504: Web Framework and Services

Unit-2: PHP functions

- **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/>";
}
sayHello("Sonoo");
?>
```

Call by reference:

By default, value passed to the function is called 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
function add($x,$y)	//add() function with two parameter
{
	$sum=$x+$y;
	echo "Sum = $sum <br><br>";
}
function sub($x,$y)	//sub() function with two parameter
{
	$sub=$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>
```

2.1 Math functions, Date and Time functions GET and POST methods

- **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)	4 -4

		ceil(14.81)	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)	WowWowWow
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
strspn()	Gives the count of the characters found in a given string that contains only characters from a specified charlist	strspn("Hello world!","kHlleo")	5
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 variable in a more human-readable way.	$a = array("red", "green", "blue"); print_r($a);	Array ([0] => 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
			2013-05-01 12:20:55

date()	Formats a local date and time Note check table 1 for format	echo date("l") . " ";	Thursday
getdate()	Returns date/time	print_r(getdate());	Array ([seconds] =>

	information of a timestamp or the current local date/time		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	$a1=array("a"=>"red","b"=>"green","c"=>"blue"); $a2=array("a"=>"red","c"=>"blue","d"=>"pink");	Array ([b] => green)

	only)	$result=array_diff_key($a1,$a2); print_r($result); ?>	
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);	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $result=array_flip($a1); print_r($result);
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-dimensional 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	$a1=array("red","green"); $a2=array("blue","yellow"); print_r(array_replace($a1,$a2));	Array ([0] => blue [1] => yellow)

	the values from following arrays		
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 values in an array	$a=array(5,15,25); echo array_sum($a);	45
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	$cars=array("Volvo","BMW","Toyota"); echo count($cars);	3

	array		

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
		{
		echo "Match not found";
		}
			Match 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");

- **Math Functions**

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        echo(pi());
        echo(min(0, 150, 30, 20, -8, -200) . "<br>");
        echo(max(0, 150, 30, 20, -8, -200));
        echo(abs(-6.7));
        echo(sqrt(64));         // returns 8
        echo(round(0.60)); // returns 1
        echo(round(0.49)); // returns 0
        echo(rand());
        echo (ceil(3.3)."<br/>");          // 4
        echo (ceil(7.333)."<br/>");      // 8
        echo (ceil(-4.8)."<br/>");       // -4
        echo   (floor(3.3)."<br/>");   //   3
        echo  (floor(7.333)."<br/>");  //  7
        echo (floor(-4.8)."<br/>");     // -5

        $num=pow(3, 2);
        echo "Your number is = pow (3, 2)".'<br>';
```

```
        echo "By using sqrt function Your number is : ".$num;
        $x = 7;
        $y = 2;
        echo "Your Given Nos is : $x=7, $y=2"
        $result ="By using 'fmod()' Function your value is:".fmod($x,$y);
        echo $result;
?>
</body></html>
```

- **Date and Time functions:**

Example:

```
<!DOCTYPE html>
<html><body>
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
</body></html>
```

Example:

```
<!DOCTYPE html>
<html><body>
<?php
var_dump(checkdate(12,31,-400));
echo "<br>";
var_dump(checkdate(2,29,2003));
echo "<br>";
var_dump(checkdate(2,29,2004));
?>
</body></html>
```

- **Example: mktime() Function**

```
<!DOCTYPE html>
<html><body>
<?php
// Prints: October 3, 1975 was on a Friday
echo "Oct 3, 1975 was on a ".date("l", mktime(0,0,0,10,3,1975)) . "<br><br>";

//The mktime() function is useful for doing date arithmetic and validation.
//It will automatically calculate the correct value for out-of-range input:
echo date("M-d-Y",mktime(0,0,0,12,36,2001)) . "<br>";
echo date("M-d-Y",mktime(0,0,0,14,1,2001)) . "<br>";
echo date("M-d-Y",mktime(0,0,0,1,1,2001)) . "<br>";
echo date("M-d-Y",mktime(0,0,0,1,1,99)) . "<br>";
?>
</body></html>
```

❖ **Example: getdate() Function**

```
<!DOCTYPE html>
<html><body>
<?php
        // Print the array from getdate()
        print_r(getdate());
        echo "<br><br>";

        // Return date/time info of a timestamp; then format the output
        $mydate=getdate(date("U"));
        echo "$mydate[weekday], $mydate[month] $mydate[mday], $mydate[year]";
?>
</body></html>
```

❖ **GET and POST methods**

PHP Form Handling

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.

We can create and use forms in PHP. To get form data, we need to use PHP superglobals $_GET and $_POST.

The form request may be get or post. To retrieve data from get request, we need to use $_GET, for post request $_POST.

❖ **PHP Get Form**

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 up to 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.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Example of PHP GET method</title>
</head> <body>
<?php
```

```
if(isset($_GET["name"]))
{     echo "<p>Hi, " . $_GET["name"] . "</p>"; }
?>
<form method="get" action="<?php echo $_SERVER["PHP_SELF"];?>">
        <label for="inputName">Name:</label>
        <input type="text" name="name" id="inputName">
        <input type="submit" value="Submit">
</form>
</body>
</html>
```

- **PHP Post Form**
  - 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.

Example1: POST methods

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>Example of PHP POST method</title>
</head>
<body>
        <?php
        if(isset($_POST["name"]))
        {
                echo "<p>Hi, " . $_POST["name"] . "</p>";
        }
        ?>
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
        <label for="inputName">Name:</label>
        <input type="text" name="name" id="inputName">
        <input type="submit" value="Submit">
</form>
</body>
```

Example2: Login Form(One Webpage to Another webpage)

FileName : form1.html

```
<html><head>
         <title> POST Example </title>
</head>
<body>
<form action="login.php" method="post">
        <table>
        <tr><td>Name:</td><td> <input type="text" name="name"/></td></tr>
        <tr><td>Password:</td><td> <input type="password" name="password"/></td></tr>
        <tr><td colspan="2"><input type="submit" value="login"/>  </td></tr>
```

```
        </table>
</form>
</body></html>
```

FileName : login.php

```
<?php
$name=$_POST["name"];   //receiving name field value in $name variable
$password=$_POST["password"];       //receiving password field value in $password variable
echo "Welcome: $name, your password is: $password";
?>
```

GET vs. POST

- Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.

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.

```
<!DOCTYPE html>
<html lang="en">
<head>
                <title>Example of PHP POST method</title>
</head>
<body>
 <form action="<?php echo $_SERVER["PHP_SELF"];?>" method = "POST">
        Name: <input type = "text" name = "name" />
        Age: <input type = "text" name = "age" />
        <input type = "submit" />
  </form>
<?php
  if( $_SERVER["REQUEST_METHOD"] == "POST" )
  {
          $name = htmlspecialchars($_REQUEST['name']);
          $age = htmlspecialchars($_REQUEST['age']);
          if (empty($name) && empty($age))
          {
                  echo "Name and age is empty";
          }
          else
          {  echo "<br><br><br>";
                  echo "Welcome ". $_REQUEST['name']. "<br />";
                  echo "You are ". $_REQUEST['age']. " years old.";
          }
  }
?>  </body></html>
```

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

2.2 Files: include, file parsing, directories, file upload, file download

PHP Include Files

- The *include* (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.
- Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

PHP 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.
- The include and require statements are identical, except upon failure:

require will produce a fatal error (E_COMPILE_ERROR) and stop the script

include will only produce a warning (E_WARNING) and the script will continue

- So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the **require** statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.
- Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax

```
include 'filename';
or
require 'filename';
```

Example 1

Assume we have a standard footer file called "footer.php" that looks like this:

```
<?php
    echo "<p>Copyright &copy; 2011-" . date("Y") . " Udhna College </p>";
?>
```

To include the footer file in a page, use the include statement:

```
<!DOCTYPE html>
<html><body>
<h1>   Welcome to my home page!   </h1>
<p>Some text.</p>
<p>Some more text.</p>
        <?php include 'footer.php';?>
</body>
</html>
```

Example 2: Assume we have a standard menu file called **"menu.php":**

```
<?php
echo '<a href="/default.php">Home</a> -
<a href="/html/default.php">BBA</a> -
<a href="/css/default.php">BCA</a> -
<a href="/js/default.php">BCOM</a> -
<a href="default.php">About US</a>';
?>
```

All pages in the Web site should use this menu file. Here is how it can be done (we are using a element so that the menu easily can be styled with CSS later):

File Name : Welcome.php

```
<!DOCTYPE html>
<html>
<body>
<div class="menu">
        <?php include 'menu.php';?>
</div>
<h1>Welcome to Udhna College <h1>
<p>BCA College</p>
<p>BCOM College</p>
<p>BBA College</p>
</body></html>
```

PHP include vs. require

The require statement is also used to include a file into the PHP code.

However, there is one big difference between include and require; when a file is included with the include statement and PHP cannot find it, the script will continue to execute.

If we do the same example using the require statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error.

Example:

```
<!DOCTYPE html>
<html><body>
<h1>Welcome to my home page!</h1>
 <?php
        $color="red";
        $car="BMW";
        require 'noFileExists.php';
        //include 'noFileExists.php';
       echo "<h2>I have a $color $car.<h2>";
  ?>  </body></html>
```

Use require when the file is required by the application.

Use include when the file is not required and application should continue when file is not found.

❖ File parsing, directories, file upload, file download

PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called **"webdictionary.txt",** stored on the server that looks like this:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

```
<!DOCTYPE html>
<html><body>
<?php
        echo readfile("webdictionary.txt");
?>
</body></html>
```

NOTE:The readfile() function is useful if all you want to do is open up a file and read its contents.

❖ **PHP File Open/Read/Close**

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function. We will use the text file, "**webdictionary.txt**", during the lessons:

The file may be opened in one of the following modes:

No.	Modes	Description
1	r	Open a file for read only. File pointer starts at the beginning of the file
2	w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
3	a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
4	x	Creates a new file for write only. Returns FALSE and an error if file already exists
5	r+	Open a file for read/write. File pointer starts at the beginning of the file
6	w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
7	a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
8	x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

PHP Create File - fopen()

- The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.
- If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).
- The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

```
$myfile = fopen("testfile.txt", "w")
```

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

```
<!DOCTYPE html><html><body>
<?php
        $myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
        echo fread($myfile,filesize("webdictionary.txt"));
        fclose($myfile);
?></body></html>
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Read File - fread()

The fread() 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 "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

The fclose() function is used to close an open file.
The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
        $myfile = fopen("webdictionary.txt", "r");
        fclose($myfile);    // some code to be executed....
?>
```

PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.
The example below outputs the first line of the "**webdictionary.txt**" file:

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

Note: *After a call to the fgets() function, the file pointer has moved to the next line.*

PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached.
The feof() function is useful for looping through data of unknown length.
The example below reads the "**webdictionary.txt**" file line by line, until end-of-file is reached:

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

```
}
 fclose($myfile);
?></body></html>
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

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

Note: After a call to the fgetc() function, the file pointer moves to the next character.

PHP File Create/Write

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

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

The example below writes a couple of names into a new file called "newfile.txt":

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

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "Salman Khan" and second contained "Shahrukh Khan". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

Salman Khan

Shahrukh Khan

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
```

```
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both Salman and Shahrukh have vanished, and only the data we just wrote is present:
Mickey Mouse
Minnie Mouse

PHP Append Text

You can append data to a file by using the "a" mode. The "a" mode appends text to the end of the file, while the "w" mode overrides (and erases) the old content of the file.
In the example below we open our existing file "newfile.txt", and append some text to it:

```
<?php
$myfile = fopen("newfile.txt", "a") or die("Unable to open file!");
$txt = "Donald Duck\n";
fwrite($myfile, $txt);
$txt = "Scooby-Doo\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, we will see that Donald Duck and Scooby-Doo is appended to the end of the file:
Mickey Mouse
Minnie Mouse
Donald Duck
Scooby-Doo

❖ PHP Parsing Directories

Working with Directories in PHP

In the previous chapter you've learned how to work with files in PHP. Similarly, PHP also allows you to work with directories on the file system, for example, you can open a directory and read its contents, create or delete a directory, list all files in the directory, and so on.

- mkdir(): To make new directory.
- opendir(): To open directory.
- readdir(): To read from a directory after opening it.
- closedir(): To close directory with resource-id returned while opening.
- rmdir(): To remove directory.

Creating a New Directory

You can create a new and empty directory by calling the PHP mkdir() function with the path and name of the directory to be created, as shown in the example below:

Example:

```
<?php
$dir = "testdir";              // The directory path
```

```
// Check the existence of directory
if(!file_exists($dir))
{
   if(mkdir($dir))           // Attempt to create directory
         echo "Directory created successfully.";
   else
         echo "ERROR: Directory could not be created.";
}
else
         echo "ERROR: Directory already exists.";
?>
```

NOTE: To make the mkdir() function work, the parent directories in the directory path parameter has to exist already, for example, if you specify the directory path as testdir/subdir than the testdir has to exist otherwise PHP will generate an error.

Copying Files from One Location to Another

You can copy a file from one location to another by calling PHP copy() function with the file's source and destination paths as arguments. If the destination file already exists it'll be overwritten. Here's an example which creates a copy of "example.txt" file inside backup folder.

Example:

```
<?php//Copying Files from One Location to Another

$file = "example.txt";// Source file path
$newfile = "backup/example.txt";// Destination file path

if(file_exists($file))// Check the existence of file
{
   if(copy($file, $newfile)) // Attempt to copy file
      echo "<h2> File copied successfully. </h2>";
   else
      echo " <h2> ERROR: File could not be copied. </h2> ";
}
else
   echo " <h2> ERROR: File does not exist. <h2> ";
?>
```

Listing All Files in a Directory

You can use the PHP scandir() function to list files and directories inside the specified path.

Now we're going to create a custom function that will recursively list all files in a directory using PHP. This script will be helpful if you're working with deeply nested directory structure.

array_diff($a1,$a2) :Compare the values of two arrays, and return the differences

Example: Listing All Files in a Directory

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

```
        // 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"))
                {
                    echo $file . "<br>";       // Display filename
                } else if(is_dir("$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("mydir");
?>
```

❖ PHP File Upload

PHP allows you to upload single and multiple files through few lines of code only.
PHP file upload features allow you to upload binary and text files both. Moreover, you can have the full control over the file to be uploaded through PHP authentication and file operation functions.

PHP $_FILES

The PHP global $_FILES contains all the information of file. By the help of $_FILES global, we can get file name, file type, file size, temp file name and errors associated with file.
Here, we are assuming that file name is filename.

$_FILES['filename']['name']
returns file name.

$_FILES['filename']['type']
returns MIME type of the file.

$_FILES['filename']['size']
returns size of the file (in bytes).

$_FILES['filename']['tmp_name']
returns temporary file name of the file which was stored on the server.

$_FILES['filename']['error']
returns error code associated with this file.

- **move_uploaded_file() function**

The move_uploaded_file() function moves the uploaded file to a new location. The move_uploaded_file() function checks internally if the file is uploaded thorough the POST request. It moves the file if it is uploaded through the POST request.

Syntax:

```
bool move_uploaded_file ( string $filename , string $destination )
```

PHP File Upload Example: uploadform.html

```
<html><body>
<head>   <title> File Upload </title>  </head>
<form action="uploader.php" method="post" enctype="multipart/form-data">
   Select File:
   <input type="file" name="fileToUpload"/>
   <input type="submit" value="Upload Image" name="submit"/>
</form>
</body></html>
```

File: uploader.php

```
<?php
$target_path = "C:\xampp\htdocs\CH2\FileUpload\";
$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!";
?>
```

- **PHP Download File**

PHP enables you to download file easily using built-in readfile() function. The readfile() function reads a file and writes it to the output buffer.

PHP readfile() function

Syntax:

```
int readfile ( string $filename [, bool $use_include_path = false [, resource $context ]] )
```

Parameters:

- **$filename:** represents the file name
- **$use_include_path:** it is the optional parameter. It is by default false. You can set it to true to the search the file in the included_path.
- **$context:** represents the context stream resource.
- **int:** it returns the number of bytes read from the file.

PHP Download File Example: (Text File) download1.php

```
<?php
$file_url  =  'C:\xampp\htdocs\CH2\FileUpload\data1.txt“;
header('Content-Type: application/octet-stream');
header("Content-Transfer-Encoding: utf-8");
header("Content-disposition: attachment; filename=\"" . basename($file_url) . "\"");
readfile($file_url);
?>
```

PHP Download File Example: (Binary File) download2.php

```
<?php
$file_url = „binary.dat“;
header('Content-Type: application/octet-stream');
header("Content-Transfer-Encoding: Binary");
header("Content-disposition: attachment; filename=\"" . basename($file_url) . "\"");
readfile($file_url);
?>
```

2.3 Cookies and Sessions, Send Email

❖ **What is a Cookie?**

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.

Tip: Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.

Setting a Cookie in PHP

The setcookie() function is used to set a cookie in PHP. Make sure you call the setcookie() function before any output generated by your script otherwise cookie will not set. The basic syntax of this function can be given with:

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

The parameters of the setcookie() function have the following meanings:

Parameter	Description
name	The name of the cookie.
value	The value of the cookie. Do not store sensitive information since this value is stored on the user's computer.
expires	The expiry date in UNIX timestamp format. 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.

Tip: If the expiration time of the cookie is set to 0, or omitted, the cookie will expire at the end of the session i.e. when the browser closes.

Here's an example that uses setcookie() function to create a cookie named username and assign the value 'Abc' to it. It also specifies that the cookie will expire after 30 days (30 days * 24 hours * 60 min * 60 sec).

Example 1:

```
<?php
// Setting a cookie
setcookie("username", "Abc", time()+30*24*60*60);
?>
```

Note: All the arguments except the name are optional. You may also replace an argument with an empty string ("") in order to skip that argument, however to skip expire argument use a zero (0) instead, since it is an integer.

Example2:Set_Cookie.php

```
<?php
$cookie_name = "username";
$cookie_value = "Abc";
setcookie($cookie_name, $cookie_value, time() + (24*60*60 * 30), "/");  // to expire in 30 days
?>
<html>
<body>
<?php
        if(!isset($_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];       }
?>
<br><br>
<h3><a href="Welcome.php"> Welcome </a></h3>
</body></html>
```

Warning: Don't store sensitive data in cookies since it could potentially be manipulated by the malicious user. To store the sensitive data securely use sessions instead.

Accessing Cookies Values

The PHP $_COOKIE **superglobal** variable is used to retrieve a cookie value. It typically an associative array that contains a list of all the cookies values sent by the browser in the current request, keyed by cookie name. The individual cookie value can be accessed using standard array notation, for example to display the username cookie set in the previous example, you could use the following code.

Example:

```
<?php
       // Accessing an individual cookie value
      echo $_COOKIE["username"];          // Output is Abc
?>
```

It's a good practice to check whether a cookie is set or not before accessing its value. To do this you can use the PHP isset() function, like this:

Example:Welcome.php

```
<?php
// Verifying whether a cookie is set or not
if(isset($_COOKIE["username"])){
    echo "Hi " . $_COOKIE["username"];
} else{
    echo "Welcome Guest!";
}
?>
```

Removing Cookies

You can delete a cookie by calling the same setcookie() function with the cookie name and any value (such as an empty string) however this time you need the set the expiration date in the past, as shown in the example below:

```
<?php       // Deleting a cookie
        setcookie("username", "", time()-3600);
        echo "Cookie 'username' is deleted.";
?>
```

Tip: You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

❖ **What is a Session?**

- Although you can 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.
- 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.
- You can solve both of these issues by using the PHP session. A PHP 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. This unique session ID is used to link each user with their own information on the server like emails, posts, etc.

Tip: *The session IDs are randomly generated by the PHP engine which is almost impossible to guess. Furthermore, because the session data is stored on the server, it doesn't have to be sent with every browser request.*

Starting a PHP Session

Before you can store any information in session variables, you must first start up the session. To begin a new session, simply call the PHP session_start() function. It will create a new session and generate a unique session ID for the user.

```
session_start()
```

The session_start() function first checks to see if a session already exists by looking for the presence of a session ID. If it finds one, i.e. if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID.

Storing and Accessing Session Data

You can 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. Consider the following script, which creates a new session and registers two session variables.

The PHP code in the **example** below simply starts a new session.- **Session.php**

```
<?php
        session_start();    // Start the session
?>
<!DOCTYPE html>
<html><body>
<?php// Set session variables
        $_SESSION["firstname"] = "Abc";
        $_SESSION["lastname"] = "Patel";
        echo "Session variables are set.";
?>
<h2><a href="AccessSession.php"> Home </a></h2>
</body></html>
```

To access the session data we set on our previous example from any other page on the same web domain — simply recreate the session by calling session_start() and then pass the corresponding key to the $_SESSION associative array.

Example: AccessSession.php

```
<?php
session_start();
// Accessing session data
echo 'Hi, ' . $_SESSION["firstname"] . ' ' . $_SESSION["lastname"]; ?>
```

Note: *To access the session data in the same page there is no need to recreate the session since it has been already started on the top of the page.*

Modify a PHP Session Variable

To change a session variable, just overwrite it.

Destroying a Session

If you want to remove certain session data, simply unset the corresponding key of the $_SESSION associative array, as shown in the following example:

```
<?php// Starting session
session_start();
if(isset($_SESSION["lastname"]))   // Removing session data
{
   unset($_SESSION["lastname"]);
}
?>
```

However, to destroy a session completely, simply call the session_destroy() function. This function does not need any argument and a single call destroys all the session data.

```
<?php
session_start();        // Starting session

session_destroy();              // Destroying session
?>
```

Every PHP session has a timeout value — a duration, measured in seconds — which determines how long a session should remain alive in the absence of any user activity. You can adjust this timeout duration by changing the value of session.gc_maxlifetime variable in the PHP configuration file (php.ini).

The PHP mail() Function

- Sending email messages are very common for a web application, for example, sending welcome email when a user create an account on your website, sending newsletters to your registered users, or getting user feedback or comment through website's contact form, and so on.
- You can use the PHP built-in mail() function for creating and sending email messages to one or more recipients dynamically from your PHP application either in a plain-text form or formatted HTML. The basic **syntax** of this function can be given with:

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

The following table summarizes the parameters of this function.

Parameter	Description
Required — The following parameters are required	
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.
Optional — The following parameters are optional	
headers	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	Used to pass additional parameters.

Sending Plain Text Emails

The simplest way to send an email with PHP is to send a text email. In the example below we first declare the variables — recipient's email address, subject line and message body — then we pass these variables to the mail() function to send the email.

Example:

```
<?php
$to  = „patelpalak@gmail.com';
$subject = 'Lunch Invitation';
$message = 'Hi palak, Can we go for Lunch?';
$from = 'patelpiyush@email.com';

if(mail($to, $subject, $message))                // Sending email
      echo 'Your mail has been sent successfully.';
else
      echo 'Unable to send email. Please try again.';
?>
```

Sending HTML Formatted Emails

When you send a text message using PHP, all the content will be treated as simple text. We're going to improve that output, and make the email into a HTML-formatted email.

To send an HTML email, the process will be the same. However, this time we need to provide additional headers as well as an HTML formatted message.

Example:

```
<?php
$to = 'maryjane@email.com';
$subject = 'Morning Greetings';
$from = 'peterparker@email.com';

// To send HTML mail, the Content-type header must be set
$headers  = 'MIME-Version: 1.0' . "\r\n";
$headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";

// Create email headers
$headers .= 'From: '.$from."\r\n".
   'Reply-To: '.$from."\r\n" .
   'X-Mailer: PHP/' . phpversion();

// Compose a simple HTML email message
$message = '<html><body>';
$message .= '<h1 style="color:#f40;">Hi Abc!</h1>';
$message .= '<p style="color:#080;font-size:18px;">How are you today?</p>';
$message .= '</body></html>';

if(mail($to, $subject, $message, $headers))   // Sending email
     echo 'Your mail has been sent successfully.';
else
     echo 'Unable to send email. Please try again.';
?>
```

Note: *However, the PHP mail() function is a part of the PHP core but you need to set up a mail server on your machine to make it really work.*

2.4 Forms: creating, handling, validation of forms, PHP filters, Json parsing

❖ PHP Forms: Creating, Handling, Validation Of Forms

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 format.

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:

1) <input>
2) <textarea>
3) <button>
4) <select>
5) <option>
6) <optgroup>
7) <fieldset>
8) <legend>
9) <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
month	use 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.

range	use for input fields that should contain value within 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 is also referring 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>

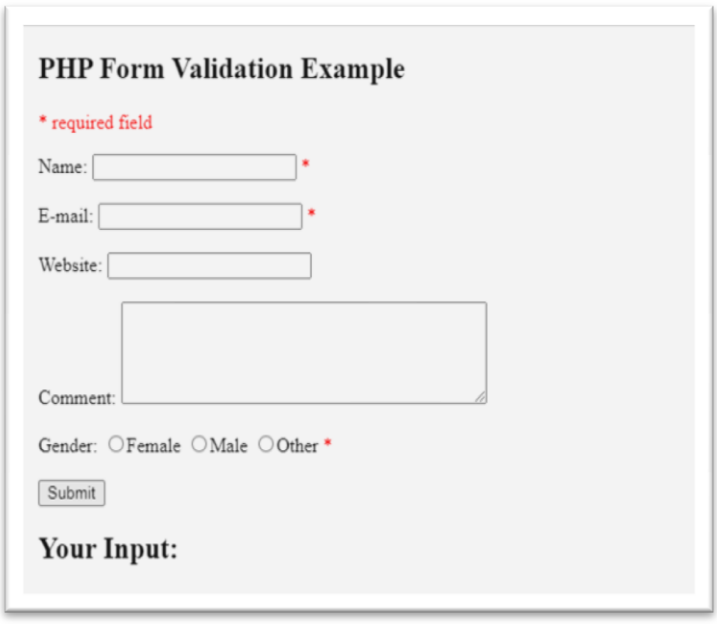
The <label> tag defines a label for an <input> element. The for attribute of the <label> tag should be equal to the id attribute of the related element to bind them together.

Attribute:

for: Specifies which form element a label is bound to

Exmaple:

The HTML form we will be working at in these chapters contains various input fields: required and optional text fields, radio buttons, and a submit button:



The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

When the form is submitted, the form data is sent with method="post".

What is the $_SERVER["PHP_SELF"] variable?

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars() function?

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!
If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data with PHP

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.
When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

```
<script>location.href('http://www.hacked.com')</script>
```

- this would not be executed, because it would be saved as HTML escaped code, like this:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Remove whitespaces from both sides of a string (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test_input().Now, we can check each $_POST variable with the test_input() function, and the script looks like this:

FileName: ValidateForm.php

```
<!DOCTYPE HTML>
<html><head>      <title>Validate Form Data With PHP  </title></head>
<body>
<?php
        // define variables and set to empty values
        $name = $email = $gender = $comment = $website = "";

        if($_SERVER["REQUEST_METHOD"] == "POST")
        {
          $name = test_input($_POST["name"]);
          $email = test_input($_POST["email"]);
          $website = test_input($_POST["website"]);
          $comment = test_input($_POST["comment"]);
          $gender = test_input($_POST["gender"]);
        }
        function test_input($data)
        {
          $data = trim($data);
          $data = stripslashes($data);
          $data = htmlspecialchars($data); //replace HTML characters like < and > with &lt; and &gt;
          return $data;
        }
?>
<h2>PHP Form Validation Example</h2>
<table border="2">
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"  >

          <tr><td>Name: </td><td><input type="text" name="name"></td></tr>
          <tr><td>E-mail:</td><td><input type="text" name="email"></td></tr>
          <tr><td>Website:</td><td><input type="text" name="website"></td></tr>
          <tr><td>Comment:</td><td><textarea name="comment" rows="5" cols="40"></textarea></td></tr>
          <tr><td>Gender: </td><td> <input type="radio" name="gender" value="female">Female
                                    <input type="radio" name="gender" value="male">Male
                                    <input type="radio" name="gender" value="other">Other</td>
         </tr>
         <tr><td><input type="reset" name="clear" value="Clear"> </td>
              <td><input type="submit" name="submit" value="Submit"> </td>
         </tr>
</form></table>
<?php
        echo "<h2>Your Input:</h2>";
        echo "Your Name is :".$name;
        echo "<br>";
```

```
        echo "Your Email is :".$email;
        echo "<br>";
        echo "Your Website is :".$website;
        echo "<br>";
        echo "Given Comment is :".$comment;
        echo "<br>";
        echo "Your gender is :".$gender;
?></body></html>
```

Notice that at the start of the script, we check whether the form has been submitted using $_SERVER["REQUEST_METHOD"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed.

PHP - Required Fields

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

In the previous chapter, all input fields were optional.

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

❖ **PHP - Display the Error Messages**

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

PHP Form Validation Example which Display the Error Messages

FileName: ValidateFormwithError.php

```
<!DOCTYPE HTML>
<html><head>
<style>
     .error {color: #FF0000;}
</style>
</head>
<body>
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);     }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);     }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]); }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);   }
}
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <input type="radio" name="gender" value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
```

```
</form>
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
</body></html>
```

❖ PHP Filters

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

The PHP Filter Extension

- PHP filters are used to validate and sanitize external input.
- The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.

The filter_list() function can be used to list what the PHP filter extension offers:

```
<!DOCTYPE html>
<html><head>
<style>
table, th, td
{
   border: 1px solid black;
   border-collapse: collapse;
}
th, td  {
  padding: 5px;}
</style></head>
<body>
<table>
<tr>
<td>Filter Name</td>
<td>Filter ID</td>
</tr>
<?php
  foreach(filter_list() as $id =>$filter) {
    echo '<tr><td>' . $filter . '</td><td>' . filter_id($filter) . '</td></tr>';
  }
?>
</table>
</body></html>
```

Why Use Filters?

Many web applications receive external input. External input/data can be:

- User input from a form
- Cookies
- Web services data
- Server variables
- Database query results

Note: You should always validate external data!
Invalid submitted data can lead to security problems and break your webpage!
By using PHP filters you can be sure your application gets the correct input!

PHP filter_var() Function

The filter_var() function both validate and sanitize data.

The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:

- The variable you want to check
- The type of check to use

Sanitize a String

The following example uses the filter_var() function to remove all HTML tags from a string:

Filename: Filter_var.php

```
<!DOCTYPE html>
<html><body>
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
</body></html>
```

Validate an Integer

The following example uses the filter_var() function to check if the variable $int is an integer. If $int is an integer, the output of the code below will be: "Integer is valid". If $int is not an integer, the output will be: "Integer is not valid":

```
<?php
$int = 100;
if (!filter_var($int, FILTER_VALIDATE_INT) === false)
        echo("Integer is valid");
else
        echo("Integer is not valid");
?>
```

Tip: filter_var() and Problem With 0

In the example above, if $int was set to 0, the function above will return "Integer is not valid". To solve this problem, use the code below:

```
<!DOCTYPE html>
<html><body>
<?php
$int = 0;
if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int, FILTER_VALIDATE_INT) === false)
      echo("Integer is valid");
else
```

```
  echo("Integer is not valid");
?></body></html>
```

Validate an IP Address

The following example uses the filter_var() function to check if the variable $ip is a valid IP address:

```
<!DOCTYPE html>
<html><body>
<?php
          $ip = "127.0.0.1";
        if (!filter_var($ip, FILTER_VALIDATE_IP) === false)
              echo("$ip is a valid IP address");
        else
               echo("$ip is not a valid IP address");
?>
</body></html>
```

Sanitize and Validate an Email Address

The following example uses the filter_var() function to first remove all illegal characters from the $email variable, then check if it is a valid email address:

```
<!DOCTYPE html>
<html><body>
<?php
$email = "abc@gmail.com";
// Remove all illegal characters from email
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false)     // Validate e-mail
    echo("$email is a valid email address");
else
     echo("$email is not a valid email address");
?>
</body></html>
```

Sanitize and Validate a URL

The following example uses the filter_var() function to first remove all illegal characters from a URL, then check if $url is a valid URL:

```
<!DOCTYPE html>
<html><body>
<?php
$url = "https://ucc.org ";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);

if (!filter_var($url, FILTER_VALIDATE_URL) === false) // Validate url
         echo("$url is a valid URL");
else
         echo("$url is not a valid URL");
?>
</body></html>
```

Validate an Integer Within a Range

The following example uses the filter_var() function to check if a variable is both of type INT, and between 1 and 200:

```
<!DOCTYPE html>
<html><body>
<?php
$int = 122;          /* variable to check */
$min = 1;           /* min value */
$max = 200;         /* max value */
if (filter_var($int, FILTER_VALIDATE_INT, array("options" => array("min_range"=>$min,
"max_range"=>$max))) === false)
          echo("Variable value is not within the legal range");
else
          echo("Variable value is within the legal range");
?>
</body></html>
```

Validate IPv6 Address

The following example uses the filter_var() function to check if the variable $ip is a valid IPv6 address:

```
<!DOCTYPE html>
<html><body>
<?php
  // Variable to check
  $ip = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";

// Validate ip as IPv6
if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6) === false)
        echo("$ip is a valid IPv6 address");
else
        echo("$ip is not a valid IPv6 address");
?>
</body></html>
```

❖ **Json parsing**

What is JSON?

JSON stands for ***JavaScript Object Notation***, and is syntax for storing and exchanging data.
Since the JSON format is a text-based format, it can easily be sent to and from a server, and used as a data format by any programming language.

PHP and JSON

PHP has some built-in functions to handle JSON.
First, we will look at the following two functions:

1. json_encode()
2. json_decode()

PHP - json_encode()

The json_encode() function is used to encode a value to JSON format.

Example: This example shows how to encode an associative array into a JSON object:

```
<!DOCTYPE html>
<html><body>
<?php
```

```
        $age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);
        echo json_encode($age);
?>
</body></html>
```

Example : This example shows how to encode an indexed array into a JSON array:

```
<!DOCTYPE html>
<html><body>
<?php
        $cars = array("Volvo", "BMW", "Toyota");
        echo json_encode($cars);
?>
</body></html>
```

PHP - json_decode()

The json_decode() function is used to decode a JSON object into a PHP object or an associative array.

```
<!DOCTYPE html>
<html><body>
<?php
          $jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';
          var_dump(json_decode($jsonobj));
          // var_dump(json_decode($jsonobj, true));
?>
</body></html>
```

OUTPUT:

object(stdClass)#1 (3) { ["Peter"]=> int(35) ["Ben"]=> int(37) ["Joe"]=> int(43) }

The json_decode() function returns an object by default. The json_decode() function has a second parameter, and when set to true, JSON objects are decoded into associative arrays.

OUTPUT:

array(3) { ["Peter"]=> int(35) ["Ben"]=> int(37) ["Joe"]=> int(43) }

PHP - Accessing the Decoded Values

Here are two examples of how to access the decoded values from an object and from an associative array:

Example: This example shows how to access the values from a PHP object:

```
<!DOCTYPE html>
<html><body>
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';
$obj = json_decode($jsonobj);
echo $obj->Peter;
echo $obj->Ben;
echo $obj->Joe;
?></body></html>
```

Example: This example shows how to access the values from a PHP associative array:

```
<!DOCTYPE html>
<html><body>
<?php
	$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';
	$arr = json_decode($jsonobj, true);
	echo $arr["Peter"];
	echo $arr["Ben"];
	echo $arr["Joe"];
?>
</body></html>
```

PHP - Looping Through the Values

You can also loop through the values with a foreach() loop:

Example: This example shows how to loop through the values of a PHP object:

```
<!DOCTYPE html><html><body>
<?php
	  $jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';
	  $obj = json_decode($jsonobj);
	 //shows how to loop through the values of a PHP associative array:
	//$arr = json_decode($jsonobj, true);
	foreach($obj as $key => $value)
	{
		echo $key . " => " . $value . "<br>";
	}
?>
</body></html>
```

2.5 Classes and objects in PHP

PHP What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

PHP - What are Classes and Objects?

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

Look at the following illustration to see the difference between class and objects:

Another example:

So, a class is a template for objects, and an object is an instance of a class.
When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.
Look at the next chapters to learn more about OOP.

PHP OOP - Classes and Objects

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

OOP Case

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.
When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

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 ({}). All its properties and methods go inside the braces:

Syntax:

```
<?php
class Fruit
{
    // code goes here...
}
?>
```

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set_name() and get_name() for setting and getting the $name property:

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

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

```
  {
    return $this->name;
  }
}
?>
```

Note: *In a class, variables are called properties and functions are called methods!*

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 are created using the new keyword.
- In the example below, $apple and $banana are instances of the class Fruit:

```
<!DOCTYPE html>
<html><body>
<?php
class Fruit
{
  public $name;                    // Properties
  public $color;
  function set_name($name)         // Methods
  {
      $this->name = $name;
  }
  function get_name()
  {
       return $this->name;
  }
}
$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');
echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
</body></html>
```

2.6 Regular expressions and exception handling

❖ **What is Regular Expression?**

- Regular Expressions, commonly known as "regex" or "RegExp", are a specially formatted text strings used to find patterns in text.
- Regular expressions are one of the most powerful tools available today for effective and efficient text processing and manipulations.
- For example, it can be used to verify whether the format of data i.e. name, email, phone number, etc. entered by the user was correct or not, find or replace matching string within text content, and so on.
- PHP (version 5.3 and above) supports Perl style regular expressions via its **preg_** family of functions.

- Why Perl style regular expressions? Because Perl (Practical Extraction and Report Language) was the first mainstream programming language that provided integrated support for regular expressions and it is well known for its strong support of regular expressions and its extraordinary text processing and manipulation capabilities.

Let's begin with a brief overview of the commonly used PHP's built-in pattern-matching functions.

Function	What it Does
preg_match()	Perform a regular expression match.
preg_match_all()	Perform a global regular expression match.
preg_replace()	Perform a regular expression search and replace.
preg_grep()	Returns the elements of the input array that matched the pattern.
preg_split()	Splits up a string into substrings using a regular expression.
preg_quote()	Quote regular expression characters found within a string.

Note: The PHP preg_match() function stops searching after it finds the first match, whereas the preg_match_all() function continues searching until the end of the string and find all possible matches instead of stopping at the first match.

Regular Expression Syntax

- Regular expression syntax includes the use of special characters (do not confuse with the HTML special characters).
- The characters that are given special meaning within a regular expression, are: . * ? + [ ] ( ) { } ^ $ | \. You will need to backslash these characters whenever you want to use them literally. For example, if you want to match ".", you'd have to write \.. All other characters automatically assume their literal meanings.

The following sections describe the various options available for formulating patterns:

Character Classes

- Square brackets surrounding a pattern of characters are called a character class e.g. [abc]. A character class always matches a single character out of a list of specified characters that means the expression [abc] matches only a, b or c character.
- Negated character classes can also be defined that match any character except those contained within the brackets. A negated character class is defined by placing a caret (^) symbol immediately after the opening bracket, like this [^abc].

You can also define a range of characters by using the hyphen (-) character inside a character class, like [0-9]. Let's look at some examples of character classes:

RegExp	What it Does
[abc]	Matches any one of the characters a, b, or c.
[^abc]	Matches any one character other than a, b, or c.
[a-z]	Matches any one character from lowercase a to lowercase z.
[A-Z]	Matches any one character from uppercase a to uppercase z.
[a-Z]	Matches any one character from lowercase a to uppercase Z.
[0-9]	Matches a single digit between 0 and 9.
[a-z0-9]	Matches a single character between a and z or between 0 and 9.

The following example will show you how to find whether a pattern exists in a string or not using the regular expression and PHP preg_match() function:

Example:

```
<?php
$pattern = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
if(preg_match($pattern, $text)){
   echo "Match found!";
} else{
     echo "Match not found.";
}?>
```

Similarly, you can use the preg_match_all() function to find all matches within a string:

```
<?php
$pattern = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
?>
```

Tip: Regular expressions aren't exclusive to PHP. Languages such as Java, Perl, Python, etc. use the same notation for finding patterns in text.

Predefined Character Classes

Some character classes such as digits, letters, and whitespaces are used so frequently that there are shortcut names for them. The following table lists those predefined character classes:

Shortcut	What it Does
.	Matches any single character except newline \n.
\d	matches any digit character. Same as [0-9]
\D	Matches any non-digit character. Same as [^0-9]
\s	Matches any whitespace character (space, tab, newline or carriage return character). Same as [\t\n\r]
\S	Matches any non-whitespace character. Same as [^ \t\n\r]
\w	Matches any word character (definned as a to z, A to Z,0 to 9, and the underscore). Same as [a-zA-Z_0-9]
\W	Matches any non-word character. Same as [^a-zA-Z_0-9]

The following **example** will show you how to find and replace space with a hyphen character in a string using regular expression and PHP preg_replace() function:

```
<?php
        $pattern = "/\s/";
        $replacement = "-";
        $text = "Earth revolves around\nthe\tSun";
       // Replace spaces, newlines and tabs
       echo preg_replace($pattern, $replacement, $text);
       echo "<br>";
       echo str_replace(" ", "-", $text); // Replace only spaces
?>
```

Repetition Quantifiers

In the previous section we've learnt how to match a single character in a variety of fashions. But what if you want to match on more than one character? For example, let's say you want to find out words

containing one or more instances of the letter p, or words containing at least two p's, and so on. This is where quantifiers come into play. With quantifiers you can specify how many times a character in a regular expression should match.

The following table lists the various ways to quantify a particular pattern:

RegExp	What it Does
p+	Matches one or more occurrences of the letter p.
p*	Matches zero or more occurrences of the letter p.
p?	Matches zero or one occurrences of the letter p.
p{2}	Matches exactly two occurrences of the letter p.
p{2,3}	Matches at least two occurrences of the letter p, but not more than three occurrences of the letter p.
p{2,}	Matches two or more occurrences of the letter p.
p{,3}	Matches at most three occurrences of the letter p

The regular expression in the following **example** will splits the string at comma, sequence of commas, whitespace, or combination thereof using the PHP preg_split() function:

```
<?php
$pattern = "/[\s,]+/";
$text = "My favourite colors are red, green and blue";
$parts = preg_split($pattern, $text);
foreach($parts as $part)   // Loop through parts array and display substrings
{
   echo $part . "<br>";
}
?>
```

Position Anchors

There are certain situations where you want to match at the beginning or end of a line, word, or string. To do this you can use anchors. Two common anchors are caret (^) which represent the start of the string, and the dollar ($) sign which represent the end of the string.

RegExp	What it Does
^p	Matches the letter p at the beginning of a line.
p$	Matches the letter p at the end of a line.

The regular expression in the following **example** will display only those names from the names array which start with the letter "J" using the PHP preg_grep() function:

```
<?php
$pattern = "/^J/";
$names = array("Jhon Carter", "Clark Kent", "John Rambo");
$matches = preg_grep($pattern, $names);
// Loop through matches array and display matched names
foreach($matches as $match){
   echo $match . "<br>";
}
?>
```

Pattern Modifiers

A pattern modifier allows you to control the way a pattern match is handled. Pattern modifiers are placed directly after the regular expression, for example, if you want to search for a pattern in a case-insensitive

manner, you can use the i modifier, like this: /pattern/i. The following table lists some of the most commonly used pattern modifiers.

Modifier	What it Does
i	Makes the match case-insensitive manner.
m	Changes the behavior of ^ and $ to match against a newline boundary (i.e. start or end of each line within a multiline string), instead of a string boundary.
g	Perform a global match i.e. finds all occurrences.
o	Evaluates the expression only once.
s	Changes the behavior of . (dot) to match all characters, including newlines.
x	Allows you to use whitespace and comments within a regular expression for clarity.

The following example will show you how to perform a global case-insensitive search using the i modifier and the PHP preg_match_all() function.

```
<?php
$pattern = "/color/i";
$text = "Color red is more visible than color blue in daylight.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
?>
```

Similarly, the following example shows how to match at the beginning of every line in a multi-line string using ^ anchor and m modifier with PHP preg_match_all() function.

```
$pattern = "/^color/im";
```

❖ **Exception Handling**

What is an Exception?

- An exception is a signal that indicates some sort of exceptional event or error has occurred. Exceptions can be caused due to various reasons, for example, database connection or query fails, file that you're trying to access doesn't exist, and so on.
- PHP provides a powerful exception handling mechanism that allows you to handle exceptions in a graceful way. As opposed to PHP's traditional error-handling system, exception handling is the object-oriented method for handling errors, which provides more controlled and flexible form of error reporting. Exception model was first introduced in PHP 5.

Syntax: The following syntax is used in exception handling to handle runtime errors -

```
<?php
        try {
            //code that can throw exception
        }
        catch (Exception $e) {
            //code to print exception caught in the block
        }
        finally {
            //any code that will always execute
        }
?>
```

Let's learn about try-throw-catch in more detail:

❖ **try**

The try block contains the code that may contain an exception. An exception raised in try block during runtime is caught by the catch block. Therefore, each try block must have at least one catch block. It consists of the block of code in which an exception can occur.

Following points needs to be noted about the try:

- The try block must be followed by a catch or finally block.
- A try block must have at least one catch block.
- There can be multiple catch blocks with one try block.

❖ **catch**

The catch block catches the exception raised in the try block. It contains the code to catch the exception, which is thrown by throw keyword in the try block. The catch block executes when a specific exception is thrown. PHP looks for the matching catch block and assigns the exception object to a variable.

Following points to be noted about the catch:

- There can be more than one catch block with a try.
- The thrown exception is caught and resolved by one or more catch.
- The catch block is always used with a try block. It cannot be used alone.
- It comes just after the try block.

❖ **throw**

It is a keyword, which is used to throw an exception. Note that one throw at least has one "catch block" to catch the exception.

It lists the exceptions thrown by function, which cannot be handled by the function itself.

❖ **finally**

- It is a block that contains the essential code of the program to execute. The finally block is also used for clean-up activity in PHP. It is similar to the catch block, which is used to handle exception. The only difference is that it always executes whether an exception is handled or not.
- The finally block can be specified after or in place of catch block. It always executes just after the try and catch block whether an exception has been thrown or not, and before the normal execution restarts. It is useful in the following scenarios - Closing of database connection, stream.

❖ **Using Throw and Try...Catch Statements**

In exception-based approach, program code is written in a try block, an exception can be thrown using the throw statement when an exceptional event occurs during the execution of code in a try block. It is then caught and resolved by one or more catch blocks.

The following **example** demonstrates how exception handling works:

```
<?php
function division($dividend, $divisor)
{
   // Throw exception if divisor is zero
   if($divisor == 0)
   {
      throw new Exception('Division by zero.');
   } else
   {
      $quotient = $dividend / $divisor;
      echo "<p>$dividend / $divisor = $quotient</p>";
   }
}
```

```
try{
   division(10, 2);
   division(30, -4);
   division(15, 0);
         // If exception is thrown following line won't execute
         echo '<p>All divisions performed successfully.</p>';
} catch(Exception $e){
   // Handle the exception
   echo "<p>Caught exception: " . $e->getMessage() . "</p>";
}
     // Continue execution
     echo "<p>Hello World!</p>";
?>
```

The PHP's Exception class also provides getCode(), getFile(), getLine() and getTraceAsString() methods that can be used to generate detailed debugging information.

```
<?php
// Turn off default error reporting
error_reporting(0);
 try{
   $file = "somefile.txt";

   $handle = fopen($file, "r");       // Attempt to open the file
   if(!$handle)
  {
      throw new Exception("Cannot open the file!", 5);
   }
     $content = fread($handle, filesize($file));     // Attempt to read the file contents

   if(!$content){
      throw new Exception("Could not read file!", 10);
   }
   // Closing the file handle
   fclose($handle);
      echo $content; // Display file contents
 }
 catch(Exception $e)
 {
   echo "<h3>Caught Exception!</h3>";
   echo "<p>Error message: " . $e->getMessage() . "</p>";
   echo "<p>File: " . $e->getFile() . "</p>";
   echo "<p>Line: " . $e->getLine() . "</p>";
   echo "<p>Error code: " . $e->getCode() . "</p>";
   echo "<p>Trace: " . $e->getTraceAsString() . "</p>";
 }
?>
```