Лабораторная № 7

7. Циклы

Циклы служат для многократного повторения фрагмента программы.

Цикл `while`

Цикл `while` работает аналогично оператору `while..do` языка Паскаль.

```
while (выражение) {
       // тело цикла
}
```

До тех пор, пока выражение в скобках истинно, тело цикла выполняется. Как правило, при этом некоторым образом изменяется какая-то переменная или переменные от которых зависит значение выражения в скобках. Если такой зависимости нет, то цикл будет выполняться бесконечно. В листинге 7.1 приведен пример цикла, который выводит таблицу умножения на 2.

Листинг 7.1. Цикл `while`

```
<html> <head>
<title> Листинг 7-1. Цикл while </title>
</head> <body>
<?php
$count = 1;
while ($count <= 10)
{
print "$count умножить на 2 будет ".($count*2)."<br>";
$count++;
}
?>
</body> </html>
```

Здесь мы инициализируем переменную `$count`. В условном выражении цикла проверяется ее значение. До тех пор, пока значение переменной меньше или равно 10, цикл выполняется. В теле цикла переменная `$count` умножается на 2 и выводится в окно браузера. После этого ее значение увеличивается на 1. Данный момент особенно важен. Если вы забудете это сделать, то условное выражение цикла никогда не станет ложным и цикл будет выполняться бесконечно.

Цикл do..while

Цикл do..while напоминает цикл while, перевернутый с ног на голову. Разница состоит в том, что цикл do..while сначала выполняется, а потом проверяется истинность его условия (совершенно аналогично циклу repeat..until языка Паскаль).

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

Тестовое выражение цикла do..while обязательно должно заканчиваться точкой с запятой.

Этот цикл может быть полезен в том случае, если вам нужно, чтобы его тело выполнилось хотя бы один раз, независимо от значения тестового выражения. В листинге 7.2 приведен пример использования цикла do..while. В данном примере цикл будет выполнен один раз.

Листинг 7.2. Цикл do..while

```
<html> <head>
<title> Листинг 7-2. Цикл do..while </title>
</head> <body>
<?php
$num = 1;
  do
    {
     print "Номер прохода: $num<br>\n";
     $num++;
    }
while ($num > 200 && $num < 400);
?>
</body> </html>
```

В условии этого цикла проверяется значение переменной $num, а именно — находится ли оно в интервале от 200 до 400. Поскольку мы инициализировали переменную значением 1, условие ложно, но, несмотря на это, цикл выполнится один раз и отправит на браузер одну строку.

Цикл for

Цикл for отличается от цикла while только тем, что условие изменяется в самой управляющей конструкции, а не где-то внутри блока команд. Цикл for выполняется до тех пор, пока проверяемое условие остается истинным.

```
for (инициализация;  условие;  приращение)
    {
     //тело цикла
    }
```

Выражения в скобках должны быть разделены точками с запятой. В первом выражении счетчику цикла присваивается некоторое начальное значение (инициализация), во втором выражении проверяется условие цикла, а в третьем выражении происходит увеличение или уменьшение счетчика. В листинге 7.3 приведен пример использования цикла for, в котором первые 10 натуральных чисел умножаются на 2.

Листинг 7.3. Цикл for

```
<html> <head>
<title> Листинг 7-3. Цикл for </title>
</head> <body>
<?php
for ($count = 1;  $count <=10;  $count ++ )
{print "$count умножить на 2 будет ".($count*2)."<br>";
}
?>
</body> </html>
```

Результат работы программ из листинга 7.1 и 7.3 совершенно одинаков, однако вторая программа выглядит компактнее. Благодаря тому, что счетчик цикла инициализируется и увеличивается в первой строке цикла, логика программы понятнее и проще.

Когда программа доходит до цикла for, инициализируется счетчик цикла и проверяется его условие. Если значение условия равно true, цикл выполняется. После выполнения всего тела цикла его счетчик изменяется и условие проверяется опять. Этот процесс продолжается до тех пор, пока условие не станет ложным.

Прерывание циклов командой `break`

В циклах `for` и `while` присутствует условие, которое определяет продолжительность выполнения цикла. Однако такое выполнение можно прервать досрочно с помощью команды `break`. Как правило, для этого проверяется некоторое дополнительное условие. Это обычно делается при обнаружении ошибочного состояния. В листинге 7.4 приведен пример программы, в которой одно большое число делится в цикле на другое, постоянно возрастающее, и результат этого деления выводится на экран браузера. В этом примере начальное значение счетчика цикла задано независимо от цикла (до него) и в проверочном выражении цикла `for` сравнивается с числом 10.

Казалось бы, все в порядке. Однако, поскольку начальное значение переменной счетчика было задано отрицательным, то на 5-м шаге выполнения цикла делитель станет равным нулю, а все знают, что деление на ноль запрещено. В данном листинге такая ситуация предусмотрена, и цикл в этом случае прерывается командой `break`.

Листинг 7.4. Использование команды `break`

```
<html> <head>
<title> Листинг 7-4. Использование команды break
</title> </head> <body>
<?php
$count = -4;
for ( ; $count <=10; $count ++ )
    {if ($count == 0)
     break;
     $temp = 4000/$count;
     print "4000 разделить на $count будет $temp<br>";
    }
?>
</body> </html>
```

Деление числа на ноль не вызывает в PHP фатальной ошибки. Вместо этого генерируется предупреждение и выполнение программы продолжается.

С помощью оператора `if` мы проверяем значение счетчика, и если оно равно нулю, команда `break` немедленно прерывает выполнение цикла. Программа продолжает работать с того места, где цикл заканчивается. Обратите внимание на то, что мы инициализируем счетчик цикла вне самого цикла, для того чтобы

имитировать ситуацию когда его значение берется из данных, переданных пользователем или из базы данных.

Любое из трех выражений в скобках цикла `for` может быть опущено, но даже в таком случае точку с запятой нужно указать:

```
$count = 5;
for (; ; $count +=2)
    {print "$count ";
     if ($count == 15) break; // выход из цикла
    }
```

Пропуск итераций с помощью команды `continue`

Команда `continue` служит для пропуска текущей итерации цикла, но не прерывает выполнение цикла окончательно. В результате ее выполнения программа переходит к следующему значению счетчика цикла. В примере листинга 7.4 использование команды `break` было хотя и законным, но с точки зрения здравого смысла не очень оправданным. Разумнее было бы применить команду `continue`, тогда итерация, в которой значение счетчика равно нулю, была бы пропущена и выполнение цикла было бы продолжено со следующего значения, т.е. с 1.

Листинг 7.5. Использование команды `continue`

```
<html> <head>
<title> Листинг 7-5. Использование команды continue
</title> </head> <body>
<?php
$count = -4;
for (  ;  $count <= 10;  $count++ )
    {if ($count == 0)
     continue;
     $temp = 4000/$count;
     print "4000 разделить на $count будет $temp <br>";
    }
?>
</body> </html>
```

Использование команд `continue` и `break` несколько затрудняет чтение и понимание программ, а иногда даже приводит к досадным ошибкам. Пользуйтесь этими командами с некоторой осторожностью.

Вложенные циклы

Тело цикла может содержать в себе другой цикл. Такая конструкция особенно удобна при работе с таблицами. В листинге 7.6 приведен пример использования вложенных циклов, которые выводят на экран браузера таблицу умножения в представленном справа виде.

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

Листинг 7.6. Вложенные циклы `for`

```
<html> <head>
<title> Листинг 7-6.  Вложенные циклы for </title>
</head> <body>
<?php
print "<table border=1>\n";
for ($y=1;  $y <= 5;  $y++)
     {print "<tr>\n";
      for ($x=1;  $x <= 5;  $x++)
          {print "\t<td>";
           print ($x*$y);
           print "</td>\n";
          }
      print "</tr>\n";
     }
print "</table>";
?>
</body> </html>
```

Внешний цикл инициализирует переменную `$y` и записывает в нее значение 1. В этом цикле проверяется, не превысило ли значение счетчика число 5, и счетчик увеличивается на 1. В каждой итерации этого цикла выводится тег `<tr>` (строка таблицы) и запускается другой цикл. Этот внутренний цикл инициализирует переменную `$y`, выводит тег `<td>` (ячейка таблицы) и печатает в ячейку результат умножения `$x` на `$y`. В результате получается таблица умножения.

8. Функции

Функция — это блок команд, который не исполняется немедленно, но может быть вызван программой в случае необходимости. Функции бывают встроенные и созданные пользователем. Они могут требовать для своей работы некоторые данные и возвращать полученное значение.

Вызов функции

Функции бывают двух видов — встроенные, т.е. получаемые вместе с компилятором языка, и те, которые вы создаете самостоятельно. Первая программа (листинг 1.1) состояла из единственного вызова функции

```
print "Hello, Web!";
```

`print()` — не совсем обычная функция. Она не требует, чтобы ее аргумент был заключен в скобки. Таким образом,

```
print "Hello, Web!";
```

и

```
print("Hello, Web!");
```

являются эквивалентными конструкциями. Но это единственное исключение. Все остальные функции требуют открывающей и закрывающей скобок после своего имени, даже если им не передаются никакие аргументы.

Вызов функции состоит из имени функции, в данном случае `print`, и следующей за ним пары скобок. Аргументы функции помещаются в скобках. При создании функции указываются условные имена аргументов. Потом этими именами можно пользоваться в теле функции как локальными переменными. Если у функции аргументов несколько, то они должны быть разделены запятыми.

```
name_function($argument_1, $argument_2);
```

Функция `print()`, как и положено, возвращает вам некоторое значение. Большинство функций, закончив свою работу, возвращают значение, хотя бы просто для того, чтобы показать, успешно ли была выполнена данная работа. Функция `print()` возвращает для этого булево значение.

Функция `abs()`, например, вычисляет абсолютную величину числа; для этого она принимает число со знаком, а возвращает неотрицательное число:

```
print(abs(-321));
```

Функции, которые вы создаете самостоятельно, используются точно так же.

Создание функции

Для создания или, как принято говорить, определения функции используется ключевое слово `function`.

```
function Primer($argument_1, $argument_2)
 {
  // тело функции
 }
```

Имя функции указывается после ключевого слова `function`, а после него следует пара скобок. Если вы хотите, чтобы ваша функция принимала аргументы, то должны поместить в скобках несколько имен переменных, разделив их запятыми. Потом, в теле функции, эти переменные получат те значения, которые вы укажете при вызове функции. Даже если ваша функция не использует никаких аргументов, вы все равно должны после ее имени поставить пару скобок.

В листинге 8.1 приведен пример определения функции с аргументами.

Листинг 8.1. Определение функции с аргументами

```
<html> <head>
<title> Листинг 8-1. Определение функции с аргументами
</title> </head> <body>
<?php
function PrintBR($txt) { print ("$txt<br>\n"); }
PrintBR("Это строка");
PrintBR("Это следующая строка");
PrintBR("Это еще одна строка");
?>
</body> </html>
```

Функция `PrintBR()` должна получать аргумент — строку, поэтому мы при определении функции в ее скобках поместили переменную `$txt`. Значение, переданное функции при вызове, будет записано в эту переменную. В теле функции мы выводим переменную `$txt`, тег `<br>` и символ перевода строки.

Теперь, если нам нужно вывести на экран браузера строку, можно воспользоваться функцией `PrintBR()`, вместо того чтобы вызывать встроенную функцию `print()`, каждый раз добавляя к выводимой строке тег `<br>`.

Создание функции, возвращающей значение

Функция может вернуть значение или объект с помощью оператора `return`. Этот оператор прекращает выполнение функции и посылает возвращаемое значение в вызвавшую программу.

В листинге 8.2 приведен пример функции, возвращающей сумму двух чисел.

Листинг 8.2. Функция, возвращающая значение

```
<html> <head>
<title> Листинг 8-2. Функция, возвращающая значение
</title> </head> <body>
<?php
function AddNums($firstnum, $secondnum) {
         $result = $firstnum + $secondnum;
         return $result;
}
print AddNums(3,5); //будет выведено 8
?>
</body> </html>
```

Программа из листинга 8.2 выведет число 8. Функция `AddNums()` вызывается с двумя числовыми аргументами, в данном случае 3 и 5. Значения этих аргументов записываются в переменные `$firstnum` и `$secondnum`. Функция `AddNums()` складывает эти числа и записывает результат в переменную `$result`. Как уже говорилось раньше, можно сократить текст функции, обойдясь без переменной `$result`, а записав следующим образом:

```
{ return ($firstnum + $secondnum); }
```

Оператор `return` может возвращать значение, объект или не возвращать ничего. Существует несколько способов для того, чтобы указывать возвращаемое значение в операторе `return`. Можно вернуть константу:

```
return 4;
```

а можно — результат выражения:

```
return ($a/$b);
```

или результат вызова функции:

```
return (name_function($argument));
```

Функции-переменные

Одной из интересных конструкций PHP являются *функции-переменные*: имя функции можно присвоить некоторой строковой переменной, а затем обращаться с этим именем точно так же, как с самой функцией. В листинге 8.3 демонстрируется эта непривычная, но полезная возможность. Допустим, программа выводит информацию в зависимости от языка, выбранного пользователем (русский или английский), для чего созданы две функции: `Russian()` и `English()`.

Листинг 8.3. Функция-переменная

```
<html> <head>
<title> Листинг 8-3. Функция-переменная </title>
</head> <body>
<?php
// Приветствие на русском языке
   function Russian() { print "<p>Здравствуйте!"; }
// Приветствие на английском языке
   function English() { print "<p>Hello!"; }
$language = "Russian"; // Выбрали русский язык
$language();           // Выполнение функции-переменной
?>
</body> </html>
```

В переменную `$language` записывается текстовая строка, совпадающая с именем функции `Russian()`. После этого мы можем вызвать саму функцию с помощью данной переменной, добавив к ее имени пару скобок.

Для чего может понадобиться такой способ вызова функций? В данном примере мы просто сделали лишнюю работу, сохранив имя функции в переменной. Однако от этого при других обстоятельствах можно получить определенную пользу. Например, вам может понадобиться изменять поведение программы в зависимости от действии пользователя. Тогда вы имеете возможность сформировать имя функции на основании параметра строки запроса.

Этим же способом можно воспользоваться для вызова встроенных функций PHP.

Область видимости переменных

Переменная, созданная в некоторой функции, становится локальной по отношению к данной функции. Это означает, что она недоступна ни для других функций, ни для любого фрагмента программы вне функции, в которой она создана. В больших программах вышеуказанное поможет вам уберечься от случайного изменения данных, если вы по ошибке или даже намеренно создадите в разных функциях две переменные с одинаковыми именами.

Внутри функции вы не можете просто так обратиться к переменной, которая была создана вне этой функции. Если вы попытаетесь это сделать, то создадите новую переменную с таким же именем, но локальную для данной функции.

Но иногда вам действительно может понадобиться обратиться к некоторой важной переменной, объявленной вне функции, а передавать ее в виде аргумента почему-либо не удобно. В таком случае на помощь приходит команда `global`. В листинге 8.4 приведен пример использования этой команды.

Листинг 8.4. Доступ к глобальной переменной с помощью команды `global`

```
<html> <head>
<title> Листинг 8-4. Доступ к глобальной переменной
</title> </head> <body>
<?php
$day = "воскресенье";
function FreeDay()
   {
    global $day;
    print "<p>Выходной: $day ";
   }
FreeDay();
?> </body> </html>
```

Поместив команду `global` в теле функции перед именем переменной, мы позволяем функции обратиться к переменной, созданной вне ее.

Вы должны использовать команду `global` в каждой из функций, которым нужно обращаться к некоторой переменной, внешней по отношению к ним.

Однако здесь кроется опасность. Если функция изменит значение внешней переменной, то такое изменение скажется на всей программе. Обычно

изменение аргумента функции влияет только на копию переменной, переданной в качестве этого аргумента. Но с внешними переменными это не так. Поэтому пользоваться командой `global` нужно с известной осторожностью.

Запоминание состояния функции между вызовами

Переменные функции живут хоть и счастливо, но недолго. Они возникают при вызове функции и умирают, когда та заканчивает свою работу. Такова жизнь, и это правильно. Всегда лучше построить программу в виде небольших самостоятельных блоков, и чем меньше каждый из них будет знать о состоянии других, тем лучше. Однако иногда приходится делать так, чтобы функция могла помнить что-то о своих предыдущих вызовах.

Предположим, нам нужна функция, которая помнит, сколько раз ее вызывали. Зачем? Ну хотя бы для создания нумерованных заголовков в каком-то документе, выводимом нами на экран.

Конечно, вы уже знаете достаточно много для того, чтобы добиться этого с помощью глобальной переменной. Именно так и сделана программа из листинга 8.5.

Листинг 8.5. Сохранение значения переменной между вызовами с помощью команды `global`

```
<html> <head>
<title> Листинг 8-5. Сохранение значения переменной
        между вызовами </title> </head> <body>
<?php
$num_of_calls = 0;
function ListItem($txt) {
    global $num_of_calls;
    $num_of_calls++;
    print "<b>$num_of_calls: $txt</b>";
}
ListItem("Видеокамеры");
print("<p>Sony, Panasonic");
ListItem("Фотоаппараты");
print("<p>Canon, Casio");
?>
</body> </html>
```

Мы создали переменную вне функции и, для того чтобы функция могла к ней обратиться, воспользовались командой `global`. Вот как выглядит результат работы программы из листинга 8.5:

1: Видеокамеры

Sony, Panasonic

2: Фотоаппараты

Canon, Casio

Каждый раз при вызове функции переменная `$num_of_calls` увеличивается на 1, и таким образом мы можем вывести заголовок с его порядковым номером.

Однако это решение нельзя назвать элегантным. Функцию, использующую команду `global`, невозможно считать независимым блоком, так как она может работать только в контексте данной программы; кроме того, при чтении ее текста нужно обращаться к остальным частям программы для ознакомления с глобальными переменными.

Здесь нам поможет команда `static`. Если мы создадим в функции переменную с такой командой, то данная переменная останется локальной по отношению к данной функции. Но, в отличие от простой переменной, эта будет помнить свое значение от одного вызова функции до другого. В листинге 8.6 приведен пример использования такой команды.

Листинг 8.6. Использование команды `static` для запоминания значения переменной

```
<html> <head>
<title> Листинг 8-6. Использование команды static
</title> </head> <body>
<?php
function ListItem($txt) {
    static $num_of_calls = 0;
    $num_of_calls++;
    print "<b>$num_of_calls. $txt</b>";
   }
ListItem("Видеокамеры ");
print("<p>Sony, Panasonic");
ListItem("Фотоаппараты");
print("<p>Canon, Casio");
?> </body> </html>
```

Эту функцию уже можно назвать независимой. Мы создаем переменную с именем `$num_of_calls` и инициализируем ее. При втором и последующих вызовах значение инициализации игнорируется, а вместо этого запоминается последнее присвоенное значение. Теперь данную функцию можно использовать в разных программах, не заботясь о внешних переменных. Хотя эта программа делает то, что и программа из листинга 8.5, все же такое оформление функции следует признать более правильным.

Значения аргументов по умолчанию

В языке PHP существует средство для придания функциям большей гибкости. Ранее мы уже отмечали, что функция требует один или несколько аргументов. Но дело в том, что некоторые из этих аргументов можно сделать необязательными, уменьшив таким образом требовательность функции.

Создадим функцию, которая выводит текст заданным размером шрифта. Но изменять размер шрифта приходится не так уж часто. Чаще всего мы используем размер, равный 12 пунктам. Существует способ обозначить при создании функции размер шрифта по умолчанию, для чего нужно просто указать данное значение в скобках в определении функции. Если после этого при вызове не проставить значение аргумента размера, то будет использовано именно это значение по умолчанию, т.е. второй аргумент функции становится необязательным. В листинге 8.7 приведен пример использования такого необязательного аргумента.

Листинг 8.7. Функция с необязательным аргументом

```
<html> <head>
<title> Листинг 8-7. Функция с необязательным аргументом
</title> </head> <body>
<?php
function FontSize($txt, $size=12) {
print "<div style=\"font-size:".$size."pt\">$txt</div>";
}
FontSize("<p>Крупный шрифт",16);
FontSize("<p>Нормальный шрифт, первая строка");
FontSize("<p>Нормальный шрифт, вторая строка");
?>
</body> </html>
```

Если при вызове функции `FontSize()` указать значение второго аргумента, то это указанное значение станет использоваться при работе функции. Если

второй аргумент опустить, то при работе его значение будет принято по умолчанию, т.е. будет равно 12. Можно создавать у функции сколько угодно необязательных аргументов, но если при вызове один из них опустить, то все, стоящие справа от него, также должны быть опущены и использовано значение по умолчанию.

Передача аргумента по ссылке

Когда вы передаете функции некоторую переменную в качестве аргумента, то фактически передается копия этой переменной, т.е. значение переменной записывается в аргумент функции. Все изменения, сделанные с такой переменной при работе функции, происходят локально и никак не отражаются на значении самой переменной вне функции.

Однако можно передавать аргументы по ссылке. Это означает, что функция будет работать уже не с копией переданной переменной, а с самой переменной. В таком случае все изменения, сделанные с аргументом в теле функции, отразятся на переданной переменной. Для того чтобы передать функции переменную по ссылке, нужно перед ее именем поставить знак &, причем сделать это можно как при вызове функции, так и при ее определении. В листингах 8.8 и 8.9 продемонстрированы оба эти способа.

Листинг 8.8. Передача аргумента по ссылке при вызове функции

```
<html> <head>
<title> Листинг 8-8. Передача аргумента по ссылке
        при вызове функции
</title>
</head>
<body>
<?php
function AddFive($num)
   {$num +=5;
   }
$var = 10;
AddFive(&$var);
print $var; // выводится 15, без & выведется 10
?>
</body>
</html>
```

Листинг 8.9. Передача аргумента по ссылке при определении функции

```
<html> <head>
<title> Листинг 8-9. Передача аргумента по ссылке
        при определении функции

</title>

</head>

<body>

<?php

function AddFive(&$num)
   {$num += 5;
   }

$var = 10;
AddFive($var);
print $var;

?>
</body>

</html>
```

Пожалуй, более естественно и логично ставить амперсант при определении функции, в таком случае можно быть уверенным, что она всегда будет вести себя одинаково.

* * *

В скрипте использование значений, переданных из строки вызова:

`ls.php?arg1=10&arg2=20`

осуществляется с помощью суперглобального массива **`$_GET[]`** следующим образом:

```
$var1=$_GET['arg1']; // $var1 равно 10
$var2=$_GET['arg2']; // $var2 равно 20
```

В качестве индекса массива **`$_GET[]`** стоит имя аргумента из строки вызова скрипта, находящееся после знака «вопрос» или после знака «амперсант».