**VII. Server Side Scripting (Basic)** **7 hours**

PHP: Setup/Getting started, Operators, Control, Array, Function, String operations, Math functions, Using Regx, Exception handling

# PHP Introduction

PHP scripts are executed on the server.

## What You Should Already Know

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

* HTML
* CSS
* JavaScript

If you want to study these subjects first, find the tutorials on our [Home page](http://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

|  |  |
| --- | --- |
| **Note** | **PHP is an amazing and popular language!**  It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)! It is deep enough to run the largest social network (Facebook)! It is also easy enough to be a beginner's first server side language! |

## What is a PHP File?

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

## What Can PHP Do?

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

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

## Why PHP?

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

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.

## Easy Learning with "Show PHP"

Our "Show PHP" tool makes it easy to learn PHP, it shows both the PHP source code and the HTML output of the code.

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
echo "My first PHP script!";  
?>    
  
</body>  
</html>

# PHP 5 Installation

## What Do I Need?

To start using PHP, you can:

* Find a web host with PHP and MySQL support
* Install a web server on your own PC, and then install PHP and MySQL

## Use a Web Host With PHP Support

If your server has activated support for PHP you do not need to do anything.

Just create some .php files, place them in your web directory, and the server will automatically parse them for you.

You do not need to compile anything or install any extra tools.

Because PHP is free, most web hosts offer PHP support.

## Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

* install a web server
* install PHP
* install a database, such as MySQL

The official PHP website (PHP.net) has installation instructions for PHP: <http://php.net/manual/en/install.php>

# PHP 5 Syntax

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

## Basic PHP Syntax

A PHP script can be placed anywhere in the document.

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

<?php  
// PHP code goes here  
?>

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

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

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

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

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

## Comments in PHP

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

Comments can be used to:

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

PHP supports several ways of commenting:

<!DOCTYPE html>  
<html>  
<body>  
    
<?php  
// This is a single-line comment  
  
# This is also a single-line comment  
  
/\*  
This is a multiple-lines comment block  
that spans over multiple  
lines  
\*/  
  
// You can also use comments to leave out parts of a code line  
$x = 5 /\* + 15 \*/ + 5;  
echo $x;  
?>  
    
</body>  
</html>

## PHP Case Sensitivity

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

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

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

However; all variable names are case-sensitive.

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

<!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 5 Variables

Variables are "containers" for storing information.

## Creating (Declaring) PHP Variables

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

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
  
echo $txt;  
echo "<br>";  
echo $x;  
echo "<br>";  
echo $y;  
?>  
  
</body>  
</html>

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

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

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

|  |  |
| --- | --- |
| **Note** | Think of variables as containers for storing data. |

## PHP Variables

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

Rules for PHP variables:

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

|  |  |
| --- | --- |
| **Note** | Remember that PHP variable names are case-sensitive! |

## Output Variables

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

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

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

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

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

The following example will output the sum of two variables:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$x = 5;  
$y = 4;  
echo $x + $y;  
?>  
  
</body>  
</html>

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

## PHP is a Loosely Typed Language

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

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

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

## PHP Variables Scope

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

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

PHP has three different variable scopes:

* local
* global
* static

## Global and Local Scope

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

<?php  
$x = 5; // global scope  
  
function myTest() {  
    // using x inside this function will generate an error  
    echo "<p>Variable x inside function is: $x</p>";  
}   
myTest();  
  
echo "<p>Variable x outside function is: $x</p>";  
?>

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

<?php  
function myTest() {  
    $x = 5; // local scope  
    echo "<p>Variable x inside function is: $x</p>";  
}   
myTest();  
  
// using x outside the function will generate an error  
echo "<p>Variable x outside function is: $x</p>";  
?>

|  |  |
| --- | --- |
| **Note** | 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):

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$x = 5;  
$y = 10;  
  
function myTest() {  
     global $x, $y;  
     $y = $x + $y;  
}   
  
myTest(); // run function  
echo $y; // output the new value for variable $y  
?>  
  
</body>  
</html>

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:

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

## 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:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
function myTest() {  
     static $x = 0;  
     echo $x;  
     $x++;  
}  
  
myTest();  
echo "<br>";  
myTest();  
echo "<br>";  
myTest();  
echo "<br>";  
myTest();  
echo "<br>";  
myTest();  
?>    
  
</body>  
</html>

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

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

# PHP 5 echo and print Statements

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

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

## PHP echo and print Statements

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

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

## The PHP echo Statement

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

**Display Text**

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

## Example

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

**Display Variables**

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

## Example

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
echo "<h2>" . $txt1 . "</h2>";  
echo "Study PHP at " . $txt2 . "<br>";  
echo $x + $y;  
?>  
  
</body>  
</html>

## 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

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

**Display Variables**

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

## Example

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
print "<h2>" . $txt1 . "</h2>";  
print "Study PHP at " . $txt2 . "<br>";  
print $x + $y;  
?>  
  
</body>  
</html>

# PHP 5 Data Types

## PHP Data Types

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

PHP supports the following data types:

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

## PHP String

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

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

## Example

<?php   
$x = "Hello world!";  
$y = 'Hello world!';  
  
echo $x;  
echo "<br>";   
echo $y;

?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_datatypes_string)

## PHP Integer

An integer is a whole number (without decimals).  It is a number between -2,147,483,648 and +2,147,483,647.

Rules for integers:

* An integer must have at least one digit (0-9)
* An integer cannot contain comma or blanks
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

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

## Example

<?php   
$x = 5985;  
var\_dump($x);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_datatypes_integer)

## PHP Float

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

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

## Example

<?php   
$x = 10.365;  
var\_dump($x);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_datatypes_float)

## PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

$x = true;  
$y = false;

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

## PHP Array

An array stores multiple values in one single variable.

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

## Example

<?php   
$cars = array("Volvo","BMW","Toyota");  
var\_dump($cars);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_datatypes_array)

You will learn a lot mo0re about arrays in later chapters of this tutorial.

## PHP Object

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

In PHP, an object must be explicitly declared.

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

## Example

<?php  
class Car {  
    function Car() {  
        $this->model = "VW";  
    }  
}  
  
// create an object  
$herbie = new Car();  
  
// show object properties  
echo $herbie->model;  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_datatypes_object)

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

## PHP NULL Value

The special NULL value represents that a variable has no value. NULL is the only possible value of data type NULL.

The NULL value identifies whether a variable is empty or not. Also useful to differentiate between the empty string and null values of databases.

Variables can be emptied by setting the value to NULL:

## Example

<?php  
$x = "Hello world!";  
$x = null;  
var\_dump($x);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_datatypes_null)

# PHP 5 String Functions

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

## PHP String Functions

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

## The PHP strlen() function

The strlen() function returns the length of a string (number of characters).

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

## Example

<?php  
echo strlen("Hello world!");  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_string_length)

The output of the code above will be: 12.

## The PHP strpos() function

The strpos() function is used to search for a specified character or text within a string.

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

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

## Example

<?php  
echo strpos("Hello world!", "world");  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_string_pos)

The output of the code above will be: 6.

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

# PHP 5 String Functions

## PHP 5 String Functions

The PHP string functions are part of the PHP core. No installation is required to use these functions.

|  |  |
| --- | --- |
| **Function** | **Description** |
| [addcslashes()](http://www.w3schools.com/php/func_string_addcslashes.asp) | Returns a string with backslashes in front of the specified characters |
| [addslashes()](http://www.w3schools.com/php/func_string_addslashes.asp) | Returns a string with backslashes in front of predefined characters |
| [bin2hex()](http://www.w3schools.com/php/func_string_bin2hex.asp) | Converts a string of ASCII characters to hexadecimal values |
| [chop()](http://www.w3schools.com/php/func_string_chop.asp) | Removes whitespace or other characters from the right end of a string |
| [chr()](http://www.w3schools.com/php/func_string_chr.asp) | Returns a character from a specified ASCII value |
| [chunk\_split()](http://www.w3schools.com/php/func_string_chunk_split.asp) | Splits a string into a series of smaller parts |
| [convert\_cyr\_string()](http://www.w3schools.com/php/func_string_convert_cyr_string.asp) | Converts a string from one Cyrillic character-set to another |
| [convert\_uudecode()](http://www.w3schools.com/php/func_string_convert_uudecode.asp) | Decodes a uuencoded string |
| [convert\_uuencode()](http://www.w3schools.com/php/func_string_convert_uuencode.asp) | Encodes a string using the uuencode algorithm |
| [count\_chars()](http://www.w3schools.com/php/func_string_count_chars.asp) | Returns information about characters used in a string |
| [crc32()](http://www.w3schools.com/php/func_string_crc32.asp) | Calculates a 32-bit CRC for a string |
| [crypt()](http://www.w3schools.com/php/func_string_crypt.asp) | One-way string encryption (hashing) |
| [echo()](http://www.w3schools.com/php/func_string_echo.asp) | Outputs one or more strings |
| [explode()](http://www.w3schools.com/php/func_string_explode.asp) | Breaks a string into an array |
| [fprintf()](http://www.w3schools.com/php/func_string_fprintf.asp) | Writes a formatted string to a specified output stream |
| [get\_html\_translation\_table()](http://www.w3schools.com/php/func_string_get_html_translation_table.asp) | Returns the translation table used by htmlspecialchars() and htmlentities() |
| [hebrev()](http://www.w3schools.com/php/func_string_hebrev.asp) | Converts Hebrew text to visual text |
| [hebrevc()](http://www.w3schools.com/php/func_string_hebrevc.asp) | Converts Hebrew text to visual text and new lines (\n) into <br> |
| [hex2bin()](http://www.w3schools.com/php/func_string_hex2bin.asp) | Converts a string of hexadecimal values to ASCII characters |
| [html\_entity\_decode()](http://www.w3schools.com/php/func_string_html_entity_decode.asp) | Converts HTML entities to characters |
| [htmlentities()](http://www.w3schools.com/php/func_string_htmlentities.asp) | Converts characters to HTML entities |
| [htmlspecialchars\_decode()](http://www.w3schools.com/php/func_string_htmlspecialchars_decode.asp) | Converts some predefined HTML entities to characters |
| [htmlspecialchars()](http://www.w3schools.com/php/func_string_htmlspecialchars.asp) | Converts some predefined characters to HTML entities |
| [implode()](http://www.w3schools.com/php/func_string_implode.asp) | Returns a string from the elements of an array |
| [join()](http://www.w3schools.com/php/func_string_join.asp) | Alias of [implode()](http://www.w3schools.com/php/func_string_implode.asp) |
| [lcfirst()](http://www.w3schools.com/php/func_string_lcfirst.asp) | Converts the first character of a string to lowercase |
| [levenshtein()](http://www.w3schools.com/php/func_string_levenshtein.asp) | Returns the Levenshtein distance between two strings |
| [localeconv()](http://www.w3schools.com/php/func_string_localeconv.asp) | Returns locale numeric and monetary formatting information |
| [ltrim()](http://www.w3schools.com/php/func_string_ltrim.asp) | Removes whitespace or other characters from the left side of a string |
| [md5()](http://www.w3schools.com/php/func_string_md5.asp) | Calculates the MD5 hash of a string |
| [md5\_file()](http://www.w3schools.com/php/func_string_md5_file.asp) | Calculates the MD5 hash of a file |
| [metaphone()](http://www.w3schools.com/php/func_string_metaphone.asp) | Calculates the metaphone key of a string |
| [money\_format()](http://www.w3schools.com/php/func_string_money_format.asp) | Returns a string formatted as a currency string |
| [nl\_langinfo()](http://www.w3schools.com/php/func_string_nl_langinfo.asp) | Returns specific local information |
| [nl2br()](http://www.w3schools.com/php/func_string_nl2br.asp) | Inserts HTML line breaks in front of each newline in a string |
| [number\_format()](http://www.w3schools.com/php/func_string_number_format.asp) | Formats a number with grouped thousands |
| [ord()](http://www.w3schools.com/php/func_string_ord.asp) | Returns the ASCII value of the first character of a string |
| [parse\_str()](http://www.w3schools.com/php/func_string_parse_str.asp) | Parses a query string into variables |
| [print()](http://www.w3schools.com/php/func_string_print.asp) | Outputs one or more strings |
| [printf()](http://www.w3schools.com/php/func_string_printf.asp) | Outputs a formatted string |
| [quoted\_printable\_decode()](http://www.w3schools.com/php/func_string_quoted_printable_decode.asp) | Converts a quoted-printable string to an 8-bit string |
| [quoted\_printable\_encode()](http://www.w3schools.com/php/func_string_quoted_printable_encode.asp) | Converts an 8-bit string to a quoted printable string |
| [quotemeta()](http://www.w3schools.com/php/func_string_quotemeta.asp) | Quotes meta characters |
| [rtrim()](http://www.w3schools.com/php/func_string_rtrim.asp) | Removes whitespace or other characters from the right side of a string |
| [setlocale()](http://www.w3schools.com/php/func_string_setlocale.asp) | Sets locale information |
| [sha1()](http://www.w3schools.com/php/func_string_sha1.asp) | Calculates the SHA-1 hash of a string |
| [sha1\_file()](http://www.w3schools.com/php/func_string_sha1_file.asp) | Calculates the SHA-1 hash of a file |
| [similar\_text()](http://www.w3schools.com/php/func_string_similar_text.asp) | Calculates the similarity between two strings |
| [soundex()](http://www.w3schools.com/php/func_string_soundex.asp) | Calculates the soundex key of a string |
| [sprintf()](http://www.w3schools.com/php/func_string_sprintf.asp) | Writes a formatted string to a variable |
| [sscanf()](http://www.w3schools.com/php/func_string_sscanf.asp) | Parses input from a string according to a format |
| [str\_getcsv()](http://www.w3schools.com/php/func_string_str_getcsv.asp) | Parses a CSV string into an array |
| [str\_ireplace()](http://www.w3schools.com/php/func_string_str_ireplace.asp) | Replaces some characters in a string (case-insensitive) |
| [str\_pad()](http://www.w3schools.com/php/func_string_str_pad.asp) | Pads a string to a new length |
| [str\_repeat()](http://www.w3schools.com/php/func_string_str_repeat.asp) | Repeats a string a specified number of times |
| [str\_replace()](http://www.w3schools.com/php/func_string_str_replace.asp) | Replaces some characters in a string (case-sensitive) |
| [str\_rot13()](http://www.w3schools.com/php/func_string_str_rot13.asp) | Performs the ROT13 encoding on a string |
| [str\_shuffle()](http://www.w3schools.com/php/func_string_str_shuffle.asp) | Randomly shuffles all characters in a string |
| [str\_split()](http://www.w3schools.com/php/func_string_str_split.asp) | Splits a string into an array |
| [str\_word\_count()](http://www.w3schools.com/php/func_string_str_word_count.asp) | Count the number of words in a string |
| [strcasecmp()](http://www.w3schools.com/php/func_string_strcasecmp.asp) | Compares two strings (case-insensitive) |
| [strchr()](http://www.w3schools.com/php/func_string_strchr.asp) | Finds the first occurrence of a string inside another string (alias of strstr()) |
| [strcmp()](http://www.w3schools.com/php/func_string_strcmp.asp) | Compares two strings (case-sensitive) |
| [strcoll()](http://www.w3schools.com/php/func_string_strcoll.asp) | Compares two strings (locale based string comparison) |
| [strcspn()](http://www.w3schools.com/php/func_string_strcspn.asp) | Returns the number of characters found in a string before any part of some specified characters are found |
| [strip\_tags()](http://www.w3schools.com/php/func_string_strip_tags.asp) | Strips HTML and PHP tags from a string |
| [stripcslashes()](http://www.w3schools.com/php/func_string_stripcslashes.asp) | Unquotes a string quoted with addcslashes() |
| [stripslashes()](http://www.w3schools.com/php/func_string_stripslashes.asp) | Unquotes a string quoted with addslashes() |
| [stripos()](http://www.w3schools.com/php/func_string_stripos.asp) | Returns the position of the first occurrence of a string inside another string (case-insensitive) |
| [stristr()](http://www.w3schools.com/php/func_string_stristr.asp) | Finds the first occurrence of a string inside another string (case-insensitive) |
| [strlen()](http://www.w3schools.com/php/func_string_strlen.asp) | Returns the length of a string |
| [strnatcasecmp()](http://www.w3schools.com/php/func_string_strnatcasecmp.asp) | Compares two strings using a "natural order" algorithm (case-insensitive) |
| [strnatcmp()](http://www.w3schools.com/php/func_string_strnatcmp.asp) | Compares two strings using a "natural order" algorithm (case-sensitive) |
| [strncasecmp()](http://www.w3schools.com/php/func_string_strncasecmp.asp) | String comparison of the first n characters (case-insensitive) |
| [strncmp()](http://www.w3schools.com/php/func_string_strncmp.asp) | String comparison of the first n characters (case-sensitive) |
| [strpbrk()](http://www.w3schools.com/php/func_string_strpbrk.asp) | Searches a string for any of a set of characters |
| [strpos()](http://www.w3schools.com/php/func_string_strpos.asp) | Returns the position of the first occurrence of a string inside another string (case-sensitive) |
| [strrchr()](http://www.w3schools.com/php/func_string_strrchr.asp) | Finds the last occurrence of a string inside another string |
| [strrev()](http://www.w3schools.com/php/func_string_strrev.asp) | Reverses a string |
| [strripos()](http://www.w3schools.com/php/func_string_strripos.asp) | Finds the position of the last occurrence of a string inside another string (case-insensitive) |
| [strrpos()](http://www.w3schools.com/php/func_string_strrpos.asp) | Finds the position of the last occurrence of a string inside another string (case-sensitive) |
| [strspn()](http://www.w3schools.com/php/func_string_strspn.asp) | Returns the number of characters found in a string that contains only characters from a specified charlist |
| [strstr()](http://www.w3schools.com/php/func_string_strstr.asp) | Finds the first occurrence of a string inside another string (case-sensitive) |
| [strtok()](http://www.w3schools.com/php/func_string_strtok.asp) | Splits a string into smaller strings |
| [strtolower()](http://www.w3schools.com/php/func_string_strtolower.asp) | Converts a string to lowercase letters |
| [strtoupper()](http://www.w3schools.com/php/func_string_strtoupper.asp) | Converts a string to uppercase letters |
| [strtr()](http://www.w3schools.com/php/func_string_strtr.asp) | Translates certain characters in a string |
| [substr()](http://www.w3schools.com/php/func_string_substr.asp) | Returns a part of a string |
| [substr\_compare()](http://www.w3schools.com/php/func_string_substr_compare.asp) | Compares two strings from a specified start position (binary safe and optionally case-sensitive) |
| [substr\_count()](http://www.w3schools.com/php/func_string_substr_count.asp) | Counts the number of times a substring occurs in a string |
| [substr\_replace()](http://www.w3schools.com/php/func_string_substr_replace.asp) | Replaces a part of a string with another string |
| [trim()](http://www.w3schools.com/php/func_string_trim.asp) | Removes whitespace or other characters from both sides of a string |
| [ucfirst()](http://www.w3schools.com/php/func_string_ucfirst.asp) | Converts the first character of a string to uppercase |
| [ucwords()](http://www.w3schools.com/php/func_string_ucwords.asp) | Converts the first character of each word in a string to uppercase |
| [vfprintf()](http://www.w3schools.com/php/func_string_vfprintf.asp) | Writes a formatted string to a specified output stream |
| [vprintf()](http://www.w3schools.com/php/func_string_vprintf.asp) | Outputs a formatted string |
| [vsprintf()](http://www.w3schools.com/php/func_string_vsprintf.asp) | Writes a formatted string to a variable |
| [wordwrap()](http://www.w3schools.com/php/func_string_wordwrap.asp) | Wraps a string to a given number of characters |

# PHP 5 Constants

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

## PHP Constants

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

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

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

## Create a PHP Constant

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

### **Syntax**

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

Parameters:

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_constant1)

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_constant2)

## Constants are Global

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

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

## Example

<?php  
define("GREETING", "Welcome to W3Schools.com!");  
  
function myTest() {  
    echo GREETING;  
}  
   
myTest();  
?>

# PHP 5 Operators

This chapter shows the different operators that can be used in PHP scripts.

## PHP Arithmetic Operators

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

Here is a complete list of PHP's arithmetic operators:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** | **Show it** |
| + | 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 | Remainder of $x divided by $y |  |
| \*\* | Exponentiation | $x \*\* $y | Result of raising $x to the $y'th power (Introduced in PHP 5.6) |  |

## PHP Assignment Operators

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment** | **Same as...** | **Description** | **Show it** |
| 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 String Operators

The string operators are used with strings.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| . | Concatenation | $txt1 = "Hello" $txt2 = $txt1 . " world!" | Now $txt2 contains "Hello world!" |
| .= | Concatenation assignment | $txt1 = "Hello" $txt1 .= " world!" | Now $txt1 contains "Hello world!" |

The example below shows the results of using the string operators:

## Example

<?php  
$a = "Hello";  
$b = $a . " world!";  
echo $b; // outputs Hello world!   
  
$x = "Hello";  
$x .= " world!";  
echo $x; // outputs Hello world!   
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_operator_string)

## 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 |

The example below shows the different results of using the different increment/decrement operators:

## Example

<?php  
$x = 10;   
echo ++$x; // outputs 11  
  
$y = 10;   
echo $y++; // outputs 10  
  
$z = 5;  
echo --$z; // outputs 4  
  
$i = 5;  
echo $i--; // outputs 5  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_operator_increment)

## PHP Comparison Operators

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| == | Equal | $x == $y | True if $x is equal to $y |
| === | Identical | $x === $y | True if $x is equal to $y, and they are of the same type |
| != | Not equal | $x != $y | True if $x is not equal to $y |
| <> | Not equal | $x <> $y | True if $x is not equal to $y |
| !== | Not identical | $x !== $y | True if $x is not equal to $y, or they are not of the same type |
| > | Greater than | $x > $y | True if $x is greater than $y |
| < | Less than | $x < $y | True if $x is less than $y |
| >= | Greater than or equal to | $x >= $y | True if $x is greater than or equal to $y |
| <= | Less than or equal to | $x <= $y | True if $x is less than or equal to $y |

The example below shows the different results of using some of the comparison operators:

## Example

<?php  
$x = 100;   
$y = "100";  
  
var\_dump($x == $y);  
echo "<br>";  
var\_dump($x === $y);  
echo "<br>";  
var\_dump($x != $y);  
echo "<br>";  
var\_dump($x !== $y);  
echo "<br>";  
  
$a = 50;  
$b = 90;  
  
var\_dump($a > $b);  
echo "<br>";  
var\_dump($a < $b);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_operator_comparison)

## 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 | True if either $x or $y is true |
| ! | Not | !$x | True if $x is not true |

## PHP Array Operators

The PHP array operators are used to compare arrays:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| + | Union | $x + $y | Union of $x and $y (but duplicate keys are not overwritten) |
| == | Equality | $x == $y | True if $x and $y have the same key/value pairs |
| === | Identity | $x === $y | True if $x and $y have the same key/value pairs in the same order and of the same types |
| != | Inequality | $x != $y | True if $x is not equal to $y |
| <> | Inequality | $x <> $y | True if $x is not equal to $y |
| !== | Non-identity | $x !== $y | True if $x is not identical to $y |

The example below shows the different results of using the different array operators:

## Example

<?php  
$x = array("a" => "red", "b" => "green");   
$y = array("c" => "blue", "d" => "yellow");   
$z = $x + $y; // union of $x and $y  
  
var\_dump($z);  
var\_dump($x == $y);  
var\_dump($x === $y);  
var\_dump($x != $y);  
var\_dump($x <> $y);  
var\_dump($x !== $y);  
?>

# PHP 5 if...else...elseif Statements

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

## PHP Conditional Statements

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

In PHP we have the following conditional statements:

* **if statement** - executes some code only if a specified condition is true
* **if...else statement** - executes some code if a condition is true and another code if the condition is false
* **if...elseif....else statement** - specifies a new condition to test if the first condition is false
* **switch statement** - selects one of many blocks of code to be executed

## PHP - The if Statement

The if statement is used to execute some code **only if a specified condition is true**.

### **Syntax**

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_if)

## PHP - The if...else Statement

Use the if....else statement to execute some code **if a condition is true and another code if the condition is false**.

### **Syntax**

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_if_else)

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

Use the if....elseif...else statement to **specify a new condition to test if the first condition is false.**

### **Syntax**

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_if_elseif)

# PHP 5 switch Statement

## The PHP switch Statement

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

### **Syntax**

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

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

## Example

<?php  
$favcolor = "red";  
  
switch ($favcolor) {  
    case "red":  
        echo "Your favorite color is red!";  
        break;  
    case "blue":  
        echo "Your favorite color is blue!";  
        break;  
    case "green":  
        echo "Your favorite color is green!";  
        break;  
    default:  
        echo "Your favorite color is neither red, blue, or green!";  
}  
?>

# PHP 5 while Loops

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

## PHP Loops

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

In PHP, we have the following looping statements:

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

## The PHP while Loop

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

### **Syntax**

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_loop_while)

## The PHP do...while Loop

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

### **Syntax**

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

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

## Example

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

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

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

## Example

<?php   
$x = 6;  
  
do {  
    echo "The number is: $x <br>";  
    $x++;  
} while ($x<=5);  
?>

# PHP 5 for Loops

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

## The PHP for Loop

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

### **Syntax**

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

Parameters:

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

The example below displays the numbers from 0 to 10:

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_loop_for)

## The PHP foreach Loop

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

### **Syntax**

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

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

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

## Example

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

# PHP 5 Functions

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

## PHP User Defined Functions

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

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

A function will not execute immediately when a page loads.

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

## Create a User Defined Function in PHP

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

### **Syntax**

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

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

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

|  |  |
| --- | --- |
| **Note** | Function names are NOT case-sensitive. |

In the example below, we create a function named "writeMsg()". The opening curly brace ( { ) indicates the beginning of the function code and the closing curly brace ( } ) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_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 seperate 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 Roshan.<br>";  
}  
  
familyName("Jani");  
familyName("Hege");  
familyName("Stale");  
familyName("Kai Jim");  
familyName("Borge");  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_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");  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_function3)

## PHP Default Argument Value

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

## Example

<?php  
function setHeight($minheight = 50) {  
    echo "The height is : $minheight <br>";  
}  
  
setHeight(350);  
setHeight(); // will use the default value of 50  
setHeight(135);  
setHeight(80);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_function4)

## PHP Functions - Returning values

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

## Example

<?php  
function sum($x, $y) {  
    $z = $x + $y;  
    return $z;  
}  
  
echo "5 + 10 = " . sum(5, 10) . "<br>";  
echo "7 + 13 = " . sum(7, 13) . "<br>";  
echo "2 + 4 = " . sum(2, 4);  
?>

# PHP 5 Arrays

An array stores multiple values in one single variable:

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_num)

## What is an Array?

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

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

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

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

The solution is to create an array!

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

## Create an Array in PHP

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

array();

In PHP, there are three types of arrays:

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

## PHP Indexed Arrays

There are two ways to create indexed arrays:

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

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

or the index can be assigned manually:

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_num)

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_length)

## Loop Through an Indexed Array

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

## Example

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
$arrlength = count($cars);  
  
for($x = 0; $x < $arrlength; $x++) {  
    echo $cars[$x];  
    echo "<br>";  
}  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_num_loop)

## 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.";  
?>

## 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 5 Array Functions

## PHP Array Introduction

The array functions allow you to access and manipulate arrays.

Simple and multi-dimensional arrays are supported.

## Installation

The array functions are part of the PHP core. There is no installation needed to use these functions.

## PHP 5 Array Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [array()](http://www.w3schools.com/php/func_array.asp) | Creates an array |
| [array\_change\_key\_case()](http://www.w3schools.com/php/func_array_change_key_case.asp) | Changes all keys in an array to lowercase or uppercase |
| [array\_chunk()](http://www.w3schools.com/php/func_array_chunk.asp) | Splits an array into chunks of arrays |
| [array\_column()](http://www.w3schools.com/php/func_array_column.asp) | Returns the values from a single column in the input array |
| [array\_combine()](http://www.w3schools.com/php/func_array_combine.asp) | Creates an array by using the elements from one "keys" array and one "values" array |
| [array\_count\_values()](http://www.w3schools.com/php/func_array_count_values.asp) | Counts all the values of an array |
| [array\_diff()](http://www.w3schools.com/php/func_array_diff.asp) | Compare arrays, and returns the differences (compare values only) |
| [array\_diff\_assoc()](http://www.w3schools.com/php/func_array_diff_assoc.asp) | Compare arrays, and returns the differences (compare keys and values) |
| [array\_diff\_key()](http://www.w3schools.com/php/func_array_diff_key.asp) | Compare arrays, and returns the differences (compare keys only) |
| [array\_diff\_uassoc()](http://www.w3schools.com/php/func_array_diff_uassoc.asp) | Compare arrays, and returns the differences (compare keys and values, using a user-defined key comparison function) |
| [array\_diff\_ukey()](http://www.w3schools.com/php/func_array_diff_ukey.asp) | Compare arrays, and returns the differences (compare keys only, using a user-defined key comparison function) |
| [array\_fill()](http://www.w3schools.com/php/func_array_fill.asp) | Fills an array with values |
| [array\_fill\_keys()](http://www.w3schools.com/php/func_array_fill_keys.asp) | Fills an array with values, specifying keys |
| [array\_filter()](http://www.w3schools.com/php/func_array_filter.asp) | Filters the values of an array using a callback function |
| [array\_flip()](http://www.w3schools.com/php/func_array_flip.asp) | Flips/Exchanges all keys with their associated values in an array |
| [array\_intersect()](http://www.w3schools.com/php/func_array_intersect.asp) | Compare arrays, and returns the matches (compare values only) |
| [array\_intersect\_assoc()](http://www.w3schools.com/php/func_array_intersect_assoc.asp) | Compare arrays and returns the matches (compare keys and values) |
| [array\_intersect\_key()](http://www.w3schools.com/php/func_array_intersect_key.asp) | Compare arrays, and returns the matches (compare keys only) |
| [array\_intersect\_uassoc()](http://www.w3schools.com/php/func_array_intersect_uassoc.asp) | Compare arrays, and returns the matches (compare keys and values, using a user-defined key comparison function) |
| [array\_intersect\_ukey()](http://www.w3schools.com/php/func_array_intersect_ukey.asp) | Compare arrays, and returns the matches (compare keys only, using a user-defined key comparison function) |
| [array\_key\_exists()](http://www.w3schools.com/php/func_array_key_exists.asp) | Checks if the specified key exists in the array |
| [array\_keys()](http://www.w3schools.com/php/func_array_keys.asp) | Returns all the keys of an array |
| [array\_map()](http://www.w3schools.com/php/func_array_map.asp) | Sends each value of an array to a user-made function, which returns new values |
| [array\_merge()](http://www.w3schools.com/php/func_array_merge.asp) | Merges one or more arrays into one array |
| [array\_merge\_recursive()](http://www.w3schools.com/php/func_array_merge_recursive.asp) | Merges one or more arrays into one array recursively |
| [array\_multisort()](http://www.w3schools.com/php/func_array_multisort.asp) | Sorts multiple or multi-dimensional arrays |
| [array\_pad()](http://www.w3schools.com/php/func_array_pad.asp) | Inserts a specified number of items, with a specified value, to an array |
| [array\_pop()](http://www.w3schools.com/php/func_array_pop.asp) | Deletes the last element of an array |
| [array\_product()](http://www.w3schools.com/php/func_array_product.asp) | Calculates the product of the values in an array |
| [array\_push()](http://www.w3schools.com/php/func_array_push.asp) | Inserts one or more elements to the end of an array |
| [array\_rand()](http://www.w3schools.com/php/func_array_rand.asp) | Returns one or more random keys from an array |
| [array\_reduce()](http://www.w3schools.com/php/func_array_reduce.asp) | Returns an array as a string, using a user-defined function |
| [array\_replace()](http://www.w3schools.com/php/func_array_replace.asp) | Replaces the values of the first array with the values from following arrays |
| [array\_replace\_recursive()](http://www.w3schools.com/php/func_array_replace_recursive.asp) | Replaces the values of the first array with the values from following arrays recursively |
| [array\_reverse()](http://www.w3schools.com/php/func_array_reverse.asp) | Returns an array in the reverse order |
| [array\_search()](http://www.w3schools.com/php/func_array_search.asp) | Searches an array for a given value and returns the key |
| [array\_shift()](http://www.w3schools.com/php/func_array_shift.asp) | Removes the first element from an array, and returns the value of the removed element |
| [array\_slice()](http://www.w3schools.com/php/func_array_slice.asp) | Returns selected parts of an array |
| [array\_splice()](http://www.w3schools.com/php/func_array_splice.asp) | Removes and replaces specified elements of an array |
| [array\_sum()](http://www.w3schools.com/php/func_array_sum.asp) | Returns the sum of the values in an array |
| [array\_udiff()](http://www.w3schools.com/php/func_array_udiff.asp) | Compare arrays, and returns the differences (compare values only, using a user-defined key comparison function) |
| [array\_udiff\_assoc()](http://www.w3schools.com/php/func_array_udiff_assoc.asp) | Compare arrays, and returns the differences (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values) |
| [array\_udiff\_uassoc()](http://www.w3schools.com/php/func_array_udiff_uassoc.asp) | Compare arrays, and returns the differences (compare keys and values, using two user-defined key comparison functions) |
| [array\_uintersect()](http://www.w3schools.com/php/func_array_uintersect.asp) | Compare arrays, and returns the matches (compare values only, using a user-defined key comparison function) |
| [array\_uintersect\_assoc()](http://www.w3schools.com/php/func_array_uintersect_assoc.asp) | Compare arrays, and returns the matches (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values) |
| [array\_uintersect\_uassoc()](http://www.w3schools.com/php/func_array_uintersect_uassoc.asp) | Compare arrays, and returns the matches (compare keys and values, using two user-defined key comparison functions) |
| [array\_unique()](http://www.w3schools.com/php/func_array_unique.asp) | Removes duplicate values from an array |
| [array\_unshift()](http://www.w3schools.com/php/func_array_unshift.asp) | Adds one or more elements to the beginning of an array |
| [array\_values()](http://www.w3schools.com/php/func_array_values.asp) | Returns all the values of an array |
| [array\_walk()](http://www.w3schools.com/php/func_array_walk.asp) | Applies a user function to every member of an array |
| [array\_walk\_recursive()](http://www.w3schools.com/php/func_array_walk_recursive.asp) | Applies a user function recursively to every member of an array |
| [arsort()](http://www.w3schools.com/php/func_array_arsort.asp) | Sorts an associative array in descending order, according to the value |
| [asort()](http://www.w3schools.com/php/func_array_asort.asp) | Sorts an associative array in ascending order, according to the value |
| [compact()](http://www.w3schools.com/php/func_array_compact.asp) | Create array containing variables and their values |
| [count()](http://www.w3schools.com/php/func_array_count.asp) | Returns the number of elements in an array |
| [current()](http://www.w3schools.com/php/func_array_current.asp) | Returns the current element in an array |
| [each()](http://www.w3schools.com/php/func_array_each.asp) | Returns the current key and value pair from an array |
| [end()](http://www.w3schools.com/php/func_array_end.asp) | Sets the internal pointer of an array to its last element |
| [extract()](http://www.w3schools.com/php/func_array_extract.asp) | Imports variables into the current symbol table from an array |
| [in\_array()](http://www.w3schools.com/php/func_array_in_array.asp) | Checks if a specified value exists in an array |
| [key()](http://www.w3schools.com/php/func_array_key.asp) | Fetches a key from an array |
| [krsort()](http://www.w3schools.com/php/func_array_krsort.asp) | Sorts an associative array in descending order, according to the key |
| [ksort()](http://www.w3schools.com/php/func_array_ksort.asp) | Sorts an associative array in ascending order, according to the key |
| [list()](http://www.w3schools.com/php/func_array_list.asp) | Assigns variables as if they were an array |
| [natcasesort()](http://www.w3schools.com/php/func_array_natcasesort.asp) | Sorts an array using a case insensitive "natural order" algorithm |
| [natsort()](http://www.w3schools.com/php/func_array_natsort.asp) | Sorts an array using a "natural order" algorithm |
| [next()](http://www.w3schools.com/php/func_array_next.asp) | Advance the internal array pointer of an array |
| [pos()](http://www.w3schools.com/php/func_array_pos.asp) | Alias of [current()](http://www.w3schools.com/php/func_array_current.asp) |
| [prev()](http://www.w3schools.com/php/func_array_prev.asp) | Rewinds the internal array pointer |
| [range()](http://www.w3schools.com/php/func_array_range.asp) | Creates an array containing a range of elements |
| [reset()](http://www.w3schools.com/php/func_array_reset.asp) | Sets the internal pointer of an array to its first element |
| [rsort()](http://www.w3schools.com/php/func_array_rsort.asp) | Sorts an indexed array in descending order |
| [shuffle()](http://www.w3schools.com/php/func_array_shuffle.asp) | Shuffles an array |
| [sizeof()](http://www.w3schools.com/php/func_array_sizeof.asp) | Alias of [count()](http://www.w3schools.com/php/func_array_count.asp) |
| [sort()](http://www.w3schools.com/php/func_array_sort.asp) | Sorts an indexed array in ascending order |
| [uasort()](http://www.w3schools.com/php/func_array_uasort.asp) | Sorts an array by values using a user-defined comparison function |
| [uksort()](http://www.w3schools.com/php/func_array_uksort.asp) | Sorts an array by keys using a user-defined comparison function |
| [usort()](http://www.w3schools.com/php/func_array_usort.asp) | Sorts an array using a user-defined comparison function |

# PHP 5 Sorting Arrays

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

## PHP - Sort Functions For Arrays

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

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

## Sort Array in Ascending Order - sort()

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_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);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_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);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_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);  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_rsort_num)

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_asort)

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_ksort)

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_array_arsort)

## Sort Array in 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);  
?>

# PHP 5 Global Variables - Superglobals

## PHP Global Variables - Superglobals

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

The PHP superglobal variables are:

* $GLOBALS
* $\_SERVER
* $\_REQUEST
* $\_POST
* $\_GET
* $\_FILES
* $\_ENV
* $\_COOKIE
* $\_SESSION

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

## PHP $GLOBALS

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

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

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

## Example

<?php   
$x = 75;   
$y = 25;  
   
function addition() {   
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];   
}  
   
addition();   
echo $z;   
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_global_global)

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

## PHP $\_SERVER

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

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

## Example

<?php   
echo $\_SERVER['PHP\_SELF'];  
echo "<br>";  
echo $\_SERVER['SERVER\_NAME'];  
echo "<br>";  
echo $\_SERVER['HTTP\_HOST'];  
echo "<br>";  
echo $\_SERVER['HTTP\_REFERER'];  
echo "<br>";  
echo $\_SERVER['HTTP\_USER\_AGENT'];  
echo "<br>";  
echo $\_SERVER['SCRIPT\_NAME'];  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?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 |

## PHP $\_REQUEST

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_global_request)

## PHP $\_POST

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_global_post)

## PHP $\_GET

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

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

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

<html>  
<body>  
  
<a href="test\_get.php?subject=PHP&web=W3schools.com">Test $GET</a>  
  
</body>  
</html>

When a user clicks on the link "Test $GET", the parameters "subject" and "web" is sent to "test\_get.php", and you can then acces their values in "test\_get.php" with $\_GET.

The example below shows the code in "test\_get.php":

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_global_get)

## PHP Global Variables - Superglobals

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

The PHP superglobal variables are:

* $GLOBALS
* $\_SERVER
* $\_REQUEST
* $\_POST
* $\_GET
* $\_FILES
* $\_ENV
* $\_COOKIE
* $\_SESSION

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

## PHP $GLOBALS

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

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

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

## Example

<?php   
$x = 75;   
$y = 25;  
   
function addition() {   
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];   
}  
   
addition();   
echo $z;   
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_global_global)

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

## PHP $\_SERVER

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

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

## Example

<?php   
echo $\_SERVER['PHP\_SELF'];  
echo "<br>";  
echo $\_SERVER['SERVER\_NAME'];  
echo "<br>";  
echo $\_SERVER['HTTP\_HOST'];  
echo "<br>";  
echo $\_SERVER['HTTP\_REFERER'];  
echo "<br>";  
echo $\_SERVER['HTTP\_USER\_AGENT'];  
echo "<br>";  
echo $\_SERVER['SCRIPT\_NAME'];  
?>

[**Run example »**](http://www.w3schools.com/php/showphp.asp?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 |

## PHP $\_REQUEST

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

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

## Example

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

## PHP $\_POST

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

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

## Example

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

[**Run example »**](http://www.w3schools.com/php/showphp.asp?filename=demo_global_post)

## PHP $\_GET

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

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

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

<html>  
<body>  
  
<a href="test\_get.php?subject=PHP&web=W3schools.com">Test $GET</a>  
  
</body>  
</html>

When a user clicks on the link "Test $GET", the parameters "subject" and "web" is sent to "test\_get.php", and you can then acces 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 Exception Handling**

Exceptions are used to change the normal flow of a script if a specified error occurs.

**What is an Exception?**

With PHP 5 came a new object oriented way of dealing with errors.

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

This is what normally happens when an exception is triggered:

* The current code state is saved
* The code execution will switch to a predefined (custom) exception handler function
* Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

**We will show different error handling methods:**

* Basic use of Exceptions
* Creating a custom exception handler
* Multiple exceptions
* Re-throwing an exception
* Setting a top level exception handler

**Note:** Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

**Basic Use of Exceptions**

When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.

If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

**Lets try to throw an exception without catching it:**

<?php  
//create function with an exception  
function checkNum($number) {  
  if($number>1) {  
    throw new Exception("Value must be 1 or below");  
  }  
  return true;  
}  
  
//trigger exception  
checkNum(2);  
?>

The code above will get an error like this:

**Fatal error**: Uncaught exception 'Exception'  
with message 'Value must be 1 or below' in C:\webfolder\test.php:6  
Stack trace: #0 C:\webfolder\test.php(12):  
checkNum(28) #1 {main} thrown in **C:\webfolder\test.php** on line **6**

Try, throw and catch

To avoid the error from the example above, we need to create the proper code to handle an exception.

Proper exception code should include:

1. try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
2. throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
3. catch - A "catch" block retrieves an exception and creates an object containing the exception information

Lets try to trigger an exception with valid code:

<?php  
//create function with an exception  
function checkNum($number) {  
  if($number>1) {  
    throw new Exception("Value must be 1 or below");  
  }  
  return true;  
}  
  
//trigger exception in a "try" block  
try {  
  checkNum(2);  
  //If the exception is thrown, this text will not be shown  
  echo 'If you see this, the number is 1 or below';  
}  
  
//catch exception  
catch(Exception $e) {  
  echo 'Message: ' .$e->getMessage();  
}  
?>

The code above will get an error like this:

Message: Value must be 1 or below

Example explained:

The code above throws an exception and catches it:

1. The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
2. The checkNum() function is called in a "try" block
3. The exception within the checkNum() function is thrown
4. The "catch" block retrieves the exception and creates an object ($e) containing the exception information
5. The error message from the exception is echoed by calling $e->getMessage() from the exception object

However, one way to get around the "every throw must have a catch" rule is to set a top level exception handler to handle errors that slip through.

Creating a Custom Exception Class

To create a custom exception handler you must create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()  
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example...com";  
  
try {  
  //check if  
  if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE) {  
    //throw exception if email is not valid  
    throw new customException($email);  
  }  
}  
  
catch (customException $e) {  
  //display custom message  
  echo $e->errorMessage();  
}  
?>

The new class is a copy of the old exception class with an addition of the errorMessage() function. Since it is a copy of the old class, and it inherits the properties and methods from the old class, we can use the exception class methods like getLine() and getFile() and getMessage().

Example explained:

The code above throws an exception and catches it with a custom exception class:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is not a valid e-mail address
4. The "try" block is executed and an exception is thrown since the e-mail address is invalid
5. The "catch" block catches the exception and displays the error message

Multiple Exceptions

It is possible for a script to use multiple exceptions to check for multiple conditions.

It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()  
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example.com";  
  
try {  
  //check if  
  if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE) {  
    //throw exception if email is not valid  
    throw new customException($email);  
  }  
  //check for "example" in mail address  
  if(strpos($email, "example") !== FALSE) {  
    throw new Exception("$email is an example e-mail");  
  }  
}  
  
catch (customException $e) {  
  echo $e->errorMessage();  
}  
  
catch(Exception $e) {  
  echo $e->getMessage();  
}  
?>

Example explained:

The code above tests two conditions and throws an exception if any of the conditions are not met:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block is executed and an exception is not thrown on the first condition
5. The second condition triggers an exception since the e-mail contains the string "example"
6. The "catch" block catches the exception and displays the correct error message

If the exception thrown were of the class customException and there were no customException catch, only the base exception catch, the exception would be handled there.

Re-throwing Exceptions

Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.

A script should hide system errors from users. System errors may be important for the coder, but are of no interest to the user. To make things easier for the user you can re-throw the exception with a user friendly message:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = $this->getMessage().' is not a valid E-Mail address.';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example.com";  
  
try {  
  try {  
    //check for "example" in mail address  
    if(strpos($email, "example") !== FALSE) {  
      //throw exception if email is not valid  
      throw new Exception($email);  
    }  
  }  
  catch(Exception $e) {  
    //re-throw exception  
    throw new customException($email);  
  }  
}  
  
catch (customException $e) {  
  //display custom message  
  echo $e->errorMessage();  
}  
?>

Example explained:

The code above tests if the email-address contains the string "example" in it, if it does, the exception is re-thrown:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block contains another "try" block to make it possible to re-throw the exception
5. The exception is triggered since the e-mail contains the string "example"
6. The "catch" block catches the exception and re-throws a "customException"
7. The "customException" is caught and displays an error message

If the exception is not caught in its current "try" block, it will search for a catch block on "higher levels".

Set a Top Level Exception Handler

The set\_exception\_handler() function sets a user-defined function to handle all uncaught exceptions:

<?php  
function myException($exception) {  
  echo "<b>Exception:</b> " . $exception->getMessage();  
}  
  
set\_exception\_handler('myException');  
  
throw new Exception('Uncaught Exception occurred');  
?>

The output of the code above should be something like this:

**Exception:** Uncaught Exception occurred

In the code above there was no "catch" block. Instead, the top level exception handler triggered. This function should be used to catch uncaught exceptions.

Rules for exceptions

* Code may be surrounded in a try block, to help catch potential exceptions
* Each try block or "throw" must have at least one corresponding catch block
* Multiple catch blocks can be used to catch different classes of exceptions
* Exceptions can be thrown (or re-thrown) in a catch block within a try block

A simple rule: If you throw something, you have to catch it.

**Using Regx**

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.

Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.

* POSIX Regular Expressions
* PERL Style Regular Expressions

## POSIX Regular Expressions

The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

The simplest regular expression is one that matches a single character, such as g, inside strings such as g, haggle, or bag.

Lets give explanation for few concepts being used in POSIX regular expression. After that we will introduce you with regular expression related functions.

### **Brackets**

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **[0-9]**  It matches any decimal digit from 0 through 9. |
| 2 | **[a-z]**  It matches any character from lower-case a through lowercase z. |
| 3 | **[A-Z]**  It matches any character from uppercase A through uppercase Z. |
| 4 | **[a-Z]**  It matches any character from lowercase a through uppercase Z. |

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

### **Quantifiers**

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, \*, ?, {int. range}, and $ flags all follow a character sequence.

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **p+**  It matches any string containing at least one p. |
| 2 | **p\***  It matches any string containing zero or more p's. |
| 3 | **p?**  It matches any string containing zero or one p's. |
| 4 | **p{N}**  It matches any string containing a sequence of **N** p's |
| 5 | **p{2,3}**  It matches any string containing a sequence of two or three p's. |
| 6 | **p{2, }**  It matches any string containing a sequence of at least two p's. |
| 7 | **p$**  It matches any string with p at the end of it. |
| 8 | **^p**  It matches any string with p at the beginning of it. |

### **Examples**

Following examples will clear your concepts about matching characters.

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **[^a-zA-Z]**  It matches any string not containing any of the characters ranging from a through z and A through Z. |
| 2 | **p.p**  It matches any string containing p, followed by any character, in turn followed by another p. |
| 3 | **^.{2}$**  It matches any string containing exactly two characters. |
| 4 | **<b>(.\*)</b>**  It matches any string enclosed within <b> and </b>. |
| 5 | **p(hp)\***  It matches any string containing a p followed by zero or more instances of the sequence php. |

### **Predefined Character Ranges**

For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set −

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **[[:alpha:]]**  It matches any string containing alphabetic characters aA through zZ. |
| 2 | **[[:digit:]]**  It matches any string containing numerical digits 0 through 9. |
| 3 | **[[:alnum:]]**  It matches any string containing alphanumeric characters aA through zZ and 0 through 9. |
| 4 | **[[:space:]]**  It matches any string containing a space. |

## PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions −

|  |  |
| --- | --- |
| **Sr.No** | **Function & Description** |
| 1 | [ereg()](https://www.tutorialspoint.com/php/php_ereg.htm)  The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise. |
| 2 | [ereg\_replace()](https://www.tutorialspoint.com/php/php_ereg_replace.htm)  The ereg\_replace() function searches for string specified by pattern and replaces pattern with replacement if found. |
| 3 | [eregi()](https://www.tutorialspoint.com/php/php_eregi.htm)  The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive. |
| 4 | [eregi\_replace()](https://www.tutorialspoint.com/php/php_eregi_replace.htm)  The eregi\_replace() function operates exactly like ereg\_replace(), except that the search for pattern in string is not case sensitive. |
| 5 | [split()](https://www.tutorialspoint.com/php/php_split.htm)  The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string. |
| 6 | [spliti()](https://www.tutorialspoint.com/php/php_spliti.htm)  The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive. |
| 7 | [sql\_regcase()](https://www.tutorialspoint.com/php/php_sql_regcase.htm)  The sql\_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters. |

## PERL Style Regular Expressions

Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions. In fact, you can use any of the quantifiers introduced in the previous POSIX section.

Lets give explanation for few concepts being used in PERL regular expressions. After that we will introduce you wih regular expression related functions.

### **Meta characters**

A meta character is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for large money sums using the '\d' meta character: **/([\d]+)000/**, Here **\d** will search for any string of numerical character.

Following is the list of meta characters which can be used in PERL Style Regular Expressions.

**Character Description**

. a single character

\s a whitespace character (space, tab, newline)

\S non-whitespace character

\d a digit (0-9)

\D a non-digit

\w a word character (a-z, A-Z, 0-9, \_)

\W a non-word character

[aeiou] matches a single character in the given set

[^aeiou] matches a single character outside the given set

(foo|bar|baz) matches any of the alternatives specified

### **Modifiers**

Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

**Modifier Description**

i Makes the match case insensitive

m Specifies that if the string has newline or carriage

return characters, the ^ and $ operators will now

match against a newline boundary, instead of a

string boundary

o Evaluates the expression only once

s Allows use of . to match a newline character

x Allows you to use white space in the expression for clarity

g Globally finds all matches

cg Allows a search to continue even after a global match fails

## PHP's Regexp PERL Compatible Functions

PHP offers following functions for searching strings using Perl-compatible regular expressions −

|  |  |
| --- | --- |
| **Sr.No** | **Function & Description** |
| 1 | [preg\_match()](https://www.tutorialspoint.com/php/php_preg_match.htm)  The preg\_match() function searches string for pattern, returning true if pattern exists, and false otherwise. |
| 2 | [preg\_match\_all()](https://www.tutorialspoint.com/php/php_preg_match_all.htm)  The preg\_match\_all() function matches all occurrences of pattern in string. |
| 3 | [preg\_replace()](https://www.tutorialspoint.com/php/php_preg_replace.htm)  The preg\_replace() function operates just like ereg\_replace(), except that regular expressions can be used in the pattern and replacement input parameters. |
| 4 | [preg\_split()](https://www.tutorialspoint.com/php/php_preg_split.htm)  The preg\_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern. |
| 5 | [preg\_grep()](https://www.tutorialspoint.com/php/php_preg_grep.htm)  The preg\_grep() function searches all elements of input\_array, returning all elements matching the regexp pattern. |
| 6 | [preg\_ quote()](https://www.tutorialspoint.com/php/php_preg_quote.htm)  Quote regular expression characters |