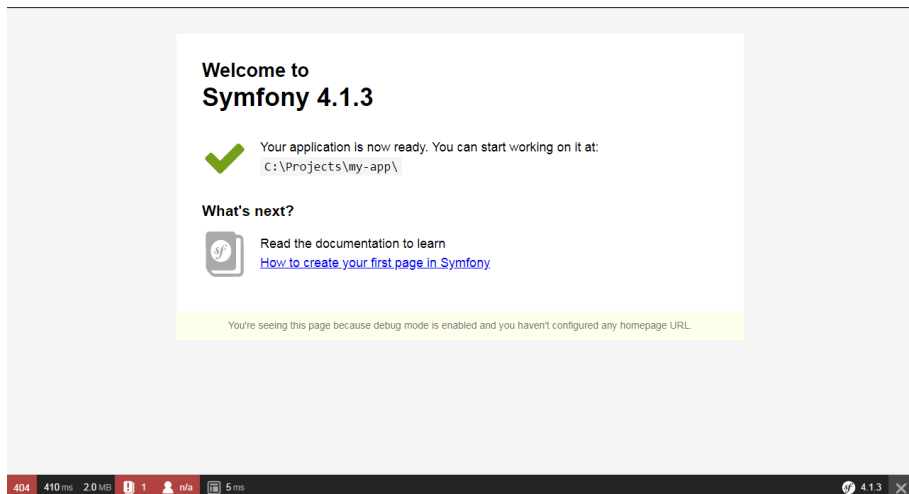
После того как вы запустите локальный сервер, у вас должно выведется сообщение о том на каком хосте был запущен локальный сервер Symfony:



```
[OK] Server listening on http://127.0.0.1:8000
// Quit the server with CONTROL-C.
```

- *Рис. 44 - Сообщение в командной строке о том что локальный сервер Symfony был запущен*

Если у вас появилось данное сообщение, то вы уже можете перейти в браузер и посмотреть на первую страницу, которая установлена по умолчанию. Если же у вас не появилось такого сообщения, подождите, поскольку процесс запуска локального сервера иногда может занять некоторое время. А вот как выглядит первая приветствующая страница в вашем приложении, когда вы не вносили никаких изменений:



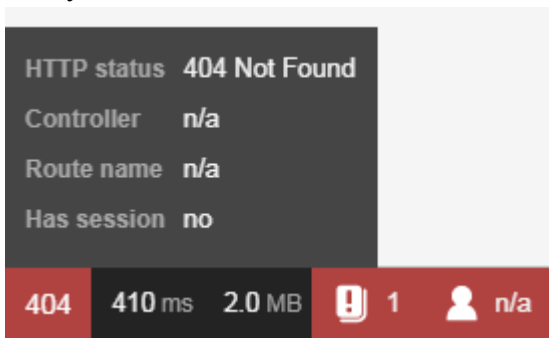
- *Рис. 45 - Страница созданная по умолчанию в приложении после установки Symfony*

На странице вы можете увидеть какая у вас установлена версия Symfony, а также то по какому адресу находится ваше приложение в файловой система вашего компьютера. Кроме этого есть ссылка на документацию Symfony где можно найти инструкции по созданию своего первого приложения. Если вам надо будет остановить локальный сервер Symfony то воспользуйтесь комбинацией клавиш **Ctrl+C**.

Внизу на скриншоте вы можете увидеть внизу большое темное поле. Это profiler данного фреймворка, он позволяет делать отладку кода сразу же. И это уже в несколько раз нам облегчает работу. Поскольку с помощью данного профайлера мы можем узнать очень много информации о том, как работает скрипт. Например, сколько времени идет на генерацию внешнего вида страницы, сколько оперативной памяти использует PHP, сколько времени тратит скрипт на то чтобы сделать определенные запросы к базе данных.

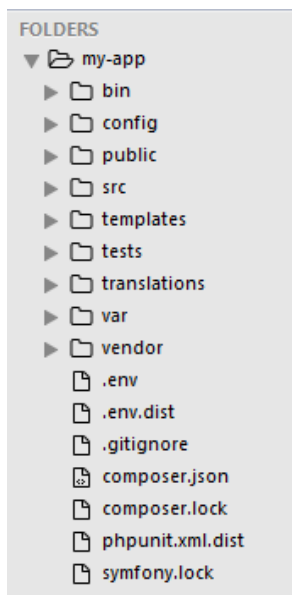
Можно узнать очень много информации. Для этого достаточно перейти на отдельную страницу профайлера, где пишет вся информация

об том, как работает приложение. Чтобы перейти на страницу профайлера кликните слева внизу:



- Рис. 46 - Небольшой кусочек вида профайлера в Symfony.

После этого вы перейдете в профайлер где сможете найти очень много информации об работе HTTP в вашем приложении. Но давайте вернемся к нашему приложению, и поговорим об том, что мы будем делать дальше. Вообще об профайлере я сделаю отдельный подраздел в этом разделе. В первую очередь нам надо будет разобраться с файловой системой приложения, и узнать какая папка за что отвечает. Вот как выглядит файловая система приложения на Symfony(фото слева).



Вы можете здесь увидеть ту самую папку об которой я вам говорил, то есть bin. Мы её никогда не трогаем, это системная папка для того чтобы работать с консолью.

Дальше идет папка config, в которой размещаются файлы конфигурации, об которых мы скоро поговорим. Дальше идет уже папка public ,внутри которой уже хранятся файлы конфигурации, когда уже приложение будет поставлено на полностью рабочий сервер. Следующей папкой является папка src, внутри которой хранятся контроллеры, модели,

структуры таблиц баз данных, миграции баз данных и репозитории.

- *Рис. 47 - Структура папок в приложении созданном на Symfony*

Внутри данной папки есть 4 папки с такими названиями:

- `Controller` - папки с контроллерами.
- `Entity` - модели приложения, а точнее файл с описанием структур таблиц.
- `Migrations` - миграции баз данных.
- `Repository` - репозитории баз данных.

Если вы загляните в данные папки, то там ничего не будет, кроме файлов игнорирования системы контроля версий `Git`. Двигаемся дальше, и следующей папкой у нас является папка `templates`, в которой хранятся шаблоны приложения. В нашем случае там уже есть один шаблон, который был установлен по умолчанию после установки `Symfony`. После этой папки идет папка `tests`, в ней хранятся тесты к приложению. И вот почти последняя папка – `translations`, внутри которой хранятся файлы с переводами языков страниц нашего приложения.

Последней папкой является папка `vendor` которую мы также никогда не трогаем, поскольку в данной папке как вы уже смогли догадаться хранятся все библиотеки и инструменты для разработки приложения. Теперь, когда я объяснил какие папки за что отвечают, переходим к конфигурации базы данных для нашего приложения.

19.4 Создание блога

Для того чтобы настроить наше приложение на базы данных, нам надо открыть файл `env` который находится в корневой папке. После этого в конце файла введите такую строку:

```
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:306/db_name"
```

Теперь объясню что это за параметры. Вы видите что здесь есть переменная `DATABASE_URL` которой мы как раз и присваиваем параметры сервера баз данных. Первым параметром является `db_user` - это логин пользователя MySQL, `db_password` - пароль пользователя MySQL.

Дальше после символа `@` идет адрес и порт сервера баз данных. В нашем случае адресом есть `127.0.0.1` и портом является `3306`. И в конце есть параметр `db_name` с названием базы данных, которую мы хотим использовать в нашем приложении. После того как я вам рассказал за что какие параметры отвечают, возьмите и замените их, на параметры вашего локального сервера баз данных. Также рекомендую вам создать отдельную базу данных для вашего приложения.

Теперь после того как вы настроили приложение на ваш локальный сервер баз данных, надо начинать создавать наш первый контроллер. Наш контроллер будет называться `PostController`, поэтому в папке `Controller` создайте файл `PostController.php`:

```
<?php
namespace App\Controller; // Указываем
пространство имен чтобы Symfony мог обращаться к
классу

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
// Сверху мы подключаем компоненты Symfony
// которые будут использовать в приложении

class PostController {
    /* Ниже делаем маршрутизацию для нашей
первой
    страницы, то есть указываем какой она
будет
```

```

иметь адрес в адресной строке чтобы
перейти
    на неё
*/
* @Route("/", name="list")
*/
public function indexAction()
{
    return new Response("Index page");
    // С помощью создания нового объекта
    класса
    // Response, делаем ответ на
    странице
    // при входе на неё
}
}
?>

```

На первый взгляд может показаться, что мы только что просто создали новый класс. Так оно и есть, но поскольку мы указали пространство имен так, что данный класс находится в папке для контроллеров, а также подключили нужные для нас компоненты - то это уже не класс, а контроллер Symfony.

Чтобы система зарегистрировала данный класс в системе как контроллер, мы должны указать в пространстве имен, что это контроллер. Мы это сделали в начале файла. Дальше после того как мы указали пространство имен, мы подключили все нужные компоненты Symfony, которые будут использоваться в нашем контроллере.

Мы подключили только 2 компонента которые нам понадобятся для того чтобы наш контроллер мог нормально работать для начала. Первым компонентом который мы подключили, является компонент Response который нам понадобился для того чтобы мы могли сделать ответ когда пользователь перейдет на нашу первую страницу.

Следующий компонент который мы подключили, понадобился нам для того чтобы мы могли задать маршрут для нашей первой страницы. Этим компонентом является компонент `Route`. Дальше после подключения компонентов, мы создали сам класс нашего контроллера.

Заметьте что название нашего класса такое же самое как и название файла. Теперь после этого мы создали первую функцию, которая отвечает за главную страницу. С помощью компонента `Route` мы задали маршрут для этой страницы в такой способ: сначала первым аргументом функции `Route()` мы указали адрес страницы, в нашем случае это символ `/`.

Вторым аргументом функции является название данного маршрута, это нужно для того чтобы мы могли потом с легкостью обратиться к нему, если будет такая потребность. Заметьте, что маршрутизацию для нашей первой страницы мы задали с помощью специального типа комментариев.

Ну и также с помощью подключенного компонента `Route`. Задав параметры маршрутизации страницы, мы создали саму функцию для страницы которая называется `indexAction`. Теперь, когда у нас есть маршрутизация для страницы и функция, которая будет отвечать за эту страницу, можно переходить к созданию содержимого страницы.

Сразу хочу сказать, что создавать страницы и вообще писать скрипты как раньше мы не будем. Здесь мы будем использовать инструменты фреймворка чтобы нам было намного легче создать своё первое приложение. Чтобы на странице показался ответ при заходе на нашу первую страницу мы использовали в конце функции аргумент `return`, а после этого мы создали объект класса `Response` для того чтобы сделать ответ в виде текстовых данных, которые были записаны внутри объекта.

А теперь давайте протестируем что мы создали, запустите локальный сервер(если он у вас не запущен), и перейдите по адресу который был записан в функции `Route`. Вы можете указать в параметрах

маршрутизации свой адрес который вы хотите использовать для вашей страницы.

После того как вы зайдете на страницу вы должны увидеть те текстовые данные которые вы указали в новом созданном объекте класса `Response`. Если у вас нету никаких ошибок, а вывелась надпись указанная в объекте - то значит, у вас все работает. Хорошо, а теперь, когда мы умеем работать с контроллером - давайте уже создадим полноценную статистическую страницу для теста. Вы помните что я вам говорил об папке с шаблонами - `templates`. Создайте в этой папке папку - `post`, в которой мы будем хранить шаблоны для нашего контроллера.

Наша первая страница будет просто статистической, и не будет из себя представлять что то особенное. Это будет простая веб-страница, написанная на HTML. Вот код нашей страницы:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <div class="container">
            <h1>My blog</h1>
            <p>
                Here I will write my posts.
            </p>
        </div>
    </body>
</html>
```

Сохраните этот в файл в папке `templates/posts`, назвав его `home.html.twig`. Думаю что, некоторые читатели меня спросят что это за формат такой у файла. Ясно, что это HTML файл, но он использует при этом шаблонизатор Twig, который установлен в Symfony.

Twig позволяет с легкостью выводить какие-то данные, которые мы хотим передать через РНР на страницу. Это как и есть одна из особенностей данного фреймворка. Теперь, когда у нас есть шаблон, который мы хотим использовать для нашей первой страницы, давайте перейдем в файл нашего контроллера и пропишем функцию чтобы при переходе на эту страницу показывался созданный нами шаблон:

```
<?php
...
class Post Controller {
    ...
    public function indexAction()
    {
        return $this->render("posts/home.html.twig");
        /*
        Просто вместо того чтобы создавать
        новый объект, воспользуйтесь
        функцией render, и укажите
        правильный путь
        к папке в которой находится
        шаблон, т.е. в
        папке templates
        */
    }
}
?>
```

Как вы можете увидеть из этого примера, и комментария в данном примере - то мы просто как бы сказать “зарендерели” шаблон для нашей функции страницы. Можете перейти по адресу страницы и проверить работает ли все и отображается ли все то что было запрограммировано в файле.

Хорошо, а теперь давайте разберемся, какие функции будет иметь наш блог. Только четыре функции будут в нашем блоге, это создание/редактирование/удаление/просмотр поста. Поэтому давайте создадим их сначала пустыми в нашем контроллере:

```
<?php
...
use Symfony\Component\HttpFoundation\Request;

class PostController {
    /**
     * @Route("/", name="index")
     */
    public function indexAction()
    {
        return $this->
>render("posts/home.html.twig");
    }

    /**
     * @Route("/create", name="create_post")
     */
    public function createAction()
    {

    }

    /**
     * @Route("/edit/{id}", name="edit_post")
     */
    public function editAction($id)
    {
```

```

    }

    /**
     * @Route("/show/{id}", name="index")
     */
    public function indexAction($id)
    {

    }
}
?>

```

Для того чтобы мы могли делать запросы на получение данных через форму мы подключили компонент Request. Пока что наши функции, которые отвечают за новые страницы - пусты, но вы можете заметить, что в таких функциях как `createAction()` и `editAction()` есть локальная переменная `$id`.

Но также когда мы с вами задавали параметры для маршрутизации данных функций, в конце названия адреса вы могли также заметить локальную переменную, которая объявленная в такой способ `{id}`. Это и есть та самая локальная переменная, которую мы указали в функции. Объясню зачем мы её добавили в адрес маршрута к странице, теперь когда мы будем переходить, например, на страницу для редактирования записи, то нам надо будет просто ввести `id` записи которую мы хотим отредактировать.

Вместо того чтобы брать и указывать локальную переменную с помощью вопросительного знака, как например здесь показано `?id=1`, можно просто после слэша поставить номер `id` поста. И не надо указывать какие-то дополнительные символы и названия переменных. Хорошо, мы смогли определиться, какие функции будет выполнять наш контроллер. А теперь давайте будем наполнять их понемногу. Но перед этим мы должны с вами создать таблицу в базе данных для наших постов.

И делать мы это будем не так как раньше, а с помощью библиотеки Doctrine которая есть в Symfony.

Doctrine - это PHP библиотека, которая легко позволяет управлять базой данных с помощью консоли. Поэтому давайте создадим нашу первую таблицу, для этого давайте введем эту команду:

```
php bin/console make:entity
```

После ввода данной команды у вас запуститься меню для создания нового Entity, т.е. объекта для баз данных. Entity - это php скрипт который хранит в себе все данные об таблице которая есть или будет в базе данных. Сначала вы должны будете указать в консоли имя Entity. Далее после этого вам будет предложено создание полей для таблицы. Для каждого поля надо будет указать такие параметры как имя поля, тип данных поля, уникальные параметры и т.д.

Процесс создания нового поля для нового Entity выглядит вот так:

```
Class name of the entity to create or update:  
> Post
```

```
New property name (press <return> to stop  
adding fields):  
> title
```

```
Field type (enter ? to see all types) [string]:  
> string
```

```
Field length [255]:  
> 60
```

```
Can this field be null in the database  
(nullable) (yes/no) [no]:  
> no
```

И после того как мы указываем все параметры мы можем ещё раз создать новое поле. Важно только указывать все корректно, чтобы не приходилось делать все заново. То, что указано в квадратных скобках - это то что будет указано по умолчанию если программист просто нажмёт enter.

Объясню вам, что значит каждый шаг в создании нового поля.

1 Шаг: Мы указываем название нашего Entity.

2 Шаг: Указываем имя для нового поля.

3 Шаг: Указываем тип данных для этого самого поля.

4 Шаг: Указываем длину для поля. (Длина зависит от того какой тип данных вы укажете)

5 Шаг: Ставим разрешение на то можно ли содержать данному полю в таблице значении NULL. Думаю, вы уже здесь вспомнили об том как это делать в SQL, т.е. с помощью оператора NOT и директивы NULL. По умолчанию значение стоит на no. Когда поле будет создано, вам будет предложено создать ещё одно поле. Сразу хочу сказать, что добавлять поле id не надо, оно уже автоматически добавляется с помощью Doctrine в таблицу.

Если вы хотите завершить процесс создания структуры таблицы, тогда просто нажмите enter, после того как создадите все нужные вам поля. Теперь, когда я вам объяснил, как создаются Entity, создайте Entity для постов, которое будет иметь такие поля как title(заголовок), content(контент), author(автор). Да, и это было ваше первое задание. Теперь, если вы это сделали то нам надо создать миграцию для базы данных, чтобы наша таблица появилась в БД.

Только перед этим убедитесь, что у вас запущен XAMPP, а также правильны ли все параметры в файле env, который находится в корневой папке приложения. Чтобы сделать миграцию для баз данных надо ввести эту команду:

```
php bin/console make:migration
```

И что дальше? После этого в вашей базе данных, которая была указана в конфигурации для базы данных, появится таблица с названием

вашего Entity. В нашем случае, нашим Entity является Post. А теперь, когда у нас есть уже таблица в нашей БД, мы можем начать наполнять наши функции кодом. И первой функцией будет createAction():

```
<?php
...
use App\Entity\Post; // Чтобы мы могли создать
новый объект этого Entity, мы его подключаем
use Symfony\Component\HttpFoundation\Request;
use
Symfony\Component\Form\Extension\Core\Type\TextType;
use
Symfony\Component\Form\Extension\Core\Type\TextareaType;
;
use
Symfony\Component\Form\Extension\Core\Type\SubmitType
;

// Подключаем типы данных для полей формы
...
class PostController {
    ...
    /**
     * ...
     */
    public function createAction(Request
$request)
    {
        $post = new Post();

        $form = $this-
>createFormBuilder($post)
        ->add("title", TextType::class)
        ->add("content", Textarea::class)
```



```

->add("author", TextType::class)
->add("save", SubmitType::class)
->getForm();

$form->handleRequest($request);

if ($form->isSubmitted() && $form-
>isValid()) {
    // Ниже мы присваиваем переменным
значения
    // данных которые были введены
в форму
    $title = $form["title"]-
>getForm();
    $content = $form["content"]-
>getForm();
    $author = $form["author"]-
>getForm();

    // Дальше мы уже присваиваем
значени
    // для нового поста
    $post->setTitle($title);
    $post->setContent($content);
    $post->setAuthor($author);

    $em = $this->getDoctrine()-
>getManager();

    $em->persist($post);
    $em->flush();

```

```

        return $this->
>redirectToRoute("home");
    }

    return $this->
>render("posts/create.html.twig",
        array(
            "form" => $form->createView()
        ));
    }
}

?>

```

Много кода, но перед этим как я начну вам объяснять, что делает написанный нами код, мы должны создать шаблон для нашей функции, назвав его `create.html.wig`:

```

<div class="container">
    {{ form_start(form) }}
    {{ form_widget(form) }}
    {{ form_end(form) }}
</div>

```

Не забудьте сохранить шаблон в папке `templates/post`. (Если что, то все шаблоны контроллера, который отвечает за посты как раз и хранятся в этой папке). Ну а теперь, когда у нас есть все файлы, теперь надо объяснить, как работает код написанный в них. Начнём с нашего контроллера и функции, которая внутри него. Во-первых, я думаю, вы заметили что мы подключаем новые компоненты к нашему контроллеру.

Это типы полей для нашей формы. Например, первый тип который мы подключаем это `TextType`. Мы подключаем три типа, это `TextType`, `TextareaType` и `SubmitType`. Объясню, что значат эти типы полей.

`TextType` - это обычное небольшое текстовое поле.

`Textarea` - это уже полноценное текстовое поле для ввода текста который имеет определенный размер. Например ввод какого-нибудь реферата.

`SubmitType` - это уже кнопка для отправления данных через форму. Кроме того что мы подключили типы полей мы также подключили наш `Entity`, для того чтобы мы могли создавать записи в данной таблице. Заметьте что в функции есть локальная переменная `$request`, которая используется для того чтобы можно было делать запросы. Создание нового поста происходит с помощью переменной `$post` которая является созданным объектом `Entity - Post`. После создания переменной `$post`, мы начинаем создавать форму в переменной `$form`. Для создания формы мы воспользовались функцией `createFormBuilder()`. (То есть вызвали конструктор форм).

Вообще сама функция переводится как “создать конструктор формы”. Сразу хочу добавить, что создать форму с помощью конструктора форм без какого-нибудь `Entity` невозможно. Заметьте, что мы указали в локальную переменную функцию `Entity` которое есть в переменной `$post`. То есть вообще указали наше `Entity` в наш конструктор форм.

После этого, мы начинаем добавлять поля к нашей форме с помощью функции `add()`, думаю вы заметили что мы указываем только 2 параметра. Первый параметр это название поля которое мы хотим добавить, второй параметр это тип поля. Если мы создаём поле с помощью конструктора форм, то мы можем добавлять только те поля - которые есть в `Entity`.

То есть, например, если мы хотим с вами добавить поле `description`, но его нету в самом `Entity`, то мы получим ошибку. А все потому что мы создаём форму на определенный `Entity`. Поэтому мы можем добавлять в форму только те поля, которые есть в нашем `Entity`. После того как мы добавили все поля в нашу форму, мы завершаем создание формы с помощью функции `getForm()`.

Теперь когда в нашей переменной `$form` есть создана форма, мы указываем что с помощью локальной переменной `$request` в функции `handleRequest()` можно делать запросы через форму. После указания этого параметра, мы создаём логическую конструкцию в условии которого есть проверка - была ли отправлена форма и были ли введены данные в все поля.

Эта проверка делается с помощью двух функций `isSubmitted()` и `isValid()`. Первая функция проверяет, была ли отправлена форма, а вторая проверяет были ли заполнены все поля а также валидацию формы. И только внутри данной логической конструкции мы начинаем доставать данные, которые были введены в форму.

Вы можете увидеть 3 переменные, которым мы присваиваем значения из полей формы. Как мы это делаем? Например, у нас есть переменная `$title`, которая получает значение из поля `title`. Делаем мы это обращаясь к нашей форме, обращаясь как к массиву, указывая название нужного поля и используя функцию `getData()`, для того чтобы достать данные из поля.

Когда мы получили значения из полей, мы начали присваивать их объекту нашему `Entity` для того чтобы можно было потом создать новую запись в базе данных с помощью этого объекта. Это мы сделали с помощью функций, которые были автоматически сгенерированы с помощью консоли. Если вы хотите посмотреть на эти функции, то можете зайти в файл нашего `Entity`.

Там вы сможете найти функции, которые позволяют присваивать значения полям объекта, а также доставать данные из этих полей объекта поста. Например, в коде нашей функции для создания нового поста, для того чтобы присвоить значения для поля `title` в нашем объекте мы воспользовались функцией `setTitle()` которая была сгенерирована автоматически в `Entity`.

Такое же самое мы сделали из другими полями, с помощью их функций для присвоения данных. После того как наш объект уже получил значения полей из формы, мы создали переменную `$em` в которой мы вызвали менеджер базы данных, чтобы мы могли добавить новую запись в таблицу. Далее с помощью функции `persist()` мы указываем, что мы хотим добавить в базу данных. А уже после этого с помощью функции `flush()` автоматически создаются запросы к базе данных и данные добавляются. Хочу сразу же заметить, что обращались мы к этим функциям с помощью менеджера баз данных, который мы вызвали в переменной `$em`.

Заканчивается наша логическая конструкция тем, что после создания новой записи в таблице, пользователя перенаправит на домашнюю страницу приложения. Перенаправление сработает только тогда, когда запись будет создана. А вообще, функция возвращает шаблон при переходе на страницу, но также я думаю вы заметили что кроме того что функция возвращает шаблон, она также возвращает форму.

Функция возвращает шаблон и массив, в котором содержатся переменные для шаблонизатора Twig. После того как шаблон получает эти переменные он может к ним обращаться и работать с ними с помощью шаблонизатора Twig. То есть, с помощью Twig переменных вы можете передавать данные из контроллера в шаблон. Например, посмотрите на этот код:

```
<?php
...
$test = "Hello world!";

return $this->render("posts/create.html.twig",
[
    "test" => $test
]);
...
?>
```

После этого в шаблоне я вывожу данную переменную, при этом я обращаюсь к Twig переменной с помощью фигурных скобок:

```
<div class="container">
    {{ test }}
</div>
```

Хочу добавить, что мы можем передавать шаблону столько Twig переменных - сколько нам надо. Хорошо, а теперь вернемся к нашему шаблону с формой. С помощью функций `form_start()`, `form_widget()` и `form_end()` мы создаём форму. Заметьте, что мы указываем Twig переменную с нашей формой как локальную в этих функциях. Все потому что, что для того чтобы создать форму надо переменную содержащую в себе данные об форме. Теперь я кратко объясню, что делает каждая из этих трех функций.

`form_start()` - задает начало формы.

`form_widget()` - показывает содержимое формы, т.е. все поля которые есть в форме.

`form_end()` - задает конец формы.

Кроме того, есть отдельные функции, которые позволяют по отдельности работать с каждым элементом формы. Об этом вы можете более детально почитать в официальной документации разработчиков. В такой способ создаются формы с помощью шаблонизатора Twig. Хорошо, теперь вы понимаете тот код, который мы написали. Переходим дальше. У нас есть функция для того чтобы создавать посты, давайте протестируем её, для того чтобы быть уверенным что все работает.

Просто перейдите на страницу по созданию нового поста, заполните поля и отправьте данные. Если вас поле отправки данных перенаправило на домашнюю страницу, значит, все работает. Хорошо, дальше у нас идет функция, с помощью которой можно редактировать записи. Функция с помощью которой можно будет редактировать посты будет использовать те же самые функции которые использовались для того чтобы создавать пост, только уже немного по-другому.

Просто посмотрите на код данной функции показанный ниже:

```
<?php
...
class PostController {
    ...
    public function editAction($id)
    {
        $post = $this->getDoctrine()
            ->getRepository(Post::class)
            ->find($id);

        $form = $this-
>createFormBuilder($post)
            ->add("title", TextType::class)
            ->add("content",
Textarea::class)
            ->add("author",
TextType::class)
            ->add("save",
SubmitType::class)
            ->getForm();

        $post->setTitle($post->getTitle());
        $post->setContent($post-
>getContent());
        $post->setAuthor($post-
>getAuthor());

        if ($form->isSubmitted() && $form-
>isValid()) {
            $title = $form["title"]-
>getData();
```

```

        $content = $form["content"]-
>getData();
        $author = $form["author"]-
>getData();

        $post->setTitle($title);
        $post->content($content);
        $post->author($author);

        $em = $this->getDoctrine()-
>getManager();
        $em->flush();

        return $this-
>redirectToRoute("home");
    }

    return $this-
>render("posts/edit.html.twig", [
        "form" => $form->createView()
    ]);
}
}

```

?>

Здесь уже нам не приходится создавать новый объект Post с “чистого листа”, мы берем его уже с базы данных. Это делается, вызывая Doctrine и с помощью функции `getRepository()` мы вызываем наш Entity. А дальше с помощью функции `find()` мы находим наш пост по `id` который был указан в адресной строке.

Чтобы в нашей форме отображались те данные, которые есть в таблице базы данных, мы с помощью функций для присвоения и

доставания значений задали им значения. Дальше мы создали логическую конструкцию суть которой в том что она просто задала новые значения из формы, а потом сделала запрос к базе данных с помощью менеджера БД.

Заметьте, что здесь мы не использовали функцию `persist()`, поскольку мы не собираемся добавлять новый объект в базу данных, а только хотим обновить существующий. А дальше уже все как в функции с созданием поста, т.е. перенаправление на домашнюю страницу или отображение заданного шаблона при переходе на страницу.

Сразу хочу сказать, что код шаблона для данной функции будет таким же самым, как и для функции создания постов. Мы создали 2 функции, осталось теперь осталось ещё 2. Дальше будет намного легче создавать функции. Сейчас мы будем создавать функцию для того чтобы просматривать определенную запись, вот код:

```
<?php
...
class PostController {
    ...
    public function viewAction($id) {
        $post = $this->getDoctrine()
            ->getRepository(Post::class)
            ->find($id);

        return $this-
>render("posts/view.html.twig", [
            "post" => $post
        ]);
    }
    ...
}
...
?>
```

И код шаблона:

```
<div class="container">
    {% for post_data in post %}
        <b>{{ post_data.title }}</b>
        <p>{{ post_data.content }}</p>
        <p>{{ post_data.author }}</p>
    {% endfor %}
</div>
```

Код функции вы уже сами можете понять. В коде шаблона мы воспользовались циклом `for` для того чтобы вывести данные о посте. Заметьте то, в какой способ мы создали цикл. Мы создаём переменную `post_data` которая имеет данные массива `post`. Мы создали цикл для того чтобы перебрать элементы массива, потому если бы мы с вами попробовали бы без цикла обратиться к элементам массива мы получили бы сразу ошибку, о том что невозможно конвертировать массив в строку. И именно для того чтобы обратиться к элементам массива мы с помощью цикла конвертировали каждый элемент в строку.

А дальше уже мы просто вывели их на страницу, обращаясь к их названиям, которые есть в таблице базы данных. То есть, какое название поля - такое будет названия элемента в массиве. Нам осталось создать последнюю функцию, функция удаления записи из базы данных. Все будет работать по тому же алгоритму, т.е. поиск записи по `id` в базе данных, получение данных и удаление данных. Посмотрите на этот код:

```
<?php
...
class PostController {
    ...
    /**
     *      @Route("/delete/{id}")
     */
    public function deleteAction($id)
    {
        $post = $this->getDoctrine()
```

```
->getRepository(Post::class)
->find($id);

$em = $this->getDoctrine->getManager();
$em->remove($post); // Удаляем пост
$em->flush();

return $this->redirectToRoute("home");
}
}
?>
```

Функция выглядит намного проще, чем предыдущие. Это ясно, поскольку здесь выполняется только 2 действия. Найти пост и удалить его. Здесь мы достали объект, а потом вызвали менеджер баз данных, после чего с его помощью, а также функции `remove()` удалили пост. Теперь протестируйте все эти функции. Если все работает, то тогда я вас поздравляю! Вы написали собственный примитивный блог на фреймворке Symfony. Если что то не работает, то повторите все шаги сначала и прочитай все более внимательно.

Теперь, когда вы уже ознакомились с тем как работает паттерн MVC на практике, а также создали своё первое приложение на фреймворке Symfony, можно сделать заключение об данном разделе. Если вас заинтересовала разработка веб-приложений на данном фреймворке, то тогда я вам советую заглянуть в официальную документацию фреймворка а также посмотреть бесплатные видео-уроки на YouTube.

Я дал вам только небольшие основы данного фреймворка. Если вам не понравился данный фреймворк, то я вам рекомендую попробовать другие, такие например как Laravel или CakePHP. И изучать по тому же самому пути, т.е. читать официальную документацию и смотреть видео-уроки на YouTube.

Заключение

Конец книги уже скоро. В этой главе вы узнали об том какие существуют фреймворки, поняли за каким принципом сейчас разрабатываются приложения, узнали об том как ведется разработка и вы создали собственный примитивный блог на основе фреймворка Symfony. И я думаю, некоторые новички были в шоке когда узнали что им придется все забыть и начать изучать фреймворк. Поскольку теперь, никто не разрабатывает приложения на чистом листе. А если разрабатывает, то тогда частенько такая система “изобретает велосипед” или является не надежной.

Те знания которые я вам предоставил в книге являются основами которые позволяют вам понимать работу фреймворка. Так что вот так сейчас и разрабатываются приложения, и это очень сильно облегчает работу программистам, как я и говорил. Книга, видео-уроки, курсы, не смогут дать те знания, которые получают программисты через опыт.

Самым лучшим способом развивать свой навык программирования и вообще любой навык - будет не только использование всех возможных ресурсов как например видео-уроки или книги, но также и создание какого-нибудь собственного проекта. Это может быть, например, интернет магазин, форум или вообще что-то новое. То что никто ещё не создавал до вас.

Поэтому пользуйтесь всеми ресурсами подряд и изучайте дальше программирование если это вам нравится. Я, как и всегда попрошу читателя ответить на вопросы и выполнить упражнения которые находятся ниже. Но перед этим хочу ещё добавить. Что дальше уже будет предпоследний раздел, который не будет иметь подразделов, а суть его будет заключаться в том что там больше будет речи об том где ещё можно использовать PHP и также об сообществе разработчиков на PHP.

Вообще планировалось чтобы в предпоследнем разделе книги была речь об шаблонах проектирования, но автор решил что об этом лучше не писать, поскольку это займет очень и очень много места. А

также лучше читателю самому найти книги по шаблонам проектирования на PHP. Шаблоны проектирования это отдельный разговор который требует времени и знаний для того чтобы его начать. Поэтому я завершаю этот раздел, и вы можете перейти к вопросам и упражнениям.

Вопросы и упражнения для самоконтроля

1. Что такое фреймворк?
2. Как вы можете описать разработку приложений в наше время?
3. Для чего нужны фреймворки?
4. Какие вы знаете фреймворки?
5. Объясните что такое MVC-паттерн, а также расскажите об его работе.
6. За что отвечает контроллер, модель и шаблон? Приведите простые примеры.
7. Какая библиотека используется для того чтобы работать с базами данных через Symfony?
8. Как происходит получение данных с помощью Symfony и какие есть функции для этого?
9. Что нужно сделать для того чтобы создать форму в Symfony?

20 Больше об PHP

Этот раздел будет кратким и вообще не будет содержать каких-то подразделов. Суть раздела заключается в том чтобы больше рассказать об PHP и где можно использовать данный язык программирования. Но также об том где ещё можно найти материалы по изучению данного языка программирования и о том какие существуют сообщества и конференции по PHP. Вопрос, где можно ещё использовать PHP кроме создания веб-приложений?

Сейчас идет развитие создания нейросетей на PHP. Если кто то не знает что такое нейросети, то нейросети это искусственный интеллект который имеет способность к обучению а также к использованию предыдущего опыта для лучшего выполнения алгоритмов. И вообще нейросети сейчас становятся очень популярными, и очень много компаний их используют. А где можно использовать нейросети?

К примеру, сейчас нейросети используют в распознавании изображений. Одним из простых примеров нейросетей может быть нейросеть которая умеет распознавать на изображениях цифры, которые были написаны человеческой рукой. Можно много привести примеров таких нейросетей.

Это один из способов, где ещё можно использовать PHP. Я думаю, читатель сейчас удивится когда услышит что с помощью PHP можно создавать программное обеспечение, а также даже писать игры. К примеру для того чтобы писать программное обеспечение на PHP можно воспользоваться программой PHP Delver Studio. Это одна из тех программ которые можно использовать для того чтобы создавать ПО на PHP.

Конечное же есть и другие варианты. С PHP можно много чего делать, если вам интересно можете также поискать много информации об этом в Google. Я назвал вам самые популярные фреймворки, приводит примеры, больше не буду. Поскольку я считаю, что несколько примеров с одним фреймворком достаточно чтобы понять как все работает.

Поскольку другие фреймворки имеют такой же самый принцип разработки приложения. Главное понимать принципы разработки, поскольку именно они дают основу для разработки чего то. Фреймворки обновляются, становятся ещё большими и функциональными, но принципы разработки остаются теми самими. И знание фреймворка не так уж и играет роль как знание принципов разработки.

Чтобы лучше разобраться с выбором фреймворка я рекомендую вам посмотреть некоторые видео-уроки на YouTube. Если вы хотите немного поиграться с PHP и больше подкрепить свои знания, то можете воспользоваться порталом <http://codecademy.com> - там вы сможете найти много разных онлайн заданий по программированию.

Кроме того что вы будете тренироваться, вы также будете учиться. А что же можно сказать об том какие существуют PHP конференции и ивенты? Самым простым способом найти какую-то конференцию по PHP будет зайти на официальный сайт разработчиков PHP и перейти в раздел событий <https://php.net/conferences/>.

Там вы можете найти много конференций которые проходят по всему миру, а также возможно вы сможете найти конференцию которая проходит в вашем городе. Конечно же вы можете поискать на веб-сайтах своего города какие будут проходить конференции и сходки разработчиков. Я рекомендую новичкам сходить на какую-то сходку

разработчиков по PHP или же по любому другому языку программирования.

Это важно, поскольку на таких конференциях и ивентах в вашем городе вы можете познакомиться с целым сообществом программистов с которыми вы можете хорошо подружиться. Но кроме того что вы сможете найти себе собеседников, вам также возможно будет легче найти работу с помощью таких ивентов.

А все потому что очень часто эти ивенты и конференции спонсируют многие ИТ-компании и часто им нужны новые сотрудники. И на этом я завершу этот раздел. Думаю, некоторые читатели меня спросят, но что же делать дальше? Что же делать, если мы уже выучили основы программирования на PHP?

Дальше следует выучить 1 один фреймворк для начала, почитать об шаблонах проектирования, попробовать создать что то своё и поработать с какими то библиотеками для PHP для того чтобы больше расширить своё кругозрение разработчика. Книга уже подходит к концу, вы прошли путь новичка, а впереди на вас ждет путь намного сложнее и интереснее чем этот. Поэтому пользуйтесь знаниями, полученными от этой книги и давайте уже сделаем итог о том что мы выучили.

Итог изученного

2/4 книги я говорил об основах программированию на PHP. А другую половину книги я давал знания, которые сейчас являются очень востребованными для младших разработчиков. Теперь настало время подбить и итоги и решить что же делать дальше читатель после того как он прочитал мою книгу.

Да, я уже говорил об этом немного в конце предыдущего раздела. Но, я должен дать список других ресурсов которые более помогут подкрепить знания. Ресурсы, которые я вам дам имеют множество других

интересных курсов по веб-разработке. Но суть заключается не только в том, чтобы подкрепить больше знания, но и также поделаться ещё больше реальных примеров.

- Codecademy - является очень и очень хорошим бесплатным ресурсом по программированию. Кроме того что вы сможете найти кучу бесплатных курсов по веб-разработке, вы сможете найти также задания которые помогут вам проверить их.
- Codeschool - также бесплатный ресурс, на данный момент имеет всего 2 курса по PHP. Но ими также хорошо воспользоваться чтобы больше подкрепить свои знания а также проверить их.
- Treehouse - это также хороший ресурс по изучению программирования, но тут вы получаете бесплатный доступ на некоторое время. После этого уже надо платить, если вы хотите дальше им пользоваться.
- Udemy - это целый портал, на котором можно найти очень и очень много разных курсов на разные темы. И среди них вы можете найти курсы по программированию. Некоторые могут быть платными, а другие бесплатными.

И это ещё не все ресурсы которые вы можете использовать для того чтобы научиться программировать. Я дал вам основу для того чтобы развиваться дальше в веб-разработке. Дальше уже для того чтобы больше научиться надо разрабатывать реальные проекты и иметь чёткую цель того что вы хотите сделать.

Некоторые изучают программирование чисто в целях хобби, некоторые для работы, а некоторые для личного бизнеса. Дальше уже это выбор читателя, в каких целях он хочет использовать свои знания программировать. Хочет ли он их развивать или хочет их использовать для создания чего-то нового - это уже его личное дело.

Возможно, я напишу улучшенное издание данной книги в будущем. Но на этом моменте я завершаю книгу, которую я писал полгода. Хочу поблагодарить читателя за то что он нашёл время чтобы прочитать эту новую книгу по PHP. Некоторые говорят, что PHP уже не в

моде, но это не так. Его до сих пор очень часто используют для того чтобы разрабатывать веб-сайты и разные приложения.

Поэтому используйте полученные знания от данной книги и я надеюсь что читатель будет дальше совершенствоваться не только в программировании, но вообще в любой сфере которая приносит пользу. Поэтому моё итоговое слово - спасибо.

Благодарности

Хочу поблагодарить Леонида К. за создание хорошего дизайна обложки для книги. Большое ему спасибо, поскольку он смог тем самым усовершенствовать мою книгу. Хотя в начале написания книги я уже создал первую версию обложки, но именно версия Леонида выглядела очень красиво и гармонично. Кроме этого Леонид также помог корректировать книгу и это очень мне помогло.

Благодарность Александру Г. который помог мне в написании некоторых разделов этой книги. Большое ему спасибо также за то что был одним из тех кто первый начал читать эту книгу в целях того чтобы проверить правильность материала.

В общем, спасибо тем, кто помог мне во время написания книги. Спасибо тем кто первыми брали и читали эту книгу ради того чтобы помочь в дополнении и корректировании материала. А также спасибо людям которые прочитали эту книгу, надеюсь, что они получили то чего ожидали от этой книги. Поэтому, спасибо!

- *Demian Kostelny. Программист. Писатель. - 2018 год.*

Структура книги

Предисловие от автора	2
0 Добро пожаловать в PHP!	3
0.1 Что такое PHP?	3
0.2 Как работает сервер	4
0.3 Установка локального сервера	6
0.4 Редактор кода	8
Заключение	9
1 Первые шаги	9
1.1 Комментарии в коде	10
1.2 Переменные в PHP	11
1.3 Немного об типах данных	13
1.4 Строки в PHP	14
1.4.1 Операторы для работы со строками	16
1.4.2 Управление последовательностью	17
1.4.3 Синтаксис heredoc и nowdoc	21
Вопросы и упражнения для самоконтроля	21
2 Числа	22
2.1 Операторы для работы с числами	22
2.2 Числа с плавающей точкой	24
Заключение	25

Вопросы и упражнения для самоконтроля	25
3 Логические операции	26
3.1 Псевдокод	26
3.2 Логическая операция if	27
3.3 Логические операторы	30
3.4 Логическая операция switch	36
3.5 Логический тип данных	39
3.6 Джордж Буль	40
Заключение	42
Вопросы и упражнения для самоконтроля	42
4 Циклы	43
4.1 Цикл while	43
4.2 Цикл do...while	44
4.3 Цикл for	45
Заключение	46
Вопросы и упражнения для самоконтроля	47
5 Массивы	47
5.1 Создание массивов	47
5.2 Операторы для массивов	49
5.3 Многомерные массивы	51
5.4 Массив ключа-значение	51
Вопросы и упражнения для самоконтроля	53
6 Функции	54
6.1 Создание функций	54
6.2 Функции для работы со строками	57
6.3 Функции для работы с числами	60
6.4 Функции для работы с массивами	63
6.5 Анонимные функции	66
Заключение	66
Вопросы и упражнения для самоконтроля	67
7 PHP и ООП	68
7.1 Классы	69

7.2 Конструкторы и деструктор	72
7.3 Абстрактные классы	74
7.4 Интерфейсы	75
7.5 Перегрузка и магические методы	76
7.6 Трейты	79
Заключение	79
Вопросы и упражнения для самоконтроля	80
8 Дата и файлы	81
8.1 Работа с файлами	81
8.2 Смена прав доступа	84
8.3 Проверка файлов	87
8.4 Функции для работы с файлами	89
8.5 Дата	95
9 PHP и HTTP	104
9.1 Работа с формами	105
9.2 HTTP-методы	109
9.3 PHP и Cookies	117
9.4 \$_SERVER	119
Вопросы и упражнения для самоконтроля	121
10 PHP и базы данных	122
10.1 MySQL	125
10.2 Таблицы в SQL	131
10.3 Типы данных в SQL	137
10.4 Операторы в SQL	144
10.5 Агрегатны функции	146
10.6 Транзакции	148
10.7 Резервное копирование базы данных	149
Вопросы и упражнения для самоконтроля	151
11 Доступ к MySQL через PHP	152
11.1 Доступ к MySQL через PHP	154
11.2 Создание запросов к MySQL в PHP	157
11.3 Создание блога	166

11.4 Загрузка файлов и MySQL	166
Вопросы и упражнения для самоконтроля	171
12 Сессии	172
12.1 Общие понятия об сессиях	173
12.2 Создание регистрации	174
12.3 Создание аутентификации	178
12.4 Просмотр аккаунта	180
Вопросы и упражнения для самоконтроля	182
13 Безопасность	182
13.1 Фильтрация данных	183
13.2 Безопасность файловой системы	191
13.3 Криптография	194
13.4 Шифры	203
13.5 Хэширование паролей	218
Вопросы и упражнения для самоконтроля	221
14 Работа с другими веб-службами	222
14.1 Что такое JSON?	223
14.2 JSON и PHP	225
Заключение	235
Вопросы и упражнения для самоконтроля	236
15 Ошибки в коде	237
15.1 Виды ошибок	237
15.2 Управление ошибками	239
15.3 Отладка кода	242
Заключение	246
Вопросы и упражнения для самоконтроля	246
16 Пакетный менеджер	247
16.1 Установка	248
16.2 Использование	248
Заключение	252
Вопросы и упражнения для самоконтроля	254
17 Отправка электронных писем	255

17.1 Основы	256
17.2 Загрузка файлов	256
17.3 Создание рассылки	259
17.4 РНР средства	261
Заключение	263
Вопросы и упражнения для самоконтроля	265
18 РНР и тестирование	266
18.1 Установка	267
18.2 Первый тест	267
18.3 Data provider	268
18.4 Больше методов для написания тестов	272
Заключение	275
Вопросы и упражнения для самоконтроля	277
19 Фреймворки	278
19.1 Какие есть РНР фреймвокри	279
19.2 Общее понятия	279
19.3 Фреймворк Symfony	280
19.4 Создание блога	282
Заключение	286
Вопросы и упражнения самоконтроля	308
20 Больше об РНР	309
Итог изученного	310
Благодарности	312
Структура книги	315