PHP

1)Introduction:-
a)PHP is a server scripting language, and is a powerful tool for making dynamic and interactive Web pages quickly.
b)PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.
c) PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.
d) PHP is an acronym for "PHP Hypertext Preprocessor",PHP costs nothing, it is free to download and use.
e) **What is a PHP File?**
i)PHP files can contain text, HTML, CSS, JavaScript, and PHP code
ii)PHP code are executed on the server, and the result is returned to the browser as plain HTML
iii)PHP files have extension ".php".
f)What Can PHP Do?
i)PHP can generate dynamic page content
ii)PHP can create, open, read, write, delete, and close files on the server
iii)PHP can collect form data
v)PHP can send and receive cookies
v)PHP can add, delete, modify data in your database
vi)PHP can restrict users to access some pages on your website
vii)PHP can encrypt data
viii)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.
g)Why PHP?
i)PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
ii)PHP is compatible with almost all servers used today (Apache, IIS, etc.)
iii)PHP supports a wide range of databases
iv)PHP is easy to learn and runs efficiently on the server side
h) **Characteristics of PHP**
Five important characteristics make PHP's practical nature possible:
i)Simplicity ii) Efficiency iii)Security iv)Flexibility v)Familiarity.
2) PHP Installation:-
a)To run php you required , you must:
i)install a web server
ii)install PHP:-
iii)install a database, such as MySQL
3)PHP Syntax:-
The PHP script is executed on the server, and the plain HTML result is sent back to the browser.
a)A PHP script can be placed anywhere in the document.
b)A PHP script starts with **<?php** and ends with **?>**:

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

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

e)Example

```
<!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).

f) 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!.

Example

```
<!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>
```

g)PHP Case Sensitivity

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

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

iii)Example

```
<!DOCTYPE html>
<html> <body>
<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>
</body></html>
```

h) 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):

Example

```
<!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>
```

4) PHP Variables:-

a)PHP variables can be used to hold values (x=5) or expressions (z=x+y).
b)A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
c)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).

d)**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.

e)PHP is a Loosely Typed Language
i)In the example above, notice that we did not have to tell PHP which data type the variable is.
ii)PHP automatically converts the variable to the correct data type, depending on its value.
iii)In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

f) PHP Variables Scope
i)In PHP, variables can be declared anywhere in the script.
ii)The scope of a variable is the part of the script where the variable can be referenced/used.
iii)PHP has three different variable scopes:

- local
- global
- static

aa)Local and Global Scope
i)A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function.
ii)A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function.
iii)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";
?>
```

bb) 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.

e)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.

bb)The global Keyword
i)The global keyword is used to access a global variable from within a function.
ii)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
?>
```

iv)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.

Example

```
<?php
$x=5;
$y=10;
function myTest() {
  $GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

cc) The static Keyword

i)Normally, when a function is completed/executed, all of its variables are deleted(i.e. lost the initialized value). However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

ii)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();
?>
```

```
Output:-0
        1
        2
```

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

v)Note: The variable is still local to the function.

5) echo and print Statements

a)In PHP there are two basic ways to get output: echo and print.

b)There are some differences between echo and print:

- echo - can output one or more strings
- print - can only output one string, and returns always 1.

echo is marginally faster compared to print as echo does not return any value.

c) The PHP echo Statement

echo is a language construct, and can be used with or without parentheses: echo or echo().

Display Strings

The following example shows how to display different strings with the echo command (also notice that the strings can contain HTML markup):

Example

```
<?php
echo "<h2>PHP is fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This", " string", " was", " made", " with multiple parameters.";
?>
```

Output:- PHP is fun!

```
Hello world!
I'm about to learn PHP!
This string was made with multiple parameters.
```

If we run above code using print statement it will give an error on line no 5.because print statement take only one parameter.

d) The print Statement

print is also a language construct, and can be used with or without parentheses: print or print().

Display Strings

The following example shows how to display different strings with the print command (also notice that the strings can contain HTML markup):

Example

```
<?php
print "<h2>PHP is fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

Output:- PHP is fun!

```
Hello world!
I'm about to learn PHP!
```

6)PHP 5 Data Types

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

a)PHP Strings

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

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

Example

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

Output:- Hello world!

b) Integers

i)An integer is a number without decimals.

ii)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);
?>
```

```
Output:-  int(5985)
          int(-345)
          int(140)
          int(39).
```

c) PHP Floating Point Numbers

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

Example

```
<?php
$x = 10.365;
var_dump($x);  echo "<br>";
$x = 2.4e3;
var_dump($x);  echo "<br>";
$x = 8E-5;
var_dump($x);
?>
```

```
Output:- float(10.365)
float(2400)
float(8.0E-5).
```

d) PHP Booleans

Booleans can be either TRUE or FALSE.

$x=true; $y=false; Booleans are often used in conditional testing.

e) 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);
?>
Output:- array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
```

f) PHP NULL Value

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

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

g) 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:

```
<!DOCTYPE html><html><body><?php
class Car
{    var $color;
   function Car($color) {
     $this->color = $color;
   }
   function what_color() {
   echo $this->color;
   }
}
// instantiate one object
$a = new Car("white");
$a-> what_color();
?>  </body></html>
```

7) PHP String Functions:-

a)String Concatenation Operator

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

```
<?php
$string1="Hello World";
$string2="1234";
echo $string1 . " " . $string2;
?>
```

This will produce following result:Hello World 1234

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.
Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Function name	Explation	Function Code
str_ireplace()	Replaces some characters in a string (case-insensitive)	<?php echo str_ireplace("WORLD","Peter","Hello world!"); ?> output:- Hello Peter!
str_pad()	Pads a string to a new length	<?php $str = "Hello World"; echo str_pad($str,20,"."); ?> Output:- Hello World.........
str_repeat()	Repeats a string a specified number of times	<?php echo str_repeat(".",13); ?> Output:- ....
str_replace()	Replaces some characters in a string (case-sensitive)	<?php echo str_replace("world","Peter","Hello world!"); ?> Output:-Hello Peter
str_rot13()	Performs the ROT13 encoding on a string	<?php echo str_rot13("Hello World"); ?> Outout:- Uryyb Jbeyq
str_shuffle()	Randomly shuffles all characters in a string	<?php echo str_shuffle("Hello World"); ?> Output:- drloWHeol l
str_split()	Splits a string into an array	<?php print_r(str_split("Hello")); ?> Output:- Array ([0] => H [1] => e [2] => l [3] => l [4] => o)
str_word_count()	Count the number of words in a string	<?php echo str_word_count("Hello world!"); ?> output:-2

strcasecmp()	Compares two strings (case-insensitive)	<?php echo strcasecmp("Hello world!","HELLO WORLD!"); ?> Output:-0
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())	<?php echo strchr("Hello world!","world"); ?> Output:- world!
strcmp()	Compares two strings (case-sensitive)	<?php echo strcmp("Hello world!","Hello world!"); ?> output=0
strcspn()	Returns the number of characters found in a string before any part of some specified characters are found	<?php echo strcspn("Hello world!","w"); ?> Output:- 6
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)	<?php echo stripos("I love php, I love php too!","PHP"); ?> Output:- 7
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)	<?php echo stristr("Hello world!","WORLD"); ?> Output:- world!
strlen()	Returns the length of a string	<?php echo strlen("Hello"); ?> Output:-5
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)	<?php echo strpos("I love php, I love php too!","php"); ?> Output:- 7
strrchr()	Finds the last occurrence of a string inside another string	<?php echo strrchr("Hello world!","world"); ?> Output:- world!
strrev()	Reverses a string	<?php echo strrev("Hello World!"); ?> Output:- !dlroW olleH
strtolower()	Converts a string to lowercase letters	<?php echo strtolower("Hello WORLD."); ?> Output:- hello world.
strtoupper()	Converts a string to uppercase letters	<?php echo strtoupper("Hello WORLD!"); ?> Output:- HELLO WORLD!
strtr()	Translates **certain** characters in a string	<?php echo strtr("Hilla Warld","ia","eo"); ?> Output:- Hello World
substr()	Returns a part of a string	<?php

		echo substr("Hello world",6); ?> Output:- world
ucfirst()	Converts the first character of a string to uppercase	<?php echo ucfirst("hello world!"); ?> Output:- Hello world!
ucwords()	Converts the first character of each word in a string to uppercase	<?php echo ucwords("hello world"); ?> Output:- Hello World

8)Constants:-

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 jspm!":

Example

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

Constants are like variables except that once they are defined they cannot be changed or undefined.

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

Example

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

9)What is Operator? PHP language supports following type of operators.

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

a)Arithmatic Operators:

There are following arithmatic operators supported by PHP language:
Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200

/	Divide numerator by denumerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

e.g:-
```
<html>
<head><title>Arithmetical Operators</title><head> <body>
<?php
   $a = 42;     $b = 20;
      $c = $a + $b;
   echo "Addtion Operation Result: $c <br/>";
      $c = $a++;
   echo "Increment Operation Result: $c <br/>";
   $c = $a--;
   echo "Decrement Operation Result: $c <br/>";
?>
</body></html>
```

b)Comparison Operators:

There are following comparison operators supported by PHP language
Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

```
<html>  <head><title>Comparision Operators</title><head>
<body>
```

```
<?php
   $a = 42;
   $b = 20;
   if( $a == $b ){
     echo "TEST1 : a is equal to b<br/>";
   }else{
     echo "TEST1 : a is not equal to b<br/>";
   }
   ?> </body> </html>
```

c) Logical Operators:

There are following logical operators supported by PHP language
Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

```
<html>  <head><title>Logical Operators</title><head>
<body>
<?php
   $a = 42;    $b = 0;
      if( $a && $b ){
     echo "TEST1 : Both a and b are true<br/>";
   }else{
     echo "TEST1 : Either a or b is false<br/>";
   }
   if( $a and $b ){
     echo "TEST2 : Both a and b are true<br/>";
   }else{
     echo "TEST2 : Either a or b is false<br/>";
   }
?>
</body></html>
```

d)Assignment Operators:

There are following assignment operators supported by PHP language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C

e) Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax:

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

```
<html><head><title>Arithmetical Operators</title><head>
<body>
<?php
   $a = 10;    $b = 20;
      /* If condition is true then assign a to result otheriwse b */
   $result = ($a > $b ) ? $a :$b;
   echo "TEST1 : Value of result is $result<br/>";
?>
</body></html>
```

f)Operators Categories:

All the operators we have discussed above can be categorised into following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.

- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

g)Precedence of PHP Operators:

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

10)Decision making:-

a)The if, elseif ...else and switch statements are used to take decision based on the different condition.

b)You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements:

Statement	Description	Example

if...else statement	use this statement if you want to execute a set of code when a condition is true and another if the condition is not true	`<html><body>`
		`<?php`
		`$d=date("D");`
		`if ($d=="Fri")`
		`  echo "Have a nice weekend!";`
		`else`
		`  echo "Have a nice day!";`
		`?></body></html>`

elseif statement	is used with the if...else statement to execute a set of code if **one** of several condition are true	`<html><body>`
		`<?php`
		`$d=date("D");`
		`if ($d=="Fri")`

		echo "Have a nice weekend!"; elseif ($d=="Sun") echo "Have a nice Sunday else echo "Have a nice day!"; ?> </body></html>

switch statement	is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.	<html><body>
		<?php
		$d=date("D");
		switch ($d)
		{
		case "Mon":
		echo "Today is Monday";
		break;
		case "Tue":
		echo "Today is Tuesday";
		break;
		default:
		echo "Wonder which day this ?";
		}
		?></body></html>

11)Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

Statement	**Description**	**Example**
for	loops through a block of code a specified number of times.	<html><body> <?php $a = 0; $b = 0; for($i=0; $i<5; $i++) { $a += 10; $b += 5; } echo ("At the end of the loop a=$a and b=$b"); ?></body></html>
while	loops through a block of code if and as long as a	<html><body>

	specified condition is true.	`<?php`
		`$i = 0;   $num = 50;`
		`while( $i < 10)`
		`{   $num--;`
		`   $i++;`
		`}`
		`echo ("Loop stopped at i = $i and num = $num" );`
		`?>`
		`</body></html>`

do…while	The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.	`<html><body>`
		`<?php`
		`$i = 0;`
		`$num = 0;`
		`do`
		`{  $i++;`
		`}while( $i < 10 );`
		`echo ("Loop stopped at i = $i" );`
		`?>`
		`</body></html>`

foreach	The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.	`<html><body>`
		`<?php`
		`$array = array( 1, 2, 3, 4, 5);`
		`foreach( $array as $value )`
		`{`
		`  echo "Value is $value <br />";`
		`}`
		`?></body></html>`

e)The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

```
<html><body>
<?php
$i = 0;
while( $i < 10)
{   $i++;
   if( $i == 3 )break;
}
echo ("Loop stopped at i = $i" );
?></body></html>
```

result:Loop stopped at i = 3

f) The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

```
<html><body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{  if( $value == 3 )continue;
  echo "Value is $value <br />";
}
?>
</body></html>
```

Result:-Value is 1,Value is 2,Value is 4,Value is 5

12) **An array** :-is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.
There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array -** An array with a numeric index. Values are stored and accessed in linear fashion also called **Indexed arrays.**
- **Associative array -** An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array -** An array containing one or more arrays and values are accessed using multiple indices

a)Numeric array:

i)These arrays can store numbers, strings and any object but their index will be prepresented by numbers. By default array index starts from zero.

Ii)Example:Here we have used **array()** function to create array.

```
<html><body>
<?php
$numbers = array( 1, 2, 3, 4, 5); /* First method to create array. */
foreach( $numbers as $value )
{
  echo "Value is $value <br />";
}
$numbers[0] = "one"; /* Second method to create array. */
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";
foreach( $numbers as $value )
{  echo "Value is $value <br />";
```

```
Output:Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
}Value is four
?></body></html>
Value is five
```

iii) Get The Length of an Array - The count() Function
The count() function is used to return the length (the number of elements) of an array:

```
<!DOCTYPE html><html><body>
<?php
$cars=array("Volvo","BMW","Toyota");
echo count($cars);
?>
</body>
</html>output:-3
```

iv) Loop Through an Indexed Array
To loop through and print all the values of an indexed array, you could use a for loop, like this:

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

for($x=0;$x<$arrlength;$x++) {
  echo $cars[$x];
  echo "<br>";
}
?></body></html>
Output:- Volvo
BMW
Toyota
```

v)
```
<!DOCTYPE html><html><body>
<?php
$cars=array("Volvo","BMW","Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
</body></html>
ouput:- I like Volvo, BMW and Toyota.
```

b) Associative Arrays
i)The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.
ii) Example

```
<html><body>
<?php
/* First method to associate create array. */
$salaries = array(
                    "mohammad" => 2000,
                    "qadir" => 1000,
                    "zara" => 500
                   );

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
echo "Salary of zara is ".  $salaries['zara']. "<br />";

/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
echo "Salary of zara is ".  $salaries['zara']. "<br />";
?>
</body>
</html>
```

This will produce following result:

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

iii)Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

```
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
foreach($age as $x=>$x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
```

Output:- Key=Peter, Value=35

```
Key=Ben, Value=37
Key=Joe, Value=43
```

c) Multidimensional Arrays
A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example
In this example we create a two dimensional array to store marks of three students in three subjects: This example is an associative array, you can create numeric array in the same fashion.

```
<html><body>
<?php
  $marks = array(
                    "mohammad" => array
                    ("physics" => 35,
                    "maths" => 30,
                    "chemistry" => 39
                    ),
                    "qadir" => array
          ( "physics" => 30,
          "maths" => 32,
          "chemistry" => 29
          ),
          "zara" => array
          ( "physics" => 31,
          "maths" => 22,
          "chemistry" => 39
          )
             );
  /* Accessing multi-dimensional array values */
  echo "Marks for mohammad in physics : " ;
  echo $marks['mohammad']['physics'] . "<br />";
  echo "Marks for qadir in maths : ";
  echo $marks['qadir']['maths'] . "<br />";
  echo "Marks for zara in chemistry : " ;
  echo $marks['zara']['chemistry'] . "<br />";
?>
</body></html>
```

This will produce following result:

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
```

Marks for zara in chemistry : 39

Numeric array:-

```
<?php
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );

echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

Output:-

```
Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15
```

Other functions:-
Print_r function is used to print array in human readable form.

Function name	Explanation	Function Code

array_change_key_case()	Changes all keys in an array to lowercase or uppercase	`<?php`
		`$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");`
		`print_r(array_change_key_case($age,CASE_UPPER));`
		`?>`
		output:- Array ([PETER] => 35 [BEN] => 37 [JOE] => 43)

array_column()	Returns the values from a single column in the input array	`<?php`
		`$a = array(`
		`  array(`
		`    'id' => 5698,      'first_name' => 'Peter',`
		`    'last_name' => 'Griffin',     ),`
		`  array(`
		`    'id' => 4767,    'first_name' => 'Ben',     'last_name' =>`

		'Smith',)); $last_names = array_column($a, 'last_name'); print_r($last_names); ?> Output:- Array ([0] => Griffin [1] => Smith)
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array	<?php $fname=array("Peter","Ben","Joe"); $age=array("35","37","43"); $c=array_combine($fname,$age); print_r($c); ?> Output:- Array ([Peter] => 35 [Ben] => 37 [Joe] => 43)
array_count_values()	Counts all the values of an array	<?php $a=array("A","Cat","Dog","A","Dog"); print_r(array_count_values($a)); ?>output:- Array ([A] => 2 [Cat] => 1 [Dog] => 2)
array_diff()	Compare arrays, and returns the differences (compare values only)	<?php $a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("e"=>"red","f"=>"black","g"=>"purple"); $a3=array("a"=>"red","b"=>"black","h"=>"yellow"); $result=array_diff($a1,$a2,$a3); print_r($result); ?>output:- Array ([b] => green [c] => blue)
array_diff_assoc()	Compare arrays, and returns the differences (compare keys and values)	<?php $a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("a"=>"red","f"=>"green","g"=>"blue"); $a3=array("h"=>"red","b"=>"green","g"=>"blue"); $result=array_diff_assoc($a1,$a2,$a3); print_r($result); ?> output:- Array ([c] => blue [d] => yellow)
		<?php $a1=array("a"=>"red","b"=>"green","c"=>"blue");

		$a2=array("c"=>"yellow","d"=>"black","e"=>"brown"); $a3=array("f"=>"green","c"=>"purple","g"=>"red"); $result=array_diff_key($a1,$a2,$a3); print_r($result); ?>output:- Array ([a] => red [b] => green)

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

array_keys()	Returns all the keys of an array	<?php $a=array("Volvo"=>"XC90","BMW"=>"X5","Toyota"=>"Highlander"); print_r(array_keys($a)); ?> output:- Array ([0] => Volvo [1] => BMW [2] => Toyota)
array_merge()	Merges one or more arrays into one array	<?php $a1=array("red","green"); $a2=array("blue","yellow"); print_r(array_merge($a1,$a2)); ?> output:- Array ([0] => red [1] => green [2] => blue [3] => yellow)

array_multisort()	Sorts multiple or multi-dimensional arrays	<?php
		$a1=array("Dog","Dog","Cat");
		$a2=array("Pluto","Fido","Missy");
		array_multisort($a1,SORT_ASC,$a2,SORT_DESC);
		print_r($a1);
		print_r($a2);
		?>
		Output:- Array ([0] => Cat [1] => Dog [2] => Dog)
		Array ([0] => Missy [1] => Pluto [2] => Fido)

array_pad()	Inserts a specified number of items, with a specified value, to an array	<?php $a=array("red","green"); print_r(array_pad($a,5,"blue")); ?> output:- Array ([0] => red [1] => green [2] => blue [3] => blue [4] => blue)
array_pop()	Deletes the last element	<?php

	of an array	$a=array("red","green","blue");
		array_pop($a);
		print_r($a);
		?> output:- Array ([0] => red [1] => green)

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

array_rand()	Returns one or more random keys from an array	<?php $a=array("a"=>"red","b"=>"green","c"=>"blue","d"=>" yellow"); print_r(array_rand($a,2)); ?>output:- Array ([0] => b [1] => c)

array_reduce()	Returns an array as a string, using a user-defined function.
	Third parameter is initial value.
		<?php
		function myfunction($v1,$v2)
		{
		return $v1 . "-" . $v2;
		}
		$a=array("Dog","Cat","Horse");
		print_r(array_reduce($a,"myfunction",5));
		?>output:- 5-Dog-Cat-Horse OR
		<?php
		function myfunction($v1,$v2)
		{
		return $v1+$v2;
		}
		$a=array(10,15,20);
		print_r(array_reduce($a,"myfunction",5));
		?>output:- 50

array_replace()	Replaces the values of the first array with the values from following arrays	<?php $a1=array("red","green"); $a2=array("blue","yellow"); print_r(array_replace($a1,$a2)); ?> output:- Array ([0] => blue [1] => yellow)
array_reverse()	Returns an array in the reverse order	<?php $a=array("a"=>"Volvo","b"=>"BMW","c"=>"Toyota"); print_r(array_reverse($a)); ?> output:- Array ([c] => Toyota [b] => BMW [a] => Volvo)
array_search()	Searches an array for a given value and returns	<?php $a=array("a"=>"red","b"=>"green","c"=>"blue");

	the key	echo array_search("red",$a); ?> output:- a
array_shift()	Removes the first element from an array, and returns the value of the removed element	<?php $a=array("a"=>"red","b"=>"green","c"=>"blue"); echo array_shift($a); print_r ($a); ?>output:- red Array ([b] => green [c] => blue)
array_slice()	Returns selected parts of an array	<?php $a=array("red","green","blue","yellow","brown"); print_r(array_slice($a,2)); ?> output:- Array ([0] => blue [1] => yellow [2] => brown)

array_splice()	Removes and replaces specified elements of an array	<?php
		$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=> "yellow");
		$a2=array("a"=>"purple","b"=>"orange");
		array_splice($a1,0,2,$a2);
		print_r($a1);
		?> output:- Array ([0] => purple [1] => orange [c] => blue [d] => yellow)

array_sum()	Returns the sum of the values in an array	<?php $a=array("a"=>52.2,"b"=>13.7,"c"=>0.9); echo array_sum($a); ?>output:- 66.8
array_unique()	Removes duplicate values from an array	<?php $a=array("a"=>"red","b"=>"green","c"=>"red"); print_r(array_unique($a)); ?>output:- Array ([a] => red [b] => green)

array_unshift()	Adds one or more elements to the beginning of an array	<?php
		$a=array("a"=>"red","b"=>"green");
		array_unshift($a,"blue");
		print_r($a);
		?>output:- Array ([0] => blue [a] => red [b] => green)

array_values()	Returns all the values of an array	<?php $a=array("Name"=>"Peter","Age"=>"41","Country"=>" USA"); print_r(array_values($a)); ?>output:- Array ([0] => Peter [1] => 41 [2] => USA)

arsort()	Sorts an associative array in descending order, according to the value	<?php
		$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		arsort($age);
		foreach($age as $x=>$x_value)

		{ echo "Key=" . $x . ", Value=" . $x_value; echo " "; } ?>output:- Key=Joe, Value=43 Key=Ben, Value=37 Key=Peter, Value=35
asort()	Sorts an associative array in ascending order, according to the value	<?php $age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43"); asort($age); foreach($age as $x=>$x_value) { echo "Key=" . $x . ", Value=" . $x_value; echo " "; } ?>output:- Key=Peter, Value=35 Key=Ben, Value=37 Key=Joe, Value=43
compact()	Create array containing variables and their values	<?php $firstname = "Peter"; $lastname = "Griffin"; $age = "41"; $result = compact("firstname", "lastname", "age"); print_r($result); ?>output:- Array ([firstname] => Peter [lastname] => Griffin [age] => 41)
count()	Returns the number of elements in an array	<?php $cars=array("Volvo","BMW","Toyota"); echo count($cars); ?>output:-3
extract()	Imports variables into the current symbol table from an array	<?php $a = "Original"; $my_array = array("a" => "Cat","b" => "Dog", "c" => "Horse"); extract($my_array); echo "\$a = $a; \$b = $b; \$c = $c"; ?>output:- $a = Cat; $b = Dog; $c = Horse
in_array()	Checks if a specified value exists in an array	<?php $people = array("Peter", "Joe", "Glenn", Cleveland"); if (in_array("Glenn", $people)) { echo "Match found"; } else

		`  {   echo "Match not found";    }`
		`?>output:- Match found`

key()	Fetches a key from an array	`<?php`
		`$people=array("Peter","Joe","Glenn","Cleveland");`
		`echo "The key from the current position is: " .`
		`key($people);`
		`?>output:- The key from the current position is: 0`

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

list()	Assigns variables as if they were an array	`<?php`
		`$my_array = array("Dog","Cat","Horse");`
		`list($a, $b, $c) = $my_array;`
		`echo "I have several animals, a $a, a $b and a $c.";`
		`?>output:- I have several animals, a Dog, a Cat and a Horse.`

rsort()	Sorts an indexed array in descending order	`<?php` `$cars=array("Volvo","BMW","Toyota");` `rsort($cars);` `$clength=count($cars);` `for($x=0;$x<$clength;$x++)` `  {` `  echo $cars[$x];` `  echo " ";`

		} ?>output:- Volvo Toyota BMW
sort()	Sorts an indexed array in ascending order	`<?php` `$cars=array("Volvo","BMW","Toyota");` `sort($cars);` `$clength=count($cars);` `for($x=0;$x<$clength;$x++)` `{` `echo $cars[$x];` `echo " ";` `}`<br>`?>`output:- BMW Toyota Volvo

13)PHP Functions

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

a)PHP User Defined Functions

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.

b)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).

Ex.

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

Output:- Hello world!

c) PHP Function Arguments

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

ii)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.

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

Output:- Jani Refsnes.

Hege Refsnes.

d) The following example has a function with two arguments ($fname and $year):

```
<?php
function familyName($fname,$year) {
  echo "$fname Refsnes. Born in $year <br>";
}
familyName("Hege","1975");?>
```

Output:- Hege Refsnes. Born in 1975

e) 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:

```
<?php
function setHeight($minheight=50) {
  echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight();
setHeight(80);
?>
```

Output: The height is : 350

The height is : 50

The height is : 80.

f)PHP Functions - Returning values

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

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

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

Output:- 5 + 10 = 15
2 + 4 = 6

14)PHP 5 Global Variables - Superglobals

a)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.

b)The PHP superglobal variables are:

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

i) 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.

```
<?php
$x = 75;  $y = 25;
function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z;
?>  Output:- 100
```

ii) PHP $_SERVER

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

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

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as

	Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using

iii) 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
$name = $_REQUEST['fname'];
echo $name;
?></body></html>
```

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

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

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

$_GET can also collect data sent in the URL.
Assume we have an HTML page that contains a hyperlink with parameters:

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

When a user clicks on the link "Test $GET", the parameters "subject" and "web" is sent to "test_get.php", and you can then acces their values in "test_get.php" with $_GET.
The example below shows the code in "test_get.php":

Example

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

15) PHP Date and Time

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

a)The PHP Date() Function

The PHP date() function formats a timestamp to a more readable date and time.

Syntax

date(*format*,*timestamp*)

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

i)The required *format* parameter of the date() function specifies how to format the date (or time).
Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- 1 - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.
The example below formats today's date in three different ways:

Example

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

Output:- Today is 2014/08/05

Today is 2014.08.05

Today is Tuesday

ii)Get a Simple Time

Here are some characters that is commonly used for times:

- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

Example

```
<?php
echo "The time is " . date("h:i:sa");   ?>
```

Output:- The time is 01:16:20am/

iii) Get Your Time Zone

If the time you got back from the code is not the right time, it's probably because your server is in another country or set up for a different timezone.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

Example

```
<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

iv)Create a Date With PHP mktime()

The optional *timestamp* parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used (as shown in the examples above).

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d);      ?>
```

Output:- Created date is 2014-08-12 11:14:54am.

v) Create a Date From a String With PHP strtotime()

The PHP strtotime() function is used to convert a human readable string to a Unix time.

Eg.
```
<?php
$d=strtotime("10:30pm April 15 2014");
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```
output:- Created date is 2014-04-15 10:30:00pm

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";
$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";
```

```
$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

More Date Examples
The example below outputs the dates for the next six Saturdays:
Example

```
<?php
$startdate = strtotime("Saturday");
$enddate = strtotime("+6 weeks",$startdate);
while ($startdate < $enddate) {
  echo date("M d", $startdate),"<br>";
  $startdate = strtotime("+1 week", $startdate);
}
?>
```

16) PHP Include Files:-

a)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.
b)Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.
i.e 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.
c) **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.

d) 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.
e) 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.
f) Syntax:-

include '*filename*';   or

require '*filename*';

e) PHP include Examples
Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " Example.com</p>";
?>
```

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

Example

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

Output: Welcome to my home page!

```
Some text.
Some more text.
Copyright © 1999-2014 W3Schools.com
```

f) Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.google.com/index.htm">Home</a> -
<a href="http:// www.google.com /ebxml">XML</a> -
<a href="http:// www.google.com /ajax">AJAX</a> -
<a href="http:// www.google.com /perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

```
<html>
<body>
<?php include("menu.php"); ?>
<p>This is an example to show how to include PHP file!</p>
</body></html>
```

This will produce following result

```
Home - XML - AJAX - PERL
This is an example to show how to include PHP file.
```

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

```
<html><body>
<?php include("xxmenu.php");
echo "I have a $color $car."; ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This will produce following result: I have a .

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

Now lets try same example with require() function.

```
<html><body>
<?php require("xxmenu.php");
echo "I have a $color $car."; ?>
?>
<p>This is an example to show how to include wrong PHP file!</p>
</body></html>
```

This time file execution halts and nothing is displayed.
NOTE: You may get plain warning messages or fatal error messages or nothing at all. This depends on your PHP Server configuration.
17) PHP File Handling:-

g)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
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):
Example

```
<?php
echo readfile("webdictionary.txt");
?>
```

The readfile() function is useful if all you want to do is open up a file and read its contents.
The next chapters will teach you more about file handling.
h)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:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
```

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:
Example

PHP

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

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

Modes	Description
r	**Open a file for read only**. File pointer starts at the beginning of the file
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
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
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file
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
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
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

i)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"));
```

j)PHP Close File - fclose()

The fclose() function is used to close an open file.

> It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

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

k)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:

Example

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

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

l)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:

Example

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

m)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:

Example

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

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

n) PHP Write to File - fwrite()

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

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

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Jspm's abacus";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

The directory functions allow you to retrieve information about directories and their contents.

Function	Description	Example

chdir()	Changes the current directory	<?php // Get current directory
		echo getcwd() . "<br>";<br>// Change directory
		chdir("images");
		// Get current directory
		echo getcwd();?>

closedir()	Closes a directory handle	<?php
		$dir = "/images/";
		// Open a directory, and read its contents
		if (is_dir($dir)){
		  if ($dh = opendir($dir)){
		    while (($file = readdir($dh)) !== false){
		      echo "filename:" . $file . "
		";
		    }
		    closedir($dh);
		  }
		}
		?>

dir()	Returns an instance of the Directory class	<?php
		$d = dir(getcwd());
		echo "Handle: " . $d->handle . "<br>";<br>echo "Path: " . $d->path . "
		";
		
		while (($file = $d->read()) !== false){
		  echo "filename: " . $file . "
		";
		}
		$d->close();

		?>
getcwd()	Returns the current working directory	<?php echo getcwd() ?>

opendir()	Opens a directory handle	<?php
		$dir = "/images/";
		
		// Open a directory, and read its contents
		if (is_dir($dir)){
		  if ($dh = opendir($dir)){
		    while (($file = readdir($dh)) !== false){
		      echo "filename:" . $file . "
		";
		    }
		    closedir($dh);
		  }
		}
		?>

readdir()	Returns an entry from a directory handle	?php
		$dir = "/images/";
		
		// Open a directory, and read its contents
		if (is_dir($dir)){
		  if ($dh = opendir($dir)){
		    while (($file = readdir($dh)) !== false){
		      echo "filename:" . $file . "
		";
		    }
		    closedir($dh);
		  }
		}
		?>

rewinddir()	The rewinddir() function resets the directory handle created by opendir().
	Open a directory, list its files, reset directory handle, list its files once again, then close:
		<?php
		$dir = "/images/";
		// Open a directory, and read its contents
		if (is_dir($dir)){
		  if ($dh = opendir($dir)){
		    // List files in images directory
		    while (($file = readdir($dh)) !== false){
		      echo "filename:" . $file . "
		";

		}
		 rewinddir();
		 // List once again files in images directory
		 while (($file = readdir($dh)) !== false){
		 echo "filename:" . $file . "
		";
		 }
		 closedir($dh);
		 }
		 }
		 ?>

scandir()	Returns an array of files and directories of a specified directory	<?php
		 $dir = "/images/";
		 // Sort in ascending order - this is default
		 $a = scandir($dir);
		 // Sort in descending order
		 $b = scandir($dir,1);
		 print_r($a); print_r($b);?>

18)PHP Error Handling:-

a)The default error handling in PHP is very simple. An error message with filename, line number and a message describing the error is sent to the browser.

b) PHP Error Handling

When creating scripts and web applications, error handling is an important part. If your code lacks error checking code, your program may look very unprofessional and you may be open to security risks.

We will show different error handling methods:

i)Simple "die()" statements.

ii)Custom errors and error triggers

iii)Error reporting

i)Basic Error Handling: Using the die() function

The first example shows a simple script that opens a text file:

```
<?php
$file=fopen("welcome.txt","r");
?>
```

If the file does not exist you might get an error like this:

Warning: fopen(welcome.txt) [function.fopen]: failed to open stream:
No such file or directory in **C:\webfolder\test.php** on line **2**

To prevent the user from getting an error message like the one above, we test whether the file exist before we try to access it and if not present give you your error message .

```
<?php
```

```
if(!file_exists("welcome.txt")) {
  die("File not found");
} else {
  $file=fopen("welcome.txt","r");
}
?>
```

Now if the file does not exist you get an error like this:File not found

ii) Creating a Custom Error Handler

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.

This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

error_function(error_level,error_message, error_file,error_line,error_context)

Parameter	**Description**
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

Value	**Constant**	**Description**
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

Now lets create a function to handle errors:

```
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}
```

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.

aa) Set Error Handler

i)The default error handler for PHP is the built in error handler. We are going to make the function above the default error handler for the duration of the script.

ii)It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler for all errors:

```
set_error_handler("customError");
```

iii)Since we want our custom function to handle all errors, the set_error_handler() only needed one parameter, a second parameter could be added to specify an error level.

e.g Testing the error handler by trying to output variable that does not exist:

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr";
}
set_error_handler("customError");
        echo($test);    ?>
```

The output of the code above should be something like this: **Error:** [8] Undefined variable: test

iii)Trigger an Error

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the trigger_error() function.

Example

In this example an error occurs if the "test" variable is bigger than "1":

```
<?php
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below");
}
?>
```

The output of the code above should be something like this:

Notice: Value must be 1 or below in **C:\webfolder\test.php** on line **6**

An error can be triggered anywhere you wish in a script, and by adding a second parameter, you can specify what error level is triggered.Possible error types:

- E_USER_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted

- E_USER_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted
- E_USER_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally

Example

In this example an E_USER_WARNING occurs if the "test" variable is bigger than "1". If an E_USER_WARNING occurs we will use our custom error handler and end the script:

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}
//set error handler
set_error_handler("customError",E_USER_WARNING);
//trigger error
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

The output of the code above should be something like this:

Error: [512] Value must be 1 or below
Ending Script

iv)Error Logging

a)By default, PHP sends an error log to the server's logging system or a file, depending on how the error_log configuration is set in the php.ini file. By using the error_log() function you can send error logs to a specified file or a remote destination.
Sending error messages to yourself by e-mail can be a good way of getting notified of specific errors.

b)Send an Error Message by E-Mail

In the example below we will send an e-mail with an error message and end the script, if a specific error occurs:

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "someone@example.com","From: webmaster@example.com");
}
//set error handler
set_error_handler("customError",E_USER_WARNING);
```

```
//trigger error
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

The output of the code above should be something like this:

```
Error: [512] Value must be 1 or below
Webmaster has been notified
```

And the mail received from the code above looks like this:

```
Error: [512] Value must be 1 or below
```

This should not be used with all errors. Regular errors should be logged on the server using the default PHP logging system.

19)PHP Exception Handling:-

a)What is an Exception

i)Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

ii)This is what normally happens when an exception is triggered:

- The current code state is saved
- The code execution will switch to a predefined (custom) exception handler function
- Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

iii)We will show different error handling methods:

- Basic use of Exceptions
- Creating a custom exception handler
- Multiple exceptions
- Re-throwing an exception
- Setting a top level exception handler

Note: Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

b)Basic Use of Exceptions

i)When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.

ii)If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

iii)Lets try to throw an exception without catching it:

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
//trigger exception
checkNum(2);                    ?>
```

The code above will get an error like this:

Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in **C:\webfolder\test.php** on line **6**

c)Try, throw and catch

To avoid the error from the example above, we need to create the proper code to handle an exception.
Proper exception code should include:

1. Try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
2. Throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
3. Catch - A "catch" block retrieves an exception and creates an object containing the exception information

Lets try to trigger an exception with valid code:

```
<?php
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
}
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();  }
?>
```

The code above will get an error like this: Message: Value must be 1 or below

d)Creating Custom Exception Handler

You can define your own custome excpetion handler. Use following function to set a user-defined exception handler function. There are following functions which can be used from **Exception** class.

- **getMessage()-** message of exception
- **getCode() -** code of exception
- **getFile() -** source filename
- **getLine() -** source line
- **getTrace() -** n array of the backtrace()
- **getTraceAsString() -** formated string of trace

i)Creating a Custom Exception Class

i)Creating a custom exception handler is quite simple. We simply create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

ii)The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}
$email = "someone@example...com";
try {
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
    throw new customException($email);
  }
}
catch (customException $e) {
   echo $e->errorMessage();}
?>
```

ii)The set_exception_handler():-

function sets a user-defined function to handle all uncaught exceptions.

```
<?php
function myException($exception) {
  echo "<b>Exception:</b> " . $exception->getMessage();
}
set_exception_handler('myException');
throw new Exception('Uncaught Exception occurred');
?>
```

The output of the code above should be something like this:

Exception: Uncaught Exception occurred

In the code above there was no "catch" block. Instead, the top level exception handler triggered. This function should be used to catch uncaught exceptions.

20) PHP 5 Form Handling

i)The PHP superglobals $_GET and $_POST are used to collect form-data.

ii)PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example

```
<html><body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body></html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.
iii)To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html><body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body></html>
```

iv)The output could be something like this: Welcome John
Your email address is john.doe@example.com
Same example we can write with get method.instead of post we just write get method.
iii)Differnce between get and post:-

a)GET vs. POST

i)Both GET and POST create an array (e.g. array(key => value, 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.
ii)Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
iii)$_GET is an array of variables passed to the current script via the URL parameters.
iv)$_POST is an array of variables passed to the current script via the HTTP POST method.

b)When to use GET?

i)Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. Ii)The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
iii)GET may be used for sending non-sensitive data.
iv)Note: GET should NEVER be used for sending passwords or other sensitive information!

c)When to use POST?

i)Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.
ii)Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
iii)However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

21) PHP 5 Form Validation

i)For validation consider following form:-
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male

ii)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 &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

iv) PHP Form Security

aa)The $_SERVER["PHP_SELF"] variable can be used by hackers!
bb)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.
cc)Assume we have the following form in a page named "test_form.php":
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
Now, if a user enters the normal URL in the address bar like "http://localhost/test_form.php", the above code will be translated to: <form method="post" action="test_form.php">
So far, so good.
dd)However, consider that a user enters the following URL in the address bar:
http:// localhost/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
In this case, the above code will be translated to:
ee)<form method="post" action="test_form.php"/><script>alert('hacked')</script>
This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.
Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

ff)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!
Following is code :-

PHP

```
<?php
// define variables and initialize with empty values
$nameErr = $addrErr = $emailErr = $howManyErr = $formCountriesErr = "";
$name = $address = $email = $howMany = formCountries="";
 if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    }
    else {
        $name = $_POST["name"];
 if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
       $nameErr = "Only letters and white space allowed";
     }
    }
     if (empty($_POST["address"])) {
        $addrErr = "Missing";
    }
    else {
        $address = $_POST["address"];
 if (!preg_match("/^[a-zA-Z0-9 ]*$/",$name)) {
       $nameErr = "Only letters ,white space and numbers
allowed";
     }
    }
     if (empty($_POST["email"]))  {
        $emailErr = "Missing";
    }
    else {
        $email = $_POST["email"];
 if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
       $emailErr = "Invalid email format";
     }
    }
     if (!isset($_POST["howMany"])) {
        $howManyErr = "You must select 1 option";
    }
    else {
        $howMany = $_POST["howMany"];
    }

   if (!isset($_POST["formCountries"])) {
       $formCountriesErr = "You did'nt select any country";
    }
   else {
      $formCountries = $_POST["formCountries"];
   }
 }
?>
<form method="POST" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
 Name: <input type="text" name="name" value="<?php echo $name;?>">
   <span class="error">* <?php echo $nameErr;?></span>    <br><br>
Address <input type="text" name="address" value="<?php echo $address;?>">
   <span class="error">* <?php echo  $addrErr;?></span>    <br><br>
 E-mail: <input type="text" name="email" value="<?php echo $email;?>">
   <span class="error">* <?php echo $emailErr;?></span>    <br><br>
<input type="radio" name="howMany" value="zero"
 <?php if (isset($howMany) && $howMany == "zero") echo "checked"; ?> >  0 <br>
<input type="radio" name="howMany"  value="one"
 <?php if (isset($howMany) && $howMany == "one") echo "checked"; ?> > 1<br>
<input type="radio" name="howMany" value="two"
 <?php if (isset($howMany) && $howMany == "two") echo "checked"; ?> > 2<br>
<input type="radio" name="howMany" value="twoplus"
 <?php if (isset($howMany) && $howMany == "twoplus") echo "checked"; ?> > More than 2
 <span class="error">* <?php echo $howManyErr;?></span>   <br><br>
<select multiple="multiple" name="formCountries">
   <option value="US" <?php if (isset($formCountries) && $formCountries == "US") echo "checked";
?>>United States</option>
   <option value="UK" <?php if (isset($formCountries) && $formCountries == "UK") echo "checked";
?>>United Kingdom</option>
   <option value="France" <?php if (isset($formCountries) && $formCountries == "France") echo
"checked"; ?> >France</option>
   <option value="Mexico" <?php if (isset($formCountries) && $formCountries == "Mexico") echo
"checked"; ?> >Mexico</option>
   <option value="Russia" <?php if (isset($formCountries) &&  $formCountries == "Russia") echo
"checked"; ?> >Russia</option>
   <option value="Japan" <?php if (isset($formCountries) && $formCountries == "Japan") echo
"checked"; ?> >Japan</option>
</select>
<span class="error">* <?php echo $formCountriesErr;?></span>   <br><br>
 <input type="submit" name="submit" value="Submit"></form>
```

21] PHP Cookies:-A cookie is often used to identify a user.

a)Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.
b)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.

b)How to Create a Cookie?
The setcookie() function is used to set a cookie.
Note: The setcookie() function must appear BEFORE the <html> tag.

Syntax

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

Here is the detail of all the arguments:

- **Name -** This sets the name of the cookie and is stored in an environment variable call 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 00:00:00 GMT on 1st Jan 1970. After this tir cookie will become inaccessible. If this parameter is not set then cookie will automatically expire wh the Web Browser is closed.
- **Path -**This specifies the directories for which the cookie is valid. A single forward slash charact 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 lea 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 transmissi using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Example 1

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

```
<?php
setcookie("user","John", time()+(3600), "/");
?>
```

Example 2

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.
cookie will expire after 30 days. 60*60*24=86400 = 1 day

```
<?php
$expire=time()+60*60*24*30;
setcookie("user", "Alex ", $expire);
?>
```

In the example above the expiration time is set to a month (*60 sec * 60 min * 24 hours * 30 days*).

c)How to Retrieve a Cookie Value?
The PHP $_COOKIE variable is used to retrieve a cookie value.
In the example below, we retrieve the value of the cookie named "user" and display it on a page:
```
<?php
// Print a cookie
echo $_COOKIE["user"];
// A way to view all cookies
print_r($_COOKIE);
?>
```
In the following example we use the isset() function to find out if a cookie has been set:
```
<html><body>
<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br>";
else
  echo "Welcome guest!<br>";
?>
</body></html>
```
d)How to Delete a Cookie?
When deleting a cookie you should assure that the expiration date is in the past.
Delete example:
```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time()-3600);
?>
```
22] PHP Sessions:

a)A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

b)Starting a PHP Session
Before you can store user information in your PHP session, you must first start up the session.
Note: The session_start() function must appear BEFORE the <html> tag:
```
<?php session_start(); ?>
```
The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

c)Storing a Session Variable:
Session variables are stored in associative array called $_SESSION[]. These variables can be accessed during lifetime of a session.
```
<?php
session_start();    // Start the session
```

```
?>
<!DOCTYPE html>
<html><body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?></body></html>
```

d) Get Session Variable Values

```
<?php
session_start();
?>
<!DOCTYPE html>
<html><body>
<?php
// Echo session variables that were set on previous example
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body></html>
```

Another way to show all the session variable values for a user session is to run the following code:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html><body>
<?php
print_r($_SESSION);
?>
</body></html>
```

e)Modify a PHP Session Variable

To change a session variable, just overwrite it:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html><body>
<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
```

```
?>
</body></html>
```

f)The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

```
<?php
      session_start();
      if(isset($_SESSION["count"])) {
          $accesses = $_SESSION["count"] + 1;
       } else {
          $accesses = 1;
       }
        $_SESSION["count"] = $accesses;
  ?>
  <html>  <head>  <title>Access counter</title>  </head>
    <body>
 <h1>Access counter</h1>
   <p>You have accessed this page <?php echo $accesses; ?> times today.</p>
   </body>  </html>
```

d)Destroying a Session

If you wish to delete some session data, you can use the unset() or the session_destroy() function.
The unset() function is used to free the specified session variable:

```
<?php
session_start();
if(isset($_SESSION['views']))
  unset($_SESSION['views']);
?>
```

You can also completely destroy the session by calling the session_destroy() function:

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

Note: session_destroy() will reset your session and you will lose all your stored session data.

23)Sending email:-

a)PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the the message and the actual message additionally there are other two optional parameters.

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

b)Here is the description for each parameters.

Parameter	**Description**
to	Required. Specifies the receiver / receivers of the email

subject	Required. Specifies the subject of the email. This parameter cannot contain any newline characters
message	Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters
headers	Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n)
parameters	Optional. Specifies an additional parameter to the sendmail program

c)As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed.
d)Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.
e)Example:
Following example will send an HTML email message to xyz@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<html><head><title>Sending email using PHP</title></head><body>
<?php
  $to = "xyz@somedomain.com";
  $subject = "This is subject";
  $message = "This is simple text message.";
  $header = "From:abc@somedomain.com \r\n";
  $retval = mail ($to,$subject,$message,$header);
  if( $retval == true )
  {     echo "Message sent successfully...";
  }
  else
  {     echo "Message could not be sent...";
  }
?>
</body></html>
```

24) **Object Oriented Concepts in PHP:-**
a)We can imagine our universe made of different objects like sun, earth, moon etc. Similarly we can imagine our car made of different objects like wheel, steering, gear etc. Same way there is object oriented programming concepts which assume everything as an object and implement a software using different objects.
Impb)important terms related to Object Oriented Programming.

- **Class:** This is a programmer-defined datatype, 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.

- **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.
- **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.
- **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.
- **Polymorphism:** This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it make take different number of arguments and can do different task.
- **Overloading:** a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.
- **Data Abstraction:** Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation:** refers to a concept where we encapsulate all the data and member functions together to form an object.
- **Constructor:** refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructors:** refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

a)Defining PHP Classes:

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

```
<?php
class  Books{    /* Member variables */
   var $price;    var $title;
   /* Member functions */
   function setPrice($par){
     $this->price = $par;
   }
   function getPrice(){
     echo $this->price ."<br/>";
   }
   }
?>
```

The variable **$this** is a special variable and it refers to the same object ie. itself.
b)Here is the description of each line:

- The special form **class**, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form **var**, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.
- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

d)Creating Objects in PHP
Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using **new** operator.

```
$physics = new Books;
$maths = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existance separately.
Next we will see how to access member function and process member variables.
e)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.
Following example shows how to set prices for the two books by calling member functions.

```
$physics->setPrice( 10 );
$maths->setPrice( 7 );
```

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

```
$physics->getPrice();
$maths->getPrice();
```

This will produce follwoing result: 10, 7

f)Constructor Functions:
Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behaviour, by initializing many things through constructor functions.
PHP provides a special function called **__construct()** to define a constructor. You can pass as many as arguments you like into the constructor function.
Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
 function __construct( $par1, $par2 ){
   $this->price = $par1;
   $this->title = $par2;
 }
```

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

```
   $physics = new Books( 10 );
   $maths = new Books (15 );
   /* Get those set values */

   $physics->getPrice();
     $maths->getPrice();
```

This will produce following result: 10 15

g)Destructor:

Like a constructor function you can define a destructor function using function **__destruct()**. You can release all the resourceses with-in a destructor.

Example using constructor and desructor function

```
<?php
class  Books{
   /* Member variables */
   var $price;
   var $title;
   /* Member functions */
function __construct( $par1, $par2 ){
  $this->title = $par1;
  $this->price = $par2;
}
      function getPrice(){
     echo $this->price ."<br/>";
   }
     function getTitle(){
     echo $this->title ." <br/>";
   }
function __destruct()
{echo "destructor called";
}
}//End of class
$physics = new Books( "Physics for High School", 10 );
$maths = new Books ( "Advanced Chemistry", 15 );
/* Get those set values */
$physics->getTitle();
```

```
$maths->getTitle();

$physics->getPrice();
$maths->getPrice();
?>
```

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

```
class Child extends Parent {
  <definition body>
}
```

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:

- Automatically has all the member variable declarations of the parent class.
- Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

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

```
class Novel extends Books{
  var publisher;
  function setPublisher($par){
    $this->publisher = $par;
  }
  function getPublisher(){
    echo $this->publisher. "<br />";
  }
}
```

Now apart from inherited functions, class Novel keeps two additional member functions.
i)Function Overriding:
Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.
In the follwoing example getPrice and getTitle functions are overriden to retrun some values.

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

j)Public Members:
Unless you specify otherwise, properties and methods of a class are public. That is to say, they

may be accessed in three possible situations:

- From outside the class in which it is declared
- From within the class in which it is declared
- From within another class that implements the class in which it is declared

Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class then you define class members as **private** or **protected**.

Private members:

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.

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

```
class MyClass {
  private $car = "skoda";
  $driver = "SRK";
  function __construct($par) {
    // Statements here run every time
    // an instance of the class
    // is created.
  }
  function myPublicFunction() {
    return("I'm visible!");
  }
  private function myPrivateFunction() {
    return("I'm  not visible outside!");
  }
}
```

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

k)Protected members:

A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using **protected** keyword infront of the member.

Here is different version of MyClass:

```
class MyClass {   protected  $car = "skoda";
  $driver = "SRK";
  function __construct($par) {      // Statements here run every time
    // an instance of the class  is created.
  }
  function myPublicFunction() {
    return("I'm visible!");
  }
  protected function myPrivateFunction() {
    return("I'm  visible in child class!");
  } }
```

l)Interfaces:

Interfaces are defined to provide a common function names to the implementors. Different implementors can implement those interfaces according to theri requirements. You can say, interfaces are skeltons which are implemented by developers.

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

```
  interface Mail {
    public function sendMail();   }
```

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

```
  class Report implements Mail {
    // sendMail() Definition goes here  }
```

m)Constants:

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

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

```
  class MyClass {
    const requiredMargin = 1.7;
    function __construct($incomingValue) {
      // Statements here run every time
      // an instance of the class
      // is created.
    }
  }
```

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

n)Abstract Classes:

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword **abstract**, like this:

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

```
  abstract class MyAbstractClass {
    abstract function myAbstractFunction() {
    }
  }
```

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

o)Static Keyword:

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

Try out following example:

```
<?php
class Foo
{   public static $my_static = 'foo';
    public function staticValue() {
        return self::$my_static;
    }
}
print Foo::$my_static . "\n";
$foo = new Foo();
print $foo->staticValue() . "\n";
```

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

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

```
<?php
class BaseClass {
   public function test() {
      echo "BaseClass::test() called<br>";
   }
     final public function moreTesting() {
      echo "BaseClass::moreTesting() called<br>";
   }
}
class ChildClass extends BaseClass {
   public function moreTesting() {
      echo "ChildClass::moreTesting() called<br>";
   }
}?>
```

q) Calling parent constructors

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

```
<?php
class Name
{
   var $_firstName;
   var $_lastName;
   function Name($first_name, $last_name)
   {
     $this->_firstName = $first_name;
     $this->_lastName = $last_name;
   }
   function display() {
     return($this->_lastName .", " .$this->_firstName);
   }
 }
 class NameSub1 extends Name
 {
   var $_middleInitial;
   function NameSub1($first_name, $middle_initial, $last_name) {
      Name::Name($first_name, $last_name);
      $this->_middleInitial = $middle_initial;
   }
   function display() {
      return(Name::display() . " " . $this->_middleInitial);
   }
 }
 $a=new NameSub1("abc","pqr","xyz");
 $b=$a->display();
 echo $b;
 ?>
```

In this example, we have a parent class (Name), which has a two-argument constructor, and a subclass (NameSub1), which has a three-argument constructor. The constructor of NameSub1 functions by calling its parent constructor explicitly using the :: syntax (passing two of its arguments along) and then setting an additional field. Similarly, NameSub1 defines its nonconstructor display() function in terms of the parent function that it overrides.

25)Code to inset ,update ,delete and display

```
<form method="POST"  action="connectform.php">
 Name: <input type="text" name="name" ><br>
  Address <input type="text" name="address" value=""><br>
   E-mail: <input type="text" name="email" value=""><br>
   <input type="radio" name="howMany"
 <?php if (isset($howMany) && $howMany == "zero") echo "checked"; ?>
 value="zero"> 0<br>
<input type="radio" name="howMany"
```

PHP

```
 <?php if (isset($howMany) && $howMany == "one") echo "checked"; ?>
 value="one"> 1<br>
<input type="radio" name="howMany"
 <?php if (isset($howMany) && $howMany == "two") echo "checked"; ?>
 value="two"> 2<br>
<input type="radio" name="howMany"
 <?php if (isset($howMany) && $howMany == "twoplus") echo "checked"; ?>
 value="twoplus"> More than 2 <br>

 <select multiple="multiple" name="formCountries">
   <option value="US" <?php if (isset($formCountries) && $formCountries == "US") echo "checked";
?>>United States</option>
   <option value="UK" <?php if (isset($formCountries) && $formCountries == "UK") echo "checked";
?>>United Kingdom</option>
   <option value="France" <?php if (isset($formCountries) && $formCountries == "France") echo
"checked"; ?> >France</option>
   <option value="Mexico" <?php if (isset($formCountries) && $formCountries == "Mexico") echo
"checked"; ?> >Mexico</option>
   <option value="Russia" <?php if (isset($formCountries) && $formCountries == "Russia") echo
"checked"; ?> >Russia</option>
   <option value="Japan" <?php if (isset($formCountries) && $formCountries == "Japan") echo
"checked"; ?> >Japan</option>
</select> <br>
<input type="submit" name="submit" value="Submit">
<input type="submit" name="display" value="display">
<input type="submit" name="Update" value="Update">
<input type="submit" name="Delete" value="Delete">  </form>
```

connection.php:-

```
<?php
define('DB_HOST', 'localhost');
define('DB_USER','root');
define('DB_PASSWORD','');

$con=mysql_connect(DB_HOST,DB_USER,DB_PASSWORD) or die("Failed to connect to MySQL: " .
mysql_error());
 mysql_query("CREATE DATABASE USER");
$db=mysql_select_db('USER',$con) or die("Failed to connect to MySQL: " . mysql_error());
```

```
?>
connectform.php:-
<?php
include 'connection.php';
if(isset($_POST['submit']))
{
 mysql_query("CREATE TABLE usertable ( Name VARCHAR(30),Addres VARCHAR(30),Email
VARCHAR(30),gender VARCHAR(30),country  VARCHAR(30))");
$name = $_POST["name"];
 $address = $_POST["address"];
$email = $_POST["email"];
$howMany = $_POST["howMany"]; if(isset($_POST['formCountries'])) {    $qty =
$_POST['formCountries'];
$query = "INSERT INTO
usertable(Name,Addres,Email,gender,country)VALUES('$name','$address','$email','$howMany','{$qty}
')";
$result = mysql_query($query);
if($result)
        {            echo "Successfully inserted in database";
        }
        else
        {          die('Error: '.mysql_error($con));
        }
}
echo'<a href="testconn.php">Back</a>';
mysql_close($con);
}
if(isset($_POST['display']))
{
$result = mysql_query("SELECT * FROM usertable"); // selecting data through mysql_query()
echo '<table border=1px>';  // opening table tag
echo'<th>Name</th><th>Addres</th><th>Email</th><th>gender</th><th>country</th>';
while($data = mysql_fetch_array($result))
{
echo'<tr>'; // printing table row
echo'<td>'.$data['Name'].'</td>';
echo'<td>'.$data['Addres'].'</td>';
echo'<td>'.$data['Email'].'</td>';
echo'<td>'.$data['gender'].'</td>';
echo'<td>'.$data['country'].'</td>';
echo'</tr>'; // closing table row
}
echo '</table>';  //closing table tag
echo'<a href="testconn.php">Back</a>';
        mysql_close($con);
}
if(isset($_POST['Update']))
{
?>
<html>       <head>
        <title>Form Edit Data</title>     </head>
                <body>
        <table border=1>
          <tr>     <td align=center>Form Edit Employees Data</td>           </tr>
          <tr>
   <td>
            <table>
            <?php
           $name1=$_POST['name'];
$query="SELECT * FROM usertable WHERE Name='$name1'";
            $result=mysql_query($query);
           $row=mysql_fetch_array($result);
            ?>
            <form method="post" action="edit_data.php">
                    <input type="hidden" name="name" value="<?php echo "$row[Name]";?>">
             <tr>
              <td>address</td>
 <td>  <input type="text" name="address" value="<?php echo "$row[Addres]";?>">
              </td>
             </tr>
             <tr>   <td>Email</td>
 <td>      <input type="text" name="email" value="<?php echo "$row[Email]";?>">
              </td>
             </tr>
            <tr>  <td>Gender</td>
 <td>      <input type="text" name="gender" value="<?php echo "$row[gender]";?>">
                  </td>
                 </tr>
                        <tr>      <td>Cuntry</td>
          <td>     <input type="text" name="country" value="<?php echo "$row[country]";?>">
                  </td>
                 </tr>
  <td align="right">    <input type="submit" name="submit value" value="Edit">     </td>
                 </tr>
               </form>
               </table>
             </td>          </tr>
          </table>
          </body>       </html>
<a href="testconn.php">Back</a>
          <? php  mysql_close($con);
 }
if(isset($_POST['Delete']))
{  $name1=$_POST['name'];
             $sql="DELETE FROM usertable WHERE name='$name1'";
$result=mysql_query($sql);
// if successfully deleted
if($result){   echo "Deleted Successfully";   echo "<BR>";
echo "<a href='testconn.php'>Back to main page</a>";
}
else {  echo "ERROR";
}
mysql_close($con);
}
?>
edit_data:-
<?php
include 'connection.php';
$name=$_POST['name'];
          $address = $_POST['address'];
$email = $_POST['email'];
```

PHP

```
$gender = $_POST['gender'];
$country = $_POST['country'];
        $order = "UPDATE usertable
              SET Addres='$address',Email='$email',gender='$gender',country='$country' WHERE 
Name='$name'";
        mysql_query($order);
if($order){  echo "Successfully updated\n";
echo "<a href='testconn.php'>Back to main page</a>";
}
else {echo "ERROR";
}
?>
```