Тема 3.8

«Функции, определяемые пользователем»

Вступление

Несмотря на то, что в PHP существует огромное количество «готовых» функций, часто приходится писать свои собственные, т.к. это удобно и верно с точки зрения грамотного программирования.

Несколько фактов о функциях в PHP:

1) Тип возвращаемых данных НЕ указывается.
2) Типы аргументов НЕ указываются.
3) Функция может легко возвратить любое количество значений в виде сколь угодно сложного массива.
4) В функция в PHP можно передать количество параметров равное или превышающее количество объявленных параметров.
5) В PHP сразу пишется реализация функции (без отдельного объявления, как в С-образных языках).

Объявление функции

Синтаксис, используемый для объявления (определения) функций, таков:

```
function имя_функции ([$аргумент1,] [$аргумент2,] ...)
 {
  опреаторы_составляющие_тело_функции
 }
```

Пример:

```
function sum ($a, $b)
{
 echo $a+$b;
}
```

Пример вызова функции:

```
sum(5,8);
```

или

```
sum($x,$y);
```

Функции, зависящие от условий

Внутри функции можно использовать любой корректный PHP-код, в том числе даже объявлять другие функции.

В PHP 3 функции должны были быть определены прежде, чем они будут использованы. Начиная с PHP 4 такого ограничения нет, исключая тот случай, когда функции определяются условно, как это показано в двух следующих примерах.

В случае, когда функция определяется в зависимости от какого-либо условия, обработка описания функции должна предшествовать её вызову.

Пример 1 (определение функции, зависящей от условия):

```
if ($a==10)
{
  function func1() { … }
}
```

Функция func1() не может быть вызвана, пока выполнение кода не «зайдёт» в ветку, срабатывающую по условию $a==10.

Функции, зависящие от условий

Пример 2 (функция, определяемая внутри функции):

```
function func1 ()
{
 function func2() { … }
}
```

Функция func2() не может быть вызвана до первого вызова функции func1().

Внимание! Повторный вызов функции func1() приведёт к ошибке с сообщением о невозможности повторного объявления функции func2(). В то же время функцию func2() после первого вызова функции func1() можно вызывать неограниченное количество раз.

Функции, зависящие от условий

Применение функций, зависящих от условий, удобно в том случае, когда нам нужно использовать «одну и ту же функцию, работающую по-разному».

Например, мы хотим создать функцию, выполняющую запрос к СУБД, но знаем, что наш скрипт может работать с разными СУБД:

Пример:

```
if ($subd==“MySQL”)
 {
  function dbms_query($query) { … }
 }
  else
{
 function dbms_query($query) { … }
}
```

Функции, зависящие от условий

PHP не поддерживает перегрузку функции (но это ограничение можно обойти с использованием функций, зависящих от условий), также отсутствует возможность переопределить или удалить объявленную ранее функцию.

Имена функций регистронезависимы, тем не менее, более предпочтительно вызывать функции так, как они были объявлены.

Аргументы функций

Функция может принимать информацию в виде списка аргументов, который является списком разделённых запятыми переменных и/или констант.

PHP поддерживает передачу аргументов по значению (по умолчанию), передачу аргументов по ссылке, и значения по умолчанию.

Списки аргументов переменной длины поддерживаются, начиная с PHP 4.

Пример (передача массива в функцию):

```
function takes_array($input)
 {
  echo "$input[0] + $input[1] = ", $input[0]+$input[1];
 }
```

Передача аргументов по ссылке и значению

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение переменной-аргумента внутри функции, то вне функции значение переменной-аргумента останется прежним).

Пример:

```
$a=10;
function test($a)
{
 $a=20;
}
test($a);
echo $a; // 10
```

Примечание: имя аргумента функции и имя передаваемой в качестве аргумента переменной, естественно, может не совпадать во всех случаях использования функций.

Передача аргументов по ссылке и значению

Если вы хотите разрешить функции модифицировать свои переменные-аргументы, вы должны передавать их по ссылке (перед именем аргумента указывается амперсанд).

Пример:

```
$a=10;
function test(&$a)
{
 $a=20;
}
test($a);
echo $a; // 20
```

Внимание! В старых версиях PHP допускалась передача по ссылке аргумента, объявленного как аргумент с передачей по значению. В новых версиях PHP это запрещено.

Значения аргументов по умолчанию

Функция может определять значения по умолчанию в стиле C++ для своих аргументов.

Пример:

```
function func1($a, $b=20) { … }
```

Если такую функцию вызвать с двумя аргументами, значение $b будет равно значению второго переданного аргумента. Если же функцию вызвать с одним аргументом, значение $b будет равно 20.

Внимание! Аргументы со значениями по умолчанию должны идти В КОНЦЕ списка аргументов, т.к. PHP присваивает значения аргументам в том порядке, в каком они переданы при вызове функции. Т.о. выполнение такого кода приведёт к ОШИБКЕ (и сообщению о том, что в функцию передано недостаточное количество параметров):

```
function func1($a=20, $b) { … }
func1(50); // аргументу $b «не хватило» значения
```

Значения аргументов по умолчанию

PHP также позволяет использовать массивы и специальный тип NULL в качестве значений по умолчанию.

Пример:

```
function test($a = array(1,2,3), $b = NULL)    { … }
```

Значение по умолчанию должно быть константным выражением, а не (к примеру) переменной или вызовом функции/метода.

Списки аргументов переменной длины

PHP, начиная с версии 4, поддерживает списки аргументов переменной длины для функций, определяемых пользователем.

Реализация этой возможности заключается в использовании специальных «сервисных» функций func_num_args(), func_get_arg() и func_get_args().

Необходимости в специфическом синтаксисе нет, при этом список аргументов также может быть указан явно и будет обладать тем же поведением.

Итак, рассмотрим «сервисные» функции:

int func_num_args (void) – возвращает количество аргументов, переданных функции. Генерирует предупреждение при вызове вне определения функции.

Пример:

```
function test()
 {
   $numargs = func_num_args();
   echo "Количество аргументов: ". $numargs;
 }
test('a', 'B', 56); // Количество аргументов: 3
```

Списки аргументов переменной длины

mixed func_get_arg (int arg_num) – возвращает arg_num-ый аргумент из списка аргументов функции.

Нумерация аргументов функции начинается с нуля. Генерирует предупреждение при вызове вне определения функции.

Если arg_num больше количества переданных аргументов, будет сгенерировано предупреждение и func_get_arg() вернёт FALSE.

Пример:

```
function test()
 {
  $numargs = func_num_args();
  if ($numargs >= 2)
   {
     echo "Второй аргумент: " . func_get_arg(1)";
   }
 }
test (1, 2, 3); // Второй аргумент: 2
```

Списки аргументов переменной длины

array func_get_args (void) – возвращает индексный массив, в котором каждый элемент является соответствующим членом списка аргументов функции. Генерирует предупреждение при вызове вне определения функции.

Пример:

```
function test()
 {
  $arg_list = func_get_args();
  print_r($arg_list);
 }

test (‘A’, ‘B’, ‘C’);
// Array ([0] => A, [1] => B, [2] => C)
```

Возврат значений

Функция может не только выводить что-то в выходной поток, но и возвращать значения (что, к слову, является наиболее частым случаем).

Значения функций возвращаются при помощи необязательного оператора возврата **return**. К слову, в одной и той же функции может быть много return, но сработает за один раз выполнения функции не более одного.

Возвращаемые значения могут быть любого типа, в том числе это могут быть массивы и объекты.

Возврат значения приводит к завершению выполнения функции и передаче управления обратно к той строке кода, в которой данная функция была вызвана.

Пример:

```
function square($num)
 {
   return $num * $num;
 }
echo square(4);   // 16
```

Возврат значений

Функция не может возвращать несколько значений, но аналогичного результата можно добиться, возвращая массив.

Пример:

```
function test($a, $b, $c)
{
 return array(++$a, ++$b, ++$c);
}
```

Подумайте (проверьте), что будет, если строку возврата значений записать как return array($a++, $b++, $c++);

Обращение к функциям через переменные

PHP поддерживает концепцию «переменных функций». Это означает, что если к имени переменной присоединены круглые скобки, PHP ищет функцию с тем же именем, что и результат вычисления переменной, и пытается её выполнить.

Эту возможность можно использовать для реализации обратных вызовов, таблиц функций и множества других вещей.

Переменные функции не будут работать с такими языковыми конструкциями как echo(), print(), unset(), isset(), empty(), include(), require() и другими подобными им операторами.

Пример (работа с функциями посредством переменных):

```
function test($x, $y, $z) { … }
$a="test";
$a($x,$y,$z); // вызов функции test()
```

Вы также можете вызвать методы объекта, используя возможности PHP для работы с переменными функциями.

Область видимости переменных

Переменные в функциях имеют локальную область видимости. Это значит, что даже если локальная и внешняя переменные имеют одинаковые имена, то изменение локальной переменной не повлияет на внешнюю переменную.

Пример:

```
function test()
 {
  $var = 5; // локальная переменная
  echo $var; // 5
 }
$var = 10; // глобальная переменная
test(); // 5 (локальная переменная)
echo $var; // 10 (глобальная переменная)
```

Область видимости переменных

Локальную переменную можно сделать глобальной, если перед её именем указать ключевое слово global. Если внешняя (глобальная) переменная объявлена как global, то к ней возможен доступ из любой функции:

Пример:

```
function test()
 {
  global $var;
  $var = 5; // изменяет глобальную переменную
  echo $var; // 5
 }
$var = 10;
echo $var; // 10
test(); // 5 (глобальная переменная изменена)
```

Область видимости переменных

Доступ к глобальным переменным можно получить также через ассоциативный массив $GLOBALS.

Пример:

```
function test()
 {
  $GLOBALS["var"] = 20; // изменяет глобальную переменную
  echo $GLOBALS["var"]; // 20
 }
$var = 10;
echo $var; // 10
test(); // 20 (глобальная переменная изменена)
```

Массив $GLOBALS доступен в области видимости любой функции и содержит все глобальные перемене, которые используются в скрипте.

Время жизни переменных

Временем жизни переменной называется интервал выполнения программы, в течение которого переменная существует.

Поскольку локальные переменные имеют своей областью видимости функцию, то время жизни локальной переменной определяется временем выполнения функции, в которой она объявлена.

Это означает, что в разных функциях совершенно независимо друг от друга могут использоваться переменные с одинаковыми именами.

Локальная переменная при каждом вызове функции инициализируется заново, поэтому функция-счётчик, в приведённом ниже примере всегда будет возвращать значение 1.

Пример:

```
function counter()
 {
  $counter = 0;
  return ++$counter;
}
```

Время жизни переменных

Для того, чтобы локальная переменная сохраняла своё предыдущее значение при новых вызовах функции, её можно объявить статической при помощи ключевого слова static.

Пример:

```
function counter()
 {
  static $counter = 0;
  return ++$counter;
 }
```

Временем жизни статических переменных является время выполнения скрипта. Т.е., если пользователь перезагружает страницу, что приводит к новому выполнению скрипта, переменная $counter в этом случае инициализируется заново.

Время жизни переменных

Время жизни глобальных переменных равно времени выполнения всего скрипта, т.е. как только PHP завершил выполнение скрипта и возвратил результаты работы веб-серверу, все переменные уничтожаются.

Чтобы сохранить какие-то значения для последующего использования можно применять следующие механизмы:

- **сессии** (будет рассмотрено позднее) – самый предпочтительный вариант для «быстрого хранения» в течение сеанса работы пользователя;
- **куки** (cookies) (будет рассмотрено позднее) – для «длительного (не гарантированного!) хранения» значений, когда они понадобятся через неделю, месяц и т.п.;
- сохранять значения переменных **в файлах** (возможно, использую сериализацию (будет рассмотрена позднее)).