What is PHP?

- PHP is an acronym for "PHP Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP costs nothing, it is free to download and use

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can restrict users to access some pages on your website
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource
- PHP is easy to learn and runs efficiently on the server side

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

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

```
<?php
// PHP code goes here
?>
```

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

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

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Eg:

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

Note: PHP statements are terminated by semicolon (;). The closing tag of a block of PHP code also automatically implies a semicolon (so you do not have to have a semicolon terminating the last line of a PHP block).

PHP Constants

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

Set a PHP Constant

To set a constant, use the define() function - it takes three parameters: The first parameter defines the name of the constant, the second parameter defines the value of the constant, and the optional third parameter specifies whether the constant name should be case-insensitive. Default is false.

The example below creates a **case-sensitive constant**, with the value of "Welcome to W3Schools.com!":

Eg:

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

the example below creates a **case-insensitive constant**, with the value of "Welcome to W3Schools.com!":

eg:

```
<?php
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>
```

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is editing the code!

Comments are useful for:

- To let others understand what you are doing - Comments let other programmers understand what you were doing in each step (if you work in a group)
- To remind yourself what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports three ways of commenting:

Eg: <!DOCTYPE html>
<html>
<body>

<?php
// This is a single line comment

This is also a single line comment

/*

```
This is a multiple lines comment block
that spans over more than
one line
*/
?>

</body>
</html>
```

PHP Case Sensitivity

In PHP, all user-defined functions, classes, and keywords (e.g. if, else, while, echo, etc.) are NOT case-sensitive.

In the example below, all three echo statements below are legal (and equal):

Eg:
```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>
```

However; in PHP, all variables are case-sensitive.

In the example below, only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables):

Eg:

```
<!DOCTYPE html>
<html>
<body>

<?php
$color="red";
echo "My car is " . $color . "<br>";
```

```
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>

</body>
</html>
```

PHP 5 Variables

Variables are "containers" for storing information:

Example

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

Much Like Algebra

```
x=5
y=6
z=x+y
```

In algebra we use letters (like x) to hold values (like 5).

From the expression z=x+y above, we can calculate the value of z to be 11.

In PHP these letters are called **variables.**

 Think of variables as containers for storing data.

PHP Variables

As with algebra, PHP variables can be used to hold values (x=5) or expressions (z=x+y).

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

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

 Remember that PHP variable names are case-sensitive!

Creating (Declaring) PHP Variables

PHP has no command for declaring a variable.

A variable is created the moment you first assign a value to it:

Example

```
<?php
$txt="Hello world!";
$x=5;
$y=10.5;
?>
```

After the execution of the statements above, the variable **txt** will hold the value **Hello world!**, the variable **x** will hold the value **5**, and the variable **y** will hold the value **10.5**.

Note: When you assign a text value to a variable, put quotes around the value.

PHP is a Loosely Type Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

PHP Variables Scope

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

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Local and Global Scope

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

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

The following example tests variables with local and global scope:

Example

```
<?php
$x=5; // global scope

function myTest() {
  $y=10; // local scope
  echo "<p>Test variables inside the function:</p>";
  echo "Variable x is: $x";
  echo "<br>";
  echo "Variable y is: $y";
}

myTest();

echo "<p>Test variables outside the function:</p>";
echo "Variable x is: $x";
echo "<br>";
echo "Variable y is: $y";
?>
```

In the example above there are two variables $x and $y and a function myTest(). $x is a global variable since it is declared outside the function and $y is a local variable since it is created inside the function.

When we output the values of the two variables inside the myTest() function, it prints the value of $y as it is the locally declared, but cannot print the value of $x since it is created outside the function.

Then, when we output the values of the two variables outside the myTest() function, it prints the value of $x, but cannot print the value of $y since it is a local variable and it is created inside the myTest() function.


You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

PHP The global Keyword

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

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

Example

```
<?php
$x=5;
$y=10;

function myTest() {
  global $x,$y;
  $y=$x+$y;
}

myTest();
echo $y; // outputs 15
?>
```

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

```
<?php
$x=5;
$y=10;

function myTest() {
  $GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

PHP The static Keyword

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

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

Example

```
<?php

function myTest() {
  static $x=0;
  echo $x;
  $x++;
}

myTest();
myTest();
myTest();

?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

?>

PHP 5 if...else...elseif Statements

Conditional statements are used to perform different actions based on different conditions.

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true
- **if...else statement** - executes some code if a condition is true and another code if the condition is false
- **if...elseif....else statement** - selects one of several blocks of code to be executed
- **switch statement** - selects one of many blocks of code to be executed

---

PHP - The if Statement

The if statement is used to execute some code **only if a specified condition is true**.

Syntax

```
if (condition) {
  code to be executed if condition is true;
}
```

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<?php
$t=date("H");

if ($t<"20") {
  echo "Have a good day!";
```

```
}
?>
```

PHP - The if...else Statement

Use the if....else statement to execute some code **if a condition is true and another code if the condition is false**.

Syntax

```
if (condition) {
  code to be executed if condition is true;
} else {
  code to be executed if condition is false;
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?php
$t=date("H");

if ($t<"20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>
```

PHP - The if...elseif....else Statement

Use the if....elseif...else statement to **select one of several blocks of code to be executed**.

Syntax

```
if (condition) {
  code to be executed if condition is true;
} elseif (condition) {
  code to be executed if condition is true;
} else {
  code to be executed if condition is false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php
$t=date("H");

if ($t<"10") {
  echo "Have a good morning!";
} elseif ($t<"20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>
```

PHP 5 switch Statement

The switch statement is used to perform different actions based on different conditions.

The PHP switch Statement

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
  case label1:
    code to be executed if n=label1;
    break;
  case label2:
    code to be executed if n=label2;
    break;
  case label3:
    code to be executed if n=label3;
    break;
  ...
  default:
```

```
  code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

Example

```
<?php
$favcolor="red";

switch ($favcolor) {
 case "red":
  echo "Your favorite color is red!";
  break;
 case "blue":
  echo "Your favorite color is blue!";
  break;
 case "green":
  echo "Your favorite color is green!";
  break;
 default:
  echo "Your favorite color is neither red, blue, or green!";
}
?>
```

PHP 5 while Loops

PHP while loops execute a block of code while the specified condition is true.

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

---

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
  code to be executed;
}
```

The example below first sets a variable $x to 1 ($x=1;). Then, the while loop will continue to run as long as $x is less than, or equal to 5. $x will increase by 1 each time the loop runs ($x++;):

Example

```
<?php
$x=1;

while($x<=5) {
  echo "The number is: $x <br>";
  $x++;
}
?>
```

The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
  code to be executed;
} while (condition is true);
```

The example below first sets a variable $x to 1 ($x=1;). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

```
<?php
$x=1;

do {
  echo "The number is: $x <br>";
  $x++;
} while ($x<=5);
?>
```

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition fails the first time.

The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

```
<?php
$x=6;

do {
  echo "The number is: $x <br>";
  $x++;
} while ($x<=5);
?>
```

PHP 5 for Loops

PHP for loops execute a block of code a specified number of times.

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
  code to be executed;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 10:

Example

```
<?php
for ($x=0; $x<=10; $x++) {
  echo "The number is: $x <br>";
}
?>
```

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
  code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

Example

```
<?php
$colors = array("red","green","blue","yellow");

foreach ($colors as $value) {
  echo "$value <br>";
}
?>
```

PHP 5 Data Types

String, Integer, Floating point numbers, Boolean, Array, Object, NULL.

PHP Strings

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

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

Example

```
<?php
$x = "Hello world!";
echo $x;
echo "<br>";
$x = 'Hello world!';
echo $x;
?>
```

PHP Integers

An integer is a number without decimals.

Rules for integers:

- An integer must have at least one digit (0-9)
- An integer cannot contain comma or blanks
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example we will test different numbers. The PHP var_dump() function returns the data type and value of variables:

Example

```
<?php
$x = 5985;
var_dump($x);
echo "<br>";
$x = -345; // negative number
var_dump($x);
echo "<br>";
$x = 0x8C; // hexadecimal number
var_dump($x);
echo "<br>";
$x = 047; // octal number
var_dump($x);
?>
```

PHP Floating Point Numbers

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

In the following example we will test different numbers. The PHP var_dump() function returns the data type and value of variables:

Example

```
<?php
$x = 10.365;
var_dump($x);
echo "<br>";
```

```
$x = 2.4e3;
var_dump($x);
echo "<br>";
$x = 8E-5;
var_dump($x);
?>
```

PHP Booleans

Booleans can be either TRUE or FALSE.

```
$x=true;
$y=false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Arrays

An array stores multiple values in one single variable.

In the following example we create an array, and then use the PHP var_dump() function to return the data type and value of the array:

Example

```
<?php
$cars=array("Volvo","BMW","Toyota");
var_dump($cars);
?>
```

PHP Objects

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

In PHP, an object must be explicitly declared.

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

We then define the data type in the object class, and then we use the data type in instances of that class:

Example

```
<?php
class Car
{
 var $color;
 function Car($color="green") {
  $this->color = $color;
 }
 function what_color() {
  return $this->color;
 }
}
?>
```

PHP NULL Value

The special NULL value represents that a variable has no value. NULL is the only possible value of data type NULL.

The NULL value identifies whether a variable is empty or not. Also useful to differentiate between the empty string and null values of databases.

Variables can be emptied by setting the value to NULL:

Example

```
<?php
$x="Hello world!";
$x=null;
var_dump($x);
?>
```

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

PHP User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName() {
  code to be executed;
}
```

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

 Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php
function writeMsg() {
  echo "Hello world!";
}

writeMsg(); // call the function
?>
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just seperate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {
  echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

Example

```
<?php
function familyName($fname,$year) {
  echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege","1975");
familyName("Stale","1978");
familyName("Kai Jim","1983");
?>
```

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?php
function setHeight($minheight=50) {
  echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

pHP Functions - Returning values

To let a function return a value, use the return statement:

Example

```
<?php
function sum($x,$y) {
  $z=$x+$y;
  return $z;
}

echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>";
echo "2 + 4 = " . sum(2,4);
?>
```

An array stores multiple values in one single variable:

Example

```
<?php
$cars=array("Volvo","BMW","Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Operators

Operators are used to perform operation on variables and values.

It can be divided as follows:

1. Arithmetic operators
2. Assignment operator
3. Comparison operators
4. Increment/decrement operators
5. Logical operators
6. String operator
7. Array operator

1) Arithmetic Operators

i) Addition(+) ex: $x+ $y
ii) Subtraction(-) ex: $x - $y
iii) Multiplication (*) ex: $ x * $y
iv) Division(/) ex: $ x / $y
v) Exponentiation (* *) ex: $x * * $y

2) Assignment Operator

i) = ex: $ x=5 value 5 is assigned to variable x

3) **Comparison operators**

i) ==equal ex: $x= $y returns true if x and y are equal.
ii) ===identical ex: $x===$yreturns true if $x is equal to $y and they are of the same type.
iii) <>or!=not equal ex:$x<>$y returns true if x and y are not equal.
iv) !==not identical ex: $x! == returns true if $x is not equal to $y and they are not of the same type.
v) >greater than ex: $x>$y returns true if x is greater than y.
vi) <lesser than ex: $x<$y returns true if x is lesser than y.
vii) >= greater than equal to ex: $x>=$y returns true if x is greater than equal to y.
viii) <= lesser than equal to ex: $x<$y returns true if x is lesser than equal to y.

4. **Increment /Decrement operators**

i) ++$x➔ Pre-increment

Ex: $a=++$x Increment the value of x and assigns to variable a.

ii) $x++➔ Post-increment

Ex: $a=$x ++ Assigns the value to variable a and gets incremented.

iii)--$x➔ Pre-decrement

Ex: $a=--$x Decrement the value of x and assigns to variable a.

iv)$x--➔ Post-decrement

Ex: $a=--$x Assigns the value to variable a and gets decremented .

5.Logical operators

AND, && --- $x and $y is True if both $x and $y are true.

OR , || --- $x or $y is True if either $x or $y is true.

NOT,!--- !$x is True if $x is not true.

XOR --- $x xor $y is True if either $x or $y is true, both not both.

6. String operators

. *Concatenation*

Ex: $txt1. $txt2

Concatenation of $tx1 and $tx2.

.= concatenation assignment

Ex: $tx1. = $tx2

Appends $tx1 to $tx2

7. Array Operators

Array operators are used with the array elements.

Operator	Name	Example	Result

+	Union	$x+$y	Union of $x and $y.
==	Equality	$x==$y	Returns true if $x and $y have the same key/value pairs.
===	Identity	$x===$y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types.
!=	Inequality	$x!=$y	Returns true if $x is not equal to $y.
<>	Inequality	$x<>$y	Returns true if $x is not equal to $y.
!==	Non-identity	$x!==$y	Returns true if $x is not identical to $y.

Functions:

The real power of php comes from its functions; it has more than 1000 built-in function.

Php user defined functions:

Besides the built-in-function, we can create our own function.

- A function is a block of statements that can be used repeatedly in program.
- A function will not execute immediately when page loads.
- A function will be executed by a call to the function

Creating a user defined function in php:

A user defined function declaration starts with the word "function"

General Syntax

Function functionname() {

Code to be executed;

}

NOTE: A function name can start with a letter or underscore (not a number), Function names are NOT case-sensitive.

Ex

<?php

Function writemsg(){

Echo” helloworld!”;

}

Writemsg();

}

?>

Php function arguments:

Information can be passed to the function through arguments;an argument is just like variables.

Arguments are specified after the function name inside the parenthesis,we can add as many arguments as we want, just separate them with a comma.

Ex: **Function with one argument** ($fname)

<?php

Function greetingcard ($greeting)

{

Echo ”$greeting Sam
”;

}

greetingcard (“hello”);

greetingcard (“hai”);

greetingcard (“welcome”);

?>

o/p:

hello Sam

hai Sam

welcome Sam

Ex: The function with two arguments ($fname and $year)

<?php

Function Familyname($fname,$year){

Echo “$fname, born in $year
”;

}

Familyname(“Jai”,”1996”);

Familyname(“Raj”,”1997”);

Familyname(“Sam”,”1999”);

?>

o/p:

Jai born in 1996.

Raj born in 1997.

Sam born in 1999.

Functions with default argument values

The function uses default parameter. If we call the function setheight () without arguments it takes the default value as argument.

Ex

<?php

Function setheight($minheight = 50)

{

Echo” the height is : $minheight
”;

Setheight(350);

Setheight ();//will use the default value 50

Setheight(135);

?>

o/p:

350

50

135

Functions with return statements

To let a function return a value, use return statements.

<?php

Function sum($x,$y) {

$z=$x+$y;

Return $z;

}

Echo”5+10”.sum(5,10).”
”;

Echo”7+13”.sum(7,13).”
”;

?>

Function with Call by Reference

It is possible to pass arguments to function by address.

<?php

Function addfive($num)

```
{
$num+=5;
}
Function addsix($num)
{
$num+=6;
}
$orginum=10;
addfive(&$orginum);
Echo "The output of addfive is.$orginum<br>";
addsix(&$orginum);
Echo "The output of addsix is.$orginum";
?>
```

o/p:

The output of addfive is 15
The output of addsix is 21

UNIT-II

Printing String functions

Print

The print() function outputs one or more strings.

Note: The print() function is not actually a function, so you are not required to use parentheses with it.

Tip: The print() function is slightly slower than echo().

Ex:

```
<?php
print "Hello world!";
?>
```

o/p

Hello world!

Print_r

print_r used to print array where as echo is used to give output the data on screen.

Print_r() is used for printing the array in human readable format.

Print has a return value of 1 so it can be used in expressions.

Ex:

```
<?php
$age=array("ram"=>"22","kumar"=>"19","jai"=>"25");
print_r($age);
?>
```

o/p

Array ([ram] => 22 [kumar] => 19 [jai] => 25)

printf

The printf() function outputs a formatted string.

The arg1, arg2, ++ parameters will be inserted at percent (%) signs in the main string. This function works "step-by-step". At the first % sign, arg1 is inserted, at the second % sign, arg2 is inserted, etc.

syntax

printf(format ,arg1, arg2, arg3....)

Description

Each conversion specification starts with a single percent sign (%) and ends with the following conversion characters.

% - returns a percent sign.
b - the argument is treated as an integer and display it as a binary number.

c - the argument is treated as an integer and display it as a an ASCII value.
d - the argument is treated as an integer and display as a signed decimal number.
e - the argument is treated as scientific notation (e.g. 1.2e+2).
E - the argument treated as scientific notation (e.g.1.2E+2).
u - the argument is treated as an integer, and display as an unsigned decimal number.
f- the argument is treated as a float, and display as a floating-point number. (local aware)
F - the argument is treated as a float, and display as a floating-point number (non-locale aware).
g - shorter of %e and %f.
G - shorter of %E and %f.
o- the argument is treated as an integer, and display as an octal number.
s - the argument is treated as string and display as a string.
x - the argument is treated as an integer and display as a hexadecimal number (with lowercase letters).
X - the argument is treated as an integer and display as a hexadecimal number (with uppercase letters).

Ex:

```
<?php

$age=22;

$str = "john";

printf("%s age is %u",$str,$age);

?>
```

o/p

```
<?php

$age=22;

$str = "john";

printf("%s age is %u",$str,$age);

?>
```

o/p

John age is 22

Echo

The echo() function outputs one or more strings

Note: The echo() function is not actually a function, so you are not required to use parentheses with it. However, if you want to pass more than one parameter to echo(), using parentheses will generate a parse error.

echo has no return value

echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

Ex:

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

o/p

Hello world!

Var dump

The var_dump() function is used to dump information about a variable. This function displays structured information such as type and value of the given variable.

Ex:

```
<?php
$cars=array("addi","benz","bmw");
var_dump($cars);
?>
```

o/p

```
array
  0 => string 'addi' (length=4)
  1 => string 'benz' (length=4)
  2 => string 'bmw' (length=3)
```

Strings functions

trim()

> The **trim**() function removes whitespace and other predefined characters from both sides of a string.

trim($string, $charlist)

Ex:

```
<?php
 $str = "Hello World!";
echo trim($str, "H!");
?>
```

o/p: ello World

Ltrim()

ltrim() - Removes whitespace or other predefined characters from the left side of a string.

Ex:

```
<?php
 $str = "Hello World!";
echoltrim($str, "Helo!");
?>
```

o/p: World!

Rtrim()

rtrim() - Removes whitespace or other predefined characters from the right side of a string.

Ex:

```
<?php
 $str = "Hello World!";
echortrim($str, "ld!");
?>
```

o/p: Hello Wor

strtolower( )

The strtolower() function is used to convert the given string from upper case to lower case.

Ex:

```
<?php
echo strtolower(“BCA”);
?>
```

o/p: bca

strtoupper( )

The strtoupper() function is used to convert the given string from lower case to upper case.

Ex:

```
<?php
echo   strtoupper(“bca”);
?>
```

o/p: BCA

ucfirst( )

The ucfirst() function converts the first character into uppercase.

Ex:

```
<?php
$str = "bca";
echo ucfirst($str)
?>
```

o/p: Bca

ucwords( )

The ucwords () function converts first character of all words into uppercase.

Ex:

```
<?php
$str = "st josephs";
echoucwords($str)
?>
```

o/p: St Josephs

strpos( )

The strpos() function is used to find a position of a character in the given string.

Ex:

```
<?php
Echo strpos(“helloworld”,”w”);
?>
```

o/p: 5

substr( )

The substr( ) function is used to select the part of a word from a given string.

Ex:

```
<?php
echosubstr(“helloworld”,”5”,”3”);
?>
```

o/p: wor

chartocode( )

chartocode() function converts the character to code.

Ex:

```
<?php
$str = "a";
echo ord($str)
?>
```

o/p : 97

strlen( )

The strlen() function returns the length of a string (total number of characters)

Ex:

```
<?php
Echo strlen(“hello world”);
?>
```

o/p: l2

strrev( )

The strrev() function reverses the given string.

Ex:

```
<?php
Echo strrev(“helloworld”);
?>
```

o/p:

dlrowolleh

str_word_count( )

The str_word_count() function counts the number of words present in a given string.

Ex:

```
<?php
Echo str_word_count(“hello world”);
?>
```

o/p: 2

Strcmp()

The strcmp() function is used to compare the two given strings are equal.

Ex:

```
<?php
        $str1=”hello”;
        $str2=”hello”;
  If (strcmp($str1,$str2)==0)
         Echo “the given strings are equal”;
Else
     Echo “ the given strings are not equal”;
?>
```

o/p: The given strings are equal;

Strcasecmp();

The strcasecmp() function is used to compare the two given strings are equal and with caseinsenstive.

Ex:

```
<?php
        $str1=”hello”;
        $str2=”HEllo”;
  If (strcasecmp($str1,$str2)==0)
0       Echo “the given strings are equal”;
Else
    Echo “ the given strings are not equal”;

?>
```

o/p: The given strings are equal;

str_replace( )

The str_replace function is used to replace the text within a given string.

Ex:

<?php

Echo str_replace(“world”,”dolly”,”hello world”);

?>

o/p: hello dolly

ARRAYS

In php, there are three types of arrays:

i) Indexed arrays. (Arrays with numeric index).
ii) Associative arrays.(Arrays with named keys).
iii) Multi-dimensional arrays.(Arrays containing one or more arrays).

Indexed arrays

There are two ways to create indexed arrays.

Type:1

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

$cars=array(“volvo”,”bmw”,”toyoto”);

Type:2

The index can be assigned manually.

$cars[0]=”volvo”;

$cars[1]=”bmw”;

$cars[2]=”toyoto”;

Ex

<?php

$cars=array(“volvo”,”bmw”,”toyoto”);

Echo “I like “.$cars[0].”,”.$cars[1]. “and”.$cars[2].”.”;

?>

o/p:

I like volvo,bmw and toyoto.

Getting the length of an array

Ex:

<?php

$cars=array(“volvo”,”bmw”,”toyoto”);

Echo count($cars);

?>

Loop through an indexed array

<?php

$cars=array(“volvo”,”bmw”,”toyoto”);

$arrlength=count($cars);

for($x=0;$x<$arrlength;$x++)

{

Echo $cars[x];

Echo “
”;

}

?>

o/p:

volvo

bmw

toyoto

Php Associative arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array.

$age=array(“peter”=>”35”,”ben”=>”37”,”joe”=>”43”);

Or

$age[“peter”]=”35”;

$age[“ben”]=”37”;

$age[“joe”]=”43”;

Ex 1:

```
<?php
$age=array(“peter”=>”35”,”ben”=>”37”,”joe”=>”43”);
Echo”peter is”.$age[‘peter’].”years old”;
?>
```

o/p

peter is 35 years old.

Loop through an associative array

Ex 2:

```
<?php
$age=array(“peter”=>”35”,”ben”=>”37”,”joe”=>”43”);
Foreach($age as $x=> $x_value)
{
Echo “key=”.$x.”,value=”.$x_value;
}
Echo”<br>”;
?>
```

o/p:

key=peter,value=35

key=ben,value=37

key=joe,value=43

Multi-Dimensional (Two dimensional arrays)

A two dimensional array is array of arrays (a three-dimensional array is an array of arrays).

Name	Stock	Sold
Volvo	22	15
Bmw	15	13
Toyoto	25	18
Suzuki	7	3

```
<?php
$cars=array (array (“Volvo”, 22,18)
array(“bmw”,15,13)
array(“toyoto”,25,18)
array(“suzuki”,7,3));
Echo $cars[0] [0].”:Instock:”.$cars[0][1].”,sold.”$cars[0][2].”<br>;
Echo $cars[1] [0].”:Instock:”.$cars[1][1].”,sold.”$cars[1][2].”<br>;
Echo $cars[2] [0].”:Instock:”.$cars[2][1].”,sold.”$cars[2][2].”<br>;
?>
```

o/p:

volvo:instock:22,sold:18

bmw:instock:15,sold:13

toyoto:instock:25,sold:18

suzuki:instock:7,sold:3

For loop to get the elements of $cars array

```
<?php
$cars=array (array (“Volvo”, 22,18)
array(“bmw”,15,13)
```

```
array(“toyoto”,25,18)
array(“suzuki”,7,3));
For ($row=0;$row<4;$row++)
{
Echo”<p><b>row number $row</b></p>”;
Echo”<ul>”;
For($col=0;$col<3;$col++)
{
Echo”<li>”.$cars[$row][$col].”<li>”;
}
Echo”</ul>”;
?>
```

o/p:

Row number 0

Volvo

22

18

Row number 1

Bmw

15

13

Row number 2

Toyoto

25

18

Sorting Array Functions

sort()----Sorting an indexed array in Ascending order

Ex:

```
<?php
$cars=array("Maruthi","BMW","Volvo","Nissan","Ford");
$len=count($cars);
echo " THE UNSORTED ARRAY <br>";
for($x=0;$x<$len;$x++)
{
echo $cars[$x]."<br>";
}
sort($cars);
echo "The Sorted Indexed Array in Ascending order <br>";
for($x=0;$x<$len;$x++)
{
echo $cars[$x]."<br>";
}
```

rsort()----Sorting an indexed array in Descending order

Ex:

```
echo " The Sorted  Indexed Array in Descending order <br>";
rsort($cars);
foreach($cars as $x)
{
echo "$x <br>";
}
?>
```

asort()----Sorting an associative array in Ascending order based on values.

Ex:

```
<?php
$age = array("Peter"=>"35", "Ben"=>"10", "Joe"=>"43","Jack"=>"5","Rose"=>"12");
echo " THE UNSORTED ARRAY<BR>";
foreach($age as $x => $x_value)
{
echo "Key => " . $x . "Value => " . $x_value;
echo "<br>";
}
```

```
asort($age);
echo " The Sorted  Associative array in Ascending order based on values <br>";
foreach($age as $x => $x_value) {
echo “Key => " . $x . “,Value = > ". $x_value;
echo "<br>";
}
```

ksort()----Sorting an associative array in ascending order based on keys

Ex:

```
ksort($age);
echo "The Sorted associative array in ascending order based on keys </br>";
foreach($age as $x => $x_value) {
echo "Key = " . $x . ",Value = " . $x_value;
echo "<br>";
}
```

arsort()----Sorting an associative array in Descending order based on values.

Ex:

```
arsort($age);
echo " The Sorted  associative array in Descending order based on values</br> ";
foreach($age as $x => $x_value) {
echo "Key = " . $x . ", Value = " . $x_value;
echo "<br>";
}
```

krsort()----Sorting an associative array in Descending order based on Keys

Ex:

```
krsort($age);
echo "<br>";
echo "The Sorted associative array in Descending order based on Keys <br>";
foreach($age as $x => $x_value) {
echo "Key=" . $x . ", Value=" . $x_value
echo "<br>";
```

}
?>

usort() -The usort() function in PHP sorts a given indexed array by using user-defined comparison function.

boolean usort($array, "function_name");

Parameters: This function accepts two parameters as shown in the above syntax and are described below:

1. **$array**: This parameter specifies the array which u want to sort.
2. **function_name** : This parameter specifies the name of a user-defined function which compares the values and sort the array specified by the parameter *$array*. This function returns an integer value based on the following conditions. If two argument are equal then it returns 0, If first argument is greater than second, it returns 1 and if first argument is smaller than second, it returns -1.

   Return Value: This function returns boolean type of value. It returns TRUE in case of success and FALSE in case of failure.

Ex:

```
<?php

   functioncompfunc( $x, $y)
   {
      if ($x== $y)
         return 0;

if ($x < $y)
         return -1;
      else
         return 1;
   }

   // Input array
   $arr= array(2, 9, 1, 3, 5);

   usort($arr, ‘compfunc’);

print_r($arr);

?>
```

Output:

```
Array
(
   [0] => 1
```

```
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 9
)
```

uasort() -The uasort() function is a built in function in PHP and is used to sort an associative array based on values,such that array indices maintain their correlation with the array elements they are associated with, using a user-defined comparison function.

Syntax:

boolean uasort(array_name, user_defined_function);

Parameter: This function accepts two parameters and are described below:

array_name: This parameter represents the array which we need to sort.

user_defined_function: This is a comparator function and is used to compare values and sort the array. This function returns three types of values described below:

It return 0 when a=b

It return 1 when a>b and we want to sort input array in ascending order otherwise it will return -1 if we want to sort input array in descending order.

It return -1 when a<b and we want to sort input array in ascending order otherwise it will return 1 if we want to sort input array in descending order.

Return Value: It returns a boolean value, i.e. either TRUE on sucess and FALSE on failure.

Ex:

Sorting in ascending order: To sort the input array in ascending order, in the comparator function we will return 1 when a>b or -1 when a<b.

```
<?php
function sorting($a,$b)
{
if ($a==$b) return 0;
return ($a<$b)?-1:1;
}

// input array
$arr=array("a"=>4,"b"=>2,"g"=>8,"d"=>6,"e"=>1,"f"=>9);

uasort($arr,"sorting");

// printing sorted array.
print_r($arr);
```

```
?>
```
Output:
```
Array
(
    [e] => 1
    [b] => 2
    [a] => 4
    [d] => 6
    [g] => 8
    [f] => 9
)
```
Sorting in descending order: To sort the input array in descending order, in the comparator function we will return -1 when a>b or 1 when a<b. Below program illustrates this:

```
<?php
function sorting($a, $b)
{
if ($a == $b) return 0;
return ($a > $b) ? -1 : 1;
}

// input array
$input = array("d"=>"R", "a"=>"G", "b"=>"X", "f"=>"Z" );

uasort($input, "sorting");

// printing sorted array.
print_r($input);

?>
```
Output:
```
Array
(
    [f] => Z
    [b] => X
    [d] => R
    [a] => G
)
```

uksort( )— Sort an array by keys using a user-defined comparison function.

Ex:

Sorting in ascending order: To sort the input array in ascending order, in the comparator function we will return 1 when a>b or -1 when a<b.

```
<?php
function sorting($a,$b)
{
if ($a==$b) return 0;
return ($a<$b)?-1:1;
}

// input array
$arr=array("a"=>4,"b"=>2,"g"=>8,"d"=>6,"e"=>1,"f"=>9);

uksort($arr,"sorting");

// printing sorted array.
print_r($arr);

?>
```

Output:

```
Array
(
    [a] => 4
    [b] => 2
    [d] => 6
    [e] => 1
    [f] => 9
    [g] => 8
)
```

Sorting in descending order: To sort the input array in descending order, in the comparator function we will return -1 when a>b or 1 when a<b. Below program illustrates this:

```
<?php
function sorting($a, $b)
{
if ($a == $b) return 0;
return ($a > $b) ? -1 : 1;
}

// input array
$input = array("d"=>"R", "a"=>"G", "b"=>"X", "f"=>"Z" );

uasort($input, "sorting");
```

```
// printing sorted array.
print_r($input);

?>
```

Output:

```
Array
(
    [f] => Z
    [d] => R
    [b] => X
    [a] => G
)
```

OOPS in PHP

Class

Class :This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

Member Variable : These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

Member function : These are the function defined inside a class and are used to access object data.

Ex:

Defining PHP Classes

```
<?php
class Books {
     /* Member variables */
 $title;

 /* Member functions */

function  setTitle($par){
        $this->title = $par;
       }

function getTitle(){
echo $this->title ." <br/>";
     }
  }
```

```
?>
```

Object : An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.

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

Creating Objects in PHP

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

```
$chemistry= new Books;
$maths = new Books;
```

Here we have created two objects and these objects are independent of each other and they will have their existence separately.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.

```
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );
```

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

```
$chemistry->getTitle();
$maths->getTitle();
```

Inheritance

When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

Parent class − A class that is inherited from by another class. This is also called a base class or super class.

Child Class − A class that inherits from another class. This is also called a subclass or derived class.

Inheritance is very useful if we want to create several similar classes. We can put the common properties and methods in one parent class and then inherit it in the child classes.

Some important points to remember while using inheritance are:

1. Child class can access and use only the **non-private** properties and methods on the parent class.
2. Child class can have its own methods too, which will not be available to the parent class.
3. Child class can also override a method defined in the parent class and provide its own implementation for it.

Ex:

```
<?php
class person
{
function setname($par){
echo"<br>".$this->name = $par ;
    }}
class male extends person
{
function  malename()
{
echo "</br>  i am male".$this-> name;
}}
class female extends person
{
function  femalename()
{
echo "</br>  i am female".$this-> name;
}}
$person1=new male;
$person1->setname("kumar");
```

```
$person1->malename();


$person2 = new female;
$person2->setname("jeni");
$person2->femalename();
?>
```

Constructor

refers to a special type of function which will be called automatically whenever there is an object formation from a class.

When you create a new object, it is useful to initialize its properties.PHP provides you with a special method to help initialize object's properties called *constructor*.

To add a constructor to a class, you simply add a special method with the name __construct().

Ex:

```
<?php
class person
{
function  getname(){
echo $this->name ."<br/>";
    }
function   getage(){
echo $this->age ." <br/>";
    }
function __construct( $par1, $par2 ) {
  $this->name = $par1;
  $this->age = $par2;
```

```
}
}
$person1= new person("sam",25);
$person1->getname();
$person1->getage();
?>
```

UNIT-III

BUILT IN FUNCTIONS IN PHP

Mathematical functions:

Floor( )

Round numbers down to the nearest integer:

Syntax

floor(*number*);

number Required. Specifies the value to round down

Ex:

```
<?php
$val=2.3;
$val1=-2.3
```

```
echo "The floor  of $val is ".floor($val)."</br>";

echo "The floor of $val is ".floor($val1);

?>
```

O/P: The flor of 2.3 is 2
The floor of -2.3 is 3

Fmod( )

The fmod() function returns the remainder (modulo) of x/y.

Syntax:

fmod(*x*,*y*);

Ex:

```
<?php
$x = 7;
$y = 2;
$result = fmod($x,$y);
echo $result;
?>
```

O/p: 1

Pow( )

The pow() function in PHP is used to calculate a **base** raised to the power of exponent. It is a generic function which can be used with number raised to any value.It takes two **parameters** which are the **base** and exponent and returns the desired answer.

Syntax

pow(*x*,*y*)

x	Required. Specifies the base to use
y	Required. Specifies the exponent

Ex:

```
<?php
$val= 3;

$pow= 3;

echo “The power of  $val is ”.pow($val,$pow);
```

?>
O/P: The power of 3 is 27

Round( )

The round() function rounds a floating-point number.

Syntax

round(*number,precision,mode*);

numberRequired. Specifies the value to round
precision Optional. Specifies the number of decimal digits to round to. Default is 0

```
<?php

$val=23.6;

echo "The round value of $val is ".round($val);

?>
```

O/P:

The round value of 23.6 is 24.

Rand( )

The rand() function generates a random integer.

Syntax

rand(); or rand(*min,max*);

Ex:

```
<?php

$val1=1;

$val2=50;

echo "The Random value of $val1 to $val2 is ".rand($val1,$val2);

?>
```

O/P: The Random value of 1 to 50 is 41**.**

Sqrt( );

The sqrt() function returns the square root of a number.

Syntax

sqrt(*number*);

Ex:
<?php

$val=64;

echo "The square root of $val is ".sqrt($val);

?>
O/P: The square root of 64 is 8.

Max( );
The max() function returns the highest value in an array, or the highest value of several specified values.

Syntax

max(*array_values*);ormax(*value1,value2,...*);

Ex:
<?php
$val1=10;
$val2=8;
$val3=15;
$val4=25;
$val5=20;
echo "The Greatest of $val1,$val2,$val3,$val4,$val5 is".max($val1,$val2,$val3,$val4,$val5);
?>

O/P: The Greatest of 10,8,15,25,20 is 25

Min( );
The min() function returns the lowest value in an array, or the lowest value of several specified values.

Syntax

min(*array_values*);ormin(*value1,value2,...*)

array_values Required. Specifies an array containing the values
value1,value2,... Required. Specifies the values to compare (must be at least two values)

Ex:

```
<?php
$val1=10;
$val2=8;
$val3=15;
$val4=25;
$val5=20;
echo "The Lowest of $val1,$val2,$val3,$val4,$val5 is ".min($val1,$val2,$val3,$val4,$val5);
?>
```

O/P:

The Lowest of 10,8,15,25,20 is 8

Log( );

The log() function returns the natural logarithm of a number, or the logarithm of *number* to *base*.

Syntax

log(*number,base*);

number Required. Specifies the value to calculate the logarithm for
base Optional. The logarithmic base to use. Default is 'e'

Ex:

```
<?php
   echo(log(2.7183) . "<br>");
   echo(log(2) . "<br>");
   echo(log(1) . "<br>");
   ?>
```

o/p:

1.000006684914

0.69314718055995

0

Hexadec( ) ;

The *hexdec()* function in PHP converts a hexadecimal number to a decimal number.

Syntax:

hexdec($value);

Ex:

```
<?php

$val=76;

echo "The hexadecimal value of $val is ".hexdec($val1)."</br>";

?>
```

O/P: The hexadecimal value of is 118.

Date and Time Functions

Date and time:

The date () function is used to format a date and /or a time

Syntax

Date (format)-mandatory

d-represents of the date/day of a month in number(01 to 31)

D- a textual representation of a day(three letters).

l-a textual representation of the day.

m-represents a month in number(01 to 12).

M- a short textual representation of the month(three letters)

F- a full textual representation of a month.

y-a two digit representation of the year.

Y- a four digit representation of the year.

h-12 hour format of an hour with leading zeros(01 to 12).

H-24 hour format of an hour with leading zeros (00 to 23).

g -12 hour format of an hour (1 to 12)

G-24 hour format of an hour (0 to 23)

i-minutes with leading zeros (00 to 59)

s-seconds with leading zeros (00 to 59)

a – lowercase am or pm.

A – uppercase Am or PM.

other characters like “/”,”.”, or”_” can also be inserted between the characters

Timestamp(Optional)

This is an integer Unix timestamp that defaults to the current local time if a timestamp is not given

Ex:

```
<html>

<body>

<?php

echo "Today is " . date("y/m/d") . "<br>";

echo "Today is " . date("Y.M.D") . "<br>";

echo "Today is " . date("Y-F-d") . "<br>";

echo "Today is " . date("l")     . "<br>";

echo "Time is "  . date("h:i:sa"). "<br>";


// Setting the default time zone

date_default_timezone_set("Asia/Kolkata");

echo "Time is "  . date("H:i:sA"). "<br>";

// To create a date with syntax mktime(hour,minute,second,month,day,year)

$d = mktime(12,15,57,02,22,2016);

echo "The time created is".date("y-m-d h:i:s",$d);

echo "<br>";
```

// To create a date from a string with strtotime(time,now)

```
$da=strtotime("tomorrow");
echo "The tomorrow date is".date("y-m-d h:i:sa",$da)."<br>";

$dy=strtotime("10:30:00 pm april 14 2016");
echo "The date is created is".date("y-m-d h:i:sa",$dy)."<br>";

$dr=strtotime("next friday");
echo "The comming Friday date is".date("y-m-d h:i:sa",$dr)."<br>";

$mon=strtotime("+3 months");
echo "The date after three months is ".date("y-m-d h:i:sa",$mon)."<br>";
?>
```

O/P

```
Today is 19/03/11
Today is 2019.Mar.Mon
Today is 2019-March-11
Today is Monday
Time is 06:21:18pm
Time is 23:51:18PM
The time created is16-02-22 12:15:57
The tomorrow date is19-03-12 12:00:00am
The date is created is16-04-14 10:30:00pm
The coming  Friday date is 19-03-15 12:00:00am
The date after three months is 19-06-11 11:51:18pm
```

Handling files

It is an important part of any web application, it is often need to open and process a file for different tasks.

Php manipulating files

Php has several functions for creating, reading, updating and editing files.

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

R➔Opens a file for read only (Read)

File pointer starts at the beginning of the file.

W➔Open a file for write only (Write)

Erases the contents of a file or creates a new file if it does not exist, file pointer starts at the beginning of the file.

A➔open a file for write only (Append)

The existing data in file is preserved, File pointer starts at the end of the file. It creates a new file if the file does not exist.

X➔creates a new file for write only

Returns FALSE and an error if file already exist.

Rt➔open a file for read/write

File pointer starts at the beginning of the file.

Wt➔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.

At➔open a file for read/write

The existing data in a file is preserved, File pointer starts at the end of the file. Creates a new file if the file doesn't exist.

Xt➔creates a new file for read /write

Returns FALSE and an error if file already exists.

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.

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:

Ex:

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

PHP fread() Function

The fread() reads from an open file.

The function will stop at the end of the file or when it reaches the specified length, whichever comes first.

This function returns the read string, or FALSE on failure.

Syntax:

fread(file,length)

file Required. Specifies the open file to read from

length Required. Specifies the maximum number of bytes to read

Ex:

```
<?php
$file = fopen("test.txt","r");
fread($file,"10");
fclose($file);
?>
```

PHP fwrite() Function

The fwrite() writes to an open file.

The function will stop at the end of the file or when it reaches the specified length, whichever comes first.

This function returns the number of bytes written, or FALSE on failure.

Syntax

fwrite(file,string,length)

file	Required. Specifies the open file to write to
string	Required. Specifies the string to write to the open file
length	Optional. Specifies the maximum number of bytes to write.

Include( ) or Require( )

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.

Syntax

include '*filename*';orrequire '*filename*';

header.html

```
<html><body>

<font color="green" ><h1>Welcome to Include function</h1></font>

</body></html>
```

Footer .htnl

```
<html><body>

<font color="red"><h1> THE END</h1></font>

</body></html>
```

Var.php

```
<?php

$color='red';
```

```
$car='bmw';
?>
```

Include.php

```
<?php
include 'heading.html';
include 'heading.html';
include 'var.php';
echo "My favourtecolor is $color</br>";
echo "My favourte car is $car<br>";
include 'footer.html';
?>
```

o/p:

Welcome to Include function

My favourtecolor is red
Myfavourte car is bmw
THE END

Fclose( );

The fclose() function closes an open file.

This function returns TRUE on success or FALSE on failure.

Syntax

fclose(file)

file Required. Specifies the file to close

Ex:

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

PHP unlink() Function

The unlink() function deletes a file.

This function returns TRUE on success, or FALSE on failure.

Syntax

unlink(filename,context)

filename Required. Specifies the file to delete
contextOptional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Ex:

```
<?php
$file = "test.txt";
if (!unlink($file))
  {
  echo ("Error deleting $file");
  }
else
  {
  echo ("Deleted $file");
  }
?>
```

PHP fgets() Function

The fgets() function returns a line from an open file.

The fgets() function stops returning on a new line, at the specified length, or at EOF, whichever comes first.

This function returns FALSE on failure.

Syntax

fgets(file,length)

file Required. Specifies the file to read from
length Optional. Specifies the number of bytes to read. Default is 1024 bytes.

Ex:

```
<?php
$file = fopen("test.txt","r");
echo fgets($file);
fclose($file);
?>
```

PHP fgetc() Function

The fgetc() function returns a single character from an open file.

file Required. Specifies the file to check.

Ex:

```
<?php
$file = fopen("test2.txt","r");
echo fgetc($file);
fclose($file);
?>
```

PHP feof() Function

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

This function returns TRUE if an error occurs, or if EOF has been reached. Otherwise it returns FALSE.

Syntax

feof(file)

file Required. Specifies the open file to check.

```
<?php
$myfile=fopen("sample.txt","r")or die("unable to open file");
While(!feof($myfile)){
Echo "<br>".fgetc($myfile);}
Fclose($myfile);
```

Require_once( )

The require_once() or include_once function can be used to include a PHP file in another one, when you may need to include the called file more than once. If it is found that the file has already been included, calling script is going to ignore further inclusions.

```
<?php
require_once 'heading.html';
require_once 'heading.html';
require 'var.php';
echo "My favourtecolor is $color</br>";
echo "My favourte car is $car<br>";
require 'footer.html';
?>
```

UNIT-IV

HTML

HTML Introduction

HTML is the standard mark-up language for creating Web pages.

- HTML stands for Hyper Text Mark-up Language
- HTML describes the structure of Web pages using mark-up
- HTML elements are the building blocks of HTML pages

- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

HTML TAGS

HTML

This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.

HEAD

This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.

TITLE

The <title> tag is used inside the <head> tag to mention the document title.

BODY

This tag represents the document's body which keeps other HTML tags like <h1>, , etc.

PARAGRAPH

HTML paragraphs are defined with the ** ** tag

Ex: This is a paragraph.

HEADING

HTML headings are defined with the **<h1>** to **<h6>** tags.

<h1> defines the most important heading. <h6> defines the least important heading:

Ex: <h1>This is heading 1</h1>

LINK

HTML links are defined with the **<a>** tag:

Ex: This is a link

IMAGE

HTML images are defined with the **<img>** tag.

The source file (src), alternative text (alt), width, and height are provided as attributes:

Ex: <img src="winter.jpg" alt="google.com" width="104" height="142">

CENTER

The   tag works for paragraphs, tables, headings, and images. Everything between the tags will be centred horizontally on the page.

BREAK

The
 tag stands for "**br**eak", or "line **br**eak". The
 tag will cause the text immediately after the tag to appear on a new line.

MARQUEE

<marquee> tag define different movement behaviours.

Ex: <marquee direction="right" scrollamount="1">BIODATA</marquee>

DIVISION

The element defines an arbitrary block of content which can be placed and styled as a single unit.

```
<div class=”mainsection”>
  The text is present in the div element and styled.</br>
  The class attribute is used to identify which style to apply.</br>
  The style is defined in head section
</div>

<style type=”text/css”>
 .mainsection { color: red; text-align: center}
</style>
```

ORDERED LISTS

The ordered list numbers each item consecutively. The basic form for an ordered list is the <ol> tag, a collection of **l**ist **i**tems identified by <li> tags, and then the tags

Ex:

```
<ol>
<li>List item 1</li>
<li>List item 2</li>
 <li>List item 3</li>
<li>List item 4</li>
</ol>
```

UNORDERED LISTS

In an unordered list bullets are placed before each list item. The basic form for an unordered list uses the <ul> tag, a collection of <li> tags for each of the list items, and finally a to tell the browser that the list is complete.

Ex:

```
<ul>
 <li>List item 1</li>
 <li>List item 2</li>
<li>List item 3</li>
<li>List item 4</li>
</ul>
```

NESTED LISTS

Nested Lists One of the more useful features of lists is the ability to create nested lists. A nested list is a list within a list. These lists may be any combination of unordered and ordered lists.

Ex:

```
<ul>
<li>Vegetables</li>
 <ul>
<li>Carrots</li>
<li>Beans</li>
 </ul>
<li>Fruits</li>
<ul>
 <li>Apples</li>
<li>Oranges</li>
 <li>Bananas</li>
 </ul>
<li>Meat</li>
<ul>
<li>Ground Beef</li>
 <li>Steak</li>
 <li>Ham</li>
```


Definiton list

<DL> tag is used to create a definition list,<DT> tag indicates that the enclosed text is a term in a term list, <DD>tag indicates that the enclosed text is a definition in a definition list.

Ex:
<DL>
<DT>Term 1</DT>
<LH>List Header</LH>
<dd>This is the definition of the first term.</dd>
<LH>List Header</LH>
<dd>This is the definition of the second term.</dd>
</DL>

FONT

The font tag does nothing on its own, but when combined with other keywords as options in the tag, it's quite powerful. You can string several options together in one command

Ex: Your text here.

TABLE

<TABLE> </TABLE> Indicates the beginning and end of the table.

<TR> </TR> Indicates the beginning and end of a table row.

<TH> </TH> Indicates the beginning and end of a table heading cell.

<TD> </TD> Indicates the beginning and end of a table data cell.

Ex:

<table border=2>

<tr><th>RollNO</th>Name</th><th>age</th></tr>

<tr><td>a15amd01</td><td>Raj</td><td>25</td></tr>

</table>

FIELDSET

FIELDSET tag creates a form for all elements in it.

Ex: **<fieldset>**Find a rounded-corner box around this text.**</fieldset>**

FRAMESET

Frameset tags define a layout of frames.

Ex :

<html>

<frameset cols="45%, *">

<frame src ="/htmlcodes/left-frame.html" />

<frame src ="/htmlcodes/right-frame.html" />

</frameset></html>

FRAME

Frame tags define each frame within a frameset.

Ex:

<frameset cols="35%, 65%">

<frame src ="/htmlcodes/left-frame.html" />

<frame src ="/htmlcodes/right-frame.html" />

</frameset>

FORM

Form tags define a form.

<form action="contact.html" method="post"></form>

INPUT

Input tags define input fields, such as text boxes, check boxes, radio buttons and etc.

Ex:

Your Name: <input type="text" name="visitor-name" maxlength="80" value="" />

LABEL

This tag defines a label to a form control.

Ex:

<label for ="name">NAME:</label>

SELECT & OPTION

Select tag creates a menu on a form. The option tag creates a drop-down menu. OPTION tag works only in conjunction with SELECT tag.

Ex:

```
Major subject: <select  name="major">
   <option value="Computer  Applications">Computer  Applications</option>
    <option value="Commerce">Commerce</option>
    <option value="physics">PHYSICS</option>
    <option value="maths"> MATHS</option>
 </select>
```

Java script

Java script is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

Where JavaScript is used

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation
- Dynamic drop-down menus

- Displaying data and time
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
- Displaying clocks etc.

JavaScript -Syntax

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>**.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script.

<script>
Javascript code
</script>

The script tag takes two important attributes −

- **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

```
<script language="javascript"  type="text/javascript">
        document.write("Hello World!")
    </script>
```

A simple syntax of your JavaScript will appear as follows.

```
<html>
  <body>
```

```
  <script language="javascript"  type="text/javascript">
        document.write("Hello World!")
    </script>
   </body>
</html>
```

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements is placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript"  type="text/javascript">
        var1 =10
        var2 = 20
  </script>
```

But when formatted in a single line as follows, you must use semicolons –

```
<script language="javascript"  type="text/javascript">
        var1 =10;  var2 = 20 ;
  </script>
```

Note − It is a good programming practice to use semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

NOTE − Care should be taken while writing variable and function names in JavaScript.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

JavaScript - Placement in HTML File

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows −

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows −

```
<html>
 <head>
 <script type="text/javascript">
   function sayHello()
{
       alert("Hello World")
```

```
   }
    </script>
  </head>
    <body>
    <input type="button" onclick="sayHello()" value="Say Hello" />
  </body>
  </html>
```

JavaScript in <body>...</body> section

If we need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

```
<html>
<head>
 </head>
  <body>
<script type="text/javascript">
        <!--
          document.write("Hello World")
        //-->
      </script>
      <p>This is web page body </p>
    </body>
</html>
```

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows −

```
<html>
   <head>
```

```
    <script type="text/javascript">
      <!--
        function sayHello() {
          alert("Hello World")
        }
      //-->
    </script>
  </head>
  <body>
    <script type="text/javascript">
      <!--
        document.write("Hello World")
      //-->
    </script>
     <input type="button" onclick="sayHello()" value="Say Hello" />
     </body>
</html>
```

JavaScript in External File

The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

```
<html>
<head>
<script type=”text/javascript” src=”exfile.js”></script>
</head
<body>
This is an example of usage of external java script </br>
```

<input type="button" onclick="sayHello()" value="Say Hello" />

</body>

</html>

To use JavaScript from an external file source, we need to write all the JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **exfile.js** file and then we can use **sayHello** function in your HTML file after including the exfile.js file.

exfile.js

```
Function sayHello( )
{
alert (“ hello world”);
}
```

JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types −

- **Numbers,** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
        Var  rollno;
        Var name;
  </script>
```

we can also declare multiple variables with the same **var** keyword as follows

```
<script type="text/javascript">
        Var rollno, name;
  </script>
```

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

```
<script type="text/javascript">
        Var  rollno =101;
        Var name;
        Name = " kumar";
  </script>
```

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

Java script form validation

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Ex:1 Basic Validation

```
<html>
<head>
<script type="text/javascript">
    function validate()
    {
if( document.myForm.Rollno.value == "" )
      {
        alert( "Please provide your Rollno!" );
```

```
          return false;
        }
Var x = document.myForm.Name.value;
 if(x == "" )
        {
          alert( "Please provide your name!" );
          document.myForm.Name.focus() ;
          return false;
        }
         if( document.myForm.Department.value == "-1" )
        {
          alert( "Please provide your Department!" );
          return false;
        }
        return( true );
     }
</script>
</head>
<body>
<form name="myForm"     action="" method=”post”/>
<center><table cellspacing="2" cellpadding="2" border="1">
 <tr><td>Enter your Rollno:</td><td><input type ="text" name ="Rollno"</td></tr>
<tr> <td align="right">Name</td><td><input type="text" name="Name" /></td> </tr>
<tr><td align="right">Department</td><td><select name="Department">
                  <option value="-1" selected>[choose yours]</option>
                  <option value="1">M.Sc(IT)</option>
                  <option value="2">BCA</option>
                  <option value="3">CS</option>
                  <option value="4">Maths</option>
                </select>
  </td></tr>
```

```
<tr><td align="right"></td><td><input type="submit" value="Submit" onClick="validate()"
/></td></tr>
  </table></center>
    </form>
</html>
```

Ex: 2 Data Format Validation

The following example shows how to validate an entered email address and rollno. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign and the rollno should contain atleast eight digits .

```
<html>
<head>
<script type="text/javascript">
    function validate()
    {

var rollno = document.myForm.Rollno.value;

var emailID = document.myForm.EMail.value;

      atpos = emailID.indexOf("@");

      dotpos = emailID.lastIndexOf(".");

 if (atpos < 1 || (dotpos+2>=emailID.length))

      {

        alert("Please enter correct email ID")

        document.myForm.EMail.focus() ;

        return false;

      }
```

```
Else if( rollno.length < 8)
        {
          alert( "Please provide rollno with minimum eight digits!" );
         return false;
        }
return( true );
}

</script>

</head>

<body>

<form name="myForm"      action="" method=”post” onsubmit= “return validate()” />
<center><table cellspacing="2" cellpadding="2" border="1">
 <tr><td>Enter your Rollno:</td><td><input type ="text" name ="Rollno"</td></tr>
<tr><td align="right">Enter your Email:</td><td><input type="text" name="EMail"/></td>
</tr>
<tr><td align="right"></td><td><input type="submit" value="Submit" /></td></tr>
  </table></center>
     </form>
</html>
```

Unit-IV (continuation)

File upload

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

Creating an upload form

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

upload.html

```
<html>
<body>
<form action ="upload.php" method ="POST"
enctype = "multipart/form-data">
<input type = "file" name="myfile" size="10"  >
<input type = "submit" value= "upload-file" name="upload" >
</form>
</body>
</html>
```

upload.php

```
if($_FILES['myfile']['name'] !="")
{
move_uploaded_file($_FILES['myfile']['tmp_name'],"temp");
echo"file succesfully moved";
}
else
echo "Please select the file";
?>
```

Download Button

The download attribute specifies that the target will be downloaded when a user clicks on the hyperlink.

This attribute is only used if the href attribute is set.

The value of the attribute will be the name of the downloaded file.

Syntax

<a download="filename">

Value	Description
filename	Optional. Specifies the new filename for the downloaded file

```
<a href="image.bmp" download>
  <img src="image.bmp" alt="Wamp server" width="104" height="142">
</a>
```

PHP Global Variables - Superglobals

Several predefined variables in PHP 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.

The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_COOKIE
- $_SESSION

PHP $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

Ex:

```
<?php
$x = 75;
$y = 25;

function addition() {
   $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z;
?>
```

O/p: 100

PHP $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

Ex:

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
?>
```

PHP $_REQUEST

PHP $_REQUEST is used to collect data after submitting an HTML form.

```
<html>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
```

```
  <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // collect value of input field
   $name = $_REQUEST['fname'];
   if (empty($name)) {
      echo "Name is empty";
   } else {
      echo $name;
   }}
?></body></html>
```

PHP $_POST

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

$_POST is an array of variables passed to the current script via the HTTP POST method.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // collect value of input field
   $name = $_POST['fname'];
   if (empty($name)) {
      echo "Name is empty";
   } else {
      echo $name;
   }}?>
```

PHP $_GET

PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".

$_GET is an array of variables passed to the current script via the URL parameters.

test_get.php

```
<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>
```

test_get.html

```
<html>
<body>
<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>
</body>
</html>
```

PHP $_FILES

$_FILES[“FormElementName”]

: It can be used to upload files from a client computer/system to a server.

OR

$_FILES[“FormElementName”][“ArrayIndex”]

: Such as File Name, File Type, File Size, File temporary name.

Ex:

```
<html>
<body>
<form action ="" method ="POST" enctype = "multipart/form-data">
<input type = "file" name="myfile" size="10"  >
<input type = "submit" value= "upload-file" name="upload" >
<?php
echo $_FILES['myfile']['size'];
echo"<br>".$_FILES['myfile']['name'];
echo"<br>".$_FILES['myfile']['type'];
?>
</body>
</html>
```

PHP Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose.

PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Creating Cookies with PHP

A cookie is created with the setcookie() function.

Syntax

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

Only the *name* parameter is required. All other parameters are optional.

- **Name** − This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** − This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** − This specify a future time in seconds since,After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** − This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** − This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** − This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

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

```
<?php

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

setcookie("age", "35", time()+3600, "/", "", 0);

//Checking the function if a cookie is set or not.

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

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

else

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

?>
```

Deleting Cookie with PHP

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

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

```
<?php

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

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

?>

<body>

<?php echo "Deleted Cookies"?>

</body>
```

SESSION

A HTML session is a collection of variables that keeps its state as the user navigates the pages of a certain site. It will only be available to that domain that created it, and will be deleted soon after the user left the site or closed his browser.

So, the session has a semi-permanent nature, and it can be used to pass variables along different pages on which the visitor lands during a visit to the site.

session_start()function.: This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

Ex:

```
<?php
session_start( ) ;
?>
```

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables.

Ex:

```
<?php
session_destroy( );
?>
```

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

Syntax:

```
<?php
Unset ($_SESSION[‘variablename’];
?>
```

Ex:

Sessionpage1.php

```
<html>

<body>

<font color="red"><h1>  <marquee direction="right" scrollamount="1">BIODATA</marquee>
</h1></font>

<form method="post" action="session2.php">

<table border="2" cellspacing="10" cellpadding="10">

<tr><td><label for ="name">REGNO:</label></td> <td> <input type="text" name="regno" /></td></tr>

<tr><td><label for ="name">NAME:</label></td> <td> <input type="text" name="name" /></td></tr>

</table>

<tr><td <input type="submit" name="SEND" value="CONTINUE" /></td></tr>

</body>

</html>
```

Sessionpage2.php

```
<html>

<body>

<?php

// starting session

session_start();

//storing the posted variables in session variables

$_SESSION['regno']=$_POST['regno'];

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

?>

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

<table>

<tr><td><label for ="name">DOB:</label></td> <td> <input type="DATE" name="dob" /></td></tr>

<tr><td><label for ="name">MOBILENO:</label></td> <td> <input type="number" name="mobileno"
/></td></tr>
```

```
</table >
<tr><td><input type="submit" name="SEND" value="submit" /></td></tr>
</html>
```

Sessionfinal.php

```
<html>
<body>
<?php
session_start();
$regno =$_SESSION['regno'];
$name= $_SESSION['name'];
$dob=$_POST['dob'];
$mobileno=$_POST['mobileno'];
echo " </br></br></br></br><center><H3>BIODATA</H3> <table border=2>
<tr><th>REGNO:</TH><TD>$regno</td></tr>
<tr><th>NAME:</TH><TD>$name</td></tr>
<tr><th>DOB:</TH><TD>$dob</td></tr>
<tr><th>MOBILE NO:</TH><TD>$mobileno</td></tr>
</table></center>"
session_destroy( );
?>
</body>
</html>
```

UNIT-V

Creating a MYSQL database

```
<?php
//Connection string
if(!mysql_connect("localhost","root",""))
{
die('Could not connect: ' . mysql_error());
}
Else
{
$sql = "CREATE Database sample ";
$retval = mysql_query($conn);
if(! $retval )
{
die('Error: '. mysql_error());
}
else
{
echo " Database created successfully\n";
}
}
?>
```

Creating an New table

Ex:

```
<?php
if(!mysql_connect("localhost","root","","sample"))
```

```
{
die ('Error : '.mysql_error());
}
Else
{
$sql = "CREATE TABLE student (
rollnovarchar(10),
firstname VARCHAR(30) ,
address VARCHAR(50) ,
email VARCHAR(30),
birth_date date)";
$retval=mysql_query($sql);
if (! $retval)
{
 die ('Error : '.mysql_error());
}
Else
{
echo "  Table created successfully";
}
}
?>
```

Inserting data into the table

The INSERT statement is used to add new records to a MySQL table.

Ex:

```
<?php
mysql_connect("localhost","root","");
mysql_select_db("sample");



$q="insert into student values ('101','sam')";
mysql_query($q);
If(!$q)
{ mbbbb
die(‘Error: ‘mysql_error());
}
Else
{
echo "RECORD SUCCESSFULLY INSERTED";
}
?>
```

Updating data into the table in a database

The UPDATE statement is used to update existing records in a table.

Ex:

```
<?php
// Connection to a Database
mysql_connect("localhost","root","");
mysql_select_db("sample");
// SQL Query to update data in a table
```

```
$sql = "update student set sname='kumar' where rno=101";
mysql_query($sql);
if (!$sql)
{
Die(‘Error: ‘mysql_error()) ;
}
else {
echo"successfully updated";
}
```
?>**Deletingdata from the table in a database**

The DELETE statement is used to delete records from a table.

Ex:

```
<?php
// Connection to a Database
mysql_connect("localhost","root","");
mysql_select_db("sample");
// SQL Query to delete a row in a table
$sql= "delete from student where rno=101";
mysql_query($sql);
if (!$sql)
{
Die(‘Error:’mysql_error()) ;
}
else {
echo"successfully deleted";
```

```
}
?>
```

Selecting a Data from the Table

The Select statement is used to select the data from the database.

```
<?php
//Connection to a Database
Mysql_connect("localhost","root","");
Mysql_select_db(“sample”);

// SQL query to select data from a table
$sql=mysql_query("select * from student WHERE Rollno=101");
If(!sql)
{
Die(‘Error:’mysql_error());
}
Else
{
echo"<tableborder=5><tr><th>RollNO</th><th>NAME</th><th>Address</th><th>
Emailid</th><th>DOb</th></tr>";
while($r=mysql_fetch_array($sql))
{
echo"<tr><td>".$r["0"]."</td><td>".$r["1"]."</td><td>".$r["2"]."</td><td>".$r["3"]."</td><td>
".$r["4"]."</td></tr>";
}
echo"</table></CENTER>";
```

```
}
?>
```