The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases.

PHP is a server side scripting language that is embedded in HTML.

<html>

<head>

<title>Hello World</title>

</head>

<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>

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

<html>

<head>

<title>Online PHP Script Execution</title>

</head>

<body>

<?php

echo "<h1>Hello, PHP!</h1>";

?>

</body>

</html>

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.

——————

**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**

\* 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 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";

?>

\* 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";

?>

**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 =

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>

<?php

$capital = 67;

print("Variable capital is $capital<br>");

print("Variable CaPiTaL is $CaPiTaL<br>");

?>

</body>

</html>

This will produce the following result −

Variable capital is 67

Variable CaPiTaL is

**Statements are expressions terminated by semicolons**

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

$greeting = "Welcome to PHP!";

**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 the following result −

Hello PHP!!!!!

**Variable Types**

\* 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.

\* Doubles

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

<?php

$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>”);

\* 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

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

<?php

$variable = "name";

$literally = 'My $variable will not print!';

print($literally);

print "<br>";

$literally = "My $variable will print!";

print($literally);

?>

This will produce following result −

My $variable will not print!\n

My name will print

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!

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>

**Constants Types**

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase.

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

?>

PHP Magic constants

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).

**Operator Types**

Conditional Operator

Operator Description Example

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

**Decision Making**

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>

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>

<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>

It will produce the following result −

Today is Sunday

**Loop Types**

<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 the following result −

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

<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 the following result −

Loop stopped at i = 10 and num = 40

<html>

<body>

<?php

$i = 0;

$num = 0;

do{

$i++;

}

while( $i < 10 );

echo ("Loop stopped at i = $i" );

?>

</body>

</html>

This will produce the following result −

Loop stopped at i = 10

<html>

<body>

<?php

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

foreach( $array as $value )

{

echo "Value is $value <br />";

}

?>

</body>

</html>

This will produce the following result −

Value is 1

Value is 2

Value is 3

Value is 4

Value is 5

<html>

<body>

<?php

$i = 0;

while( $i < 10)

{

$i++;

if( $i == 3 )break;

}

echo ("Loop stopped at i = $i" );

?>

</body>

</html>

This will produce the following result −

Loop stopped at i = 3

<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 the following result −

Value is 1

Value is 2

Value is 4

Value is 5

**Arrays**

<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 the following result −

Value is 1

Value is 2

Value is 3

Value is 4

Value is 5

Value is one

Value is two

Value is three

Value is four

Value is five

<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 the following result −

Salary of mohammad is 2000

Salary of qadir is 1000

Salary of zara is 500

Salary of mohammad is high

Salary of qadir is medium

Salary of zara is low

<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 the following result −

Marks for mohammad in physics : 35

Marks for qadir in maths : 32

Marks for zara in chemistry : 39

**Strings**

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

<?php

$string1="Hello World";

$string2="1234";

echo $string1 . " " . $string2;

?>

This will produce the following result −

Hello World 1234

<?php

echo strlen("Hello world!");

?>

This will produce the following result −

12

<?php

echo strpos("Hello world!","world");

?>

This will produce the following result −

6

**Identifying Browser & Platform**

<html>

<body>

<?php

function getBrowser()

{

$u\_agent = $\_SERVER['HTTP\_USER\_AGENT'];

$bname = 'Unknown';

$platform = 'Unknown';

$version= "";

//First get the platform?

if (preg\_match('/linux/i', $u\_agent)) {

$platform = 'linux';

}

elseif (preg\_match('/macintosh|mac os x/i', $u\_agent)) {

$platform = 'mac';

}

elseif (preg\_match('/windows|win32/i', $u\_agent)) {

$platform = 'windows';

}

// Next get the name of the useragent yes seperately and for good reason

if(preg\_match('/MSIE/i',$u\_agent) && !preg\_match('/Opera/i',$u\_agent))

{

$bname = 'Internet Explorer';

$ub = "MSIE";

}

elseif(preg\_match('/Firefox/i',$u\_agent))

{

$bname = 'Mozilla Firefox';

$ub = "Firefox";

}

elseif(preg\_match('/Chrome/i',$u\_agent))

{

$bname = 'Google Chrome';

$ub = "Chrome";

}

elseif(preg\_match('/Safari/i',$u\_agent))

{

$bname = 'Apple Safari';

$ub = "Safari";

}

elseif(preg\_match('/Opera/i',$u\_agent))

{

$bname = 'Opera';

$ub = "Opera";

}

elseif(preg\_match('/Netscape/i',$u\_agent))

{

$bname = 'Netscape';

$ub = "Netscape";

}

// finally get the correct version number

$known = array('Version', $ub, 'other');

$pattern = '#(?<browser>' . join('|', $known) . ')[/ ]+(?<version>[0-9.|a-zA-Z.]\*)#';

if (!preg\_match\_all($pattern, $u\_agent, $matches)) {

// we have no matching number just continue

}

// see how many we have

$i = count($matches['browser']);

if ($i != 1) {

//we will have two since we are not using 'other' argument yet

//see if version is before or after the name

if (strripos($u\_agent,"Version") < strripos($u\_agent,$ub)){

$version= $matches['version'][0];

}

else {

$version= $matches['version'][1];

}

}

else {

$version= $matches['version'][0];

}

// check if we have a number

if ($version==null || $version=="") {$version="?";}

return array(

'userAgent' => $u\_agent,

'name' => $bname,

'version' => $version,

'platform' => $platform,

'pattern' => $pattern

);

}

// now try it

$ua=getBrowser();

$yourbrowser= "Your browser: " . $ua['name'] . " " . $ua['version'] . " on " .$ua['platform'] . " reports: <br >" . $ua['userAgent'];

print\_r($yourbrowser);

?>

</body>

</html>

It will produce the following result −

Your browser: Mozilla Firefox 41.0 on mac reports:

Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:41.0) Gecko/20100101 Firefox/41.0

**Display Images Randomly**

<html>

<body>

<?php

srand( microtime() \* 1000000 );

$num = rand( 1, 4 );

switch( $num )

{

case 1: $image\_file = "/php/images/logo.png";

break;

case 2: $image\_file = "/php/images/php.jpg";

break;

case 3: $image\_file = "/php/images/logo.png";

break;

case 4: $image\_file = "/php/images/php.jpg";

break;

}

echo "Random Image : <img src=$image\_file />";

?>

</body>

</html>

**Using HTML Forms**

<?php

if( $\_POST["name"] || $\_POST["age"] )

{

if (preg\_match("/[^A-Za-z'-]/",$\_POST['name'] ))

{

die ("invalid name and name should be alpha");

}

echo "Welcome ". $\_POST['name']. "<br />";

echo "You are ". $\_POST['age']. " years old.";

exit();

}

?>

<html>

<body>

<form action="<?php $\_PHP\_SELF ?>" method="POST">

Name: <input type="text" name="name" />

Age: <input type="text" name="age" />

<input type="submit" />

</form>

</body>

</html>

It will produce the following result −

Welcome test

You are 12 years old.

The PHP default variable $\_PHP\_SELF is used for the PHP script name and when you click "submit" button then same PHP script will be called.

**Browser Redirection**

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

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

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

<?php

if( $\_POST["location"] )

{

$location = $\_POST["location"];

header( "Location:$location" );

exit();

}

?>

<html>

<body>

<p>Choose a site to visit :</p>

<form action="<?php $\_SERVER['PHP\_SELF'] ?>" method="POST">

<select name="location">.

<option value="http://www.tutorialspoint.com">

Tutorialspoint.com

</option>

<option value="http://www.google.com">

Google Search Page

</option>

</select>

<input type="submit" />

</form>

</body>

</html>

**Displaying "File Download" Dialog Box**

Sometime it is desired that you want to give option where a use will click a link and it will pop up a "File Download" box to the user in stead of displaying actual content. This is very easy and will be achieved through HTTP header.

The HTTP header will be different from the actual header where we send Content-Type as text/html\n\n. In this case content type will be application/octet-stream and actual file name will be concatenated along with it.

For example,if you want make a FileName file downloadable from a given link then its syntax will be as follows.

#!/usr/bin/perl

# HTTP Header

print "Content-Type:application/octet-stream; name=\"FileName\"\r\n";

print "Content-Disposition: attachment; filename=\"FileName\"\r\n\n";

# Actual File Content

open( FILE, "<FileName" );

while(read(FILE, $buffer, 100) )

{

print("$buffer");

}

**GET & POST Methods**

<?php

if( $\_GET["name"] || $\_GET["age"] )

{

echo "Welcome ". $\_GET['name']. "<br />";

echo "You are ". $\_GET['age']. " years old.";

exit();

}

?>

<html>

<body>

<form action="<?php $\_PHP\_SELF ?>" method="GET">

Name: <input type="text" name="name" />

Age: <input type="text" name="age" />

<input type="submit" />

</form>

</body>

</html>

It will produce the following result −

Welcome test

You are 12 years old.

<?php

if( $\_POST["name"] || $\_POST["age"] )

{

if (preg\_match("/[^A-Za-z'-]/",$\_POST['name'] ))

{

die ("invalid name and name should be alpha");

}

echo "Welcome ". $\_POST['name']. "<br />";

echo "You are ". $\_POST['age']. " years old.";

exit();

}

?>

<html>

<body>

<form action="<?php $\_PHP\_SELF ?>" method="POST">

Name: <input type="text" name="name" />

Age: <input type="text" name="age" />

<input type="submit" />

</form>

</body>

</html>

It will produce the following result −

Welcome test

You are 12 years old.

The PHP $\_REQUEST variable contains the contents of both $\_GET, $\_POST, and $\_COOKIE.

<?php

if( $\_REQUEST["name"] || $\_REQUEST["age"] )

{

echo "Welcome ". $\_REQUEST['name']. "<br />";

echo "You are ". $\_REQUEST['age']. " years old.";

exit();

}

?>

<html>

<body>

<form action="<?php $\_PHP\_SELF ?>" method="POST">

Name: <input type="text" name="name" />

Age: <input type="text" name="age" />

<input type="submit" />

</form>

</body>

</html>

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

It will produce the following result −

Welcome test

You are 12 years old.

**File Inclusion**

＊ 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>

It will produce the following result −

Home -

ebXML -

AJAX -

PERL

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

＊ The require() Function

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

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

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

<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 the 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.

**Files & I/O**

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 = "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>$filetext</pre>" );

?>

</body>

</html>

It will produce the following result −

File size : 278 bytes

The PHP Hypertext Preprocessor (PHP) is a programming

language that allows web developers to create dynamic

content that interacts with databases.

PHP is basically used for developing web based software

applications. This tutorial helps you to build your base

with PHP.

<?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 );

?>

**Functions**

<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!

<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

<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 10

Original Value is 16

<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

<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

<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

**PHP接口类interface的正确使用方法**

当有很多人一起开发一个项目时，可能都会去调用别人写的一些类，那你就会问，我怎么知道他的某个功能的实现方法是怎么命名的呢，这个时候PHP接口类interface就起到作用了，当我们定义了一个接口类时，它里面的方式是下面的子类必须实现的，比如 :

interface Shop

{

public function buy($gid);

public function sell($gid);

public function view($gid);

}

我声明一个shop接口类，定义了三个方法：买(buy),卖(sell),看(view),那么继承此类的所有子类都必须实现这3个方法少一个都不行，如果子类没有实现这些话，就无法运行。实际上接口类说白了，就是一个类的模板，一个类的规定，如果你属于这类，你就必须遵循我的规定，少一个都不行，但是具体你怎么去做，我不管，那是你的事，如：

class BaseShop implements Shop

{

public function buy($gid)

{

echo('你购买了ID为 :'.$gid.'的商品');

}

public function sell($gid)

{

echo('你卖了ID为 :'.$gid.'的商品');

}

public function view($gid)

{

echo('你查看了ID为 :'.$gid.'的商品');

}

}

**Cookies**

＊ The Anatomy of a Cookie

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

HTTP/1.1 200 OK

Date: Fri, 04 Feb 2000 21:03:38 GMT

Server: Apache/1.3.9 (UNIX) PHP/4.0b3

Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;

path=/; domain=tutorialspoint.com

Connection: close

Content-Type: text/html

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

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

GET / HTTP/1.0

Connection: Keep-Alive

User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)

Host: zink.demon.co.uk:1126

Accept: image/gif, \*/\*

Accept-Encoding: gzip

Accept-Language: en

Accept-Charset: iso-8859-1,\*,utf-8

Cookie: name=xyz

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

＊ Setting Cookies with PHP

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

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

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

<?php

setcookie("name", "John Watkin", time()+3600, "/","", 0);

setcookie("age", "36", time()+3600, "/", "", 0);

?>

<html>

<head>

<title>Setting Cookies with PHP</title>

</head>

<body>

<?php echo "Set Cookies"?>

</body>

</html>

＊ Accessing Cookies with PHP

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

<html>

<head>

<title>Accessing Cookies with PHP</title>

</head>

<body>

<?php

echo $\_COOKIE["name"]. "<br />";

/\* is equivalent to \*/

echo $HTTP\_COOKIE\_VARS["name"]. "<br />";

echo $\_COOKIE["age"] . "<br />";

/\* is equivalent to \*/

echo $HTTP\_COOKIE\_VARS["name"] . "<br />";

?>

</body>

</html>

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

<html>

<head>

<title>Accessing Cookies with PHP</title>

</head>

<body>

<?php

if( isset($\_COOKIE["name"]))

echo "Welcome " . $\_COOKIE["name"] . "<br />";

else

echo "Sorry... Not recognized" . "<br />";

?>

</body>

</html>

＊ Deleting Cookie with PHP

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

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

<?php

setcookie( "name", "", time()- 60, "/","", 0);

setcookie( "age", "", time()- 60, "/","", 0);

?>

<html>

<head>

<title>Deleting Cookies with PHP</title>

</head>

<body>

<?php echo "Deleted Cookies" ?>

</body>

</html>

**Sessions**

<?php

session\_start();

if( isset( $\_SESSION['counter'] ) )

{

$\_SESSION['counter'] += 1;

}

else

{

$\_SESSION['counter'] = 1;

}

$msg = "You have visited this page ". $\_SESSION['counter'];

$msg .= "in this session.";

?>

<html>

<head>

<title>Setting up a PHP session</title>

</head>

<body>

<?php echo ( $msg ); ?>

</body>

</html>

It will produce the following result −

You have visited this page 1in this session.

If you want to destroy a single session variable then you can use unset() function to unset a session variable.

Here is the example to unset a single variable −

<?php

unset($\_SESSION['counter']);

?>

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

<?php

session\_destroy();

?>

Turning on Auto Session

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

**File Uploading**

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

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

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

<?php

if(isset($\_FILES['image'])){

$errors= array();

$file\_name = $\_FILES['image']['name'];

$file\_size =$\_FILES['image']['size'];

$file\_tmp =$\_FILES['image']['tmp\_name'];

$file\_type=$\_FILES['image']['type'];

$file\_ext=strtolower(end(explode('.',$\_FILES['image']['name'])));

$expensions= array("jpeg","jpg","png");

if(in\_array($file\_ext,$expensions)=== false){

$errors[]="extension not allowed, please choose a JPEG or PNG file.";

}

if($file\_size > 2097152){

$errors[]='File size must be excately 2 MB';

}

if(empty($errors)==true){

move\_uploaded\_file($file\_tmp,"images/".$file\_name);

echo "Success";

}

else{

print\_r($errors);

}

}

?>

<html>

<body>

<form action="" method="POST" enctype="multipart/form-data">

<input type="file" name="image" />

<input type="submit"/>

<ul>

<li>Sent file: <?php echo $\_FILES['image']['name']; ?>

<li>File size: <?php echo $\_FILES['image']['size']; ?>

<li>File type: <?php echo $\_FILES['image']['type'] ?>

</ul>

</form>

</body>

</html>

It will produce the following result −

Sent file:

File size:

File type:

**Object Oriented Programming in PHP**

＊ Defining PHP Classes

The general form for defining a new class in PHP is as follows −

<?php

class phpClass{

var $var1;

var $var2 = "constant string";

function myfunc ($arg1, $arg2) {

[..]

}

[..]

}

?>

<?php

class Books{

/\* Member variables \*/

var $price;

var $title;

/\* Member functions \*/

function setPrice($par){

$this->price = $par;

}

function getPrice(){

echo $this->price ."<br/>";

}

function setTitle($par){

$this->title = $par;

}

function getTitle(){

echo $this->title ." <br/>";

}

}

?>

The variable $this is a special variable and it refers to the same object ie. itself.

＊ Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using new operator.

$physics = new Books;

$maths = new Books;

$chemistry = new Books;

＊ Calling Member Functions

Following example shows how to set title and prices for the three books by calling member functions.

$physics->setTitle( "Physics for High School" );

$chemistry->setTitle( "Advanced Chemistry" );

$maths->setTitle( "Algebra" );

$physics->setPrice( 10 );

$chemistry->setPrice( 15 );

$maths->setPrice( 7 );

Now you call another member functions to get the values set by in above example −

$physics->getTitle();

$chemistry->getTitle();

$maths->getTitle();

$physics->getPrice();

$chemistry->getPrice();

$maths->getPrice();

This will produce the following result −

Physics for High School

Advanced Chemistry

Algebra

10

15

7

＊ Constructor Functions

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

function \_\_construct( $par1, $par2 ){

$this->price = $par1;

$this->title = $par2;

}

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check following example below −

$physics = new Books( "Physics for High School", 10 );

$maths = new Books ( "Advanced Chemistry", 15 );

$chemistry = new Books ("Algebra", 7 );

/\* Get those set values \*/

$physics->getTitle();

$chemistry->getTitle();

$maths->getTitle();

$physics->getPrice();

$chemistry->getPrice();

$maths->getPrice();

This will produce the following result −

Physics for High School

Advanced Chemistry

Algebra

10

15

7

＊ Destructor

Like a constructor function you can define a destructor function using function \_\_destruct(). You can release all the resources with-in a destructor.

＊ Inheritance

PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows −

class Child extends Parent {

<definition body>

}

Following example inherit Books class and adds more functionality based on the requirement.

class Novel extends Books{

var publisher;

function setPublisher($par){

$this->publisher = $par;

}

function getPublisher(){

echo $this->publisher. "<br />";

}

}

Now apart from inherited functions, class Novel keeps two additional member functions.

＊ Function Overriding

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.

In the following example getPrice and getTitle functions are overridden to return some values.

function getPrice(){

echo $this->price . "<br/>";

return $this->price;

}

function getTitle(){

echo $this->title . "<br/>";

return $this->title;

}

＊ Private members

A class member can be made private by using private keyword infront of the member.

class MyClass {

private $car = "skoda";

$driver = "SRK";

function \_\_construct($par) {

// Statements here run every time

// an instance of the class

// is created.

}

function myPublicFunction() {

return("I'm visible!");

}

private function myPrivateFunction() {

return("I'm not visible outside!");

}

}

When MyClass class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

＊ Interfaces

As of PHP5, it is possible to define an interface, like this −

interface Mail {

public function sendMail();

}

Then, if another class implemented that interface, like this −

class Report implements Mail {

// sendMail() Definition goes here

}

\* Constants

A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable. Once you declare a constant, it does not change.

Declaring one constant is easy, as is done in this version of MyClass −

class MyClass {

const requiredMargin = 1.7;

function \_\_construct($incomingValue) {

// Statements here run every time

// an instance of the class

// is created.

}

}

In this class, requiredMargin is a constant. It is declared with the keyword const, and under no circumstances can it be changed to anything other than 1.7. Note that the constant's name does not have a leading $, as variable names do.

* Abstract Classes

An abstract class is one that cannot be instantiated, only inherited.

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

abstract class MyAbstractClass {

abstract function myAbstractFunction() {

}

}

Note that function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

* Static Keyword

Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static can not be accessed with an instantiated class object (though a static method can).

Try out following example −

<?php

class Foo

{

public static $my\_static = 'foo';

public function staticValue() {

return self::$my\_static;

}

}

print Foo::$my\_static . "\n";

$foo = new Foo();

print $foo->staticValue() . "\n";

?>

* Final Keyword

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.

Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

<?php

class BaseClass {

public function test() {

echo "BaseClass::test() called<br>";

}

final public function moreTesting() {

echo "BaseClass::moreTesting() called<br>";

}

}

class ChildClass extends BaseClass {

public function moreTesting() {

echo "ChildClass::moreTesting() called<br>";

}

}

?>

* Calling parent constructors

Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass.

class Name

{

var $\_firstName;

var $\_lastName;

function Name($first\_name, $last\_name)

{

$this->\_firstName = $first\_name;

$this->\_lastName = $last\_name;

}

function toString() {

return($this->\_lastName .", " .$this->\_firstName);

}

}

class NameSub1 extends Name

{

var $\_middleInitial;

function NameSub1($first\_name, $middle\_initial, $last\_name) {

Name::Name($first\_name, $last\_name);

$this->\_middleInitial = $middle\_initial;

}

function toString() {

return(Name::toString() . " " . $this->\_middleInitial);

}

}

**Design Patterns**

\* Singleton

A Class has one instance, It provides a global access point to it, Following code will explain about singleton concept.

<?php

class Singleton

{

public static function getInstance()

{

static $instance = null;

if (null === $instance) {

$instance = new static();

}

return $instance;

}

protected function \_\_construct()

{

}

private function \_\_clone()

{

}

private function \_\_wakeup()

{

}

}

class SingletonChild extends Singleton

{

}

$obj = Singleton::getInstance();

var\_dump($obj === Singleton::getInstance());

$anotherObj = SingletonChild::getInstance();

var\_dump($anotherObj === Singleton::getInstance());

var\_dump($anotherObj === SingletonChild::getInstance());

?>

Above Example implemented based on static method creation is getInstance()

\* Factory

A Class Simple Creates the object and you want to use that object, Following example will explain about factory design pattern.

<?php

class Automobile

{

private $bikeMake;

private $bikeModel;

public function \_\_construct($make, $model)

{

$this->bikeMake = $make;

$this->bikeModel = $model;

}

public function getMakeAndModel()

{

return $this->bikeMake . ' ' . $this->bikeModel;

}

}

class AutomobileFactory

{

public static function create($make, $model)

{

return new Automobile($make, $model);

}

}

$pulsar = AutomobileFactory::create('ktm', 'Pulsar');

print\_r($pulsar->getMakeAndModel());

?>

The main difficulty with factory pattern is it will increase the complexity and it is not reliable for good programmers.

\* Strategy pattern

Strategy pattern makes a family algorithm and encapsulates each algorithm. Here each algorithm should be inter-changeable within the family.

<?php

$elements = array(

array(

'id' => 2,

'date' => '2011-01-01',

),

array(

'id' => 1,

'date' => '2011-02-01'

)

);

$collection = new ObjectCollection($elements);

$collection->setComparator(new IdComparator());

$collection->sort();

echo "Sorted by ID:\n";

print\_r($collection->elements);

$collection->setComparator(new DateComparator());

$collection->sort();

echo "Sorted by date:\n";

print\_r($collection->elements);

?>