

http://www.w3schools.com/php/php_intro.asp

PHP 5 Introduction

[« Previous](#)

[Next Chapter »](#)

PHP scripts are executed on the server.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- CSS
- JavaScript

If you want to study these subjects first, find the tutorials on our [Home page](#).

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

PHP is an amazing and popular language!

It is powerful enough to be at the core of the biggest blogging system on the web (WordPress).
It is deep enough to run the largest social network (Facebook)!

It is also easy enough to be a beginner's first server side language!

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
 - PHP code are executed on the server, and the result is returned to the browser as plain HTML
 - PHP files have extension ".php"
-

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

PHP 5 Syntax

[« Previous](#)

[Next Chapter »](#)

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with **<?php** and ends with **?>**:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php".

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>
```

```
<?php
echo "Hello World!";
?>

</body>
</html>
```

[Run example »](#)

Note: PHP statements end with a semicolon (;).

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand what you are doing
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
// This is a single-line comment

# This is also a single-line comment
```

```
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/

// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>

</body>
</html>
```

Run example »

PHP Case Sensitivity

In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.

In the example below, all three echo statements below are legal (and equal):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>
```

[Run example »](#)

However; all variable names are case-sensitive.

In the example below, only the first statement will display the value of the \$color variable (this is because \$color, \$COLOR, and \$coLOR are treated as three different variables):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>

</body>
</html>
```

[Run example »](#)

PHP 5 Variables

[« Previous](#)

[Next Chapter »](#)

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the \$ sign, followed by the name of the variable:

Example

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

Run example »

After the execution of the statements above, the variable **\$txt** will hold the value **Hello world!**, the variable **\$x** will hold the value **5**, and the variable **\$y** will hold the value **10.5**.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Think of variables as containers for storing data.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

- Variable names are case-sensitive (\$age and \$AGE are two different variables)

Remember that PHP variable names are case-sensitive!

Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example

```
<?php
$txt = "W3Schools.com";
echo "I love $txt!";
?>
```

[Run example »](#)

The following example will produce the same output as the example above:

Example

```
<?php
$txt = "W3Schools.com";
echo "I love " . $txt . "!";
?>
```

[Run example »](#)

The following example will output the sum of two variables:

Example


```
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
```

[Run example »](#)

Note: You will learn more about the echo statement and how to output data to the screen in the next chapter.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
 - global
 - static
-

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

```
<?php
$x = 5; // global scope

function myTest() {
    // using x inside this function will generate an error
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

[Run example »](#)

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Example

```
<?php
function myTest() {
    $x = 5; // local scope
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

[Run example »](#)

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
    global $x, $y;
    $y = $x + $y;
}

myTest();
echo $y; // outputs 15
?>
```

[Run example »](#)

PHP also stores all global variables in an array called `$GLOBALS[index]`. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

[Run example »](#)

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the **static** keyword when you first declare the variable:

Example

```
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}

myTest();
myTest();
myTest();
?>
```

[Run example »](#)

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function

PHP 5 echo and print Statements

[« Previous](#)

[Next Chapter »](#)

In PHP there are two basic ways to get output: echo and print.

In this tutorial we use echo (and print) in almost every example. So, this chapter contains a little more info about those two output statements.

PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

Example

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple
parameters.";
?>
```

Run example »

Display Variables

The following example shows how to output text and variables with the echo statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>$txt1</h2>";
echo "Study PHP at $txt2<br>";
echo $x + $y;
?>
```

Run example »

The PHP print Statement

The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

Run example »

Display Variables

The following example shows how to output text and variables with the print statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>$txt1</h2>";
print "Study PHP at $txt2<br>";
print $x + $y;
?>
```

Run example »

PHP 5 Data Types

[« Previous](#)

[Next Chapter »](#)

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

Example

```
<?php
$x = "Hello world!";
$y = 'Hello world!';
```



```
echo $x;  
echo "<br>";  
echo $y;  
?>
```

[Run example »](#)

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example \$x is an integer. The PHP var_dump() function returns the data type and value:

Example

```
<?php  
$x = 5985;  
var_dump($x);  
?>
```

[Run example »](#)

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example \$x is a float. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$x = 10.365;
var_dump($x);
?>
```

[Run example »](#)

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

An array stores multiple values in one single variable.

In the following example \$cars is an array. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
var_dump($cars);
?>
```

[Run example »](#)

You will learn a lot more about arrays in later chapters of this tutorial.

PHP Object

An object is a data type which stores data and information on how to process that data.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
    function Car() {
        $this->model = "VW";
    }
}

// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

[Run example »](#)

You will learn more about objects in a later chapter of this tutorial.

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

Tip: If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

Example

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

Run example »

PHP 5 Strings

[« Previous](#)

[Next Chapter »](#)

A string is a sequence of characters, like "Hello world!".

PHP String Functions

In this chapter we will look at some commonly used functions to manipulate strings.

Get The Length of a String

The PHP `strlen()` function returns the length of a string.

The example below returns the length of the string "Hello world!":

Example

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

Run example »

The output of the code above will be: 12.

Count The Number of Words in a String

The PHP `str_word_count()` function counts the number of words in a string:

Example

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

Run example »

The output of the code above will be: 2.

Reverse a String

The PHP `strrev()` function reverses a string:

Example

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

Run example »

The output of the code above will be: !dlrow olleH.

Search For a Specific Text Within a String

The PHP `strpos()` function searches for a specific text within a string.

If a match is found, the function returns the character position of the first match. If no match is found, it will return `FALSE`.

The example below searches for the text "world" in the string "Hello world!":

Example

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

Run example »

The output of the code above will be: 6.

Tip: The first character position in a string is 0 (not 1).

Replace Text Within a String

The PHP `str_replace()` function replaces some characters with some other characters in a string.

The example below replaces the text "world" with "Dolly":

Example

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs
Hello Dolly!
?>
```

Run example »

The output of the code above will be: Hello Dolly!

Complete PHP String Reference

For a complete reference of all string functions, go to our complete [PHP String Reference](#).

The PHP string reference contains description and example of use, for each function!

PHP 5 Constants

[« Previous](#)

[Next Chapter »](#)

Constants are like variables except that once they are defined they cannot be changed or undefined.

PHP Constants

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no \$ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

Create a PHP Constant

To create a constant, use the `define()` function.

Syntax

```
define(name, value, case-insensitive)
```

Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

The example below creates a constant with a **case-sensitive** name:

Example


```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

[Run example »](#)

The example below creates a constant with a **case-insensitive** name:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>
```

[Run example »](#)

Constants are Global

Constants are automatically global and can be used across the entire script.

The example below uses a constant inside a function, even if it is defined outside the function:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");

function myTest() {
    echo GREETING;
}

myTest();
?>
```

[Run example »](#)

PHP 5 Operators

[« Previous](#)[Next Chapter »](#)

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result	S
+	Addition	$\$x + \y	Sum of $\$x$ and $\$y$	S
-	Subtraction	$\$x - \y	Difference of $\$x$ and $\$y$	S
*	Multiplication	$\$x * \y	Product of $\$x$ and $\$y$	S

/	Division	$\$x / \y	Quotient of $\$x$ and $\$y$	S
%	Modulus	$\$x \% \y	Remainder of $\$x$ divided by $\$y$	S
**	Exponentiation	$\$x ** \y	Result of raising $\$x$ to the $\$y$ 'th power (Introduced in PHP 5.6)	

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description	S
$x = y$	$x = y$	The left operand gets set to the value of the expression on the right	S
$x += y$	$x = x + y$	Addition	S
$x -= y$	$x = x - y$	Subtraction	S
$x *= y$	$x = x * y$	Multiplication	S
$x /= y$	$x = x / y$	Division	S
$x \% = y$	$x = x \% y$	Modulus	S

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result	S
----------	------	---------	--------	---

==	Equal	<code>\$x == \$y</code>	Returns true if <code>\$x</code> is equal to <code>\$y</code>
===	Identical	<code>\$x === \$y</code>	Returns true if <code>\$x</code> is equal to <code>\$y</code> , and they are of the same type
!=	Not equal	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<>	Not equal	<code>\$x <> \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
!==	Not identical	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code> , or they are not of the same type
>	Greater than	<code>\$x > \$y</code>	Returns true if <code>\$x</code> is greater than <code>\$y</code>
<	Less than	<code>\$x < \$y</code>	Returns true if <code>\$x</code> is less than <code>\$y</code>
>=	Greater than or equal to	<code>\$x >= \$y</code>	Returns true if <code>\$x</code> is greater than or equal to <code>\$y</code>
<=	Less than or equal to	<code>\$x <= \$y</code>	Returns true if <code>\$x</code> is less than or equal to <code>\$y</code>

PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
<code>++\$x</code>	Pre-increment	Increments <code>\$x</code> by one, then returns <code>\$x</code>
<code>\$x++</code>	Post-increment	Returns <code>\$x</code> , then increments <code>\$x</code> by one
<code>--\$x</code>	Pre-decrement	Decrements <code>\$x</code> by one, then returns <code>\$x</code>
<code>\$x--</code>	Post-decrement	Returns <code>\$x</code> , then decrements <code>\$x</code> by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result	S
and	And	\$x and \$y	True if both \$x and \$y are true	S
or	Or	\$x or \$y	True if either \$x or \$y is true	S
xor	Xor	\$x xor \$y	True if either \$x or \$y is true, but not both	S
&&	And	\$x && \$y	True if both \$x and \$y are true	S
	Or	\$x \$y	True if either \$x or \$y is true	S
!	Not	!\$x	True if \$x is not true	S

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result	S
.	Concatenation	\$txt1 . \$txt2	Concatenation of \$txt1 and \$txt2	S
.=	Concatenation assignment	\$txt1 .= \$txt2	Appends \$txt2 to \$txt1	S

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result	S
+	Union	\$x + \$y	Union of \$x and \$y	S
==	Equality	\$x == \$y	Returns true if \$x and \$y have the	S

same key/value pairs			
===	Identity	<code>\$x === \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs in the same order and of the same types
!=	Inequality	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<>	Inequality	<code>\$x <> \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
!==	Non-identity	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not identical to <code>\$y</code>

PHP 5 if...else...elseif Statements

[« Previous](#)

[Next Chapter »](#)

Conditional statements are used to perform different actions based on different conditions.

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif...else statement** - executes different codes for more than two conditions
- **switch statement** - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement executes some code if one condition is true.

Syntax

```
if (condition) {  
    code to be executed if condition is true;  
}
```

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<?php  
$t = date("H");  
  
if ($t < "20") {  
    echo "Have a good day!";  
}  
?>
```

[Run example »](#)

PHP - The if...else Statement

The if....else statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {  
    code to be executed if condition is true;  
} else {  
    code to be executed if condition is false;  
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?php  
$t = date("H");  
  
if ($t < "20") {  
    echo "Have a good day!";  
} else {  
    echo "Have a good night!";  
}  
?>
```

[Run example »](#)

PHP - The if...elseif....else Statement

The if....elseif...else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {  
    code to be executed if this condition is true;  
} elseif (condition) {  
    code to be executed if this condition is true;  
} else {
```



```
    code to be executed if all conditions are false;  
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php  
$t = date("H");  
  
if ($t < "10") {  
    echo "Have a good morning!";  
} elseif ($t < "20") {  
    echo "Have a good day!";  
} else {  
    echo "Have a good night!";  
}  
?>
```

[Run example »](#)

PHP - The switch Statement

The switch statement will be explained in the next chapter.

PHP 5 switch Statement

[« Previous](#)

[Next Chapter »](#)

The switch statement is used to perform different actions based on different conditions.

The PHP switch Statement

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {  
    case label1:  
        code to be executed if n=label1;  
        break;  
    case label2:  
        code to be executed if n=label2;  
        break;  
    case label3:  
        code to be executed if n=label3;  
        break;  
    ...  
    default:  
        code to be executed if n is different from all labels;  
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

Example

```
<?php  
$favcolor = "red";
```

```
switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

[Run example »](#)

PHP 5 while Loops

[« Previous](#)

[Next Chapter »](#)

PHP while loops execute a block of code while the specified condition is true.

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **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 PHP while Loop

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

Syntax

```
while (condition is true) {  
    code to be executed;  
}
```

The example below first sets a variable \$x to 1 (\$x = 1). Then, the while loop will continue to run as long as \$x is less than, or equal to 5 (\$x <= 5). \$x will increase by 1 each time the loop runs (\$x++):

Example

```
<?php  
$x = 1;  
  
while($x <= 5) {  
    echo "The number is: $x <br>";  
    $x++;  
}  
?>
```

[Run example »](#)

The PHP do...while Loop

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

Syntax

```
do {  
    code to be executed;  
} while (condition is true);
```

The example below first sets a variable \$x to 1 (\$x = 1). Then, the do while loop will write some output, and then increment the variable \$x with 1. Then the condition is checked (is \$x less than, or equal to 5?), and the loop will continue to run as long as \$x is less than, or equal to 5:

Example

```
<?php  
$x = 1;  
  
do {  
    echo "The number is: $x <br>";  
    $x++;  
} while ($x <= 5);  
?>
```

[Run example »](#)

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

The example below sets the \$x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

```
<?php  
$x = 6;
```

```
do {  
    echo "The number is: $x <br>";  
    $x++;  
} while ($x <= 5);  
?>
```

[Run example »](#)

The for loop and the foreach loop will be explained in the next chapter.

PHP 5 for Loops

[« Previous](#)

[Next Chapter »](#)

PHP for loops execute a block of code a specified number of times.

The PHP for Loop

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

Syntax

```
for (init counter; test counter; increment counter) {  
    code to be executed;  
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.

- *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 10:

Example

```
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
?>
```

Run example »

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
    code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to `$value` and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array (`$colors`):

Example

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}
?>
```

[Run example »](#)

You will learn more about arrays in a later chapter.

PHP 5 Functions

[« Previous](#)

[Next Chapter »](#)

The real power of PHP comes from its functions; it has more than 1000 built-in functions.

PHP User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName() {  
    code to be executed;  
}
```

Note: A function name can start with a letter or underscore (not a number).

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

Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". 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!". To call the function, just write its name:

Example

```
<?php  
function writeMsg() {  
    echo "Hello world!";  
}  
  
writeMsg(); // call the function  
?>
```

Run example »

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

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

[Run example »](#)

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

Example

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

[Run example »](#)

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

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

[Run example »](#)

PHP Functions - Returning values

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

Example

```
<?php
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);  
?>
```

[Run example »](#)

PHP 5 Arrays

[« Previous](#)

[Next Chapter »](#)

An array stores multiple values in one single variable:

Example

```
<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo "I like " . $cars[0] . ", " . $cars[1] . " and " .  
$cars[2] . ".";  
?>
```

[Run example »](#)

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";  
$cars2 = "BMW";  
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

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

Create an Array in PHP

In PHP, the `array()` function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

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

or the index can be assigned manually:

```
$cars[0] = "Volvo";  
$cars[1] = "BMW";  
$cars[2] = "Toyota";
```

The following example creates an indexed array named `$cars`, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo "I like " . $cars[0] . ", " . $cars[1] . " and " .  
$cars[2] . ".";  
?>
```

[Run example »](#)

Get The Length of an Array - The `count()` Function

The `count()` function is used to return the length (the number of elements) of an array:

Example

```
<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo count($cars);  
?>
```

[Run example »](#)

Loop Through an Indexed Array

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

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>
```

Run example »

PHP Associative Arrays

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

There are two ways to create an associative array:

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

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

Run example »

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

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

Run example »

Multidimensional Arrays

[Multidimensional arrays](#) will be explained in the PHP advanced section.

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete [PHP Array Reference](#).

The reference contains a brief description, and examples of use, for each function!

PHP 5 Sorting Arrays

[« Previous](#)

[Next Chapter »](#)

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- `sort()` - sort arrays in ascending order
 - `rsort()` - sort arrays in descending order
 - `asort()` - sort associative arrays in ascending order, according to the value
 - `ksort()` - sort associative arrays in ascending order, according to the key
 - `arsort()` - sort associative arrays in descending order, according to the value
 - `krsort()` - sort associative arrays in descending order, according to the key
-

Sort Array in Ascending Order - `sort()`

The following example sorts the elements of the \$cars array in ascending alphabetical order:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
?>
```

[Run example »](#)

The following example sorts the elements of the \$numbers array in ascending numerical order:

Example

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

[Run example »](#)

Sort Array in Descending Order - rsort()

The following example sorts the elements of the \$cars array in descending alphabetical order:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
```

```
rsort($cars);  
?>
```

[Run example »](#)

The following example sorts the elements of the \$numbers array in descending numerical order:

Example

```
<?php  
$numbers = array(4, 6, 2, 22, 11);  
rsort($numbers);  
?>
```

[Run example »](#)

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
asort($age);  
?>
```

[Run example »](#)

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

Run example »

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

Run example »

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```

Run example »

Complete PHP Array Reference

For a complete reference of all array functions, go to our complete [PHP Array Reference](#).

The reference contains a brief description, and examples of use, for each function!

PHP 5 Global Variables - Superglobals

[« Previous](#)

[Next Chapter »](#)

Superglobals were introduced in PHP 4.1.0, and are built-in variables that are always available in all scopes.

PHP Global Variables - Superglobals

Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

- `$GLOBALS`
- `$_SERVER`
- `$_REQUEST`
- `$_POST`
- `$_GET`
- `$_FILES`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`

This chapter will explain some of the superglobals, and the rest will be explained in later chapters.

PHP `$GLOBALS`

`$GLOBALS` is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called `$GLOBALS[index]`. The index holds the name of the variable.

The example below shows how to use the super global variable `$GLOBALS`:

Example

```
<?php
$x = 75;
$y = 25;

function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

[Run example »](#)

In the example above, since z is a variable present within the \$GLOBALS array, it is also accessible from outside the function!

PHP \$_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

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

[Run example »](#)

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

Element/Code	Description
<code>\$_SERVER['PHP_SELF']</code>	Returns the filename of the currently executing script
<code>\$_SERVER['GATEWAY_INTERFACE']</code>	Returns the version of the Common Gateway Interface that the server is using
<code>\$_SERVER['SERVER_ADDR']</code>	Returns the IP address of the host server
<code>\$_SERVER['SERVER_NAME']</code>	Returns the name of the host server (such as <code>www.w3schools.com</code>)
<code>\$_SERVER['SERVER_SOFTWARE']</code>	Returns the server identification string (such as <code>Apache/2.2.24</code>)
<code>\$_SERVER['SERVER_PROTOCOL']</code>	Returns the name and revision of the information protocol (such as <code>HTTP/1.1</code>)
<code>\$_SERVER['REQUEST_METHOD']</code>	Returns the request method used to access the page (GET, POST)
<code>\$_SERVER['REQUEST_TIME']</code>	Returns the timestamp of the start of the request (such as <code>1377687496</code>)
<code>\$_SERVER['QUERY_STRING']</code>	Returns the query string if the page is accessed via a GET method (string)
<code>\$_SERVER['HTTP_ACCEPT']</code>	Returns the Accept header from the current request
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	Returns the Accept_Charset header from the current request (such as <code>utf-8,ISO-8859-1</code>)
<code>\$_SERVER['HTTP_HOST']</code>	Returns the Host header from the current request
<code>\$_SERVER['HTTP_REFERER']</code>	Returns the complete URL of the current page (not always set because not all user-agents support it)
<code>\$_SERVER['HTTPS']</code>	Is the script queried through a secure HTTP protocol?
<code>\$_SERVER['REMOTE_ADDR']</code>	Returns the IP address from where the user is viewing the page

	current page
<code>\$_SERVER['REMOTE_HOST']</code>	Returns the Host name from where the user is viewing the current page
<code>\$_SERVER['REMOTE_PORT']</code>	Returns the port being used on the user's machine to communicate with the web server
<code>\$_SERVER['SCRIPT_FILENAME']</code>	Returns the absolute pathname of the currently executing script
<code>\$_SERVER['SERVER_ADMIN']</code>	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, e.g. someone@w3schools.com)
<code>\$_SERVER['SERVER_PORT']</code>	Returns the port on the server machine being used by the server for communication (such as 80)
<code>\$_SERVER['SERVER_SIGNATURE']</code>	Returns the server version and virtual host name added to server-generated pages
<code>\$_SERVER['PATH_TRANSLATED']</code>	Returns the file system based path to the current script
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script
<code>\$_SERVER['SCRIPT_URI']</code>	Returns the URI of the current page

PHP \$_REQUEST

PHP `$_REQUEST` is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the `<form>` tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable `$_REQUEST` to collect the value of the input field:

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") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

[Run example »](#)

PHP \$_POST

PHP \$_POST is widely used to collect form data after submitting an HTML form with method="post". \$_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice.

Then, we can use the super global variable `$_POST` to collect the value of the input field:

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") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

[Run example »](#)

PHP \$_GET

PHP `$_GET` can also be used to collect form data after submitting an HTML form with `method="get"`.

`$_GET` can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=w3schools.com">Test
$GET</a>

</body>
</html>
```

When a user clicks on the link "Test \$GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with `$_GET`.

The example below shows the code in "test_get.php":

Example

```
<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
</html>
```

[Run example »](#)

Tip: You will learn more about `$_POST` and `$_GET` in the [PHP Forms](#) chapter.