# **Лабораторная №8. Управляющие конструкции языка PHP (Часть 1)**

Любой сценарий PHP сформирован из ряда конструкций. Конструкцией могут быть **операторы**, **функции**, **циклы**, **условные операторы**, даже конструкции, которые не делают ничего (пустые конструкции). Конструкции обычно заканчиваются точкой с запятой. Кроме того, конструкции могут быть сгруппированы в группу, формируя группу конструкций с изогнутыми фигурными скобками **{...}**.

Группа конструкций – это также отдельная конструкция. Конструкции языка PHP похожи на конструкции языка Си.

В PHP существуют **шесть** основных групп управляющих конструкций.

Итак, группы управляющих конструкций PHP, которые будут рассмотрены в рамках данной лекции:

## **Условные операторы**

Условные операторы являются, пожалуй, наиболее распространенными конструкциями во всех алгоритмических языках программирования. Рассмотрим основные условные операторы языка PHP.

### **Конструкция if**

Синтаксис конструкции **if** аналогичен конструкции **if** в языке Си:

<?php  
if (логическое выражение) оператор;  
?>

Согласно выражениям PHP, конструкция **if**содержит логическое выражение. Если логическое выражение истинно (**true**), то оператор, следующий за конструкцией **if** будет исполнен, а если логическое выражение ложно (**false**), то следующий за **if** оператор исполнен не будет. Приведем примеры:

<?php   
if ($a > $b) echo "значение a больше, чем b";  
?>

В следующем примере, если переменная **$a** не равна нулю, будет выведена строка «*значение a истинно (true)*»:

<?php   
if ($a) echo "значение a истинно (true) ";  
?>

В следующем примере, если переменная **$a** равна нулю, будет выведена строка «значение a ложно (false)»:

<?php

$a = false;  
if (!$a) echo "значение a ложно (false) ";  
?>

Часто вам будет необходим блок операторов, который будет выполняться при определенном условном критерии, тогда эти операторы необходимо поместить в фигурные скобки **{...}** Пример:

<?php

$a = 5;

$b = 1;

if ($a > $b) {

echo "a больше b";

$c = $a - $b;

echo "<br>", "разность равна $c";

}

?>

Приведенный пример выведет сообщение, «a больше b», если **$a** > **$b**, а затем будет высчитана и выведена разница между ними. Заметим, что данные операторы выполняются в теле конструкции **if**.

### **Конструкция else**

Часто возникает потребность исполнения операторов не только в теле конструкции **if**, если выполнено какое-либо условие конструкции **if**, но и в случае, если условие конструкции **if** не выполнено. В данной ситуации нельзя обойтись без конструкции **else**. В целом, такая конструкция будет называться конструкцией **if-else**.

Синтаксис конструкции **if-else** такой:

if (логическое\_выражение)  
инструкция\_1;  
else  
инструкция\_2;

Действие конструкции **if-else** следующее: если логическое выражение *истинно*, то выполняется **инструкция\_1**, а иначе – **инструкция\_2**. Как и в любом другом языке, конструкция **else** может опускаться, в этом случае при получении должного значения просто ничего не делается.

Если **инструкция\_1** или **инструкция\_2** должны состоять из нескольких команд, то они, как всегда, заключаются в фигурные скобки. Например:

<?php

$a = 5;

$b = 6;  
if ($a > $b) {  
     echo "a больше, чем b";  
} else {  
     echo "a НЕ больше, чем b";  
}  
?>

Конструкция **if-else** имеет еще один **альтернативный синтаксис**:

if (логическое\_выражение):  
 команды;  
elseif(другое\_логическое\_выражение):  
 другие\_команды;  
else:  
 иначе\_команды;  
endif

<?php

$a = 5;

$b = 6;

if ($a > $b) :

echo "a больше, чем b";

else :

echo "a НЕ больше, чем b";

endif

?>

Обратите внимание на расположение **двоеточия (:)**! Если его пропустить, будет сгенерировано сообщение об ошибке. И еще: как обычно, блоки **elseif** и **else** можно опускать.

### **Конструкция elseif**

**elseif** – это комбинация конструкций **if** и **else**. Эта конструкция расширяет условную конструкцию **if-else**.

Приведем синтаксис конструкции **elseif**:

if (логическое\_выражение\_1)  
 оператор\_1;  
elseif (логическое\_выражение\_2)  
 оператор\_2;  
else  
 оператор\_3;

Практический пример использования конструкции **elseif**:

<?php

$a = 5;

$b = 5;  
if ($a > $b) {  
     echo "a больше, чем b";  
} elseif ($a == $b) {  
     echo "a равен b";  
} else {  
     echo "a меньше, чем b";  
}  
?>

Вообще, конструкция **elseif** не очень удобна, поэтому применяется не так часто.

## **Циклы в PHP**

На втором месте по частоте использования, после конструкций условий (условных операторов), находятся **циклы**.

Циклы позволяют повторять определенное (и даже неопределенное – когда работа цикла зависит от условия) количество раз различные операторы. Данные операторы называются **телом цикла**. Проход цикла называется **итерацией**.

PHP поддерживает **три вида циклов**:

* Цикл с предусловием (**while**);
* Цикл с постусловием (**do-while**);
* Цикл со счетчиком (**for**);
* Специальный цикл перебора массивов (**foreach**).

При использовании циклов есть возможность использования операторов **break** и **continue.**Первый из них прерывает работу всего цикла, а второй – только текущей итерации.

Рассмотрим циклы PHP:

### **Цикл с предусловием while**

Цикл с предусловием **while** работает по следующим принципам:

1. Вычисляется значение логического выражения.
2. Если значение истинно, выполняется тело цикла, в противном случае – переходим на следующий за циклом оператор.

Синтаксис цикла с предусловием:

while (логическое\_выражение)   
 инструкция;

В данном случае телом цикла является инструкция. Обычно тело цикла состоит из большого числа операторов. Приведем пример цикла с предусловием **while**:

<?php  
$x = 0;  
while ($x++<10) echo $x;  
*// Выводит 12345678910*?>

Обратите внимание на последовательность выполнения операций условия $x++<10. Сначала проверяется условие, а только потом увеличивается значение переменной. Если мы поставим операцию инкремента перед переменной (++$x<10), то сначала будет выполнено увеличение переменной, а только затем – сравнение. В результате мы получим строку123456789. Этот же цикл можно было бы записать по-другому:

<?php  
$x=0;  
while ($x<10)  
{  
$x++; *// Увеличение счетчика*  
echo $x;  
}  
*// Выводит 12345678910*  
?>

Если мы увеличим счетчик после выполнения оператора **echo**, мы получим строку 0123456789. В любом случае, мы имеем **10** итераций. **Итерация** – это выполнение операторов внутри тела цикла.

Подобно конструкции условного оператора **if**, можно группировать операторы внутри тела цикла **while**, используя следующий альтернативный синтаксис:

while (логическое\_выражение):  
 инструкция;  
...  
endwhile;

Пример использования альтернативного синтаксиса:

<?php  
$x = 1;  
while ($x <= 10):  
     echo $x;  
     $x++;  
endwhile;  
?>

### **Цикл с постусловием do while**

В отличие от цикла **while**, этот цикл проверяет значение выражения не **до**, а **после** каждого прохода (итерации). Таким образом, тело цикла выполняется хотя бы один раз. Синтаксис цикла с постусловием такой:

do   
{  
тело\_цикла;  
}  
while (логическое\_выражение);

После очередной итерации проверяется, истинно ли **логическое\_выражение**, и, если это так, управление передается вновь на начало цикла, в противном случае цикл обрывается.

Альтернативного синтаксиса для **do-while** разработчики PHP не предусмотрели (видимо, из-за того, что, в отличие от прикладного программирования, этот цикл довольно редко используется при программировании web-приложений).

Пример скрипта, показывающего работу цикла с постусловием **do-while**:

<?php  
$x = 1;  
do {  
     echo $x;  
} while ($x++<10);  
?>

Рассмотренный сценарий выводит: 12345678910

### **Цикл со счетчиком for**

Цикл со счетчиком используется для выполнения тела цикла определенное число раз. С помощью цикла **for** можно (и нужно) создавать конструкции, которые будут выполнять действия совсем не такие тривиальные, как простая переборка значения счетчика.

Синтаксис цикла **for** такой:

for (инициализирующие\_команды; условие\_цикла; команды\_после\_итерации) {

тело\_цикла; }

Цикл **for** начинает свою работу с выполнения **инициализирующих\_команд**. Данные команды выполняются только один раз. После этого проверяется **условие\_цикла**, если оно истинно (**true**), то выполняется **тело\_цикла**. После того, как будет выполнен последний оператор тела, выполняются **команды\_после\_итерации**. Затем снова проверяется **условие\_цикла**. Если оно истинно (**true**), выполняется **тело\_цикла** и **команды\_после\_итерации**, и.т.д.

<?php  
for ($x=0; $x<10; $x++) echo $x;  
?>

Данный сценарий выводит: 0123456789

Есть вариант вывода строки 12345678910:

<?php  
for ($x=0; $x++<10;) echo $x;  
*// Выводит 12345678910*?>

В данном примере мы обеспечили увеличение счетчика при проверке логического выражения. В таком случае нам не нужны были команды, выполняющиеся после итерации.

Если необходимо указать несколько команд, их можно разделить запятыми, пример:

<?php  
for ($x=0, $y=0; $x<10; $x++, $y++) echo $x;  
*// Выводит 0123456789*  
?>

Приведем еще один, более практичный пример использования нескольких команд в цикле **for**:

<?php  
for($i=0,$j=0,$k="Точки"; $i<10; $j++,$i+=$j) {

$k=$k."."; echo $k;

}  
*// Выводит Точки.Точки..Точки...Точки....*  
?>

Рассмотренный пример (да и вообще любой цикл **for**) можно реализовать и через **while**, только это будет выглядеть не так изящно и лаконично.

Для цикла **for** имеется и альтернативный синтаксис:

for(инициализирующие\_команды; условие\_цикла; команды\_после\_итерации):  
операторы;  
endfor;

### **Цикл перебора массивов foreach**

В PHP4 появился еще один специальный тип цикла – **foreach**. Данный цикл предназначен специально для перебора массивов.

Синтаксис цикла **foreach** выглядит следующим образом:

foreach (массив as $ключ=>$значение)  
команды;

Здесь команды циклически выполняются для каждого элемента массива, при этом очередная пара **ключ=>значение** оказывается в переменных **$ключ** и **$значение**. Приведем пример работы цикла **foreach**:

<?php  
$names["Иванов"] = "Андрей";  
$names["Петров"] = "Борис";  
$names["Волков"] = "Сергей";  
$names["Макаров"] = "Федор";  
foreach ($names as $key => $value) {  
echo "<b>$value $key</b><br>";  
}  
?>

Рассмотренный сценарий выводит:

**Андрей Иванов**  
**Борис Петров**  
**Сергей Волков**  
**Федор Макаров**

У цикла **foreach** имеется и другая форма записи, которую следует применять, когда нас не интересует значение ключа очередного элемента.

Выглядит она так:

foreach (массив as $значение)  
команды;

В этом случае доступно лишь значение очередного элемента массива, но не его ключ. Это может быть полезно, например, для работы с массивами-списками:

<?php  
$names[] = "Андрей";  
$names[] = "Борис";  
$names[] = "Сергей";  
$names[] = "Федор";  
foreach ($names as $value) {  
echo "<b>$value</b><br>";  
}  
?>

Рассмотренный сценарий выводит:

**Андрей**  
**Борис**   
**Сергей**   
**Федор**

**Важно**: Цикл **foreach** оперирует не исходным массивом, а его копией. Это означает, что любые изменения, которые вносятся в массив, не могут быть «видны» из тела цикла. Что позволяет, например, в качестве массива использовать не только переменную, но и результат работы какой-нибудь функции, возвращающей массив (в этом случае функция будет вызвана всего один раз – до начала цикла, а затем работа будет производиться с копией возвращенного значения).

### **Конструкция break**

Очень часто для того, чтобы упростить логику какого-нибудь сложного цикла, удобно иметь возможность его прервать в ходе очередной итерации (к примеру, при выполнении какого-нибудь особенного условия). Для этого и существует конструкция **break**, которая осуществляет немедленный выход из цикла. Она может задаваться с одним необязательным параметром – числом, которое указывает, из какого вложенного цикла должен быть произведен выход. По умолчанию используется **1**, т. е. выход из текущего цикла, но иногда применяются и другие значения. Синтаксис конструкции **break**:

break; *// По умолчанию*  
break(номер\_цикла); *// Для вложенных циклов (указывается номер прерываемого цикла)*

Приведем примеры:

<?php  
$x = 0;  
while ($x++<10) {  
if ($x==3) break;  
echo "<b>Итерация $x</b><br>";  
}  
*// Когда $x равен 3, цикл прерывается*  
?>

Рассмотренный сценарий выводит:

**Итерация 1**  
**Итерация 2**

Если нам нужно прервать работу определенного (вложенного) цикла, то нужно передать конструкции **break** параметр – **номер\_цикла**, например, **break(1)**. Нумерация циклов выглядит следующим образом:

for (...) *// Третий цикл*  
{  
  for (...) *// Второй цикл*  
  {  
     for (...) *// Первый цикл*  
     {  
     }  
  }  
}

**<?php**

**for ($i=1;$i<=4;$i++){**

**for ($j=-2;$j<=15;$j++){**

**echo "i={$i}","j=$j","<br>";**

**if ($j>=$i) break(2);**

**}**

**echo "<br>";**

**}**

**?>**

**<?php**

**for ($i=1;$i<=4;$i++){**

**for ($j=-2;$j<=15;$j++){**

**echo "i={$i}","j=$j","<br>";**

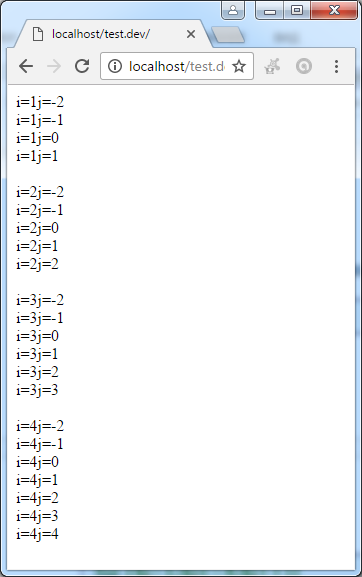
**if ($j>=$i) break(1);**

**}**

**echo "<br>";**

**}**

**?>**



### **Конструкция continue**

Конструкция **continue** так же, как и **break**, работает только «в паре» с циклическими конструкциями. Она немедленно завершает текущую итерацию цикла и переходит к новой (конечно, если выполняется условие цикла для цикла с предусловием). Точно так же, как и для **break**, для **continue** можно указать уровень вложенности цикла, который будет продолжен по возврату управления.

В основном **continue** позволяет вам сэкономить количество фигурных скобок в коде и увеличить его удобочитаемость. Это чаще всего бывает нужно в циклах-фильтрах, когда требуется перебрать некоторое количество объектов и выбрать из них только те, которые удовлетворяют определенным условиям. Приведем пример использования конструкции **continue**:

<?php  
$x=0;  
 while ($x++<5) {  
 if ($x==3) continue;  
echo "<b>Итерация $x</b><br>";  
}  
*// Цикл прервется только на третьей итерации*?>

Рассмотренный скрипт выводит:

**Итерация 1**  
**Итерация 2**

**Итерация 4**  
**Итерация 5**

Грамотное использование **break** и **continue** позволяет заметно улучшить «читабельность» кода и количество блоков **else**.

## **Конструкции выбора**

Часто вместо нескольких расположенных подряд инструкций **if-else** целесообразно воспользоваться специальной конструкцией выбора **switch-case**. Данная конструкция предназначена для выбора действий, в зависимости от значения указанного выражения. Конструкция **switch-case** чем-то напоминает конструкцию **if-else**, который, по сути, является ее аналогом. Конструкцию выбора можно использовать, если предполагаемых вариантов много, например, более 5, и для каждого варианта нужно выполнить специфические действия. В таком случае, использование конструкции **if-else** становится действительно неудобным.

Синтаксис конструкции **switch-case** такой:

switch(выражение) {  
 case значение1: команды1; [break;]  
 case значение2: команды2; [break;]  
. . .  
 case значениеN: командыN; [break;]  
 [default: команды\_по\_умолчанию; [break]]  
}

Принцип работы конструкции **switch-case** такой:

1. Вычисляется значение выражения;
2. Просматривается набор значений. Пусть **значение1** равно значению выражения, вычисленного на первом шаге. Если не указана конструкция (оператор) **break**, то будут выполнены команды i, i+1, i+2, ... , N. В противном случае (есть **break**) будет выполнена только команда с номером i.
3. Если ни одно значение из набора не совпало со значением выражения, тогда выполняется блок **default**, если он указан.

Приведем примеры использования конструкции **switch-case**:

<?php

$x = 1;

echo "Работает if else", "<br>";

if ($x == 0) {

echo "x=0<br>";

} elseif ($x == 1) {

echo "x=1<br>";

} elseif ($x == 2) {

echo "x=2<br>";

}

*// Используем switch-case*

echo "Работает switch", "<br>";

switch ($x) {

case 0:

echo "x=0<br>";

break;

case 1:

echo "x=1<br>";

break;

case 2:

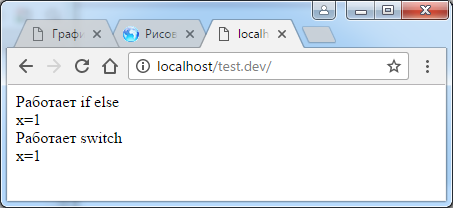
echo "x=2<br>";

break;

}

?>

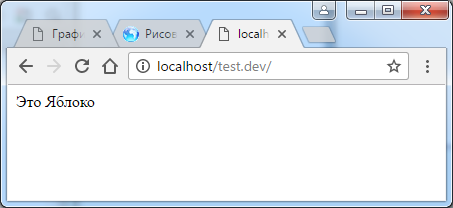
Рассмотренный сценарий выводит *x=1* дважды.



Еще пример использования конструкции **switch-case**:

<?php  
$x = "Яблоко";  
switch ($x) {  
case "Яблоко":  
    echo "Это Яблоко";  
    break;  
case "Груша":  
    echo "Это Груша";  
    break;  
case "Арбуз":  
    echo "Это Арбуз";  
    break;  
}  
?>

Данный скрипт выводит «Это Яблоко».



Конструкция **switch** выполняется поэтапно. Сперва никакой код не исполнен. Только, когда конструкция **case** найдена со значением, которое соответствует значению выражения **switch**, PHP начинает исполнять конструкции. PHP продолжает исполнять конструкции до конца блока **switch**, пока не встречается оператор **break**. Если не использовать конструкции (операторы) **break**, скрипт будет выглядеть так:

<?php

$x = 1;

switch ($x) {

case 0:

echo "x=$x<br>";

case 1:

echo "x=$x<br>";

case 2:

echo "x=$x<br>";

}

*// Без использования break выводит*

*// x=1*

*// x=1*

?>

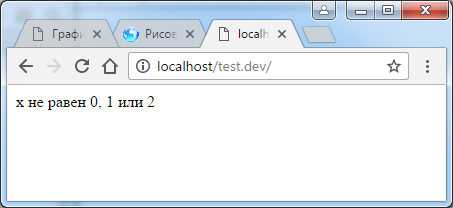
Операторный список для **case** может быть также пуст, он просто передает управление в операторный список до следующей конструкции **case**:

<?php  
switch ($x) {  
 case 0:  
 case 1:  
 case 2:  
     echo "x меньше, чем 3, но не отрицателен";  
     break;  
 case 3:  
     echo "x=3";  
}  
?>

Когда ни одно значение из набора не совпало со значением выражения, тогда выполняется блок **default**, если он указан, например:

<?php  
$x = 3;  
 switch ($x) {  
 case 0:  
     echo "x=0";  
     break;  
 case 1:  
     echo "x=1";  
     break;  
 case 2:  
     echo "x=2";  
     break;  
 default:  
     echo "x не равен 0, 1 или 2";  
}  
?>

Данный скрипт выводит *«x не равен 0, 1 или 2»,* поскольку переменная **$x=3**.



Конструкция **switch-case** также имеет альтернативный синтаксис:

switch(выражение):  
 case значение1: команды1; [break;]  
. . .  
 case значениеN: командыN; [break;]  
 [default: команды\_по\_умолчанию; [break]]  
endswitch;

Практический пример использования альтернативного синтаксиса для конструкции **switch-case**:

<?php  
$x = 3;  
 switch ($x):  
 case 0:  
    echo "x=0";  
     break;  
 case 1:  
    echo "x=1";  
     break;  
 case 2:  
    echo "x=2";  
     break;  
 default:  
    echo "x не равен 0, 1 или 2";  
 endswitch;  
?>

Как вы уже поняли, данный скрипт выводит *«x не равен 0, 1 или 2»*, так как **$x=3**.