**BUILDING BLOCKS OF PHP:-**

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.

**What is PHP**

1. PHP stands for HyperTextPreprocessor.
2. PHP is an interpreted language, i.e. there is no need for compilation.
3. PHP is a server side scripting language.
4. PHP is faster than other scripting language e.g. asp and jsp.

**PHP Features**

1. **Performance**: Script written in PHP executes much faster then those scripts written in other languages such as JSP & ASP.
2. **Open Source Software**: PHP source code is free available on the web, you can developed all the version of PHP according to your requirement without paying any cost.
3. **Platform Independent**: PHP is available for WINDOWS, MAC, LINUX& UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.
4. **Compatibility**: PHP is compatible with almost all local servers used today like Apache, IIS etc.
5. **Embedded**: PHP code can be easily embedded within HTML tags and script.

**Basic PHP Syntax**

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

A PHP script can be placed anywhere in the document.

A PHP script starts with **<?php** and ends with **?>**:

<?php  
// PHP code goes here  
?>

The default file extension for PHP files is ".php".

A PHP file normally contains HTML tags, and some PHP scripting code.

Let's see a simple PHP example where we are writing some text using PHP echo command.

**<html>**

**<body>**

**<?php**

echo "**<h2>**Hello First PHP**</h2>**";

**?>**

**</body>**

**</html>**

**Echo** and **print** are two commands to print data on to the screen.

## COMMENTS IN PHP

A comment in PHP code is a line that is not read/executed as part of the program.

we use // to make a single-line comment

/\* \*/ to make a large comment block.

**Example:**

<html>

<body>

<?php

//This is a comment

/\*

This is

a comment

block

\*/

?>

</body>

</html>

**VARIABLE**

A variable in PHP is a name of memory location that holds data. A variable is a temporary storage that is used to store data temporarily. It hold values like number, string, object, array or a Boolean.

In PHP, a variable is declared using $ sign followed by variable name.

Syntax of declaring a variable in PHP is given below:

$variablename=value;

**Example:**

$str=”php”;

$x=5;

**Rules for PHP variables:**

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

**CONSTANTS**

Constant is a name or identifier that can't be changed during the execution of the script. PHP constants can be defined in 2 ways:

1. **Using define() function**

**syntax**:define(name, value, case-insensitive)

1. name: specifies the constant name
2. value: specifies the constant value
3. case-insensitive: Default value is false. It means it is case sensitive by default.

**<?php**

define("MESSAGE","Hello JavaTpoint PHP",true/false);

echo MESSAGE;

**?>**

1. **Using const keyword**

It is always case sensitive. It is faster that define( ).

**<?php**

const MESSAGE="Hello const by JavaTpoint PHP";

echo MESSAGE;

**?>**

**DATA TYPES**

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

* String
* Integer
* Float (floating point numbers - also called double)
* Boolean
* Array
* Object
* NULL
* Resource

## PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

### Example

<?php   
$x = "Helloworld!";  
$y = 'Helloworld!';  
echo $x;  
echo "<br>";   
echo $y;  
?>

## PHP Integer

## An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

## Rules for integers:

* An integer must have at least one digit
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example $x is an integer. The PHP var\_dump() function returns the data type and value:

**Example**

<?php   
$x = 5985;  
var\_dump($x);  
?>

## PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

### Example:

<?php   
$x = 10.365;  
var\_dump($x);  
?>

## PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

$x = true;  
$y = false;

## PHP Array

An array stores multiple values in one single variable.

### Example:

### <?php  $cars = array("Volvo","BMW","Toyota"); var\_dump($cars); ?>

## PHP Object

An object is a data type which stores data and information on how to process that data.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

### Example

<?php  
class Car {  
    function Car() {  
        $this->model = "VW";  
    }  
}  
// create an object  
$herbie = new Car();  
// show object properties  
echo $herbie->model;  
?>

## PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

**Tip:** If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

### Example:

<?php  
$x = "Hello world!";  
$x = null;  
var\_dump($x);  
?>

**PHP Resource**

Reference to a third-party resource( a database, for example)

**CHANGING TYPE WITH settype()**

PHP also provides the function settype(), which is used to change the type of avariable.

settype($variabletochange, ‘new type’);

**Example:**

<?php

$x = 3.14;

echo “is “.$undecided.” a double? “.is\_double($x).”<br/>”; // double

settype($x, ‘string’);

echo “is “.$x.” a string? “.is\_string($x).”<br/>”; // string

settype($x, ‘integer’);

echo “is “.$undecided.” an integer? “.is\_integer($x).”<br/>”; //

?>

**CHANGING TYPE BY CASTING**

The principal difference between using settype() to change the type of an existingvariable and changing type by *casting*is the fact that casting produces a copy, leavingthe original variable untouched. To change type through casting, you indicatethe name of a data type, in parentheses, in front of the variable you are copying.

$newvar = (integer) $originalvar

**Example:**

<?php

$x = 3.14;

$holder = (double) $x;

echo “is “.$holder.” a double? “.is\_double($holder).”<br/>”; // double

$holder = (string) $x;

echo “is “.$holder.” a string? “.is\_string($holder).”<br/>”; // string

**?>**

**OPERATORS AND EXPRESSIONS**

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.

An expression is a sequence of operands and operators that reduces to a single value.

1. **Arithmetic Operators:** Arithmetic operators are used to perform arithmetic calculations.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition  <?php  $a=10;  $b=20;  echo $a+$b."<br>";  echo $b-$a."<br>";  echo $a\*$b."<br>";  echo $b/$a."<br>";  echo $b%$a."<br>";  ?> |
|  | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus (division remainder) |

1. **Comparison Operators:** The comparison operators are used to compare two values and gives either true (1) or false (0) result. The following are the comparison operators.

<?php

$a=10;

$b=20;

echo " a < b : " .($a < $b)."<br>";

echo "b > a :" .($b > $a)."<br>";

echo " a!=b:" .($a != $b)."<br>";

echo "a==b:".($a==$b)."<br>";

echo "a <= b:".($a<=$b)."<br>";

echo "a >= b:".($b>=$a)."<br>";

?>

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | is equal to |
| != | is not equal |
| > | is greater than |
| < | is less than |
| >= | is greater than or equal to |
| <= | is less than or equal to |
|  |  |

1. **Logical Operators:** These are used to combine two or more comparison expression and give the result either true or false.

<?php

$a=10;

$b=20;

echo ($a > 4) && ($a <33);

?>

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | and |
| || | Or |
| ! | Not |

1. **Assignment Operators:** Assignment operator is used to assign the result of an expression to a variable.

**V op=exp**

1. **Increment / Decrement Operator:** The increment and decrement operator are unary operators.

<?php

$a=10;

$b=20;

echo "a++:".$a++."<br>";

echo "++a:".++$a."<br>";

echo "--b:".--$b."<br>";

echo "b--:".$b--."<br>";

echo "b--:".$b--."<br>";

?>

The increment operator increases the value of an operand by 1.

The decrement operator decreases the value of an operand by 1.

Increment types are pre (++m) and post (m++) increment.

Decrement types are pre (--m) and post (m--) decrement.

1. **Concatenation operator:** The concatenation operator is represented by a single period (.). This operator appends the right-side operand to the left-side operand.

**Example:**

**“ hello”. “ world”**

It returns hello world

**7. Conditional or Ternary Operator:** The general form of conditional operator is as follows:

**Syntax:** (Exp1)? Exp2:Exp3;

Working: Exp1 is evaluated first, if it is true, Exp2 will be executed. If Exp1 is false Exp3 will be executed.

**Example:**

<?php

$x=10;

$y=23;

($x>$y)?print " $x is larger" : print "$y is larger";

?>

**FLOW CONTROL FUNCTIONS IN PHP:-**

**SWITCHING FLOW**

It is common for scripts to evaluate conditions and change their behavior accordingly. These decisions are what make your PHP pages dynamic—that is, able to change output according to circumstances. Like most programming languages, PHP enables you to do this with an if statement.

**Example:**

<?php

$mood = “happy”;

if ($mood == “happy”) {

echo “Hooray! I’m in a good mood!”;

}

?>

**If statement:**

Syntax**:** if (*expression*) {

// statement-block

}

If this expression results in a true value, the statement is executed.

Otherwise, the statement is skipped entirely.

**Example:**

<?php

$mood = “sad”;

if ($mood == “happy”) {

echo “Hooray! I’m in a good mood!”;

} else {

echo “I’m in a $mood mood.”;

}

?>

**If..Else Statement:**

Syntax:if (*expression*) {

// True block statement(s)

} else {

// False-block statement(s)

}

If the expression is true, the true block statement(s) are executed.

Otherwise, the false-block statement(s) are executed.

## The ElseIf Statement

Syntax: if (*expression*) {

// statement-1

} elseif (*another expression*) {

// statement-2

} else {

## Example:

## <?php

$mood = “sad”;

if ($mood == “happy”) {

echo “Hooray! I’m in a good mood!”;

} elseif ($mood == “sad”) {

echo “Awww. Don’t be down!”;

} else {

echo “I’m neither happy nor sad, but $mood.”;

}

## ?>

// default-statement

## }

## The expressions are evaluated from top downwards.

## As soon as the true condition is found, the statement

## associated with it is executed. Otherwise, the final else

## statement will be executed.

## The switch statement:

**Syntax:**switch (*expression*) {

case*case1*:

// execute this if expression results in result1

break;

casecase*2*:

// execute this if expression results in result2

break;

default:

// execute this if no break statement

// has been encountered hitherto

## }

## The switch statement tests the value of a given expression against a list of case values and when a match is found, a block of statements associated with that case is executed.

## Example:

<?php

$x = “sad”;

switch ($x) {

case “happy”:

echo “Hooray! I’m in a good mood!”;

break;

case “sad”:

echo “Awww. Don’t be down!”;

break;

default:

echo “I’m neither happy nor sad, but $mood.”;

break;

}

## ?>

## LOOPS

## The process of repeatedly executing a block of statements is known as Looping.

## Looping statements

## 1. While Loop:

## Syntax:

**while**(condition){

//code to be executed

}

## The control loops through a block of code if and as long as a specified condition is true.

## Example:

## <?php

## $x=1;

## $sum=0;

## while($x<10)

## {

## $sum=$sum+$x;

## $x++;

## }

## print " The sum of natural number is ".$sum;

## ?>

## 2. Do..While:

## Syntax:

**do**{

//code to be executed

}**while**(condition);

 Loops through a block of code once, and then repeats the loop as long as a special condition is true.

**Example:**

<?php

$x=1;

$sum=0;

do

{

$sum=$sum+$x;

$x++;

}

while($x<10);

print " The sum of natural number is ".$sum;

?>

## 3. For statement:

## Syntax:

**for**(initialization; condition; increment/decrement){

//code to be executed

}

Execution of the **for** statement is as follows

1. Initialization of the control variable is done first.
2. The value of the control variable is tested using the test condition. When the loop is true, the body of the loop is executed. Otherwise the loop is terminated.
3. When the body of the loop is executed, the control is transferred back to the**for** statement after evaluating the last statement in the loop. Now the control variable is incremented or decremented. The new value of the control variable is tested again to see whether it satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

**Example:**

<?php

$x;

$sum=0;

for($x=1;$x<10;$x++)

{

$sum=$sum+$x;

}

print " The sum of natural number is ".$sum;

?>

**Difference b/w while and do-while:**

|  |  |
| --- | --- |
| **While** | **Do -while** |
| It is an entry-control loop | it is an exit-controlled loop |
| Test condition is tested first | Test condition is tested after executing the body of the loop |
| It is pre-testing loop | It is post-testing loop |
| It will not give guarantee to execute the body of the loop minimum once | It will give the guarantee to execute the body of the loop minimum once |
| Syntax: while (condition)  Simple or compound statement; | Syntax: do  Simple (or) compound statement;  while(condition); |

**BREAK STATEMENT:**

The break statement is used to terminate the loop. When the **break**statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

**Syntax: break;**

**Example:**

<?php

$i;$sum=0;$n=10;

for($i=1;$i<=$n;$i++)

{

if($i==6)

break;

$sum=$sum+$i;

}

print " The sum is " .$sum;

?>

**CONTINUE STATEMENT:**

The continue statement tells the compiler. “SKIP THE FOLLOWING STATEMENTS AND CONTIUNE WITH THE NEXT ITERATION”. The format of the continue statement is simply

**Syntax: continue;**

**Example:**

<?php

$i;$sum=0;$n=10;

for($i=1;$i<=$n;$i++)

{

if($i==6)

continue;

$sum=$sum+$i;

}

print " The sum is " .$sum;

?>

**Nested Loops:**

Nesting of loops, that is, one **for** statement within another **for**statement.

**Example:**

<?php

$row;

$col;

echo "<table border='1'>";

for($row=1;$row<=10;$row++)

{

echo "<tr>";

for($col=1;$col<=10;$col++)

{

echo "<td>".($row\*$col);

echo "</td>";

}

echo "</tr>";

}

echo "</table>";

?>

**CODE BLOCKS AND BROWSER OUTPUT**

Imagine a script that outputs a table of values only when a variable is set to the Boolean value true.A simplified HTML table constructed with the code block of an ifstatement.

##### **A Code Block Containing Multiple print() Statements**

<?php

$display\_prices = true;

if ( $display\_prices ) {

print "<table border=\"1\">";

print "<tr><td colspan=\"3\">";

print "today's prices in dollars";

print "</td></tr>";

print "<tr><td>14</td><td>32</td><td>71</td></tr>";

print "</table>";

}

?>

We can return to HTML mode within a code block. By this we can save ourselves some typing. We avoided using print statement.

<?php

$display\_prices = true;

if ( $display\_prices ) {

?>

<table border="1">

<tr><td colspan="3">today's prices in dollars</td></tr>

<tr><td>14</td><td>32</td><td>71</td>

</table>

<?php

}

?>

**WORKING WITH FUNCTIONS:-**

**Function:**A functions is a block of code to perform task. Functions are different types like built-in functions and user defined functions. We have hundreds of built-in functions in PHP.

User defined functions are functions which are defined by user.

By using functions we can easily understand the program easy, and we can test and debug the program easy.

**Syntax:**functionfunction\_name(arguments/parameterlist)

**Example for built-in function**

**Example for user defined function**

<?php

function test($x)

{

echo $x;

}

test("hello world");

?>

<?php

$num = -321;

$newnum = abs($num);

echo $newnum;

?>

**Function Definition (or) called function:**

**Syntax:**

functionfunction\_name(parameter-list)

{

Local variable declarations;

Executable statement1;

Executable statement2; function body

…………..

Return statement;

}

It contains the actual coding of the function.

**Function\_name** specifies the name of the function.

**Parameter-list**declares variables that will receive the data sent by calling program.

Function body contains the declarations and statements necessary for performing the required task. The body is enclosed with curly braces { }.

**Function Call or calling function:**

A function can be called by simply using the function name followed by a list of actual parameters, if any, enclosed in parentheses.

Whenever a function is called, the execution control is transferred to that function. After completion of function execution, the control returns to the calling point.

The general form of calling a function is as follows:

function\_name(argument-list);

**Example;**

sum(10,29);

square(2);

**Return Statement:(Return values and their Types)**

The return statement is used to return a value from a user-defined function to its calling point. The statement can be used in the following ways.

**Syntax-1**: return;

**Syntax-2**: return (value);

**Examples:**

1.return;

2. return (25);

3. return (x+y);

4. return x+y;

There are two types of parameters. They are

1. Actual Parameters and
2. Formal parameters.

**Actual Parameters:** The parameters that are included in function calling point are called “actual parameters”. These are used to send vales to the called function.

**Formal Parameters:** The parameters that are included in function definition are called “formal parameters”. These are used to receive values from the calling point.

While using actual and formal parameters the following rules must be followed:

1. The number of actual parameters must be equal to the number of formal parameters.

**VARIABLE SCOPE**

In PHP, variables can be declared anywhere in the script.

PHP has three different variable scopes:

* local
* global
* static

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function.

<?php  
$x = 5; // global scope  
function myTest() {  
    // using x inside this function will generate an error  
    echo "<p>Variable x inside function is: $x</p>";  
}   
myTest();  
echo "<p>Variable x outside function is: $x</p>";  
?>

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function.

<?php  
function myTest() {  
    $x = 5; // local scope  
    echo "<p>Variable x inside function is: $x</p>";  
}   
myTest();  
// using x outside the function will generate an error  
echo "<p>Variable x outside function is: $x</p>";  
?>

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

<?php  
$x = 5;  
$y = 10;  
function myTest() {  
    global $x, $y;  
    $y = $x + $y;  
}  
myTest();  
echo $y; // outputs 15  
?>

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the **static** keyword when you first declare the variable:

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

**Example:**

**CALL-BY-VALUE:**

In this method, actual arguments of function call are passed to the formal parameters of the called function. Inside the function, the value of formal parameters changes according to the need. When the control is returned to its calling point, the changed values of formal parameters are not transferred. This type of passing technique is called call-by-value mechanism.

**(Or)**

In case of PHP call by value, actual value is not modified if it is modified inside the function.

**<?php**

**functionaddFive($num)**

**{**

**$num += 5;**

**}**

**$orignum = 10;**

**addFive($orignum);**

**echo $orignum;**

**?>**

<?php

function text($x)

{

echo $x;

}

functionmul($x,$y)

{

return ($x\*$y);

}

function add()

{

$x=10;

$y=20;

return ($x+$y);

}

function test()

{

$x=10;

echo "hello"."<br>";

echo $x;

}

text(" ashwikareddy");

echo "<br>";

$a=mul(13,2);

echo $a."<br>";

$b=add();

echo $b."<br>";

test();

?>

**CALL-BY-REFERENCE**

In case of PHP call by reference, actual value is modified if it is modified inside the function. In such case, you need to use & (ampersand) symbol with formal arguments. The & represents reference of the variable.

**<?php**

**functionaddFive(&$num)**

**{**

**$num += 5;**

**}**

**$orignum = 10;**

**addFive($orignum);**

**echo $orignum;**

**?>**

ARRAYS

An array can store one or more values in a single variable name.

Each element in the array has its own ID so that it can be easily accessed.

In PHP there are three types of arrays:

1. **Indexed array**- Arrays with numeric index
2. **Associative array**- Arrays with named keys
3. **Multidimensional array**- Arrays containing one or more arrays

**Indexed/Numeric array:** An indexed array stores each element with a numeric ID key.

There are different ways to create indexed array.

The index can be assigned automatically (index starts at 0).

**Example:**

<?php

$season=**array**("summer","winter","spring","autumn");

echo "Season are: $season[0], $season[1], $season[2] and $season[3]";

?>

The index can be assigned manually

**Example:**

<?php

$season[0]="summer";

$season[1]="winter";

$season[2]="spring";

$season[3]="autumn";

echo "Season are: $season[0], $season[1], $season[2] and $season[3]";

?>

**Associative array:** Associative arrays are arrays that use named keys where you assign value to them.

There are two ways to create an associative array:

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

**or:**

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

**Example:**

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
echo "Peter is " . $age['Peter'] . " years old.";  
?>

**Multidimensional Array:** Arrays containing one or more arrays.

<?php

$emp = **array**(

**array**(1,"sonoo",400000),

**array**(2,"john",500000),

**array**(3,"rahul",300000));

**for** ($row = 0; $row < 3; $row++) {

**for** ($col = 0; $col < 3; $col++) {

    echo $emp[$row][$col]."  ";

  }    echo "<br/>";  }

 ?>

**ARRAY FUNCTIONS**

1. **array():** Creates an array.

**$cars=array(“Bmw”,”logon”,”XUV”);**

1. **count() and sizeof():** Counts the number of elements in an array.

count($colors)

1. **array\_change\_key\_case():** Changes all keys in an array to lowercase or uppercase.
2. **array\_column():** Returns the value from a single column in the input array.
3. **array\_push():** This function adds one or more elements to the end of an existing array.

array\_push($existingArray, “element 1”, “element 2”, “element 3”);

1. **array\_pop():** This function removes (and returns) the last element of an existing array.

$last\_element = array\_pop($existingArray);

1. **array\_unshift():** This function adds one or more elements to the beginning of an existing arrays.

array\_unshift($existingArray, “element 1”, “element 2”, “element 3”);

1. **array\_shift:** This function removes (and returns) the first element of an existing array.

$first\_element = array\_shift($existingArray);

1. **array\_merge():** This function combines two or more existing arrays.

$newArray = array\_merge($array1, $array2);

1. **array\_keys():**Returns all the keys of an array

$x = array\_keys($existingArray);

1. **array\_values():** This function returns an array containing all the values within a given array.

$x = array\_values($existingArray);

**Example:**

<?php

$cars=array("A"=>" BMW","B"=>"XUV","C"=>"I20","D"=>"I10");

print\_r (array\_change\_key\_case($cars,CASE\_LOWER));

print " The number of elements are ".count($cars)."<br>";

array\_push($cars,"toyota","jaquar");

print\_r($cars);

$x=array\_pop($cars);

print\_r($x);

$months=array("jan","feb","march");

print "<br>";

array\_unshift($months,"april","may");

print\_r($months);

print "<br>";

array\_shift($months);

print\_r($months);

print "<br>";

$y=array\_keys($months);

print\_r($y);

print "<br>";

$z=array\_values($months);

print\_r($z);

print\_r(array\_merge($months,$cars));

?>

WORKING WITH OBJECTS:-

A class is the way to bind the data and its associated functions together.

**(OR)**

A Class can be defined as a template/blue print that describes the behaviours/states that object

of its type support.

**Syntax:**

class classname

{

[fields declaration]

[methods declaration]

}

**Objects:** Objects are basic run time entities in an object oriented programming. An object may be defined as an identifiable entity with some state and behaviour. We are often surrounded by a variety of objects. The computer we may be working on, the chair we sit on, a telephone, a clock, etc. are all objects.

The variables in the class are called data members and methods in the class are called member function.

Objects are created using the new operator. The new operator creates an object of the specified class and returns a reference to that object. Example

$rect1=new Rectangle(); //instantiate the object

We can access the class members i.e. instance variables and methods in the following way

Objectname->variablename=value;

Objectname->methodname(parameterlist);

**Example:**

<?php

class rectangle

{

public $length;

public $width;

function data($len,$wid)

{

$this->length=$len;

$this->width=$wid;

}

function area()

{

$area1=$this->length\*$this->width;

echo $area1;

}

}

$r=new rectangle();

$r->data(4,5);

$r->area();

?>

**OBJECT INHERITANCE**

Inheritance is a process of creating a new class from the existing class. The new class inherits all the capabilities of the existing class.

Example of how a child class can override the methods of the parent class.

<?php

class myClass

{

public $name = "sbvr";

function mylass($n)

{

$this->name = $n;

}

function sayHello()

{

echo " Hello my name is ".$this->name;

}

}

class childClass extends myClass {

function sayHello()

{

echo " Hello my name is sreenu";

}

}

$object1 =new childClass('ashwika');

$object1->mylass('college');

$object1 ->sayHello();

?>

**FORMATTING STRINGS WITH PHP**

PHP provides two functions printf( ) and sprintf( ) to format strings in many different ways.

**printf( ):** The printf() function outputs a formatted string.

**Syntax:**

printf(format string,arg1,arg2,arg++)

Format string is a special code, known as a conversion specification. A conversion specification begins with a percent (%) symbol.

**Example:**

<?php

printf(' This is my number: %o ', 55);

?>

In the above example conversion specification is followed by type specifiers.

**Type specifiers**

|  |  |
| --- | --- |
| **Specifier** | **Description** |
| d | Displays argument as a decimal number |
| b | Display an integer as a binary number |
| c | Display an integer as ASCII equivalent |
| f | Display an integer as a floating-point number (double) |
| o | Display an integer as an octal number (base 8) |
| s | Display argument as a string |
| x | Display an integer as a lowercase hexadecimal number (base 16) |
| X | Display an integer as an uppercase hexadecimal number (base 16) |

**Example:**

<?php

**Output:**

Decimal: 543  
Binary: 1000011111  
Double: 543.000000  
Octal: 1037  
String: 543  
Hex (lower): 21f  
Hex (upper): 21F

$number = 543;

printf('Decimal: %d<br/>', $number);

printf('Binary: %b<br/>', $number);

printf('Double: %f<br/>', $number);

printf('Octal: %o<br/>', $number);

printf('String: %s<br/>', $number);

printf('Hex (lower): %x<br/>', $number);

printf('Hex (upper): %X<br/>', $number);

?>

**Sign specifier:**

printf() only puts a sign symbol in front of negative numbers, not positive numbers:

<?php

printf( "%d", 36 );

printf( "%d", -36 );

?>

**Padding:**

To add padding, insert a **padding specifier** between the '%' character and the type specifier. A padding specifier takes the format:

<padding character><width>

<padding character> can be a zero or a space. If you miss it out, spaces are used. If you want to pad using a different character, write an apostrophe (') followed by the character to use.

<width> is the number of characters to pad the value out to. A positive number adds padding to the left; a negative number adds padding to the right.

**Example:**

<?php

**Output:**

0012   
1234   
12345   
Hello   
Hello   
\*\*\*\*\*Hello   
Hello\*\*\*\*

printf( "%04d </br>", 12 );

printf( "%04d </br>", 1234 );

printf( "%04d </br>", 12345 );

printf( "% 10s </br>", "Hello" );

printf( "%10s </br>", "Hello" );

printf( "%'\*10s </br>", "Hello" );

printf( "%'\*-10s </br>", "Hello" );

?>

**Specifying the width:**

You can specify the number of spaces within which your output should sit. A *field width specifier* is an integer that should be placed after the percent sign.

**Example:**

<?php

echo "<pre>";

printf("%20s\n", "Books");

printf("%20s\n", "CDs");

printf("%20s\n", "DVDs");

printf("%20s\n", "Games");

printf("%20s\n", "Magazines");

echo "</pre>";

?>

**Sprintf( )** is identical to printf, except that rather than directly outputting the result, it returns it so that you can store it in a variable.

**Example:**

<?php

$result = sprintf( "Andhra pradesh has %d districts", 13);

echo $result;

?>

**INVESTIGATING STRINGS IN PHP**

PHP provides many functions that enable you to acquire information about strings.

**strlen( ):** It is a built-in function to find the length of a string.

**strlen(string)**

**strstr( ):** This function searches for the first occurrence of a string inside another string.

**strstr(string,search,before\_search)**

**strpos( ):** This function  finds the position of the first occurrence of a string inside another string.

**strpos(string,find,start)**

**substr( ):** This function returns a part of a string.

**substr(string,start,length)**

**strrev( ):** This function reverses a string.

**strrev(string)**

**strtok( ):** This function splits a string into smaller strings (tokens).

**strtok(string,split)**

**Example for the above:**

<?php

$str="hi every body how are you";

echo strlen($str)."<br>";

echo strstr($str,'how')."<br>";

echo strpos($str,'how')."<br>";

echo substr($str,4,5)."<br>";

$str1=strtok($str," ")."<br>";

while ($str1 !== false)

{

echo "$str1<br>";

$str1 = strtok(" ");

}

echo strrev($str)."<br>";

?>

**MANIPULATING STRINGS WITH PHP**

**ltrim( ):** This function removes whitespace or other predefined characters from left side of a string.

**ltrim(string,charlist)**

**rtrim( ):** This function removes whitespace or other predefined characters from right side of a string.

**rtrim(string,charlist)**

**trim():** This function removes whitespace or predefined characters from both sides of a string.

**trim(string,charlist)**

**strip\_tags( ):** This function strips a string from HTML, XML, and PHP tags.

**substr\_replace( ):** This function replaces a part of a string with another string.

**substr\_replace(string,replacement,start,length)**

**str\_replace( ):** This function replaces some characters with some other characters in a string.

**str\_replace(find,replace,string,count)**

**Example:**

<?php

$str="Good morning every body";

echo ltrim($str,"Good")."<br>";

echo rtrim($str,"body")."<br>";

echo trim($str, "Gdy")."<br>";

echo strip\_tags(" Hello <b>world</b>")."<br>";

echo substr\_replace("hello world ","buddy",6,5)."<br>";

echo str\_replace("world","ashwika","Hello world!");

?>

**CONVERTING CASE:**

**strtolower( ):** This function converts a string to lower case.

**strtolower(string)**

**strtoupper( ):** This function converts a string to upper case.

**strtoupper(string)**

**ucwords( ):** This function converts the first character of each word in a string to uppercase.

**ucwords(string)**

**ucfirst( ):** This function converts the first character of a string to uppercase.

**ucfirst(string)**

**lcfirst( ):** This function converts the first character of a string to lowercase.

**lcfirst(string)**

**Example:**

<?php

echo strtolower(" RAHUL DRAVID")."<br>";

echo strtoupper(" rahul dravid")."<br>";

echo ucwords("good morning india ")."<br>";

echo ucfirst( "good morning ")."<br>";

echo lcfirst("Good morning ")."<br>";

?>

**Wrapping text:**

**nl2br( ):** This function inserts HTML line breaks (<br>) in front of each newline (\n) in a string.

**wordwrap( ):** This function wraps a string into new lines when it reaches a specific length.

**wordwrap(string,width,break,cut)**

**Breaking Arrays With Explode:**

**explode( ):** This function breaks a string into an array.

**explode(separator,string,limit)**

**Example:**

<?php  
$str = "Hello world. It's a beautiful day.";  
print\_r (explode(" ",$str));  
?>

**DATE AND TIME FUNCTIONS IN PHP**

**time( ):** This function gives you all the information about the current date and time. It returns the current time as Unix timestamp.

**Example:** echo time();

**getdate():** Returns date/time information of a timestamp or the current local date/time.

**Example:** print\_r(getdate());

Associative array returned by getdate()

* [seconds] - seconds
* [minutes] - minutes
* [hours] - hours
* [mday] - day of the month
* [wday] - day of the week
* [mon] - month
* [year] - year
* [yday] - day of the year
* [weekday] - name of the weekday
* [month] - name of the month
* [0] - seconds since Unix Epoch

**date( ):** Formats local date and time.

Specifies the format of the outputted date string. The following characters can be used:

* d - The day of the month (from 01 to 31)
* D - A textual representation of a day (three letters)
* j - The day of the month without leading zeros (1 to 31)
* l (lowercase 'L') - A full textual representation of a day
* N - The ISO-8601 numeric representation of a day (1 for Monday, 7 for Sunday)
* S - The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j)
* w - A numeric representation of the day (0 for Sunday, 6 for Saturday)
* z - The day of the year (from 0 through 365)
* W - The ISO-8601 week number of year (weeks starting on Monday)
* F - A full textual representation of a month (January through December)
* m - A numeric representation of a month (from 01 to 12)
* M - A short textual representation of a month (three letters)
* n - A numeric representation of a month, without leading zeros (1 to 12)
* t - The number of days in the given month
* L - Whether it's a leap year (1 if it is a leap year, 0 otherwise)
* o - The ISO-8601 year number
* Y - A four digit representation of a year
* y - A two digit representation of a year
* a - Lowercase am or pm
* A - Uppercase AM or PM
* B - Swatch Internet time (000 to 999)
* g - 12-hour format of an hour (1 to 12)
* G - 24-hour format of an hour (0 to 23)
* h - 12-hour format of an hour (01 to 12)
* H - 24-hour format of an hour (00 to 23)
* i - Minutes with leading zeros (00 to 59)
* s - Seconds, with leading zeros (00 to 59)
* u - Microseconds (added in PHP 5.2.2)
* e - The timezone identifier (Examples: UTC, GMT, Atlantic/Azores)
* I (capital i) - Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise)
* O - Difference to Greenwich time (GMT) in hours (Example: +0100)
* P - Difference to Greenwich time (GMT) in hours:minutes (added in PHP 5.1.3)
* T - Timezone abbreviations (Examples: EST, MDT)
* Z - Timezone offset in seconds. The offset for timezones west of UTC is negative (-43200 to 50400)
* c - The ISO-8601 date (e.g. 2013-05-05T16:34:42+00:00)
* r - The RFC 2822 formatted date (e.g. Fri, 12 Apr 2013 12:01:05 +0200)
* U - The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)

**Example:**

<?php

$ts = time();

echo date("m/d/y G:i:s e", $ts)."<br>";

?>

**mktime( ):** This function returns the unix timestamp for a date.

It accepts up to six integer arguments in the following order:

Hour

Minute

Second

Month

Day of month

Year

**Example:**

<?php

$ts = mktime(21, 34, 0, 1, 17, 2012);

echo date("m/d/y G:i:s e", $ts)."<br>";

echo date("M-d-y",mktime(1,45,2,4,5,1984))."<br>";

?>

**Checkdate( ):** This function is used to validate a Gregorian date.

**Example:**

<?php  
var\_dump(checkdate(12,31,-400));  
echo "<br>";  
var\_dump(checkdate(2,29,2003));  
echo "<br>";  
var\_dump(checkdate(2,29,2004));  
?>

**WORKING WITH FORMS**

Forms are used to get input from the user and submit it to the web server for processing.

**Form tag:**

The beginning and ending tag for form creation is <form> </form>.

The most important attributes of the form are “**Action**” and “**Method**”.

<form action=url method=”post/get”>

</form>

The **action** attribute defines the action to be performed when the form is submitted.

GET method: The default method when submitting form data is GET. Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases. GET may be used for sending non-sensitive data.

POST method: information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

**CREATING THE SIMPLE INPUT FORM:**

A simple form is created below with name test.html.

Example 1:

<html>

<body>

<form action="welcome1.php" method="post">

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

E-mail: <input type="text" name="email"><br>

<input type="submit">

</form>

</body>

</html>

**Reading input from a Form**

Save the file with name **welcome1.php**

<html>

<body>

Welcome <?php echo $\_POST["name"]; ?><br>

Your email address is: <?php echo $\_POST["email"]; ?>

</body>

</html>

**ACCESSING FORM INPUT WITH USER DEFINED ARRAYS:**

From the below example we can choose one or more items from the HTML file and display data on the PHP file.

Create a HTML file and name this as product.html

<html>

<head>

<title>An HTML form with checkboxes</title>

</head>

<body>

<form action="product.php" method="POST">

Select Some Products:<br/>

<input type="checkbox" id="cooler" name="product[]" value="COOLER">COOLER<br/>

<input type="checkbox" id="fridge" name="product[]" value="FRIDGE">FRIDGE<br/>

<input type="checkbox" id="tv" name="product[]" value="TELEVISION">TELEVISION</br>

<input type="submit" value="click">

</form>

</body>

</html>

Now create a PHP file with name product.php

<html>

<head>

<title>Reading checkboxes</title>

</head>

<body>

<p>Your product choices are:

<?php

if (!empty($\_POST['product'])) {

echo "<ul>";

foreach ($\_POST['product'] as $value) {

echo "<li>$value</li>";

}

echo "</ul>";

} else {

echo "None";

}

?>

</body>

</html>

Open the HTML file **product.html** in the web browser. Enter the data in the form and press submit button. On clicking submit button it redirects to **product.php,** displaying the data of the product.html

**COMBING HTML AND PHP CODE ON A SINGLE PAGE:**

**Example is given below:**

<?php

$num\_to\_guess = 42;

$message = "";

if ( ! isset( $\_POST['guess'] ) )

$message = "Welcome to the guessing machine!";

elseif ( $\_POST['guess'] > $num\_to\_guess )

$message = $\_POST['guess'] ."is too big! Try a smaller number";

elseif ( $\_POST['guess'] < $num\_to\_guess )

$message = $\_POST['guess']." is too small! Try a larger number";

else

$message = "Well done!";

?>

<html>

<head>

<title>Listing 9.9 A PHP number guessing script</title>

</head>

<body>

<h1>

<?php print $message ?>

</h1>

<form method="POST">

Type your guess here: <input type="text" name="guess">

<input type="submit">

</form>

</body>

</html>

**Explain the Hidden fields to save state?**

We can use hidden fields to keep track of the values. A hidden field behaves the same as a text field; expect that the user cannot see it unless he views the html source of the document that contains it.

**Program:**

<?php

$num\_to\_guess = 42;

$num\_tries = (isset($\_POST['num\_tries'])) ? $\_POST['num\_tries'] + 1 : 1;

if (!isset($\_POST['guess'])) {

$message = "Welcome to the guessing machine!";

} elseif (!is\_numeric($\_POST['guess'])) {

$message = "I don’t understand that response.";

} elseif ($\_POST['guess'] == $num\_to\_guess) {

$message = "Well done!";

} elseif ($\_POST['guess'] > $num\_to\_guess) {

$message = $\_POST['guess']." is too big! Try a smaller number.";

} elseif ($\_POST['guess'] < $num\_to\_guess) {

$message = $\_POST['guess']." is too small! Try a larger number.";

} else {

$message = "I am terribly confused.";

}

?>

<html>

<head>

<title>A PHP number guessing script</title>

</head>

<body>

<h1><?php echo $message; ?></h1>

<p><strong>Guess number:</strong> <?php echo $num\_tries; ?></p>

<form method="POST">

<p><label>Type your guess here:</label><br/>

<input type="text" id="guess" name="guess" /></p>

<input type="hidden" name="num\_tries" value="<?php echo $num\_tries; ?>"/>

<input type="submit">

</form>

</body>

</html>

**Explain the Redirecting the user?**

We use header( ) function to send a new HTTP header but this must be sent to the browser.

When a server script communicates with a client, it must first send some headers that provide information about the document to follow. PHP usually handles this for you automatically, but you can choose to send your own header lines with PHP’s header() function.

By sending the Location header instead of PHP’s default header, we can change the browser to be redirected to a new page.

**Ex: header(“Location:congrats.html”);**

<?php

$num\_to\_guess = 42;

$num\_tries = (isset($\_POST['num\_tries'])) ? $\_POST['num\_tries'] + 1 : 1;

if (!isset($\_POST['guess'])) {

$message = "Welcome to the guessing machine!";

} elseif ($\_POST['guess'] == $num\_to\_guess) {

$message = "Well done!";

header ("Location:congrats.html");

} elseif ($\_POST['guess'] > $num\_to\_guess) {

$message = $\_POST['guess']." is too big! Try a smaller number.";

} elseif ($\_POST['guess'] < $num\_to\_guess) {

$message = $\_POST['guess']." is too small! Try a larger number.";

} else {

$message = "I am terribly confused.";

}

?>

<html>

<head>

<title>Using header() to redirect user</title>

</head>

<body>

<h1><?php echo $message; ?></h1>

<p><strong>Guess number:</strong> <?php echo $num\_tries; ?></p>

<form method="POST">

<p><label for="guess">Type your guess here:</label><br/>

<input type="text" id="guess" name="guess" /></p>

<input type="hidden" name="num\_tries" value="<?php echo $num\_tries; ?>"/>

<button type="submit" name="submit" value="submit">Submit</button>

</form>

</body>

</html>

**Explain the sending mail on form submission?**

Before we can use the mail () function to send mail a few directories must be set up in the php.ini file so that the function works properly. Open php.ini with a text editor and make some changes in the file as below

Set

SMTP = smtp.yourisp.net

sendmail\_from = youraddress@yourdomain.com

Through these forms user is able to send his/her suggestion or feedback via email to respective organization.

We used following PHP **mail()** function with four parameters to send email as follows:

mail ("$to", $subject, $message, $headers);

* Here, **$to** variable is to store receiver’s email id.
* **$subject** is a variable to store mail subject.
* **$message** is a variable to store user’s message.
* **$headers** contain other email parameters like **BCc, Cc** etc.

<html>

<form action="send.php" method="post">

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

Request:<br><textarea name="request" ></textarea><br>

<input type="submit">

</form>

</html>

<?php

$name=$\_POST['name'];

$request=$\_POST['request'];

$to="sri999111@gmail.com";

$subject="Tutorial request ";

$body="This is an automated message. Plese don't reply to this mail \n\n $request";

if(mail($to,$subject,$body))

{

echo 'message sent successfully';

}

else

{

echo 'not ok';

}

?>

**WORKING WITH FILE UPLOADS**

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 about the uploaded file becomes available to you in the $\_FILES superglobal, which is indexed by the name of the upload field (or fields) in the form.

PHP would create following variables:

1. $\_FILES [“file”] [“tmp\_name”] The uploaded file in the temporary directory on the web server.
2. $\_FILES [“file”] [“name”] The original name of the uploaded file.
3. $\_FILES [“file”] [“size”] The size in bytes of the uploaded file.
4. $\_FILES [“file”] [“type”] The MIME type of the uploaded file.

**Create file upload form:**

**Name the file as upload.html**

<html>

<head>

<title>A simple file upload form</title>

</head>

<body>

<form action="do\_upload.php" enctype="multipart/form-data" method=”POST”>

<input type="hidden" name="MAX\_FILE\_SIZE" value="1048576" />

<p><label for=”fileupload”>File to Upload:</label>

<input type="file" id="fileupload" name="fileupload" /></p>

<button type=”submit” name=”submit” value=”send”>Upload File</button>

</form>

</body>

</html>

**Creating the file upload script:**

The uploaded file information is available in the superglobal variable $\_FILES. We first create the $file\_dir variable to store path information. Make sure that this path must exist on your system.

The foreach loop is used to store the upload file’s name in the $file\_name variable and the file information in the $file\_array variable.

**Program:**

<?php

$file\_dir = "c:\users\cnu\";

foreach($\_FILES as $file\_name => $file\_array) {

echo "path: ".$file\_array['tmp\_name']."<br/>\n";

echo "name: ".$file\_array['name']."<br/>\n";

echo "type: ".$file\_array['type']."<br/>\n";

echo "size: ".$file\_array['size']."<br/>\n";

if (is\_uploaded\_file($file\_array['tmp\_name'])) {

move\_uploaded\_file($file\_array['tmp\_name'],

"$file\_dir/".$file\_array['name'])

or die ("Couldn’t move file");

echo "File was moved!";

} else {

echo "No file found.";

}

}

?>

**WORKING WITH FILES AND DIRECTORIES**

**Including files:**

The include statement enables you to include (usually other PHP scripts) other files into your PHP documents. PHP code in these included files will then be executed as if it were part of the main document.

**Syntax:**

include ‘filename’

**or**

require ‘filename’

**Example:**

Assume we have a file called "vars.php", with some variables defined:

<?php  
$color='red';  
$car='BMW';  
?>

Then, if we include the "vars.php" file, the variables can be used in the calling file:

<html>  
<body>  
<h1>Welcome to my home page!</h1>  
<?php include 'vars.php';  
echo "I have a $color $car.";  
?>  
</body>  
</html>

**Validating files:**

Before you work with a file or directory within your code, it is often a good idea to learn more about it, and determining whether it actually exists. PHP provides many functions to know about files on your system.

**file\_exists():** This function is used to check whether a file exist or not.

if (file\_exists(‘test.txt’))

{

echo “The file exists!”;

}

If the file is found this function returns true; otherwise, it returns false.

**is\_file( ):** This function is used to test whether the specified file is a regular file.

**Syntax:**is\_file(file)

**is\_dir():** This function checks whether the specified file is a directory.

**Syntax:** is\_dir(file)

**is\_readable( ):** This function checks whether the specified file is readable. This function returns true if the file is readable.

**Syntax:** is\_readable(file)

**is\_writable( ):** This function checks whether the specified file is writeable. This function returns true if the file is writeable.

**Syntax:** is\_writable(file)

**is\_executable( ):** This function checks whether the specified file is executable. This function returns true if the file is executable.

**Syntax:** is\_executable(file)

**filesize():** This function returns the size of the specified file. This function returns the file size in bytes on success or FALSE on failure.

**Syntax:** filesize (filename)

**fileatime( ):** This function returns the last access time of the specified file.

**Syntax:**fileatime(filename)

**filemtime():** This function return the last time the file content was modified. This function returns the last change time as Unix timestamp on success, FALSE on failure.

**Syntax:** filemtime(filename)

**filectime( ):** This function returns the last time the specified file was changed.

**Syntax:** filectime(filename)

**Example:**

<?php

function output($f) {

if (!file\_exists($f)) {

echo "<p>$f does not exist</p>";

return;

}

echo "<p>$f is ".(is\_file($f) ? "" : "not ")."a file</p>";

echo "<p>$f is ".(is\_dir($f) ? "" : "not ")."a directory</p>";

echo "<p>$f is ".(is\_readable($f) ? "": "not ")."readable</p>";

echo "<p>$f is ".(is\_writable($f) ? "": "not ")."writable</p>";

echo "<p>$f is ".(is\_executable($f) ? "": "not ")."executable</p>";

echo "<p>$f is ".(filesize($f))." bytes</p>";

echo "<p>$f was accessed on ".date( "D d M Y g:i A",fileatime($f))."</p>";

echo "<p>$f was modified on ".date( "D d M Y g:i A",filemtime($f))."</p>";

echo "<p>$f was changed on ".date( "D d M Y g:i A",filectime($f) )."</p>";

}

$file = "d:\s.txt";

output($file);

?>

**CREATING AND DELETING FILE**

If a file does not yet exist, you can create it with the **touch()** function. If the file already exists, its contents is not disturbed, but the modification date is updated to reflect the time at which the function executed:

**Syntax:** touch(‘myfile.txt’);

You can remove an existing file with the **unlink(). F**unction unlink() accepts a file path:

**Syntax:** unlink(‘myfile.txt’);

**OPENING A FILE FOR WRITING, READING, OR APPENDING**

**fopen( )** function which is used to open a file. The file can be opened in different modes like below:

|  |  |
| --- | --- |
| r | Open a file for read only |
| w | Open a file for write only |
| a | Open a file for write only |
| r+ | Open a file for read/write |
| w+ | Open a file for read/write |
| a+ | Open a file for read/write |

**fopen(“filename”, “mode”)**

To open a file for reading, you use the following:

$fp = fopen(“test.txt”, “r”);

You use the following to open a file for writing:

$fp = fopen(“test.txt”, “w”);

To open a file for *appending* (that is, to add data to the end of a file), you use this:

$fp = fopen(“test.txt”, “a”);

**fclose ():** This function is used to close the file.

**Example:**

<?php  
$myfile = fopen("one.txt", "r");  
fclose($myfile);  
?>

**READING FROM FILES**

PHP provides a number of functions for reading data from files. These functions enable you to read by the byte, by the whole line, and even by the single character.

**fgets( ):** This function is used to read a single line from a file.

**Example:**

<?php  
$myfile = fopen("e:\one.txt", "r") or die("Unable to open file!");  
echo fgets($myfile);  
fclose($myfile);

?>

**feof():** This function checks if the “end-of-file” (EOF) has been reached.

**Example:**

<?php  
$myfile = fopen("e:\.txt", "r") or die("Unable to open file!");  
while(!feof($myfile)) {  
  echo fgets($myfile) . "<br>";  
}  
fclose($myfile);  
?>

**fread( ):** This function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "one.txt" file to the end:

**fread($myfile,filesize("one.txt"));**

<?php

$filename = “test.txt”;

$fp = fopen($filename, “r”) or die(“Couldn’t open $filename”);

while (!feof($fp)) {

$chunk = fread($fp, 8);

echo $chunk.”<br/>”;

} ?>

**fgetc( ):** This function is used to read a single character from a file.

**Example:**

<?php  
$myfile = fopen("e:\one.txt", "r") or die("Unable to open file!");  
// Output one character until end-of-file  
while(!feof($myfile)) {  
  echo fgetc($myfile);  
}  
fclose($myfile);  
?>

**fseek ():** This function is used to seek in an open file.

This function moves the file pointer from its current position to a new position, forward or backward, specified by the number of bytes.

This function returns 0 on success, or -1 on failure. Seeking past EOF will not generate an error.

**Syntax:** fseek(file,offset,whence)

**file\_get\_contents():** This function read a file into a string.

**Example:**

<?php  
echo file\_get\_contents("d:\s.txt");  
?>

**WRITING OR APPENDING TO A FILE**

Writing to a file with fwrite() or fputs() functions

**fwrite():** This function writes to an open file.

<?php  
$file = fopen(“d:\s.txt","w");  
echo fwrite($file,"Hello World. Testing!");  
fclose($file);  
?>

**fputs():** This function writes to an open file.

**Example:**

<?php  
$file = fopen("d:\s.txt","w");  
echo fputs($file,"Hello World. Testing!");  
fclose($file);  
?>

**file\_put\_contents():** This function writes a string to a file.

<?php  
echo file\_put\_contents("d:\s.txt",” hello world”);  
?>

**WORKING WITH DIRECTORIES**

**mkdir():** This function enables you to create a directory. This function returns TRUE on success, or FALSE on failure.

**Example:**

<?php

mkdir("d:\cnu");

?>

**rmdir():** This function removes an empty directory. This function returns TRUE on success, or FALSE on failure.

**Example:**

<?php  
$path = "d:\cnu";  
if(!rmdir($path))  
  {  
  echo ("Could not remove $path");  
  }  
?>

**opendir():** This function opens a directory handle.

**readdir():** This function returns the name of the next entry in a direction.

**Example:**

<?php

$dir = "d:\material\_c";

if (is\_dir($dir)){

if ($dh = opendir($dir)){

while (($file = readdir($dh)) !== false){

echo "filename:" . $file . "<br>";

}

closedir($dh);

}

}

?>

**OPENING PIPES TO AND FROM PROCESSES**

**Using popen()**

PHP provides popen( ) function to open a pipe to a process.

**Syntax: open(“command”,model);**

The *mode* is either r (read) or w (write).

**Example:**

<?php

$file\_handle = popen("text.txt", "r");

$read = fread($file\_handle, 2096);

echo $read;

pclose($file\_handle);

?>

When you execute the above program, it opens test.txt file for reading and displays the contents of a file in the browser. Any error message stored in the $handler pointer is read and printed to the screen. Finally closes the file pointer.

**RUNNING COMMANDS WITH EXEC()**

PHP provides popen( ) function to open a pipe to a process.

**Syntax:exec(“command”,$output\_array,$return\_valmode);**

Where command is that will be executed.

$output\_array if the output argument is present, then the specified array will be filled with every line of output from the command.

**Example:**

<?php

exec("ls -al .", $output\_array, $return\_val);

echo "Returned ".$return\_val."<br/><pre>";

foreach ($output\_array as $o) {

echo $o."\n";

}

echo "</pre>";

?>

**RUNNING COMMANDS WITH SYSTEM( ) OR PASSTHRU( ) FUNCTION**

1. **system( ):** PHP provides the system( ) to execute an external program and display the output to the browser. The system( ) executes the given command and outputs the results.

**Syntax:** system(“command”,$return\_var);

2. **passthru( ):** PHP provides the passthru( ) to execute an external program and displays raw output.

**Syntax:**passthru(“command”,&$return\_var);