PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.

PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

PHP 7 is the latest stable release.

[Start learning PHP now »](https://www.w3schools.com/php/php_intro.asp)

## **What You Should Already Know**

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

* [HTML](https://www.w3schools.com/html/default.asp)
* [CSS](https://www.w3schools.com/css/default.asp)
* [JavaScript](https://www.w3schools.com/js/default.asp)

If you want to study these subjects first, find the tutorials on our [Home page](https://www.w3schools.com/default.asp).

## **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 is 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](http://www.php.net/)
* PHP is easy to learn and runs efficiently on the server side

## **What's new in PHP 7**

* PHP 7 is much faster than the previous popular stable release (PHP 5.6)
* PHP 7 has improved Error Handling
* PHP 7 supports stricter Type Declarations for function arguments
* PHP 7 supports new operators (like the spaceship operator: <=>)

PHP Syntax

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

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:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My first PHP page</h1>  
  
<?php  
echo "Hello World!";  
?>  
  
</body>  
</html>

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

## **PHP Case Sensitivity**

In PHP, 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 equal and legal:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
ECHO "Hello World!<br>";  
echo "Hello World!<br>";  
EcHo "Hello World!<br>";  
?>  
  
</body>  
</html>

**Note:** However; all variable names are case-sensitive!

Look at 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:

<!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>

# PHP Comments

## **Comments in PHP**

A comment in PHP code is a line that is not executed as a 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 your code
* 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**

Syntax for single-line comments:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// This is a single-line comment  
  
# This is also a single-line comment  
?>  
  
</body>  
</html>

### **Example**

Syntax for multiple-line comments:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
/\*  
This is a multiple-lines comment block  
that spans over multiple  
lines  
\*/  
?>  
  
</body>  
</html>

### **Example**

Using comments to leave out parts of the code:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// You can also use comments to leave out parts of a code line  
$x = 5 /\* + 15 \*/ + 5;  
echo $x;  
?>  
  
</body>  
</html>

# PHP Echo

PHP echo is a language construct, not a function. Therefore, you don't need to use parenthesis with it. But if you want to use more than one parameter, it is required to use parenthesis.

ADVERTISEMENT

The syntax of PHP echo is given below:

1. void echo ( string $arg1 [, string $... ] )

PHP echo statement can be used to print the string, multi-line strings, escaping characters, variable, array, etc. Some important points that you must know about the echo statement are:

* echo is a statement, which is used to display the output.
* echo can be used with or without parentheses: echo(), and echo.
* echo does not return any value.
* We can pass multiple strings separated by a comma (,) in echo.
* echo is faster than the print statement.

## **PHP echo: printing string**

*File: echo1.php*

1. **<?php**
2. echo "Hello by PHP echo";
3. **?>**

**Output:**

Hello by PHP echo

## **PHP echo: printing multi line string**

ADVERTISEMENT

*File: echo2.php*

1. **<?php**
2. echo "Hello by PHP echo
3. this is multi line
4. text printed by
5. PHP echo statement
6. ";
7. **?>**

**Output:**

Hello by PHP echo this is multi line text printed by PHP echo statement

## **PHP echo: printing escaping characters**

*File: echo3.php*

1. **<?php**
2. echo "Hello escape \"sequence\" characters";
3. **?>**

**Output:**

Hello escape "sequence" characters

## **PHP echo: printing variable value**

*File: echo4.php*

1. **<?php**
2. $msg="Hello JavaTpoint PHP";
3. echo "Message is: $msg";
4. **?>**

**Output:**

Message is: Hello JavaTpoint PHP

# PHP Print

Like PHP echo, PHP print is a language construct, so you don't need to use parenthesis with the argument list. Print statement can be used with or without parentheses: print and print(). Unlike echo, it always returns 1.

ADVERTISEMENT

The syntax of PHP print is given below:

1. int print(string $arg)

PHP print statement can be used to print the string, multi-line strings, escaping characters, variable, array, etc. Some important points that you must know about the echo statement are:

* print is a statement, used as an alternative to echo at many times to display the output.
* print can be used with or without parentheses.
* print always returns an integer value, which is 1.
* Using print, we cannot pass multiple arguments.
* print is slower than the echo statement.

## **PHP print: printing string**

*File: print1.php*

1. **<?php**
2. print "Hello by PHP print ";
3. print ("Hello by PHP print()");
4. **?>**

**Output:**

Hello by PHP print Hello by PHP print()

## **PHP print: printing multi line string**

ADVERTISEMENT

*File: print2.php*

1. **<?php**
2. print "Hello by PHP print
3. this is multi line
4. text printed by
5. PHP print statement
6. ";
7. **?>**

**Output:**

Hello by PHP print this is multi line text printed by PHP print statement

## **PHP print: printing escaping characters**

*File: print3.php*

1. **<?php**
2. print "Hello escape \"sequence\" characters by PHP print";
3. **?>**

**Output:**

Hello escape "sequence" characters by PHP print

## **PHP print: printing variable value**

*File: print4.php*

1. **<?php**
2. $msg="Hello print() in PHP";
3. print "Message is: $msg";
4. **?>**

**Output:**

Message is: Hello print() in PHP

**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.

# PHP echo and print Statements

We frequently use the echo statement to display the output. There are two basic ways to get the output in PHP:

ADVERTISEMENT

* echo
* print

echo and print are language constructs, and they never behave like a function. Therefore, there is no requirement for parentheses. However, both the statements can be used with or without parentheses. We can use these statements to output variables or strings.

## **Difference between echo and print**

### **echo**

* echo is a statement, which is used to display the output.
* echo can be used with or without parentheses.
* echo does not return any value.
* We can pass multiple strings separated by comma (,) in echo.
* echo is faster than print statement.

### **print**

* print is also a statement, used as an alternative to echo at many times to display the output.
* print can be used with or without parentheses.
* print always returns an integer value, which is 1.
* Using print, we cannot pass multiple arguments.
* print is slower than echo statement.

You can see the difference between echo and print statements with the help of the following programs.

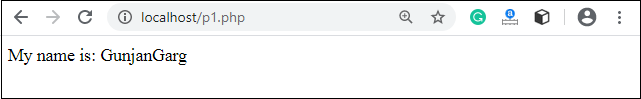
ADVERTISEMENT

### **For Example (Check multiple arguments)**

You can pass multiple arguments separated by a comma (,) in echo. It will not generate any syntax error.

1. **<?php**
2. $fname = "Gunjan";
3. $lname = "Garg";
4. echo "My name is: ".$fname,$lname;
5. **?>**

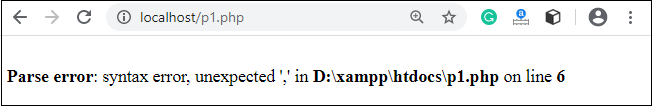
**Output:**



It will generate a **syntax error** because of multiple arguments in a print statement.

1. **<?php**
2. $fname = "Gunjan";
3. $lname = "Garg";
4. print "My name is: ".$fname,$lname;
5. **?>**

**Output:**

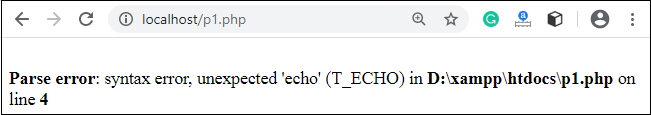


### **For Example (Check Return Value)**

echo statement does not return any value. It will generate an error if you try to display its return value.

1. **<?php**
2. $lang = "PHP";
3. $ret = echo $lang." is a web development language.";
4. echo "**</br>**";
5. echo "Value return by print statement: ".$ret;
6. **?>**

**Output:**

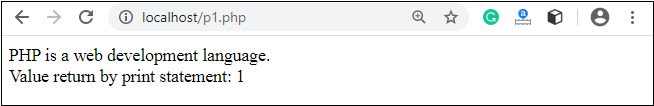


As we already discussed that print returns a value, which is always 1.

ADVERTISEMENT

1. **<?php**
2. $lang = "PHP";
3. $ret = print $lang." is a web development language.";
4. print "**</br>**";
5. print "Value return by print statement: ".$ret;
6. **?>**

**Output:**



# PHP Variables

<="" p="" style="color: rgb(0, 0, 0); font-family: verdana, helvetica, arial, sans-serif; font-size: 14px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: 400; letter-spacing: normal; orphans: 2; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-thickness: initial; text-decoration-style: initial; text-decoration-color: initial;">

In PHP, a variable is declared using a **$ sign** followed by the variable name. Here, some important points to know about variables:

ADVERTISEMENT

* As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyzes the values and makes conversions to its correct datatype.
* After declaring a variable, it can be reused throughout the code.
* Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

1. $variablename=value;

Rules for declaring PHP variable:

* A variable must start with a dollar ($) sign, followed by the variable name.
* It can only contain alpha-numeric character and underscore (A-z, 0-9, \_).
* A variable name must start with a letter or underscore (\_) character.
* A PHP variable name cannot contain spaces.
* One thing to be kept in mind that the variable name cannot start with a number or special symbols.
* PHP variables are case-sensitive, so $name and $NAME both are treated as different variable.

## **PHP Variable: Declaring string, integer, and float**

Let's see the example to store string, integer, and float values in PHP variables.

*File: variable1.php*

1. **<?php**
2. $str="hello string";
3. $x=200;
4. $y=44.6;
5. echo "string is: $str **<br/>**";
6. echo "integer is: $x **<br/>**";
7. echo "float is: $y **<br/>**";
8. **?>**

**Output:**

ADVERTISEMENT

string is: hello string

integer is: 200

float is: 44.6

## **PHP Variable: Sum of two variables**

*File: variable2.php*

1. **<?php**
2. $x=5;
3. $y=6;
4. $z=$x+$y;
5. echo $z;
6. **?>**

**Output:**

11

## **PHP Variable: case sensitive**

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

*File: variable3.php*

1. **<?php**
2. $color="red";
3. echo "My car is " . $color . "**<br>**";
4. echo "My house is " . $COLOR . "**<br>**";
5. echo "My boat is " . $coLOR . "**<br>**";
6. **?>**

**Output:**

My car is red

Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4

My house is

Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5

My boat is

## **PHP Variable: Rules**

PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

ADVERTISEMENT

*File: variablevalid.php*

1. **<?php**
2. $a="hello";//letter (valid)
3. $\_b="hello";//underscore (valid)
5. echo "$a **<br/>** $\_b";
6. **?>**

**Output:**

hello

hello

*File: variableinvalid.php*

1. **<?php**
2. $4c="hello";//number (invalid)
3. $\*d="hello";//special symbol (invalid)
5. echo "$4c **<br/>** $\*d";
6. **?>**

**Output:**

Parse error: syntax error, unexpected '4' (T\_LNUMBER), expecting variable (T\_VARIABLE)

or '$' in C:\wamp\www\variableinvalid.php on line 2

## **PHP: Loosely typed language**

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

# PHP Variable Scope

The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."

ADVERTISEMENT

PHP has three types of variable scopes:

1. Local variable
2. Global variable
3. Static variable

## **Local variable**

The variables that are declared within a function are called local variables for that function. These local variables have their scope only in that particular function in which they are declared. This means that these variables cannot be accessed outside the function, as they have local scope.

A variable declaration outside the function with the same name is completely different from the variable declared inside the function. Let's understand the local variables with the help of an example:

*File: local\_variable1.php*

1. <?php
2. **function** local\_var()
3. {
4. $num = 45;  //local variable
5. echo "Local variable declared inside the function is: ". $num;
6. }
7. local\_var();
8. ?>

**Output:**

Local variable declared inside the function is: 45

*File: local\_variable2.php*

1. <?php
2. **function** mytest()
3. {
4. $lang = "PHP";
5. echo "Web development language: " .$lang;
6. }
7. mytest();
8. //using $lang (local variable) outside the function will generate an error
9. echo $lang;
10. ?>

**Output:**

ADVERTISEMENT

Web development language: PHP

Notice: Undefined variable: lang in D:\xampp\htdocs\program\p3.php on line 28

## **Global variable**

The global variables are the variables that are declared outside the function. These variables can be accessed anywhere in the program. To access the global variable within a function, use the GLOBAL keyword before the variable. However, these variables can be directly accessed or used outside the function without any keyword. Therefore there is no need to use any keyword to access a global variable outside the function.

Let's understand the global variables with the help of an example:

### **Example:**

*File: global\_variable1.php*

1. <?php
2. $name = "Sanaya Sharma";        //Global Variable
3. **function** global\_var()
4. {
5. **global** $name;
6. echo "Variable inside the function: ". $name;
7. echo "</br>";
8. }
9. global\_var();
10. echo "Variable outside the function: ". $name;
11. ?>

**Output:**

Variable inside the function: Sanaya Sharma

Variable outside the function: Sanaya Sharma

#### **Note: Without using the global keyword, if you try to access a global variable inside the function, it will generate an error that the variable is undefined.**

### **Example:**

*File: global\_variable2.php*

1. <?php
2. $name = "Sanaya Sharma";        //global variable
3. **function** global\_var()
4. {
5. echo "Variable inside the function: ". $name;
6. echo "</br>";
7. }
8. global\_var();
9. ?>

**Output:**

ADVERTISEMENT

Notice: Undefined variable: name in D:\xampp\htdocs\program\p3.php on line 6

Variable inside the function:

### **Using $GLOBALS instead of global**

Another way to use the global variable inside the function is predefined $GLOBALS array.

**Example:**

*File: global\_variable3.php*

1. <?php
2. $num1 = 5;      //global variable
3. $num2 = 13;     //global variable
4. **function** global\_var()
5. {
6. $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
7. echo "Sum of global variables is: " .$sum;
8. }
9. global\_var();
10. ?>

**Output:**

Sum of global variables is: 18

If two variables, local and global, have the same name, then the local variable has higher priority than the global variable inside the function.

**Example:**

*File: global\_variable2.php*

1. <?php
2. $x = 5;
3. **function** mytest()
4. {
5. $x = 7;
6. echo "value of x: " .$x;
7. }
8. mytest();
9. ?>

**Output:**

Value of x: 7

#### **Note: local variable has higher priority than the global variable.**

## **Static variable**

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.

ADVERTISEMENT

Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

### **Example:**

*File: static\_variable.php*

1. <?php
2. **function** static\_var()
3. {
4. **static** $num1 = 3;       //static variable
5. $num2 = 6;          //Non-static variable
6. //increment in non-static variable
7. $num1++;
8. //increment in static variable
9. $num2++;
10. echo "Static: " .$num1 ."</br>";
11. echo "Non-static: " .$num2 ."</br>";
12. }
14. //first function call
15. static\_var();
17. //second function call
18. static\_var();
19. ?>

**Output:**

Static: 4

Non-static: 7

Static: 5

Non-static: 7

You have to notice that $num1 regularly increments after each function call, whereas $num2 does not. This is why because $num1 is not a static variable, so it freed its memory after the execution of each function call.

# PHP $ and $$ Variables

The **$var** (single dollar) is a normal variable with the name var that stores any value like string, integer, float, etc.

ADVERTISEMENT

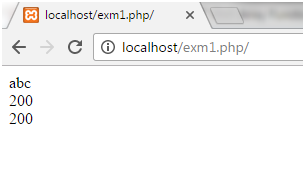
The **$$var** (double dollar) is a reference variable that stores the value of the $variable inside it.

To understand the difference better, let's see some examples.

### **Example 1**

1. <?php
2. $x = "abc";
3. $$x = 200;
4. echo $x."<br/>";
5. echo $$x."<br/>";
6. echo $abc;
7. ?>

**Output:**



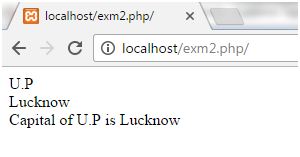
In the above example, we have assigned a value to the variable **x** as **abc**. Value of reference variable **$$x** is assigned as **200**.

Now we have printed the values **$x, $$x** and **$abc**.

### **Example2**

1. <?php
2. $x="U.P";
3. $$x="Lucknow";
4. echo $x. "<br>";
5. echo $$x. "<br>";
6. echo "Capital of $x is " . $$x;
7. ?>

**Output:**



In the above example, we have assigned a value to the variable **x** as **U.P**. Value of reference variable **$$x** is assigned as **Lucknow.**

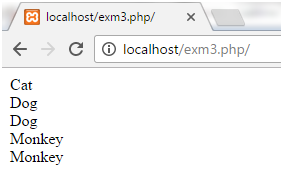
Now we have printed the values **$x, $$x** and a string.

ADVERTISEMENT

### **Example3**

1. <?php
2. $name="Cat";
3. ${$name}="Dog";
4. ${${$name}}="Monkey";
5. echo $name. "<br>";
6. echo ${$name}. "<br>";
7. echo $Cat. "<br>";
8. echo ${${$name}}. "<br>";
9. echo $Dog. "<br>";
10. ?>

**Output:**



In the above example, we have assigned a value to the variable name **Cat**. Value of reference variable **${$name}** is assigned as **Dog** and **${${$name}}** as **Monkey**.

Now we have printed the values as **$name, ${$name}, $Cat, ${${$name}}** and **$Dog.**

# PHP Constants

PHP constants are name or identifier that can't be changed during the execution of the script except for [magic constants](https://www.javatpoint.com/php-magic-constants), which are not really constants. PHP constants can be defined by 2 ways:

ADVERTISEMENT

1. Using define() function
2. Using const keyword

Constants are similar to the variable except once they defined, they can never be undefined or changed. They remain constant across the entire program. PHP constants follow the same PHP variable rules. **For example**, it can be started with a letter or underscore only.

Conventionally, PHP constants should be defined in uppercase letters.

#### **Note: Unlike variables, constants are automatically global throughout the script.**

## **PHP constant: define()**

Use the define() function to create a constant. It defines constant at run time. Let's see the syntax of define() function in PHP.

1. define(name, value, **case**-insensitive)
2. **name:** It specifies the constant name.
3. **value:** It specifies the constant value.
4. **case-insensitive:** Specifies whether a constant is case-insensitive. Default value is false. It means it is case sensitive by default.

Let's see the example to define PHP constant using define().

ADVERTISEMENT

*File: constant1.php*

1. **<?php**
2. define("MESSAGE","Hello JavaTpoint PHP");
3. echo MESSAGE;
4. **?>**

**Output:**

Hello JavaTpoint PHP

Create a constant with **case-insensitive** name:

*File: constant2.php*

1. **<?php**
2. define("MESSAGE","Hello JavaTpoint PHP",true);//not case sensitive
3. echo MESSAGE, "**</br>**";
4. echo message;
5. **?>**

**Output:**

Hello JavaTpoint PHP

Hello JavaTpoint PHP

*File: constant3.php*

1. **<?php**
2. define("MESSAGE","Hello JavaTpoint PHP",false);//case sensitive
3. echo MESSAGE;
4. echo message;
5. **?>**

**Output:**

Hello JavaTpoint PHP

Notice: Use of undefined constant message - assumed 'message'

in C:\wamp\www\vconstant3.php on line 4

message

## **PHP constant: const keyword**

PHP introduced a keyword **const** to create a constant. The const keyword defines constants at compile time. It is a language construct, not a function. The constant defined using const keyword are **case-sensitive**.

*File: constant4.php*

ADVERTISEMENT

1. **<?php**
2. const MESSAGE="Hello const by JavaTpoint PHP";
3. echo MESSAGE;
4. **?>**

**Output:**

Hello const by JavaTpoint PHP

## **Constant() function**

There is another way to print the value of constants using constant() function instead of using the echo statement.

**Syntax**

The syntax for the following constant function:

1. constant (name)

*File: constant5.php*

1. **<?php**
2. define("MSG", "JavaTpoint");
3. echo MSG, "**</br>**";
4. echo constant("MSG");
5. //both are similar
6. **?>**

**Output:**

JavaTpoint

JavaTpoint

| **Constant** | **Variables** |
| --- | --- |
| Once the constant is defined, it can never be redefined. | A variable can be undefined as well as redefined easily. |
| A constant can only be defined using define() function. It cannot be defined by any simple assignment. | A variable can be defined by simple assignment (=) operator. |
| There is no need to use the dollar ($) sign before constant during the assignment. | To declare a variable, always use the dollar ($) sign before the variable. |
| Constants do not follow any variable scoping rules, and they can be defined and accessed anywhere. | Variables can be declared anywhere in the program, but they follow variable scoping rules. |
| Constants are the variables whose values can't be changed throughout the program. | The value of the variable can be changed. |
| By default, constants are global. | Variables can be local, global, or static. |

## **Constant vs Variables**

Next Topic[PHP Magic Constants](https://www.javatpoint.com/php-magic-constants)

# Magic Constants

Magic constants are the predefined constants in PHP which get changed on the basis of their use. They start with double underscore (\_\_) and ends with double underscore.

ADVERTISEMENT

They are similar to other predefined constants but as they change their values with the context, they are called **magic** constants.

There are **nine** magic constants in PHP. In which eight magic constants start and end with double underscores (\_\_).

1. [\_\_LINE\_\_](https://www.javatpoint.com/php-magic-constants#LINE)
2. [\_\_FILE\_\_](https://www.javatpoint.com/php-magic-constants#FILE)
3. [\_\_DIR\_\_](https://www.javatpoint.com/php-magic-constants#DIR)
4. [\_\_FUNCTION\_\_](https://www.javatpoint.com/php-magic-constants#FUNCTION)
5. [\_\_CLASS\_\_](https://www.javatpoint.com/php-magic-constants#CLASS)
6. [\_\_TRAIT\_\_](https://www.javatpoint.com/php-magic-constants#TRAIT)
7. [\_\_METHOD\_\_](https://www.javatpoint.com/php-magic-constants#METHOD)
8. [\_\_NAMESPACE\_\_](https://www.javatpoint.com/php-magic-constants#NAMESPACE)
9. [ClassName::class](https://www.javatpoint.com/php-magic-constants#ClassName)

All of the constants are resolved at compile-time instead of run time, unlike the regular constant. Magic constants are case-insensitive.

## **Changelog**

| **Version** | **Description** |
| --- | --- |
| 5.3.0 | Added **\_\_DIR\_\_** and **\_\_NAMESPACE\_\_** magic constant |
| 5.4.0 | Added **\_\_TRAIT\_\_** magic constant |
| 5.5.0 | Added ::class magic constant |

All the constants are defined below with the example code:

ADVERTISEMENT

### **1. \_\_LINE\_\_**

It returns the current line number of the file, where this constant is used.

**Example:**

1. <?php
2. echo "<h3>Example for \_\_LINE\_\_</h3>";
3. // print Your current line number i.e;4
4. echo "You are at line number " . **\_\_LINE\_\_** . "<br><br>";
5. ?>

**Output:**

### Example for \_\_LINE\_\_

You are at line number 4

### **2. \_\_FILE\_\_:**

This magic constant returns the full path of the executed file, where the file is stored. If it is used inside the include, the name of the included file is returned.

**Example:**

1. <?php
2. echo "<h3>Example for \_\_FILE\_\_</h3>";
3. //print full path of file with .php extension
4. echo **\_\_FILE\_\_** . "<br><br>";
5. ?>

**Output:**

### Example for \_\_FILE\_\_

D:\xampp\htdocs\program\magic.php

### **3. \_\_DIR\_\_:**

It returns the full directory path of the executed file. The path returned by this magic constant is equivalent to dirname(\_\_FILE\_\_). This magic constant does not have a trailing slash unless it is a root directory.

**Example:**

ADVERTISEMENT

1. <?php
2. echo "<h3>Example for \_\_DIR\_\_</h3>";
3. //print full path of directory where script will be placed
4. echo \_\_DIR\_\_ . "<br><br>";
5. //below output will equivalent to above one.
6. echo dirname(**\_\_FILE\_\_**) . "<br><br>";
7. ?>

**Output:**

### Example for \_\_DIR\_\_

D:\xampp\htdocs\program

D:\xampp\htdocs\program

### **4. \_\_FUNCTION\_\_:**

This magic constant returns the function name, where this constant is used. It will return blank if it is used outside of any function.

**Example:**

1. <?php
2. echo "<h3>Example for \_\_FUNCTION\_\_</h3>";
3. //Using magic constant inside function.
4. **function** test(){
5. //print the function name i.e; test.
6. echo 'The function name is '. **\_\_FUNCTION\_\_** . "<br><br>";
7. }
8. test();
10. //Magic constant used outside function gives the blank output.
11. **function** test\_function(){
12. echo 'Hie';
13. }
14. test\_function();
15. //give the blank output.
16. echo  **\_\_FUNCTION\_\_** . "<br><br>";
17. ?>

**Output:**

### Example for \_\_FUNCTION\_\_

The function name is test

Hie

### **5. \_\_CLASS\_\_:**

It returns the class name, where this magic constant is used. \_\_CLASS\_\_ constant also works in traits.

**Example:**

1. <?php
2. echo "<h3>Example for \_\_CLASS\_\_</h3>";
3. **class** JTP
4. {
5. **public** **function** \_\_construct() {
6. ;
7. }
8. **function** getClassName(){
9. //print name of the class JTP.
10. echo **\_\_CLASS\_\_** . "<br><br>";
11. }
12. }
13. $t = **new** JTP;
14. $t->getClassName();
16. //in case of multiple classes
17. **class** base
18. {
19. **function** test\_first(){
20. //will always print parent class which is base here.
21. echo **\_\_CLASS\_\_**;
22. }
23. }
24. **class** child **extends** base
25. {
26. **public** **function** \_\_construct() {
27. ;
28. }
29. }
30. $t = **new** child;
31. $t->test\_first();
32. ?>

**Output:**

ADVERTISEMENT

### Example for \_\_CLASS\_\_

JTP

base

### **6. \_\_TRAIT\_\_:**

This magic constant returns the trait name, where it is used.

**Example:**

1. <?php
2. echo "<h3>Example for \_\_TRAIT\_\_</h3>";
3. trait created\_trait {
4. **function** jtp(){
5. //will print name of the trait i.e; created\_trait
6. echo \_\_TRAIT\_\_;
7. }
8. }
9. **class** Company {
10. **use** created\_trait;
11. }
12. $a = **new** Company;
13. $a->jtp();
14. ?>

**Output:**

### Example for \_\_TRAIT\_\_

created\_trait

### **7. \_\_METHOD\_\_:**

It returns the name of the class method where this magic constant is included. The method name is returned the same as it was declared.

**Example:**

1. <?php
2. echo "<h3>Example for \_\_METHOD\_\_</h3>";
3. **class** method {
4. **public** **function** \_\_construct() {
5. //print method::\_\_construct
6. echo **\_\_METHOD\_\_** . "<br><br>";
7. }
8. **public** **function** meth\_fun(){
9. //print method::meth\_fun
10. echo **\_\_METHOD\_\_**;
11. }
12. }
13. $a = **new** method;
14. $a->meth\_fun();
15. ?>

**Output:**

### Example for \_\_METHOD\_\_

method:: construct

method:: meth\_fun

### **8. \_\_NAMESPACE\_\_:**

ADVERTISEMENT

It returns the current namespace where it is used.

**Example:**

1. <?php
2. echo "<h3>Example for \_\_NAMESPACE\_\_</h3>";
3. **class** name {
4. **public** **function** \_\_construct() {
5. echo 'This line will print on calling namespace.';
6. }
7. }
8. $class\_name = \_\_NAMESPACE\_\_ . '\name';
9. $a = **new** class\_name;
10. ?>

**Output:**

### Example for \_\_NAMESPACE\_\_

This line will print on calling namespace.

### **9. ClassName::class:**

This magic constant does not start and end with the double underscore (\_\_). It returns the fully qualified name of the ClassName. ClassName::class is added in **PHP 5.5.0**. It is useful with namespaced classes.

**Example:**

1. <?php
2. namespace Technical\_Portal;
3. echo "<h3>Example for CLASSNAME::CLASS </h3>";
4. **class** javatpoint {
5. }
6. echo javatpoint::**class**;    //ClassName::class
7. ?>

**Output:**

### Example for ClassName::class

Technical\_Portal\javatpoint

#### **Note: Remember namespace must be the very first statement or after any declare call in the script, otherwise it will generate Fatal error.**

Next Topic[PHP Data Types](https://www.javatpoint.com/php-data-types)

# PHP Data Types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

ADVERTISEMENT

1. Scalar Types (predefined)
2. Compound Types (user-defined)
3. Special Types

## **PHP Data Types: Scalar Types**

It holds only single value. There are 4 scalar data types in PHP.

1. [boolean](https://www.javatpoint.com/php-data-types#boolean)
2. [integer](https://www.javatpoint.com/php-data-types#integer)
3. [float](https://www.javatpoint.com/php-data-types#float)
4. [string](https://www.javatpoint.com/php-data-types#string)

## **PHP Data Types: Compound Types**

It can hold multiple values. There are 2 compound data types in PHP.

1. [array](https://www.javatpoint.com/php-data-types#array)
2. [object](https://www.javatpoint.com/php-data-types#object)

## **PHP Data Types: Special Types**

There are 2 special data types in PHP.

1. [resource](https://www.javatpoint.com/php-data-types#resource)
2. [NULL](https://www.javatpoint.com/php-data-types#NULL)

## **PHP Boolean**

ADVERTISEMENT

Booleans are the simplest data type works like switch. It holds only two values: **TRUE (1)** or **FALSE (0)**. It is often used with conditional statements. If the condition is correct, it returns TRUE otherwise FALSE.

**Example:**

1. <?php
2. **if** (TRUE)
3. echo "This condition is TRUE.";
4. **if** (FALSE)
5. echo "This condition is FALSE.";
6. ?>

**Output:**

This condition is TRUE.

## **PHP Integer**

Integer means numeric data with a negative or positive sign. It holds only whole numbers, i.e., numbers without fractional part or decimal points.

**Rules for integer:**

* An integer can be either positive or negative.
* An integer must not contain decimal point.
* Integer can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
* The range of an integer must be lie between 2,147,483,648 and 2,147,483,647 i.e., -2^31 to 2^31.

**Example:**

1. <?php
2. $dec1 = 34;
3. $oct1 = 0243;
4. $hexa1 = 0x45;
5. echo "Decimal number: " .$dec1. "</br>";
6. echo "Octal number: " .$oct1. "</br>";
7. echo "HexaDecimal number: " .$hexa1. "</br>";
8. ?>

**Output:**

Decimal number: 34

Octal number: 163

HexaDecimal number: 69

## **PHP Float**

ADVERTISEMENT

A floating-point number is a number with a decimal point. Unlike integer, it can hold numbers with a fractional or decimal point, including a negative or positive sign.

**Example:**

1. <?php
2. $n1 = 19.34;
3. $n2 = 54.472;
4. $sum = $n1 + $n2;
5. echo "Addition of floating numbers: " .$sum;
6. ?>

**Output:**

Addition of floating numbers: 73.812

## **PHP String**

A string is a non-numeric data type. It holds letters or any alphabets, numbers, and even special characters.

String values must be enclosed either within **single quotes** or in **double quotes**. But both are treated differently. To clarify this, see the example below:

**Example:**

1. <?php
2. $company = "Javatpoint";
3. //both single and double quote statements will treat different
4. echo "Hello $company";
5. echo "</br>";
6. echo 'Hello $company';
7. ?>

**Output:**

Hello Javatpoint

Hello $company

## **PHP Array**

An array is a compound data type. It can store multiple values of same data type in a single variable.

**Example:**

1. <?php
2. $bikes = **array** ("Royal Enfield", "Yamaha", "KTM");
3. var\_dump($bikes);   //the var\_dump() function returns the datatype and values
4. echo "</br>";
5. echo "Array Element1: $bikes[0] </br>";
6. echo "Array Element2: $bikes[1] </br>";
7. echo "Array Element3: $bikes[2] </br>";
8. ?>

**Output:**

ADVERTISEMENT

array(3) { [0]=> string(13) "Royal Enfield" [1]=> string(6) "Yamaha" [2]=> string(3) "KTM" }

Array Element1: Royal Enfield

Array Element2: Yamaha

Array Element3: KTM

You will learn more about array in later chapters of this tutorial.

## **PHP object**

Objects are the instances of user-defined classes that can store both values and functions. They must be explicitly declared.

**Example:**

1. <?php
2. **class** bike {
3. **function** model() {
4. $model\_name = "Royal Enfield";
5. echo "Bike Model: " .$model\_name;
6. }
7. }
8. $obj = **new** bike();
9. $obj -> model();
10. ?>

**Output:**

Bike Model: Royal Enfield

This is an advanced topic of PHP, which we will discuss later in detail.

## **PHP Resource**

Resources are not the exact data type in PHP. Basically, these are used to store some function calls or references to external PHP resources. **For example** - a database call. It is an external resource.

This is an advanced topic of PHP, so we will discuss it later in detail with examples.

## **PHP Null**

Null is a special data type that has only one value: **NULL**. There is a convention of writing it in capital letters as it is case sensitive.

The special type of data type NULL defined a variable with no value.

**Example:**

ADVERTISEMENT

1. <?php
2. $nl = NULL;
3. echo $nl;   //it will not give any output
4. ?>

**Output:**

Next Topic[PHP Operators](https://www.javatpoint.com/php-operators)

# PHP Operators

PHP Operator is a symbol i.e used to perform operations on operands. In simple words, operators are used to perform operations on variables or values. For example:

ADVERTISEMENT

1. $num=10+20;//+ is the operator and 10,20 are operands

In the above example, + is the binary + operator, 10 and 20 are operands and $num is variable.

PHP Operators can be categorized in following forms:

* [Arithmetic Operators](https://www.javatpoint.com/php-operators#Arithmetic)
* [Assignment Operators](https://www.javatpoint.com/php-operators#Assignment)
* [Bitwise Operators](https://www.javatpoint.com/php-operators#Bitwise)
* [Comparison Operators](https://www.javatpoint.com/php-operators#Comparison)
* [Incrementing/Decrementing Operators](https://www.javatpoint.com/php-operators#Incrementing)
* [Logical Operators](https://www.javatpoint.com/php-operators#Logical)
* [String Operators](https://www.javatpoint.com/php-operators#String)
* [Array Operators](https://www.javatpoint.com/php-operators#Array)
* [Type Operators](https://www.javatpoint.com/php-operators#Type)
* [Execution Operators](https://www.javatpoint.com/php-operators#Execution)
* [Error Control Operators](https://www.javatpoint.com/php-operators#Error)

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

* **Unary Operators:** works on single operands such as ++, -- etc.
* **Binary Operators:** works on two operands such as binary +, -, \*, / etc.
* **Ternary Operators:** works on three operands such as "?:".

## **Arithmetic Operators**

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

ADVERTISEMENT

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| + | Addition | $a + $b | Sum of operands |
| - | Subtraction | $a - $b | Difference of operands |
| \* | Multiplication | $a \* $b | Product of operands |
| / | Division | $a / $b | Quotient of operands |
| % | Modulus | $a % $b | Remainder of operands |
| \*\* | Exponentiation | $a \*\* $b | $a raised to the power $b |

The exponentiation (\*\*) operator has been introduced in PHP 5.6.

## **Assignment Operators**

The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| = | Assign | $a = $b | The value of right operand is assigned to the left operand. |
| += | Add then Assign | $a += $b | Addition same as $a = $a + $b |
| -= | Subtract then Assign | $a -= $b | Subtraction same as $a = $a - $b |
| \*= | Multiply then Assign | $a \*= $b | Multiplication same as $a = $a \* $b |
| /= | Divide then Assign (quotient) | $a /= $b | Find quotient same as $a = $a / $b |
| %= | Divide then Assign (remainder) | $a %= $b | Find remainder same as $a = $a % $b |

## **Bitwise Operators**

The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

ADVERTISEMENT

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| & | And | $a & $b | Bits that are 1 in both $a and $b are set to 1, otherwise 0. |
| | | Or (Inclusive or) | $a | $b | Bits that are 1 in either $a or $b are set to 1 |
| ^ | Xor (Exclusive or) | $a ^ $b | Bits that are 1 in either $a or $b are set to 0. |
| ~ | Not | ~$a | Bits that are 1 set to 0 and bits that are 0 are set to 1 |
| << | Shift left | $a << $b | Left shift the bits of operand $a $b steps |
| >> | Shift right | $a >> $b | Right shift the bits of $a operand by $b number of places |

## **Comparison Operators**

Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given:

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| == | Equal | $a == $b | Return TRUE if $a is equal to $b |
| === | Identical | $a === $b | Return TRUE if $a is equal to $b, and they are of same data type |
| !== | Not identical | $a !== $b | Return TRUE if $a is not equal to $b, and they are not of same data type |
| != | Not equal | $a != $b | Return TRUE if $a is not equal to $b |
| <> | Not equal | $a <> $b | Return TRUE if $a is not equal to $b |
| < | Less than | $a < $b | Return TRUE if $a is less than $b |
| > | Greater than | $a > $b | Return TRUE if $a is greater than $b |
| <= | Less than or equal to | $a <= $b | Return TRUE if $a is less than or equal $b |
| >= | Greater than or equal to | $a >= $b | Return TRUE if $a is greater than or equal $b |
| <=> | Spaceship | $a <=>$b | Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b |

## **Incrementing/Decrementing Operators**

The increment and decrement operators are used to increase and decrease the value of a variable.

ADVERTISEMENT

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| ++ | Increment | ++$a | Increment the value of $a by one, then return $a |
| $a++ | Return $a, then increment the value of $a by one |
| -- | decrement | --$a | Decrement the value of $a by one, then return $a |
| $a-- | Return $a, then decrement the value of $a by one |

## **Logical Operators**

The logical operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| and | And | $a and $b | Return TRUE if both $a and $b are true |
| Or | Or | $a or $b | Return TRUE if either $a or $b is true |
| xor | Xor | $a xor $b | Return TRUE if either $ or $b is true but not both |
| ! | Not | ! $a | Return TRUE if $a is not true |
| && | And | $a && $b | Return TRUE if either $a and $b are true |
| || | Or | $a || $b | Return TRUE if either $a or $b is true |

## **String Operators**

The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below:

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| . | Concatenation | $a . $b | Concatenate both $a and $b |
| .= | Concatenation and Assignment | $a .= $b | First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b |

## **Array Operators**

The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

ADVERTISEMENT

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| + | Union | $a + $y | Union of $a and $b |
| == | Equality | $a == $b | Return TRUE if $a and $b have same key/value pair |
| != | Inequality | $a != $b | Return TRUE if $a is not equal to $b |
| === | Identity | $a === $b | Return TRUE if $a and $b have same key/value pair of same type in same order |
| !== | Non-Identity | $a !== $b | Return TRUE if $a is not identical to $b |
| <> | Inequality | $a <> $b | Return TRUE if $a is not equal to $b |

## **Type Operators**

The type operator **instanceof** is used to determine whether an object, its parent and its derived class are the same type or not. Basically, this operator determines which certain class the object belongs to. It is used in object-oriented programming.

1. **<?php**
2. //class declaration
3. class Developer
4. {}
5. class Programmer
6. {}
7. //creating an object of type Developer
8. $charu = new Developer();
10. //testing the type of object
11. if( $charu instanceof Developer)
12. {
13. echo "Charu is a developer.";
14. }
15. else
16. {
17. echo "Charu is a programmer.";
18. }
19. echo "**</br>**";
20. var\_dump($charu instanceof Developer);           //It will return true.
21. var\_dump($charu instanceof Programmer);       //It will return false.
22. **?>**

**Output:**

Charu is a developer.

bool(true) bool(false)

## **Execution Operators**

PHP has an execution operator **backticks (``)**. PHP executes the content of backticks as a shell command. Execution operator and **shell\_exec()** give the same result.

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| `` | backticks | echo `dir`; | Execute the shell command and return the result. Here, it will show the directories available in current folder. |

#### **Note: Note that backticks (``) are not single-quotes.**

## **Error Control Operators**

PHP has one error control operator, i.e., **at (@) symbol**. Whenever it is used with an expression, any error message will be ignored that might be generated by that expression.

ADVERTISEMENT

| **Operator** | **Name** | **Example** | **Explanation** |
| --- | --- | --- | --- |
| @ | at | @file ('non\_existent\_file') | Intentional file error |

## **PHP Operators Precedence**

Let's see the precedence of PHP operators with associativity.

| **Operators** | **Additional Information** | **Associativity** |
| --- | --- | --- |
| clone new | clone and new | non-associative |
| [ | array() | left |
| \*\* | arithmetic | right |
| ++ -- ~ (int) (float) (string) (array) (object) (bool) @ | increment/decrement and types | right |
| instanceof | types | non-associative |
| ! | logical (negation) | right |
| \* / % | arithmetic | left |
| + - . | arithmetic and string concatenation | left |
| << >> | bitwise (shift) | left |
| < <= > >= | comparison | non-associative |
| == != === !== <> | comparison | non-associative |
| & | bitwise AND | left |
| ^ | bitwise XOR | left |
| | | bitwise OR | left |
| && | logical AND | left |
| || | logical OR | left |
| ?: | ternary | left |
| = += -= \*= \*\*= /= .= %= &= |= ^= <<= >>= => | assignment | right |
| and | logical | left |
| xor | logical | left |
| or | logical | left |
| , | many uses (comma) | left |

Next Topic[PHP Comments](https://www.javatpoint.com/php-comments)

# PHP Comments

PHP comments can be used to describe any line of code so that other developer can understand the code easily. It can also be used to hide any code.

ADVERTISEMENT

PHP supports single line and multi line comments. These comments are similar to C/C++ and Perl style (Unix shell style) comments.

## **PHP Single Line Comments**

There are two ways to use single line comments in PHP.

* // (C++ style single line comment)
* # (Unix Shell style single line comment)

1. <?php
2. // this is C++ style single line comment
3. # this is Unix Shell style single line comment
4. echo "Welcome to PHP single line comments";
5. ?>

**Output:**

Welcome to PHP single line comments

## **PHP Multi Line Comments**

In PHP, we can comments multiple lines also. To do so, we need to enclose all lines within /\* \*/. Let's see a simple example of PHP multiple line comment.

1. <?php
2. /\*
3. Anything placed
4. within comment
5. will not be displayed
6. on the browser;
7. \*/
8. echo "Welcome to PHP multi line comment";
9. ?>

**Output:**

Welcome to PHP multi line comment

Next Topic[PHP If Else](https://www.javatpoint.com/php-if-else)

[← Prev](https://www.javatpoint.com/php-operators)[Next →](https://www.javatpoint.com/php-if-else)

<https://www.javatpoint.com/php-comments>

15-6-2021

# PHP Regular Expressions

[❮ Previous](https://www.w3schools.com/php/php_superglobals_get.asp)[Next ❯](https://www.w3schools.com/php/php_forms.asp)

## **What is a Regular Expression?**

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of text search and text replace operations.

## **Syntax**

In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

$exp = "/w3schools/i";

In the example above, / is the **delimiter**, *w3schools* is the **pattern**that is being searched for, and i is a **modifier**that makes the search case-insensitive.

The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

## **Regular Expression Functions**

PHP provides a variety of functions that allow you to use regular expressions. The preg\_match(), preg\_match\_all() and preg\_replace() functions are some of the most commonly used ones:

| **Function** | **Description** |
| --- | --- |
| preg\_match() | Returns 1 if the pattern was found in the string and 0 if not |
| preg\_match\_all() | Returns the number of times the pattern was found in the string, which may also be 0 |
| preg\_replace() | Returns a new string where matched patterns have been replaced with another string |

## **Using preg\_match()**

The preg\_match() function will tell you whether a string contains matches of a pattern.

### **Example**

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

<?php  
$str = "Visit W3Schools";  
$pattern = "/w3schools/i";  
echo preg\_match($pattern, $str); // Outputs 1  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_regex_match)

## **Using preg\_match\_all()**

The preg\_match\_all() function will tell you how many matches were found for a pattern in a string.

### **Example**

Use a regular expression to do a case-insensitive count of the number of occurrences of "ain" in a string:

<?php  
$str = "The rain in SPAIN falls mainly on the plains.";  
$pattern = "/ain/i";  
echo preg\_match\_all($pattern, $str); // Outputs 4  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_regex_match_all)

## **Using preg\_replace()**

The preg\_replace() function will replace all of the matches of the pattern in a string with another string.

### **Example**

Use a case-insensitive regular expression to replace Microsoft with W3Schools in a string:

<?php  
$str = "Visit Microsoft!";  
$pattern = "/microsoft/i";  
echo preg\_replace($pattern, "W3Schools", $str); // Outputs "Visit W3Schools!"  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_regex_replace)

## **Regular Expression Modifiers**

Modifiers can change how a search is performed.

| **Modifier** | **Description** |
| --- | --- |
| i | Performs a case-insensitive search |
| m | Performs a multiline search (patterns that search for the beginning or end of a string will match the beginning or end of each line) |
| u | Enables correct matching of UTF-8 encoded patterns |

## **Regular Expression Patterns**

Brackets are used to find a range of characters:

| **Expression** | **Description** |
| --- | --- |
| [abc] | Find one character from the options between the brackets |
| [^abc] | Find any character NOT between the brackets |
| [0-9] | Find one character from the range 0 to 9 |

## **Metacharacters**

Metacharacters are characters with a special meaning:

| **Metacharacter** | **Description** |
| --- | --- |
| | | Find a match for any one of the patterns separated by | as in: cat|dog|fish |
| . | Find just one instance of any character |
| ^ | Finds a match as the beginning of a string as in: ^Hello |
| $ | Finds a match at the end of the string as in: World$ |
| \d | Find a digit |
| \s | Find a whitespace character |
| \b | Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b |
| \uxxxx | Find the Unicode character specified by the hexadecimal number xxxx |

## **Quantifiers**

Quantifiers define quantities:

| **Quantifier** | **Description** |
| --- | --- |
| n+ | Matches any string that contains at least one *n* |
| n\* | Matches any string that contains zero or more occurrences of *n* |
| n? | Matches any string that contains zero or one occurrences of *n* |
| n{x} | Matches any string that contains a sequence of *X* *n*'s |
| n{x,y} | Matches any string that contains a sequence of X to Y *n*'s |
| n{x,} | Matches any string that contains a sequence of at least X *n*'s |

**Note:** If your expression needs to search for one of the special characters you can use a backslash ( \ ) to escape them. For example, to search for one or more question marks you can use the following expression: $pattern = '/\?+/';

## **Grouping**

You can use parentheses ( ) to apply quantifiers to entire patterns. They also can be used to select parts of the pattern to be used as a match.

### **Example**

Use grouping to search for the word "banana" by looking for *ba* followed by two instances of *na*:

<?php  
$str = "Apples and bananas.";  
$pattern = "/ba(na){2}/i";  
echo preg\_match($pattern, $str); // Outputs 1  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_regex_grouping)

## **Complete RegExp Reference**

For a complete reference, go to our [Complete PHP Regular Expression Reference](https://www.w3schools.com/php/php_ref_regex.asp).

The reference contains descriptions and examples of all Regular Expression functions.

## **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

## **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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_length)

## **Complete PHP Array Reference**

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

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

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Use the correct function to output the number of items in an array.

$fruits = array("Apple", "Banana", "Orange");

echo ;



Bottom of Form

# PHP Indexed Arrays

[❮ Previous](https://www.w3schools.com/php/php_arrays.asp)[Next ❯](https://www.w3schools.com/php/php_arrays_associative.asp)

## **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] . ".";  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_num)

## **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>";  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_num_loop)

## **Complete PHP Array Reference**

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

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

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Output the second item in the $fruits array.

$fruits = array("Apple", "Banana", "Orange");

echo ;



Submit Answer »

Bottom of Form

[❮ Previous](https://www.w3schools.com/php/php_arrays.asp)[Next ❯](https://www.w3schools.com/php/php_arrays_associative.asp)

# PHP Associative Arrays

[❮ Previous](https://www.w3schools.com/php/php_arrays_indexed.asp)[Next ❯](https://www.w3schools.com/php/php_arrays_multidimensional.asp)

## **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.";  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_assoc)

## **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>";  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_assoc_loop)

## **Complete PHP Array Reference**

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

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

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Create an associative array containing the age of Peter, Ben and Joe.

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



Submit Answer »

Bottom of Form

[❮ Previous](https://www.w3schools.com/php/php_arrays_indexed.asp)[Next ❯](https://www.w3schools.com/php/php_arrays_multidimensional.asp)

# PHP Multidimensional Arrays

[❮ Previous](https://www.w3schools.com/php/php_arrays_associative.asp)[Next ❯](https://www.w3schools.com/php/php_arrays_sort.asp)

In the previous pages, we have described arrays that are a single list of key/value pairs.

However, sometimes you want to store values with more than one key. For this, we have multidimensional arrays.

## **PHP - Multidimensional Arrays**

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

**The dimension of an array indicates the number of indices you need to select an element.**

* For a two-dimensional array you need two indices to select an element
* For a three-dimensional array you need three indices to select an element

## **PHP - Two-dimensional Arrays**

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

| **Name** | **Stock** | **Sold** |
| --- | --- | --- |
| Volvo | 22 | 18 |
| BMW | 15 | 13 |
| Saab | 5 | 2 |
| Land Rover | 17 | 15 |

We can store the data from the table above in a two-dimensional array, like this:

$cars = array (  
  array("Volvo",22,18),  
  array("BMW",15,13),  
  array("Saab",5,2),  
  array("Land Rover",17,15)  
);

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

### **Example**

<?php  
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";  
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";  
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";  
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_multi)

We can also put a for loop inside another for loop to get the elements of the $cars array (we still have to point to the two indices):

### **Example**

<?php  
for ($row = 0; $row < 4; $row++) {  
  echo "<p><b>Row number $row</b></p>";  
  echo "<ul>";  
  for ($col = 0; $col < 3; $col++) {  
    echo "<li>".$cars[$row][$col]."</li>";  
  }  
  echo "</ul>";  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_multi2)

# PHP Sorting Arrays

[❮ Previous](https://www.w3schools.com/php/php_arrays_multidimensional.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals.asp)

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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_sort_alpha)

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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_sort_num)

## **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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_rsort_alpha)

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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_rsort_num)

## **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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_asort)

## **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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_ksort)

## **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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_arsort)

## **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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_array_krsort)

## **Complete PHP Array Reference**

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

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

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Use the correct array method to sort the $colors array alphabetically.

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

;



Bottom of Form

# PHP Functions

[❮ Previous](https://www.w3schools.com/php/php_looping_break.asp)[Next ❯](https://www.w3schools.com/php/php_arrays.asp)

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 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 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!

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 followed by brackets ():

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_function1)

## **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");  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_function2)

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");  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_function3)

## **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 associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatches.

In the following example we try to send both a number and a string to the function without using strict:

### **Example**

<?php  
function addNumbers(int $a, int $b) {  
  return $a + $b;  
}  
echo addNumbers(5, "5 days");  
// since strict is NOT enabled "5 days" is changed to int(5), and it will return 10  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_func_non-strict)

To specify strict we need to set declare(strict\_types=1);. This must be on the very first line of the PHP file.

In the following example we try to send both a number and a string to the function, but here we have added the strict declaration:

### **Example**

<?php declare(strict\_types=1); // strict requirement  
  
function addNumbers(int $a, int $b) {  
  return $a + $b;  
}  
echo addNumbers(5, "5 days");  
// since strict is enabled and "5 days" is not an integer, an error will be thrown  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_func_strict)

The strict declaration forces things to be used in the intended way.

## **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 declare(strict\_types=1); // strict requirement  
function setHeight(int $minheight = 50) {  
  echo "The height is : $minheight <br>";  
}  
  
setHeight(350);  
setHeight(); // will use the default value of 50  
setHeight(135);  
setHeight(80);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_function4)

## **PHP Functions - Returning values**

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

### **Example**

<?php declare(strict\_types=1); // strict requirement  
function sum(int $x, int $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);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_function5)

## **PHP Return Type Declarations**

PHP 7 also supports Type Declarations for the return statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

To declare a type for the function return, add a colon ( : ) and the type right before the opening curly ( { )bracket when declaring the function.

In the following example we specify the return type for the function:

### **Example**

<?php declare(strict\_types=1); // strict requirement  
function addNumbers(float $a, float $b) : float {  
  return $a + $b;  
}  
echo addNumbers(1.2, 5.2);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_func_return_strict)

You can specify a different return type, than the argument types, but make sure the return is the correct type:

### **Example**

<?php declare(strict\_types=1); // strict requirement  
function addNumbers(float $a, float $b) : int {  
  return (int)($a + $b);  
}  
echo addNumbers(1.2, 5.2);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_func_return_strict2)

## **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:

<?php  
function add\_five(&$value) {  
  $value += 5;  
}  
  
$num = 2;  
add\_five($num);  
echo $num;  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_func_pass_ref)

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Create a function named myFunction.

{



echo "Hello World!";

}

Submit Answer »

Bottom of Form

[❮ Previous](https://www.w3schools.com/php/php_looping_break.asp)[Next ❯](https://www.w3schools.com/php/php_arrays.asp)

# PHP foreach Loop

[❮ Previous](https://www.w3schools.com/php/php_looping_for.asp)[Next ❯](https://www.w3schools.com/php/php_looping_break.asp)

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

## **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.

### **Examples**

The following example will output the values of the given array ($colors):

### **Example**

<?php  
$colors = array("red", "green", "blue", "yellow");  
  
foreach ($colors as $value) {  
  echo "$value <br>";  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_loop_foreach)

The following example will output both the keys and the values of the given array ($age):

### **Example**

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
  
foreach($age as $x => $val) {  
  echo "$x = $val<br>";  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_loop_foreach2)

# PHP Break and Continue

[❮ Previous](https://www.w3schools.com/php/php_looping_foreach.asp)[Next ❯](https://www.w3schools.com/php/php_functions.asp)

## **PHP Break**

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a loop.

This example jumps out of the loop when **x** is equal to **4**:

### **Example**

<?php  
for ($x = 0; $x < 10; $x++) {  
  if ($x == 4) {  
    break;  
  }  
  echo "The number is: $x <br>";  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_break)

## **PHP Continue**

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of **4**:

### **Example**

<?php  
for ($x = 0; $x < 10; $x++) {  
  if ($x == 4) {  
    continue;  
  }  
  echo "The number is: $x <br>";  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_continue)

## **Break and Continue in While Loop**

You can also use break and continue in while loops:

### **Break Example**

<?php  
$x = 0;  
  
while($x < 10) {  
  if ($x == 4) {  
    break;  
  }  
  echo "The number is: $x <br>";  
  $x++;  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_break_while)

### **Continue Example**

<?php  
$x = 0;  
  
while($x < 10) {  
  if ($x == 4) {  
    $x++;  
    continue;  
  }  
  echo "The number is: $x <br>";  
  $x++;  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_continue_while)

# PHP Global Variables - Superglobals

[❮ Previous](https://www.w3schools.com/php/php_arrays_sort.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals_globals.asp)

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

## **PHP Global Variables - Superglobals**

Some 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

The next chapters will explain some of the superglobals, and the rest will be explained in later chapters.

# PHP Superglobal - $GLOBALS

[❮ Previous](https://www.w3schools.com/php/php_superglobals.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals_server.asp)

Super global variables are built-in variables that are always available in all scopes.

## **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;  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_global_global)

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

[❮ Previous](https://www.w3schools.com/php/php_superglobals.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals_server.asp)

# PHP Superglobal - $\_SERVER

[❮ Previous](https://www.w3schools.com/php/php_superglobals_globals.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals_request.asp)

Super global variables are built-in variables that are always available in all scopes.

## **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'];  
?>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_global_server)

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 |

[❮ Previous](https://www.w3schools.com/php/php_superglobals_globals.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals_request.asp)

# PHP Superglobal - $\_REQUEST

[❮ Previous](https://www.w3schools.com/php/php_superglobals_server.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals_post.asp)

Super global variables are built-in variables that are always available in all scopes.

## **PHP $\_REQUEST**

PHP $\_REQUEST is a PHP super global variable which 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>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_global_request)

# PHP Superglobal - $\_POST

[❮ Previous](https://www.w3schools.com/php/php_superglobals_request.asp)[Next ❯](https://www.w3schools.com/php/php_superglobals_get.asp)

Super global variables are built-in variables that are always available in all scopes.

## **PHP $\_POST**

PHP $\_POST is a PHP super global variable which is 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>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_global_post)

**Tip:** You will learn more about $\_POST in the [PHP Forms](https://www.w3schools.com/php/php_forms.asp) chapter.

# PHP Superglobal - $\_GET

[❮ Previous](https://www.w3schools.com/php/php_superglobals_post.asp)[Next ❯](https://www.w3schools.com/php/php_regex.asp)

Super global variables are built-in variables that are always available in all scopes.

## **PHP $\_GET**

PHP $\_GET is a PHP super global variable which is 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>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_global_get)

**Tip:** You will learn more about $\_GET in the [PHP Forms](https://www.w3schools.com/php/php_forms.asp) chapter.

[❮ Previous](https://www.w3schools.com/php/php_superglobals_post.asp)[Next ❯](https://www.w3schools.com/php/php_regex.asp)

# PHP Form Handling

[❮ Previous](https://www.w3schools.com/php/php_regex.asp)[Next ❯](https://www.w3schools.com/php/php_form_validation.asp)

The PHP superglobals $\_GET and $\_POST are used to collect form-data.

## **PHP - A Simple HTML Form**

The example below displays a simple HTML form with two input fields and a submit button:

### **Example**

<html>  
<body>  
  
<form action="welcome.php" method="post">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>  
  
</body>  
</html>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_form_post)

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

<html>  
<body>  
  
Welcome <?php echo $\_POST["name"]; ?><br>  
Your email address is: <?php echo $\_POST["email"]; ?>  
  
</body>  
</html>

The output could be something like this:

Welcome John  
Your email address is john.doe@example.com

The same result could also be achieved using the HTTP GET method:

### **Example**

<html>  
<body>  
  
<form action="welcome\_get.php" method="get">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>  
  
</body>  
</html>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_form_get)

and "welcome\_get.php" looks like this:

<html>  
<body>  
  
Welcome <?php echo $\_GET["name"]; ?><br>  
Your email address is: <?php echo $\_GET["email"]; ?>  
  
</body>  
</html>

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

**Think SECURITY when processing PHP forms!**

This page does not contain any form validation, it just shows how you can send and retrieve form data.

However, the next pages will show how to process PHP forms with security in mind! Proper validation of form data is important to protect your form from hackers and spammers!

## **GET vs. POST**

Both GET and POST create an array (e.g. array( key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $\_GET and $\_POST. These 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.

$\_GET is an array of variables passed to the current script via the URL parameters.

$\_POST is an array of variables passed to the current script via the HTTP POST method.

## **When to use GET?**

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

**Note:** GET should NEVER be used for sending passwords or other sensitive information!

## **When to use POST?**

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

**Developers prefer POST for sending form data.**

Next, lets see how we can process PHP forms the secure way!

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

If the form in the white section below gets submitted, how can you, in welcome.php, output the value from the "first name" field?

<form action="welcome.php" method="get">

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

</form>

<html>

<body>

Welcome <?php echo ; ?>



</body>

</html>

Submit Answer »

Bottom of Form

# PHP Form Validation

[❮ Previous](https://www.w3schools.com/php/php_forms.asp)[Next ❯](https://www.w3schools.com/php/php_form_required.asp)

This and the next chapters show how to use PHP to validate form data.

## **PHP Form Validation**

**Think SECURITY when processing PHP forms!**

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

The validation rules for the form above are as follows:

| **Field** | **Validation Rules** |
| --- | --- |
| Name | Required. + Must only contain letters and whitespace |
| E-mail | Required. + Must contain a valid email address (with @ and .) |
| Website | Optional. If present, it must contain a valid URL |
| Comment | Optional. Multi-line input field (textarea) |
| Gender | Required. Must select one |

First we will look at the plain HTML code for the form:

## **Text Fields**

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

Name: <input type="text" name="name">  
E-mail: <input type="text" name="email">  
Website: <input type="text" name="website">  
Comment: <textarea name="comment" rows="5" cols="40"></textarea>

## **Radio Buttons**

The gender fields are radio buttons and the HTML code looks like this:

Gender:  
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male  
<input type="radio" name="gender" value="other">Other

## **The Form Element**

The HTML code of the form looks like this:

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

When the form is submitted, the form data is sent with method="post".

**What is the $\_SERVER["PHP\_SELF"] variable?**  
  
The $\_SERVER["PHP\_SELF"] is a super global variable that returns the filename of the currently executing script.

So, the $\_SERVER["PHP\_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

**What is the htmlspecialchars() function?**  
  
The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

## **Big Note on PHP Form Security**

The $\_SERVER["PHP\_SELF"] variable can be used by hackers!

If PHP\_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

**Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.**

Assume we have the following form in a page named "test\_form.php":

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

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test\_form.php", the above code will be translated to:

<form method="post" action="test\_form.php">

So far, so good.

However, consider that a user enters the following URL in the address bar:

http://www.example.com/test\_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

In this case, the above code will be translated to:

<form method="post" action="test\_form.php/"><script>alert('hacked')</script>

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP\_SELF variable can be exploited.

Be aware of that **any JavaScript code can be added inside the <script> tag!**A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

## **How To Avoid $\_SERVER["PHP\_SELF"] Exploits?**

$\_SERVER["PHP\_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP\_SELF variable, it will result in the following output:

<form method="post" action="test\_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">

The exploit attempt fails, and no harm is done!

## **Validate Form Data With PHP**

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

<script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this:

&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test\_input().

Now, we can check each $\_POST variable with the test\_input() function, and the script looks like this:

### **Example**

<?php  
// define variables and set to empty values  
$name = $email = $gender = $comment = $website = "";  
  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  $name = test\_input($\_POST["name"]);  
  $email = test\_input($\_POST["email"]);  
  $website = test\_input($\_POST["website"]);  
  $comment = test\_input($\_POST["comment"]);  
  $gender = test\_input($\_POST["gender"]);  
}  
  
function test\_input($data) {  
  $data = trim($data);  
  $data = stripslashes($data);  
  $data = htmlspecialchars($data);  
  return $data;  
}  
?>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_form_validation_escapechar)

Notice that at the start of the script, we check whether the form has been submitted using $\_SERVER["REQUEST\_METHOD"]. If the REQUEST\_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed.

# PHP Forms - Required Fields

[❮ Previous](https://www.w3schools.com/php/php_form_validation.asp)[Next ❯](https://www.w3schools.com/php/php_form_url_email.asp)

This chapter shows how to make input fields required and create error messages if needed.

## **PHP - Required Fields**

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

| **Field** | **Validation Rules** |
| --- | --- |
| Name | Required. + Must only contain letters and whitespace |
| E-mail | Required. + Must contain a valid email address (with @ and .) |
| Website | Optional. If present, it must contain a valid URL |
| Comment | Optional. Multi-line input field (textarea) |
| Gender | Required. Must select one |

In the previous chapter, all input fields were optional.

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $\_POST variable. This checks if the $\_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test\_input() function:

<?php  
// define variables and set to empty values  
$nameErr = $emailErr = $genderErr = $websiteErr = "";  
$name = $email = $gender = $comment = $website = "";  
  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  if (empty($\_POST["name"])) {  
    $nameErr = "Name is required";  
  } else {  
    $name = test\_input($\_POST["name"]);  
  }  
  
  if (empty($\_POST["email"])) {  
    $emailErr = "Email is required";  
  } else {  
    $email = test\_input($\_POST["email"]);  
  }  
  
  if (empty($\_POST["website"])) {  
    $website = "";  
  } else {  
    $website = test\_input($\_POST["website"]);  
  }  
  
  if (empty($\_POST["comment"])) {  
    $comment = "";  
  } else {  
    $comment = test\_input($\_POST["comment"]);  
  }  
  
  if (empty($\_POST["gender"])) {  
    $genderErr = "Gender is required";  
  } else {  
    $gender = test\_input($\_POST["gender"]);  
  }  
}  
?>

## **PHP - Display The Error Messages**

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

### **Example**

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">  
  
Name: <input type="text" name="name">  
<span class="error">\* <?php echo $nameErr;?></span>  
<br><br>  
E-mail:  
<input type="text" name="email">  
<span class="error">\* <?php echo $emailErr;?></span>  
<br><br>  
Website:  
<input type="text" name="website">  
<span class="error"><?php echo $websiteErr;?></span>  
<br><br>  
Comment: <textarea name="comment" rows="5" cols="40"></textarea>  
<br><br>  
Gender:  
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male  
<input type="radio" name="gender" value="other">Other  
<span class="error">\* <?php echo $genderErr;?></span>  
<br><br>  
<input type="submit" name="submit" value="Submit">  
  
</form>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_form_validation_required)

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

# PHP Forms - Validate E-mail and URL

[❮ Previous](https://www.w3schools.com/php/php_form_required.asp)[Next ❯](https://www.w3schools.com/php/php_form_complete.asp)

This chapter shows how to validate names, e-mails, and URLs.

## **PHP - Validate Name**

The code below shows a simple way to check if the name field only contains letters, dashes, apostrophes and whitespaces. If the value of the name field is not valid, then store an error message:

$name = test\_input($\_POST["name"]);  
if (!preg\_match("/^[a-zA-Z-' ]\*$/",$name)) {  
  $nameErr = "Only letters and white space allowed";  
}

**The**[**preg\_match()**](https://www.w3schools.com/php/func_regex_preg_match.asp)**function searches a string for pattern, returning true if the pattern exists, and false otherwise.**

## **PHP - Validate E-mail**

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter\_var() function.

In the code below, if the e-mail address is not well-formed, then store an error message:

$email = test\_input($\_POST["email"]);  
if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {  
  $emailErr = "Invalid email format";  
}

## **PHP - Validate URL**

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

$website = test\_input($\_POST["website"]);  
if (!preg\_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~\_|!:,.;]\*[-a-z0-9+&@#\/%=~\_|]/i",$website)) {  
  $websiteErr = "Invalid URL";  
}

## **PHP - Validate Name, E-mail, and URL**

Now, the script looks like this:

### **Example**

<?php  
// define variables and set to empty values  
$nameErr = $emailErr = $genderErr = $websiteErr = "";  
$name = $email = $gender = $comment = $website = "";  
  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  if (empty($\_POST["name"])) {  
    $nameErr = "Name is required";  
  } else {  
    $name = test\_input($\_POST["name"]);  
    // check if name only contains letters and whitespace  
    if (!preg\_match("/^[a-zA-Z-' ]\*$/",$name)) {  
      $nameErr = "Only letters and white space allowed";  
    }  
  }  
  
  if (empty($\_POST["email"])) {  
    $emailErr = "Email is required";  
  } else {  
    $email = test\_input($\_POST["email"]);  
    // check if e-mail address is well-formed  
    if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {  
      $emailErr = "Invalid email format";  
    }  
  }  
  
  if (empty($\_POST["website"])) {  
    $website = "";  
  } else {  
    $website = test\_input($\_POST["website"]);  
    // check if URL address syntax is valid (this regular expression also allows dashes in the URL)  
    if (!preg\_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~\_|!:,.;]\*[-a-z0-9+&@#\/%=~\_|]/i",$website)) {  
      $websiteErr = "Invalid URL";  
    }  
  }  
  
  if (empty($\_POST["comment"])) {  
    $comment = "";  
  } else {  
    $comment = test\_input($\_POST["comment"]);  
  }  
  
  if (empty($\_POST["gender"])) {  
    $genderErr = "Gender is required";  
  } else {  
    $gender = test\_input($\_POST["gender"]);  
  }  
}  
?>

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_form_validation_special)

The next step is to show how to prevent the form from emptying all the input fields when the user submits the form.

# PHP Complete Form Example

[❮ Previous](https://www.w3schools.com/php/php_form_url_email.asp)[Next ❯](https://www.w3schools.com/php/php_date.asp)

This chapter shows how to keep the values in the input fields when the user hits the submit button.

## **PHP - Keep The Values in The Form**

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the $name, $email, $website, and $comment variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

Name: <input type="text" name="name" value="<?php echo $name;?>">  
  
E-mail: <input type="text" name="email" value="<?php echo $email;?>">  
  
Website: <input type="text" name="website" value="<?php echo $website;?>">  
  
Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>  
  
Gender:  
<input type="radio" name="gender"  
<?php if (isset($gender) && $gender=="female") echo "checked";?>  
value="female">Female  
<input type="radio" name="gender"  
<?php if (isset($gender) && $gender=="male") echo "checked";?>  
value="male">Male  
<input type="radio" name="gender"  
<?php if (isset($gender) && $gender=="other") echo "checked";?>  
value="other">Other

## **PHP - Complete Form Example**

Here is the complete code for the PHP Form Validation Example:

### **Example**

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_form_validation_complete)

## **PHP - Complete Form Example**

Here is the complete code for the PHP Form Validation Example:

### **Example**

[Try it Yourself »](https://tryphp.w3schools.com/showphp.php?filename=demo_form_validation_complete)

[❮ Previous](https://www.w3schools.com/php/php_form_url_email.asp)[Next ❯](https://www.w3schools.com/php/php_date.asp)

[❮ Previous](https://www.w3schools.com/php/php_form_url_email.asp)[Next ❯](https://www.w3schools.com/php/php_date.asp)

# PHP Date and Time

[❮ Previous](https://www.w3schools.com/php/php_form_complete.asp)[Next ❯](https://www.w3schools.com/php/php_includes.asp)

The PHP date() function is used to format a date and/or a time.

## **The PHP Date() Function**

The PHP date() function formats a timestamp to a more readable date and time.

### **Syntax**

date(*format*,*timestamp*)

| **Parameter** | **Description** |
| --- | --- |
| format | Required. Specifies the format of the timestamp |
| timestamp | Optional. Specifies a timestamp. Default is the current date and time |

A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

## **Get a Date**

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

* d - Represents the day of the month (01 to 31)
* m - Represents a month (01 to 12)
* Y - Represents a year (in four digits)
* l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

The example below formats today's date in three different ways:

### **Example**

<?php  
echo "Today is " . date("Y/m/d") . "<br>";  
echo "Today is " . date("Y.m.d") . "<br>";  
echo "Today is " . date("Y-m-d") . "<br>";  
echo "Today is " . date("l");  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date1)

## **PHP Tip - Automatic Copyright Year**

Use the date() function to automatically update the copyright year on your website:

### **Example**

&copy; 2010-<?php echo date("Y");?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date_copyright)

## **Get a Time**

Here are some characters that are commonly used for times:

* H - 24-hour format of an hour (00 to 23)
* h - 12-hour format of an hour with leading zeros (01 to 12)
* i - Minutes with leading zeros (00 to 59)
* s - Seconds with leading zeros (00 to 59)
* a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

### **Example**

<?php  
echo "The time is " . date("h:i:sa");  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date2)

Note that the PHP date() function will return the current date/time of the server!

## **Get Your Time Zone**

If the time you got back from the code is not correct, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set the timezone you want to use.

The example below sets the timezone to "America/New\_York", then outputs the current time in the specified format:

### **Example**

<?php  
date\_default\_timezone\_set("America/New\_York");  
echo "The time is " . date("h:i:sa");  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date3)

## **Create a Date With mktime()**

The optional *timestamp* parameter in the date() function specifies a timestamp. If omitted, the current date and time will be used (as in the examples above).

The PHP mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

### **Syntax**

mktime(*hour, minute, second, month, day, year*)

The example below creates a date and time with the date() function from a number of parameters in the mktime() function:

### **Example**

<?php  
$d=mktime(11, 14, 54, 8, 12, 2014);  
echo "Created date is " . date("Y-m-d h:i:sa", $d);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date4)

## **Create a Date From a String With strtotime()**

The PHP strtotime() function is used to convert a human readable date string into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

### **Syntax**

strtotime(*time, now*)

The example below creates a date and time from the strtotime() function:

### **Example**

<?php  
$d=strtotime("10:30pm April 15 2014");  
echo "Created date is " . date("Y-m-d h:i:sa", $d);  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date5)

PHP is quite clever about converting a string to a date, so you can put in various values:

### **Example**

<?php  
$d=strtotime("tomorrow");  
echo date("Y-m-d h:i:sa", $d) . "<br>";  
  
$d=strtotime("next Saturday");  
echo date("Y-m-d h:i:sa", $d) . "<br>";  
  
$d=strtotime("+3 Months");  
echo date("Y-m-d h:i:sa", $d) . "<br>";  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date6)

However, strtotime() is not perfect, so remember to check the strings you put in there.

## **More Date Examples**

The example below outputs the dates for the next six Saturdays:

### **Example**

<?php  
$startdate = strtotime("Saturday");  
$enddate = strtotime("+6 weeks", $startdate);  
  
while ($startdate < $enddate) {  
  echo date("M d", $startdate) . "<br>";  
  $startdate = strtotime("+1 week", $startdate);  
}  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date7)

The example below outputs the number of days until 4th of July:

### **Example**

<?php  
$d1=strtotime("July 04");  
$d2=ceil(($d1-time())/60/60/24);  
echo "There are " . $d2 ." days until 4th of July.";  
?>

[Try it Yourself »](https://www.w3schools.com/php/phptryit.asp?filename=tryphp_date8)

## **Complete PHP Date Reference**

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

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

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Use the correct date function to output the weekday name of today (monday, tuesday etc.).

echo ;



Submit Answer »

Bottom of Form

# [❮ Previous](https://www.w3schools.com/php/php_form_complete.asp)[Next ❯](https://www.w3schools.com/php/php_includes.asp)PHP Include Files

[❮ Previous](https://www.w3schools.com/php/php_date.asp)[Next ❯](https://www.w3schools.com/php/php_file.asp)

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

## **PHP include and require Statements**

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

**The include and require statements are identical, except upon failure:**

* require will produce a fatal error (E\_COMPILE\_ERROR) and stop the script
* include will only produce a warning (E\_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

### **Syntax**

include '*filename*';  
  
or  
  
require '*filename*';

## **PHP include Examples**

### **Example 1**

Assume we have a standard footer file called "footer.php", that looks like this:

<?php  
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";  
?>

To include the footer file in a page, use the include statement:

### **Example**

<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
<p>Some more text.</p>  
<?php include 'footer.php';?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_include1)

### **Example 2**

Assume we have a standard menu file called "menu.php":

<?php  
echo '<a href="/default.asp">Home</a> -  
<a href="/html/default.asp">HTML Tutorial</a> -  
<a href="/css/default.asp">CSS Tutorial</a> -  
<a href="/js/default.asp">JavaScript Tutorial</a> -  
<a href="default.asp">PHP Tutorial</a>';  
?>

All pages in the Web site should use this menu file. Here is how it can be done (we are using a <div> element so that the menu easily can be styled with CSS later):

### **Example**

<html>  
<body>  
  
<div class="menu">  
<?php include 'menu.php';?>  
</div>  
  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
<p>Some more text.</p>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_include2)

### **Example 3**

Assume we have a file called "vars.php", with some variables defined:

<?php  
$color='red';  
$car='BMW';  
?>

Then, if we include the "vars.php" file, the variables can be used in the calling file:

### **Example**

<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<?php include 'vars.php';  
echo "I have a $color $car.";  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_include3)

## **PHP include vs. require**

The require statement is also used to include a file into the PHP code.

However, there is one big difference between include and require; when a file is included with the include statement and PHP cannot find it, the script will continue to execute:

### **Example**

<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<?php include 'noFileExists.php';  
echo "I have a $color $car.";  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_include4)

If we do the same example using the require statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

### **Example**

<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<?php require 'noFileExists.php';  
echo "I have a $color $car.";  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_include5)

Use require when the file is required by the application.

Use include when the file is not required and application should continue when file is not found.

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Write a correct syntax to include a file named "footer.php".

<?php ;?>



Submit Answer »

Bottom of Form

[❮ Previous](https://www.w3schools.com/php/php_date.asp)[Next ❯](https://www.w3schools.com/php/php_file.asp)

# PHP Sessions

[❮ Previous](https://www.w3schools.com/php/php_cookies.asp)[Next ❯](https://www.w3schools.com/php/php_filter.asp)

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

## **What is a PHP Session?**

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

**Tip:** If you need a permanent storage, you may want to store the data in a [database](https://www.w3schools.com/php/php_mysql_intro.asp).

## **Start a PHP Session**

A session is started with the session\_start() function.

Session variables are set with the PHP global variable: $\_SESSION.

Now, let's create a new page called "demo\_session1.php". In this page, we start a new PHP session and set some session variables:

### **Example**

<?php  
// Start the session  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// Set session variables  
$\_SESSION["favcolor"] = "green";  
$\_SESSION["favanimal"] = "cat";  
echo "Session variables are set.";  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_session1)

**Note:** The session\_start() function must be the very first thing in your document. Before any HTML tags.

## **Get PHP Session Variable Values**

Next, we create another page called "demo\_session2.php". From this page, we will access the session information we set on the first page ("demo\_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session\_start()).

Also notice that all session variable values are stored in the global $\_SESSION variable:

### **Example**

<?php  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// Echo session variables that were set on previous page  
echo "Favorite color is " . $\_SESSION["favcolor"] . ".<br>";  
echo "Favorite animal is " . $\_SESSION["favanimal"] . ".";  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_session2)

Another way to show all the session variable values for a user session is to run the following code:

### **Example**

<?php  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
print\_r($\_SESSION);  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_session3)

**How does it work? How does it know it's me?**  
  
Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

## **Modify a PHP Session Variable**

To change a session variable, just overwrite it:

### **Example**

<?php  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// to change a session variable, just overwrite it  
$\_SESSION["favcolor"] = "yellow";  
print\_r($\_SESSION);  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_session4)

## **Destroy a PHP Session**

To remove all global session variables and destroy the session, use session\_unset() and session\_destroy():

### **Example**

<?php  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// remove all session variables  
session\_unset();  
  
// destroy the session  
session\_destroy();  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_session5)

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Create a session variable named "favcolor".

session\_start();

["favcolor"] = "green";



Submit Answer »

Bottom of Form

# PHP Cookies

[❮ Previous](https://www.w3schools.com/php/php_file_upload.asp)[Next ❯](https://www.w3schools.com/php/php_sessions.asp)

## **What is a Cookie?**

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

## **Create Cookies With PHP**

A cookie is created with the setcookie() function.

### **Syntax**

setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

## **PHP Create/Retrieve a Cookie**

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 \* 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $\_COOKIE). We also use the isset() function to find out if the cookie is set:

### **Example**

<?php  
$cookie\_name = "user";  
$cookie\_value = "John Doe";  
setcookie($cookie\_name, $cookie\_value, time() + (86400 \* 30), "/"); // 86400 = 1 day  
?>  
<html>  
<body>  
  
<?php  
if(!isset($\_COOKIE[$cookie\_name])) {  
  echo "Cookie named '" . $cookie\_name . "' is not set!";  
} else {  
  echo "Cookie '" . $cookie\_name . "' is set!<br>";  
  echo "Value is: " . $\_COOKIE[$cookie\_name];  
}  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_cookie1)

**Note:** The setcookie() function must appear BEFORE the <html> tag.

**Note:** The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

## **Modify a Cookie Value**

To modify a cookie, just set (again) the cookie using the setcookie() function:

### **Example**

<?php  
$cookie\_name = "user";  
$cookie\_value = "Alex Porter";  
setcookie($cookie\_name, $cookie\_value, time() + (86400 \* 30), "/");  
?>  
<html>  
<body>  
  
<?php  
if(!isset($\_COOKIE[$cookie\_name])) {  
  echo "Cookie named '" . $cookie\_name . "' is not set!";  
} else {  
  echo "Cookie '" . $cookie\_name . "' is set!<br>";  
  echo "Value is: " . $\_COOKIE[$cookie\_name];  
}  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_cookie3)

## **Delete a Cookie**

To delete a cookie, use the setcookie() function with an expiration date in the past:

### **Example**

<?php  
// set the expiration date to one hour ago  
setcookie("user", "", time() - 3600);  
?>  
<html>  
<body>  
  
<?php  
echo "Cookie 'user' is deleted.";  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_cookie4)

## **Check if Cookies are Enabled**

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $\_COOKIE array variable:

### **Example**

<?php  
setcookie("test\_cookie", "test", time() + 3600, '/');  
?>  
<html>  
<body>  
  
<?php  
if(count($\_COOKIE) > 0) {  
  echo "Cookies are enabled.";  
} else {  
  echo "Cookies are disabled.";  
}  
?>  
  
</body>  
</html>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_cookie5)

## **Complete PHP Network Reference**

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

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Create a cookie named "username".

("username", "John", time() + (86400 \* 30), "/");



Bottom of Form

# PHP File Handling

[❮ Previous](https://www.w3schools.com/php/php_includes.asp)[Next ❯](https://www.w3schools.com/php/php_file_open.asp)

File handling is an important part of any web application. You often need to open and process a file for different tasks.

## **PHP Manipulating Files**

PHP has several functions for creating, reading, uploading, and editing files.

**Be careful when manipulating files!**

When you are manipulating files you must be very careful.

You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

## **PHP readfile() Function**

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML  
CSS = Cascading Style Sheets  
HTML = Hyper Text Markup Language  
PHP = PHP Hypertext Preprocessor  
SQL = Structured Query Language  
SVG = Scalable Vector Graphics  
XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

### **Example**

<?php  
echo readfile("webdictionary.txt");  
?>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_file_readfile)

The readfile() function is useful if all you want to do is open up a file and read its contents.

The next chapters will teach you more about file handling.

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Assume we have a file named "webdict.txt", write the correct syntax to open and read the file content.

echo ;



Submit Answer »

Bottom of Form

# PHP File Open/Read/Close

[❮ Previous](https://www.w3schools.com/php/php_file.asp)[Next ❯](https://www.w3schools.com/php/php_file_create.asp)

In this chapter we will teach you how to open, read, and close a file on the server.

## **PHP Open File - fopen()**

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

AJAX = Asynchronous JavaScript and XML  
CSS = Cascading Style Sheets  
HTML = Hyper Text Markup Language  
PHP = PHP Hypertext Preprocessor  
SQL = Structured Query Language  
SVG = Scalable Vector Graphics  
XML = EXtensible Markup Language

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

### **Example**

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fread($myfile,filesize("webdictionary.txt"));  
fclose($myfile);  
?>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_file_fopen)

**Tip:** The fread() and the fclose() functions will be explained below.

The file may be opened in one of the following modes:

| **Modes** | **Description** |
| --- | --- |
| r | **Open a file for read only**. File pointer starts at the beginning of the file |
| w | **Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file |
| a | **Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist |
| x | **Creates a new file for write only**. Returns FALSE and an error if file already exists |
| r+ | **Open a file for read/write**. File pointer starts at the beginning of the file |
| w+ | **Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file |
| a+ | **Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist |
| x+ | **Creates a new file for read/write**. Returns FALSE and an error if file already exists |

## **PHP Read File - fread()**

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

fread($myfile,filesize("webdictionary.txt"));

## **PHP Close File - fclose()**

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

<?php  
$myfile = fopen("webdictionary.txt", "r");  
// some code to be executed....  
fclose($myfile);  
?>

## **PHP Read Single Line - fgets()**

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

### **Example**

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fgets($myfile);  
fclose($myfile);  
?>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_file_fgets)

**Note:** After a call to the fgets() function, the file pointer has moved to the next line.

## **PHP Check End-Of-File - feof()**

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

### **Example**

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
// Output one line until end-of-file  
while(!feof($myfile)) {  
  echo fgets($myfile) . "<br>";  
}  
fclose($myfile);  
?>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_file_feof)

## **PHP Read Single Character - fgetc()**

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

### **Example**

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
// Output one character until end-of-file  
while(!feof($myfile)) {  
  echo fgetc($myfile);  
}  
fclose($myfile);  
?>

[Run example »](https://tryphp.w3schools.com/showphp.php?filename=demo_file_fgetc)

**Note:** After a call to the fgetc() function, the file pointer moves to the next character.

## **Complete PHP Filesystem Reference**

For a complete reference of filesystem functions, go to our complete [PHP Filesystem Reference](https://www.w3schools.com/php/php_ref_filesystem.asp).

## **PHP Exercises**

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

Open a file, and write the correct syntax to output one character at the time, until end-of-file.

$myfile = fopen("webdict.txt", "r");

while(!($myfile)) {



echo ($myfile);



}

Bottom of Form

# PHP File Create/Write

[❮ Previous](https://www.w3schools.com/php/php_file_open.asp)[Next ❯](https://www.w3schools.com/php/php_file_upload.asp)

In this chapter we will teach you how to create and write to a file on the server.

## **PHP Create File - fopen()**

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

### **Example**

$myfile = fopen("testfile.txt", "w")

## **PHP File Permissions**

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

## **PHP Write to File - fwrite()**

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

### **Example**

<?php  
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");  
$txt = "John Doe\n";  
fwrite($myfile, $txt);  
$txt = "Jane Doe\n";  
fwrite($myfile, $txt);  
fclose($myfile);  
?>

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

John Doe  
Jane Doe

## **PHP Overwriting**

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

### **Example**

<?php  
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");  
$txt = "Mickey Mouse\n";  
fwrite($myfile, $txt);  
$txt = "Minnie Mouse\n";  
fwrite($myfile, $txt);  
fclose($myfile);  
?>

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

Mickey Mouse  
Minnie Mouse

## **Complete PHP Filesystem Reference**

For a complete reference of filesystem functions, go to our complete [PHP Filesystem Reference](https://www.w3schools.com/php/php_ref_filesystem.asp).

# PHP File Upload

[❮ Previous](https://www.w3schools.com/php/php_file_create.asp)[Next ❯](https://www.w3schools.com/php/php_cookies.asp)

With PHP, it is easy to upload files to the server.

However, with ease comes danger, so always be careful when allowing file uploads!

## **Configure The "php.ini" File**

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the file\_uploads directive, and set it to On:

file\_uploads = On

## **Create The HTML Form**

Next, create an HTML form that allow users to choose the image file they want to upload:

<!DOCTYPE html>  
<html>  
<body>  
  
<form action="upload.php" method="post" enctype="multipart/form-data">  
  Select image to upload:  
  <input type="file" name="fileToUpload" id="fileToUpload">  
  <input type="submit" value="Upload Image" name="submit">  
</form>  
  
</body>  
</html>

Some rules to follow for the HTML form above:

* Make sure that the form uses method="post"
* The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form

Without the requirements above, the file upload will not work.

Other things to notice:

* The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The form above sends data to a file called "upload.php", which we will create next.

## **Create The Upload File PHP Script**

The "upload.php" file contains the code for uploading a file:

<?php  
$target\_dir = "uploads/";  
$target\_file = $target\_dir . basename($\_FILES["fileToUpload"]["name"]);  
$uploadOk = 1;  
$imageFileType = strtolower(pathinfo($target\_file,PATHINFO\_EXTENSION));  
// Check if image file is a actual image or fake image  
if(isset($\_POST["submit"])) {  
  $check = getimagesize($\_FILES["fileToUpload"]["tmp\_name"]);  
  if($check !== false) {  
    echo "File is an image - " . $check["mime"] . ".";  
    $uploadOk = 1;  
  } else {  
    echo "File is not an image.";  
    $uploadOk = 0;  
  }  
}  
?>

PHP script explained:

* $target\_dir = "uploads/" - specifies the directory where the file is going to be placed
* $target\_file specifies the path of the file to be uploaded
* $uploadOk=1 is not used yet (will be used later)
* $imageFileType holds the file extension of the file (in lower case)
* Next, check if the image file is an actual image or a fake image

**Note:** You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there.

## **Check if File Already Exists**

Now we can add some restrictions.

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

// Check if file already exists  
if (file\_exists($target\_file)) {  
  echo "Sorry, file already exists.";  
  $uploadOk = 0;  
}

## **Limit File Size**

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 500KB, an error message is displayed, and $uploadOk is set to 0:

// Check file size  
if ($\_FILES["fileToUpload"]["size"] > 500000) {  
  echo "Sorry, your file is too large.";  
  $uploadOk = 0;  
}

## **Limit File Type**

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

// Allow certain file formats  
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"  
&& $imageFileType != "gif" ) {  
  echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";  
  $uploadOk = 0;  
}

## **Complete Upload File PHP Script**

The complete "upload.php" file now looks like this:

<?php  
$target\_dir = "uploads/";  
$target\_file = $target\_dir . basename($\_FILES["fileToUpload"]["name"]);  
$uploadOk = 1;  
$imageFileType = strtolower(pathinfo($target\_file,PATHINFO\_EXTENSION));  
  
// Check if image file is a actual image or fake image  
if(isset($\_POST["submit"])) {  
  $check = getimagesize($\_FILES["fileToUpload"]["tmp\_name"]);  
  if($check !== false) {  
    echo "File is an image - " . $check["mime"] . ".";  
    $uploadOk = 1;  
  } else {  
    echo "File is not an image.";  
    $uploadOk = 0;  
  }  
}  
  
// Check if file already exists  
if (file\_exists($target\_file)) {  
  echo "Sorry, file already exists.";  
  $uploadOk = 0;  
}  
  
// Check file size  
if ($\_FILES["fileToUpload"]["size"] > 500000) {  
  echo "Sorry, your file is too large.";  
  $uploadOk = 0;  
}  
  
// Allow certain file formats  
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"  
&& $imageFileType != "gif" ) {  
  echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";  
  $uploadOk = 0;  
}  
  
// Check if $uploadOk is set to 0 by an error  
if ($uploadOk == 0) {  
  echo "Sorry, your file was not uploaded.";  
// if everything is ok, try to upload file  
} else {  
  if (move\_uploaded\_file($\_FILES["fileToUpload"]["tmp\_name"], $target\_file)) {  
    echo "The file ". htmlspecialchars( basename( $\_FILES["fileToUpload"]["name"])). " has been uploaded.";  
  } else {  
    echo "Sorry, there was an error uploading your file.";  
  }  
}  
?>

## **Complete PHP Filesystem Reference**

For a complete reference of filesystem functions, go to our complete [PHP Filesystem Reference](https://www.w3schools.com/php/php_ref_filesystem.asp).

[❮ Previous](https://www.w3schools.com/php/php_file_create.asp)[Next ❯](https://www.w3schools.com/php/php_cookies.asp)