PHP

What is PHP?

* PHP is an acronym for "PHP: Hypertext Pre-processor"
* PHP is a widely-used, open-source scripting language
* PHP scripts are executed on the server

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

# **PHP Syntax**

## Basic PHP Syntax

A PHP script can be placed anywhere in the document.  PHP statements end with a semicolon (;). 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>

## PHP Case Sensitivity

In PHP, keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are not case-sensitive. ECHO, echo, Echo are same.

## Comments in PHP

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

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

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 Variables**

## Creating (Declaring) PHP Variables

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

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

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

Rules for PHP variables:

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

## Output Variables

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

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

### **Example**

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

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

### **Example**

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

## PHP Variables Scope

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used. PHP has three different variable scopes: local, global, static

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

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

## PHP The global Keyword

The global keyword is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function)

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

The example above can be rewritten like this:

### **Example**

<?php  
$x = 5;  
$y = 10;  
function myTest() {  
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];  
}  
myTest();  
echo $y; // outputs 15  
?>

## PHP The static Keyword

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

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

### **Example**

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

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

**Note:** The variable is still local to the function.

# **PHP echo and print Statements**

## PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen. The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

## The PHP print Statement

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

**Display Text**

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

### **Example**

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

## PHP Data Types

PHP supports the following data types: String, Integer, Float (floating point numbers - also called double), Boolean, Array, Object, NULL, Resource.

## PHP String

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

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

The PHP var\_dump() function returns the data type and value.

## PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

## PHP NULL Value

Null is a special data type which can have only one value: NULL. A variable of data type NULL is a variable that has no value assigned to it. **Tip:** If a variable is created without a value, it is automatically assigned a value of NULL. Variables can also be emptied by setting the value to NULL.

## PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common example of using the resource data type is a database call.

## PHP String Functions

## strlen() - Return the Length of a String

## str\_word\_count() - Count Words in a String

## strrev() - Reverse a String

## strpos() - Search For a Text Within a String

The PHP strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

## str\_replace() - Replace Text Within a String

<?php  
echo str\_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!  
?>

PHP Integers

2, 256, -256, 10358, -179567 are all integers.

An integer is a number without any decimal part.

An integer data type is a non-decimal number between -2147483648 and 2147483647 in 32 bit systems, and between -9223372036854775808 and 9223372036854775807 in 64 bit systems. A value greater (or lower) than this, will be stored as float, because it exceeds the limit of an integer.

**Note:** Another important thing to know is that even if 4 \* 2.5 is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some rules for integers:

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

PHP has the following predefined constants for integers:

* PHP\_INT\_MAX - The largest integer supported
* PHP\_INT\_MIN - The smallest integer supported
* PHP\_INT\_SIZE -  The size of an integer in bytes

PHP has the following functions to check if the type of a variable is integer:

* is\_int()
* is\_integer() - alias of is\_int()
* is\_long() - alias of is\_int()

PHP Floats

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

2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.

The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits.

PHP has the following predefined constants for floats (from PHP 7.2):

* PHP\_FLOAT\_MAX - The largest representable floating point number
* PHP\_FLOAT\_MIN - The smallest representable positive floating point number
* - PHP\_FLOAT\_MAX - The smallest representable negative floating point number
* PHP\_FLOAT\_DIG - The number of decimal digits that can be rounded into a float and back without precision loss
* PHP\_FLOAT\_EPSILON - The smallest representable positive number x, so that x + 1.0 != 1.0

PHP has the following functions to check if the type of a variable is float:

* is\_float()
* is\_double() - alias of is\_float()

PHP Infinity

A numeric value that is larger than PHP\_FLOAT\_MAX is considered infinite.

PHP has the following functions to check if a numeric value is finite or infinite:

* [is\_finite()](https://www.w3schools.com/php/func_math_is_finite.asp)
* [is\_infinite()](https://www.w3schools.com/php/func_math_is_infinite.asp)

However, the PHP var\_dump() function returns the data type and value

PHP NaN

NaN stands for Not a Number.

NaN is used for impossible mathematical operations.

PHP has the following functions to check if a value is not a number:

* [is\_nan()](https://www.w3schools.com/php/func_math_is_nan.asp)

## PHP Numerical Strings

The PHP is\_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise

## PHP Casting Strings and Floats to Integers

Sometimes you need to cast a numerical value into another data type.

The (int), (integer), or intval() function are often used to convert a value to an integer.

# **PHP Math**

## PHP pi() Function

The pi() function returns the value of PI

## PHP min() and max() Functions

The min() and max() functions can be used to find the lowest or highest value in a list of arguments

## PHP abs() Function

The abs() function returns the absolute (positive) value of a number

## PHP sqrt() Function

The sqrt() function returns the square root of a number

## PHP round() Function

The round() function rounds a floating-point number to its nearest intege

## Random Numbers

The rand() function generates a random number. To get more control over the random number, you can add the optional *min* and *max* parameters to specify the lowest integer and the highest integer to be returned. For example, if you want a random integer between 10 and 100 (inclusive), use rand(10, 100).

## PHP Constants

Constants are like variables except that once they are defined, they cannot be changed or undefined. A constant is an identifier (name) for a simple value. The value cannot be changed during the script. A valid constant name starts with a letter or underscore (no $ sign before the constant name). **Note:** Unlike variables, constants are automatically global across the entire script.

## Create a PHP Constant

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

### **Syntax**

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

Parameters:

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

## PHP Constant Arrays

In PHP7, you can create an Array constant using the define() function.

### **Example**

Create an Array constant:

<?php  
define("cars", ["Alfa Romeo","BMW", "Toyota"]);  
echo cars[0];  
?>

## Constants are Global

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

## PHP Operators

## PHP Arithmetic Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** |  |
| + | Addition | $x + $y | Sum of $x and $y |
| - | Subtraction | $x - $y | Difference of $x and $y |
| \* | Multiplication | $x \* $y | Product of $x and $y |
| / | Division | $x / $y | Quotient of $x and $y |
| % | Modulus | $x % $y |  |
| \*\* | Exponentiation | $x \*\* $y | Result of raising $x to the $y'th power |

## PHP Assignment Operators

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

## PHP Increment / Decrement Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| ++$x | Pre-increment | Increments $x by one, then returns $x |
| $x++ | Post-increment | Returns $x, then increments $x by one |
| --$x | Pre-decrement | Decrements $x by one, then returns $x |
| $x-- | Post-decrement | Returns $x, then decrements $x by one |

## PHP Logical Operators

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

## PHP String Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| . | Concatenation | $txt1 . $txt2 | Concatenation of $txt1 and $txt2 |
| .= | Concatenation assignment | $txt1 .= $txt2 | Appends $txt2 to $txt1 |

## PHP Array Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** |  |
| + | Union | $x + $y | Union of $x and $y |
| == | Equality | $x == $y | Returns true if $x and $y have the same key/value pairs |
| === | Identity | $x === $y |  |
| != | Inequality | $x != $y | Returns true if $x is not equal to $y |
| <> | Inequality | $x <> $y | Returns true if $x is not equal to $y |
| !== | Non-identity | $x !== $y | Returns true if $x is not identical to $y |

## PHP Conditional Assignment Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| ?: | Ternary | $x = *expr1* ? *expr2* : *expr3* | Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE |
| ?? | Null coalescing | $x = *expr1* ?? *expr2* | Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7 |

# **PHP if...else...elseif Statements:**

In PHP we have the following conditional statements:

* if statement - executes some code if one condition is true
* if...else statement - executes some code if a condition is true and another code if that condition is false
* if...elseif...else statement - executes different codes for more than two conditions

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

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

### **Syntax**

if (*condition*) {  
  *code to be executed if this condition is true;*} elseif (*condition*) {  
 *code to be executed if first condition is false and this condition is true;*} else {  
  *code to be executed if all conditions are false;*}

## The PHP switch Statement

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

### **Syntax**

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

## PHP Loops

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

In PHP, we have the following loop types:

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

## The PHP while Loop

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

### **Syntax**

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

## The PHP do...while Loop

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

### **Syntax**

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

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

## The PHP for Loop

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

### **Syntax**

for (*init counter; test counter; increment counter*) {  
  *code to be executed for each iteration;*  
}

Parameters:

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

## The PHP foreach Loop

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

### **Syntax**

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

### **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>";  
}  
?>

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>";  
}  
?>

## 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.

## 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.

# **PHP Functions**

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.

Calling the function:

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

## 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.

<?php  
function familyName($fname, $year) {  
  echo "$fname Refsnes. Born in $year <br>";  
}  
familyName("Hege", "1975");  
familyName("Stale", "1978");  
familyName("Kai Jim", "1983");  
?>

## PHP is a Loosely Typed Language

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  
?>

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  
?>

## 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

## PHP Functions - Returning values

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

## 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);  
?>

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

## 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;  
?>

# **PHP Arrays**

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

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

# **PHP Indexed Arrays**

## PHP Indexed Arrays

There are two ways to create indexed arrays:

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

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

or the index can be assigned manually:

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

## 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>";  
}  
?>

## 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";

## 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>";  
}  
?>

# **PHP 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)  
);

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>";  
}  
?>

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

## 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 super global variables are: $GLOBALS, $\_SERVER, $\_REQUEST, $\_POST, $\_GET, $\_FILES, $\_ENV, $\_COOKIE, $\_SESSION.

## 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;  
?>

## PHP $\_SERVER

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

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

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

## PHP $\_REQUEST

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 htmlspecialchars($\_SERVER['PHP\_SELF']);?>">  
  Name: <input type="text" name="fname">  
  <input type="submit">  
</form>  
<?php  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  // collect value of input field  
  $name = $\_REQUEST['fname'];  
  if (empty($name)) {  
    echo "Name is empty";  
  } else {  
    echo $name;  
  }  
}  
?>  
</body>  
</html>

## 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

## 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>

# **PHP Regular Expressions**

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\_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!"  
?>

## 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 = '/\?+/';

# **PHP Form Handling**

## 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>

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

## 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.**

# **PHP Form Validation**

## 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!

## 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.**

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

# **PHP Forms - Required Fields:**

# **PHP Forms - Validate E-mail and URL:**

## PHP - Validate Name

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

## PHP - Validate E-mail

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

## PHP - Validate URL

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

<!DOCTYPE HTML>    
<html>  
<head>  
<style>  
.error {color: #FF0000;}  
</style>  
</head>  
<body>    
<?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  
    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"]);  
  }  
}  
function test\_input($data) {  
  $data = trim($data);  
  $data = stripslashes($data);  
  $data = htmlspecialchars($data);  
  return $data;  
}  
?>  
<h2>PHP Form Validation Example</h2>  
<p><span class="error">\* required field</span></p>  
<form method="post" action="<?php echo htmlspecialchars($\_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>  
<?php  
echo "<h2>Your Input:</h2>";  
echo $name;  
echo "<br>";  
echo $email;  
echo "<br>";  
echo $website;  
echo "<br>";  
echo $comment;  
echo "<br>";  
echo $gender;  
?>  
</body>  
</html>

## 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

Connection between MySQL AND PHP

Insert Data Into MySQL Using MySQLi and PDO

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

* The SQL query must be quoted in PHP
* String values inside the SQL query must be quoted
* Numeric values must not be quoted
* The word NULL must not be quoted
* The INSERT INTO statement is used to add new records to a MySQL table:
* INSERT INTO table\_name (column1, column2, column3,...)  
  VALUES (value1, value2, value3,...)

Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have three main advantages:

* Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
* Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
* Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

## Prepared Statements in MySQLi

The following example uses prepared statements and bound parameters in MySQLi:

### **Example (MySQLi with Prepared Statements)**

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
$dbname = "myDB";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password, $dbname);  
  
// Check connection  
if ($conn->connect\_error) {  
  die("Connection failed: " . $conn->connect\_error);  
}  
  
// prepare and bind  
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");  
$stmt->bind\_param("sss", $firstname, $lastname, $email);  
  
// set parameters and execute  
$firstname = "John";  
$lastname = "Doe";  
$email = "john@example.com";  
$stmt->execute();  
  
$firstname = "Mary";  
$lastname = "Moe";  
$email = "mary@example.com";  
$stmt->execute();  
  
$firstname = "Julie";  
$lastname = "Dooley";  
$email = "julie@example.com";  
$stmt->execute();  
  
echo "New records created successfully";  
  
$stmt->close();  
$conn->close();  
?>

Code lines to explain from the example above:

"INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"

In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.

Then, have a look at the bind\_param() function:

$stmt->bind\_param("sss", $firstname, $lastname, $email);

This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

The argument may be one of four types:

* i - integer
* d - double
* s - string
* b - BLOB

We must have one of these for each parameter.

By telling mysql what type of data to expect, we minimize the risk of SQL injections.

**Note:** If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

## Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column\_name(s) FROM table\_name

or we can use the \* character to select ALL columns from a table:

SELECT \* FROM table\_name

## Select and Filter Data From a MySQL Database

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

SELECT column\_name(s) FROM table\_name WHERE column\_name operator value

## Select and Order Data From a MySQL Database

The ORDER BY clause is used to sort the result-set in ascending or descending order.

The ORDER BY clause sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

SELECT column\_name(s) FROM table\_name ORDER BY column\_name(s) ASC|DESC

## Delete Data From a MySQL Table Using MySQLi and PDO

The DELETE statement is used to delete records from a table:

DELETE FROM table\_name  
WHERE some\_column = some\_value

**Notice the WHERE clause in the DELETE syntax:** The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

## Update Data In a MySQL Table Using MySQLi

The UPDATE statement is used to update existing records in a table:

UPDATE table\_name  
SET column1=value, column2=value2,...  
WHERE some\_column=some\_value

**Notice the WHERE clause in the UPDATE syntax:** The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

## Limit Data Selections From a MySQL Database

MySQL provides a LIMIT clause that is used to specify the number of records to return.

The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.

Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

$sql = "SELECT \* FROM Orders LIMIT 30";

When the SQL query above is run, it will return the first 30 records.

What if we want to select records 16 - 25 (inclusive)?

Mysql also provides a way to handle this: by using OFFSET.

The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

$sql = "SELECT \* FROM Orders LIMIT 10 OFFSET 15";

You could also use a shorter syntax to achieve the same result:

$sql = "SELECT \* FROM Orders LIMIT 15, 10";

Notice that the numbers are reversed when you use a comma.

## 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.

## 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");?>

## 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");  
?>

## 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");  
?>

## 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)

### **Example**

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

## 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);  
?>

# **PHP Include Files**

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>

## 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

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

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 File Handling**

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

## 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");  
?>

## 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);  
?>

|  |  |
| --- | --- |
| **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 Read Single Line - fgets()

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

## HP 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);  
?>

## PHP Read Single Character - fgetc()

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

## 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 File Upload**

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.

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

## 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.";  
  }  
}  
?>

PHP What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

**Tip:** The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

## PHP - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:

## class

Fruit

## objects

Apple

Banana

Mango

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviours from the class, but each object will have different values for the properties.