Управляющие конструкции

if

Конструкция if предоставляет возможность условного выполнения фрагментов кода.

if (выражение) инструкция

Выражение вычисляется в булево значение. Если выражение принимает значение TRUE, PHP выполнит инструкцию, а если оно принимает значение FALSE - проигнорирует.

Инструкции if могут быть бесконечно вложены в другие инструкции if.

else

Часто необходимо выполнить одно выражение, если определенное условие верно, и другое выражение, если условие не верно. Именно для этого else и используется. else расширяет оператор if, чтобы выполнить выражение, в случае если условие в операторе if равно FALSE.

elseif/else if

Конструкция elseif есть сочетание if и else. В отличии от else, выполнение альтернативного выражения произойдет только тогда, когда условие оператора elseif будет являться равным TRUE. Может быть несколько elseif в одном if выражении. Первое же выражение elseif (если будет хоть одно) равное TRUE будет выполнено. В PHP вы также можете написать 'else if' (в два слова), и тогда поведение будет идентичным 'elseif' (в одно слово).

Заметьте, что elseif и else if будут равнозначны только при использовании фигурных скобок. Если используются двоеточие для определения условий if/elseif, Вы не должны разделять else if в два слова, иначе это вызовет фатальную ошибку в PHP.

```php
<?php
    /* Некорректный способ: */
    if($a > $b):
        echo 1;
    else if($a == $b): // Не скомпилируется.
        echo 2;
    endif;

    /* Корректный способ: */
    if($a > $b):
        echo 1;
    elseif($a == $b): // Заметьте, тут одно слово.
        echo 2;
    endif;
?>
```

Альтернативный синтаксис управляющих структур

PHP предлагает альтернативный синтаксис для некоторых его управляющих структур, а именно: if, while, for, foreach и switch. В каждом случае основной формой альтернативного синтаксиса является изменение открывающей фигурной скобки на двоеточие (:), а закрывающей скобки на endif;, endwhile;, endfor;, endforeach; или endswitch; соответственно.

```
<?php if ($a == 5): ?>
A равно 5
<?php endif; ?>
```

Внимание Любой вывод (включая пробельные символы) между выражением *switch* и первым *case* приведут к синтаксической ошибке. Например этот код не будет работать:

```
<?php switch ($foo): ?>
    <?php case 1: ?>
    ...
<?php endswitch ?>
```

В то же время следующий пример будет работать, так как завершающий перевод строки после выражения *switch* считается частью закрывающего ?> и следовательно ничего не выводится между *switch* и *case*:

```
<?php switch ($foo): ?>
<?php case 1: ?>
    ...
<?php endswitch ?>
```

while

Циклы while являются простейшим видом циклов в PHP.

while (expr) statement

Смысл выражения while очень прост. Оно указывает PHP выполнять вложенные выражения повторно до тех пор, пока выражение в самом while является TRUE. Значение выражения expr проверяется каждый раз перед началом цикла, поэтому даже если значение выражения изменится в процессе выполнения вложенных выражений в цикле, выполнение не прекратится до конца итерации (каждый раз, когда PHP выполняет выражения в цикле - это одна итерация). В том случае, если выражение while равно FALSE с самого начала, вложенные выражения ни разу не будут выполнены.

```php
<?php
    $i = 1;

    while ($i <= 10) {
        echo $i++;
    }

    /* выводится будет значение переменной
    $i перед её увеличением
    (post-increment) */
    // 12345678910
?>
```

do-while

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

```php
<?php
    $i = 0;

    do {
        echo $i;
    } while ($i > 0);

    // 0
?>
```

for

Цикл for самый сложный цикл в PHP.

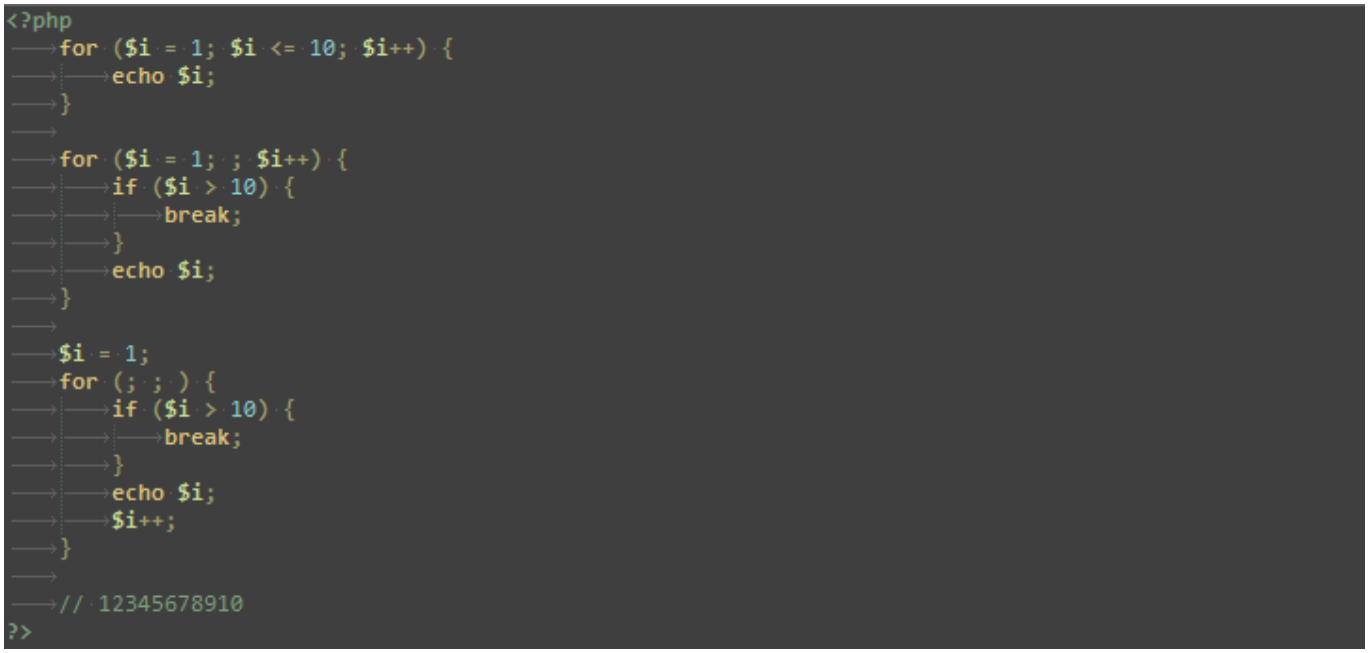
for (expr1; expr2; expr3) statement

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

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

В конце каждой итерации выражение expr3 вычисляется (выполняется).

Каждое из выражений может быть пустым или содержать несколько выражений, разделенных запятыми. В expr2 все выражения, разделенные запятыми, вычисляются, но результат берется из последнего. Если выражение expr2 отсутствует, это означает, что цикл будет выполняться бесконечно. (PHP неявно воспринимает это значение как TRUE).



```php
<?php
    for ($i = 1; $i <= 10; $i++) {
        echo $i;
    }

    for ($i = 1; ; $i++) {
        if ($i > 10) {
            break;
        }
        echo $i;
    }

    $i = 1;
    for (; ; ) {
        if ($i > 10) {
            break;
        }
        echo $i;
        $i++;
    }

    // 12345678910
?>
```

foreach

Конструкция foreach предоставляет простой способ перебора массивов. Foreach работает только с массивами и объектами, и будет генерировать ошибку при попытке использования с переменными других типов или неинициализированными переменными. Существует два вида синтаксиса:

foreach (array_expression as $value)
statement

foreach (array_expression as $key => $value)
statement

Первый цикл перебирает массив, задаваемый с помощью array_expression. На каждой итерации значение текущего элемента присваивается переменной $value и внутренний указатель массива увеличивается на единицу (таким образом, на следующей итерации цикла работа будет происходить со следующим элементом).

Второй цикл будет дополнительно соотносить ключ текущего элемента с переменной $key на каждой итерации.

Для того, чтобы напрямую изменять элементы массива внутри цикла, переменной $value должен предшествовать знак &. В этом случае значение будет присвоено по ссылке. Ссылка $value на последний элемент массива останется после окончания цикла foreach. Рекомендуется уничтожать ее с помощью unset(). Оператор foreach не поддерживает возможность подавления сообщений об ошибках с помощью префикса '@'. В PHP 5.5 была добавлена возможность обхода массива массивов с распаковкой вложенного массива в переменные цикла, передав list() в качестве значения.

```php
<?php
    $arr = array(1, 2, 3, 4);

    print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )

    foreach ($arr as &$value) {
        $value = $value * 2;
    }

    print_r($arr); // Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 8 )

    /*********************************/

    $array = [
        [1, 2],
        [3, 4],
    ];

    foreach ($array as list($a, $b)) {
        echo "A: $a; B: $b\n"; // A: 1; B: 2 A: 3; B: 4
    }
?>
```

break

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

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

continue

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

В PHP, структура switch считается циклической, и внутри нее может использоваться continue. Если continue не передано аргументов, то он ведет себя аналогично break. Если switch расположен внутри цикла, continue 2 продолжит выполение внешнего цикла со следующей итерации.

continue принимает необязательный числовой аргумент, который указывает на скольких уровнях вложенных циклов будет пропущена оставшаяся часть итерации. Значением по умолчанию является 1, при которой пропускается оставшаяся часть текущего цикла.

switch

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

Важно понять, как оператор switch выполняется, чтобы избежать ошибок. Оператор switch исполняет строчка за строчкой (на самом деле выражение за выражением). В начале никакой код не исполняется. Только в случае нахождения оператора case, значение которого совпадает со значением выражения в операторе switch, PHP начинает исполнять операторы. PHP продолжает исполнять операторы до конца блока switch либо до тех пор, пока не встретит оператор break. Если вы не напишете оператор break в конце секции case, PHP будет продолжать исполнять команды следующей секции case.

В операторе switch выражение вычисляется один раз и этот результат сравнивается с каждым оператором case. В выражении elseif, выражение вычисляется снова. Если ваше условие более сложное, чем простое сравнение и/или находится в цикле, конструкция switch может работать быстрее.

Список операторов для исполнения в секции case также может быть пустым, что просто передает управление списку операторов в следующей секции case.

Специальный вид конструкции case -- default. Сюда управление попадает тогда, когда не сработал ни один из других операторов case.

Возможно использование точки с запятой вместо двоеточия после оператора case.

```php
<?php
    $i = 1;

    switch ($i) {
        case 0:
            echo 0;
            break;
        case 1:
            echo 1;
            break;
        case 2:
        case 3:
        case 'Hello':
            echo 'Hello';
            break;
        default:
            echo 'Nothing!';
    }

    // 1

    /***********************/

    $beer = 'Tuborg';

    switch($beer) {
        case 'Tuborg';
        case 'Carlsberg';
        case 'Heineken';
            echo 'Good!';
        break;
        default;
            echo 'Please, select the beer!';
        break;
    }

    // Good!
?>
```

declare

Конструкция declare используется для установки директив исполнения для блока кода. Синтаксис declare схож с синтаксисом других конструкций управления исполнением:

declare (directive)
statement

Секция directive позволяет установить поведение блока declare. В настоящее время распознаются только две директивы: директива ticks и директива encoding.

Так как директива обрабатывается при компиляции файла, то только символьные данные могут использоваться как значения директивы. Нельзя использовать переменные и константы.

Тики

Тик - это событие, которое случается каждые N низкоуровневых операций, выполненных парсером внутри блока declare. Значение N задается, используя ticks=N внутри секции directive блока declare.

Событие (или несколько событий), которое возникает на каждом тике определяется, используя register_tick_function().

Кодировка

Кодировка скрипта может быть указана для каждого скрипта используя директиву encoding.

return

return возвращает управление программой модулю, из которого была вызвана функция. Выполнение программы продолжается с инструкции, следующей за местом вызова.

Если вызвано из функции, выражение return немедленно прекращает выполнение текущей функции и возвращает свой аргумент как значение данной функции.

require

require идентично include за исключением того, что при ошибке оно также выдаст фатальную ошибку уровня E_COMPILE_ERROR. Другими словами, она остановит выполнение скрипта, тогда как include только выдала бы предупреждение E_WARNING, которое позволило бы скрипту продолжить выполнение.

require_once

Выражение require_once идентично require за исключением того, что PHP проверит, включался ли уже данный файл, и, если да, не будет включать его еще раз.

include

Выражение include включает и выполняет указанный файл.

include_once

Выражение include_once включает и выполняет указанный файл во время выполнения скрипта. Его поведение идентично выражению include, с той лишь разницей, что если код из файла уже один раз был включен, он не будет включен и выполнен повторно и вернёт TRUE. Как видно из имени, он включит файл только один раз (include once).

include_once может использоваться в тех случаях, когда один и тот же файл может быть включен и выполнен более одного раза во время выполнения скрипта, в данном случае это поможет избежать проблем с переопределением функций, переменных и т.д.

goto

Оператор goto используется для перехода в другую часть программы. Место, куда необходимо перейти указывается с помощью метки, за которой ставится двоеточие, после оператора goto указывается желаемая метка для перехода. Целевая метка должна находиться в том же файле, в том же контексте. Имеется ввиду, что вы не можете ни перейти за границы функции или метода, ни перейти внутрь одной из них. Вы также не можете перейти внутрь любой циклической структуры или оператора switch. Но вы можете выйти из них, и обычным применением оператора goto является использование его вместо многоуровневых break.