PHP FUNCTIONS

USER-DEFINED FUNCTIONS

We can declare and call user-defined functions easily

Syntax

```
function functionname(arguments){
//code to be executed
Return;
}
```

Example:

```
<?php
function sayHello(){
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

FUNCTION ARGUMENTS

We can pass the information in PHP function through arguments which is separated by comma. PHP supports **Call by Value** (default), **Call by Reference**, **Default argument values** and **Variable-length argument list**.

CALL BY VALUE:

```
<?php
function sayHello($name){
echo "Hello $name<br/>";
}
$name=’Chirag’;
sayHello("Sonoo");
?>
```

CALL BY REFERENCE:

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

```
<?php
function adder(&$str2)
{
   $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

DEFAULT ARGUMENT VALUE

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument

```
<?php
function sayHello($name="Sonoo"){
echo "Hello $name<br/>";
```

```
}
sayHello("Rajesh");
sayHello();//passing no value
sayHello("John");
?>
```

RETURNING VALUE

```
<?php
function cube($n){
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

we have passed two parameters **$x** and **$y** inside two functions **add()** and **sub()**.

```
<?php
//add() function with two parameter
function add($x,$y)
{
$sum=$x+$y;
echo "Sum = $sum <br><br>";
}
//sub() function with two parameter
function sub($x,$y)
{
$sub=$x-$y;
echo "Diff = $sub <br><br>";
}
//call function, get  two argument through input box and click on add or sub button
if(isset($_POST['add']))
{
//call add() function
 add($_POST['first'],$_POST['second']);
}
if(isset($_POST['sub']))
{
//call add() function
sub($_POST['first'],$_POST['second']);
}
?>
<form method="post">
Enter first number: <input type="number" name="first"/><br><br>
Enter second number: <input type="number" name="second"/><br><br>
<input type="submit" name="add" value="ADDITION"/>
<input type="submit" name="sub" value="SUBTRACTION"/>
</form>
```

IN-BUILD FUNCTIONS

MATH FUNCTIONS

Function	Description	Example & output	Output
abs()	This function is used to return the absolute (positive) value of a number	abs(-3.5) abs(5) abs(-5)	3.5 5 5
acos()	This function is used to return the arc cosine of a number. value in the range of -1 to +1,otherwise it return NaN Note: same functions asin(),atan(),atan2()	acos(-0.35) acos(5)	1.92 NAN
ceil()	This function is used to round a number up to the nearest integer	ceil(3.35) ceil(-4.35) ceil(14.81)	4 -4 15
cos()	This function is used to return the cosine of a number Note:Same function sin(),tan()	cos(0)	1
floor()	This function is generally used to round a number down to the nearest integer	floor(3.35) floor(-2.35) floor(14.81)	3 -3 14
log()	This function is basically used to return the natural logarithm of a number	log(2)	0.6931
log10()	This function is basically used to return the base-10 logarithm of a number	log10(455)	2.6580
max()	This function is basically used to return the highest value in an array or it can be said that the highest value of various specified values	max(2,4,6,8,10)	10
min()	This function is basically used to return the lowest value in an array or it can be said that the lowest value of various specified values	min(2,4,6,8,10)	2
pi()	This function is simply used to return the value of the PI	Pi()	3.14
pow()	This function is simply used to return x raised to the power of y	pow(2,4)	16
rand()	It is the function that is used in order to generate a random integer	Rand() Rand(10,20)	1132363175 13
round()	This function is generally used to round a floating-point number	round(3.35) round(-2.35)	3 -2
sqrt()	This function is simply used to return the square root of a number	sqrt(9)	3

STRING FUNCTION

Function	Description	Example	Output
chop()	Deletes the whitespace or other characters present in the string from the end of a string	chop(“Hello World”,"World!")	Hello
chr()	Used to get the specific character value of a ascii value.	chr(65)	A

echo()	This function is used to print one or more string as the output	echo("Hello world!");	Hello world!
implode()	Converts the elements of an array into a string.	$arr = array('Hello','World!','Beautiful','Day!'); echo implode(" ",$arr);	Hello World! Beautiful Day!
join()	Alias of implode()	$arr = array('Hello','World!','Beautiful','Day!'); echo join(" ",$arr);	Hello World! Beautiful Day!
lcfirst()	This function Changes the first character of the given string in lowercase	lcfirst("Hello world!")	hello world
ltrim()	This function eliminates the whitespaces or other characters from the left side of the given string	$str = "Hello World!"; echo $str . " "; echo ltrim($str,"Hello");	Hello World! World!
ord()	This function gives the ASCII value of the first character of any given string	ord("A")	65
print()	This function print the Output as one or more strings	print "Hello world!";	Hello world!
printf()	This function is used to print the Output as a formatted string Note: format specifier are according to c language	$number = 9; $str = "Beijing"; printf("There are %d million bicycles in %s.",$number,$str);	There are 9 million bicycles in Beijing.
rtrim()	This function eliminates whitespaces or other characters from the right side of the given string	echo $str; echo rtrim($str,"World!");	Hello World! Hello
similar_text()	This function is used to check the similarity between the two strings	similar_text("Hello World","Hello Peter")	7
str_ireplace()	This function is used to Replace some characters of a string	str_ireplace("WORLD","Peter","Hello good world!")	Hello good Peter!
str_pad()	This function is used for the Padding of a string to a new length	str_pad($str,20,".")	Hello World.....
str_repeat()	Repeats a string as many number of times you want	str_repeat("Wow",3)	WowWow Wow
str_replace()	Replaces parts of a string	str_replace("world","Peter", "Hello world!")	Hello Peter!
str_shuffle()	This function Randomly shuffles all characters of the given string	str_shuffle("Hello World")	lWl eodrloH (randomly shuffle)
str_split()	str_split() function splits(break) a string into an array.	print_r(str_split("Hello"))	Array ([0] => H [1] => e [2] => l [3] => l [4]

			=> o)
str_word_count()	Calculates the number of words in a string	str_word_count("Hello world!")	2
strcasecmp()	Compares two strings	strcasecmp("Hello world!","HELLO WORLD!")	0 (not match)
strchr()	This function Finds the very first presence of a string inside another string.	strchr("Hello world!","world");	world
strcmp()	Compares two strings(case-sensitive) 0 - if the two strings are equal <0 - if string1 is less than string2 >0 - if string1 is greater than string2	strcmp("Hello world!","Hello world!")	0
strcspn()	Counts the number of characters found in a string	strcspn("Hello world!","w")	6
stripos()	This function gives the position of the first presence of a string inside another string	stripos("I love php, I love php too!","PHP")	7
strlen()	Calculates the number of characters in a string	strlen("Hello")	5
strpos()	Gives the position of the first presence of a string inside any other string(case-sensitive)	strpos("I love php, I love php too!","php")	7
strrev()	Retrun reverse of a given string	strrev("Hello World!")	!dlroW olleH
strripos()	Locates the position of the last presence of a string inside another string(case-insensitive)	strripos("I love php, I love php too!","PHP")	19
strrpos()	Locates the position of the last presence of a string inside another string(case-sensitive)	strrpos("I love php, I love php too!","php")	19
strtolower()	Converts in Lowercases a string	strtolower("Hello WORLD.")	hello world.
strtoupper()	Converts in Uppercases a string	strtoupper("Hello WORLD!");	HELLO WORLD.
substr()	Retrieves a section of a string	substr("Hello world",6,6) substr("Hello world",-4) substr("Hello world",4)	world orld o world
trim()	Removes leading and trailing whitespaces from a string	$str = "Hello World!"; echo trim($str,"Hed!");	llo Worl
ucfirst()	Converts in uppercase the first character of a string	ucfirst("hello world!")	Hello world!
ucwords()	Converts in uppercases the first character of every word of a string	ucwords("hello world");	Hello World
Print_r()	prints the information about a	$a	Array ([0]

	variable in a more human-readable way.	= array("red", "green", "blue"); print_r($a);	=> red [1] => green [2] => blue)

DATE AND TIME FUNCTIONS

Function	Description	Example	Output
date_create()	Returns a new DateTime object	$date=date_create("2013-03-15"); echo date_format($date,"Y/m/d");	2013/03/15
date_default_timezone_get()	Returns the default timezone used by all date/time functions	date_default_timezone_get();	UTC
date_default_timezone_set()	Sets the default timezone used by all date/time functions	date_default_timezone_set("Asia/India"); echo date_default_timezone_get();	UTC
date_diff()	Returns the difference between two dates	$date1=date_create("2013-03-15"); $date2=date_create("2013-12-12"); $diff=date_diff($date1,$date2);	+272 days
date_format()	Returns a date formatted according to a specified format	$date=date_create("2013-03-15"); echo date_format($date,"Y/m/d H:i:s");	2013/03/15 00:00:00

date_time_set()	Sets the time	$date=date_create("2013-05-01");
		date_time_set($date,13,24);
		echo date_format($date,"Y-m-d H:i:s");
			2013-05-01 13:24:00

date()	Formats a local date and time Note check table 1 for format	echo date("l") . " ";	Thursday
getdate()	Returns date/time information of a timestamp or the current local date/time	print_r(getdate());	Array ([seconds] => 44 [minutes] => 39 [hours] => 6 [mday] => 23 [wday] => 4 [mon] => 6 [year] => 2022 [yday] => 173 [weekday] => Thursday [month] =>

			June [0] => 1655966384)
time()	Returns the current time as a Unix timestamp	$t=time(); echo(date("Y-m-d",$t));	2022-06-23

Table 1

- 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)

ARRAY FUNCTIONS(INCLUDE SORING ARRAY FUNCTIONS ALSO)

Function	Description	Example	Output
array()	Creates an array	$cars=array("Volvo","BMW","Toyota") $age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43"); $cars=array (array("Volvo",100,96), array("BMW",60,59), array("Toyota",110,100))	
array_change_key_case()	Changes all keys in an array to lowercase or uppercase	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43"); print_r(array_change_key_case($age,CASE_LOWER));	Array ([peter] => 35 [ben] => 37 [joe] => 43)
array_chunk()	Splits an array into chunks of arrays	$cars=array("Volvo","BMW","Toyota","Honda","Mercedes","Opel"); print_r(array_chunk($cars,2));	Array ([0] => Array ([0] => Volvo [1] => BMW) [1] => Array ([0] => Toyota [1] => Honda) [2] => Array ([0] => Mercedes [1] => Opel))
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array	$fname=array("Peter","Ben","Joe"); $age=array("35","37","43"); $c=array_combine($fname,$age); print_r($c);	Array ([Peter] => 35 [Ben] => 37 [Joe] => 43)
array_count_values()	Counts all the values of an array	$a=array("A","Cat","Dog","A","Dog"); print_r(array_count_values($a)); ;	Array ([A] => 2 [Cat] => 1 [Dog] => 2)
array_diff()	Compare arrays, and returns the differences (compare values only)	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("e"=>"red","f"=>"green","g"=>"blue"); $result=array_diff($a1,$a2); print_r($result);	Array ([d] => yellow)

array_diff_assoc()	Compare arrays, and returns the differences (compare keys and values)	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("a"=>"red","b"=>"green","c"=>"blue"); $result=array_diff_assoc($a1,$a2); print_r($result);	Array ([d] => yellow)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)	$a1=array("a"=>"red","b"=>"green","c"=>"blue"); $a2=array("a"=>"red","c"=>"green","d"=>"blue"); $result=array_diff_key($a1,$a2); print_r($result); ?>	Array ([b] => green [d] => blue)
array_fill()	Fills an array with values	$a1=array_fill(3,4,"blue"); print_r($a1);	Array ([3] => blue [4] => blue [5] => blue [6] => blue)
array_flip()	Flips/Exchanges all keys with their associated values in an array	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $result=array_flip($a1); print_r($result);	Array ([red] => a [green] => b [blue] => c [yellow] => d)
array_intersect()	Compare arrays, and returns the matches (compare values only)	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("e"=>"red","f"=>"green","g"=>"blue"); $result=array_intersect($a1,$a2); print_r($result);	Array ([a] => red [b] => green [c] => blue)

array_key_exists()	Checks if the specified key exists in the array	$a=array("Volvo"=>"XC90","BMW"=>"X5");
		if (array_key_exists("Volvo",$a))
		  {
		  echo "Key exists!";
		  }
		else
		  {
		  echo "Key does not exist!";
			Key exists!

		}	
array_merge()	Merges one or more arrays into one array	$a1=array("red","green"); $a2=array("blue","yellow"); print_r(array_merge($a1,$a2));	Array ([0] => red [1] => green [2] => blue [3] => yellow)

array_multisort()	Sorts multiple or multi-dimensiona l arrays	$a=array("Dog","Cat","Horse","Bear","Zebra");
		array_multisort($a);
		print_r($a);
			Array ([0] => Bear [1] => Cat [2] => Dog [3] => Horse [4] => Zebra)

array_pad()	Inserts a specified number of items, with a specified value, to an array	$a=array("red","green"); print_r(array_pad($a,5,"blue"));	Array ([0] => red [1] => green [2] => blue [3] => blue [4] => blue)

array_pop()	Deletes the last element of an array	$a=array("red","green","blue");
		array_pop($a);
		print_r($a);
			Array ([0] => red [1] => green)

array_push()	Inserts one or more elements to the end of an array	$a=array("red","green");
		array_push($a,"blue","yellow");
		print_r($a);
			Array ([0] => red [1] => green [2] => blue [3] => yellow)

array_replace()	Replaces the values of the first array with the values from following arrays	$a1=array("red","green"); $a2=array("blue","yellow"); print_r(array_replace($a1,$a2));	Array ([0] => blue [1] => yellow)
array_reverse()	Returns an array in the reverse order	$a=array("a"=>"Volvo","b"=>"BMW","c"=>"Toyota"); print_r(array_reverse($a));	Array ([c] => Toyota [b] => BMW [a] => Volvo)
array_search()	Searches an array for a given value and returns the key	$a=array("a"=>"red","b"=>"green","c"=>"blue"); echo array_search("red",$a);	a
array_slice()	Returns selected parts of an array	$a=array("red","green","blue","yellow","brown"); print_r(array_slice($a,2));	Array ([0] => blue [1] => yellow [2] => brown)
array_sum()	Returns the sum of the	$a=array(5,15,25); echo array_sum($a);	45

	values in an array		
array_unique()	Removes duplicate values from an array	$a=array("a"=>"red","b"=>"green","c"=>"red"); print_r(array_unique($a));	Array ([a] => red [b] => green)

arsort()	Sorts an associative array in descending order, according to the value	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		arsort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Joe, Value=43
			Key=Ben, Value=37
			Key=Peter, Value=35

asort()	Sorts an associative array in ascending order, according to the value	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		asort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Peter, Value=35
			Key=Ben, Value=37
			Key=Joe, Value=43

compact()	Create array containing variables and their values	$firstname = "Peter"; $lastname = "Griffin"; $age = "41"; $result = compact("firstname", "lastname", "age"); print_r($result);	Array ([firstname] => Peter [lastname] => Griffin [age] => 41)
count()	Returns the number of elements in an array	$cars=array("Volvo","BMW","Toyota"); echo count($cars);	3

in_array()	Checks if a specified value exists in an array	$people = array("Peter", "Joe", "Glenn", "Cleveland");
		
		if (in_array("Glenn", $people))
		{
		echo "Match found";
		}
		else
		{
			Match found

		echo "Match not found"; }	

krsort()	Sorts an associative array in descending order, according to the key	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		krsort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Peter, Value=35
			Key=Joe, Value=43
			Key=Ben, Value=37

ksort()	Sorts an associative array in ascending order, according to the key	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		krsort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Ben, Value=37
			Key=Joe, Value=43
			Key=Peter, Value=35

rsort()	Sorts an indexed array in descending order	$cars=array("Volvo","BMW","Toyota"); rsort($cars);	Volvo Toyota BMW
sort()	Sorts an indexed array in ascending order	$cars=array("Volvo","BMW","Toyota"); sort($cars);	BMW Toyota Volvo

VARIABLE HANDLING FUNCTIONS

Function	**Description**
boolval()	Returns the boolean value of a variable
empty()	Checks whether a variable is empty
is_array()	Checks whether a variable is an array
is_bool()	Checks whether a variable is a Boolean
is_float()/is_double()	Checks whether a variable is of type float
is_int()/is_integer()	Checks whether a variable is of type integer
is_null()	Checks whether a variable is NULL
is_numeric()	Checks whether a variable is a number or a numeric string
is_string()	Checks whether a variable is of type string
isset()	Checks whether a variable is set (declared and not NULL)
print_r()	Prints the information about a variable in a human-readable

	way
unset()	Unsets a variable
var_dump()	Dumps information about one or more variables

MISCELLANEOUS FUNCTIONS:

Function	**Description**	**Example**
Define	define(*name,value,case_insensitive*) A constant's value cannot be changed after it is set Constant names do not need a leading dollar sign ($) Constants can be accessed regardless of scope Constant values can only be strings and numbers	define("pi",3.14); echo pi; Output:3.14
Exit()	exit(*message*) prints a message and terminates the current script	$x = 1; exit ($x);
Die()	die(*message*) Print a message and terminate the current script Note: exit() is used to stop the execution of the program, and die() is used to throw an exception and stop the execution.	mysql_connect("hostname","mysqlusername","") or die('We are aware of the problem and working on it');
Header	header(*header*, *replace*, *http_response_code*) Sends a raw HTTP header to a client	header("Expires: Mon, 26 Jul 1997 05:00:00 GMT"); header("Cache-Control: no-cache");

GET,POST AND REQUET METHODS

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

- The GET Method
- The POST Method

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

name1=value1&name2=value2&name3=value3

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

THE GET METHOD

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

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

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

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

```
<?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>
```

THE POST METHOD

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

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

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

```
<?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>
```

THE $_REQUEST VARIABLE

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

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

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

```
<?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.

FILES COMMANDS

INCLUDE AND REQUIRE STATEMENTS

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

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

Syntax:

include '*filename*';
or
require '*filename*';

WORKING WITH FILES IN PHP

To create, access, and manipulate files on your web server using the PHP file system functions.

OPENING A FILE WITH PHP FOPEN() FUNCTION

To work with a file you first need to open the file. The PHP fopen() function is used to open a file.

Syntax:

fopen(*filename*, *mode*)

Example:

```
<?php $file = "data.txt"; // Check the existence of file
if(file_exists($file))
{ // Attempt to open the file
$handle = fopen($file, "r");
}
Else
{ echo "ERROR: File does not exist."; }
 ?>
```

Note: check whether a file or directory exists or not before trying to access it, with the PHP `file_exists()` function.

Mode

Modes	What it does
R	Open the file for reading only.
r+	Open the file for reading and writing.
w	Open the file for writing only and clears the contents of file. If the file does not exist, PHP will attempt to create it.
w+	Open the file for reading and writing and clears the contents of file. If the file does not exist, PHP will attempt to create it.
a	Append. Opens the file for writing only. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
a+	Read/Append. Opens the file for reading and writing. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
x	Open the file for writing only. Return FALSE and generates an error if the file already exists. If the file does not exist, PHP will attempt to create it.

Modes	What it does
x+	Open the file for reading and writing; otherwise it has the same behavior as 'x'.

FCLOSE() FUNCTION:

Closing a File

```
<?php $file = "data.txt"; // Check the existence of file
if(file_exists($file))
{ // Open the file for reading
$handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
/* Some code to be executed */
// Closing the file handle
        fclose($handle);
}
Else
{
        echo "ERROR: File does not exist.";
}
?>
```

FREAD():

The fread() function can be used to read a specified number of characters from a file.

Syntax:

```
        fread(FILE HANDLE, LENGTH IN BYTES)
```

Example

```
<?php $file = "data.txt";
        if(file_exists($file))
        { $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
                $content = fread($handle, "20");
                 fclose($handle);
                 echo $content; }
     else{ echo "ERROR: File does not exist."; }
?>
```

Note:

1. we can use filesize($filename) at the place of 20 for size of the file in bytes for reading entire file.
2. readfile() function,allows you to read the contents of a file without needing to open it.

FGETS():

The fgets() function is used to read a single line from a file.

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

FEOF():

feof() function checks if the "end-of-file" (EOF) has been reached.

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

FWRITE():

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

RENAME():RENAME A FILE OR DIRECTORY

rename(Old filename, New file name)

FILE UPLOAD

You can upload any kind of file like images, videos, ZIP files, Microsoft Office documents, PDFs, as well as executables files and a wide range of other file types.

```
<html>
   <body>
   <?php
$target_path = "uploads/";
$target_path = $target_path.basename( $_FILES['fileToUpload']['name']);

if(move_uploaded_file($_FILES['fileToUpload']['tmp_name'], $target_path)) {
   echo "File uploaded successfully!";
} else{
   echo "Sorry, file not uploaded, please try again!";
}
?>
   <form  method="post" enctype="multipart/form-data">
   Select File:
   <input type="file" name="fileToUpload"/>
   <input type="submit" value="Upload Image" name="submit"/>
</form>
  </body>
</html>
```

Note:

Basename()->**function which is used to return the base name of a file**

$_FILES['fileToUpload']['name'])-> file name

$_FILES['fileToUpload']['type'])-> file type

$_FILES['fileToUpload']['size'])-> file size
$_FILES['fileToUpload']['tmp_name'])->file temporary name

FILE DOWNLOAD

```
<?php
$file = "logo_ldg.png"; //Let say If I put the file name Bang.png
echo "<a href='download.php?nama=".$file."'>download</a> ";

?>
```

Download.php

```
<?php
$filename = $_GET["nama"];
$contenttype = "application/force-download";
header("Content-Type: " . $contenttype);
header("Content-Disposition: attachment; filename=\"" . basename($filename) . "\";");
readfile("your file uploaded path".$filename);
exit();
?>
```

- Header: The header function sets the headers for an HTTP Response given by the server.
- Redirect page.
  It is used to redirect a from one web page to another web page in PHP.
  header('Location:give your url here');
- Set Content-Type in header response:
  PHP defaults to sending Content-Type:text/html.If we want to change the Content-Type
- Content-Disposition
  In a regular HTTP response, the Content-Disposition response header is a header indicating if the content is expected to be displayed INLINE in the browser, that is, as a Web page or as part of a Web page, or as an ATTACHMENT, that is downloaded and saved locally.

WORKING WITH DIRECTORIES

We can open a directory and read its contents, create or delete a directory, list all files in the directory, and so on.

MKDIR(): CREATING A NEW DIRECTORY

You can create a new and empty directory by calling the `mkdir()` function with the path and name of the directory to be created,

```
<?php // The directory path
 $dir = "testdir"; // Check the existence of directory
if(!file_exists($dir)){ // Attempt to create directory
        if(mkdir($dir))
        { echo "Directory created successfully."; }
        Else
        { echo "ERROR: Directory could not be created."; }
}
Else
{ echo "ERROR: Directory already exists."; }
```

```
?>
```

COPY(SOURCE,DESTINATION): COPYING FILES FROM ONE LOCATION TO ANOTHER

```
<?php // Source file path
$file = "example.txt"; // Destination file path
$newfile = "backup/example.txt"; // Check the existence of file
 if(file_exists($file))
{ // Attempt to copy file
if(copy($file, $newfile))
{ echo "File copied successfully."; }
Else
{ echo "ERROR: File could not be copied."; }
}
else{ echo "ERROR: File does not exist.";
 } ?>
```

SCANDIR(): LISTING ALL FILES IN A DIRECTORY

scandir() function to list files and directories inside the specified path.

```
<?php
// Define a function to output files in a directory
function outputFiles($path){
    // Check directory exists or not
    if(file_exists($path) && is_dir($path)){
        // Scan the files in this directory
        $result = scandir($path);

        // Filter out the current (.) and parent (..) directories
        $files = array_diff($result, array('.', '..'));

        if(count($files) > 0){
            // Loop through retuned array
            foreach($files as $file){
                if(is_file("$path/$file")){
                    // Display filename
                    echo $file . "<br>";
                } else if(is_dir("$path/$file")){
                    // Recursively call the function if directories found
                    outputFiles("$path/$file");
                }
            }
        } else{
            echo "ERROR: No files found in the directory.";
        }
    } else {
        echo "ERROR: The directory does not exist.";
    }
}
```

```
// Call the function
outputFiles("testdir");
?>
```

Note: glob() function, which matches files based on the pattern.

COOKIES

A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. They are typically used to keeping track of information such as username that the site can retrieve to personalize the page when user visit the website next time.

SET COOKIES: USE TO SET COOKIES

setcookie(*name*, *value*, *expire*, *path*, *domain*, *secure*);

Here,

name	The name of the cookie.
value	The value of the cookie. Do not store sensitive information
expires	The expiry date. After this time cookie will become inaccessible. The default value is 0.
path	Specify the path on the server for which the cookie will be available. If set to /, the cookie will be available within the entire domain.
domain	Specify the domain for which the cookie is available to e.g www.example.com.
secure	This field, if present, indicates that the cookie should be sent only if a secure HTTPS connection exists.

Example: create a cookie named username and assign the value value John Carter to it. It also specify that the cookie will expire after 30 days (30 days * 24 hours * 60 min * 60 sec).

```
setcookie("username", "John Carter", time()+30*24*60*60);
```

ACCESS COOKIES:

$_COOKIE superglobal variable is used to retrieve a cookie value.

Example:

```
echo $_COOKIE["username"];
```

REMOVING COOKIES:

Delete a cookie by calling the same setcookie() function with the cookie name and any value or set the expiration date in the past.

Example:

```
setcookie("username", "", time()-3600);
```

Example:

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

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

</body>
</html>
```

SESSIONS

session stores data on the server rather than user's computer. In a session based environment, every user is identified through a unique number called session identifier or SID. Default session expired after 24 minutes which can be modify by changing in php.ini file.
Which resolve two problems in cookies:

1) Store data using cookies but it has some security issues. Since cookies are stored on user's computer it is possible for an attacker to easily modify a cookie content to insert potentially harmful data in your application that might break your application.
2) Also every time the browser requests a URL to the server, all the cookie data for a website is automatically sent to the server within the request. It means if you have stored 5 cookies on user's system, each having 4KB in size, the browser needs to upload 20KB of data each time the user views a page, which can affect your site's performance.

START SESSION:

store any information in session variables, you must first start up the session.
Example:
session_start();

STORING SESSION DATA

store all your session data as key-value pairs in the $_SESSION[] superglobal array. The stored data can be accessed during lifetime of a session.
Syntax:
$_SESSION["sessionname"] = value;

Example:
$_SESSION["firstname"] = "Peter";

ACCESSING SESSION DATA

To access the session data we set on our previous example from any other page on the same web domain

```
echo 'Hi, ' . $_SESSION["firstname"];
```

DESTROYING A SESSION

If you want to remove certain session data
unset($_SESSION[“session name”]);
Example:
unset($_SESSION["firstname"])
Note:
To destroy a session completely, simply call the session_destroy() function.

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

SEND EMAIL

Using mail() function for creating and sending email messages to one or more recipients
Syntax:
mail(*to*, SUBJECT, MESSAGE, HEADERS, PARAMETERS)

To	The recipient's email address.
Subject	Subject of the email to be sent. This parameter i.e. the subject line cannot contain any newline character (\n).
Message	Defines the message to be sent. Each line should be separated with a line feed-LF (\n). Lines should not exceed 70 characters.
Headers (optional)	This is typically used to add extra headers such as "From", "Cc", "Bcc". The additional headers should be separated with a carriage return plus a line feed-CRLF (\r\n).
Parameters (optional)	Used to pass additional parameters.

Example:

```php
<?php $to = 'maryjane@email.com';
 $subject = 'Marriage Proposal';
 $message = 'Hi Jane, will you marry me?';
 $from = 'peterparker@email.com'; // Sending email
if(mail($to, $subject, $message))
{ echo 'Your mail has been sent successfully.'; }
Else
{ echo 'Unable to send email. Please try again.'; }
?>
```

FORMS CREATING

The HTML <form> tag is used for creating a form for user input. A form can contain textfields, checkboxes, radio-buttons and more. Forms are used to pass user-data to a specified URL.

Attribute:

action: Specifies where to send the form-data when a form is submitted. Value is URL.

method: Specifies the HTTP method to use when sending form-data. It will be get or post.

name:Specifies the name of a form in text formate.

target:Specifies where to display the response that is received after submitting the form. It will be _blank, _self,_parent, _top.

Element:

The <form> element can contain one or more of the following form elements:

<input>
<textarea>
<button>
<select>
<option>
<optgroup>
<fieldset>
<legend>
<label>

[1] < INPUT >

The <input> tag specifies an input field where the user can enter data. <input> elements are used within a <form> element to declare input controls that allow users to input data. An input field can vary in many ways, depending on the type attribute.

Attribute:

value: Specifies the value of an <input> element. It may be a *text*.

name:Specifies the name of an <input> element. It may be a text.

type: Specifies the type <input> element to display . it will be button, checkbox, color, date, datetime, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week.

object	**description**
checkbox	Checkboxes are used when you want the user to select one or more options of a limited number of choices
color	used for input fields that should contain a color.
date	used for input fields that should contain a date.
Datetime	used for input fields that should contain a date and time.
Email	used for input fields that should contain a email
File	used for input fields that should contain a file.
Hidden	used for input fields that should contain hidden field
Image	used for input fields that should contain image

monthuse for input fields that should contain a month and year.
number use for input fields that should contain a numeric value
password use for input fields that should contain password.
radio use when you want the user to select one of a limited number of choices.
Rangeuse for input fields that should contain value with in a range.
Reset used for input fields that should reset the value.
search used for search fields (a search field behaves like a regular text field).
Submit used for input fields that should submit the value.
text used when you want the user to type letters, numbers, etc. in a form.
Tel used for input fields that should contain a telephone number.
Time used for input fields that should contain a time.
url used for input fields that should contain a url.
week used for input fields that should contain a week and year.

[2] <TEXTAREA>

An input that allows a large amount of text to be entered, and allows the height of input box to be a specified unlike the standard input tag.

Attribute:
name - The unique name assigned to the form field.
rows - The number of rows of text, defines the vertical size of the text area.
cols - The horizontal size of the text box, defined as the number of characters
(i.e. columns)

[3] <BUTTON>

A form button is similar to other form inputs

attributes:
name - Unique name for the button to be used by the action script.
type - The button type, either submit or reset, determines whether the form is to be submitted or cleared upon pressing it.
value - Text that appears on the button, such as OK or Submit.
size - Determines the length (or width) of the button.

[4] <SELECT>

A drop-down list, also refered to as a combo-box, allowing a selection to be made from a list of items.
Attribute:
name - Selector name
size - The minimum size (width) of the selection list, usually not required as the size of the items will define the list size.
multiple - Allows a user to select multiple items from the list, normally limited to one.

[5] <OPTION>

An option tag is needed for each item in the list, and must appear within the select tags. The text to be shown for the option must appear between the option tags.

Attribute:

value - The value is the data sent to the action script with the option is selected. This is not the text that appears in the list.

selected - Sets the default option that is automatically selected when the form is shown.

[6] <OPTGROUP>

The <optgroup> is used to group related options in a drop-down list. If you have a long list of options, groups of related options are easier to handle for a user.

Attribute:

label: Specifies a label for an option-group. It is a text.

[7] <FIELDSET>

used to group related elements in a form.

Attribute:

disabled: Specifies that a group of related form elements should be disabled.

form: Specifies one or more forms the fieldset belongs to.

name: Specifies a name for the fieldset.

[8] <LEGEND>

The <legend> tag defines a caption for the <fieldset> element.

Example <fieldset>

```
<legend style="float:right">Personalia:</legend>
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname"><br>
<input type="submit" value="Submit">
</fieldset>
```

[9] <LABEL>

The <label> tag defines a label for an <input> element. The for attribute of the <label> tag should be equal to the id attribute of the related element to bind them together.

Attribute:

for: Specifies which form element a label is bound to

Example:

```
<html>
  <body>
 <form name="input" action="test.html" method="get">
 User Login: <input type="text" name="user" ><br>
Password: <input type="password" name="pass">
<br>
```

```
Address:<textarea>default null</textarea><br>
Gender: <input type="radio" name="gen" value="male">
<input type="radio" name="gen" value="female" checked><br>
Language:<input type="checkbox" name="language" value="spanish"> I speak Spanish<br>
<input type="checkbox" name="language" value="french"> I speak French <br>
College
<select name="college">
<option value="v">vvk</option>
<option value="p">pts</option>
<option value="d">drb</option>
</select><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

VALIDATION OF FORMS

An HTML form contains various input fields such as text box, checkbox, radio buttons, submit button, and checklist, etc. These input fields need to be validated, which ensures that the user has entered information in all the required fields and also validates that the information provided by the user is valid and correct.
It may be :Empty String, Validate String, Validate Numbers, Validate Email, Validate URL, Input length and so on:
It will manage using Php filters & regular expressions
Php filter:
To validate data using filter extension you need to use the PHP's filter_var() function.

FUNCTIONS AND FILTERS

To filter available, use one of the following filter function:
filter_var()- Filter a single variable with a specified filter.
filter_var_array()- Filter several variables with the same or different filters.
filter_input()- Get one input variable and filter it.
filter_input_array – Get several input variables and filter them with the same or different filters.

Syntax: filter_var(*variable*, *filter*, *options*)

The first parameter is the value to be filtered, the second parameter is the ID of the filter to apply, and the third parameter is the array of options related to filter.

ID	Description
FILTER_VALIDATE_BOOLEAN, **FILTER_VALIDATE_BOOL**	Returns true for "1", "true", "on" and "yes". Returns false otherwise. If FILTER_NULL_ON_FAILURE is set, false is returned only for "0", "false", "off", "no", and "", and null is returned for all non-boolean values.
FILTER_VALIDATE_DOMAIN	Validates whether the domain name label lengths are valid.

ID	Description
FILTER_VALIDATE_EMAIL	Validates whether the value is a valid e-mail address.
FILTER_VALIDATE_FLOAT	Validates value as float, optionally from the specified range, and converts to float on success.
FILTER_VALIDATE_INT	Validates value as integer, optionally from the specified range, and converts to int on success.
FILTER_VALIDATE_IP	Validates value as IP address, optionally only IPv4 or IPv6 or not from private or reserved ranges.
FILTER_VALIDATE_MAC	Validates value as MAC address.
FILTER_VALIDATE_REGEXP	Validates value against regexp, a Perl-compatible regular expression.
FILTER_VALIDATE_URL	Validates value as URL

Example: We validate an integer using the **filter_var()** function:

```
<?php
$int=1234;
if(!filter_var($int,FILTER_VALIDATE_INT))
{
echo "Integer is not valid";
} else {
echo "Integer is valid";
}
```

REGULAR EXPRESSION:

A regular expression is a sequence of characters that forms a search pattern. A regular expression can be a single character, or a more complicated pattern.

Regular Expression Functions

Function	Description	
preg_match()	Returns 1 if the pattern was found in the string and 0 if not	$str = "I LIKE apple."; $pattern = "/like/i"; echo preg_match($pattern, $str); // Outputs 1
preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0	$str = "It is an apple.I like apple"; $pattern = "/apple/"; echo preg_match_all($pattern, $str); // Outputs 2
preg_replace()	Returns a new string where matched patterns have been replaced with another string	$str = " It is an apple.I like apple "; $pattern = "/apple/i"; echo preg_replace($pattern, "orange", $str); // Outputs " It is an orange.I like orange "

Regular Expression Patterns

Expression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

Metacharacters

Metacharacter	Description
|	Find a match for any one of the patterns separated by | as in: cat|dog|fish
.	Find just one instance of any character
^	Finds a match as the beginning of a string as in: ^Hello
$	Finds a match at the end of the string as in: World$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers

Quantifier {quantities}	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{x}	Matches any string that contains a sequence of *X n*'s
n{x,y}	Matches any string that contains a sequence of X to Y *n*'s
n{x,}	Matches any string that contains a sequence of at least X *n*'s

EMPTY STRING

Checks that the field is not empty. If the user leaves the required field empty, it will show an error message.

```
if (empty ($_POST["name"])) {
   $errMsg = "Error! You didn't enter the Name.";
         echo $errMsg;
} else {
   $name = $_POST["name"];
}
```

VALIDATE STRING

checks that the field will contain only alphabets and whitespace, for example - name. If the name field does not receive valid input from the user

```
$name = $_POST ["Name"];
if (!preg_match ("/^[a-zA-z]*$/", $name) ) {
   $ErrMsg = "Only alphabets and whitespace are allowed.";
         echo $ErrMsg;
} else {
   echo $name;
}
```

VALIDATE NUMBER

the field will only contain a numeric value.

For example - Mobile no

```
$mobileno = $_POST ["Mobile_no"];
if (!preg_match ("/^[0-9]*$/", $mobileno) ){
   $ErrMsg = "Only numeric value is allowed.";
   echo $ErrMsg;
} else {
   echo $mobileno;
}
```

VALIDATE EMAIL

A valid email must contain @ and . symbols

```
$email = $_POST ["Email"];
$pattern = "^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$^";
if (!preg_match ($pattern, $email) ){
   $ErrMsg = "Email is not valid.";
        echo $ErrMsg;
} else {
   echo "Your valid email address is: " .$email;
}
```

INPUT LENGTH VALIDATION

The input length validation restricts the user to provide the value between the specified range, for Example - Mobile Number. A valid mobile number must have 10 digits.

```
$mobileno = strlen ($_POST ["Mobile"]);
```

```
$length = strlen ($mobileno);

if ( $length < 10 && $length > 10) {
   $ErrMsg = "Mobile must have 10 digits.";
         echo $ErrMsg;
} else {
   echo "Your Mobile number is: " .$mobileno;
}
```

VALIDATE URL

code validates the URL of website provided by the user via HTML form

```
$websiteURL = $_POST["website"];
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "URL is not valid";
  echo $websiteErr;
} else {
   echo "Website URL is: " .$websiteURL;
}
```

BUTTON CLICK VALIDATE

Code validates that the user click on submit button and send the form data to the server one of the following method - get or post.

```
if (isset ($_POST['submit']) {
   echo "Submit button is clicked.";
   if ($_SERVER["REQUEST_METHOD"] == "POST") {
      echo "Data is sent using POST method ";
   }
} else {
   echo "Data is not submitted";
}
```

CLASSES AND OBJECTS IN PHP

Classes and objects are the two main aspects of object-oriented programming.

Class : Fruit

Objects: Apple,Banana,Mango

a class is a template for objects, and an object is an instance of a class.

DEFINE A CLASS

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}).

SYNTAX

```
<?php
class classname {

Variables/properties declaration

Methods
  // code goes here...
}
?>
```

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

DEFINE OBJECTS

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the new keyword.

Syntax:

```
objectname = new classname;
```

Example:

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;
```

```
  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$a = new Fruit();
$b = new Fruit();
$a->set_name('Apple');
$b->set_name('Banana');

echo $a->get_name();
echo "<br>";
echo $b->get_name();
?>
```

WHAT IS AN EXCEPTION?

An exception is an object that describes an error or unexpected behaviour of a PHP script.

Exceptions are thrown by many PHP functions and classes.

THE TRY...CATCH... STATEMENT

The try...catch statement to catch exceptions and continue the process..

SYNTAX

```
try

{
  code that can throw exceptions
}

catch(Exception $e)

{
  code that runs when an exception is caught
}
```

User defined functions and classes can also throw exceptions.

THROWING AN EXCEPTION

The throw statement allows a user defined function or method to throw an exception. When an exception is thrown, the code following it will not be executed.

If an exception is not caught, a fatal error will occur with an "Uncaught Exception" message.

SYNTAX:

```
throw new Exception("msg");
```

EXAMPLE 1:

```
<?php
function division($x,$y)
{ if ($x/$y==0)
        throw new exception("Answer is invalid.Division by zero");
        else
        return $x/$y;
}
try
{echo division(5,0);
}
catch(Exception $e)
{ echo $e->getMessage();
}
?>
```

Example 2:

```
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero", 1);
  }
  return $dividend / $divisor;
}

try

{
  echo divide(5, 0);
}

catch(Exception $ex)

{
  $code = $ex->getCode();
  $message = $ex->getMessage();
  $file = $ex->getFile();
  $line = $ex->getLine();
  echo "Exception thrown in $file on line $line: [Code $code]
  $message";

}

?>
```

WHAT IS JSON

JSON stands for **J**ava**S**cript **O**bject **N**otation. JSON is a standard lightweight data-interchange format which is quick and easy to parse and generate.

JSON, like XML, is a text-based format that's easy to write and easy to understand for both humans and computers, but unlike XML, JSON data structures occupy less bandwidth than their XML versions. JSON is based on two basic structures:

Object: This is defined as a collection of key/value pairs (i.e. `key:value`). Each object begins with a left curly bracket `{` and ends with a right curly bracket `}`. Multiple key/value pairs are separated by a comma `,`.

Array: This is defined as an ordered list of values. An array begins with a left bracket `[` and ends with a right bracket `]`. Values are separated by a comma `,`.

In JSON, keys are always strings, while the value can be
a **string**, **number**, **true** or **false**, **null** or even an **object** or an **array**. Strings must be enclosed in double quotes `"` and can contain escape characters such as `\n`, `\t` and `\`.

JSON object:

```
{
   "book": {
      "name": "Harry Potter and the Goblet of Fire",
      "author": "J. K. Rowling",
      "year": 2000,
      "genre": "Fantasy Fiction",
      "bestseller": true
   }
}
```

JSON array:

```
{
   "fruits": [
      "Apple",
      "Banana",
      "Strawberry",
      "Mango"
   ]
```

```
}
```

PARSING JSON WITH PHP

JSON data structures are very similar to PHP arrays. PHP has built-in functions to encode and decode JSON data. These functions are json_encode() and json_decode(), respectively. Both functions only works with UTF-8(Unicode Transformation Format - 8 bits) encoded (this code use for electronic communication in XML/Json/HTML5) string data.

ENCODING JSON DATA IN PHP

In PHP the json_encode() function is used to encode a value to JSON format. The value being encoded can be any PHP data type except a resource, like a database or file handle.

Example to encode a PHP associative array into a JSON object:

```
<?php
// An associative array
$marks = array("Peter"=>65, "Harry"=>80, "John"=>78, "Clark"=>90);

echo json_encode($marks);
?>
```

Output

```
{"Peter":65,"Harry":80,"John":78,"Clark":90}
```

Similarly, you can encode the PHP indexed array into a JSON array, like this:

EXAMPLE

```
<?php
// An indexed array
$colors = array("Red", "Green", "Blue", "Orange", "Yellow");

echo json_encode($colors);
?>
```

Output:

["Red","Green","Blue","Orange","Yellow"]

Note: force json_encode() function to return an PHP indexed array as JSON object by using the JSON_FORCE_OBJECT option, as shown in the example below:

```
<?php

// An indexed array

$colors = array("Red", "Green", "Blue", "Orange");


echo json_encode($colors, JSON_FORCE_OBJECT);

?>
```

The output of the above example will look like this:

{"0":"Red","1":"Green","2":"Blue","3":"Orange"}

As you can see in the above examples a non-associative array can be encoded as array or object. However, an associative array always encoded as object.

Note: error may be occurs in JSON_FORCE_OBJECT due lower version of php.

DECODING JSON DATA IN PHP

Decoding JSON data is as simple as encoding it. You can use the PHP json_decode() function to convert the JSON encoded string into appropriate PHP data type.

Example to decode or convert a JSON object to PHP object.

```
<?php

// Store JSON data in a PHP variable

$json = '{"Peter":65,"Harry":80,"John":78,"Clark":90}';


var_dump(json_decode($json));

?>
```

Output:

object(stdClass)#1 (4) { ["Peter"]=> int(65) ["Harry"]=> int(80) ["John"]=> int(78) ["Clark"]=> int(90) }

By default the json_decode() function returns an object. However, you can optionally specify a second parameter **$assoc** which accepts a boolean value that when set as true JSON objects are decoded into associative arrays. It is false by default.

Example to decode or convert a JSON object to PHP array.

```
<?php
// Store JSON data in a PHP variable
$json = '{"Peter":65,"Harry":80,"John":78,"Clark":90}';


var_dump(json_decode($json, true));
?>
```

Output

```
array(4) { ["Peter"]=> int(65) ["Harry"]=> int(80) ["John"]=> int(78) ["Clark"]=> int(90) }
```

Now let's check out an example that will show you how to decode the JSON data and access individual elements of the JSON object or array in PHP.

```
<?php
// Assign JSON encoded string to a PHP variable
$json = '{"Peter":65,"Harry":80,"John":78,"Clark":90}';


// Decode JSON data to PHP associative array
$arr = json_decode($json, true);
// Access values from the associative array
echo $arr["Peter"];  // Output: 65
echo $arr["Harry"];  // Output: 80
echo $arr["John"];   // Output: 78
echo $arr["Clark"];  // Output: 90


// Decode JSON data to PHP object
$obj = json_decode($json);
// Access values from the returned object
echo $obj->Peter;   // Output: 65
echo $obj->Harry;   // Output: 80
```

```
echo $obj->John;    // Output: 78
echo $obj->Clark;   // Output: 90
?>
```