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;

An Array stores multiple values in one single variable:

### Example

$cars = array("Volvo", "BMW", "Toyota");

## What is an Array?

An array is a special variable that can hold many values under a single name, and you can access the values by referring to an index number or name.

## PHP Array Types

In PHP, there are three types of arrays:

* [**Indexed arrays**](https://www.w3schools.com/php/php_arrays_indexed.asp) - Arrays with a numeric index
* [**Associative arrays**](https://www.w3schools.com/php/php_arrays_associative.asp) - Arrays with named keys
* [**Multidimensional arrays**](https://www.w3schools.com/php/php_arrays_multidimensional.asp) - Arrays containing one or more arrays

## Working With Arrays

In this tutorial you will learn how to work with arrays, including:

* [Create Arrays](https://www.w3schools.com/php/php_array_create.asp)
* [Access Arrays](https://www.w3schools.com/php/php_array_access.asp)
* [Update Arrays](https://www.w3schools.com/php/php_array_update.asp)
* [Remove Array Items](https://www.w3schools.com/php/php_array_remove.asp)
* [Sort Arrays](https://www.w3schools.com/php/php_array_sort.asp)

## Array Items

Array items can be of any data type.

The most common are strings and numbers (int, float), but array items can also be objects, functions or even arrays.

You can have different data types in the same array.

### Example

Array items of four different data types:

$myArr = array("Volvo", 15, ["apples", "bananas"], myFunction);

## Array Functions

The real strength of PHP arrays are the built-in array functions, like the count() function for counting array items:

### Example

How many items are in the $cars array:

$cars = array("Volvo", "BMW", "Toyota");

echo count($cars);

# PHP Indexed Arrays

## PHP Indexed Arrays

In indexed arrays each item has an index number.

By default, the first item has index 0, the second item has item 1, etc.

### Example

Create and display an indexed array:

$cars = array("Volvo", "BMW", "Toyota");

var\_dump($cars);

## Access Indexed Arrays

To access an array item you can refer to the index number.

### Example

Display the first array item:

$cars = array("Volvo", "BMW", "Toyota");

echo $cars[0];

## Change Value

To change the value of an array item, use the index number:

### Example

Change the value of the second item:

$cars = array("Volvo", "BMW", "Toyota");

$cars[1] = "Ford";

var\_dump($cars);

## Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a foreach loop, like this:

### Example

Display all array items:

$cars = array("Volvo", "BMW", "Toyota");

foreach ($cars as $x) {

echo "$x <br>";

}

For a complete reference of all array functions, go to our complete [PHP Array Reference](https://www.w3schools.com/php/php_ref_array.asp).

## Index Number

The key of an indexed array is a number, by default the first item is 0 and the second is 1 etc., but there are exceptions.

New items get the next index number, meaning one higher than the highest existing index.

So if you have an array like this:

$cars[0] = "Volvo";

$cars[1] = "BMW";

$cars[2] = "Toyota";

And if you use the array\_push() function to add a new item, the new item will get the index 3:

### Example

array\_push($cars, "Ford");

var\_dump($cars);

But if you have an array with random index numbers, like this:

$cars[5] = "Volvo";

$cars[7] = "BMW";

$cars[14] = "Toyota";

And if you use the array\_push() function to add a new item, what will be the index number of the new item?

### Example

array\_push($cars, "Ford");

var\_dump($cars);

# PHP Associative Arrays

## PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

### Example

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

var\_dump($car);

## Access Associative Arrays

To access an array item you can refer to the key name.

### Example

Display the model of the car:

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

echo $car["model"];

## Change Value

To change the value of an array item, use the key name:

### Example

Change the year item:

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

$car["year"] = 2024;

var\_dump($car);

## Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

### Example

Display all array items, keys and values:

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

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

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

}

# PHP - $\_SERVER

## $\_SERVER

$\_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in $\_SERVER:

### Example

echo $\_SERVER['PHP\_SELF'];

echo $\_SERVER['SERVER\_NAME'];

echo $\_SERVER['HTTP\_HOST'];

echo $\_SERVER['HTTP\_REFERER'];

echo $\_SERVER['HTTP\_USER\_AGENT'];

echo $\_SERVER['SCRIPT\_NAME'];

The following table lists the most important elements that can go inside $\_SERVER:

|  |  |
| --- | --- |
| **Element/Code** | **Description** |
| $\_SERVER['PHP\_SELF'] | Returns the filename of the currently executing script |
| $\_SERVER['GATEWAY\_INTERFACE'] | Returns the version of the Common Gateway Interface (CGI) the server is using |
| $\_SERVER['SERVER\_ADDR'] | Returns the IP address of the host server |
| $\_SERVER['SERVER\_NAME'] | Returns the name of the host server (such as www.w3schools.com) |
| $\_SERVER['SERVER\_SOFTWARE'] | Returns the server identification string (such as Apache/2.2.24) |
| $\_SERVER['SERVER\_PROTOCOL'] | Returns the name and revision of the information protocol (such as HTTP/1.1) |
| $\_SERVER['REQUEST\_METHOD'] | Returns the request method used to access the page (such as POST) |
| $\_SERVER['REQUEST\_TIME'] | Returns the timestamp of the start of the request (such as 1377687496) |
| $\_SERVER['QUERY\_STRING'] | Returns the query string if the page is accessed via a query string |
| $\_SERVER['HTTP\_ACCEPT'] | Returns the Accept header from the current request |
| $\_SERVER['HTTP\_ACCEPT\_CHARSET'] | Returns the Accept\_Charset header from the current request (such as utf-8,ISO-8859-1) |
| $\_SERVER['HTTP\_HOST'] | Returns the Host header from the current request |
| $\_SERVER['HTTP\_REFERER'] | Returns the complete URL of the current page (not reliable because not all user-agents support it) |
| $\_SERVER['HTTPS'] | Is the script queried through a secure HTTP protocol |
| $\_SERVER['REMOTE\_ADDR'] | Returns the IP address from where the user is viewing the current page |
| $\_SERVER['REMOTE\_HOST'] | Returns the Host name from where the user is viewing the current page |
| $\_SERVER['REMOTE\_PORT'] | Returns the port being used on the user's machine to communicate with the web server |
| $\_SERVER['SCRIPT\_FILENAME'] | Returns the absolute pathname of the currently executing script |
| $\_SERVER['SERVER\_ADMIN'] | Returns the value given to the SERVER\_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com) |
| $\_SERVER['SERVER\_PORT'] | Returns the port on the server machine being used by the web server for communication (such as 80) |
| $\_SERVER['SERVER\_SIGNATURE'] | Returns the server version and virtual host name which are added to server-generated pages |
| $\_SERVER['PATH\_TRANSLATED'] | Returns the file system based path to the current script |
| $\_SERVER['SCRIPT\_NAME'] | Returns the path of the current script |
| $\_SERVER['SCRIPT\_URI'] | Returns the URI of the current page |

# PHP - $\_REQUEST

## $\_REQUEST

$\_REQUEST is a PHP super global variable which contains submitted form data, and all cookie data.

In other words, $\_REQUEST is an array containing data from [$\_GET](https://www.w3schools.com/php/php_superglobals_get.asp), [$\_POST](https://www.w3schools.com/php/php_superglobals_post.asp), and [$\_COOKIE](https://www.w3schools.com/php/php_cookiest.asp).

You can access this data with the $\_REQUEST keyword followed by the name of the form field, or cookie, like this:

$\_REQUEST['firstname']

## Using $\_REQUEST on $\_POST Requests

POST request are usually data submitted from an HTML form.

Here is an example of how a HTML form could look like:

HTML form

<html>

<body>

<form method="post" action="demo\_request.php">

Name: <input type="text" name="fname">

<input type="submit">

</form>

</body>

</html>

When a user clicks the submit button, the form data is sent to a PHP file specified in the action attribute of the <form> tag.

In the action file we can use the $\_REQUEST variable to collect the value of the input field.

PHP file

$name = $\_REQUEST['fname'];

echo $name;

In the example below we have put the HTML form and PHP code in the same PHP file.

We have also added some extra lines for security.

### Example

<html>

<body>

<form method="post" action="**<?php** echo $\_SERVER['PHP\_SELF'];**?>**">

Name: <input type="text" name="fname">

<input type="submit">

</form>

**<?php**

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

$name = htmlspecialchars($\_REQUEST['fname']);

if (empty($name)) {

echo "Name is empty";

} else {

echo $name;

}

}

**?>**

</body>

</html>

## Using $\_REQUEST on $\_GET Requests

GET request can be form submissions as in the example above, with the method attribute of the HTML <form> element set to GET.

GET requests can also be data from a query string (information added after a URL address).

Here is an example of how an HTML hyperlink, with a query string could look like:

HTML link

<html>

<body>

<a href="demo\_phpfile.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>

</html>

When a user clicks the link, the query string data is sent to demo\_phpfile.php.

In the PHP file we can use the $\_REQUEST variable to collect the value of the query string.

### Example

The PHP file demo\_phpfile.php:

<html>

<body>

**<?php**

echo "Study " . $\_REQUEST['subject'] . " at " . $\_REQUEST['web'];

**?>**

</body>

</html>