Intoduction to server-side scripting language

Server-side scripting is technique used in web development which involves employing scripts on a web server which produce a customized response for each user's (client's) request to the website.

Many languages may be used to create these scripts. Some of them are:

- ASP
- ASP.NET
- Java
- JavaScript (using SSJS (Server-side JavaScript) e.g., node.js)
- Perl
- PHP
- Python
- R
- Ruby

Advantages of server-side scripting

- Server-side scripting prevents increasing of the load as it does not require plugins or browser scripting technology (such as Javascript). Overloading leads to problems like slow loading, high CPU usage, and even freezing.
- It is used to create pages dynamically on the fly. Based on the user interaction, new pages can instantly be created.
- Server-side scripting is necessary to run dynamic pages on browsers that do not fully support Javascript.
- Server-side scripting does not depend on browser processing as all the processing is performed on the server side.
- Website owners can create their own applications and make use of CMS (content management system) to easily create and update content on the web without coding as the Server-side scripting languages like PHP can be configured to run CMS applications, like WordPress and Joomla.
- As the scripting is done on the server, it is not sent back to the browser, which prevents it from being cloned, copied, or scrutinized for hacking vulnerabilities.
- Loading time of the web pages is often reduced with Server-side scripting which helps to improve your site's Google ranking.
- An increased security is ensured for user privacy and is the preferred choice for e-commerce, membership and social media sites.

Disadvantages of server-side scripting

- Server-side scripting is slow at times like: 1.) If the user disconnects from the internet, or 2.) The web hosting is down, Then the script may take a long time to execute.
- The scripting software has to be installed on the server.
- New security concerns are associated with the dynamic scripts as in some cases hackers exploit the code flaws to gain access to servers.
- A database is required to store the dynamic data that needs a regular backing up and has to be kept secure.
- Here the pages have to be refreshed often in order to show the dynamic content. Ajax, an innovative method of exchanging data with a server is used by the developers which resolve the problem.
- Websites using large applications and with heavy traffic have to make use of more powerful hosting methods like dedicated servers or cloud hosting to cope with demand.

Difference between client-side scripting and server-side scripting

Client-side scripting	Server-side scripting
Source code is visible to the user.	Source code is not visible to the user because its output of server-side is an HTML page.
It usually depends on the browser and its version.	In this any server-side technology can be used and it does not depend on the client.
It runs on the user's computer.	It runs on the webserver.
There are many advantages linked with this like faster. response times, a more interactive application.	The primary advantage is its ability to highly customize, response requirements, access rights based on user.

Client-side scripting	Server-side scripting
It does not provide security for data.	It provides more security for data.
It is a technique used in web development in which scripts run on the client's browser.	It is a technique that uses scripts on the webserver to produce a response that is customized for each client's request.
HTML, CSS, and javascript are used.	PHP, Python, Java, Ruby are used.

PHP Introduction

- PHP stands for PHP: Hypertext Preprocessor.
- It is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.
- It is open source, free to download and use.
- It supports many databases (MySQL, Oracle, and so on)
- PHP scripts are executed on the server.

What is a PHP File?

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

What Can PHP Do?

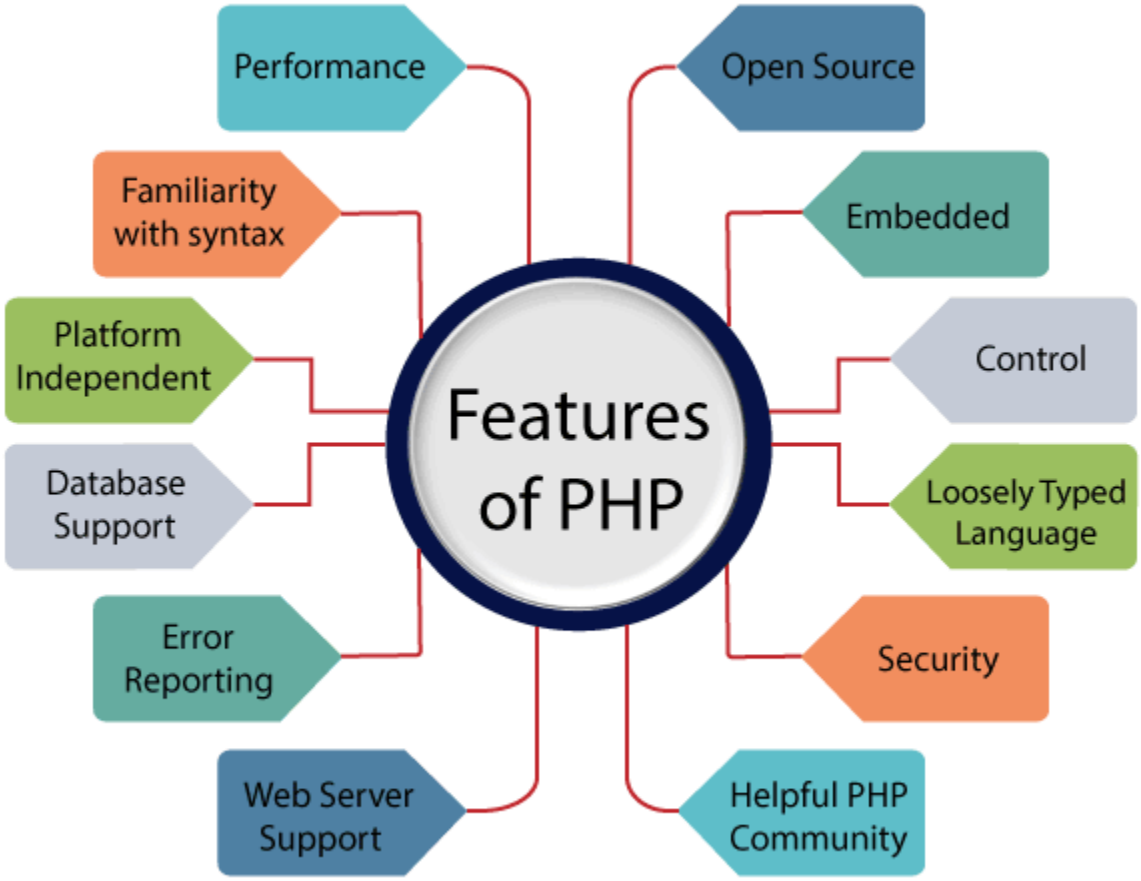
- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

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

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free.
- PHP is easy to learn and runs efficiently on the server side

PHP Features



Performance:

PHP script is executed much faster than those scripts which are written in other languages such as JSP and ASP. PHP uses its own memory, so the server workload and loading time is automatically reduced, which results in faster processing speed and better performance.

Open Source:

PHP source code and software are freely available on the web. You can develop all the versions of PHP according to your requirement without paying any cost. All its components are free to download and use.

Familiarity with syntax:

PHP has easily understandable syntax. Programmers are comfortable coding with it.

Embedded:

PHP code can be easily embedded within HTML tags and script.

Platform Independent:

PHP is available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.

Database Support:

PHP supports all the leading databases such as MySQL, SQLite, ODBC, Oracle etc.

Error Reporting –

PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.

Loosely Typed Language:

PHP allows us to use a variable without declaring its datatype. It will be taken automatically at the time of execution based on the type of data it contains on its value.

Web servers Support:

PHP is compatible with almost all local servers used today like Apache, Netscape, Microsoft IIS, etc.

Security:

PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threads and malicious attacks.

Control:

Different programming languages require long script or code, whereas PHP can do the same work in a few lines of code. It has maximum control over the websites like you can make changes easily whenever you want.

A Helpful PHP Community:

It has a large community of developers who regularly updates documentation, tutorials, online help, and FAQs. Learning PHP from the communities is one of the significant benefits.

PHP's Place in the Web World

PHP is a programming language that's used mostly for building web sites. Instead of a PHP program running on a desktop computer for the use of one person, it typically runs on a web server and is accessed by lots of people using web browsers on their own computers. This section explains how PHP fits into the interaction between a web browser and a web server.

When you sit down at your computer and pull up a web page using a browser such as Internet Explorer or Mozilla, you cause a little conversation to happen over the Internet between your computer and another computer. This conversation and how it makes a web page appear on your screen is illustrated in Figure 1-1.

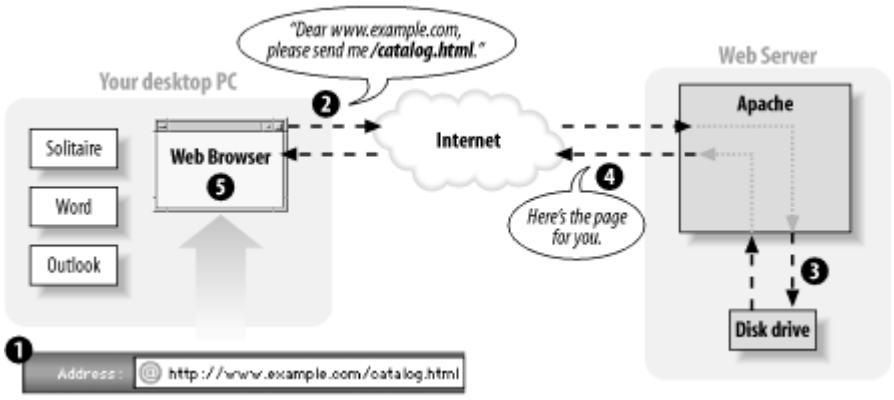


Figure 1-1. Client and server communication without PHP

Here's what's happening in the numbered steps of the diagram:

1. You type *www.example.com/catalog.html* into the location bar of browser.

2. Browser sends a message over the Internet to the computer named www.example.com asking for the */catalog.html* page.

3. Apache, a program running on the www.example.com computer, gets the message and reads the *catalog.html* file from the disk drive.

4. Apache sends the contents of the file back to your computer over the Internet as a response to Browser's request.

5. Browser displays the page on the screen, following the instructions of the HTML tags in the page.

Every time a browser asks for *http://www.example.com/catalog.html*, the web server sends back the contents of the same *catalog.html* file. The only time the response from the web server changes is if someone edits the file on the server.

When PHP is involved, however, the server does more work for its half of the conversation. Figure 1-2 shows what happens when a web browser asks for a page that is generated by PHP.

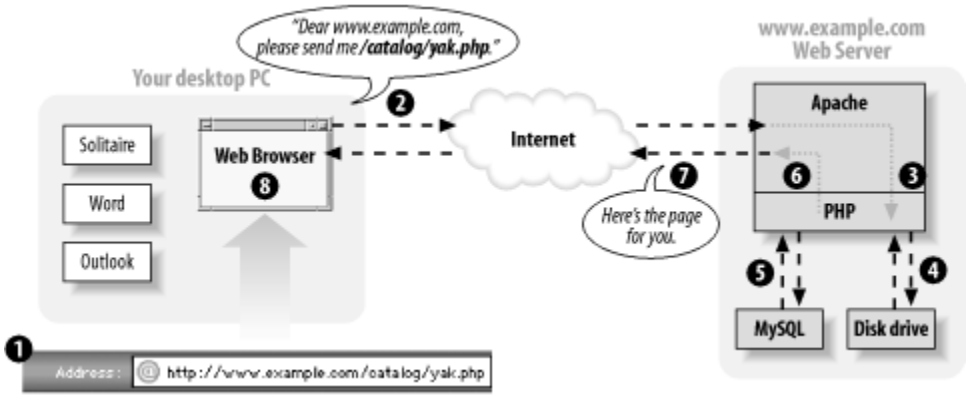


Figure 1-2. Client and server communication with PHP

Here's what's happening in the numbered steps of the PHP-enabled conversation:

1. You type *www.example.com/catalog/yak.php* into the location bar of browser.

2. Browser sends a message over the Internet to the computer named www.example.com asking for the */catalog/yak.php* page.

3. Apache, a program running on the www.example.com computer, gets the message and asks the PHP interpreter, another program running on the www.example.com computer, "What does */catalog/yak.php* look like?"

4. The PHP interpreter reads the file */usr/local/www/catalog/yak.php* from the disk drive.

5. The PHP interpreter runs the commands in *yak.php*, possibly exchanging data with a database program such as MySQL.

6. The PHP interpreter takes the *yak.php* program output and sends it back to Apache as an answer to "What does */catalog/yak.php* look like?"

7. Apache sends the page contents it got from the PHP interpreter back to your computer over the Internet in response to Browser's request.

8. Browser displays the page on the screen, following the instructions of the HTML tags in the page.

Advantages of PHP?

PHP Is Free
You do not have to pay anyone to use PHP. There are no licensing fees, support fees, maintenance fees, upgrade fees, or any other kind of charge.

PHP Is Cross-Platform
You can use PHP with a web server computer that runs Windows, Mac OS X, Linux, Solaris, and many other versions of Unix. Plus, if you switch web server operating systems, you generally don't have to change any of your PHP programs. Just copy them from your Windows server to your Unix server, and they will still work.

PHP Is Widely Used
As of March 2004, PHP is installed on more than 15 million different web sites, from countless tiny personal home pages to giants like Yahoo!. There are many books, magazines, and web sites devoted to teaching PHP and exploring what you can do with it. There are companies that provide support and training for PHP. In short, if you are a PHP user, you are not alone.

PHP Hides Its Complexity
You can build powerful e-commerce engines in PHP that handle millions of customers. You can also build a small site that automatically maintains links to a changing list of articles or press releases. When you're using PHP for a simpler project, it doesn't get in your way with concerns that are only relevant in a massive system. When you need advanced features such as caching, custom libraries, or dynamic image generation, they are available. If you don't need them, you don't have to worry about them. You can just focus on the basics of handling user input and displaying output.

PHP Is Built for Web Programming
Unlike most other programming languages, PHP was created from the ground up for generating web pages. This means that common web programming tasks, such as accessing form submissions and talking to a database, are often easier in PHP. PHP comes with the capability to format HTML, manipulate dates and times, and manage web cookies — tasks that are often available only as add-on libraries in other programming languages.

Basic rules of PHP program

Start and End Tags

The PHP interpreter runs the commands between <?php (the PHP start tag) and ?> (the PHP end tag). The PHP interpreter ignores anything outside of those tags.

Text before the start tag and after end tag is printed with no interference from the PHP interpreter.

A PHP program can have multiple start and end tag pairs.

Whitespace and Case-Sensitivity

PHP is insensitive of whitespace. This includes all types of spaces that are invisible on the screen including tabs, spaces, and carriage return. Even one space is equal to any numbers of spaces or carriage return. This means that PHP will ignore all the spaces or tabs in a single row or carriage return in multiple rows. Unless a semi-colon is encountered, PHP treats multiple lines as a single command.

PHP is case-sensitive. All the keywords, functions and class names in PHP (while, if, echo etc) are NOT case sensitive except variables. Only variables with different cases are treated differently.

Comments

A comment is something which is ignored and not read or executed by PHP engine or the language as part of a program and is written to make the code more readable and understandable.

PHP supports two types of comment:

1. Single Line comment: As the name suggests these are single line or shore relevant explanations that one can add in their code. To add this, we need to begin the line with (//) or (#).

   Example:

```php
?php
// This is a single line comment
 // These cannot be extended to more lines

echo "hello world!!!";

# This is also a single line comment

?>
```

2. Multiple Line comment: These are used to accommodate multiple lines with a single tag and can be extended to many lines as required by the user. To add this, we need to begin and end the line with (/*..*/)

   Example:

```php
<?php
/* This is a multi line comment
   In PHP variables are written by adding a $ sign at the beginning.*/
  $geek = "hello world!";
echo $geek;

?>
```

Data Types

Data Types defines the type of data a variable can store.

PHP supports the following data types:

- String: A string is a sequence of characters, like "Hello world!".
- Integer: An integer data type is a non-decimal number between -2,147,483,647 to 2,147,483,647
- Float (floating point numbers - also called double): A float is a number with a decimal point or a number in exponential form.
- Boolean: have only two possible values either true or false.
- Array: An array stores multiple values in one single variable.
- Object: instance of a class.
- NULL: variable that has no value.

Variables in PHP

Variables are used for storing values, like text strings, numbers or arrays.

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

Syntax:

$var_name = value;

Example

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

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)

PHP is a Loosely Typed Language

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

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

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators

- Assignment operators
- Comparison operators
- Logical operators
- String operators
- Conditional assignment operators

Arithmetic Operators

Arithmetic Operators are used to perform arithmetic between variables and/or values. Given that $y=5, the table below explains the arithmetic operators:

Operator	Description	Example	Result in $y	Result in $x
+	Addition	$x=$y+2	5	7
-	Subtraction	$x=$y-2	5	3
*	Multiplication	$x=$y*2	5	10
/	Division	$x=$y/2	5	2.5
%	Modulus	$x=$y%2	5	1
++	Increment	$x=++$y	6	6
		$x=$y++	6	5
--	Decrement	$x=--$y	4	4
		$x=$y--	4	5

Assignment Operators

Assignment Operators are used to assign values to variables. Given that $x=10 and $y=5, the table below explains the assignment operators:

Operator	Example	Same As	Result in x
=	$x=$y	$x=$y	5
+=	$x+=$y	$x=$x+$y	15
-=	$x-=$y	$x=$x-$y	5
*=	$x*=$y	$x=$x*$y	50

/=	$x/=$y	$x=$x/$y	2
%=	$x%=$y	$x=$x%$y	0

Comparison Operators

Comparison Operators are used in logical statements to determine equality or difference between variables and values. Given that $x=5, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
==	Equal to	$x==6	false
===	Equal value and type	$x===“5”	false
!=	Not equal	$x!=6	true
!==	Not equal value and type	$x!==“5”	true
>	Greater than	$x>2	true
<	Less than	$x<2	false
>=	Greater than or equal to	$x>=5	true
<=	Less than or equal to	$x<=5	true

Logical operator

Logical Operators are used to determine the logic between the variables and values. Given that $x=6 and $y=3, the table below explains the logical operators:

Operator	Description	Example	Returns
&&	and	x<10 && y > 1	true
||	or	x===“5” || y===“5”	false
!	not	!(x==y)	true

String Operator

PHP has two operators that are specially designed for strings.

Operator	Description	Example	Result
.	Concatenation	$a = “Shooman”.”Khatri”;	ShoomanKhatri
.=	Concatenation-assignment	$a = “Shooman” $a .= “Khatri”;	ShoomanKhatri

Conditional Assignment(Ternary) Operator

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

Conditional Statements

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

In PHP we have the following conditional statements:

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif...else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

If Statement

The if statement executes some code if one condition is true.

Syntax

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

Example

Output "Have a good day!" if the current time (HOUR) is less than 20:

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

if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {

  code to be executed if condition is true;

} else {

  code to be executed if condition is false;

}
```

Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
<?php

$t = date("H");

if ($t < "20") {

  echo "Have a good day!";

} else {

  echo "Have a good night!";

}

?>
```

if...elseif...else Statement

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

Syntax

```
if (condition) {

  code to be executed if this condition is true;

} elseif (condition) {

  code to be executed if first condition is false and this condition is true;

} else {

  code to be executed if all conditions are false;

}
```

Example

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!":

```
<?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!";

}

?>
```

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, nor green!";
}
?>
```

Looping Statements

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 statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```
for (initialization; condition; increment){
   code to be executed;
}
```

Example

```
<html>
   <body>

      <?php
         $a = 0;
         $b = 0;

         for( $i = 0; $i<5; $i++ ) {
            $a += 10;
            $b += 5;
         }

         echo "At the end of the loop a = $a and b = $b";
      ?>

   </body>
</html>
```

The while loop statement

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.

Syntax

```
while (condition) {
   code to be executed;
}
```

Example

```
<html>
   <body>

      <?php
         $i = 0;
         $num = 50;

         while( $i < 10) {
            $num--;
            $i++;
         }

         echo "Loop stopped at i = $i and num = $num";
      ?>

   </body>
</html>
```

The do...while loop statement

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

Syntax

```
do {
   code to be executed;
}
while (condition);
```

Example

```
<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.

Syntax

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

Example

```
<html>
   <body>

      <?php
         $array = array( 1, 2, 3, 4, 5);

         foreach( $array as $value ) {
            echo "Value is $value <br />";
         }
      ?>

   </body>
</html>
```

PHP Arrays

An array stores multiple values in one single variable:

Example

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

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

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

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

The solution is to create an array!

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

Create an Array in PHP

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

```
array();
```

In PHP, there are three types of arrays:

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

---

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

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

Example

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

1) PHP Indexed Arrays

There are two ways to create indexed arrays:

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

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

or the index can be assigned manually:

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

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

for($x = 0; $x < $arrlength; $x++) {
  echo $cars[$x];
  echo "<br>";
}
?>
```

2) 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:

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

or:

```php
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

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

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
```

---

3) PHP - Multidimensional Arrays

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

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

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

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

---

PHP - Two-dimensional Arrays

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

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

```
$cars = array (
  array("Volvo",22,18),
```

```
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

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

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

Example

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

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

Example

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

Some PHP Arrays Functions

PHP Sorting Arrays

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

Syntax: sort(array, sorttype)

Where sorttype is optional, specifies how to compare the array elements. Possible values of sorttype:

- SORT_REGULAR - Default. Compare items normally (don't change types)
- SORT_NUMERIC - Compare items numerically
- SORT_STRING - Compare items as strings
- SORT_LOCALE_STRING - Compare items as strings, based on current locale

PHP - Sort Functions For Arrays

- `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

Example:

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

PHP Merging arrays

PHP provides easiest way to merge list of arrays. The array_merge() function is used to merge one or more arrays into one array.

Syntax: `array_merge(array1, array2, array3, ...)`

Example:

Merge array

```
<?php
$a1=array("a"=>"red","b"=>"green");
$a2=array("c"=>"blue","b"=>"yellow");
print_r(array_merge ($a1,$a2));
?>
```

Output: Array ([a] => red [b] => yellow [c] => blue)

Merge array recursive

```
<?php
$a1=array("a"=>"red","b"=>"green");
$a2=array("c"=>"blue","b"=>"yellow");
print_r(array_merge_recursive ($a1,$a2));
?>
```

Output: Array ([a] => red [b] => Array ([0] => green [1] => yellow) [c] => blue)

Difference between two arrays

The array_diff() function compares **the values** of two (or more) arrays, and returns the differences.

This function compares the values of two (or more) arrays, and return an array that contains the entries from *array1* that are not present in *array2* or *array3*, etc.

Syntax: `array_diff(array1, array2, array3, ...)`

Example:

```php
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"green","g"=>"blue");

$result=array_diff($a1,$a2);
print_r($result);
?>
```

Output: Array ([d] => yellow)

Get common elements among two arrays

The array_intersect() function compares **the values** of two (or more) arrays, and returns the matches.

This function compares the values of two or more arrays, and return an array that contains the entries from *array1* that are present in *array2*, *array3*, etc.

Syntax: `array_intersect(array1, array2, array3, ...)`

```php
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"green","g"=>"blue");

$result=array_intersect($a1,$a2);
print_r($result);
?>
```

Output: Array ([a] => red [b] => green [c] => blue)

Getting keys/values separately from associative arrays

The array_keys() function returns an array containing the keys.

The array_values() function returns an array containing all the values of an array.

```php
<?php

$a=array("Name"=>"Peter","Age"=>"41","Country"=>"USA");
```

```
print_r(array_keys($a));

echo "<br/>";

print_r(array_values($a));

?>
```

Output:

```
Array ( [0] => Name [1] => Age [2] => Country )
Array ( [0] => Peter [1] => 41 [2] => USA )
```

PHP Functions

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

Advantages of using Function

- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

Naming rule of PHP Function

- Function names must start with a letter or an underscore but not a number
- The function name must be unique
- The function name must not contain spaces
- It is considered a good practice to use descriptive function names.
- Functions can optionally accept parameters and return values too.

Creating and invoking Function

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

Syntax

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

Example

```
<?php
function writeMsg() {
  echo "Hello world!";
}
```

```
writeMsg(); // call the function
?>
```

Functions with Parameter

You can specify parameters when you define your function to accept input values at runtime.

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be −

Syntax:

```
function functionName($parameter1, $parameter2, …) {
  code to be executed;
}
```

Example

```
<?php
function concat($str1, $str2) {
      return $str1 . $str2;
}
$greeting = concat('Welcome ', 'Admin'); // calling function
echo $greeting;
?>
```

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand(**&**) to the variable name in either the function call or the function definition.

Following example depicts both the cases.

```php
<?php
      function addFive($num) {
        $num += 5;
      }

      function addSix(&$num) {
        $num += 6;
      }

      $orignum = 10;
      addFive( $orignum );

      echo "Original Value is $orignum<br />";

      addSix( $orignum );
      echo "Original Value is $orignum<br />";
    ?>
```

Function with Optional Parameters and Default Values

You can also create functions with optional parameters. For that, just insert the parameter name, followed by an equal (=) sign, followed by a default value.

Example

```php
<?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);
?>
```

Returning values from a Function

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays and objects.

A function cannot return multiple values;

Example

```
<?php

 function sum(int $x, int $y) {
  $z = $x + $y;
  return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

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:

Example

```
<?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:

Example

```
<?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>";
?>
```

PHP The global Keyword

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

Example

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

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

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

myTest();
```

```
myTest();
myTest();
?>
```

Form Handling

We can create and use forms in PHP. To get form data, we need to use PHP superglobals $_GET and $_POST.

The form request may be get or post. To retrieve data from get request, we need to use $_GET, for post request $_POST.

Accessing Forms using GET Method

Get request is the default form request. The data passed through get request is visible on the URL browser so it is not secured. You can send limited amount of data through get request.

File: form1.html

```
<form action="welcome.php" method="get">
Name: <input type="text" name="name"/>
<input type="submit" value="visit"/>
</form>
```

File: welcome.php

```
<?php
$name=$_GET["name"];//receiving name field value in $name variable
echo "Welcome, $name";
?>
```

Accessing Forms using POST Method

Post request is widely used to submit form that have large amount of data such as file upload, image upload, login form, registration form etc.

The data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request.

```
<!doctype html>
```

```
<html>
<body>
<?php
if(isset($_POST['submit']))
{
   $name = $_POST['name'];
   echo "User Has submitted the form and entered this name : <b> $name </b>";
   echo "<br>You can use the following form again to enter a new name.";
}
?>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  <input type="text" name="name" value="<?php echo $_POST['name'] ?? ''; ?>"><br>
  <input type="submit" name="submit" value="Submit Form"><br>
</form>

</body>
</html>
```

Difference between GET and POST method

	GET	**POST**
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs

	Never use GET when sending passwords or other sensitive information!	
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Accessing and Assigning Form Elements

Index.html

```
<!doctype html>
<html>
<body>
	<form action="savingformdata.php" method="post">
	<div style="width: 30%; float:left; background: aliceblue">
		<fieldset>
		<legend>Form Controls</legend>
		<table border="0" cellspacing="0" cellpadding="2" align="center">
			<tr>
				<td>UserId</td>
				<td><input type="text" name="Userid" style="width:200px"
value="backend-wizard@gmail.com" disabled /></td>
			</tr>
			<tr>
				<td>Firstname</td>
				<td><input type="text" name="firstname" value="Shooman" /></td>
			</tr>
			<tr>
				<td>
					Lastname
				</td>
				<td>
					<input type="text" name="lastname" />
				</td>
			</tr>
			<tr>
				<td>
					Favorite Sports
				</td>
				<td>
					<input type="checkbox" name="fsports[]" value="Football" />
Football
```

```
                                        <input type="checkbox" name="fsports[]" value="Cricket" />
Cricket
                                        <input type="checkbox" name="fsports[]" value="TableTennis"
/> Table Tennis
                                </td>
                        </tr>
                        <tr>
                                <td>
                                        Gender
                                </td>
                                <td>
                                        <input type="radio" name="gender" value="Male" /> Male
                                        <input type="radio" name="gender" value="Female" /> Female
                                        <input type="radio" name="gender" value="Others" /> Others

                                </td>
                        </tr>
                        <tr>
                                <td>
                                        UserType
                                </td>
                                <td>
                                        <select name="usertype" style="width:160px">
                                                <option value="">Select...</option>
                                                <option value="Admin">Admin</option>
                                                <option value="Accountant">Accountant</option>
                                                <option value="AccountantManager">Accountant
Manager</option>
                                                <option value="Manager">Manager</option>
                                                <option value="NormalUser">User</option>
                                        </select>
                                </td>
                        </tr>
                        <tr>
                                <td>
                                        Profile Image
                                </td>
                                <td>
                                        <input type="file" name="profileImage" multiple="true"
accept="Image/*" />
                                </td>
                        </tr>
                        <tr>
                                <td>
```

```
                                    </td>
                                    <td>
                                            <input type="submit" value="Save" /> <input type="reset"
value="Reset" /> <input type="button" value="Button" />
                                            <input type="image" src="prev-orange.png" alt="" value="" />
                                    </td>
                            </tr>
                    </table>

                    </fieldset>
            </div>

            </form>
</body>
</html>
```

savingformdata.php

```
<!doctype html>
<html>
<body>
Your Input:<br>
Firstname:<?php echo $_POST["firstname"] ?> <br>
Favorite Sports:<?php var_dump($_POST['fsports']) ?> <br>
Gender:<?php echo $_POST['gender'] ?> <br>
UserType:<?php echo $_POST['usertype'] ?> <br>
</body>
</html>
```

Form Validation

Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows −

- **Client-Side Validation** − Validation is performed on the client machine web browsers.
- **Server Side Validation** − After submitted by data, The data has sent to a server and perform validation checks in server machine.

Some useful validating functions

- **htmlspecialchars**: converts some predefined characters to HTML entities
- **stripslashes**: function removes backslashes
- **trim**: removes the white spaces
- **isset**: check variable existence
- **empty**: check empty value of variable
- **addslashes**: used to add backslashes in front of the characters
- **htmlentities**: converts characters to HTML entities
- **html_entity_decode**: converts HTML entities to characters

Form Validation Example

```
<html>

  <head>
    <style>
      .error {color: #FF0000;}
    </style>
  </head>

  <body>
    <?php
      // define variables and set to empty values
      $nameErr = $emailErr = $genderErr = $websiteErr = "";
      $name = $email = $gender = $comment = $website = "";

      if ($_SERVER["REQUEST_METHOD"] == "POST") {
        if (empty($_POST["name"])) {
          $nameErr = "Name is required";
        }else {
```

```
    $name = test_input($_POST["name"]);
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  }else {
    $email = test_input($_POST["email"]);

    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  }else {
    $website = test_input($_POST["website"]);
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  }else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  }else {
    $gender = test_input($_POST["gender"]);
  }
 }

 function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
 }
?>

<h2>Absolute classes registration</h2>

<p><span class = "error">* required field.</span></p>
```

```
    <form method = "post" action = "<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
      <table>
        <tr>
          <td>Name:</td>
          <td><input type = "text" name = "name" value="<?php echo $_POST['name'] ?? ''; ?>">
            <span class = "error">* <?php echo $nameErr;?></span>
          </td>
        </tr>

        <tr>
          <td>E-mail: </td>
          <td><input type = "text" name = "email" value="<?php echo $_POST['email'] ?? ''; ?>">
            <span class = "error">* <?php echo $emailErr;?></span>
          </td>
        </tr>

        <tr>
          <td>Time:</td>
          <td> <input type = "text" name = "website" value="<?php echo $_POST['website'] ?? ''; ?>">
            <span class = "error"><?php echo $websiteErr;?></span>
          </td>
        </tr>

        <tr>
          <td>Classes:</td>
          <td> <textarea name = "comment" rows = "5" cols = "40"><?php echo $_POST['comment'] ?? '';
?></textarea></td>
        </tr>

        <tr>
          <td>Gender:</td>
          <td>
            <input type = "radio" name = "gender" value = "female" <?php if(isset($_POST['gender']) &&
'female' == $_POST['gender']) echo "checked='checked'"; ?>>Female
            <input type = "radio" name = "gender" value = "male" <?php if(isset($_POST['gender']) &&
'male' == $_POST['gender']) echo "checked='checked'"; ?>>Male
            <span class = "error">* <?php echo $genderErr;?></span>
          </td>
        </tr>

        <td>
          <input type = "submit" name = "submit" value = "Submit">
        </td>
```

```
    </table>

  </form>

  <?php
    echo "<h2>Your given values are as:</h2>";
    echo $name;
    echo "<br>";

    echo $email;
    echo "<br>";

    echo $website;
    echo "<br>";

    echo $comment;
    echo "<br>";

    echo $gender;
  ?>

 </body>
</html>
```

File Uploading

PHP allows you to upload single and multiple files through few lines of code only.

PHP file upload features allows you to upload binary and text files both. Moreover, you can have the full control over the file to be uploaded through PHP authentication and file operation functions.

PHP's File Upload Directives

1) file_uploads = On | Off; Default value: On

- determines whether PHP scripts on the server can accept file uploads

2) max_input_time = integer; Default value: 60

-determines the maximum amount of time, in seconds, that a PHP script will spend attempting to parse input before registering a fatal error.

3) max_file_uploads = integer; Default value: 20

-sets an upper limit on the number of files which can be simultaneously uploaded

4) memory_limit = integerM; Default value:16M

-sets maximum allowable amount of memory in megabytes that a script can allocate.

5) post_max_size=integer; Default value:8M

-place an upper limit on the size of data submitted via the POST method.

6) upload_max_filesize: Default value:2M

-determines the maximum size in megabytes of an uploaded file.

Create The HTML Form

Create an HTML form that allow users to choose the image file they want to upload:

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

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

</body>
</html>
```

Some rules to follow for the HTML form above:

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

PHP $_FILES

The PHP global $_FILES contains all the information of file. By the help of $_FILES global, we can get file name, file type, file size, temp file name and errors associated with file.

Here, we are assuming that file name is *fileToUpload*.

$_FILES['fileToUpload']['name']: returns file name.

$_FILES['fileToUpload']['type']: returns MIME type of the file.

$_FILES['fileToUpload']['size']: returns size of the file (in bytes).

$_FILES['fileToUpload']['tmp_name']: returns temporary file name of the file which was stored on the server.

$_FILES['fileToUpload']['error']: returns error code associated with this file.

PHP file upload functions

PHP offers two functions specifically intended to aid in the file upload process;

- is_uploaded_file(): Tells whether the file was uploaded via HTTP POST
- move_uploaded_file(): provides a convenient way for moving an uploaded file from the temporary directory to final location.

Example

```
<?php
  if(isset($_FILES['image'])){
    $errors= array();
    $file_name = $_FILES['image']['name'];
    $file_size = $_FILES['image']['size'];
    $file_tmp = $_FILES['image']['tmp_name'];
    $file_type = $_FILES['image']['type'];

    $extensions= array("image/jpeg","image/jpg","image/png");

    if(in_array($file_type,$extensions)=== false){
      $errors[]="extension not allowed, please choose a JPEG or PNG file.";
    }

    if($file_size > 2097152) {
      $errors[]='File size cannot exceed 2 MB';
    }

    if(empty($errors)==true) {
      move_uploaded_file($file_tmp,"images/".$file_name);
      echo "Success";
    }else{
```

```
      print_r($errors);
    }
  }
?>
<html>
  <body>

    <form action = "" method = "POST" enctype = "multipart/form-data">
      <input type = "file" name = "image" />
      <input type = "submit"/>

      <ul>
        <li>Sent file: <?php echo $_FILES['image']['name']??"";  ?>
        <li>File size: <?php echo $_FILES['image']['size']??"";  ?>
        <li>File type: <?php echo $_FILES['image']['type']??"" ?>
      </ul>
                <img src="images/<?php echo $_FILES['image']['name']??""; ?>" alt=""
width="100px" />
    </form>
  </body>
</html>
```

File Handling

PHP File System allows us to create file, read file line by line, read file character by character, write file, append file, delete file and close file.

Opening and Closing Files

The PHP **fopen**() function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

S.No	Mode & Purpose
1	r Opens the file for reading only. Places the file pointer at the beginning of the file.
2	r+

	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
3	w Opens the file for writing only. Erases the contents of the file or creates a new file if it does not exist. File pointer starts at the beginning of the file
4	w+ Opens the file for reading and writing only. Erases the contents of the file or creates a new file if it does not exist. File pointer starts at the beginning of the file
5	a Opens the file for writing only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
6	a+ Opens the file for reading and writing only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist

If an attempt to open a file fails then **fopen** returns a value of false otherwise it returns a file pointer which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the **fclose**() function. The **fclose**() function requires a file pointer as its argument and then returns true when the closure succeeds or false if it fails.

Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

Example

```
<?php
  $filename = "newfile.txt";
  $file = fopen( $filename, "w" );

  if( $file == false ) {
    echo ( "Error in opening new file" );
    exit();
  }
  fwrite( $file, "This is  a simple test\n" );
  fclose( $file );
?>
<html>
  <head>
    <title>Writing a file using PHP</title>
  </head>
  <body>
    <?php
      $filename = "newfile.txt";
      $file = fopen( $filename, "r" );

      if( $file == false ) {
        echo ( "Error in opening file" );
        exit();
      }

      $filesize = filesize( $filename );
      $filetext = fread( $file, $filesize );
      fclose( $file );
      echo ( "File size : $filesize bytes <br>" );
      echo ( "$filetext <br>" );
      echo("file name: $filename <br>");
    ?>
  </body>
```

```
</html>
```

Renaming Files

We can rename files and directories using the PHP's **rename**() function. **rename**() will take two parameters, first as an old filename and second as a new file name.

```
<?php
if (file_exists('originalfile.txt'))
{
	$renamed= rename('originalfile.txt', 'copiedfile.txt');

	if ($renamed)
	{
		echo "The file has been successfully renamed";
	}
	else
	{
		echo "The file has not been successfully renamed";
	}
}
else
{
	echo "The original file that you want to rename doesn't exist";
}
?>
```

Removing Files

We can delete files or directories using the PHP's **unlink**() function. **unlink**() will take single parameter as a filename to delete.

```
<?php
$filename = 'readme.txt';
if (file_exists($filename))
{
	if (unlink($filename)) {
		echo 'The file ' . $filename . ' was deleted successfully!';
	} else {
		echo 'There was a error deleting the file ' . $filename;
	}
}
else
{
```

```
        echo 'The file ' . $filename . ' that you want to delete does not exist';
}
?>
```

PHP Include and Require Files

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

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

- `require` will produce a fatal error (E_COMPILE_ERROR) and stop the script
- `include` will only produce a warning (E_WARNING) and the script will continue

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

Syntax

```
include 'filename';
or
require 'filename';
```

Footer.php

```
<?php
echo "<p>Copyright &copy; lumbinicitycollege.com</p>";
?>
```

Home.php

```
<html>
<body>

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

</body>
</html>
```

Output

Welcome to my home page!

Some text.

Some more text.

include_once():

This function is utilized to add a file only once at a time. If the code from a file has been already included then it will not be added again if we use include_once(). If it can't locate a specified file at that time it will generate a warning message but it will not stop the content execution.

require_once():

In case the code from a php file has been already included then it will not be included again if we use require_once(). It implies require_once() will add the file just once at a time. If it can't locate a specified file, at that time it will generate a fatal error but it will stop the content execution.

PHP Date and Time Function

Timestamp

A timestamp is a numeric value in seconds between the current time and value as at 1st January , 1970 00:00:00 Greenwich Mean Time (GMT)

Time Function

The time() function returns the current time as Unix timestamp. The UNIX timestamp measures time as a number of seconds since midnight of Greenwich Mean Time on January 1, 1970.

Example:

```
<!DOCTYPE html>

<html>

<body>


<?php

 $currenttime = time();

 $timeAfterOneWeek = time() + (7*24*60*60);

 $timeAfterFiftyDays = time() + (50*24*60*60);

 echo("Current timestamp is : " . $currenttime . "<br>");
```

```
 echo("Timestamp after one week is : " . $timeAfterOneWeek . "<br>");

 echo("Timestamp after 50 days is : " . $timeAfterFiftyDays . "<br>");

?>


</body>

</html>
```

Date Function

The **date()** function accepts a format string as a parameter, formats the local date/time in the specified format and returns the result.

Syntax

```
date($format, $timestamp)

Here,
$format is mandatory which specifies the format in which we want
the output date string.

$timestamp is optional which represents the timestamp of required
date
```

Formatting the Dates and Times with PHP

Some date-related formatting characters that are commonly used in format string:

- d - Represent day of the month; two digits with leading zeros (01 or 31)
- D - Represent day of the week in text as an abbreviation (Mon to Sun)
- m - Represent month in numbers with leading zeros (01 or 12)
- M - Represent month in text, abbreviated (Jan to Dec)
- y - Represent year in two digits (08 or 14)
- Y - Represent year in four digits (2008 or 2014)

The parts of the date can be separated by inserting other characters, like hyphens (`-`), dots (`.`), slashes (`/`), or spaces to add additional visual formatting.

Example

```php
<?php
echo date("d/m/Y") . "<br>";
```

```
echo date("d-m-Y") . "<br>";
echo date("d.m.Y");
?>
```

Similarly you can use the following characters to format the time string:

- h - Represent hour in 12-hour format with leading zeros (01 to 12)
- H - Represent hour in in 24-hour format with leading zeros (00 to 23)
- i - Represent minutes with leading zeros (00 to 59)
- s - Represent seconds with leading zeros (00 to 59)
- a - Represent lowercase ante meridiem and post meridiem (am or pm)
- A - Represent uppercase Ante meridiem and Post meridiem (AM or PM)

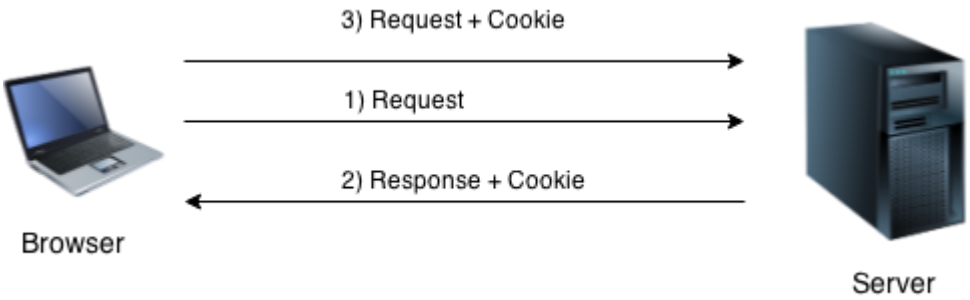
Example

```
<?php
date_default_timezone_set('Asia/Kathmandu');
echo date("h:i:s") . "<br>";
echo date("M d, Y h:i:s A") . "<br>";
echo date("h:i a");
?>
```

PHP Cookie

PHP cookie is a small piece of information which is stored at client browser. It is used to recognize the user.

Cookie is created at server side and saved to client browser. Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side.



In short, cookie can be created, sent and received at server end.

Creating cookie

PHP setcookie() function is used to set cookie with HTTP response. Once cookie is set, you can access it by $_COOKIE superglobal variable.

syntax

```
<?php

setcookie(cookie_name, cookie_value, [expiry_time], [cookie_path], [domain],
[secure]);

?>
```

HERE,

- php"setcookie" is the PHP function used to create the cookie.
- "cookie_name" is the name of the cookie that the server will use when retrieving its value from the $_COOKIE array variable. It's mandatory.
- "cookie_value" is the value of the cookie and its mandatory
- "[expiry_time]" is optional; it can be used to set the expiry time for the cookie such as 1 hour. The time is set using the PHP time() functions plus or minus a number of seconds greater than 0 i.e. time() + 3600 for 1 hour.
- "[cookie_path]" is optional; it can be used to set the cookie path on the server. The forward slash "/" means that the cookie will be made available on the entire domain. Sub directories limit the cookie access to the subdomain.
- "[domain]" is optional, it can be used to specify the domain for which the cookie will be available.
- "[secure]" is optional, the default is false. It is used to determine whether the cookie is sent via https if it is set to true or http if it is set to false.

Retrieving the Cookie value

PHP $_COOKIE superglobal variable is used to get cookie.

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); //
86400 = 1 day
?>
<html>
<body>

<?php
```

```php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Delete cookie

If you set the expiration date in past, cookie will be deleted.

Example

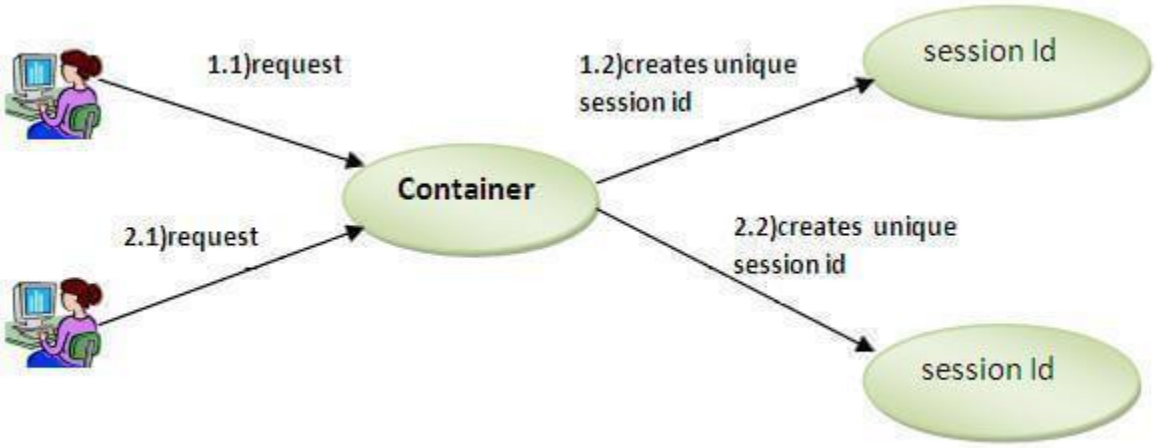
```php
<?php
setcookie ("CookieName", "", time() - 3600);// set the expiration date to one hour ago
?>
```

PHP Session

PHP session is used to store and pass information from one page to another temporarily (until user close the website).

PHP session technique is widely used in shopping websites where we need to store and pass cart information e.g. username, product code, product name, product price etc from one page to another.

PHP session creates unique user id for each browser to recognize the user and avoid conflict between multiple browsers.



Why and when to use Sessions?

- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL

Create and retrieve PHP Session

A session is started with the `session_start()` function.

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

Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
```

```php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";


// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Destroying Session Variables

The session_destroy() function is used to destroy the whole Php session variables.

If you want to destroy only a session single item, you use the unset() function.

```php
<?php

 session_destroy(); //destroy entire session

?>
<?php

unset($_SESSION['product']); //destroy product session item

?>
```

Sending Email

PHP mail is the built in PHP function that is used to send emails from PHP scripts.

The mail function accepts the following parameters:

- Email address
- Subject
- Message
- CC or BC email addresses

Syntax

```
<?php
mail($to_email_address,$subject,$message,[$headers],[$parameters]);
?>
```

HERE,

- “$to_email_address” is the email address of the mail recipient
- “$subject” is the email subject
- “$message” is the message to be sent.
- “[$headers]” is optional, it can be used to include information such as CC, BCC

Example

```
<html>

  <head>
    <title>Sending HTML email using PHP</title>
  </head>

  <body>

    <?php
      $to = "xyz@somedomain.com";
      $subject = "This is subject";

      $message = "<b>This is HTML message.</b>";
      $message .= "<h1>This is headline.</h1>";

      $header = "From:abc@somedomain.com \r\n";
      $header .= "Cc:afgh@somedomain.com \r\n";
      $header .= "MIME-Version: 1.0\r\n";
      $header .= "Content-type: text/html\r\n";

      $retval = mail ($to,$subject,$message,$header);

      if( $retval == true ) {
        echo "Message sent successfully...";
      }else {
        echo "Message could not be sent...";
      }
    ?>
```

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