РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ НА PHP&MYSQL

УРОК 2

Массивы. Функции. Обработка форм

ВВЕДЕНИЕ

Познакомившись с миром **web**-программирования вообще и с языком **PHP** в частности, мы приступим сегодня к изучению базовых возможностей языка, на которых базируются более сложные и мощные инструменты языка, благодаря которым язык **PHP** и заработал свою популярность.

1 МАССИВЫ В ЯЗЫКЕ PHP

4.1 Синтаксис и определение

Массивы – это структурированная совокупность ячеек памяти, хранящих элементы данных под общим именем.

Массивы в **PHP** немного отличаются от массивов в **C++**. В **PHP** массивы могут содержать в качестве элементов любой существующий тип, в том числе и другие массивы.

Каждый элемент массива строго идентифицируется по индексу, иначе называемому ключом, который записывается в квадратных скобках после имени массива:

Пример 1.1.1

```
$mas[7]; //Элемент массива $mas с ключом 7.
```

Таким образом, массив представляет собой набор пар «ключ-значение». И это очень отрадный факт, так как мы получаем отличный и удобный инструмент для хранения структурированной информации, в том числе в табличном виде.

Простейшим примером массива может служить обыкновенное слово, которое можно представить как массив символов.

Так же как и в **C**, в **PHP** нумерация элементов в массиве начинается с нуля.

4.2 Создание массивов

Массив может быть создан или, иначе говоря, инициализирован, несколькими способами:

1 Создание элемента и присваивание ему значения:

Пример 1.2.1

```
<?php
$my_array[0]="значение";
?>
```

Массив изначально может содержать несколько элементов, индексы (ключи) которых не обязательно должны идти по порядку.

Пример 1.2.2

```
<?php
$my_array[0]="Петя";
$my_array[4]="Вася";

// Будет создан массив, состоящий из двух элементов
?>
```

2 Создание элемента массива «на лету»:

Пример 1.2.3

```
<?php
$my_array[4]="Вася";
$my_array[]=66;
$my_array[]=651;

// Будет создан массив, состоящий из трёх элементов
// с ключами 4 (значение "Вася"), 5 (значение 66)
// и 6 (значение 651)
?>
```

Т.е. пустые квадратные скобки продолжают нумерацию индексов автоматически, инкрементируя их относительно последнего введенного индекса.

3 Создание с использованием конструкции array():

В качестве параметров она принимает любое количество разделенных запятыми пар `key => value` (ключ => значение).

```
array( [key =>] value, ...)
```

`key` может быть `integer` или `string`, `value` может быть любым значением.

Пример 1.2.4

```
<?php
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

Несмотря на то, что массивы, как и переменные, не требуют объявления перед их непосредственным использованием, хорошим тоном считается предварительное создание пустого массива для последующей работы с ним:

Пример 1.2.5

```
<?php
$my_array=array();
$my_array[4]="Вася";
$my_array[]=66;
$my_array[]=651;
```

Важным свойством массивов в PHP также является возможность использования в качестве индекса любой переменной:

Пример 1.2.6

```
<?php
$index=5;
$my_aaray[$index]=235;
echo $my_aaray[5]; //235
?>
```

4.3 Ассоциативные массивы и индексные массивы (списки).

В качестве ключей массивов в **PHP** могут применяться также и строковые данные. Такие массивы называются **ассоциативными**. В свою очередь массивы с исключительно числовыми ключами называются **индексными** или **списками**. Ассоциативные массивы существенно развивают идею о том, что массив – это набор пар «ключ-значение» и позволяют хранить информацию в более осмысленном виде. Например:

Пример 1.3.1

```
<?php
$menu[0]="белое";
$menu[1]="тёмное";
?>
```

```
<?php
$menu["вино"]="белое";
$menu["пиво"]="тёмное";
?>
```

Ключ ассоциативного массива должен быть обязательно заключён в кавычки, за исключением того, если в качестве ключа используется константа.

При инициализации ассоциативного массива конструкцией `array()` нельзя опускать описание ключей.

Ключ массива будет интерпретироваться как число, если является представлением `integer` и строковым, если заключён в кавычки.

В одном и том же массиве могут встречаться ключи числовые и строковые. Такие массивы называются смешанными.

4.4 Многомерные массивы.

Поскольку элемент массива в **PHP** может иметь любой тип данных, нам ничего не мешает определить массив, каждый элемент которого тоже является массивом. Это – двумерный массив. Мерность массива практически не ограничивается.

Для ссылки на один элемент сначала используются квадратные скобки для выборки первого измерения (строки в случае с двумерным массивом), а затем используется вторая пара квадратных скобок для выбора второго измерения (столбцы). Количество пар квадратных скобок равно мерности массива.

Создаются многомерные массивы в целом также как и одномерные.

Пример 1.4.1

```
<?php
$fruits=array(
"фрукты"=>array("a"=>"апельсин",
"b"=>"банан", "c"=>"яблоко"),
"числа"=>array(1,2,3,4,5,6),
"дырки"=>array("первая",5=>"вторая","третья")
);
```

```php
// Несколько примеров доступа к значениям
// предыдущего массива
echo $fruits["дырки"][5];    // напечатает "вторая"
echo $fruits["фрукты"]["a"]; // напечатает "апельсин"
unset($fruits["дырки"][0]);  // удалит "первая"

// Создаст новый многомерный массив
$juices["яблоко"]["зеленое"]="хорошее";
?>
```

4.5 Функция print_r()

При работе со сложными структурами массивов легко запутаться и очень часто необходимо в отладочных целях иметь возможность просмотреть все содержимое массива. На помощь придёт функция `print_r()`, которая выводит информацию о любой переменной (в том числе массиве) в читабельном виде.

Лучше всего использовать её в контексте переформатированного вывода **HTML**, т.е. элемента `<pre>`:

Пример 1.5.1

```php
<pre>
<?php
$a=array(
'a'=>'apple', 'b'=>'banana', 'c'=>array('x', 'y', 'z')
);
print_r($a);
?>
</pre>

Array
(
    [a] => apple
    [b] => banana
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
```

4.6 Перебор элементов массива

Большая часть задач в программировании при работе с массивами связана с необходимостью перебора их элементов. Технология перебора, в общем-то, очевидна – это имеющиеся в нашем распоряжении циклические конструкции `for`, `while`.

Пример 1.6.1

```
<?php
$countries=array("Украина", "Норвегия", "Польша");
$index=0;
$elements=count($countries);
while($index<$elements) {
echo ($index+1).".".$countries[$index]."<br />";
$index++;
}
?>
/*
1. Украина
2. Норвегия
3. Польша
*/
```

Здесь стоит остановиться разве что на новой для нас функции `count()`. Она вычисляет количество элементов в массиве. В нашем примере это число будет равно трём. Это значение присваивается также и переменной `$elements`.

Аналогично данная задача решается при помощи цикла `for`:

Пример 1.6.2

```
<?php
$countries=array("Украина", "Норвегия", "Польша");
for ($index=0; $index<count($countries); $index++) {
echo ($index+1).".".$countries[$index]."<br />";
}
?>
```

Данные примеры очень хорошо применимы для работы со списками, но вот для перебора ассоциативных (или смешанных) массивов они не годятся совсем.

К счастью, в **PHP** существует специальная конструкция цикла, предназначенная именно для этой цели. Это цикл `foreach`. сущест-

вует два вида записи:

```
foreach (имя_массива as $value)
{ блок_выражений; }

foreach (имя_массива as $key=>$value)
{ блок_выражений; }
```

Цикл `foreach` перебирает все элементы массива и переменной `$key` передаёт ключ элемента, а переменной `$value` – значение. В первом варианте будет доступно только значение текущего элемента массива, а вот втором еще и ключ.

Пример 1.6.3

```
<?php
$user=array(
"login"=>"Administrator",
"password"=>"ThU58!kksd",
"email"=>"admin@myhost.ua",
"status"=>"superAdministrator"
);
foreach ($user as $key=>$value) {
echo $key.": ".$value."<br />";}
?>
```

Варианты перебора можно комбинировать. Особенно это актуально для многомерных массивов.

Рассмотрим более сложный пример:

Пример 1.6.4

```
<?php
$users=array(
   0=>array(
      "login"=>"Administrator",
      "password"=>"ThU58!kksd",
      "email"=>"admin@myhost.ua",
      "status"=>"superAdministrator"
   ),
   1=>array(
      "login"=>"Administrator",
      "password"=>"ThU58!kksd",
      "email"=>"admin@myhost.ua",
      "status"=>"superAdministrator"
```

```
        ),
        2=>array(
            "login"=>"Administrator",
            "password"=>"ThU58!kksd",
            "email"=>"admin@myhost.ua",
            "status"=>"superAdministrator"
        )
    );
    echo "<table border='1'>";
    echo "<tr><th>Логин</th><th>Пароль</th>
    <th>e-mail</th><th>Статус</th></tr>";
    for ($index=0; $index<count($users); $index++) {
        echo "<tr>";
        foreach ($users[$index] as $v) {
            echo "<td>$v</td>";
        }
        echo "</tr>";
    }
    echo "</table>";?>
```

Этот код выведет в браузер информацию о пользователях в виде удобочитаемой таблицы.

На этом работа с массивами не заканчивается: в следующем уроке мы рассмотрим стандартные функции языка для работы с массивами.

2 ФУНКЦИИ

Функции – это ключ к эффективному программированию. Они делают код более структурированным и пригодным к повторному использованию и поддержке.

Общее понятие функции в **PHP** практически ничем не отличается от других **C**-подобных языков, т.е. **функция** – это независимый фрагмент кода, который может быть использован любое число раз, имеющий уникальное наименование и решающий определенную задачу.

Будем различать пользовательские функции, т.е. те, которые создаются пользователем и встроенные, т.е. те, которые описаны непосредственно в ядре языка и готовы к использованию в любом проекте. О встроенных функциях мы будем говорить много в последующих уроках, а сегодня сосредоточимся непосредственно на написании и использовании своих собственных функций - пользовательских.

2.1 Объявление функций.

Для использования функции необходимо сделать две вещи: объявить функцию, т.е. описать её в любой части программы, а затем вызвать к выполнению в том мечте программы, где это необходимо по условию задачи.

Функция объявляется при помощи ключевого слова `function` (как и в **JavaScript**), за которым следует список её параметров в круглых скобках, разделенных запятыми. Количество параметров не ограничено, причём их может не быть вовсе. Тело функции, т.е. те операторы, которые будут выполняться при вызове, записываются в фигурных скобках.

Например:

Пример 2.1.1

```
function cube($number) {
$result=$number*$number*$number;
echo $result;
}
```

В описанную функцию будет передаваться один числовой аргумент, который будет возводиться в куб, а результат будет выводиться в браузер.

Важно понимать разницу между терминами **аргумент** и **параметр**. **Аргумент** – это реальное значение, с которым будет оперировать функция, а **параметр** – это условное название (псевдоним), под которым аргумент будет фигурировать внутри функции.

2.2 Передача параметров по ссылке

По умолчанию аргументы копируют переданные значения и дальше работают с копией, что не оказывает влияния на оригинал. Этот способ передачи данных в функцию называется **передачей по значению**.

Если аргументу функции предшествует оператор `&`, переменная становится псевдонимом передаваемого значения и называется **ссылкой**, а передача параметров таким образом – **передачей параметров по ссылке**.

Пример 2.2.1

```
<?php
function stripCommas(&$text) {
$text=str_replace("", "", $text);
}
$myText="Это, текст, с, запятыми,";
stripCommas($myText);
echo $myText; //Выведет: Это текст с запятыми
?>
```

В данном примере применена встроенная функция `str_replace()`, которую мы рассмотрим в следующем уроке. Она занимается заменой участка строки на другую строку.

Если в данном примере в объявлении функции `stripCommas` убрать оператор `&`, то функция перестанет работать.

2.3 Возврат результата функциями

Функции, создаваемые пользователем, могут как возвращать какой-либо результат, так и не делать этого, просто выполняя какие-либо действия. Как правило, эти действия связаны с выводом каких-либо данных непосредственно в браузер.

Для возврата функцией значения используется оператор `return`, который может находиться в любом месте функции. Его задача – вернуть в основную программу значение выражения, записанного сразу за ним. Выполнение оператора `return` останавливает выполнение функции.

Пример 2.3.1

```
function cube($number) {
return $number*$number*$number; }
```

Как ни парадоксально, но функция может содержать несколько операторов `return`. Чаще всего это связано с применением конструкции `if` (или `switch`).

2.4 Вызов функции

После того, как функция определена, она все еще никак не влияет на программу. Теперь её нужно вызвать. Для этого необходимо в нужном месте программы записать имя функции, сопроводив его круглыми скобками со списком аргументов, которые, будучи переданы в функцию, отождествятся с параметрами:

Пример 2.4.1

```
$a=cube(6);
echo cube($a); //10077696
```

2.5 Рекурсия

Концепция рекурсии реализована и в PHP. Её реализация не отличается от других языков программирования, поэтому ограничимся примером применения рекурсии. В этом примере происходит проверка является ли аргумент функции целым числом.

Пример 2.5.1

```
<?php
function checkInteger($num) {
   if ($num>1) return checkInteger($num-1);
//Целое минус единица - целое
   elseif ($num<0) return checkInteger((-1)*$num-1);
// Отрицательные числа делаем положительными
   else {
      if (($num>0) && ($num<1)) return "Нет";
      else return "Да";
   }
}
?>
```

2.6 Области видимости переменных

Для того, чтобы избежать взаимного влияния переменных в различных функциях в PHP различают **области видимости** – свойство, которое определяет, какая таблица памяти используется для хранения переменных и какие из них доступны в данный момент. Каждая строка кода принадлежит к определенной области видимости.

С точки зрения использования переменных в функциях, их можно разделить на две группы – **локальные** и **глобальные**. Переменные, объявленные внутри функции (в том числе и параметры) называются локальными. Они не доступны и никогда не обрабатываются за пределами функции. Переменные, используемые за пределами объявления функции являются глобальными. В отличие от других языков программирования, глобальные переменные в PHP необязательно будут доступными вне глобальной области видимости.

Пример 2.6.1

```
<?php
$some_var="Глобальная переменная";
function test_global () {
echo $some_var." из функции";
}
test_global(); // Выведет: «из функции»
?>
```

Значение глобальной переменной может быть доступно внутри

функции, например, в том случае, если его передать в качестве аргумента. И наоборот, значение локальной переменной внутри функции может быть доступно для внешнего кода, только если его вернуть с помощью команды `return`.

Впрочем, существуют еще два способа доступа к глобальным переменным:

- С помощью оператора `global`
- С помощью предопределенного массива `$GLOBALS[]` (верхний регистр важен!)

Пример 2.6.2

```
<?php
$age=18;
function say_age() {
global $age; //Получаем доступ к глобальной переменной внутри функции
echo $age;
}
?>
```

Оператор `global` позволяет работать с переменной и после того, как управление передано из функции в основную программу. Любое изменение значения переменной сохраняется и после завершения работы функции. Это очень похоже на передачу параметра функции по ссылке, но отличается тем, что мы оперируем не с копией переданной по ссылке в функцию переменной, а с самой переменной непосредственно.

Объявление `global` решает все проблемы, однако в языке **PHP 5/6** более предпочтительным вариантом является использование массива `$GLOBALS[]`. Этот массив не нужно создавать, он генерируется интерпретатором автоматически и хранит все глобальные переменные текущего скрипта, имена которых сохранены в ключах элементов массива, а их значения – в значениях элементов.

Массив `$GLOBALS[]` является, так называемым, **суперглобальным массивом**. В **PHP** он не единственный. Совсем скоро мы столкнёмся и с другими суперглобальными массивами.

2.7 Статические переменные

Локальные переменные, как мы выяснили, доступны только внутри функции. Что происходит с ними после того, как функция отработала? Они уничтожаются. Их значение либо будет передано в основную программу при помощи `return`, либо будет просто потеряно. При следующем вызове функции, локальная переменная будет проинициализирована заново и получит новое значение. Рассмотрим пример:

Пример 2.7.1

```
<?php
function demo() {
$var=0;
echo $var."<br />";
$var++;
}
demo();
demo();
demo();
?>
```

Все три вызова функции `demo()` приведут к печати трёх нулей в столбик.

Для хранения значения локальной переменной между вызовами функции необходимо объявить её как **статическую**, используя ключевое слово `static` перед именем переменной.

Эту же самую задачу можно, безусловно, решить и при помощи специальной глобальной переменной, но использование `static` представляется более красивым.

С использованием `static` рассмотренный ранее пример теперь будет выводить цифры 0, 1, 2.

Пример 2.7.2

```
<?php
function demo() {
static $var=0; //!!!
echo $var."<br />";
$var++;
}
```

```
demo();
demo();
demo();
?>
```

2.8 Параметры по умолчанию

Часто бывают такие случаи, что у некоторой разрабатываемой функции должно быть довольно много параметров, причем некоторые из них будут задаваться совершенно единообразно. Например, мы пишем функцию для сортировки массива. Тогда, кроме очевидного параметра — массива — хотелось бы также задавать и второй параметр, который бы указывал: сортировать ли в убывающем или в возрастающем порядке. При этом, скажем, мы знаем, что чаще всего придется сортировать в порядке убывания. В этом случае мы можем оформить нашу функцию так:

Пример 2.8.1

```
function MySort(&$Arr, $NeedLoOrder=1)
{ ... сортируем в зависимости от $NeedLoOrder... }
```

Теперь, имея такую функцию, можно написать в программе:

Пример 2.8.2

```
MySort($my_array,0); // сортирует в порядке возрастания
MySort($my_array); // второй аргумент по умолчанию!
```

То есть, мы можем уже вообще опустить второй параметр у нашей функции, что будет выглядеть так, как будто мы его задали равным 1. Как видно, значение по умолчанию для какого-то аргумента указывается справа от него через знак равенства. Заметьте, что значения аргументов по умолчанию должны определяться справа налево, причем недопустимо, чтобы после любого из таких аргументов шел обычный "неумолчальный" аргумент. Вот, например, неверное описание:

Пример 2.8.3

```
function MySort($NeedLoOrder=1, &$Arr) // Ошибка!
{ ... сортируем в зависимости от $NeedLoOrder... }
MySort(,$my_array); // Ошибка!
```

2.9 Переменное число параметров

Как мы уже знаем, функция может иметь несколько параметров, заданных по умолчанию. Они перечисляются справа налево, и их всегда фиксированное количество. Однако иногда такая схема нас устроить не может.

Для примера создадим функцию, которая выводит «лесенкой» (с отступом) любое количество строк, передаваемых в качестве параметров:

Пример 2.9.1

```
<?php
function myecho () {
   for($i=0; $i<func_num_args(); $i++) {
      for($j=0; $j<$i; $j++)
      echo " "; // выводим отступ
      echo func_get_arg($i)."<br />\n"; //выводим элемент
      }
   }
myecho ("Меркурий", "Венера", "Земля", "Марс");
?>
```

Обратите внимание на то, что при описании `myecho()` в качестве списка параметров указаны пустые скобки, словно функция не получает ни одного параметра. На самом деле в **PHP** при вызове функции можно указывать параметров больше, чем задано в списке аргументов.

Если фактическое число параметров меньше, чем указано в описании, **PHP** выдаст сообщение об ошибке.

Для того чтобы иметь доступ к параметрам, существуют три встроенные в **PHP** функции:

- `func_num_args()` – возвращает общее число аргументов, переданных функции при вызове.
- `func_get_arg($num)` – возвращает значение аргумента с номером `$num` (целочисленный), заданного при вызове функции. Нумерация от нуля.
- `func_get_args()` – возвращает массив-список всех аргументов, указанных при вызове функции. Применение этой функции оказывается практически всегда удобнее, чем первых двух.

Пример, переписанный с применением последней функции:

Пример 2.9.2

```
<?php
function myecho() {
   foreach (func_get_args() as $v) {
      for($j=0; $j<@$i; $j++) echo " ";
      echo "$v<br />\n";
      @$i++;
      }
   }
myecho ("Меркурий", "Венера", "Земля", "Марс");
?>
```

Оператор @ применяется для отключения вывода сообщений об ошибках, которые здесь могут быть в связи с неопределенностью $i при первом проходе цикла.

2.10 Вложенные функции

Функции могут объявляться внутри других функций. Но их область видимости, как у переменных, не ограничивается своими "родителями". **PHP** делает их доступными для всей остальной части программы, но только с того момента, когда "функция-родитель" была из нее вызвана.

Итак, "вложенные" функции выглядят следующим образом:

Пример 2.10.1

```
<?php
function parent($a) {
   echo $a;
   function child($b) {
      echo $b+1;
      return $b*$b;
   }
   return $a*$a*child($a);
   // фактически возвращает $a*$a*($a+1)*($a+1)
}
// Вызываем функции
parent(10);
child(30);
?>
```

Если вызовы функций поменять местами, то это вызовет ошибку, поскольку до вызова функции `parent` функция `child` не существует.

Если вызвать `parent()` дважды, то это тоже вызовет ошибку, так как функция `child()` уже определена.

2.11 Динамический вызов функции

В момент написания сценария разработчик может не знать имя вызываемой функции. Решение об этом принимается на основании данных, полученных при работе сценария. Необходимо иметь переменную с именем функции для последующего вызова.

Пример 2.11.1

```
<?php
function write($text) {
   echo $text."<br />";
}
function write_bold() {
   echo "<b>$text</b><br />";
}
$myFunc="write";
$myFunc("Привет");
$myFunc="write_bold";
$myFunc("Пока");
?>
```

3 СОЗДАНИЕ БИБЛИОТЕК ФУНКЦИЙ И РАБОТА С НИМИ

Язык **PHP**, как и все основные языки программирования, обладает замечательной способностью подключать внешние файлы. например, объявления часто употребляемых пользовательских функций или список конфигурационных переменных и констант, вынесенных в отдельный файл для использования в нескольких сценариях сразу. Это позволит управлять этими конфигурационными данными централизованно.

Включение одного или нескольких файлов в **PHP** осуществляется с помощью функций:

- `include "путь/к/файлу";` – подключает файл, найденный по указанному пути. Если в файле будут обнаружены ошибки, то **PHP** сообщит о них и продолжит выполнять остальной код.
- `include_once "путь/к/файлу";` – выполняет те же действия, что и include, но гарантирует однократное подключение файла. Это важно на случай сложных подключений, когда некоторый файл подключается к файлу, который, в свою очередь, тоже является подключаемым и т.д.
- `require "путь/к/файлу";` – также как и `include` подключает найденный по указанному пути файл, но в случае обнаружения ошибки в файле (или отсутствия самого файла) работа сценария будет остановлена.
- `require_once "путь/к/файлу";` – работает как `require`, но следит за однократным подключением файла.

Следует отметить, что подключаться могут совершенно любые текстовые файлы вне зависимости от их расширений. Их содержание будет попросту вставляться в то место кода основного сценария, где они были вызваны. А далее будет проходить их обработка в соответствии с общими правилами обработки текста интерпретатором **PHP**.

Довольно часто вы можете встретить web-приложения на PHP, содержащие файлы с расширением `.inc`. На самом деле в них содержится код PHP, но предназначены эти файлы не для

прямого выполнения интерпретатором, поскольку он просто их не получит, так они не будут распознаны Apache-сервером, а для включения их в код основного сценария. Но здесь нужно иметь в виду следующую опасность: если настройки вашего сервера позволят пользователю попытаться запустить такой файл напрямую (без подключения к основному файлу сценария), то сервер попросту предложит браузеру пользователя скачать этот файл, а это является недопустимым, особенно если там содержатся какие-то конфигурационные данные – пароли, ключи т т.д.

Пример библиотеки (файл `math.lib.php`):

Пример 3.1.1

```php
<?php
/*
Библиотека функций для математических вычислений
@author Пилипенко Андрей
*/

//Возведение в квадрат
function math_sq($num) {
return $num*$num;
}

//возведение в куб
function math_cube($num) {
return $num*$num*$num;
}
?>
```

Теперь подключим эту библиотеку:

Пример 3.1.2

```php
<?php
require_once "math.lib.php";
echo math_cube(123);
?>
```

4 РАБОТА С ФОРМАМИ. СПОСОБЫ СВЯЗЫВАНИЯ XHTML-ФОРМЫ И PHP-СКРИПТА

XHTML-формы для любого языка **web**-программирования это не просто графический интерфейс (**GUI**), но и самый главный инструмент, обеспечивающий интерактивность. Ни одно современное web-приложение совершенно не в состоянии обойтись без форм, т.е. общения с пользователем. Не удивительно, что и появились **web**-формы уже достаточно давно, а именно были описаны уже в спецификации **HTML 2.0**, увидевшей свет в 1995 году.

Формы представляют собой web-страницы (или их части) с наличествующими на них элементами управления: кнопками, переключателями, полями для ввода и т.д.

Рядом пример формы, реализующей диалог по авторизации пользователя на сайте.

Формы создаются при помощи специальных **XHTML**-тегов. Вот код формы, приведенной на рисунке:

Пример 4.1.1

```
<form action="index.php" method="post">
 <div><img src="key.jpg" alt="" /></div>
 <fieldset class="input">
     <p><label for="username">Логин</label><br />
       <input name="username" type="text" size="18" />
     </p>
     <p><label for="passwd">Пароль</label><br />
       <input name="passwd" type="password" size="18" />
     </p>
     <p><label for="remember">Запомнить меня</label>
<input name="remember" type="checkbox" value="yes" />
     </p>
     <input type="submit" name="Submit" value="Вход" />
```

```
    </fieldset>
    <ul>…</ul>
    <input type="hidden" name="option" value="com_user" />
    <input type="hidden" name="task" value="login" />
    <input type="hidden" name="return" value="aW5kgucGhw" />
    <input type="hidden" name="46df91d92541eca" value="1" />
  </form>
```

Часть **XHTML**-кода не приведена (содержимое элемента `<ul>`), а элементы, размечающие непосредственно элементы управления и саму форму выделены полужирным.

Для работоспособности, форма обязана иметь два важных атрибута: `action` и `method`. Атрибут `action` указывает на скрипт, который должен будет заниматься на сервере обработкой данных, введенных пользователем в форму, а `method` определяет метод протокола **HTTP**, применяемый для отправки этих данных.

Если атрибут method опущен, то по-умолчанию данные будут передаваться методом `GET`.

Для эффективной работы с формами необходимо изучить работу протокола **HTTP** подробнее.

4.1 Принцип работы HTTP

Взаимодействие между клиентом и сервером в Интернет происходит в основном на основе протокола **HTTP** (**HyperText Transfer Protocol** – протокол передачи гипертекста). **HTTP** является **текстовым** протоколом, т.е. все запросы представляются в виде последовательности символов в коде **ASCII**.

В общем виде запрос имеет следующую структуру:

```
Метод URI HTTP-версия
Дополнительная информация
```

- Здесь `метод` – это либо `GET` либо `POST`. Мы рассмотрим методы подробнее далее.
- **URI (Universal Resource Identifier)** – универсальный идентификатор ресурса – имеет следующий вид:

```
[путь][файл][?параметр=значение[&параметр=значение[…]]]
```

Необязательные части заключены в квадратные скобки. Путь до файла задается относительно корневого каталога сервиса www, что

задается при настройке сервера.

Параметры, перечисляемые после `?`, – это данные, которые передаются на сервер при помощи метода `GET`. Имя и значение параметра разделяется `=`, а пара параметр-значение знаком `&`.

- `HTTP-версия` – определяет версию протокола, по которой будет производиться обмен данными.
- `Дополнительная информация` может включать **IP**-адрес клиента, имя программы-клиента, предпочитаемый язык запрашиваемого документа и т.д.

Рассмотрим пример:

```
    GET /news HTTP/1.1
    Accept-Language: en-us
    Connection: Keep-Alive
    Host: 82.178.22.01
    User_Agent:  Mozilla/5.0  (X11;  U;  Linux  i686;  ru;
rv:1.9b5) Gecko/2008050509 Firefox/3.0b5
```

По-моему, все достаточно прозрачно и без комментариев ☺.

Если запрашиваемая страница (файл с именем `index` в папке `/news`) доступна на сервере, то ответ может быть например таким:

```
    HTTP/1.1 200 OK
    Server: Apache/2.0.52 (Gentoo/Linux)
    Connection: Keep-Alive
    Date: Wed, 11 Feb 2009 11:20:59 GMT
    Content-Type: text-html
    Accept-Ranges: bytes;
    Content-Length: 2964
    Last-Modified: Wed, 11 Feb 2009 11:20:59 GMT
    Cookie: uid=456877

    <!DOCTYPE html PUBLIC …
    …
```

Как видим, ответ состоит из 2 частей, разделенных пустой строкой. Информация, расположенная до пустой строки – это **заголовки HTTP**, а то, что ниже – тело, т.е. та информация, ради которой проводился запрос. В примере фигурирует **XHTML**-код, хотя, на самом деле, тело может содержать все что угодно, даже бинарный файл (архив, изображение и т.д.)

Разобравшись с основами работы протокола **HTTP**, сосредоточимся на методах протокола. Это `GET` и `POST`, как мы выяснили.

На самом деле метолов у протокола HTTP больше, но для понимания механизма обмена данными web-формы и скрипта они не важны.

4.2 Метод GET

Метод `GET` отправляет все данные, присоединяя их к **URL** скрипта, ответственного за обработку формы, способом, который мы рассмотрели в предыдущем разделе, т.е. дописывая данные после `?`. как результат все данные формы в таком случае видны в адресной строке браузера.

Например:

Пример 4.2.1

```
<form action="./register.php" method="get">
Имя: <input type="text" name="first_name" /><br />
Фамилия: <input type="text" name="last_name" /><br />
<input type="submit" value="Отправить" />
</form>
```

Если в поля ввода с именами `first_name` и `last_name` пользователь введет значения, например `"James"` и `"Bond"` соответственно, то по нажатию на кнопке Отправить браузер будет перенаправлен на следующий **URL**:

`http://my.ua/register.php?first_name=James&last_name=Bond`

При этом реальная переадресация произойдет на `http://my.ua/register.php`.

Общее ограничение на передачу данных с помощью `GET` – 255 символов, а также передаваемые символы подлежат **URL**-кодированию. Например обыкновенный пробел при передаче будет закодирован в виде `%20`.

Применяется, в основном, для передачи небольших по размеру данных, а также при самостоятельном конструировании ссылки в теле **XHTML**-страницы.

Например:

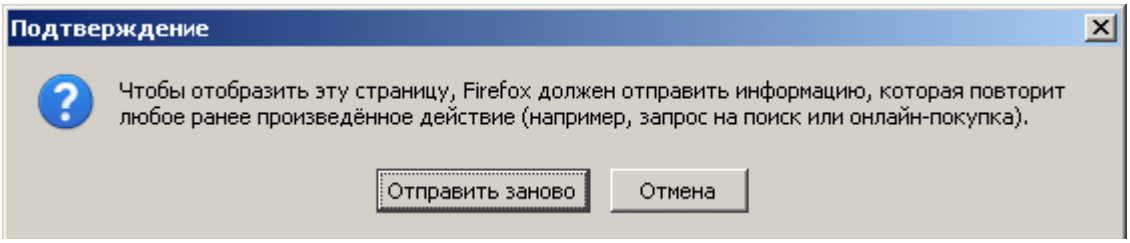
Пример 4.2.2

```
<?php
$str="pulp";
echo "<a href='./script.php?word=$str'>Ссылка</a>";
?>
```

4.3 Метод POST

Этот метод передает все данные в теле запроса. Он более распространен для передачи данных форм, так как не имеет тех ограничений, которые есть у GET.

К недостаткам метода можно отнести то, что браузеры не могут автоматически переотправлять данные с помощью метода POST, если пользователь нажмёт в браузере кнопку "Обновить" или "Назад". Это приведет к сообщению вроде:



4.4 Общие принципы обработки данных из формы.

После того, как данные из формы попали тем или иным способом на сервер, **web**-сервер, в соответствии с требованиями **CGI** размещает эти заголовки в **переменных среды (окружения)**. Когда начинается выполнение сценария, **PHP** преобразует эти переменные в переменные PHP.

Здесь есть два варианта преобразования:

- Преобразование в переменные глобальной области видимости скрипта, имя которых полностью совпадает со значением атрибута `name` соответствующего элемента управления в **web**-форме.

  Ранее мы встретились с суперглобальным массивом `$GLOBALS[]`.

- Преобразование в элементы суперглобальных ассоциативных массивов `$_GET[]`, `$_POST[]`, `$_REQUEST[]`. Ключи этих элементов соответствуют значениям атрибутов `name`, а значение элементов –

значениям переменных окружения.

За то, какой именно вариант сработает, отвечает директива `register_globals` конфигурационного файла `php.ini`. Она может принимать значение `on` и `off`. В случае, если директива включена (`on`), сработает первый вариант, иначе – второй. В версиях **PHP**, начиная с **4.2.0**, эта директива по умолчанию установлена в `off`. Это было очень спорным решением, но общество разработчиков **PHP** решило изменить значение по умолчанию этой директивы на `off`. В противном случае при написании кода разработчики не могли бы с уверенностью сказать, откуда пришла та или иная переменная и насколько она достоверна. До такого нововведения переменные, определяемые разработчиком внутри скрипта, и передаваемые пользователем внешние данные могли перемешиваться. Приведем простой пример злоупотребления директивой `register_globals`:

Пример 4.4.1

```
<?php
/* устанавливаем переменную $authorized = true
только для пользователей, прошедших авторизацию */
if (authenticated_user()) {
   $authorized = true;
}

/* Поскольку в случае неудачи при проверке авторизации
переменная $authorized не установлена, она может быть
установлена автоматически, благодаря register_globals,
например, при GET запросе GET auth.php?authorized=1.
Таким образом, пройти эту проверку можно без авторизации
*/
if ($authorized) {
   include "/highly/sensitive/data.php";
}
?>
```

В случае `register_globals = on` логика работы скрипта может быть нарушена. В случае, если установленное значение `off`, переменная `$authorized` не может быть установлена из внешних данных запроса, и скрипт будет работать корректно.

4.5 Обработка GET-данных

Для доступа к данным, отправленных методом GET, служит массив $_GET[]. Например, если в отправленной форме присутствовало поле вида: <input type="text" name="surname">, то в скрипте эти данные будут доступны как $_GET["surname"].

Например, у нас есть файлом с **web**-формой form.html. Вот фрагмент этого файла:

Пример 4.5.1

```
…
<form action="hello.php">
Введите имя:
<input type="text" name="name" value="Неизвестный" />
<br />
Введите возраст:
<input type="text" name="age" value="Неопределенный" />
<br />
<input type="submit" value="Нажми!" >
</form>
…
```

Файл-обработчик hello.php может выглядеть так:

Пример 4.5.2

```
<?
echo "Привет, ".$_GET["name"]."!<br />
Я знаю, Вам ".$_GET["age"]." лет!";
?>
```

4.6 Обработка POST-данных

Обработка данных, отправленных методом POST, осуществляется аналогичным образом, только доступ к значениям осуществляется посредством суперглобального массива $_POST[].

Зная о том, что методом POST мы можем отправлять данные большей длины, дополним наш файл form.html текстовым полем <textarea>.

Пример 4.6.1

```
…
<form action="hello.php" method="post">
Введите имя:
<input type="text" name="name" value="Неизвестный" />
<br />
Введите возраст:
<input type="text" name="age" value="Неопределенный" />
<br />
Ваше хобби:
<textarea cols="20" rows="3" name="hobby">
</textarea>
<input type="submit" value="Нажми!" >
</form>
…
```

И доработаем файл-обработчик:

Пример 4.6.2

```
<?
echo "Привет, ".$_POST["name"]."!<br />
Я знаю, Вам ".$_POST["age"]." лет!<br />
Ваше хобби - ".$_POST["hobby"];
?>
```

В PHP можно не различать GET- и POST-переменные. Если разработчику удобнее не разделять данные по методам, то можно воспользоваться суперглобальным массивом `$_REQUEST[]`, который хранит в себе то же, что и в `$_GET[]` и `$_POST[]`.

Кроме уже изученных нами суперглобальных массивов, очень большой практический интерес представляет также суперглобальный массив `$_SERVER[]`. Предлагаем Вам самостоятельно, используя методы переборки значений массива, разобраться с тем, какие переменные в нём хранятся.

4.7 Форма и её обработчик в одном сценарии

Для ввода данных в форму и их обработки совсем не обязательно использовать два разных файла – `.html` и `.php`. Эти операции можно совместить в одном сценарии. Основной принцип работы в данном случае – проверка отправки формы, а далее, либо вывода формы, либо её обработки.

Определить отправку формы можно, проверив элемент одного из массивов `$_GET[]`, `$_POST[]`, `$_REQUEST[]`, который гарантированно должен появиться после её отправки. Как правило, таким элементом часто выступает сама кнопка отправки (`submit`). Если элемент соответствующего массива с именем кнопки пуст, то форма не отправлялась и наоборот.

Можно использовать следующий шаблон:

Пример 4.7.1

```
<?php
if (!$_GET["submit_button"]) {
//код, выводящий XHTML-код формы. Как правило, это echo.
}
else {
//код, обрабатывающий форму.
}
?>
```

или такой:

Пример 4.7.2

```
<?php
if ($_GET["submit_button"]) {
//код, обрабатывающий форму.
}
else {
?>
<form action… </form> <!-- XHTML-код, выводящий форму -->
<?php
}
?>
```

4.8 Обработка различных элементов управления

Существует достаточно большое количество элементов **XHTML**-форм. Принципы трансляции данных форм, в общем-то, одинаковы, но есть некоторые нюансы.

`<input type="text" />` Однострочное текстовое поле

`<input type="password" />`	Однострочное текстовое поле с замаскированным вводом
`<input type="hidden" />`	Скрытое текстовое поле
`<input type="checkbox" />`	Флажок для многовариантного выбора
`<input type="radio" />`	Радиокнопка для одновариантного выбора
`<input type="reset" />`	Кнопка для очистки всех полей формы
`<input type="submit" />`	Кнопка отправки формы
`<select>` `<option /> </select>`	Раскрывающийся список значений
`<textarea></textarea>`	Многострочное текстовое поле

Существует также элемент `<input type="file">` для передачи файлов на сервер, но эта тема будет рассмотрена отдельно в одном из следующих уроков.

1 Однострочные текстовые поля

Все однострочные текстовые поля `hidden`, `text`, `password` обрабатываются одинаково. Их общий синтаксис записи таков:

```
<input type="тип" name="имя" value="исходное_значение" />
```

После отправки на сервер, значение поля, введенное пользователем (или значение по-умолчанию), будет доступно как `$_GET["имя"]="значение";` или `$_POST["имя"]="значение";` или `$_REQUEST["имя"]="значение";`

Пример работы:

Пример 4.8.1

```
<?php
if (!$_POST["submitted"]) {
   echo "<form action=\"\" method=\"post\">
   логин: <input type=\"text\" name=\"login\" /><br />
   пароль: <input type=\"password\" name=\"password\" />
   <br /><input type=\"hidden\" name=\"ip\"
   value=\"".$_SERVER["REMOTE_ADDR"]."\"/>
   <input type=\"submit\" value=\"Отправить\"
   name=\"submitted\" /></form>";
}
else {
```

```
    echo "Ваши данные:<br />";
    echo "Логин: ".$_POST["login"]."<br />Пароль:
    ".$_POST["password"]."<br />Ваш IP: ".$_POST["ip"];
}
?>
```

Обратите внимание на $_SERVER["REMOTE_ADDR"]. Данное значение суперглобального массива помещается в скрытое поле формы и содержит **IP**-адрес клиента, заполнившего форму.

В данном примере применена технология экранирования служебных символов, которым, в данном случае, является двойная кавычка ("). Для этого необходимо перед экранируемым символом поставить обратный слеш (\).

2 Флажки

Флажки используются в тех ситуациях, когда пользователю необходимо выбрать один или несколько вариантов. Синтаксис элемента:

```
<input type="checkbox" name="имя" value="значение" />
```

После отправки на сервер, значение поля будет доступно в том случае, если «галочка» была выбрана пользователем как $_GET["имя"]="значение"; или $_POST["имя"]="значение"; или $_REQUEST["имя"]="значение";

Пример работы:

Пример 4.8.2

```
<?php
if (!$_POST["submitted"]) {
    echo "<form action=\"\" method=\"post\">
    <h3>Мои любимые фильмы:</h3>
    <input type=\"checkbox\"
    name=\"shank\" value=\"Побег из Шоушенка\" />
    Побег из Шоушенка<br />
    <input type=\"checkbox\"
    name=\"fc\" value=\"Бойцовский Клуб\" />
    Бойцовский Клуб<br />
    <input type=\"submit\" value=\"Отправить\"
    name=\"submitted\" /></form>";
}
else {
    echo "Ваши любимые фильмы:<br />";
```

```
        if ($_POST["shank"]) {
            echo $_POST["shank"]."<br />";
        }
        if ($_POST["shank"]) {
            echo $_POST["shank"]."<br />";
        }
    }
    ?>
```

Здесь нужно обратить внимание на то, что, если флажок не был отмечен, то соответствующая переменная создана не будет, поэтому мы проверяем их наличие перед выводом.

Очень распространенным способом использования флажков есть трансляция их не в разные переменные, а в массив, элементы которого содержат значения выбранных вариантов. Для этого используется способ создания массива «на лету». В этом случае необходимо всем примененным в форме флажкам дать одинаковое имя, но сопроводить его квадратными скобками. Например, можно доработать предыдущий пример:

Пример 4.8.3

```
        …
        <input type=\"checkbox\"
        name=\"films[]\" value=\"Побег из Шоушенка\" />
        Побег из Шоушенка<br />
        <input type=\"checkbox\"
        name=\"films[]\" value=\"Бойцовский Клуб\" />
        Бойцовский Клуб<br />
        …
```

3 Радиокнопки

В отличие от текстовых полей, которые имеют уникальные имена, радиокнопкам, наоборот, имена нужно давать одинаковые на всю группу кнопок при разных значениях. В таком случае на сервере будет создана переменная с именем группы кнопок, а значение в неё будет помещено то, которое было отмечено пользователем. Применяются радиокнопки для реализации одновариантного выбора. Выбранный вариант по умолчанию должен сопровождаться атрибутом `checked`.

Синтаксис следующий:

```
<input type="radio" name="имя_группы" value="знач1"
```

```
checked />
<input type="radio" name="имя_группы" value="знач2" />
…
```

Пример:

Пример 4.8.4

```
<?php
if (!$_POST["submitted"]) {
   echo "<form action=\"\" method=\"post\">
   <h3>Мой самый любимый фильм:</h3>
   <input type=\"radio\"
   name=\"film\" value=\"Побег из Шоушенка\" />
   Побег из Шоушенка<br />
   <input type=\"radio\"
   name=\"film\" value=\"Бойцовский Клуб\" />
   Бойцовский Клуб<br />
   <input type=\"submit\" value=\"Отправить\"
   name=\"submitted\" /></form>";
}
else {
   echo "Ваш самый любимый фильм:<br />";
   echo $_POST["film"];
}
?>
```

4 Раскрывающиеся списки

Раскрывающийся (выпадающий) список является, по-сути, экономным, с точки зрения места на странице, вариантом флажков или радиокнопок, т.е. инструмента многовариантного выбора.

Синтаксис списка следующий:

```
<select name="имя_списка">
<option value="значение1">Текст_значения1
<option value="значение2">Текст_значения2
…
</select>
```

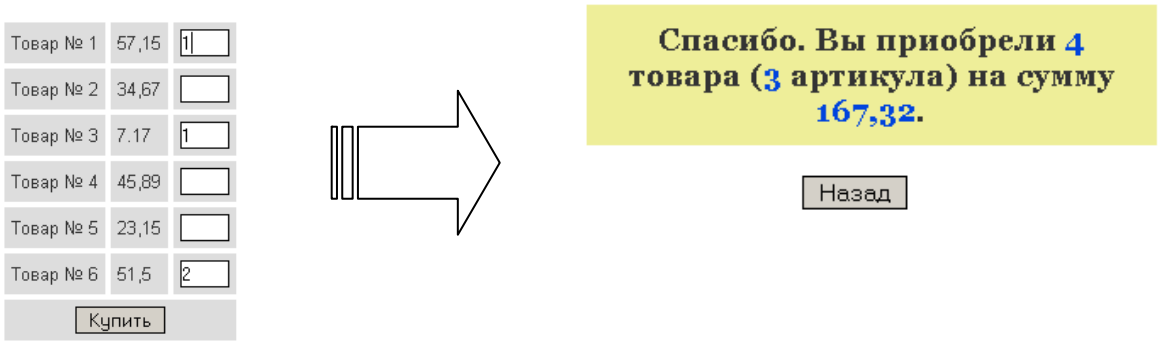
Пример обработки:

Пример 4.8.5

```
<?php
if (!$_POST["submitted"]) {
   echo "<form action=\"\" method=\"post\">
   <h3>Выберите форму оплаты:</h3>
   <select name="payway">
   <option value="Webmoney" />
   <option value="безналичный расчет" />
   </select>
   <input type=\"submit\" value=\"Отправить\"
   name=\"submitted\" /></form>";
}
else {
   echo "Вы выбрали ".$_POST["film"];
}
?>
```

ДОМАШНЕЕ ЗАДАНИЕ.

Разработать механизм оформления заказа при покупке в интернет-магазине, который будет представлять собой форму в виде таблицы, где будут представлены наименования товаров, их цены и поля для ввода, чтобы покупатель мог проставить желаемое количество. По нажатии кнопки «Купить», должен производиться подсчет суммы покупки и вывод её на экран:



Наименования товаров, их цены и их количество можно придумать самостоятельно. Вывод формы и её обработчика должен осуществляться в одном файле. Предусмотреть кнопку «Назад».

ИТОГИ.

В этом уроке мы выяснили принципы работы с массивами в **PHP** и отличия в этой работе от других языков программирования, научились писать функции, как обыкновенные, так и с переменным числом параметров или параметрами по умолчанию, а также подключать другие файлы.

А также мы разобрали важнейшую сторону работы **PHP**-разработчика, без которой не обходится ни один сценарий, – обработку **web**-форм.

Кроме того, узнали, что такое суперглобальные массивы **PHP**.

Далее мы приступим к знакомству с основными функциями ядра языка.

До новой встречи!