# Basic PHP Syntax

A PHP script can be placed anywhere in the document.

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

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

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

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

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

## Example

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

<h1>My first PHP page</h1>

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

</body>
</html>
```

**Note:** PHP statements are terminated by semicolon (;). The closing tag of a block of PHP code also automatically implies a semicolon (so you do not have to have a semicolon terminating the last line of a PHP block).

---

# 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 editing the code!

Comments are useful for:

- To let others understand what you are doing - Comments let other programmers understand what you were doing in each step (if you work in a group)

- To remind yourself 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 three ways of commenting:

### Example

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

<?php
// This is a single line comment

# This is also a single line comment

/*
This is a multiple lines comment block
that spans over more than
one line
*/
?>

</body>
</html>
```

---

## PHP Case Sensitivity

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

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

### Example

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

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

</body>
</html>
```

However; in PHP, all variables are case-sensitive.

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

### Example

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

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

</body>
</html>
```

---

Variables are "containers" for storing information:

### Example

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

---

# Much Like Algebra

x=5
y=6
z=x+y

In algebra we use letters (like x) to hold values (like 5).

From the expression z=x+y above, we can calculate the value of z to be 11.

In PHP these letters are called **variables.**

 Think of variables as containers for storing data.

## PHP Variables

As with algebra, PHP variables can be used to hold values (x=5) or expressions (z=x+y).

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 ($y and $Y are two different variables)

 Remember that PHP variable names are case-sensitive!

## Creating (Declaring) PHP Variables

PHP has no command for declaring a variable.

A variable is created the moment you first assign a value to it:

### Example

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

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.

# 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

---

# Local and Global Scope

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

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

The following example tests variables with local and global scope:

## Example

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

function myTest() {
  $y=10; // local scope
  echo "<p>Test variables inside the function:</p>";
  echo "Variable x is: $x";
  echo "<br>";
```

```
  echo "Variable y is: $y";
}

myTest();

echo "<p>Test variables outside the function:</p>";
echo "Variable x is: $x";
echo "<br>";
echo "Variable y is: $y";
?>
```

In the example above there are two variables $x and $y and a function myTest(). $x is a global variable since it is declared outside the function and $y is a local variable since it is created inside the function.

When we output the values of the two variables inside the myTest() function, it prints the value of $y as it is the locally declared, but cannot print the value of $x since it is created outside the function.

Then, when we output the values of the two variables outside the myTest() function, it prints the value of $x, but cannot print the value of $y since it is a local variable and it is created inside the myTest() function.

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

---

# PHP The global Keyword

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

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

## Example

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

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

myTest();
```

```
echo $y; // outputs 15
?>
```

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

The example above can be rewritten like this:

### Example

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

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

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

---

# PHP The static Keyword

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

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

### Example

```
<?php

function myTest() {
  static $x=0;
  echo $x;
  $x++;
}

myTest();
myTest();
myTest();

?>
```

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

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

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

There are some differences between echo and print:

- echo - can output one or more strings
- print - can only output one string, and returns always 1

**Tip:** echo is marginally faster compared to print as echo does not return any value.

---

# The PHP echo Statement

echo is a language construct, and can be used with or without parentheses: echo or echo().

### Display Strings

The following example shows how to display different strings with the echo command (also notice that the strings can contain HTML markup):

## Example

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

### Display Variables

The following example shows how to display strings and variables with the echo command:

## Example

```
<?php
$txt1="Learn PHP";
```

```
$txt2="W3Schools.com";
$cars=array("Volvo","BMW","Toyota");

echo $txt1;
echo "<br>";
echo "Study PHP at $txt2";
echo "My car is a {$cars[0]}";
?>
```

---

# The PHP print Statement

print is also a language construct, and can be used with or without parentheses: print or print().

**Display Strings**

The following example shows how to display different strings with the print command (also notice that the strings can contain HTML markup):

## Example

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

**Display Variables**

The following example shows how to display strings and variables with the print command:

## Example

```
<?php
$txt1="Learn PHP";
$txt2="W3Schools.com";
$cars=array("Volvo","BMW","Toyota");

print $txt1;
print "<br>";
print "Study PHP at $txt2";
print "My car is a {$cars[0]}";
?>
```

# PHP Strings

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

---

# PHP Integers

An integer is a number without decimals.

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 we will test different numbers. The PHP var_dump() function returns the data type and value of variables:

## Example

```
<?php
$x = 5985;
var_dump($x);
echo "<br>";
$x = -345; // negative number
var_dump($x);
echo "<br>";
$x = 0x8C; // hexadecimal number
var_dump($x);
echo "<br>";
$x = 047; // octal number
var_dump($x);
?>
```

# PHP Floating Point Numbers

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

In the following example we will test different numbers. The PHP var_dump() function returns the data type and value of variables:

## Example

```
<?php
$x = 10.365;
var_dump($x);
echo "<br>";
$x = 2.4e3;
var_dump($x);
echo "<br>";
$x = 8E-5;
var_dump($x);
?>
```

# PHP Booleans

Booleans can be either 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 Arrays

An array stores multiple values in one single variable.

In the following example we create an array, and then use the PHP var_dump() function to return the data type and value of the array:

## Example

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

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

---

# PHP Objects

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.

We then define the data type in the object class, and then we use the data type in instances of that class:

## Example

```
<?php
class Car
{
  var $color;
  function Car($color="green") {
    $this->color = $color;
  }
  function what_color() {
    return $this->color;
  }
}
?>
```

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);
?>
```

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, in characters.

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

## Example

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

The output of the code above will be: 12

**Tip:** strlen() is often used in loops or other functions, when it is important to know when a string ends. (i.e. in a loop, we might want to stop the loop after the last character in a string).

---

# 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");
?>
```

The output of the code above will be: 6.

**Tip:** The position of the string "world" in the example above is 6. The reason that it is 6 (and not 7), is that the first character position in the string is 0, and not 1.

---

# Complete PHP String Reference

For a complete reference of all string functions, go to our complete PHP String Reference.

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

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.

---

# Set a PHP Constant

To set a constant, use the define() function - it takes three parameters: The first parameter defines the name of the constant, the second parameter defines the value of the constant, and the optional third parameter specifies whether the constant name should be case-insensitive. Default is false.

The example below creates a **case-sensitive constant**, with the value of "Welcome to W3Schools.com!":

## Example

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

The example below creates a **case-insensitive constant**, with the value of "Welcome to W3Schools.com!":

### Example

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

## PHP Arithmetic Operators

| Operator | Name | Example | Result |
|---|---|---|---|
| + | 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 |

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

### Example

```
<?php
$x=10;
$y=6;
echo ($x + $y); // outputs 16
echo ($x - $y); // outputs 4
echo ($x * $y); // outputs 60
echo ($x / $y); // outputs 1.6666666666667
echo ($x % $y); // outputs 4
?>
```

---

## PHP Assignment Operators

The PHP assignment operators is used 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 |
|---|---|---|
| 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 |

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

## Example

```
<?php
$x=10;
echo $x; // outputs 10

$y=20;
$y += 100;
echo $y; // outputs 120

$z=50;
$z -= 25;
echo $z; // outputs 25

$i=5;
$i *= 6;
echo $i; // outputs 30

$j=10;
$j /= 5;
echo $j; // outputs 2

$k=15;
$k %= 4;
```

```
echo $k; // outputs 3
?>
```

---

# PHP String Operators

| Operator | Name | Example | Result |
|---|---|---|---|
| . | Concatenation | $txt1 = "Hello"<br>$txt2 = $txt1 ." world!" | Now $txt2 contains "Hello world!" |
| .= | Concatenation assignment | $txt1 = "Hello"<br>$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!
?>
```

---

# 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
?>
```

# 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);
?>
```

---

# 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** - selects one of several blocks of code to be executed
- **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!";
}
?>
```

# 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!";
}
?>
```

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

Use the if....elseif...else statement to **select one of several blocks of code to be executed**.

## 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!";
}
?>
```