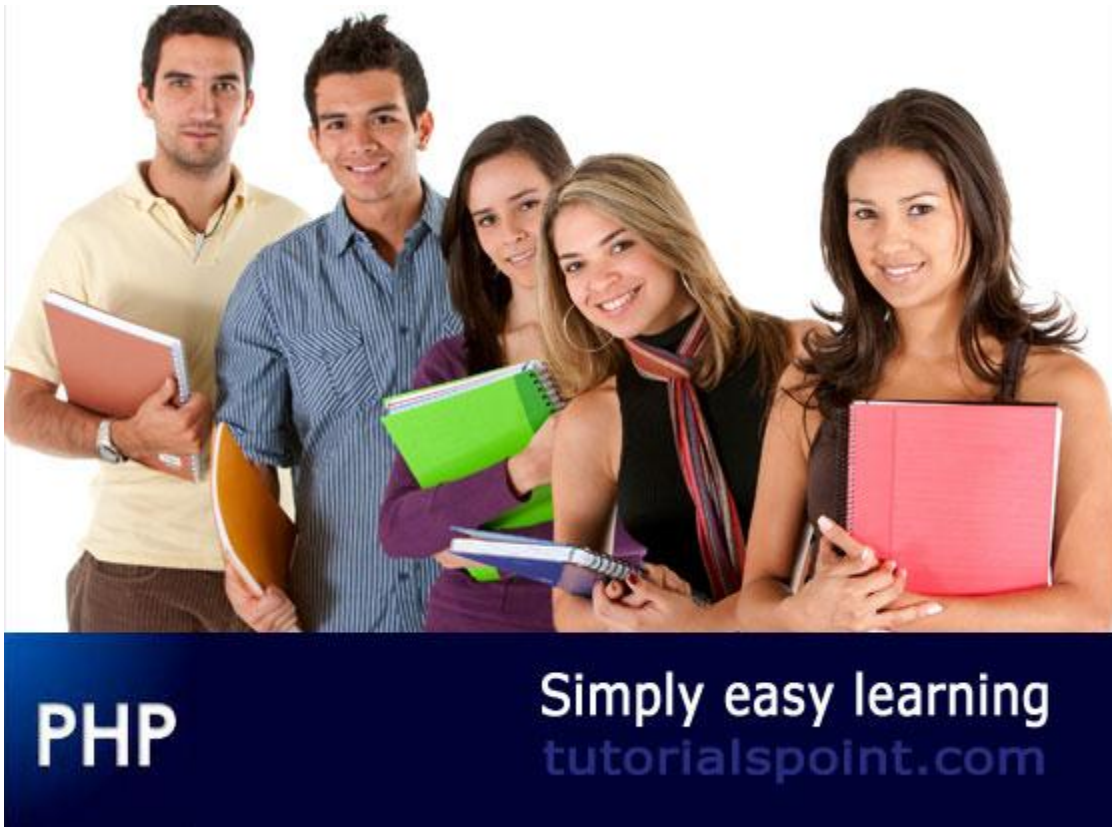
PHP Tutorial

Tutorialspoint.com



The PHP Hypertext Pre-processor (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 tutorial helps you to build your base with PHP.

Send your feedback using Contact Us Form

Audience:

This tutorial 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 tutorial you should have at least basic understanding of computer programming, Internet, Database, and MySQL etc is very helpful.

PHP Introduction

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

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire 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, thru email you can send data, return data to the user.
- You add, delete, modify elements within your database thru 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

"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:

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

Most common tag is the <?php...?> and we will also use same tag in our tutorial.

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.

PHP 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 availble Apache Server. Download Apache for free here: http://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: http://www.mysql.com/downloads/index.html
- **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 tutorial will guide you how to install PHP parser on your computer.

PHP Parser Installation:

Before you proceed it is important to make sure that you have proper environment setup on your machine to develop your web programs using PHP.

Type the following address into your browser's address box.

```
http://127.0.0.1/info.php
```

If this displays a page showing your PHP installation related information then it means you have PHP and Webserver installed properly. Otherwise you have to follow given procedure to install PHP on your computer.

This section will guide you to install and configure PHP over the following four platforms:

- PHP Installation on Linux or Unix with Apache
- PHP Installation on Mac OS X with Apache
- PHP Installation on Windows NT/2000/XP with IIS
- PHP Installation on Windows NT/2000/XP with Apache

Apache Configuration:

If you are using Apache as a Web Server then this section will guide you to edit Apache Configuration Files.

Just Check it here : PHP Configuration in Apache Server

PHP.INI File Configuration:

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality.

Just Check it here: PHP.INI File Configuration

Windows IIS Configuration:

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

PHP 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.

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

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

```
<?
# First Example
print <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
# Second Example
print "This spans
multiple lines. The newlines will be
output as well";
?>
```

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.

```
<?
/* This is a comment with multiline
    Author : Mohammad Mohtashim
    Purpose: Multiline Comments Demo
    Subject: PHP
*/
print "An example with multi line comments";
```

```
?>
```

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:

```
$four = 2 + 2; // single spaces
$four <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs
$four =
2+
2; // multiple lines
```

PHP is case sensitive:

Yeah it is true that PHP is a case sensitive language. Try out following example:

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

This will produce 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:

```
if (3 == 2 + 1)
  print("Good - I haven't totally lost my mind.<br>");

if (3 == 2 + 1)
{
   print("Good - I haven't totally");
   print("lost my mind.<br>");
}
```

Running PHP Script from Command Prompt:

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

```
<?php
   echo "Hello PHP!!!!!";
?>
```

Now run this script as command prompt as follows:

```
$ php test.php
```

It will produce following result:

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

Hope now you have basic knowledge of PHP Syntax.

PHP 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 simile data type in this chapters. 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:

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

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 hexadecimals 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:

```
$many = 2.2888800;
$many_2 = 2.2111200;
$few = $many + $many_2;
print(.$many + $many_2 = $few<br>.);
```

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:

```
if (TRUE)
   print("This will always print<br>");
else
```

```
print("This will never print<br>");
```

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.

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

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

```
$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.

```
<?
$variable = "name";
$literally = 'My $variable will not print!\\n';
print($literally);
$literally = "My $variable will print!\\n";
print($literally);
?>
```

This will produce following result:

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

```
<?php

$channel =<<<_XML_
<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
</channel>
_XML_;

echo <<<END
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
<br />
```

```
END;

print $channel;
?>
```

This will produce following result:

```
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!

<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
```

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.

PHP Constants

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:

```
<?php

define("MINSIZE", 50);

echo MINSIZE;
echo constant("MINSIZE"); // same thing as the previous line

?>
```

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:

```
// Valid constant names
define("ONE",     "first thing");
define("TWO2",    "second thing");
define("THREE_3", "third thing")
// Invalid constant names
define("2TWO",    "second thing");
define("__THREE__", "third value");
```

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 ate given below:

Name	Description
__LINE__	The current line number of the file.
__FILE__	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, **__FILE__** always contains an absolute path whereas in older versions it contained relative path under some circumstances.
__FUNCTION__	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.
__CLASS__	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.
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

PHP 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
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Arithmatic Operators:

There are following arithmatic operators supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Show Examples

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 denumerator	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:

Show Examples

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:

Show Examples

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then 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:

Show Examples

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

	to left operand	
/=	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:

Show Examples

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 precedenace 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	= += -= *= /=	

PHP 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 threedecision 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 several condition are 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!":

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

</html>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>

<body>
<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>
</body>
</html>
```

The Elself Statement

If you want to execute some code if one of 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!":

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

The Switch Statement

If you want to select one of many blocks of code to be executed, 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 lable 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 lables match then statement will execute any specified default code.

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

```
</html>
```

PHP 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 trur.
- **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:

```
<html>
<body>
<?php
$a = 0;
$b = 0;

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

This will produce 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.

```
<html>
<body>
<?php
$i = 0;
$num = 50;

while( $i < 10)
{
   $num--;
   $i++;
}
echo ("Loop stopped at i = $i and num = $num" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 1 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:

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

This will produce 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.

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

This will produce 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. If 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.

```
<html>
<body>

<?php
$i = 0;

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

This will produce 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 skip the code and next value is printed.

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

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

This will produce following result

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

PHP 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 its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c 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 stores 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 prepresented 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.

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

```
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";

foreach( $numbers as $value )
{
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce 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 employees 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 otheriwse it would not return any value.

Example

```
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
                    "mohammad" => 2000,
                    "qadir" => 1000,
                    "zara" => 500
                   );

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
echo "Salary of zara is ".  $salaries['zara']. "<br />";

/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
```

```
echo "Salary of zara is ".  $salaries['zara']. "<br />";
?>
</body>
</html>
```

This will produce 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 index.

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.

```
<html>
<body>
<?php
   $marks = array(
                "mohammad" => array
                (
                "physics" => 35,
                "maths" => 30,
                "chemistry" => 39
                ),
                "qadir" => array
               (
               "physics" => 30,
               "maths" => 32,
               "chemistry" => 29
               ),
               "zara" => array
               (
               "physics" => 31,
               "maths" => 22,
               "chemistry" => 39
               )
            );
   /* Accessing multi-dimensional array values */
   echo "Marks for mohammad in physics : " ;
   echo $marks['mohammad']['physics'] . "<br />";
   echo "Marks for qadir in maths : ";
   echo $marks['qadir']['maths'] . "<br />";
   echo "Marks for zara in chemistry : " ;
   echo $marks['zara']['chemistry'] . "<br />";
?>
</body>
</html>
```

This will produce following result:

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

PHP 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.

```
<?
$variable = "name";
$literally = 'My $variable will not print!\\n';
print($literally);
$literally = "My $variable will print!\\n";
print($literally);
?>
```

This will produce following result:

```
My $variable will not print!\n
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 (\)

String Concatenation Operator

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

```
<?php
$string1="Hello World";
$string2="1234";
echo $string1 . " " . $string2;
?>
```

This will produce 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!":

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

This will produce 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:

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

This will produce following result:

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

PHP 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 included 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="http://www.tutorialspoint.com/index.htm">Home</a> -
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -
<a href="http://www.tutorialspoint.com/perl">PERL</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.

```
<html>
<body>
<?php include("menu.php"); ?>
<p>This is an example to show how to include PHP file!</p>
</body>
</html>
```

This will produce following result

Home - ebXML - AJAX - PERL

This is an example to show how to include PHP file. You can include mean.php file in as many as files you like!

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 excution 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.

```
<html>
<body>
<?php include("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This will produce following result

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

Now lets try same example with require() function.

```
<html>
<body>
<?php require("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</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.

PHP 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.

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

Mode	Purpose

r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	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 attemts to create a file.
w+	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 attemts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attemts to create a file.
a+	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 attemts 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's 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.

```
<html>
```

```
<head>
<title>Reading a file using PHP</title>
</head>
<body>

<?php
$filename = "/home/user/guest/tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
   echo ( "Error in opening file" );
   exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );

fclose( $file );

echo ( "File size : $filesize bytes" );
echo ( "<pre>$text</pre>" );
?>

</body>
</html>
```

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 intger argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

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

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
   echo ( "Error in opening new file" );
   exit();
}
fwrite( $file, "This is  a simple test\n" );
fclose( $file );
?>

<html>
<head>
<title>Writing a file using PHP</title>
</head>
<body>

<?php
if( file_exist( $filename ) )
{
   $filesize = filesize( $filename );
   $msg = "File  created with name $filename ";
   $msg .= "containing $filesize bytes";
   echo ($msg );
}
```

```
else
{
   echo ("File $filename does not exit" );
}
?>
</body>
</html>
```

We have covered all the function related to file input and out in PHP File System Function chapter.

PHP 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 PHP Function Reference for a complete set of useful functions.

Creating PHP Function:

Its 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:

```
<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>

<?php
/* Defining a PHP Function */
function writeMessage()
{
  echo "You are really a nice person, Have a nice time!";
}
/* Calling a PHP Function */
writeMessage();
?>
</body>
</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.

```
<html>
<head>
<title>Writing PHP Function with Parameters</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  echo "Sum of the two numbers is : $sum";
}
addFunction(10, 20);
?>
</body>
</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.

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function addFive($num)
{
   $num += 5;
}

function addSix(&$num)
{
   $num += 6;
}
$orignum = 10;
addFive( &$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );
```

```
echo "Original Value is $orignum<br />";
?>
</body>
</html>
```

This will display following result:

```
Original Value is 15
Original Value is 21
```

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.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  return $sum;
}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value
?>
</body>
</html>
```

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.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function printMe($param = NULL)
{
```

```
   print $param;
}
printMe("This is test");
printMe();
?>

</body>
</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.

```
<html>
<head>
<title>Dynamic Function Calls</title>
</head>
<body>
<?php
function sayHello()
{
   echo "Hello<br />";
}
$function_holder = "sayHello";
$function_holder();
?>
</body>
</html>
```

This will display following result:

```
Hello
```

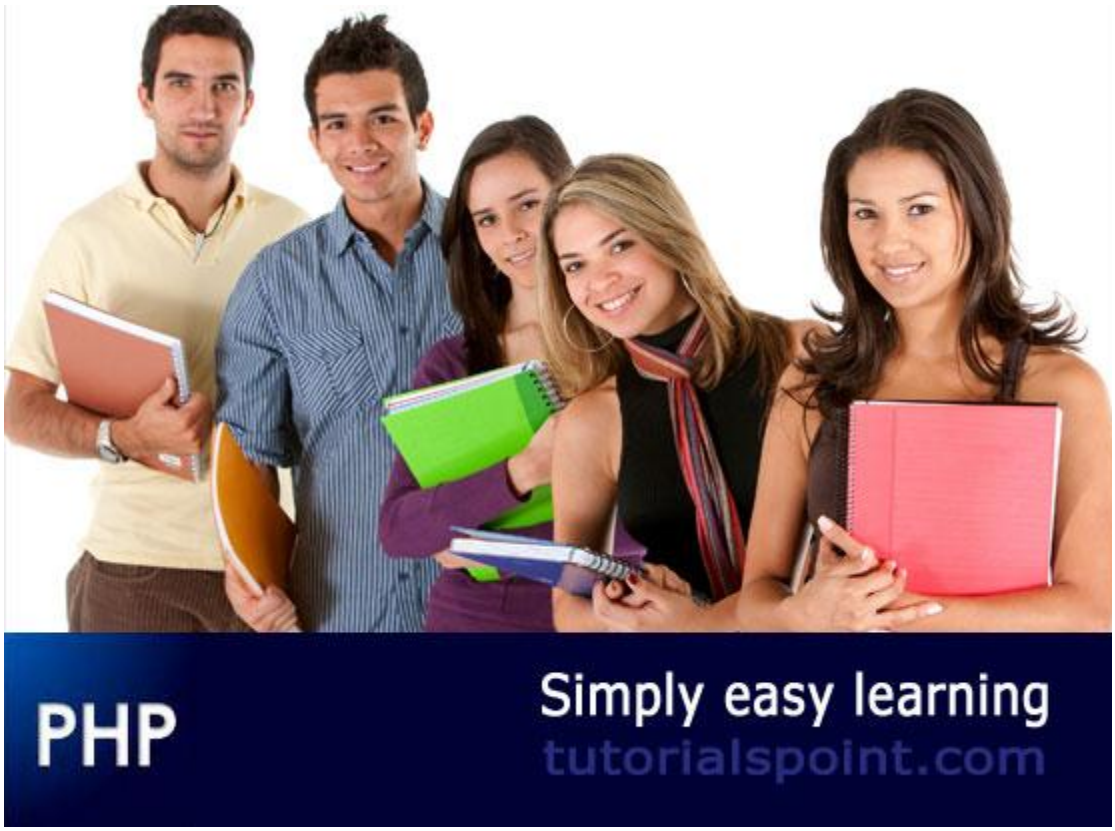
For complete Tutorial: http://www.tutorialspoint.com/php

List of Tutorials from TutorialsPoint.com	
▪ Learn JSP	▪ Learn ASP.Net
▪ Learn Servlets	▪ Learn HTML
▪ Learn log4j	▪ Learn HTML5
▪ Learn iBATIS	▪ Learn XHTML
▪ Learn Java	▪ Learn CSS
▪ Learn JDBC	▪ Learn HTTP
▪ Java Examples	▪ Learn JavaScript
▪ Learn Best Practices	▪ Learn jQuery
▪ Learn Python	▪ Learn Prototype
▪ Learn Ruby	▪ Learn script.aculo.us

▪ Learn Ruby on Rails ▪ Learn SQL ▪ Learn MySQL ▪ Learn AJAX ▪ Learn C Programming ▪ Learn C++ Programming ▪ Learn CGI with PERL ▪ Learn DLL ▪ Learn ebXML ▪ Learn Euphoria ▪ Learn GDB Debugger ▪ Learn Makefile ▪ Learn Parrot ▪ Learn Perl Script ▪ Learn PHP Script ▪ Learn Six Sigma ▪ Learn SEI CMMI ▪ Learn WiMAX ▪ Learn Telecom Billing	▪ Web Developer's Guide ▪ Learn RADIUS ▪ Learn RSS ▪ Learn SEO Techniques ▪ Learn SOAP ▪ Learn UDDI ▪ Learn Unix Sockets ▪ Learn Web Services ▪ Learn XML-RPC ▪ Learn UML ▪ Learn UNIX ▪ Learn WSDL ▪ Learn i-Mode ▪ Learn GPRS ▪ Learn GSM ▪ Learn WAP ▪ Learn WML ▪ Learn Wi-Fi
webmaster@TutorialsPoint.com	

PHP Tutorial

Tutorialspoint.com



The PHP Hypertext Pre-processor (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 tutorial helps you to build your base with PHP.

Send your feedback using Contact Us Form

Audience:

This tutorial 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 tutorial you should have at least basic understanding of computer programming, Internet, Database, and MySQL etc is very helpful.

PHP Introduction

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

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire 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, thru email you can send data, return data to the user.
- You add, delete, modify elements within your database thru 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

"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:

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

Most common tag is the <?php...?> and we will also use same tag in our tutorial.

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.

PHP 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 availble Apache Server. Download Apache for free here: http://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: http://www.mysql.com/downloads/index.html
- **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 tutorial will guide you how to install PHP parser on your computer.

PHP Parser Installation:

Before you proceed it is important to make sure that you have proper environment setup on your machine to develop your web programs using PHP.

Type the following address into your browser's address box.

```
http://127.0.0.1/info.php
```

If this displays a page showing your PHP installation related information then it means you have PHP and Webserver installed properly. Otherwise you have to follow given procedure to install PHP on your computer.

This section will guide you to install and configure PHP over the following four platforms:

- PHP Installation on Linux or Unix with Apache
- PHP Installation on Mac OS X with Apache
- PHP Installation on Windows NT/2000/XP with IIS
- PHP Installation on Windows NT/2000/XP with Apache

Apache Configuration:

If you are using Apache as a Web Server then this section will guide you to edit Apache Configuration Files.

Just Check it here : PHP Configuration in Apache Server

PHP.INI File Configuration:

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality.

Just Check it here: PHP.INI File Configuration

Windows IIS Configuration:

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

PHP 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.

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

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

```
<?
# First Example
print <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
# Second Example
print "This spans
multiple lines. The newlines will be
output as well";
?>
```

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.

```
<?
/* This is a comment with multiline
    Author : Mohammad Mohtashim
    Purpose: Multiline Comments Demo
    Subject: PHP
*/
print "An example with multi line comments";
```

```
?>
```

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:

```
$four = 2 + 2; // single spaces
$four <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs
$four =
2+
2; // multiple lines
```

PHP is case sensitive:

Yeah it is true that PHP is a case sensitive language. Try out following example:

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

This will produce 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:

```
if (3 == 2 + 1)
  print("Good - I haven't totally lost my mind.<br>");

if (3 == 2 + 1)
{
   print("Good - I haven't totally");
   print("lost my mind.<br>");
}
```

Running PHP Script from Command Prompt:

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

```
<?php
   echo "Hello PHP!!!!!";
?>
```

Now run this script as command prompt as follows:

```
$ php test.php
```

It will produce following result:

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

Hope now you have basic knowledge of PHP Syntax.

PHP 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 simile data type in this chapters. 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:

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

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 hexadecimals 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:

```
$many = 2.2888800;
$many_2 = 2.2111200;
$few = $many + $many_2;
print(.$many + $many_2 = $few<br>.);
```

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:

```
if (TRUE)
   print("This will always print<br>");
else
```

```
print("This will never print<br>");
```

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.

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

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

```
$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.

```
<?
$variable = "name";
$literally = 'My $variable will not print!\\n';
print($literally);
$literally = "My $variable will print!\\n";
print($literally);
?>
```

This will produce following result:

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

```
<?php

$channel =<<<_XML_
<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
</channel>
_XML_;

echo <<<END
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
<br />
```

```
END;

print $channel;
?>
```

This will produce following result:

```
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!

<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
```

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.

PHP Constants

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:

```
<?php

define("MINSIZE", 50);

echo MINSIZE;
echo constant("MINSIZE"); // same thing as the previous line

?>
```

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:

```
// Valid constant names
define("ONE",     "first thing");
define("TWO2",    "second thing");
define("THREE_3", "third thing")
// Invalid constant names
define("2TWO",    "second thing");
define("__THREE__", "third value");
```

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 ate given below:

Name	Description
__LINE__	The current line number of the file.
__FILE__	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, **__FILE__** always contains an absolute path whereas in older versions it contained relative path under some circumstances.
__FUNCTION__	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.
__CLASS__	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.
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

PHP 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
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Arithmatic Operators:

There are following arithmatic operators supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Show Examples

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 denumerator	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:

Show Examples

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:

Show Examples

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then 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:

Show Examples

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

	to left operand	
/=	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:

Show Examples

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 precedenace 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	= += -= *= /=	

PHP 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 threedecision 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 several condition are 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!":

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

</html>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>

<body>
<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>
</body>
</html>
```

The Elself Statement

If you want to execute some code if one of 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!":

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

The Switch Statement

If you want to select one of many blocks of code to be executed, 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 lable 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 lables match then statement will execute any specified default code.

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

```
</html>
```

PHP 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 trur.
- **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:

```
<html>
<body>
<?php
$a = 0;
$b = 0;

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

This will produce 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.

```
<html>
<body>
<?php
$i = 0;
$num = 50;

while( $i < 10)
{
   $num--;
   $i++;
}
echo ("Loop stopped at i = $i and num = $num" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 1 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:

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

This will produce 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.

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

This will produce 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. If 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.

```
<html>
<body>

<?php
$i = 0;

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

This will produce 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 skip the code and next value is printed.

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

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

This will produce following result

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

PHP 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 its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c 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 stores 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 prepresented 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.

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

```
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";

foreach( $numbers as $value )
{
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce 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 employees 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 otheriwse it would not return any value.

Example

```
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
                    "mohammad" => 2000,
                    "qadir" => 1000,
                    "zara" => 500
                   );

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
echo "Salary of zara is ".  $salaries['zara']. "<br />";

/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
```

```
echo "Salary of zara is ".  $salaries['zara']. "<br />";
?>
</body>
</html>
```

This will produce 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 index.

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.

```
<html>
<body>
<?php
   $marks = array(
                "mohammad" => array
                (
                "physics" => 35,
                "maths" => 30,
                "chemistry" => 39
                ),
                "qadir" => array
               (
               "physics" => 30,
               "maths" => 32,
               "chemistry" => 29
               ),
               "zara" => array
               (
               "physics" => 31,
               "maths" => 22,
               "chemistry" => 39
               )
            );
   /* Accessing multi-dimensional array values */
   echo "Marks for mohammad in physics : " ;
   echo $marks['mohammad']['physics'] . "<br />";
   echo "Marks for qadir in maths : ";
   echo $marks['qadir']['maths'] . "<br />";
   echo "Marks for zara in chemistry : " ;
   echo $marks['zara']['chemistry'] . "<br />";
?>
</body>
</html>
```

This will produce following result:

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

PHP 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.

```
<?
$variable = "name";
$literally = 'My $variable will not print!\\n';
print($literally);
$literally = "My $variable will print!\\n";
print($literally);
?>
```

This will produce following result:

```
My $variable will not print!\n
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 (\)

String Concatenation Operator

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

```
<?php
$string1="Hello World";
$string2="1234";
echo $string1 . " " . $string2;
?>
```

This will produce 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!":

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

This will produce 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:

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

This will produce following result:

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

PHP 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 included 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="http://www.tutorialspoint.com/index.htm">Home</a> -
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -
<a href="http://www.tutorialspoint.com/perl">PERL</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.

```
<html>
<body>
<?php include("menu.php"); ?>
<p>This is an example to show how to include PHP file!</p>
</body>
</html>
```

This will produce following result

Home - ebXML - AJAX - PERL

This is an example to show how to include PHP file. You can include mean.php file in as many as files you like!

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 excution 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.

```
<html>
<body>
<?php include("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This will produce following result

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

Now lets try same example with require() function.

```
<html>
<body>
<?php require("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</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.

PHP 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.

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

Mode	Purpose

r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	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 attemts to create a file.
w+	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 attemts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attemts to create a file.
a+	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 attemts 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's 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.

```
<html>
```

```
<head>
<title>Reading a file using PHP</title>
</head>
<body>

<?php
$filename = "/home/user/guest/tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
   echo ( "Error in opening file" );
   exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );

fclose( $file );

echo ( "File size : $filesize bytes" );
echo ( "<pre>$text</pre>" );
?>

</body>
</html>
```

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 intger argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

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

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
   echo ( "Error in opening new file" );
   exit();
}
fwrite( $file, "This is  a simple test\n" );
fclose( $file );
?>

<html>
<head>
<title>Writing a file using PHP</title>
</head>
<body>

<?php
if( file_exist( $filename ) )
{
   $filesize = filesize( $filename );
   $msg = "File  created with name $filename ";
   $msg .= "containing $filesize bytes";
   echo ($msg );
}
```

```
else
{
   echo ("File $filename does not exit" );
}
?>
</body>
</html>
```

We have covered all the function related to file input and out in PHP File System Function chapter.

PHP 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 PHP Function Reference for a complete set of useful functions.

Creating PHP Function:

Its 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:

```
<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>

<?php
/* Defining a PHP Function */
function writeMessage()
{
  echo "You are really a nice person, Have a nice time!";
}
/* Calling a PHP Function */
writeMessage();
?>
</body>
</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.

```
<html>
<head>
<title>Writing PHP Function with Parameters</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  echo "Sum of the two numbers is : $sum";
}
addFunction(10, 20);
?>
</body>
</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.

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function addFive($num)
{
   $num += 5;
}

function addSix(&$num)
{
   $num += 6;
}
$orignum = 10;
addFive( &$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );
```

```
echo "Original Value is $orignum<br />";
?>
</body>
</html>
```

This will display following result:

```
Original Value is 15
Original Value is 21
```

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.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  return $sum;
}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value
?>
</body>
</html>
```

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.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function printMe($param = NULL)
{
```

```
   print $param;
}
printMe("This is test");
printMe();
?>

</body>
</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.

```
<html>
<head>
<title>Dynamic Function Calls</title>
</head>
<body>
<?php
function sayHello()
{
   echo "Hello<br />";
}
$function_holder = "sayHello";
$function_holder();
?>
</body>
</html>
```

This will display following result:

```
Hello
```

For complete Tutorial: http://www.tutorialspoint.com/php

List of Tutorials from TutorialsPoint.com	
▪ Learn JSP	▪ Learn ASP.Net
▪ Learn Servlets	▪ Learn HTML
▪ Learn log4j	▪ Learn HTML5
▪ Learn iBATIS	▪ Learn XHTML
▪ Learn Java	▪ Learn CSS
▪ Learn JDBC	▪ Learn HTTP
▪ Java Examples	▪ Learn JavaScript
▪ Learn Best Practices	▪ Learn jQuery
▪ Learn Python	▪ Learn Prototype
▪ Learn Ruby	▪ Learn script.aculo.us

▪ Learn Ruby on Rails ▪ Learn SQL ▪ Learn MySQL ▪ Learn AJAX ▪ Learn C Programming ▪ Learn C++ Programming ▪ Learn CGI with PERL ▪ Learn DLL ▪ Learn ebXML ▪ Learn Euphoria ▪ Learn GDB Debugger ▪ Learn Makefile ▪ Learn Parrot ▪ Learn Perl Script ▪ Learn PHP Script ▪ Learn Six Sigma ▪ Learn SEI CMMI ▪ Learn WiMAX ▪ Learn Telecom Billing	▪ Web Developer's Guide ▪ Learn RADIUS ▪ Learn RSS ▪ Learn SEO Techniques ▪ Learn SOAP ▪ Learn UDDI ▪ Learn Unix Sockets ▪ Learn Web Services ▪ Learn XML-RPC ▪ Learn UML ▪ Learn UNIX ▪ Learn WSDL ▪ Learn i-Mode ▪ Learn GPRS ▪ Learn GSM ▪ Learn WAP ▪ Learn WML ▪ Learn Wi-Fi
webmaster@TutorialsPoint.com	