PHP Loops

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

* while - loops through a block of code as long as the specified condition is true
* do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
* for - loops through a block of code a specified number of times
* foreach - loops through a block of code for each element in an array

The while loop - Loops through a block of code as long as the specified condition is true.

## The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

### Example

Print  $i as long as  $i is less than 6:

$i = 1;

while ($i < 6) {

echo $i;

$i++;

}

**Note:** remember to increment $i, or else the loop will continue forever.

The while loop does not run a specific number of times, but checks after each iteration if the condition is still true.

The condition does not have to be a counter, it could be the status of an operation or any condition that evaluates to either true or false.

## The break Statement

With the break statement we can stop the loop even if the condition is still true:

### Example

Stop the loop when $i is 3:

$i = 1;

while ($i < 6) {

if ($i == 3) break;

echo $i;

$i++;

}

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Stop, and jump to the next iteration if $i is 3:

$i = 0;

while ($i < 6) {

$i++;

if ($i == 3) continue;

echo $i;

}

## Alternative Syntax

The while loop syntax can also be written with the endwhile statement like this

### Example

Print $i as long as $i is less than 6:

$i = 1;

while ($i < 6):

echo $i;

$i++;

endwhile;

--------------------------------------------------------------------------------------------------------------------------------------

If you want the while loop count to 100, but only by each 10, you can increase the counter by 10 instead 1 in each iteration:

### Example

Count to 100 by tens:

$i = 0;

while ($i < 100) {

$i+=10;

echo $i "<br>";

}

# PHP do while Loop

The do...while loop - Loops through a block of code once, and then repeats the loop as long as the specified condition is true.

## The PHP do...while Loop

The do...while loop will always execute the block of code at least once, it will then check the condition, and repeat the loop while the specified condition is true.

### Example

Print $i as long as $i is less than 6:

$i = 1;

do {

echo $i;

$i++;

} while ($i < 6);

**Note:** In a do...while loop the condition is tested AFTER executing the statements within the loop. This means that the do...while loop will execute its statements at least once, even if the condition is false. See example below.

Let us see what happens if we set the $i variable to 8 instead of 1, before execute the same do...while loop again:

### Example

Set $i = 8, then print $i as long as $i is less than 6:

$i = 8;

do {

echo $i;

$i++;

} while ($i < 6);

The code will be executed once, even if the condition is never true.

## The break Statement

With the break statement we can stop the loop even if the condition is still true:

### Example

Stop the loop when $i is 3:

$i = 1;

do {

if ($i == 3) break;

echo $i;

$i++;

} while ($i < 6);

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Stop, and jump to the next iteration if $i is 3:

$i = 0;

do {

$i++;

if ($i == 3) continue;

echo $i;

} while ($i < 6);

# PHP for Loop

The for loop - Loops through a block of code a specified number of times.

## The PHP for Loop

The for loop is used when you know how many times the script should run.

### Syntax

for (*expression1*, *expression2*, *expression3*) {

// *code block*

}

This is how it works:

* expression1 is evaluated once
* expression2 is evaluated before each iterarion
* expression3 is evaluated after each iterarion

### Example

Print the numbers from 0 to 10:

for ($x = 0; $x <= 10; $x++) {

echo "The number is: $x <br>";

}

### Example Explained

1. The first expression, $x = 0;, is evaluated once and sets a counter to 0.
2. The second expression, $x <= 10;, is evaluated before each iteration, and the code block is only executed if this expression evaluates to true. In this example the expression is true as long as $x is less than, or equal to, 10.
3. The third expression, $x++;, is evaluated after each iteration, and in this example, the expression increases the value of $x by one at each iteration.

## The break Statement

With the break statement we can stop the loop even if the condition is still true:

### Example

Stop the loop when $i is 3:

for ($x = 0; $x <= 10; $x++) {

if ($i == 3) break;

echo "The number is: $x <br>";

}

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Stop, and jump to the next iteration if $i is 3:

for ($x = 0; $x <= 10; $x++) {

if ($x == 3) continue;

echo "The number is: $x <br>";

}

## Step 10

This example counts to 100 by tens:

### Example

for ($x = 0; $x <= 100; $x+=10) {

echo "The number is: $x <br>";

}

# PHP foreach Loop

The foreach loop - Loops through a block of code for each element in an array or each property in an object.

## The foreach Loop on Arrays

The most common use of the foreach loop, is to loop through the items of an array.

### Example

Loop through the items of an indexed array:

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {

echo "$x <br>";

}

For every loop iteration, the value of the current array element is assigned to the variabe $x. The iteration continues until it reaches the last array element.

## Keys and Values

The array above is an [indexed](https://www.w3schools.com/php/php_arrays_indexed.asp) array, where the first item has the key 0, the second has the key 1, and so on.

[Associative](https://www.w3schools.com/php/php_arrays_associative.asp) arrays are different, associative arrays use named keys that you assign to them, and when looping through associative arrays, you might want to keep the key as well as the value.

This can be done by specifying both the key and value in the foreach defintition, like this:

### Example

Print both the key and the value from the $members array:

$members = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach ($members as $x => $y) {

echo "$x : $y <br>";

}

## The foreach Loop on Objects

The foreach loop can also be used to loop through properties of an object:

### Example

Print the property names and values of the $myCar object:

class Car {

public $color;

public $model;

public function \_\_construct($color, $model) {

$this->color = $color;

$this->model = $model;

}

}

$myCar = new Car("red", "Volvo");

foreach ($myCar as $x => $y) {

echo "$x: $y <br>";

}

You will learn more about objects in the [PHP Objects and Classes](https://www.w3schools.com/php/php_oop_classes_objects.asp) chapter.

## The break Statement

With the break statement we can stop the loop even if it has not reached the end:

### Example

Stop the loop if $x is "blue":

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {

if ($x == "blue") break;

echo "$x <br>";

}

## The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

### Example

Stop, and jump to the next iteration if $x is "blue":

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {

if ($x == "blue") continue;

echo "$x <br>";

}

## Foreach Byref

When looping through the array items, any changes done to the array item will, by default, NOT affect the original array:

### Example

By default, changing an array item will not affect the original array:

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {

if ($x == "blue") $x = "pink";

}

var\_dump($colors);

BUT, by using the & character in the foreach declaration, the array item is assigned by reference, which results in any changes done to the array item will also be done to the original array:

### Example

By assigning the array items by reference, changes will affect the original array:

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as &$x) {

if ($x == "blue") $x = "pink";

}

var\_dump($colors);

## Alternative Syntax

The foreach loop syntax can also be written with the endforeach statement like this

### Example

Loop through the items of an indexed array:

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) :

echo "$x <br>";

endforeach;

**PHP Functions**

The real power of PHP comes from its functions.

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

## PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

Please check out our PHP reference for a complete overview of the [PHP built-in functions](https://www.w3schools.com/php/php_ref_overview.asp).

## PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

* A function is a block of statements that can be used repeatedly in a program.
* A function will not execute automatically when a page loads.
* A function will be executed by a call to the function.

## Create a Function

A user-defined function declaration starts with the keyword function, followed by the name of the function:

### Example

function myMessage() {

echo "Hello world!";

}

**Note:** A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

**Tip:** Give the function a name that reflects what the function does!

## Call a Function

To call the function, just write its name followed by parentheses ():

### Example

function myMessage() {

echo "Hello world!";

}

myMessage();

In our example, we create a function named myMessage().

The opening curly brace { indicates the beginning of the function code, and the closing curly brace } indicates the end of the function.

The function outputs "Hello world!".

## PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name, e.g. ("Jani"), and the name is used inside the function, which outputs several different first names, but an equal last name:

### Example

function familyName($fname) {

echo "$fname Refsnes.<br>";

}

familyName("Jani");

familyName("Hege");

familyName("Stale");

familyName("Kai Jim");

familyName("Borge");

The following example has a function with two arguments ($fname, $year):

### Example

function familyName($fname, $year) {

echo "$fname Refsnes. Born in $year <br>";

}

familyName("Hege", "1975");

familyName("Stale", "1978");

familyName("Kai Jim", "1983");

## PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

### Example

function setHeight($minheight = 50) {

echo "The height is : $minheight <br>";

}

setHeight(350);

setHeight(); // will use the default value of 50

setHeight(135);

setHeight(80);

## PHP Functions - Returning values

To let a function return a value, use the return statement:

### Example

function sum($x, $y) {

$z = $x + $y;

return $z;

}

echo "5 + 10 = " . sum(5, 10) . "<br>";

echo "7 + 13 = " . sum(7, 13) . "<br>";

echo "2 + 4 = " . sum(2, 4);

## Passing Arguments by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.

When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used:

### Example

Use a pass-by-reference argument to update a variable:

function add\_five(&$value) {

$value += 5;

}

$num = 2;

add\_five($num);

echo $num;