

# PHP

## Audience

This **PHP course** is designed for PHP programmers who are completely unaware of PHP concepts but they have basic understanding on computer programming.

## Prerequisites

Before proceeding with this course, you should have at least basic understanding of computer programming, Internet, Database, and MySQL etc. is very helpful.

## Table of Contents:

### Introduction

- Environment Setup
- Syntax Overview
- Variable Types
- Date & Time
- Constants Types
- Operator Types
- Decision Making
- Loop Types
- Arrays
- Strings
- Web Concepts
- GET & POST Methods
- File Inclusion
- Files & I/O
- Functions
- Cookies
- Sessions
- Sending Emails using PHP
- File UploadingError! Reference source not found.
- Coding Standard

# Introduction

The **PHP Hypertext Preprocessor (PHP)** is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web-based software applications. This course helps you to build your base with PHP.

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 e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy 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 functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through 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

## Applications of PHP

As mentioned before, PHP is one of the most widely used language over the web. I'm going to list few of them here:

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

## "Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

Line	Code
1	<html>
2	<head>
3	<title>Hello World</title>
4	</head>
5	<body>
6	<?php echo "Hello, World!";?>
7	</body>
8	</html>

It will produce 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 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 ATE are recognised by the PHP Parser.

```
<?php PHP code goes here ?>
<? PHP code goes here ?>
<script language = "php"> PHP code goes here </script>
```

A most common tag is the <?php...?> and we will also use the same tag in this course.

From the next chapter we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

# Environment Setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server:** PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here: <https://httpd.apache.org/download.cgi>
- **Database:** PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here: <https://www.mysql.com/downloads/>
- **PHP Parser:** In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser. This course will guide you how to install PHP parser on your computer.

## Installation

Even better if you can install XAMPP that come with all these main components (plus others). Refer to the course setup guide.

### Windows IIS Configuration

To configure IIS on your Windows machine you can refer your IIS Reference Manual shipped along with IIS.

# Syntax Overview

This chapter will give you an idea of very basic syntax of PHP and very important to make your PHP foundation strong.

## Escaping to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP'. There are four ways to do this:

### Canonical PHP tags

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

### Short-open (SGML-style) tags

Short or short-open tags look like this:

```
<?...?>
```

Short tags are, as one might expect, the shortest option. You must do one of two things to enable PHP to recognize the tags:

- Choose the `--enable-short-tags` configuration option when you're building PHP.
- Set the `short_open_tag` setting in your `php.ini` file to `on`. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

### ASP-style tags

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this:

```
<%...%>
```

To use ASP-style tags, you will need to set the configuration option in your `php.ini` file.

### HTML script tags

HTML script tags look like this:

```
<script language = "PHP">...</script>
```

# 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.

Line	Code
1	<?php
2	# This is a comment, and
3	# This is the second line of the comment
4	
5	// This is a comment too. Each style comments only
6	print "An example with single line comments";
7	?>

**Multi-lines printing:** Here are the examples to print multiple lines in a single print statement:

Line	Code
1	<?php
2	# First Example
3	print <<<END
4	This uses the "here document" syntax to output
5	multiple lines with \$variable interpolation. Note
6	that the here document terminator must appear on a
7	line with just a semicolon no extra whitespace!
8	END;
9	
10	# Second Example
11	print "This spans
12	multiple lines. The newlines will be
13	output as well";
14	?>

**Multi-lines comments:** They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

Line	Code
1	<?php
2	/* This is a comment with multiline
3	Author : Mohammad Mohtashim
4	Purpose: Multiline Comments Demo
5	Subject: PHP
6	*/
7	
8	print "An example with multi line comments";
9	?>

## PHP is whitespace insensitive

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

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:

Line	Code
1	<code>&lt;?php</code>
2	<code>    \$four = 2 + 2; // single spaces</code>
3	<code>    \$four &lt;tab&gt;=&lt;tab2&gt;&lt;tab&gt;+&lt;tab&gt;2 ; // spaces and tabs</code>
4	<code>    \$four =</code>
5	<code>    2+</code>
6	<code>    2; // multiple lines</code>
7	<code>?&gt;</code>

## PHP is case sensitive

Like most famous programming languages, PHP is a case sensitive language. Try out following example:

Line	Code
1	<code>&lt;html&gt;</code>
2	<code>&lt;body&gt;</code>
3	<code>    &lt;?php</code>
4	<code>        \$capital = 67;</code>
5	<code>        print("Variable capital is \$capital&lt;br&gt;");</code>
6	<code>        print("Variable CaPiTaL is \$CaPiTaL&lt;br&gt;");</code>
7	<code>    ?&gt;</code>
8	<code>&lt;/body&gt;</code>
9	<code>&lt;/html&gt;</code>

This will produce the following result:

```
Variable capital is 67
Variable CaPiTaL is
```

## Statements are 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!";
```

## Expressions are combinations of tokens

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables (\$two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

## Braces make blocks

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent:

Line	Code
1	<?php
2	if (3 == 2 + 1)
3	print("Good - I haven't totally lost my mind. ");
4	
5	if (3 == 2 + 1) {
6	print("Good - I haven't totally");
7	print("lost my mind. ");
8	}
9	?>

## Running PHP Script from Command Prompt

You can run your PHP script on your command prompt. Assuming you have following content in test.php file

Line	Code
1	<?php
2	echo "Hello PHP!!!!!!";
3	?>

Now run this script as command prompt as follows:

```
$ php test.php
```

It will produce the following result:

```
Hello PHP!!!!!!
```

Hope now you have basic knowledge of PHP Syntax.



# 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.

- All variables in PHP are denoted with a leading dollar sign (\$).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- 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.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

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 first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

We will explain only simple data type in this module. Array and Objects will be explained separately.

## Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type. They correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so:

Line	Code
1	<?php
2	\$int_var = 12345;
3	\$another_int = -12345 + 12345;
4	?>

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimal have a leading 0x.

For most common platforms, the largest integer is  $(2^{31} - 1)$  (or 2,147,483,647), and the smallest (most negative) integer is  $-(2^{31} - 1)$  (or -2,147,483,647).

## Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code:

Line	Code
1	<?php
2	\$many = 2.2888800;
3	\$many_2 = 2.2111200;
4	\$few = \$many + \$many_2;
5	
6	print("\$many + \$many_2 = \$few  ");
7	?>

It produces the following browser output:

```
2.28888 + 2.21112 = 4.5
```

## Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

Line	Code
1	<?php
2	if (TRUE)
3	print("This will always print ");
4	else
5	print("This will never print ");
6	?>

## Interpreting other types as Booleans

Here are the rules for determine the "truth" of any value not already of the Boolean type:

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

Line	Code
1	<?php
2	>true_num = 3 + 0.14159;
3	>true_str = "Tried and true"
4	>true_array[49] = "An array element";
5	>false_array = array();
6	>false_null = NULL;
7	>false_num = 999 - 999;
8	>false_str = "";
9	?>

## NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed:

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

# Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

Line	Code
1	<?php
2	\$string_1 = "This is a string in double quotes";
3	\$string_2 = 'This is a somewhat longer, singly quoted string';
4	\$string_39 = "This string has thirty-nine characters";
5	\$string_0 = ""; // a string with zero characters
6	?>

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

Line	Code
1	<?php
2	\$variable = "name";
3	\$literally = 'My \$variable will not print!';
4	
5	print(\$literally);
6	print " ";
7	
8	\$literally = "My \$variable will print!";
9	print(\$literally);
10	?>

This will produce following result:

```
My $variable will not print!  
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with \$) are replaced with string representations of their values.

The escape-sequence replacements are:

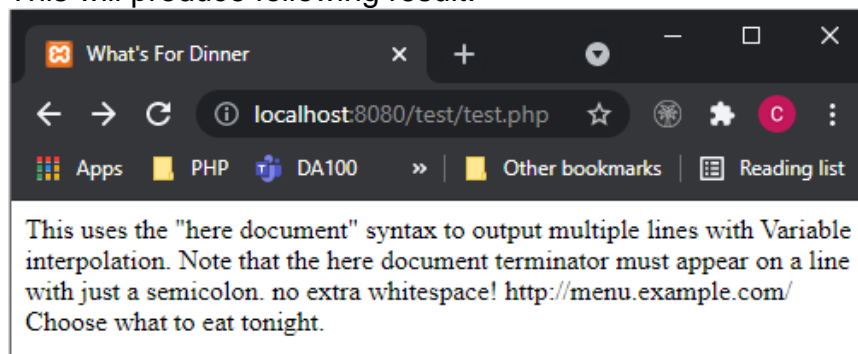
- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \\$ is replaced by the dollar sign itself (\$)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

## Here Document

You can assign multiple lines to a single string variable using here document:

Line	Code
1	<?php
2	\$channel =<<<_XML_
3	
4	<channel>
5	<title>What's For Dinner</title>
6	<link>http://menu.example.com/ </link>
7	<description>Choose what to eat tonight.</description>
8	</channel>
9	_XML_;
10	
11	echo <<<END
12	This uses the "here document" syntax to output
13	multiple lines with Variable interpolation. Note
14	that the here document terminator must appear on a
15	line with just a semicolon. no extra whitespace!
16	
17	END;
18	print \$channel;
19	?>

This will produce following result:



## Variable Scope

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:

- Local variables
- Function parameters
- Global variables
- Static variables

## 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.

## Date & Time

Dates are so much part of everyday life that it becomes easy to work with them without thinking. PHP also provides powerful tools for date arithmetic that make manipulating dates easy.

### Getting the Time Stamp with time()

PHP's **time()** function gives you all the information that you need about the current date and time. It requires no arguments but returns an integer.

The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the UNIX epoch, and the number of seconds that have elapsed since then is referred to as a time stamp.

Line	Code
1	<code>&lt;?php</code>
2	<code>  print time();</code>
3	<code>?&gt;</code>

This will produce the following result:

```
1480930103
```

This is something difficult to understand. But PHP offers excellent tools to convert a time stamp into a form that humans are comfortable with.

### Converting a Time Stamp with getdate()

The function **getdate()** optionally accepts a time stamp and returns an associative array containing information about the date. If you omit the time stamp, it works with the current time stamp as returned by time().

Following table lists the elements contained in the array returned by getdate().

No.	Key & Description	Example
1	<b>seconds</b> Seconds past the minutes (0-59)	20
2	<b>minutes</b> Minutes past the hour (0 - 59)	29
3	<b>hours</b> Hours of the day (0 - 23)	22
4	<b>mday</b> Day of the month (1 - 31)	11

5	<b>wday</b> Day of the week (0 - 6)	4
6	<b>mon</b> Month of the year (1 - 12)	7
7	<b>year</b> Year (4 digits)	1997
8	<b>yday</b> Day of year ( 0 - 365 )	19
9	<b>weekday</b> Day of the week	Thursday
10	<b>month</b> Month of the year	January
11	<b>0</b> Timestamp	948370048

Now you have complete control over date and time. You can format this date and time in whatever format you want.

### Example

Try out following example:

Line	Code
1	<code>&lt;?php</code>
2	<code>  \$date_array = getdate();</code>
3	
4	<code>  foreach ( \$date_array as \$key =&gt; \$val ) {</code>
5	<code>    print "\$key = \$val&lt;br /&gt;";</code>
6	<code>  }</code>
7	
8	<code>  \$formated_date = "Today's date: ";</code>
9	<code>  \$formated_date .= \$date_array['mday'] . "/";</code>
10	<code>  \$formated_date .= \$date_array['mon'] . "/";</code>
11	<code>  \$formated_date .= \$date_array['year'];</code>
12	
13	<code>  print \$formated_date;</code>
14	<code>?&gt;</code>

This will produce following result:

```
seconds = 10
minutes = 29
hours = 9
mday = 5
wday = 1
mon = 12
year = 2016
```

```
yday = 339
weekday = Monday
month = December
0 = 1480930150
Today's date: 5/12/2016
```

## Converting a Time Stamp with date()

The **date()** function returns a formatted string representing a date. You can exercise an enormous amount of control over the format that date() returns with a string argument that you must pass to it.

```
date(format,timestamp)
```

The date() optionally accepts a time stamp if omitted then current date and time will be used. Any other data you include in the format string passed to date() will be included in the return value.

Following table lists the codes that a format string can contain:

No.	Format	Description	Example
1	<b>a</b>	'am' or 'pm' lowercase	pm
2	<b>A</b>	'AM' or 'PM' uppercase	PM
3	<b>d</b>	Day of month, a number with leading zeroes	20
4	<b>D</b>	Day of week (three letters)	Thu
5	<b>F</b>	Month name	January
6	<b>h</b>	Hour (12-hour format - leading zeroes)	12
7	<b>H</b>	Hour (24-hour format - leading zeroes)	22
8	<b>g</b>	Hour (12-hour format - no leading zeroes)	12
9	<b>G</b>	Hour (24-hour format - no leading zeroes)	22
10	<b>i</b>	Minutes ( 0 - 59 )	23
11	<b>j</b>	Day of the month (no leading zeroes)	20
12	<b>l (Lower 'L')</b>	Day of the week	Thursday
13	<b>L</b>	Leap year ('1' for yes, '0' for no)	1



14	<b>m</b>	Month of year (number - leading zeroes)	1
15	<b>M</b>	Month of year (three letters)	Jan
16	<b>r</b>	The RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
17	<b>n</b>	Month of year (number - no leading zeroes)	2
18	<b>s</b>	Seconds of hour	20
19	<b>U</b>	Time stamp	948372444
20	<b>y</b>	Year (two digits)	06
21	<b>Y</b>	Year (four digits)	2006
22	<b>z</b>	Day of year (0 - 365)	206
23	<b>Z</b>	Offset in seconds from GMT	+5

### Example

Try out following example:

Line	Code
1	<?php
2	<code>print date("m/d/y G.i:s&lt;br&gt;", time());</code>
3	<code>echo "&lt;br&gt;";</code>
4	<code>print "Today is ";</code>
5	<code>print date("j of F Y, \a\\t g.i a", time());</code>
6	?>

This will produce following result:

```
12/05/16 9:29:47
Today is 5 2016f December 2016 at 9:29 am
```

Hope you have good understanding on how to format date and time according to your requirement.

# Constants Types

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use `define()` function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a `$`. You can also use the function `constant()` to read a constant's value if you wish to obtain the constant's name dynamically.

## constant() function

As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

## constant() example

Line	Code
1	<code>&lt;?php</code>
2	<code>define("MINSIZE", 50);</code>
3	
4	<code>echo MINSIZE;</code>
5	<code>echo constant("MINSIZE"); // same thing as the previous line</code>
6	<code>?&gt;</code>

Only scalar data (boolean, integer, float and string) can be contained in constants.

## Differences between constants and variables are

- There is no need to write a dollar sign (\$) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the `define()` function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

## Valid and invalid constant names

Line	Code
1	<?php
2	// Valid constant names
3	define("ONE",      "first thing");
4	define("TWO2",     "second thing");
5	define("THREE_3",  "third thing");
6	define("__THREE__", "third value");
7	
8	// Invalid constant names
9	define("2TWO",     "second thing");
10	?>

## PHP Magic constants

PHP provides a large number of predefined constants to any script which it runs.

There are five magical constants that change depending on where they are used. For example, the value of `__LINE__` depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:

A few "magical" PHP constants are given below:

No.	Name & Description
1	<u><b>__LINE__</b></u> The current line number of the file.
2	<u><b>__FILE__</b></u> The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, <b>__FILE__</b> always contains an absolute path whereas in older versions it contained relative path under some circumstances.
3	<u><b>__FUNCTION__</b></u> The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
4	<u><b>__CLASS__</b></u> The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
5	<u><b>__METHOD__</b></u> The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

# Operator Types

**What is Operator?** Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

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

Let's have a look on all operators one by one.

## Arithmetic Operators

There are following arithmetic operators supported by PHP language:

Assume variable A holds 10 and variable B holds 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 numerator by de-numerator	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

## Comparison Operators

There are following comparison operators supported by PHP language

Assume variable A holds 10 and variable B holds 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.

## Logical Operators

There are following logical operators supported by PHP language

Assume variable A holds 10 and variable B holds 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.

## Assignment Operators

There are following assignment operators supported by PHP language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign 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

## Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax:

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

## Operators Categories

All the operators we have discussed above can be categorised into following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

## Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example  $x = 7 + 3 * 2$ ; Here  $x$  is assigned 13, not 20 because operator  $*$  has higher precedence than  $+$  so it first get multiplied with  $3*2$  and then adds into 7.

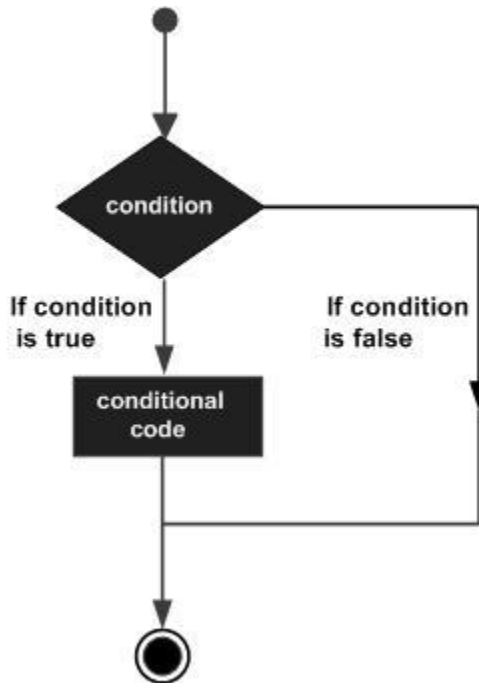
Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

# Decision Making

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 following three decision making statements:



- **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
- **elseif statement:** is used with the if...else statement to execute a set of code if **one** of the several condition is true
- **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.

### Syntax

```
if (condition)
    code to be executed if condition is true;
else
    code to be executed if condition is false;
```



## Example

The following example will output "Have a nice weekend!" if the current day is Friday, Otherwise, it will output "Have a nice day!":

Line	Code
1	<html>
2	<body>
3	<?php
4	\$d = date("D");
5	if (\$d == "Fri")
6	echo "Have a nice weekend!";
7	else
8	echo "Have a nice day!";
9	?>
10	</body>
11	</html>

It will produce the following result:

Have a nice weekend!

## The Elself Statement

If you want to execute some code if one of the several conditions are true use the elseif statement

### Syntax

```
if (condition)
    code to be executed if condition is true;
elseif (condition)
    code to be executed if condition is true;
else
    code to be executed if condition is false;
```

### Example

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!":

Line	Code
1	<html>
2	<body>
3	<?php
4	\$d = date("D");
5	if (\$d == "Fri")
6	echo "Have a nice weekend!";
7	elseif (\$d == "Sun")
8	echo "Have a nice Sunday!";
9	else
10	echo "Have a nice day!";
11	?>
12	</body>
13	</html>

It will produce the following result:

Have a nice Weekend!

# The Switch Statement

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.

## Syntax

```
switch (expression) {  
    case label1:  
        code to be executed if expression = label1;  
        break;  
    case label2:  
        code to be executed if expression = label2;  
        break;  
    default:  
        code to be executed if expression is different from both label1 and  
        label2;  
}
```

## Example

The *switch* statement works in an unusual way. First it evaluates 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 or if none of the label matches then statement will execute any specified default code.

Line	Code
1	<html>
2	<body>
3	<?php
4	\$d = date("D");
5	
6	switch (\$d) {
7	case "Mon": echo "Today is Monday";
8	break;
9	case "Tue": echo "Today is Tuesday";
10	break;
11	case "Wed": echo "Today is Wednesday";
12	break;
13	case "Thu": echo "Today is Thursday";
14	break;
15	case "Fri": echo "Today is Friday";
16	break;
17	case "Sat": echo "Today is Saturday";
18	break;
19	case "Sun": echo "Today is Sunday";
20	break;
21	default: echo "Wonder which day is this ?";
22	}
23	?>
24	</body>
25	</html>

It will produce the following result:

Today is Monday

## Loop Types

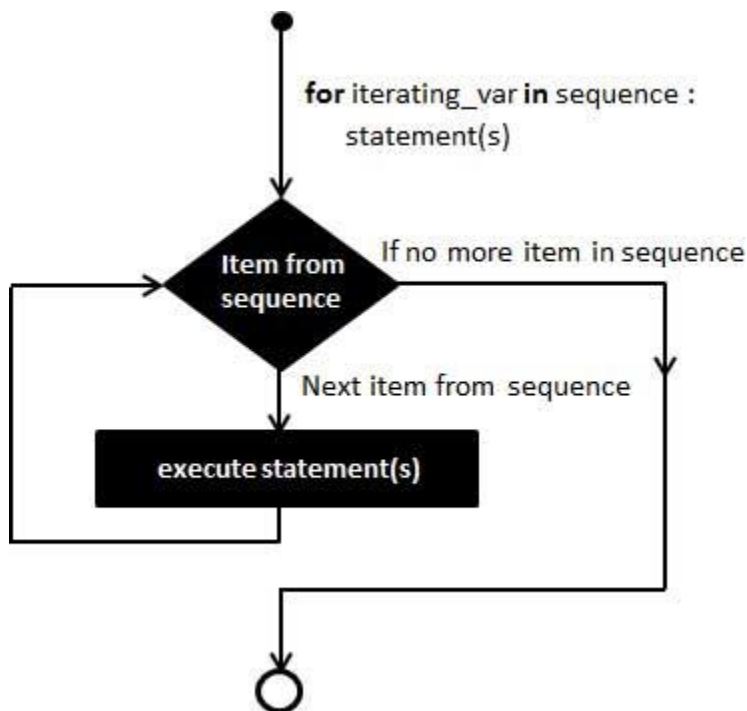
Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **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.

We will discuss about **continue** and **break** keywords used to control the loops execution.

### 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;  
}
```

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 traditional to name it \$i.

## Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop:

Line	Code
1	<html>
2	<body>
3	<?php
4	\$a = 0;
5	\$b = 0;
6	
7	for( \$i = 0; \$i<5; \$i++ ) {
8	\$a += 10;
9	\$b += 5;
10	}
11	
12	echo ("At the end of the loop a = \$a and b = \$b" );
13	?>
14	</body>
15	</html>

This will produce the following result:

At the end of the loop a = 50 and b = 25

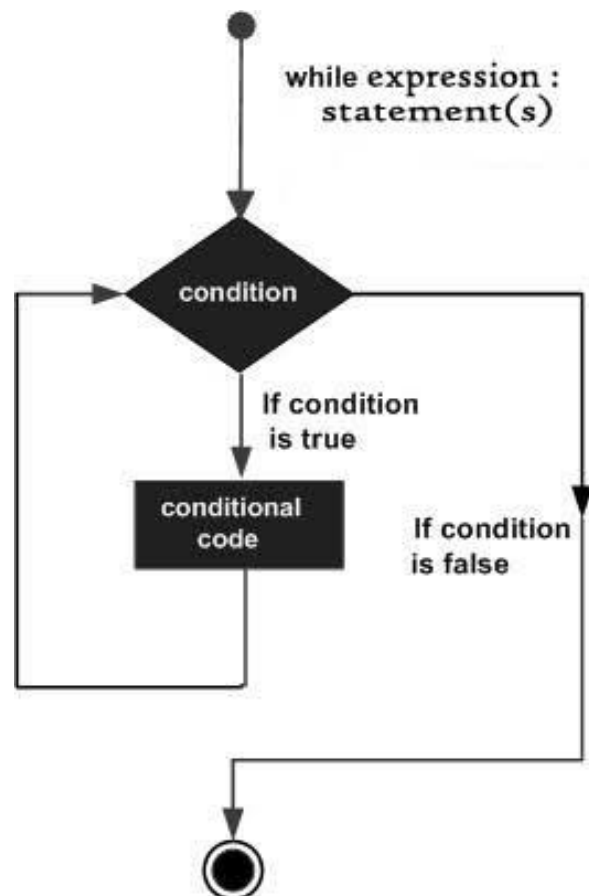
## The while loop statement

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

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

Line	Code
1	<html>
2	<body>
3	<?php
4	\$i = 0;
5	\$num = 50;
6	
7	while( \$i < 10) {
8	\$num--;
9	\$i++;
10	}
11	
12	echo ("Loop stopped at i = \$i and num = \$num" );
13	?>
14	</body>
15	</html>

This will produce the following result:

Loop stopped at i = 10 and num = 40

## 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

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:

Line	Code
1	<html>
2	<body>
3	<?php
4	\$i = 0;
5	\$num = 0;
6	do {
7	\$i++;
8	}while( \$i < 10 );
9	echo ("Loop stopped at i = \$i" );
10	?>
11	</body>
12	</html>

This will produce the following result:

Loop stopped at i = 10

## 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

Try out following example to list out the values of an array.

Line	Code
1	<html>
2	<body>
3	<?php
4	\$array = array( 1, 2, 3, 4, 5 );
5	
6	foreach( \$array as \$value ) {
7	echo "Value is \$value  ";
8	}
9	?>
10	</body>
11	</html>

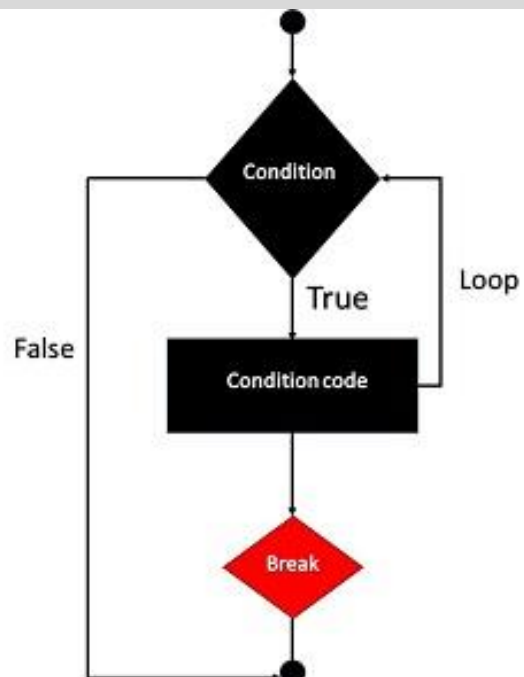
This will produce the following result:

```
Value is 1  
Value is 2  
Value is 3  
Value is 4  
Value is 5
```

## 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 immediate statement to the loop will be executed.



## Example

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

Line	Code
1	<html>
2	<body>
3	<?php
4	<i>\$i</i> = 0;
5	
6	while( <i>\$i</i> < 10) {
7	<i>\$i</i> ++;
8	if( <i>\$i</i> == 3 ) break;
9	}
10	echo ("Loop stopped at i = <i>\$i</i> " );
11	?>
12	</body>
13	</html>

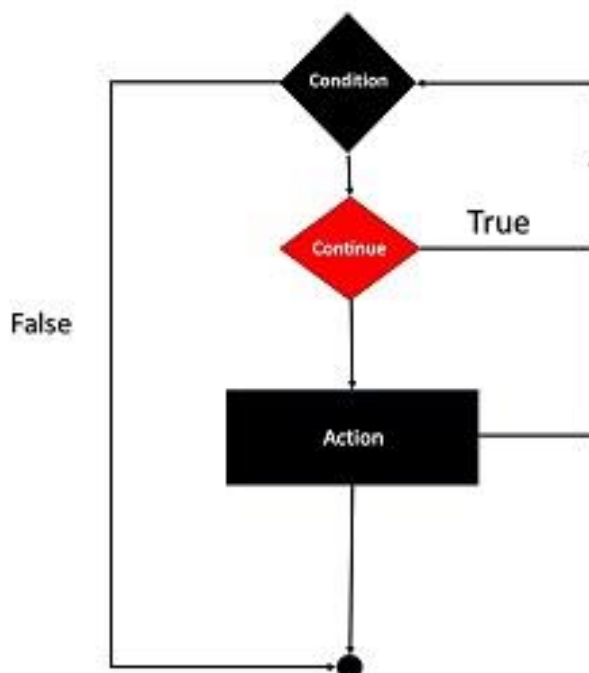
This will produce the following result:

Loop stopped at i = 3

## 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. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



## Example

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

Line	Code
1	<html>
2	<body>
3	<?php
4	\$array = array( 1, 2, 3, 4, 5 );
5	
6	foreach( \$array as \$value ) {
7	if( \$value == 3 ) continue;
8	echo "Value is \$value  ";
9	}
10	?>
11	</body>
12	</html>

This will produce the following result:

```
Value is 1
Value is 2
Value is 4
Value is 5
```



# Arrays

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's easy to define an array of 100 length.

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

- **Numeric array:** An array with a numeric index. Values are stored and accessed in linear fashion.
- **Associative array:** An array with strings as index. This store element values in association with key values rather than in a strict linear index order.
- **Multidimensional array:** An array containing one or more arrays and values are accessed using multiple indices

**NOTE:** Built-in array functions is given in function reference [PHP Array Functions](#)

## Numeric Array

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

### Example

Following is the example showing how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

Line	Code
1	<html>
2	<body>
3	<?php
4	/* First method to create array. */
5	\$numbers = array( 1, 2, 3, 4, 5);
6	
7	foreach( \$numbers as \$value ) {
8	echo "Value is \$value  ";
9	}
10	
11	/* Second method to create array. */
12	\$numbers[0] = "one";
13	\$numbers[1] = "two";
14	\$numbers[2] = "three";
15	\$numbers[3] = "four";
16	\$numbers[4] = "five";
17	
18	foreach( \$numbers as \$value ) {
19	echo "Value is \$value  ";
20	}
21	?>
22	</body>
23	</html>

This will produce the following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```

## 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 string 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 employee names as the keys in our associative array, and the value would be their respective salary.

**NOTE:** Don't keep associative array inside double quote while printing otherwise it would not return any value.

### Example

Line	Code
1	<html>
2	<body>
3	<?php
4	\$salaries = array("mohammad"=>2000, "qadir"=>1000, "zara"=>500);
5	echo "Salary of mohammad is ". \$salaries['mohammad'] . " ";
6	echo "Salary of qadir is ". \$salaries['qadir'] . " ";
7	echo "Salary of zara is ". \$salaries['zara'] . " ";
8	
9	/* Another method to create Associative Array. */
10	\$salaries['mohammad'] = "high";
11	\$salaries['qadir'] = "medium";
12	\$salaries['zara'] = "low";
13	echo "Salary of mohammad is ". \$salaries['mohammad'] . " ";
14	echo "Salary of qadir is ". \$salaries['qadir'] . " ";
15	echo "Salary of zara is ". \$salaries['zara'] . " ";
16	?>
17	</body>
18	</html>

This will produce the following result:

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

## Multidimensional Arrays

A 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 indexes.

### Example

In this example we create a two-dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

Line	Code
1	<html>
2	<body>
3	<?php
4	\$marks = array (
5	"mohammad" => array (
6	"physics" => 35,
7	"maths" => 30,
8	"chemistry" => 39
9	),
10	"qadir" => array (
11	"physics" => 30,
12	"maths" => 32,
13	"chemistry" => 29
14	),
15	"zara" => array (
16	"physics" => 31,
17	"maths" => 22,
18	"chemistry" => 39
19	)
20	);
21	
22	/* Accessing multi-dimensional array values */
23	echo "Marks for mohammad in physics : " ;
24	echo \$marks['mohammad']['physics'] . " ";
25	
26	echo "Marks for qadir in maths : " ;
27	echo \$marks['qadir']['maths'] . " ";
28	
29	echo "Marks for zara in chemistry : " ;
30	echo \$marks['zara']['chemistry'] . " ";
31	?>
32	</body>
33	</html>

This will produce the following result:

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

# Strings

They are sequences of characters, like "PHP supports string operations".

**NOTE:** Built-in string functions is given in function reference [PHP String Functions](#)

Following are valid examples of string:

```
$string_1 = "This is a string in double quotes";  
$string_2 = "This is a somewhat longer, singly quoted string";  
$string_39 = "This string has thirty-nine characters";  
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

Line	Code
1	<?php
2	\$variable = "name";
3	\$literally = 'My \$variable will not print!\n';
4	
5	print(\$literally);
6	print " ";
7	
8	\$literally = "My \$variable will print!\n";
9	
10	print(\$literally);
11	?>

This will produce the following result:

```
My $variable will not print!\n  
My name will print!\n
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with \$) are replaced with string representations of their values.

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 (\)

## String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator:

Line	Code
1	<?php
2	\$string1="Hello World";
3	\$string2="1234";
4	
5	echo \$string1 . " " . \$string2;
6	?>

This will produce the following result:

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

## Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

Line	Code
1	<?php
2	echo strlen("Hello world!");
3	?>

This will produce the following result:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

## Using the strpos() function

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

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

Line	Code
1	<?php
2	echo strpos("Hello world!", "world");
3	?>

This will produce the following result:

```
6
```

# Web Concepts

This session demonstrates how PHP can provide dynamic content according to browser type, randomly generated numbers or User Input. It also demonstrated how the client browser can be redirected.

## Identifying Browser & Platform

PHP creates some useful **environment variables** that can be seen in the phpinfo.php page that was used to setup the PHP environment.

One of the environment variables set by PHP is **HTTP\_USER\_AGENT** which identifies the user's browser and operating system.

PHP provides a function getenv() to access the value of all the environment variables. The information contained in the HTTP\_USER\_AGENT environment variable can be used to create dynamic content appropriate to the browser.

Following example demonstrates how you can identify a client browser and operating system.

**NOTE:** The reference of preg\_match() you can find in [Official Site](#).

Line	Code
1	<html>
2	<body>
3	<?php
4	function getBrowser() {
5	\$u_agent = \$_SERVER['HTTP_USER_AGENT'];
6	\$bname = 'Unknown';
7	\$platform = 'Unknown';
8	\$version = "";
9	
10	//First get the platform?
11	if (preg_match('/linux/i', \$u_agent)) {
12	\$platform = 'linux';
13	}elseif (preg_match('/macintosh mac os x/i', \$u_agent)) {
14	\$platform = 'mac';
15	}elseif (preg_match('/windows win32/i', \$u_agent)) {
16	\$platform = 'windows';
17	}
18	
19	// Next get the name of the useragent yes seperately and for
20	// good reason
21	if(preg_match('/MSIE/i',\$u_agent) &&
22	!preg_match('/Opera/i',\$u_agent)) {
23	\$bname = 'Internet Explorer';
24	\$ub = "MSIE";
25	}elseif (preg_match('/Firefox/i',\$u_agent)) {
26	\$bname = 'Mozilla Firefox';
27	\$ub = "Firefox";
28	}elseif (preg_match('/Chrome/i',\$u_agent)) {
29	\$bname = 'Google Chrome';
30	\$ub = "Chrome";

```

31 }elseif(preg_match('/Safari/i',$u_agent)) {
32     $bname = 'Apple Safari';
33     $sub = "Safari";
34 }elseif(preg_match('/Opera/i',$u_agent)) {
35     $bname = 'Opera';
36     $sub = "Opera";
37 }elseif(preg_match('/Netscape/i',$u_agent)) {
38     $bname = 'Netscape';
39     $sub = "Netscape";
40 }
41
42 // finally get the correct version number
43 $known = array('Version', $sub, 'other');
44 $pattern = '#(?<browser>' . join('|', $known) .
45             ')[/ ]+(?<version>[0-9.|a-zA-Z.]*)#';
46
47 if (!preg_match_all($pattern, $u_agent, $matches)) {
48     // we have no matching number just continue
49 }
50
51 // see how many we have
52 $i = count($matches['browser']);
53
54 if ($i != 1) {
55     //we will have two since we are not using 'other' argument yet
56
57     //see if version is before or after the name
58     if (stripos($u_agent,"Version") < stripos($u_agent,$sub)){
59         $version= $matches['version'][0];
60     }else {
61         $version= $matches['version'][1];
62     }
63 }else {
64     $version= $matches['version'][0];
65 }
66
67 // check if we have a number
68 if ($version == null || $version == "") {$version = "?";}
69 return array(
70     'userAgent' => $u_agent,
71     'name'      => $bname,
72     'version'   => $version,
73     'platform'  => $platform,
74     'pattern'   => $pattern
75 );
76 }
77
78 // now try it
79 $ua = getBrowser();
80 $yourbrowser = "Your browser: " . $ua['name'] . " " .
81                $ua['version'] . " on " . $ua['platform'] .
82                " reports: <br >" . $ua['userAgent'];
83
84 print_r($yourbrowser);
85 ?>
86 </body>
87 </html>

```

This is producing following result on my machine. This result may be different for your computer depending on what you are using.

It will produce the following result:

Your browser: Google Chrome 91.0.4472.101 on windows reports:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36  
(KHTML, like Gecko) Chrome/91.0.4472.101 Safari/537.36

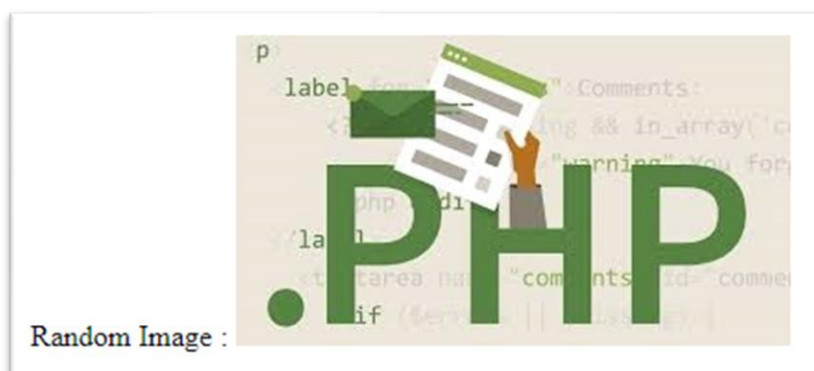
## Display Images Randomly

The PHP **rand()** function is used to generate a random number. This function can generate numbers with-in a given range. The random number generator should be seeded to prevent a regular pattern of numbers being generated. This is achieved using the **srand()** function that specifies the seed number as its argument.

Following example demonstrates how you can display different image each time out of four images:

Line	Code
1	<html>
2	<body>
3	<?php
4	\$num = rand( 1, 4 );
5	
6	switch( \$num ) {
7	case 1: \$image_file = "../images/logo.jpg";
8	break;
9	case 2: \$image_file = "../images/PHP.jpg";
10	break;
11	case 3: \$image_file = "../images/logo.jpg";
12	break;
13	case 4: \$image_file = "../images/php.jpg";
14	break;
15	}
16	echo "Random Image : <img src=\$image_file />";
17	?>
18	</body>
19	</html>

It will produce the following result:





## Using HTML Forms

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.

Try out following example by putting the source code in test.php script.

Line	Code
1	<?php
2	if( \$_POST["name"]    \$_POST["age"] ) {
3	if (preg_match("/[^\A-Za-z'-]/",\$_POST['name'])) {
4	die ("invalid name and name should be alpha");
5	}
6	
7	echo "Welcome ". \$_POST['name']. " ";
8	echo "You are ". \$_POST['age']. " years old.";
9	
10	exit();
11	}
12	?>
13	<html>
14	<body>
15	<form action = "<?php \$_PHP_SELF ?>" method = "POST">
16	Name: <input type = "text" name = "name" />
17	Age: <input type = "text" name = "age" />
18	<input type = "submit" />
19	</form>
20	</body>
21	</html>

It will produce the following result:

Name:  Age:

- The PHP default variable **\$\_PHP\_SELF** is used for the PHP script name and when you click "submit" button then same PHP script will be called and will produce following result:
- The method = "POST" is used to post user data to the server script. There are two methods of posting data to the server script which are discussed in [PHP GET & POST](#) chapter.

## Browser Redirection

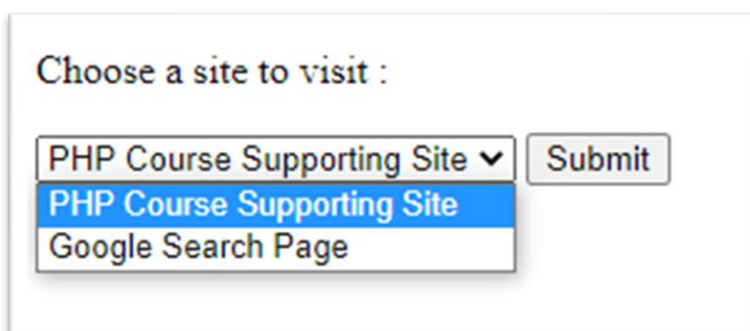
The PHP **header()** function supplies raw HTTP headers to the browser and can be used to redirect it to another location. The redirection script should be at the very top of the page to prevent any other part of the page from loading.

The target is specified by the **Location:** header as the argument to the **header()** function. After calling this function the **exit()** function can be used to halt parsing of rest of the code.

Following example demonstrates how you can redirect a browser request to another web page. Try out this example by putting the source code in test.php script.

Line	Code
1	<?php
2	if( \$_POST["location"] ) {
3	\$location = \$_POST["location"];
4	header( "Location:\$location" );
5	
6	exit();
7	}
8	?>
9	<html>
10	<body>
11	<p>Choose a site to visit :</p>
12	
13	<form action = "<?php \$_SERVER['PHP_SELF'] ?>" method = "POST">
14	<select name = "location">.
15	
16	<option value = "https://se001php.azurewebsites.net/">
17	PHP Course Supporting Site
18	</option>
19	
20	<option value = "http://www.google.com">
21	Google Search Page
22	</option>
23	</select>
24	<input type = "submit" />
25	</form>
26	</body>
27	</html>

It will produce the following result:



Choose a site to visit :

PHP Course Supporting Site ▼

Submit

PHP Course Supporting Site

Google Search Page

# GET & POST Methods

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

## The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY\_STRING environment variable.
- The PHP provides **\$\_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

Line	Code
1	<?php
2	if( \$_GET["name"]    \$_GET["age"] ) {
3	echo "Welcome ". \$_GET['name']. " ";
4	echo "You are ". \$_GET['age']. " years old.";
5	exit();
6	}
7	?>
8	<html>
9	<body>
10	<form action = "<?php \$_PHP_SELF ?>" method = "GET">
11	Name: <input type = "text" name = "name" />
12	Age: <input type = "text" name = "age" />
13	<input type = "submit" />
14	</form>
15	</body>
16	</html>

It will produce the following result:

Name:  Age:

## The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY\_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **\$\_POST** associative array to access all the sent information using POST method.

Try out following example by putting the source code in test.php script.

Line	Code
1	<?php
2	if( \$_POST["name"]    \$_POST["age"] ) {
3	if (preg_match("/[^\A-Za-z'-]/", \$_POST['name'] )) {
4	die ("invalid name and name should be alpha");
5	}
6	echo "Welcome ". \$_POST['name']. " ";
7	echo "You are ". \$_POST['age']. " years old.";
8	exit();
9	}
10	?>
11	<html>
12	<body>
13	<form action = "<?php \$_PHP_SELF ?>" method = "POST">
14	Name: <input type = "text" name = "name" />
15	Age: <input type = "text" name = "age" />
16	<input type = "submit" />
17	</form>
18	</body>
19	</html>

It will produce the following result:

Name:  Age:

## The \$\_REQUEST variable

The PHP \$\_REQUEST variable contains the contents of both \$\_GET, \$\_POST, and \$\_COOKIE. We will discuss \$\_COOKIE variable when we will explain about cookies.

The PHP \$\_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in test.php script.

Line	Code
1	<?php
2	if( \$_REQUEST["name"]    \$_REQUEST["age"] ) {
3	echo "Welcome ". \$_REQUEST['name']. " ";
4	echo "You are ". \$_REQUEST['age']. " years old.";
5	exit();
6	}
7	?>
8	<html>
9	<body>
10	<form action = "<?php \$_PHP_SELF ?>" method = "POST">
11	Name: <input type = "text" name = "name" />
12	Age: <input type = "text" name = "age" />
13	<input type = "submit" />
14	</form>
15	</body>
16	</html>

Here \$\_PHP\_SELF variable contains the name of self script in which it is being called.

It will produce the following result:

Name:

Age:

## File Inclusion

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to include one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.

### The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="https://se001php.azurewebsites.net/">Home</a> -  
<a href="https://se001php.azurewebsites.net/Reference">Reference</a> -  
<a href="https://se001php.azurewebsites.net/Home/About">About</a> -  
<a href="https://se001php.azurewebsites.net/Home/Contact">Contact</a> <br />
```

Now create as many pages as you like and include this file to create header. For example, now your test.php file can have following content.

Line	Code
1	<html>
2	<body>
3	<?php include("menu.php"); ?>
4	<p>This is an example to show how to include PHP file!</p>
5	</body>
6	</html>

It will produce the following result:

[Home](#) - [Reference](#) - [About](#) - [Contact](#)

This is an example to show how to include PHP file!

## The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.

So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

Line	Code
1	<html>
2	<body>
3	<?php include("xxmenu.php"); ?>
4	<p>This is an example to show how to include wrong PHP file!</p>
5	</body>
6	</html>

This will produce the following result:

This is an example to show how to include wrong PHP file!

Now lets try same example with require() function.

Line	Code
1	<html>
2	<body>
3	<?php require("xxmenu.php"); ?>
4	<p>This is an example to show how to include wrong PHP file!</p>
5	</body>
6	</html>

This time file execution halts and nothing is displayed.

**NOTE:** You may get plain warning messages or fatal error messages or nothing at all. This depends on your PHP Server configuration.

# Files & I/O

This chapter will explain following functions related to files:

- Opening a file
- Reading a file
- Writing a file
- Closing a 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.

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

No.	Mode	Purpose
1	<b>r</b>	Opens the file for reading only. Places the file pointer at the beginning of the file.
2	<b>r+</b>	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
3	<b>w</b>	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
4	<b>w+</b>	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
5	<b>a</b>	Opens the file for writing only. Places the file pointer at the end of the file. If files do not exist then it attempts to create a file.
6	<b>a+</b>	Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

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.



## 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.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

Line	Code
1	<html>
2	<head>
3	<title>Reading a file using PHP</title>
4	</head>
5	<body>
6	<?php
7	\$filename = "tmp.txt";
8	\$file = fopen( \$filename, "r" );
9	
10	if( \$file == false ) {
11	echo ( "Error in opening file" );
12	exit();
13	}
14	
15	\$filesize = filesize( \$filename );
16	\$filetext = fread( \$file, \$filesize );
17	fclose( \$file );
18	
19	echo ( "File size : \$filesize bytes" );
20	echo ( "<pre>\$filetext</pre>" );
21	?>
22	</body>
23	</html>

It will produce the following result:

File size : 273 bytes

```
The PHP Hypertext Preprocessor (PHP) is a programming language
that allows web developers to create dynamic content that interacts with databases.
PHP is basically used for developing web-based software applications.
This course helps you to build your base with PHP.
```

## 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 stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file\_exists()** function which takes file name as an argument

Line	Code
1	<?php
2	\$filename = "/home/user/guest/newfile.txt";
3	\$file = fopen( \$filename, "w" );
4	
5	if( \$file == false ) {
6	echo ( "Error in opening new file" );
7	exit();
8	}
9	fwrite( \$file, "This is a simple test\n" );
10	fclose( \$file );
11	?>
12	<html>
13	<head>
14	<title>Writing a file using PHP</title>
15	</head>
16	<body>
17	<?php
18	\$filename = "newfile.txt";
19	\$file = fopen( \$filename, "r" );
20	
21	if( \$file == false ) {
22	echo ( "Error in opening file" );
23	exit();
24	}
25	
26	\$filesize = filesize( \$filename );
27	\$filetext = fread( \$file, \$filesize );
28	
29	fclose( \$file );
30	
31	echo ( "File size : \$filesize bytes" );
32	echo ( "\$filetext" );
33	echo("file name: \$filename");
34	?>
35	</body>
36	</html>

It will produce the following result:

```
File size : 23 bytes
This is a simple test
file name: newfile.txt
```

# Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you:

- Creating a PHP Function
- Calling a PHP Function

In fact, you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Please refer to [Official Function Reference](#) for a complete set of useful functions.

## Creating PHP Function

It's very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called `writeMessage()` and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

Line	Code
1	<html>
2	<head>
3	<title>Writing PHP <b>Function</b> </title>
4	</head>
5	<body>
6	<?php
7	/* Defining a PHP Function */
8	<b>function</b> writeMessage() {
9	<b>echo</b> "You are really a nice person, Have a nice time!";
10	}
11	
12	/* Calling a PHP Function */
13	writeMessage();
14	?>
15	</body>
16	</html>

This will display following result:

You are really a nice person, Have a nice time!

## PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

Line	Code
1	<html>
2	<head>
3	<title>Writing PHP Function with Parameters</title>
4	</head>
5	<body>
6	<?php
7	function addFunction(\$num1, \$num2) {
8	\$sum = \$num1 + \$num2;
9	echo "Sum of the two numbers is : \$sum";
10	}
11	
12	addFunction(10, 20);
13	?>
14	</body>
15	</html>

This will display following result:

```
Sum of the two numbers is : 30
```

## 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.

Line	Code
1	<html>
2	<head>
3	<title>Passing Argument by Reference</title>
4	</head>
5	<body>
6	<?php
7	function addFive(\$num) {
8	\$num += 5;
9	}
10	
11	function addSix(&\$num) {
12	\$num += 6;
13	}
14	
15	\$orignum = 10;
16	addFive( \$orignum );

17	
18	<code>echo "Original Value is \$orignum&lt;br /&gt;";</code>
19	
20	<code>addSix( \$orignum );</code>
21	<code>echo "Original Value is \$orignum&lt;br /&gt;";</code>
22	<code>?&gt;</code>
23	<code>&lt;/body&gt;</code>
24	<code>&lt;/html&gt;</code>

This will display following result:

```
Original Value is 10
Original Value is 16
```

## PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. **return** stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

Line	Code
1	<code>&lt;html&gt;</code>
2	<code>&lt;head&gt;</code>
3	<code>&lt;title&gt;Writing PHP <b>Function</b> which returns value&lt;/title&gt;</code>
4	<code>&lt;/head&gt;</code>
5	<code>&lt;body&gt;</code>
6	<code>&lt;?php</code>
7	<code>function addFunction(\$num1, \$num2) {</code>
8	<code>    \$sum = \$num1 + \$num2;</code>
9	<code>    return \$sum;</code>
10	<code>}</code>
11	<code>\$return_value = addFunction(10, 20);</code>
12	
13	<code>echo "Returned value from the function : \$return_value";</code>
14	<code>?&gt;</code>
15	<code>&lt;/body&gt;</code>
16	<code>&lt;/html&gt;</code>

This will display following result:

```
Returned value from the function : 30
```

## Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case use does not pass any value to this function.

Line	Code
1	<html>
2	<head>
3	<title>Writing PHP <b>Function</b> which returns value</title>
4	</head>
5	<body>
6	<?php
7	function printMe(\$param = NULL) {
8	print \$param;
9	}
10	
11	printMe("This is test");
12	printMe();
13	?>
14	</body>
15	</html>

This will produce following result:

This is test

## Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

Line	Code
1	<html>
2	<head>
3	<title>Dynamic <b>Function</b> Calls</title>
4	</head>
5	<body>
6	<?php
7	function sayHello() {
8	echo "Hello ";
9	}
10	
11	\$function_holder = "sayHello";
12	\$function_holder();
13	?>
14	</body>
15	</html>

This will display following result:

Hello

# Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

## The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
           path=/; domain=se001php.azurewebsites.net
Connection: close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

A PHP script will then have access to the cookie in the environmental variables `$_COOKIE` or `$HTTP_COOKIE_VARS[]` which holds all cookie names and values. Above cookie can be accessed using `$HTTP_COOKIE_VARS["name"]`.

# Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments:

- **Name:** This sets the name of the cookie and is stored in an environment variable called HTTP\_COOKIE\_VARS. This variable is used while accessing cookies.
- **Value:** This sets the value of the named variable and is the content that you actually want to store.
- **Expiry:** This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path:** This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain:** This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security:** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

Line	Code
1	<?php
2	setcookie("name", "John Watkin", time()+3600, "/", "", 0);
3	setcookie("age", "36", time()+3600, "/", "", 0);
4	?>
5	<html>
6	<head>
7	<title>Setting Cookies with PHP</title>
8	</head>
9	<body>
10	<?php echo "Set Cookies"?>
11	</body>
12	</html>



## Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either `$_COOKIE` or `$HTTP_COOKIE_VARS` variables. Following example will access all the cookies set in above example.

Line	Code
1	<html>
2	<head>
3	<title>Accessing Cookies with PHP</title>
4	</head>
5	<body>
6	<?php
7	echo \$_COOKIE["name"]. " ";
8	
9	/* is equivalent to */
10	echo \$HTTP_COOKIE_VARS["name"]. " ";
11	
12	echo \$_COOKIE["age"] . " ";
13	
14	/* is equivalent to */
15	echo \$HTTP_COOKIE_VARS["age"] . " ";
16	?>
17	</body>
18	</html>

You can use **isset()** function to check if a cookie is set or not.

Line	Code
1	<html>
2	<head>
3	<title>Accessing Cookies with PHP</title>
4	</head>
5	<body>
6	<?php
7	if( isset(\$_COOKIE["name"]))
8	echo "Welcome " . \$_COOKIE["name"] . " ";
9	
10	else
11	echo "Sorry... Not recognized" . " ";
12	?>
13	</body>
14	</html>

## Deleting Cookie with PHP

Officially, to delete a cookie you should call `setcookie()` with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired:

Line	Code
1	<code>&lt;?php</code>
2	<code>    setcookie( "name", "", time()- 60, "/", "", 0 );</code>
3	<code>    setcookie( "age", "", time()- 60, "/", "", 0 );</code>
4	<code>?&gt;</code>
5	<code>&lt;html&gt;</code>
6	<code>&lt;head&gt;</code>
7	<code>    &lt;title&gt;Deleting Cookies with PHP&lt;/title&gt;</code>
8	<code>&lt;/head&gt;</code>
9	<code>&lt;body&gt;</code>
10	<code>    &lt;?php echo "Deleted Cookies" ?&gt;</code>
11	<code>&lt;/body&gt;</code>
12	<code>&lt;/html&gt;</code>

# Sessions

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save\_path**. Before using any session variable make sure you have setup this path.

When a session is started following things happen:

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess\_ ie sess\_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

## Starting a PHP Session

A PHP session is easily started by making a call to the **session\_start()** function. This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session\_start()** at the beginning of the page.

Session variables are stored in associative array called **\$\_SESSION[]**. These variables can be accessed during lifetime of a session.

The following example starts a session then register a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset()** function to check if session variable is already set or not.

Put this code in a test.php file and load this file many times to see the result:

Line	Code
1	<?php
2	session_start();
3	
4	if( isset( \$_SESSION['counter'] ) ) {
5	\$_SESSION['counter'] += 1;
6	}else {
7	\$_SESSION['counter'] = 1;
8	}
9	
10	\$msg = "You have visited this page ". \$_SESSION['counter'];
11	\$msg .= "in this session.";
12	?>
13	<html>
14	<head>
15	<title>Setting up a PHP session</title>
16	</head>
17	<body>
18	<?php echo ( \$msg ); ?>
19	</body>
20	</html>

It will produce the following result:

You have visited this page 1in this session.

## Destroying a PHP Session

A PHP session can be destroyed by **session\_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable:

```
<?php
unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables:

```
<?php
session_destroy();
?>
```

## Turning on Auto Session

You don't need to call `start_session()` function to start a session when a user visits your site if you can set **session.auto\_start** variable to 1 in **php.ini** file.

## Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session\_name=session\_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

Line	Code
1	<?php
2	session_start();
3	
4	if (isset(\$_SESSION['counter'])) {
5	\$_SESSION['counter'] = 1;
6	}else {
7	\$_SESSION['counter']++;
8	}
9	
10	\$msg = "You have visited this page ". \$_SESSION['counter'];
11	\$msg .= "in this session.";
12	
13	echo ( \$msg );
14	?>
15	<p>
16	To continue click following link  
17	
18	<a href = "nextpage.php<?php echo htmlspecialchars(SID); ?>">
19	</p>

It will produce the following result:

```
You have visited this page 1in this session.  
To continue click following link
```

The **htmlspecialchars()** may be used when printing the SID in order to prevent XSS related attacks.

# Sending Emails using PHP

PHP must be configured correctly in the **php.ini** file with the details of how your system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called sendmail\_from which defines your own email address.

The configuration for Windows should look something like this:

```
[mail function]
; For Win32 only.
SMTP = smtp.secureserver.net

; For win32 only
sendmail_from = staff01@ckleng.onmicrosoft.com
```

Linux users simply need to let PHP know the location of their **sendmail** application. The path and any desired switches should be specified to the sendmail\_path directive.

The configuration for Linux should look something like this:

```
[mail function]
; For Win32 only.
SMTP =

; For win32 only
sendmail_from =

; For Unix only
sendmail_path = /usr/sbin/sendmail -t -i
```

Now you are ready to go:

## Sending plain text email

PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the message and the actual message additionally there are other two optional parameters.

```
mail( to, subject, message, headers, parameters );
```

Here is the description for each parameters.

No.	Parameter	Description
1	<b>to</b>	Required. Specifies the receiver / receivers of the email
2	<b>subject</b>	Required. Specifies the subject of the email. This parameter cannot contain any newline characters
3	<b>message</b>	Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters
4	<b>headers</b>	Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n)
5	<b>parameters</b>	Optional. Specifies an additional parameter to the send mail program

As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed.

Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

## Sending HTML email

When you send a text message using PHP then all the content will be treated as simple text. Even if you will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.

While sending an email message you can specify a Mime version, content type and character set to send an HTML email.

### Example

Following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

Line	Code
1	<html>
2	<head>
3	<title>Sending HTML email using PHP</title>
4	</head>
5	<body>
6	<?php
7	\$to = "xyz@somedomain.com";
8	\$subject = "This is subject";
9	
10	\$message = "<b>This is HTML message.</b>";
11	\$message .= "<h1>This is headline.</h1>";
12	
13	\$header = "From:abc@somedomain.com \r\n";
14	\$header .= "Cc:afgh@somedomain.com \r\n";
15	\$header .= "MIME-Version: 1.0\r\n";
16	\$header .= "Content-type: text/html\r\n";
17	
18	\$retval = mail (\$to,\$subject,\$message,\$header);
19	
20	if( \$retval == true ) {
21	echo "Message sent successfully...";
22	}else {
23	echo "Message could not be sent...";
24	}
25	?>
26	</body>
27	</html>



## Sending attachments with email

To send an email with mixed content requires to set **Content-type** header to **multipart/mixed**. Then text and attachment sections can be specified within **boundaries**.

A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function **md5()** is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

Line	Code
1	<?php
2	// request variables // important
3	\$from = \$_REQUEST["from"];
4	\$emaila = \$_REQUEST["emaila"];
5	\$filea = \$_REQUEST["filea"];
6	
7	if (\$filea) {
8	function mail_attachment (
9	\$from , \$to, \$subject, \$message, \$attachment){
10	\$fileatt = \$attachment; // Path to the file
11	\$fileatt_type = "application/octet-stream"; // File Type
12	
13	\$start = strpos(\$attachment, '/') == -1 ?
14	strpos(\$attachment, '//') : strpos(\$attachment, '/')+1;
15	
16	\$fileatt_name = substr(\$attachment, \$start,
17	strlen(\$attachment)); // Filename that will be used for the
18	file as the attachment
19	
20	\$email_from = \$from; // Who the email is from
21	\$subject = "New Attachment Message";
22	
23	\$email_subject = \$subject; // The Subject of the email
24	\$email_txt = \$message; // Message that the email has in it
25	\$email_to = \$to; // Who the email is to
26	
27	\$headers = "From: ".\$email_from;
28	\$file = fopen(\$fileatt,'rb');
29	\$data = fread(\$file,filesize(\$fileatt));
30	fclose(\$file);
31	
32	\$msg_txt=
33	"\n\n You have recieved a new attachment message from \$from";
34	\$semi_rand = md5(time());
35	\$mime_boundary = "==Multipart_Boundary_x{\$semi_rand}x";
36	\$headers .= "\nMIME-Version: 1.0\n" .
37	"Content-Type: multipart/mixed;\n" .
38	"boundary=\"{\$mime_boundary}\"";
39	
40	\$email_txt .= \$msg_txt;
41	
42	\$email_message .=
43	"This is a multi-part message in MIME format.\n\n" .

```

44     "--{$mime_boundary}\n" .
45     "Content-Type:text/html; charset = \"iso-8859-1\"\n" .
46     "Content-Transfer-Encoding: 7bit\n\n" .
47     $email_txt . "\n\n";
48
49     $data = chunk_split(base64_encode($data));
50
51     $email_message .= "--{$mime_boundary}\n" .
52         "Content-Type: {$fileatt_type};\n" .
53         " name = \"{$fileatt_name}\"\n" .
54         //"Content-Disposition: attachment;\n" .
55         //" filename = \"{$fileatt_name}\"\n" .
56         //"Content-Transfer-Encoding:
57         base64\n\n" . $data . "\n\n" . "--{$mime_boundary}--\n";
58
59     $ok = mail($email_to, $email_subject, $email_message, $headers);
60
61     if($ok) {
62         echo "File Sent Successfully.";
63         unlink($attachment); // delete a file after attachment sent.
64     }else {
65         die("Sorry but the email could not be sent. Please go back and
try again!");
66     }
67 }
68 move_uploaded_file($_FILES["filea"]["tmp_name"],
69     'temp/'.basename($_FILES['filea']['name']));
70
71 mail_attachment("$from", "youreemailaddress@gmail.com",
72     "subject", "message", ("temp/".$_FILES["filea"]["name"]));
73 }
74 ?>
75 <html>
76 <head>
77     <script language = "javascript" type = "text/javascript">
78         function CheckData45() {
79             with(document.filepost) {
80                 if(filea.value != "") {
81                     document.getElementById('one').innerText =
82                         "Attaching File ... Please Wait";
83                 }
84             }
85         }
86     </script>
87 </head>
88 <body>
89     <table width = "100%" height = "100%" border = "0"
90         cellpadding = "0" cellspacing = "0">
91         <tr>
92             <td align = "center">
93                 <form name = "filepost" method = "post" action = "file.php"
94                     enctype = "multipart/form-data" id = "file">
95
96                     <table width = "300" border = "0" cellspacing = "0"
97                         cellpadding = "0">
98                         <tr valign = "bottom">
99                             <td height = "20">Your Name:</td>

```

```

100         </tr>
101
102         <tr>
103             <td><input name = "from" type = "text"
104                 id = "from" size = "30">
105             </td>
106         </tr>
107
108         <tr valign = "bottom">
109             <td height = "20">Your Email Address:</td>
110         </tr>
111
112         <tr>
113             <td class = "frmtxt2"><input name = "emaila"
114                 type = "text" id = "emaila" size = "30"></td>
115         </tr>
116
117         <tr>
118             <td height = "20" valign = "bottom">Attach File:</td>
119         </tr>
120
121         <tr valign = "bottom">
122             <td valign = "bottom"><input name = "filea"
123                 type = "file" id = "filea" size = "16">
124             </td>
125         </tr>
126
127         <tr>
128             <td height = "40" valign = "middle">
129                 <input name = "Reset2" type = "reset" id = "Reset2"
130                     value = "Reset">
131                 <input name = "Submit2" type = "submit"
132                     value = "Submit"
133                     onClick = "return CheckData45()">
134             </td>
135         </tr>
136     </table>
137 </form>
138 <center>
139     <table width = "400">
140         <tr>
141             <td id = "one"></td>
142         </tr>
143     </table>
144 </center>
145 </td>
146 </tr>
147 </table>
148 </body>
149 </html>

```

# File Uploading

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload\_tmp\_dir** and the maximum permitted size of files that can be uploaded is stated as **upload\_max\_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps:

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text filed then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual when writing files, it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail.

An uploaded file could be a text file or image file or any document.

## Creating an upload form

The following HTM code below creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

Line	Code
1	<?php
2	if(isset(\$_FILES['image'])) {
3	\$errors= array();
4	\$file_name = \$_FILES['image']['name'];
5	\$file_size = \$_FILES['image']['size'];
6	\$file_tmp = \$_FILES['image']['tmp_name'];
7	\$file_type=\$_FILES['image']['type'];
8	\$file_ext=strtolower(end(explode('.', \$_FILES['image']['name'])));
9	
10	\$extensions= array("jpeg", "jpg", "png");
11	
12	if(in_array(\$file_ext,\$extensions)=== false) {
13	\$errors[]=
14	"extension not allowed, please choose a JPEG or PNG file.";
15	}
16	
17	if(\$file_size > 2097152) {

```

18     $errors[]='File size must be excately 2 MB';
19 }
20
21 if(empty($errors)==true) {
22     move_uploaded_file($file_tmp,"images/".$file_name);
23     echo "Success";
24 }else{
25     print_r($errors);
26 }
27 }
28 ?>
29 <html>
30 <body>
31     <form action="" method="POST" enctype="multipart/form-data">
32         <input type="file" name="image" />
33         <input type="submit"/>
34     </form>
35 </body>
36 </html>

```

It will produce the following result:



## Creating an upload script

There is one global PHP variable called **\$\_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables:

- **\$\_FILES['file']['tmp\_name']**: the uploaded file in the temporary directory on the web server.
- **\$\_FILES['file']['name']**: the actual name of the uploaded file.
- **\$\_FILES['file']['size']**: the size in bytes of the uploaded file.
- **\$\_FILES['file']['type']**: the MIME type of the uploaded file.
- **\$\_FILES['file']['error']**: the error code associated with this file upload.

## Example

Below example should allow upload images and gives back result as uploaded file information.

Line	Code
1	<?php
2	if(isset(\$_FILES['image'])){
3	\$errors= array();
4	\$file_name = \$_FILES['image']['name'];
5	\$file_size = \$_FILES['image']['size'];
6	\$file_tmp = \$_FILES['image']['tmp_name'];
7	\$file_type = \$_FILES['image']['type'];
8	\$file_ext=strtolower(end(explode('.',\$_FILES['image']['name'])));
9	
10	\$extensions= array("jpeg","jpg","png");
11	
12	if(in_array(\$file_ext,\$extensions)=== false){
13	\$errors[]=
14	"extension not allowed, please choose a JPEG or PNG file.";
15	}
16	if(\$file_size > 2097152) {
17	\$errors[]='File size must be excately 2 MB';
18	}
19	if(empty(\$errors)==true) {
20	move_uploaded_file(\$file_tmp,"images/".\$file_name);
21	echo "Success";
22	}else{
23	print_r(\$errors);
24	}
25	}
26	?>
27	<html>
28	<body>
29	<form action = "" method = "POST" enctype = "multipart/form-data">
30	<input type = "file" name = "image" />
31	<input type = "submit"/>
32	
33	<ul>
34	<li>Sent file: <?php echo \$_FILES['image']['name']; ?>
35	<li>File size: <?php echo \$_FILES['image']['size']; ?>
36	<li>File type: <?php echo \$_FILES['image']['type'] ?>
37	</ul>
38	
39	</form>
40	</body>
41	</html>

It will produce the following result:

No file chosen

- Sent file:
- File size:
- File type:

# Coding Standard

Every company follows a different coding standard based on their best practices. Coding standard is required because there may be many developers working on different modules so if they will start inventing their own standards then source will become very un-manageable and it will become difficult to maintain that source code in future.

Here are several reasons why to use coding specifications:

- Your peer programmers have to understand the code you produce. A coding standard acts as the blueprint for all the team to decipher the code.
- Simplicity and clarity achieved by consistent coding saves you from common mistakes.
- If you revise your code after some time then it becomes easy to understand that code.
- Its industry standard to follow a particular standard to being more quality in software.

There are few guidelines which can be followed while coding in PHP.

- **Indenting and Line Length:** Use an indent of 4 spaces and don't use any tab because different computers use different setting for tab. It is recommended to keep lines at approximately 75-85 characters long for better code readability.
- **Control Structures:** These include if, for, while, switch, etc. Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls. You are strongly encouraged to always use curly braces even in situations where they are technically optional.

## Examples

```
if ((condition1) || (condition2)) {  
    action1;  
}elseif ((condition3) && (condition4)) {  
    action2;  
}else {  
    default action;  
}
```

You can write switch statements as follows:

```
switch (condition) {  
    case 1:  
        action1;  
        break;  
    case 2:  
        action2;  
        break;  
    default:  
        default action;  
        break;  
}
```

- **Function Calls:** Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas

and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
$var = foo($bar, $baz, $quux);
```

- **Function Definitions:** Function declarations follow the "BSD/Allman style":

```
function fooFunction($arg1, $arg2 = '') {  
    if (condition) {  
        statement;  
    }  
    return $val;  
}
```

- **Comments:** C style comments (`/* */`) and standard C++ comments (`//`) are both fine. Use of Perl/shell style comments (`#`) is discouraged.
- **PHP Code Tags:** Always use `<?php ?>` to delimit PHP code, not the `<? ?>` shorthand. This is required for PHP compliance and is also the most portable way to include PHP code on differing operating systems and setups.
- **Variable Names:**
  - Use all lower case letters
  - Use `'_'` as the word separator.
  - Global variables should be prepended with a `'g'`.
  - Global constants should be all caps with `'_'` separators.
  - Static variables may be prepended with `'s'`.
- **Make Functions Reentrant:** Functions should not keep static variables that prevent a function from being reentrant.
- **Alignment of Declaration Blocks:** Block of declarations should be aligned.
- **One Statement Per Line:** There should be only one statement per line unless the statements are very closely related.
- **Short Methods or Functions:** Methods should limit themselves to a single page of code.

There could be many more points which should be considered while writing your PHP program. Over all intention should be to be consistent throughout of the code programming and it will be possible only when you will follow any coding standard. You can device your own standard if you like something different.