# Лабораторная работа №6

Тема: Изучение PHP

**Задание**

На основе лабораторной работы №5 произвести разделение страницы на шаблоны и блоки, научиться методам GET и POST и добавить возможность загрузки изображения с помощью PHP.

**Требования**

* Шаблонизация должна состоять из базового шаблона (layot), шаблона страницы, блока статьи/поста и шаблонизатора;
* Данные из статей/постов должны быть вынесены в массив или файл;
* После отправки формы данные должны быть записаны в массив/файл;
* Загруженное изображение должно быть скопировано в папку в директории проекта и отображено на странице.

# Справочная информация

PHP — это скриптовый интерпретируемый язык программирования, специально разработанный для написания web-приложений (сценариев), исполняющихся на Web-сервере.

Аббревиатура PHP означает “Hypertext Preprocessor (Препроцессор Гипертекста)".

Синтаксис языка берет начало из C, Java и Perl и достаточно прост для изучения. Преимуществом PHP является предоставление web-разработчикам возможности быстрого создания динамически генерируемых web-страниц.

PHP-код может быть встроен непосредственно в html страницы, которые, в свою очередь будут корректно обрабатываться PHP-интерпретатором.

Особенности серверных языков

Важной стороной работы серверных языков является возможность организации непосредственного взаимодействия с системой управления базами данных (или СУБД) — сервером базы данных, в которой упорядоченно хранится информация, которая может быть вызвана в любой момент.

Это позволяет обрабатывать данные форм, генерировать динамические страницы, получать запросы и производить все действия, требующие сохранения и изменения данных.

PHP принимает входящий запрос от веб-сервера, выполняет сценарий и возвращает веб-серверу результат в виде готового HTML-кода. Сервер отправляет этот результат в браузер пользователю, который, в свою очередь, отображает её пользователю.

Примеры задач, решаемых серверными языками

* Проверка авторизации пользователя;
* Управление сессиями;
* Загрузка изображений и файлов на сервер;
* Страница курсов валют;
* Отправление и получение сообщений;
* Редактирование постов;
* Добавление товара в корзину;
* Вывод счетчика посещений и т.д.

## Установка PHP

Для выполнения лабораторных работ потребуется как минимум три вещи: PHP, веб-сервер Apache/Nginx, СУБД MySQL и веб-приложение phpMyAdmin, представляющее собой веб-интерфейс для администрирования MySQL.

Каждый из этих компонентов можно установить отдельно или воспользоваться пакетом программного обеспечения Open Server Panel.

OSPanel является портативным программным комплексом и не требует установки. Сборку можно разместить на внешнем жёстком диске или флэш-накопителе, это позволит использовать OSPanel на любом компьютере, который отвечает системным требованиям.

Дистрибутив представляет собой самораспаковывающийся архив в формате 7ZIP (расширение .exe). Запустите исполняемый файл дистрибутива и выберите путь для распаковки файлов.

Скачать можно с официального сайта <https://ospanel.io/>

Для работы будет достаточно базовой версии.

Для запуска OSPanel используйте файл OSPanel.exe. После старта программы вы увидите красный флажок в трее Windows (область возле системных часов). Чтобы включить непосредственно сам веб-сервер нажмите на флажок, далее выберите пункт меню [Меню → Запустить].

Для того, чтобы запускать свои программы с помощью OSPanel, поместите их в папку domains. Они появятся во вкладки «мои сайты» контекстного меню.

## Особенности языка PHP

### Основы языка

Простейшая программа "Hello, World!" на языке PHP выглядит следующим образом:

<?php

echo "Hello, World!";

?>

Код PHP заключается в специальные теги <?php и ?>.

Оператор echo осуществляет вывод информации на экран. Также текст можно обернуть в html-теги, например так:

<?php

echo "<html><body>";

echo "<h1>Hello, World!</h1>";

echo "</body></html>";

?>

Такая программа выведет "Hello, World!" заглавным шрифтом.

PHP-код можно встраивать в HTML. Данная запись выведет результат, аналогичный примеру выше:

<html>

<body>

<?php

echo "<h1>Hello, World!</h1>";

?>

</body>

</html>

В обоих случаях файл должен иметь расширение .php.

Когда PHP обрабатывает файл, он просто передаёт его текст, пока не встретит один из специальных тегов, который сообщает ему о необходимости начать интерпретацию текста как кода PHP. Затем он выполняет весь найденный код до закрывающего тега, говорящего интерпретатору, что далее снова идет просто текст.

Комментарии в PHP могут быть трех типов:

// Тип 1

/\* Тип2 (многострочный) \*/

# Тип3

Переменные в PHP

Имена всех переменных в PHP должны начинаться со знака $. Имена переменных чувствительны к регистру букв: например, $var — не то же самое, что $Var или $VAR:

В PHP не нужно ни описывать переменные явно, ни указывать их тип. Не допускается начинать имя переменной с цифры, а также использовать любые значения, кроме букв алфавита и знака подчеркивания.

<?php

$var = "Bob";

echo $var;

?>

### Типы данных

1. Двоичные данные (boolean): true, false
2. Целые числа (Integer): 1,2,3…
3. Числа с плавающей точкой (Float): 1.234
4. Строки (String): 'текст', "текст"
5. Массивы (Array): см.ниже

Многомерные: $A["Ivanov"] = array("name"=>"Иванов И.И.", "age"=>"25");

1. Объекты (Object): см. ниже
2. Ресурсы (Resource): см. ниже
3. Пустой тип (NULL): см. ниже
4. Смешанный (Mixed): параметр может принимать несколько типов
5. Числа (Number): параметр может быть либо integer, либо float
6. Обратного вызова (Callback): см. ниже

Тип arrays (массивы)

Массив в PHP - это упорядоченный набор данных, в котором устанавлено соответствие между значением и ключом.

Индекс (ключ) служит для однозначной идентификации элемента внутри массива. В одном массиве не может быть двух элементов с одинаковыми индексами.

PHP позволяет создавать массивы любой сложности. Рассмотрим некоторые примеры:

Простой массив (список)

Массивы, индексами которых являются числа, начинающиеся с нуля - это списки:

<?php

$names[0]="Апельсин";

$names[1]="Банан";

$names[2]="Груша";

$names[3]="Помидор";

// Здесь: names - имя массива, а 0, 1, 2, 3 - индексы массива

?>

Ассоциативные массивы

В PHP индексом массива может быть не только число, но и строка. Причем на строку не накладываются ограничения: она может содержать пробелы, специальные символы и быть любой длины.

Массивы, индексами которых являются строки, называются ассоциативными массивами. Индексы ассоциативных массивов называются ключами. Пример ассоциативного массива:

<?php

// Ассоциативный массив

$names["Иванов"]="Иван";

$names["Сидоров"]="Николай";

/\* В данном примере: фамилии - ключи ассоциативного массива, а имена - элементы массива\*/

?>

Многомерные массивы

Для создания массивов в PHP существует специальная инструкция array(). Ее удобно использовать для создания многомерных массивов. Приведем конкретный пример:

<?php

$A["Ivanov"] = array("name"=>"Иванов И.И.", "age"=>"25", "email"=>"ivanov@mail.ru");

$A["Petrov"] = array("name"=>"Петров П.П.", "age"=>"34", "email"=>"petrov@mail.ru");

?>

Тип object (объекты)

Объект является одним из базовых понятий объектно-ориентированного программирования. Для инициализации объекта используется выражение new, создающее в переменной экземпляр объекта.

<?php

class foo

{

function do\_foo()

{

echo "Doing foo.";

}

}

$bar = new foo;

$bar->do\_foo();

?>

Тип resource (ресурсы)

Ресурс - это специальная переменная, содержащая ссылку на внешний ресурс. Ресурсы создаются и используются специальными функциями. Полный перечень этих функций и соответствующих типов ресурсов смотрите здесь.

Тип NULL (пустой тип)

Специальное значение NULL говорит о том, что эта переменная не имеет значения. NULL - это единственно возможное значение типа NULL (пустой тип).

Переменная считается NULL если:

* ей была присвоена константа NULL;
* ей еще не было присвоено какое-либо значение;
* она была удалена с помощью unset().

Псевдотип callback (обратного вызова)

Некоторые функции, такие как call\_user\_func() или usort() принимают в качестве параметра определенные пользователем callback-функции. Callback-функции могут быть не только простыми функциями, но также методами объектов, включая статические методы классов.

PHP-функция передается просто как строка ее имени. Вы можете передать любую встроенную или определенную пользователем функцию за исключением array(), echo(), empty(), eval(), exit(), isset(), list(), print() и unset().

Приведем примеры callback функций:

<?php

// простой пример callback

function my\_callback\_function() {

echo 'hello world!';

}

call\_user\_func('my\_callback\_function');

// примеры callback-метода

class MyClass {

function myCallbackMethod() {

echo 'Hello World!';

}

}

// вызов метода статического класса без создания объекта

call\_user\_func(array('MyClass', 'myCallbackMethod'));

// вызов метода объекта

$obj = new MyClass();

call\_user\_func(array(&$obj, 'myCallbackMethod'));

?>

Константы в PHP

Константой называется именованная величина, которая не изменяется в процессе выполнения программы. Константы могут содержать только скалярные данные (логического, целого, плавающего и строкового типов).

В РНР константы определяются функцией define(). Эта функция имеет следующий формат:

define ($name, $value, $case\_sen), где:

$name - имя константы;

$value - значение константы;

$case\_sen - необязательный параметр логического типа, указывающий, следует ли учитывать регистр букв (true) или нет (false).

Пример определения и использования констант в PHP:

<?php

define("pi",3.14,true);

echo pi;

// Выводит 3.14

?>

Различия между константами и переменными:

* У констант нет приставки в виде знака доллара ($);
* Константы можно определить только с помощью функции define(), а не присваиванием значения;
* Константы могут быть определены и доступны в любом месте без учета области видимости;
* Константы не могут быть определены или аннулированы после первоначального объявления;
* Константы могут иметь только скалярные значения.

### Операторы

Арифметические операторы

Простые + (сложение), - (вычитание, отрицание), /(деление), \*(умножение), а также деление по модулю %, возведение в степень \*\* .

Операция деления ("/") всегда возвращает вещественный тип, даже если оба значения были целочисленными (или строками, которые преобразуются в целые числа).

PHP также поддерживает операторы инкремента и декремента ($a++, ++$a, $a--, --$a).

Операторы присвоения

Базовый оператор присвоения обозначается как =.

PHP позволяет использовать конструкции вида:

<?php

$a = ($b = 4) + 5; // результат: $a установлена значением 9, переменной $b присвоено 4.

?>

В дополнение к базовому оператору присвоения имеются "комбинированные операторы" для всех бинарных арифметических и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения. Например:

Также поддерживаются составные операторы вида $a += 5.

Побитовые операторы

$a & $b - Побитовое 'и'

$a | $b - Побитовое 'или'

$a ^ $b - Исключающее или

~ $a - Отрицание

$a << $b - Сдвиг влево

$a >> $b - Сдвиг вправо

Операторы сравнения

Меньше <, больше >, меньше или равно <=, больше или равно >=, равно ==, не равно !== или <>, тождественно равно === , тождественно не равно !===.

Логические операторы

$a and $b - Логическое 'и'

$a or $b - Логическое 'или'

$a xor $b - Исключающее 'или'

! $a - Отрицание

$a && $b - Логическое 'и'

$a || $b - Логическое 'или'

Строковые операторы

Оператор конкатенации ('.') возвращает объединение левого и правого аргумента. Оператор присвоения с конкатенацией присоединяет правый аргумент к левому.

Пример:

<?php

$a = "Hello ";

$b = $a . "World!"; // $b содержит строку "Hello World!"

$a = "Hello ";

$a .= "World!"; // $a содержит строку "Hello World!"

?>

### Конструкции PHP

**Условные операторы:**

if, else, elseif

Конструкция if является одной из наиболее важных во многих языках программирования, в том числе и PHP. Она предоставляет возможность условного выполнения фрагментов кода. Структура if реализована в PHP по аналогии с языком C:

if (выражение)

инструкция

Пример:

<?php

if ($a > $b) {

echo "a больше, чем b";

} elseif ($a == $b) {

echo "a равен b";

} else {

echo "a меньше, чем b";

}

?>

switch

Оператор switch похож на ряд операторов IF с одинаковым условием. Во многих случаях вам может понадобиться сравнивать одну и ту же переменную (или выражение) с множеством различных значений и выполнять различные участки кода в зависимости от того, какое значение принимает эта переменная (или выражение). Это именно тот случай, для которого удобен оператор switch.

switch ($i) {

case 0:

echo "i равно 0";

break;

case 1:

echo "i равно 1";

break;

case 2:

echo "i равно 2";

break;

}

?>

**Циклы**

**while**

Циклы while являются простейшим видом циклов в PHP. Они ведут себя так же, как и в языке C. Простейшей формой цикла while является следующее выражение:

while (expr)

statement

Смысл выражения while очень прост. Оно указывает PHP выполнять вложенные выражения повторно до тех пор, пока выражение в самом while является TRUE.

**do-while**

Цикл do-while очень похож на цикл while, с тем отличием, что истинность выражения проверяется в конце итерации, а не в начале. Главное отличие от обычного цикла while в том, что первая итерация цикла do-while гарантированно выполнится (истинность выражения проверяется в конце итерации), тогда как она может не выполниться в обычном цикле while (истинность выражения которого проверяется в начале выполнения каждой итерации, и если изначально имеет значение FALSE, то выполнение цикла будет прервано сразу).

**for**

Цикл for самый сложный цикл в PHP. Он ведет себя так же, как и в языке C. Синтаксис цикла for следующий:

for (expr1; expr2; expr3)

statement

Первое выражение (expr1) всегда вычисляется (выполняется) только один раз в начале цикла.

В начале каждой итерации оценивается выражение expr2. Если оно принимает значение TRUE, то цикл продолжается и выполняются вложенные операторы. Если оно принимает значение FALSE, выполнение цикла заканчивается.

В конце каждой итерации выражение expr3 вычисляется (выполняется).

for ($i = 1; $i <= 10; $i++) {

echo $i;

}

**foreach**

Конструкция foreach предоставляет простой способ перебора массивов. Существует два вида синтаксиса:

foreach (array\_expression as $value)

statement

foreach (array\_expression as $key => $value)

statement

Foreach работает только с массивами и объектами и будет генерировать ошибку при попытке использования с переменными других типов или неинициализированными переменными.

**break**

break прерывает выполнение текущей структуры for, foreach, while, do-while или switch.

break принимает необязательный числовой аргумент, который сообщает ему выполнение какого количества вложенных структур необходимо прервать. Значение по умолчанию 1, только ближайшая структура будет прервана.

$arr = array('один', 'два', 'три', 'четыре', 'стоп', 'пять');

foreach ($arr as $val) {

if ($val == 'стоп') {

break; /\* Тут можно было написать 'break 1;'. \*/

}

echo "$val<br />\n";

}

**continue**

continue используется внутри циклических структур для пропуска оставшейся части текущей итерации цикла и, при соблюдении условий, начала следующей итерации.

foreach ($arr as $key => $value) {

if (!($key % 2)) { // пропуск четных чисел

continue;

}

do\_something\_odd($value);

}

### Функции

Особенности пользовательских функций PHP:

* Доступны параметры по умолчанию. Есть возможность вызывать одну и ту же функцию с переменным числом параметров;
* Пользовательские функции могут возвращать любой тип;
* Область видимости переменных внутри функции является иерархической (древовидной);
* Есть возможность изменять переменные, переданные в качестве аргумента.
* При использовании пользовательских функций встает вопрос об области видимости переменных.

Переменные по области видимости подразделяются на глобальные и локальные.

Глобальные переменные - это переменные, которые доступны всей программе, включая подпрограммы (функции).

Локальные переменные - переменные, определенные внутри подпрограммы (функции). Они доступны только внутри функции, в которой они определены.

Для PHP все объявленные и используемые в функции переменные по умолчанию локальны для функции. То есть, по умолчанию нет возможности изменить значение глобальной переменной в теле функции.

Если вы в теле пользовательской функции будете использовать переменную с именем, идентичным имени глобальной переменной (находящейся вне пользовательской функции), то никакого отношения глобальной переменной эта локальный переменная иметь не будет. В данной ситуации в пользовательской функции будет создана локальная переменная с именем, идентичным имени глобальной переменной, но доступна данная локальная переменная будет только внутри этой пользовательской функции.

Поясним данный факт на конкретном примере:

<?php

$a = 100;

function func() {

$a = 70;

echo "$a";

}

func();

echo "$a";

?>

Сценарий выведет сперва 70, а затем 100:

Для избавления этого, в PHP существует специальная инструкция global, позволяющая пользовательской функции работать с глобальными переменными.

<?php

$a = 100;

function func() {

global $a;

echo $a;

}

func();

echo "$a";

?>

Данный сценарий выведет 100 дважды.

Помимо локальных и глобальных переменных, в PHP существует еще один тип переменных - статические переменные.

Если в теле пользовательской функции объявлена статическая переменная, то компилятор не будет ее удалять после завершения работы функции. Пример работы пользовательской функции, содержащей статические переменные:

<?php

function func()

{

static $a;

$a++;

echo "$a";

}

for ($i = 0; $i++<10;) func();

?>

Данный сценарий выводит строку:

1 2 3 4 5 6 7 8 9 10

Если мы удалим инструкцию static, будет выведена строка:

1 1 1 1 1 1 1 1 1 1

Это связано с тем, что переменная $a будет удаляться при завершении работы функции и обнуляться при каждом ее вызове. Переменная $a инкрементируется сразу после обнуления, а только потом выводится.

Создание пользовательских функций

Пользовательская функция может быть объявлена в любой части программы (скрипта), до места ее первого использования. И не нужно никакого предварительного объявления.

Синтаксис объявления функций следующий:

function Имя (аргумент1[=значение1],...,аргумент1[=значение1])

{

тело\_функции

}

Объявление функции начинается служебным словом function, затем следует имя функции, после имени функции - список аргументов в скобках. Тело функции заключается в фигурные скобки и может содержать любое количество операторов.

Требования, предъявляемые к именам функций:

* Имена функций могут содержать русские буквы, но давать функциям имена, состоящие из русских букв не рекоммендуется;
* Имена функций не должны содержать пробелов;
* Имя каждой пользовательской функции должно быть уникальным. При этом, необходимо помнить, что регистр при объявлении функций и обращении к ним не учитывается. То есть, например, функции funct() и FUNCT() имеют одинаковые имена;
* Функциям можно давать такие же имена, как и переменным, только без знака $ в начале имен.
* Типы значений, возвращаемые пользовательскими функциями, могут быть любыми. Для передачи результата работы пользовательских функций в основную программу (скрипт) используется конструкция return. Если функция ничего не возвращает, конструкцию return не указывают. Конструкция return может возвращать все, что угодно, в том числе и массивы.

Приведем примеры использования пользовательских функций:

<?php

function funct() {

$number = 777;

return $number;

}

$a = funct();

echo $a;

?>

В рассмотренном примере функция funct возвращает с помощью инструкции return число 777. Возвращенное функцией значение присваивается глобальной переменной $a, а затем оператор echo выводит значение переменной $a в браузер. В результате мы увидим в браузере число 777.

Передача аргументов пользовательским функциям

При объявлении функции можно указать список параметров, которые могут передаваться функции, например:

<?php

function funct($a, $b, /\* ..., \*/ $z) { ... };

?>

При вызове функции funct() нужно указать все передаваемые параметры, поскольку они являются обязательными. В PHP пользовательские функции могут обладать необязательными параметрами или параметрами по умолчанию, но об этом позже.

Передача аргументов по ссылке

Если вы хотите, что бы аргумент передавался по ссылке, вы должны указать амперсанд (&) перед именем аргумента в описании функции:

<?php

function funct(&$string)

{

$string .= 'а эта внутри.';

}

$str = 'Эта строка за пределами функции, ';

funct($str);

echo $str; // Выведет 'Эта строка за пределами функции, а эта внутри.'

?>

Параметры по умолчанию

В PHP функции могут возвращать любые значения в зависимости от переданных им параметров.

<?php

function makecup($type = "чая")

{

return "Сделайте чашечку $type.\n";

}

echo makecup();

echo makecup("кофе");

?>

Результат работы приведенного скрипта будет таким:

Сделайте чашечку чая

Сделайте чашечку кофе

Значение по умолчанию должно быть константным выражением.

### Подключение файлов

Для того, чтобы подключить один сценарий внутри другого существуют инструкции require и include:

require 'some\_file.php';

include 'some\_file.php';

При подключении сценария в файл, весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определен внутри этой функции. То есть, он будет в той же области видимости переменных этой функции. Таким образом, подключение файлов — это просто их склеивание в один.

В PHP нет системы модулей, которая существует в других языках программирования (Python, Java, JavaScript). Невозможно «импортировать» только отдельные переменные или функции из подключаемого сценария.

Из этого также следует, что если подключить один сценарий дважды, то переменные и функции из него тоже обьявятся повторно, а это может вызывать ошибку. Поэтому используйте include\_once или require\_once, чтобы такого не произошло.

Метод require аналогичен include, за исключением того, что при ошибке он также выдаст фатальную ошибку уровня E\_COMPILE\_ERROR. Другими словами, он остановит выполнение скрипта, тогда как include только выдал бы предупреждение E\_WARNING, которое позволило бы скрипту продолжить выполнение.

## Шаблонизация

Шаблонизация – это разделение веб-страницы на независимые блоки, которые можно в дальнейшем подключать и вкладывать друг в друга.

Шаблонизация позволяет многократно использовать статичные элементы, такие как меню или шапка сайта. Также с помощью шаблонизации создаются динамические элементы, например посты, товары или статьи. Это означает, что данные могут меняться в зависимости от различных условий. Сама информация, как правило, хранится в базе данных, а PHP-сценарий извлекает её оттуда и передаёт в шаблон.

Термины шаблонизации

Лейаут — это шаблон, который содержит HTML-код, общий для всех страниц сайта. В нём могут содержаться подключение стилей, мета-теги, шапка, подвал. Также лейаут содержит область для вставки уникального содержимого каждой страницы.

Шаблон страницы — это шаблон с уникальным для одной страницы HTML-кодом. Например, для главной страницы там может быть список новостей. Также шаблон страницы может включать в себя блоки.

Блок — это шаблон очень небольшого блока страницы. Например, это может быть один элемент списка новостей. Удобство блоков в том, один блок могут включать разные страницы.

Шаблонизатор — это функция, которая подключает файл шаблона, передаёт ему данные и возвращает сгенерированный HTML. Он скрепляет воедино отдельные шаблоны в итоговую страницу.

Работает он следующим образом: PHP-сценарий страницы выполяет все действия для подготовки необходимой информации, к примеру, запрашивает записи из базы данных. Эти записи в виде массива отправляются шаблонизатору вместе с именем шаблона страницы.

Шаблонизатор подключает указанный файл шаблона и передаёт туда всю информацию. Но, вместо вывода на экран содержимого этого шаблона, он захватывает получившийся HTML-код и возвращает его.

Затем сценарий вызывает шалбонизатор ещё раз, но теперь с его помощью подключает общий лейаут, куда отправляется общая информация, а также содержимое страницы, полученное из предыдущего шага. Весь результат работы выводится на экран.

Пример использования

Посмотрим на примере, как это всё работает. Начнём с того, что определим три шаблона: макет, шаблон страницы и какой-нибудь блок.

[layout.php]

<!DOCTYPE html>

<html lang="ru">

<head>

<title> <?= $title; ?> </title>

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

</head>

<body>

<header>Дневник погоды</header>

<div class="main-content">

<main class="content"><?= $content; ?></main>

</div>

<footer class="main-footer">Дневник наблюдения за погодой. </footer>

</body>

</html>

Шаблон страницы:

**[main.php]**

<div class="content\_\_main-col">

<h2>Последние записи</h2>

<a class="button" href="/add">Добавить новую</a>

<ul>

<?php foreach ($items as $item): ?>

<?=renderTemplate('item.php', ['item' => $item]);?>

<?php endforeach; ?>

</ul>

</div>

Обратите внимание, что шаблон страницы помимо интерации по массиву, для каждого его элемента вызывает функцию-шаблонизатор. Шаблонизатор получает контент из шаблона блока и показывает его внутри списка.

Шаблон блока для показа одной записи:

[item.php]

<li >

<div>

<img src="uploads/preview <?=$item['path'];?>">

</div>

<h3>

<a href="/view?id=<?=$item['id'];?>"><?=$item['title'];?></a>

</h3>

<div>

<span class="temp"><?=$item['temp'];?></span>

</div>

</li>

Так выглядели три шаблона. Все они примут участие в формировании итоговой страницы. Соберёт эти шаблоны и выведет страницу на экран наш сценарий — index.php:

[index.php]

<?php

// двумерный массив со списком записей

$items\_list = [];

// HTML код главной страницы

$page\_content = renderTemplate('main.php', ['items' => $items\_list]);

// окончательный HTML код

$layout\_content = renderTemplate('layout.php',

['content' => $page\_content, 'title' => 'Дневник наблюдений за погодой']);

// вывод на экран итоговой страницы

print($layout\_content);

## Работа с HTML формами

В качестве примера рассмотрим форму обратной связи. Такая форма в HTML может выглядеть следующим образом:

<form name="feedback" method="POST" action="form.php">

<label>Ваше имя: <input type="text" name="name"></label>

<label>Ваш email: <input type="text" name="email"></label>

<label>Сообщение: <textarea name="message"></textarea></label>

<input type="submit" name="send" value="Отправить">

</form>

Почти весь приведённый код описывает внешний вид и содержание формы, но следует обратить внимание на два атрибута тега <form>, которые нужны для указания на способ обработки данных:

method — этот атрибут используется для определения метода HTTP, который будет использован для передачи данных на сервер. Вы уже знакомы с HTTP-методом GET, предписывающим серверу просто вернуть определённый документ.

Метод POST сообщает о намерении передать на сервер некоторую информацию, что, впрочем, не отменяет последующее получение контента.

action — содержит адрес PHP-скрипта, который должен обработать эту форму.

После нажатия на кнопку «отправить», браузер выполняет POST запрос со введёнными данными на адрес, указанный в атрибуте action.

Обработка формы

После отправки формы управление передаётся PHP-скрипту, который должен получить переданные данные, выполнить с ними какие-либо действия (например, сохранить в базе данных) и показать результат.

Результатом может быть какое-нибудь сообщение об успешном завершении операции, например, «ваши данные успешно отправлены».

В первую очередь требуется научиться получать данные из формы в сценарии.

В PHP это делается легко — все данные из формы находятся в глобальном ассоциативном массиве $\_POST. Этот массив всегда будет неявно присутствовать в сценарии, если он был загружен по методу POST. Если был использован метод GET, данные будут в массиве $\_GET.

Каждое поле из формы будет находиться в массиве, где ключом будет значение атрибута name, а значением содержимое поля.

Например, чтобы вывести из формы всю информацию на экран, можно написать такой сценарий:

<?php

if (isset($\_POST)) {

print("Имя: " . $\_POST['name']);

print("<br>Email: " . $\_POST['email']);

print("<br>Сообщение: " . $\_POST['message']);

}

Функция isset служит для определения, существует ли переданная ей переменная. Так мы проверяем, что сценарий загружен методом POST, то есть была отправлена форма.

Как правило, после обработки формы в PHP, сценарий должен переадресовать пользователя на другую страницу. Это связано с тем, что если форма была отправлена через метод POST, то после обновления страницы данные будут отправлены ещё раз, а это, в большинстве случаев, нежелательное поведение.

## Загрузка файлов на сервер

Кроме текстовой информации, существует возможность отправлять на сервер файлы любых типов.

Загрузка файла на сервер осуществляется с помощью multipart-формы, в которой есть поле загрузки файла. Такая форма обязательно должна содержать атрибут enctype со значением multipart/form-data. Сам файл загружается при помощи поля с типом «file».

Пример формы для загрузки файла:

<form name="file\_upload" method="POST" action="form.php" enctype="multipart/form-data">

<label>Ваше фото: <input type="file" name="filename"></label>

<input type="submit" name="send" value="Отправить файл">

</form>

Обработка multipart-форм происходит следующим образом: получив файл, PHP сохраняет его во временном каталоге upload\_tmp\_dir, имя файла выбирается случайным образом. Затем он создает четыре переменных суперглобального массива $\_FILES. Этот массив содержит информацию о загруженном файле.

Содержимое массива $\_FILES для нашего примера приведено ниже. Имя файла должно соответствовать имени файла в multipart-форме.

$\_FILES['filename']['name'] - имя файла до его отправки на сервер, например, pict.gif;

$\_FILES['filename']['size'] - размер принятого файла в байтах;

$\_FILES['filename']['type'] - MIME-тип принятого файла (если браузер смог его определить), например: image/gif, image/png, image/jpeg, text/html;

$\_FILES['filename']['tmp\_name'] (так мы назвали поле загрузки файла) - содержит имя файла во временном каталоге, например: /tmp/phpV3b3qY;

$\_FILES['filename']['error'] - Код ошибки, которая может возникнуть при загрузке файла.

Пример PHP-обработчика:

<?php

if (isset($\_FILES['filename'])) {

$file = $\_FILES['filename'];

print("Загружен файл с именем " . $file['name'] . " и размером " . $file['size'] . " байт");

}

PHP автоматически сохраняет все загруженные файлы во временную папку на сервере. Но хранить там файлы нельзя, потому что эта директория периодически очищается, и ссылку на такой файл нельзя дать на сайте. Решение здесь только одно — переместить загруженный файл в другую папку. Перемещение файла всегда выполняют сразу после загрузки.

Для начала нужно убедиться, что в рабочей директории проекта существует папка для хранения загруженных файлов. Пусть она называется uploads.

Перемещение загруженного файла

Для перемещения файла нужно знать, где он находится сейчас, и адрес папки, в которую он будет переноситься.

С текущим адресом всё крайне просто — он уже находится в массиве $\_FILES. Новый адрес файла, в свою очередь, состоит из пути к папке и имени файла. Так как папка uploads находится там же, где и текущий сценарий, получить путь к ней можно так: dirname(\_\_FILE\_\_).

Код для перемещения файла в новую папку:

<?php

$current\_path = $\_FILES['avatar']['tmp\_name'];

$filename = $\_FILES['avatar']['name'];

$new\_path = dirname(\_\_FILE\_\_) . '/' . $filename;

move\_uploaded\_file($current\_path, $new\_path);

Функция move\_uploaded\_file() проверяет, что файл действительно загружен через форму и перемещает загруженный файл по новому адресу.

## Работа с Cookies

Протокол HTTP действует по схеме «запрос-ответ» и не сохраняет данные пользователей. Каждый запрос обрабатывается заново. Это создает лишнюю нагрузку на сервер и не позволяет запоминать действия и данные пользователей, например историю просмотров или последние действия. Такую возможность предоставляют сookies.

Cookies (в дальнейшем просто «куки») — небольшие фрагменты данных, которые веб-сервер отправляет браузеру. Браузер сохраняет их у себя, а при следующем посещении веб-страницы отправляет обратно. Благодаря этому, веб-сервер сможет узнать своего «старого» посетителя и идентифицировать его.

С технической стороны, куки — это обычные HTTP заголовки.

Когда веб-сервер хочет записать куки в браузер пользователя, он отсылает специальный заголовок ответа с названием Set-Cookie. В этом заголовке должна содержаться необходимая информация и дополнительные аттрибуты.

В следующий раз, когда браузер пользователя запросит веб-страницу с того же сайта, в числе прочих заголовков он передаст заголовок запроса Cookie. Веб-сервер получит эту информацию, и она будет доступна также и для PHP.

Примеры использования Cookies:

* аутентификация пользователя;
* хранение персональных предпочтений и настроек пользователя;
* отслеживание состояния сеанса доступа пользователя;
* сведения статистики о пользователях.

Рассмотрим использование Cookies на конкретном примере.

Предположим, нам нужно написать счетчик посещения сайта. Нам нужно знать, какое число посещений сайта осуществлялось каждым конкретным посетителем.

Программирование Cookies

Для установки Cookies используется функция SetCookie(). Для этой функции можно указать шесть параметров, один из которых является обязательным:

name - задает имя (строка), закрепленное за Cookie;

value - определяет значение переменной (строка);

expire - время "жизни" переменной (целое число). Если данный параметр не указать, то Cookie будут "жить" до конца сессии, то есть до закрытия браузера. Если время указано, то, когда оно наступит, Cookie самоуничтожится.

path - путь к Cookie (строка);

domain - домен (строка). В качестве значения устанавливается имя хоста, с которого Cookie был установлен;

secure - передача Cookie через защищенное HTTPS-соединение.

Обычно используются только три первые параметра.

Пример установки Cookies:

<?php

// Устанавливаем Cookie до конца сессии:

SetCookie("Test","Value");

// Устанавливаем Cookie на один час после установки:

SetCookie("My\_Cookie","Value",time()+3600);

?>

При использовании Cookies необходимо иметь в виду, что Cookies должны устанавливаться до первого вывода информации в браузер. Поэтому желательно устанавливать Cookies в самом начале скрипта. Cookies устанавливаются с помощью определенного заголовка сервера, а если скрипт выводит что-либо, то это означает, что начинается тело документа. В результате Cookies не будут установлены и может быть выведено предупреждение. Для проверки успешности установки Cookies можно использовать такой метод:

<?php

// Устанавливаем Cookie до конца сессии:

// В случае успешной установки Cookie, функция SetCookie возвращает TRUE:

if (SetCookie("Test","Value")) echo "<h3>Cookies успешно установлены!</h3>";

?>

Функция SetCookie() возвращает TRUE в случае успешной установки Cookie. В случае, если Cookie установить не удается SetCookie() возвратит FALSE и возможно, предупреждение (зависит от настроек PHP). Пример неудачной установки Cookie:

<?php

// Cookies установить не удастся, поскольку перед отправкой

// заголовка Cookie мы выводим в браузер строку 'Hello':

echo "Hello";

// Функция SetCookie возвратит FALSE:

if (SetCookie("Test","Value")) echo "<h3>Cookie успешно установлен!</h3>";

else echo "<h3>Cookie установить не удалось!</h3>";

// Выводит 'Cookie установить не удалось!'.

?>

Cookie установить не удалось, поскольку перед посылкой заголовка Cookie мы вывели в браузер строку "Hello".

Чтение значений Cookies

Получить доступ к Cookies и их значениям достаточно просто. Они хранятся в суперглобальных массивах и $\_COOKIE и $HTTP\_COOKIE\_VARS.

Доступ к значениям осуществляется по имени установленных Cookies, например:

echo $\_COOKIE['my\_cookie'];

// Выводит значения установленной Cookie 'My\_Cookie'

Пример установки Cookie и последующего его чтения:

<?php

// Устанавливаем Cookie 'test' со значением 'Hello' на один час:

setcookie("test","Hello",time()+3600);

// При следующем запросе скрипта выводит 'Hello':

echo @$\_COOKIE['test'];

?>

В рассмотренном примере при первом обращении к скрипту устанавливается Cookie "test" зо значением "hello". При повторном обращении к скрипту будет выведено значение Cookie "test", то есть строка "Hello".

При чтении значений Cookies обращайте внимание на проверку существования Cookies, например, используя оператор isset(). Либо путем подавления вывода ошибок опереатором @

А вот пример, как построить счетчик числа загрузок страницы с помощью Cookies:

<?php

// Проверяем, был ли уже установлен Cookie 'Mortal',

// Если да, то читаем его значение,

// И увеличиваем значение счетчика обращений к странице:

if (isset($\_COOKIE['Mortal'])) $cnt=$\_COOKIE['Mortal']+1;

else $cnt=0;

// Устанавливаем Cookie 'Mortal' зо значением счетчика,

// С временем "жизни" до 18/07/29,

// То есть на очень долгое время:

setcookie("Mortal",$cnt,0x6FFFFFFF);

// Выводит число посещений (загрузок) этой страницы:

echo "<p>Вы посещали эту страницу <b>".@$\_COOKIE['Mortal']."</b> раз</p>";

?>

Удаление Cookies

Иногда возникает необходимость удаления Cookies. Сделать это несложно, необходимо лишь вновь установить Cookie с идентичным именем и пустым параметром. Например:

<?php

// Удаляем Cookie 'Test':

SetCookie("Test","");

?>

Установка массива Cookies и его чтение

Мы может установить массив Cookies, используя квадратные скобки в именах Cookies [], а затем прочитать массив Cookies и значения этого массива:

<?php

// Устанавливаем массив Cookies:

setcookie("cookie[1]", "Первый");

setcookie("cookie[2]", "Второй");

setcookie("cookie[3]", "Третий");

// После перезагрузки страницы мы отобразим

// Состав массива Cookies 'cookie':

if (isset($\_COOKIE['cookie'])) {

foreach ($\_COOKIE['cookie'] as $name => $value) {

echo "$name : $value <br>";

}

}

?>

## Управление сессиями

Сессии, они же сеансы, это, по сути, просто обёртка над куками. Они также позволяют хранить данные, релевантные пользователю, но с некоторыми отличиями и ограничениями:

* Данные хранятся не произвольное время, а только до закрытия вкладки с веб-страницей.
* Чтобы сессии работали, в начале каждого сценария надо вызывать функцию session\_start().
* Запись и чтение информации при использовании сессий выглядит просто как работа со специальным массивом $\_SESSION.

Как устроены сессии

PHP генерирует уникальный идентификатор браузера.

Идентификатор сохраняется в специальную куку и передаётся с каждым запросом.

Все данные, которые записываются в сессию, PHP автоматически сохраняет в специальном файле на сервере.

Благодаря существованию сессий в PHP, мы можем сохранять любые данные так же просто, как присваивать их переменным. Но, в отличие от переменных, эти данные будут сохраняться для пользователя между запросами в пределах сеанса.

Перепишем сценарий для подсчета посещений, но теперь используем сессии:

<?php

session\_start();

$visit\_count = 1;

if (isset($\_SESSION["visit\_count"])) {

$visit\_count = $\_SESSION["visit\_count"] + 1;

}

$\_SESSION["visit\_count"] = $visit\_count;

print("Количество посещений: " . $visit\_count);

Использование сессии для контроля доступа

Сессии чаще всего используются для хранения информации о залогиненном пользователе. Принцип работы здесь очень простой: внутри сценария, ответственного за обработку формы входа, открывается новая сессия, куда записывается информация о вошедшем пользователе. Такой информацией может быть ассоциативный массив со всеми значениями из соответствующей записи из БД.

Затем добавим код, проверяющий существование сессии в сценарии, которые должны быть закрыты от анонимных пользователей. Если сессия пуста, значит данный пользователь не выполнял вход на сайт, и доступа к данной странице он не имеет.

В этом случае можно вернуть код ответа 403 и показать сообщение об ошибке, либо принудительно выполнить переадресацию на главную страницу.

Выход с сайта

Если на сайте есть вход, то должен быть и выход. Таким выходом будет специальный сценарий, который очистит сессию и переадресует на главную страницу.

Чтобы очистить сессию, достаточно очистить массив $\_SESSION:

$\_SESSION = []