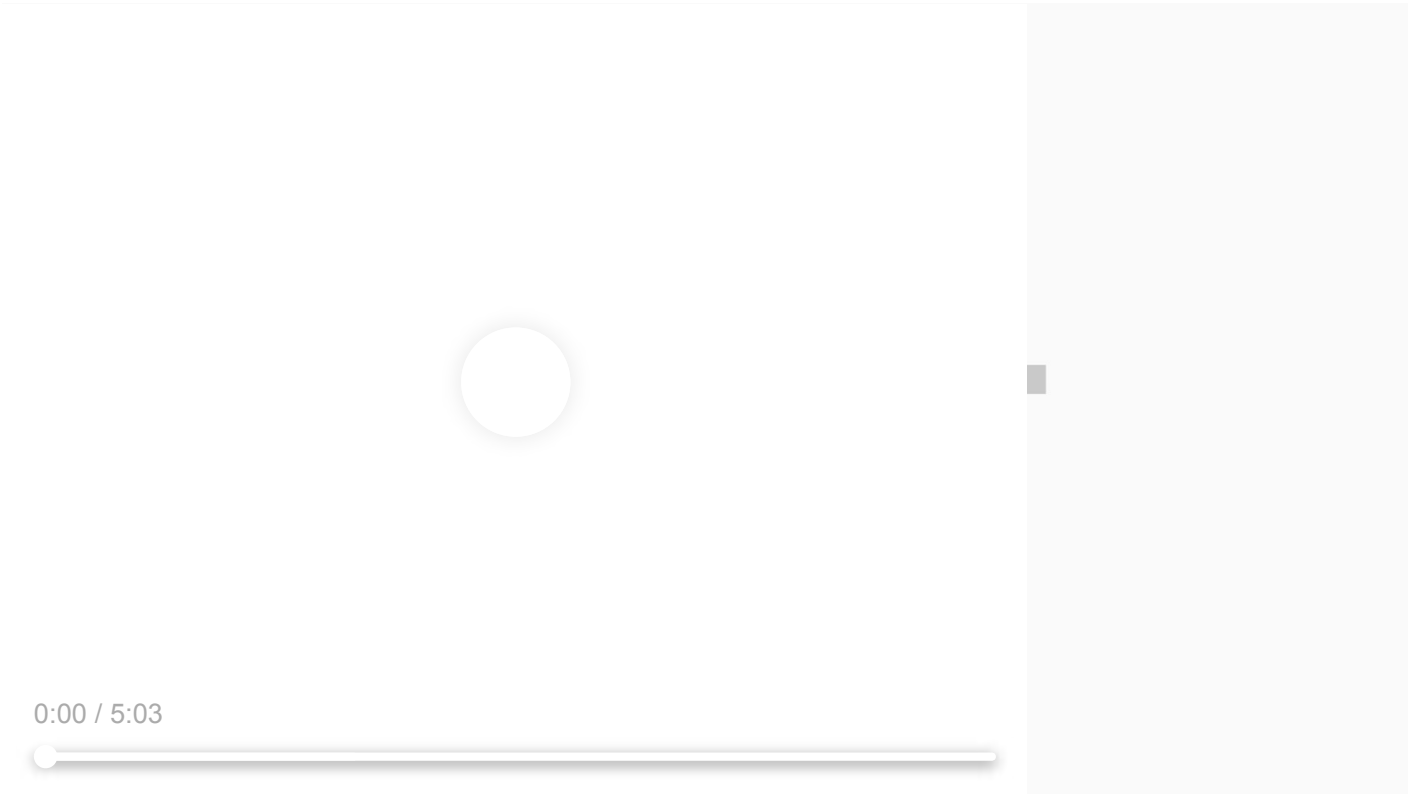
PHP | Day 1 | Pre-Work

Table of contents:



Introduction

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire ecommerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is fast in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common Uses of PHP

PHP performs system related functions on a web server: it can create, open, read, write, and close them. The other uses of PHP are:

- PHP can handle web forms, i.e. gather data from files, save data to a file,send data per email, return data to the user.
- You add, delete, modify elements within your database with PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible:

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

PHP Environment

In our MySQL chapter of this course you installed XAMPP - PHP Environment on your machine, but let’s see again what we need for PHP development.
In order to develop and run PHP Web pages, three vital components need to be installed on your computer system:

1. **Web Server:** PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but the most often used is freely available

Apache Server: https://httpd.apache.org/

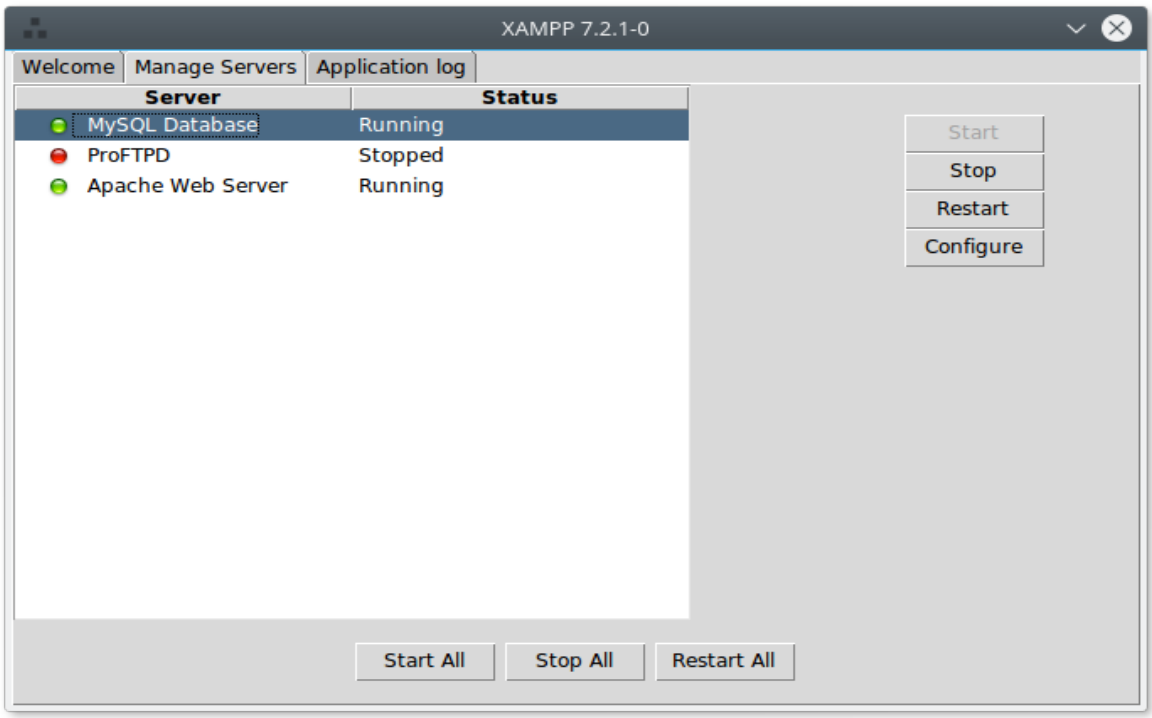
2. **Database:** PHP will work with virtually all database RDBMS, including Oracle and Sybase, but most commonly used is freely available MySQL database: http://www.mysql.com/downloads/index.html

PHP - In order to process PHP script instructions, a PHP must be installed to generate HTML output that can be sent to the Web Browser: http://php.net/manual/en/langref.php

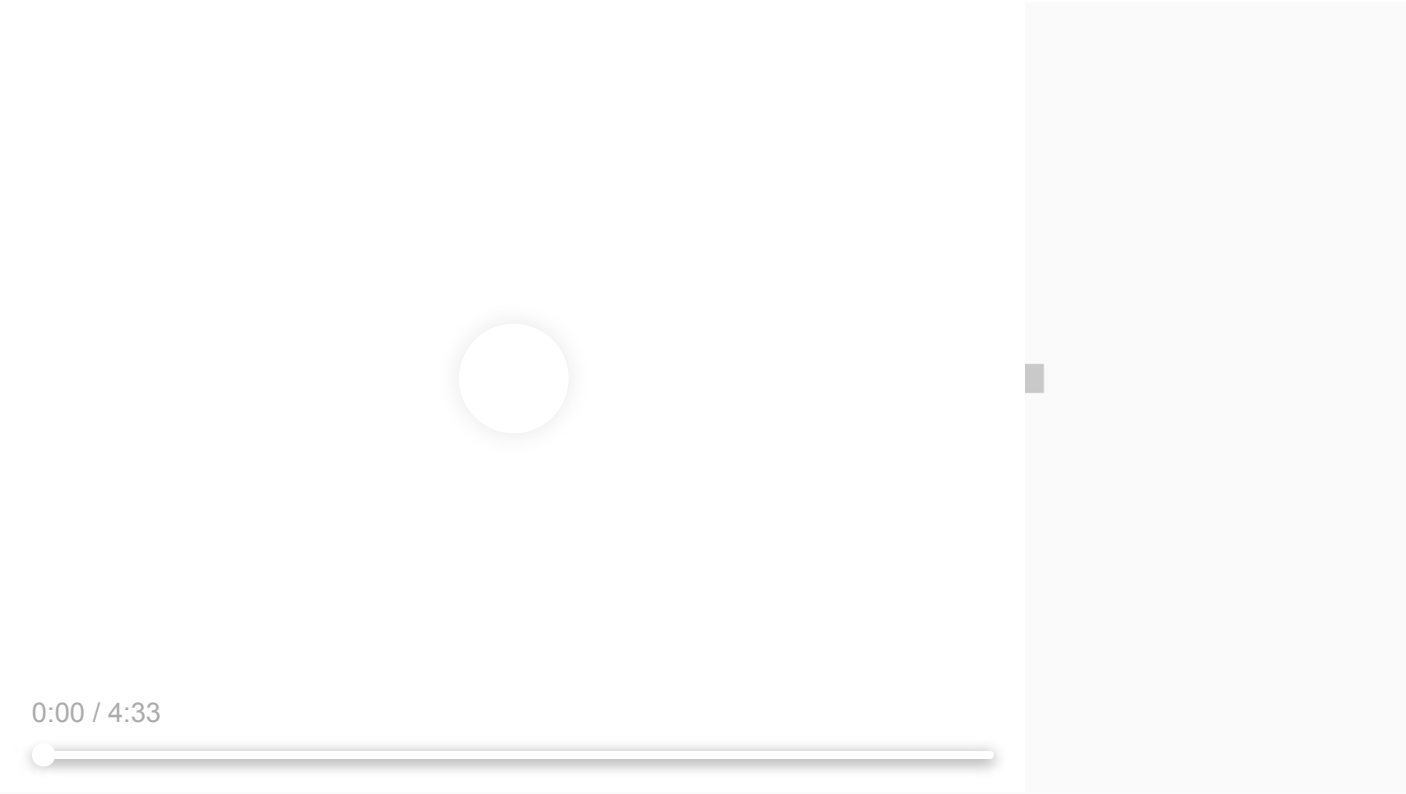
As we've already installed XAMPP, we will use XAMPP until the end of this course. XAMPP contains all what we need: Apache, MySQL and PHP Parser. It is easy to install - without special configuration required and that is exactly what we need for this course.

You should run your Apache and MySQL modules as in the image below:

To start Apache web server, select tab *Manage Servers*, then select *Apache web Server*, and then select *Start*. Repeat (if you want to use database) the same procedure for the service *My SQL Database* :

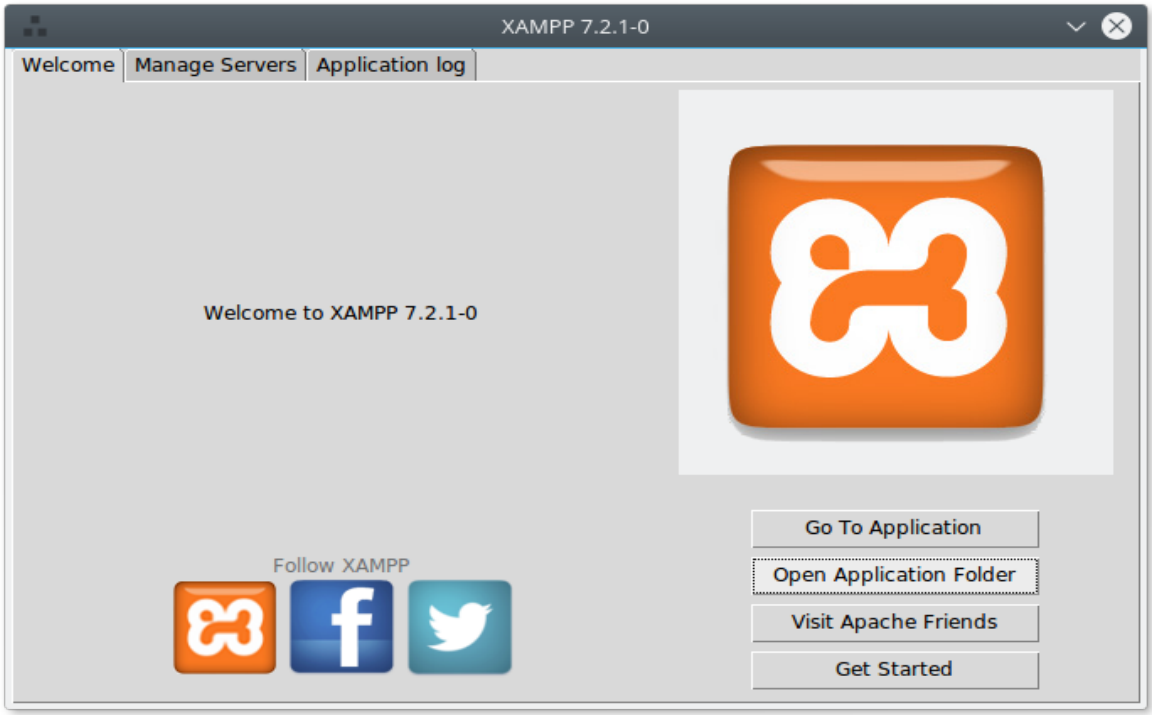


"Hello World" Script in PHP



To get a feel for PHP, first start with simple PHP scripts. We will create a friendly little "Hello, World!" script.

Very important is the **file location** - where to save your PHP files. You need to insert your files in the **htdocs** folder (this is a document root folder for Apache server). To access the htdocs on your system, press *Open Application Folder* in XAMPP:



Default path for htdocs folder is:

- **C:\xampp\htdocs** on Windows systems
- **/opt/lampp/htdocs** on Linux & MacOS systems (Note: this is the default web server root path used by XAMPP installation on Linux. If you however have used alternative method for installing Apache, PHP and mySQL/MariaDB based on your Linux distro, the common installation path for Ubuntu based distros is **/var/www/html**)

We will refer to this path in further text as **htdocs** folder.
This path will be your default path for all your web projects and your projects should be saved in the htdocs folder. Good practice is for every new project to create a new folder inside the htdocs folder - then inside this new folder save all your project's files and folders (html, css, js and php files).

Extension of your files must be **.php** instead of **.html** if you want to use PHP. Otherwise, your PHP script won't be executed on the server and you wouldn't get the expected result.

PHP can be embedded in HTML. That means that in amongst your normal HTML you'll have PHP statements like this:

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World</title>
<body>
 <?php echo "Hello, World!"; ?>
</body>
</html>
```

Save this file in the folder **php-exercises** inside your **htdocs** folder as **hello-world.php**

Now, visit http://localhost/php-exercises/hello-world.php. It should produce the following result:

```
Hello, World!
```

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed on the server side and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.
All PHP code must be included inside one of the three special markup tags recognized by the PHP Parser.

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

Important note: remember that PHP is executed on the server as a script program. Although most commonly used as server side language of choice for the web development, you are free to write and execute your scripts from the console without the necessity of working through the web server.

How does PHP & Web-Server really work: Request and Response mechanics

You request a file, the web server happens to be running PHP, and it sends HTML back to the browser (the **response**). But when you request a .php file, the web server routes this request to the PHP: it sends the HTML file as is, but PHP statements are processed by the PHP software before they're sent back to the web server. From there, the processed HTML is sent to the requester, and your <$php ?> code is not visible anymore.
When PHP language statements are processed, only the output, or anything printed to the screen is sent by the web server to the web browser. The PHP language statements, those that don't produce any output to the screen, aren't included in the output sent to the browser, so the PHP code is not normally seen by the user.

For instance, in this simple PHP statement:

```
<?php echo "<p>Hello World</p>"; ?>
```

Here, echo is a PHP instruction that tells PHP to output the upcoming text. The PHP software processes the PHP statement and outputs the following:

```
<p>Hello World</p>
```

That regular HTML statement is delivered to the user's browser. The browser interprets the statement as HTML code and displays a web page with one paragraph — Hello World. The PHP statement isn't delivered to the browser, so the web browser (and site visitor) never sees any PHP statements.

Expressions terminated by semicolons

A statement in PHP is any expression that is followed by a semicolon (;). Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called $greeting:

```
$greeting = "Welcome to PHP!";
```

Note that **variable** names in PHP begin with **dollar $ sign**.

Commenting PHP code

A comment is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

Single-line comments: They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments:

```
<?php
# This is a comment, and
# This is the second line of the comment
// This is a comment too. Each style comments only
// a single line
print "An example with single line comments";
?>
```

Multi-lines comments: They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. Here are the example of multi lines comments:

```
<?php
/* This is a comment with multi-line
   Purpose: Multi-line Comments Demo
   Subject: PHP
*/
print "An example with multi line comments";
?>
```

PHP is whitespace insensitive

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row. One whitespace character is the same as many such characters.

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable **$four** is equivalent:

```
<?php
$four = 2 + 2; // single spaces
echo $four;

echo "<br>";
$four =
2+
2;          // multiple lines
echo $four;
?>
```

In example above we use echo to print result from PHP script.

PHP is case sensitive

PHP is a **case sensitive** language. Try out the following example as a demonstration:

```
<!DOCTYPE html>
<html>
<body>
<?php
$capital = 67;
print("Variable capital is $capital<br>");
print("Variable CaPiTaL is $CaPiTaL<br>");
?>
</body>
</html>
```

Variable Types

The main way to store information in the middle of a PHP program is by using a variable. Here are the most important things to know about variables in PHP.

1. All variables in PHP are expressed with a leading dollar sign **($)**.
2. The value of a variable is the value of its most recent assignment.
3. Variables are assigned with the **=** operator, with the variable on the left-hand side and the expression to be evaluated on the right.
4. Variables can, but do not need, to be declared before assignment.
5. Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
6. Variables used before they are assigned have default values.
7. PHP does a good job of automatically converting types from one to another when necessary.

PHP has a total of eight data types which we use to construct our variables:

- **Integers:** are whole numbers, without a decimal point, like 4195.
- **Doubles:** are floating-point numbers, like 3.14159 or 49.1.
- **Booleans:** have only two possible values either true or false.
- **NULL:** is a special type that only has one value: NULL.
- **Strings:** are sequences of characters, like 'PHP supports string operations.'

- **Arrays:** are named and indexed collections of other values.
- **Objects:** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources:** are special variables that hold references to resources external to PHP (such as database connections).

The escape-sequence replacements are:

- **\n** is replaced by the newline character
- **\r** is replaced by the carriage-return character
- **\t** is replaced by the tab character
- **\$** is replaced by the dollar sign itself ($)
- **\"** is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

Variable naming

Rules for naming a variable is:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc

There is no size limit for variables.

PHP –Variables

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

1. Local variables
2. Function parameters
3. Global variables
4. Static variables

Local variables

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function.

Here is an example:

```
<?php
$a = 2;  // define variable with global scope
// define function
function Sum()
{
 $a = 1;       // Define variable with same name,
               // but with local function scope.
 echo "a inside function =";
 echo "$a \n"; // $a will be interpreted and
               // rendered as "1",
               // when you use double quotes in string.
}
Sum();         // call the function
echo "a outside the function =";
echo $a;       // renders "2"
?>
```

Function Parameters

PHP functions are covered later in this course in details, but in short, a function is a small unit of program which can take some input in the form of parameters and does some processing and may return a value (similar as JavaScript is doing). Function parameters are declared after the function name and inside parentheses.

```
<?php
$first_number = 2;  // define variables with global scope
$second_number = 3;
// define function
function Sum($a, $b)
{
 $c = $a + $b;
 echo "sum is = $c \n";
}
Sum($first_number, $second_number);   // Call the function.
                                      // output is:
                                      // "sum is = 5"
?>
```

Global variables

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to modify or access global variable inside of a function, a global variable must be explicitly declared to be **global** inside the function in which it is to be accessed/modified. This is accomplished, conveniently enough, by placing the keyword **global** in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name.

```php
<?php
$a = 2;          // variable with global scope
$b = 3;          // variable explicitly declared
                // "global" in function
// define function
function test_scope()
{
// global $a;   // commented on purpose
echo "a inside function = ";
echo "$a \n";               // produces "Undefined
variable:" notice
global $b;
echo "b inside function = ";
echo "$b \n";   // produces "3"
$b = $b + 1;    // changes global variable
}
test_scope();                          // call the function
echo "a OUTSIDE function = $a \n";  // produces "2"

echo "b OUTSIDE function = $b \n";  // produces "4"
?>
```

Static variables

The final type of variable scoping that we discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.
You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

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

myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
```

Operator Types

PHP language supports following type of operators:

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

The following arithmetic operators are supported by PHP language:
Assume **variable A =10** and **variable B = 20** then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide the numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

(Note: $ prefix is omitted in variable names for shortness)

Comparison Operators

There are following comparison operators supported by PHP language.

Assume **variable A = 10** and **variable B = 20** then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes, then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal, then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes, then condition becomes true.	(A <= B) is true.

(Note: $ prefix is omitted in variable names for shortness)

Logical Operators

The following logical operators are supported by PHP language.
Assume **variable A =10** and **variable B = 20** then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true, then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero, then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero, then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is false.

(Note: $ prefix is omitted in variable names for shortness)

Note also that all integers that are not 0 are evaluated as “true” by PHP logic operators. PHP has its own rules when it comes to logical evaluation of non-boolean variables, more information here: http://php.net/manual/en/types.comparisons.php

Assignment Operators

PHP supports the following assignment operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

(Note: $ prefix is omitted in variable names for shortness)

Conditional Operator

There is one more operator called the conditional operator. It first evaluates an expression for a true or false value and then executes one of the two given statements depending on the result of the evaluation.

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Here is an example:

```php
<?php
$x = 10;
$y = 20;
$result = ($x < $y) ? "smaller" : "not smaller";
echo "x is $result than y \n"; // produces:
                                // "x is smaller than y"
?>
```

Single quoted strings vs double quoted strings

> **Single quoted strings** will display almost completely "as is". Variables and most escape sequences will not be interpreted. The exception is that to display a literal single quote, you can escape it with a backslash **\'** and to display a backslash, you can escape it with another backslash **** .
>
> **Double quote strings** will display with escaped characters and variables in the strings evaluated (see example above). An important point here is that you can use curly braces to isolate the name of the variable you want evaluated. For example let's say you have the variable **$type** and you what to echo *"The $types are"* . That will look for the variable **$types (**with “s”). To get around this use **echo "The {$type}s are"** .

Note: you will produce a slightly faster script if you use use single quotes, since PHP would not use additional processing to interpret what is inside the single quote. When you use double quotes PHP has to parse to check if there is any variables in there.

Statements

The if, elseif ...else and switch statements are used to take decision based on the different condition. You can use conditional statements in your code to make your decisions. PHP supports the following three decision making statements:

a. **if...else** statement - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
b. **elseif** statement - is used with the if...else statement to execute a set of code if one of several condition are true
c. **switch** statement - is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

```
<!DOCTYPE html>
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
echo "Have a nice weekend!";
else
echo "Have a nice day!";
?>
</body>
</html>
```

The ElseIf Statement

If you want to execute some code if one of the several conditions is true, then use the elseif statement.
The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<!DOCTYPE html>
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
echo "Have a nice weekend!";
elseif ($d=="Sun")
echo "Have a nice Sunday!";
else
echo "Have a nice day!";

?>
</body>
</html>
```

The Switch Statement

If you want to select one of many blocks of code to be executed based on a result of single expression, you can the **switch** statement. The switch statement is used to avoid long blocks of if..elseif..else code.
The switch statement works in an unusual way. First it evaluates the given expression, then seeks a label to match the resulting value. If a matching value is found, then the code associated with the matching label will be executed. If none of the labels match, then the statement will execute any specified default code.

```
<!DOCTYPE html>
<html>
<body>
<?php
$d=date("D");
switch ($d)
{
case "Mon":
echo "Today is Monday";
break;
case "Tue":
echo "Today is Tuesday";
break;
case "Wed":
echo "Today is Wednesday";
break;
case "Thu":
echo "Today is Thursday";
break;
case "Fri":
echo "Today is Friday";
break;
case "Sat":
echo "Today is Saturday";
break;
case "Sun":
echo "Today is Sunday";
break;
default:
echo "Wonder which day is this ?";
}
?>
</body>
</html>
```

Loops

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for** - loops through a block of code a specified number of times.
- **while** - loops through a block of code if and as long as a specified condition is true.

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

The for loop

The for statement is used when you know how many times you want to execute a statement or a block of statements.
The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is tradition to name it **$i**.

```
<!DOCTYPE html>
<html>
<body>
<?php
$a = 0;
$b = 0;
for( $i=0; $i<5; $i++ )
{
$a += 10;
$b += 5;
}
// Output:
// At the end of the loop a=50 and b=25
echo ("At the end of the loop a=$a and b=$b");
?>
</body>
</html>
```

The while loop

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true, then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation becomes false and the loop ends.

```
<!DOCTYPE html>
<html>
<body>
<?php
$i = 0;
$num = 50;
while( $i < 10)
{
$num--;
$i++;
}
echo ("Loop stopped at i = $i and num = $num" );
// the output will be like "Loop stopped at i = 10 and num
= 40"
?>
</body>
</html>
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it will then repeat the loop as long as a condition is true.

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10:

```
<!DOCTYPE html>
<html>
<body>
<?php
$i = 0;
$num = 0;
do
{
$i++;
} while( $i < 10 );
echo ("Loop stopped at i = $i" );
?>
</body>
</html>
```

The foreach loop statement

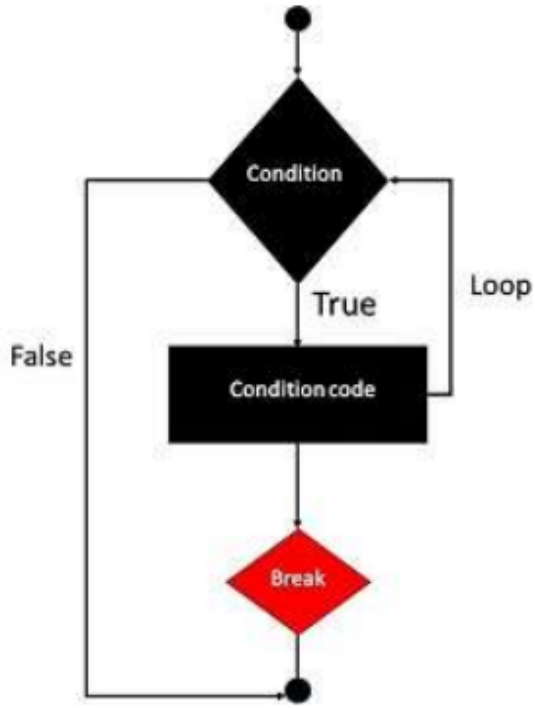
The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

For this example we will loop through an array:

```
<!DOCTYPE html>
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
echo "Value is $value <br />";
}
?>
</body>
</html>
```

The break statement

The PHP break keyword is used to terminate the execution of a loop prematurely. The break statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop the immediate statement after the loop will be executed. In simpler words, if the condition is met that iteration will be skipped (or jumped) and it will continue with the next one.



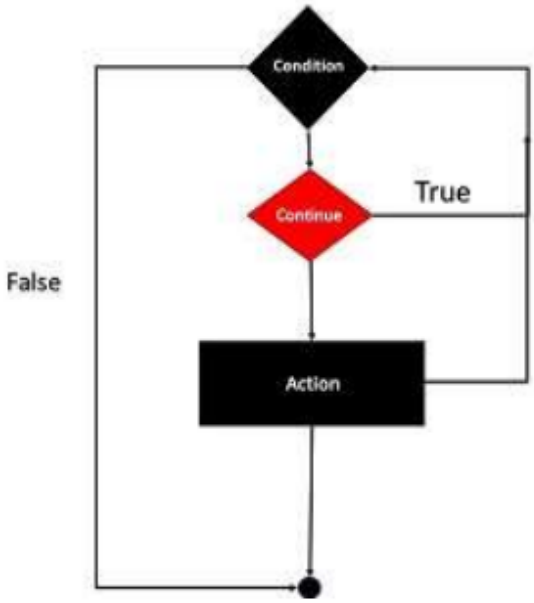
In the following example, the condition test becomes true when the counter value reaches 3 and loop terminates.

```
<!DOCTYPE html>
<html>
<body>
<?php
$i = 0;
while( $i < 10)
{
$i++;
if( $i == 3 )break;
}
echo ("Loop stopped at i = $i \n" );
?>
</body>
</html>
```

The continue statement

The PHP continue keyword is used to halt the current iteration of a loop but it does not terminate the loop.
Just like the break statement the continue statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. Using

the continue statement, the rest of the loop code is skipped and next pass starts.



In the following example, the loop prints the value of array, but when the condition becomes true, it just skips the code and the next value is printed.

```
<!DOCTYPE html>
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
 if( $value == 3 )continue;
 echo "Value is $value <br />";
}
// the output will be:
// "Value is 1 <br />Value is 2 <br />Value is 4 <br
/>Value is 5 <br />"
?>
</body>
</html>
```

Array Types

As you already know, an array is a data structure that stores one or more similar type of values in a single value. For example, if you want to store 100 numbers, then instead of defining 100 variables, it is easy to define an array of 100 length.

An array in PHP is actually an ordered map. A map is a type that associates values to keys. This type is optimized for several different uses.

There are three different kind of arrays and each array value is accessed using an ID which is called array index.

1. **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion, with vector-like accessing syntax, like **$var_name[6]**

2. **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order, like **$fruit_price['Bananas']**

3. **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices like: **$car_price['Tesla'] ['Model-S']**

Numeric Array

These arrays can store **numbers, strings and any object** but their **index will be represented by numbers**. By default, the array index starts with zero.

The following example demonstrates how to create and access numeric arrays. Here we have used array() function to create array.

```
<!DOCTYPE html>
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value )
{
echo "Value is $value <br />";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";
foreach( $numbers as $value )
{
echo "Value is $value <br />";
}
?>
</body>
</html>
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their **index as strings** so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE: Don't keep associative arrays inside double quote while printing, otherwise it would not return any values.

```
<!DOCTYPE html>
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
"mark" => 2000,
"anthony" => 1000,
"eric" => 500
);
echo "Salary of mark is ". $salaries['mark'] . "<br />";
echo "Salary of anthony is ". $salaries['anthony']. "<br
/>";
echo "Salary of eric is ". $salaries['eric']. "<br />";
/* Second method to create array. */
$salaries['mark'] = "high";
$salaries['anthony'] = "medium";
$salaries['eric'] = "low";
echo "Salary of mark is ". $salaries['mark'] . "<br />";
echo "Salary of anthony is ". $salaries['anthony']. "<br
/>";
echo "Salary of eric is ". $salaries['eric']. "<br />";
?>
</body>
</html>
```

Multidimensional Array

In an multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple indices.
In this example, we create a two dimensional array to store marks of three students in three subjects:

```
<!DOCTYPE html>
<html>
<body>
<?php
$marks = array(
"mark" => array
(
"physics" => 35,
"maths" => 30,
"chemistry" => 39
),
"anthony" => array
(
"physics" => 30,
"maths" => 32,
"chemistry" => 29
),
"eric" => array
(
"physics" => 31,
"maths" => 22,
"chemistry" => 39
)
);
/* Accessing multidimensional array values */
echo "Marks for mark in physics : " ;
echo $marks['mark']['physics'] . "<br />";
echo "Marks for anthony in maths : ";
echo $marks['anthony']['maths'] . "<br />";
echo "Marks for eric in chemistry : " ;
echo $marks['eric']['chemistry'] . "<br />";
?>
</body>
</html>
```

Last modified: Thursday, 14 June 2018, 10:38 AM