BACKEND WEB DEVELOPMENT

USING MYSQL AND PHP

- **Simple**

It is very simple and easy to use, compared to another scripting language. This is widely used all over the world.

- **Interpreted**

It is an interpreted language, i.e. there is no need for compilation.

- **Faster**

It is faster than other scripting languages e.g. asp and jsp.

- **Open Source**

Open source means you no need to pay for using PHP, you can free download and use.

- **Platform Independent**

PHP code will be run on every platform, Linux, Unix, Mac OS X, Windows.

- **Flexibility**

PHP is known for its flexibility and embedded nature as it can be well integrated with HTML, XML, Javascript and many more. PHP can run on multiple operating systems likeWindows, Unix, Mac OS, Linux, etc.

- **Case Sensitive**

PHP is case sensitive scripting language at the time of variable declaration. In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.

1.3 Variables

- Variables are used to store both numeric and non numeric data.
- The content of variable can be altered during program execution
- variables can be compared and manipulated using operators.

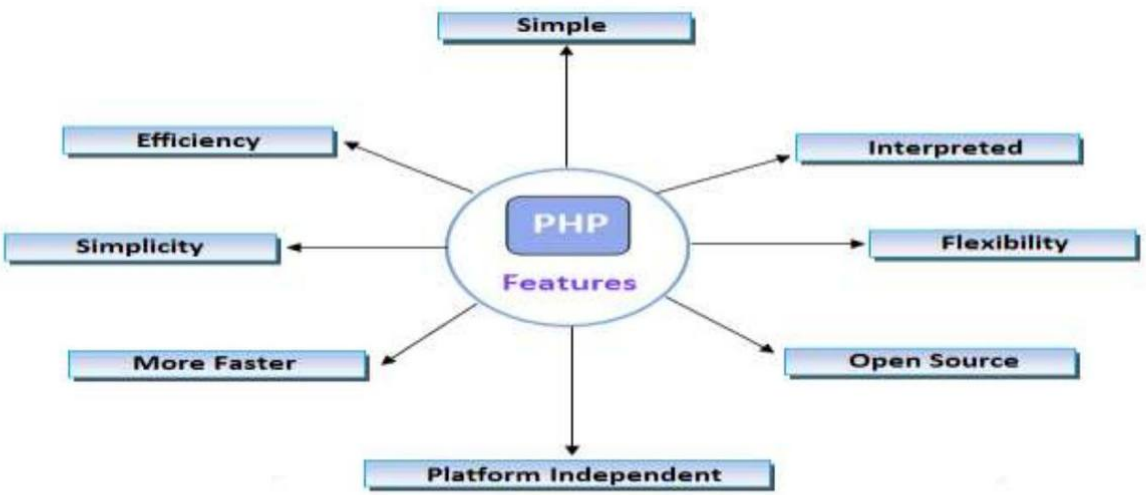
Programming with PHP and MySQL

UNIT -1

1.1 History of PHP

- PHP (PHP: Personal Home Page) was created by Rasmus Lerdorf in It was initially developed for HTTP usage logging and server-side form generation in Unix.
- PHP 2 (1995) transformed the language into a Server-side embedded scripting language. Added database support, file uploads, variables, arrays, recursive functions, conditionals, iteration, regular expressions, etc.
- PHP 3 (1998) added support for ODBC data sources, multiple platform support, protocols and new parser written by Zeev Suraski and Andi Gutmans .
- PHP 4 (2000) became an independent component of the web server for added efficiency. The parser was renamed the Zend Engine. Many security features were added.
- PHP 5 (2004) adds Zend Engine II with object oriented programming, robust XML support using the libxml2 library, SOAP extension for interoperability with Web Services, SQLite has been bundled with PHP

1.2Features of PHP



```
<?php
$today= “Aug 8 2020”;
Echo “Today is $today”;
?>
```

- PHP supports number of specialize functions to check if a variable or value belong to a specific type

1. is_bool()
2. is_string()
3. is_numberic()
4. is_float()
5. is_int()
6. is_null()
7. is_array()

- **Echo()** function is used to print data to standard output.

Example of Integer values

```
<html>
<head><title>Example</title></head>
<body>
<?php
// define variable
$first = 100;
$second = 200;
$third = $first + $second;
// print output
echo "Sum = "$third;
?>
</body>
<html>
```

Output: Sum = 300

Example of string values

```
<?php
$txt = "W3Schools.com";
echo "I love " . $txt . "!";
?>
```

Output:

I love W3Schools.com!

NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.
- If a variable is created without a value, it is automatically assigned a value of NULL.

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

Output: NULL

1.4 Statement Operators

- Operators are used to perform operations on variables and values.
- PHP divides the operators in the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Increment/Decrement operators
5. Logical operators

- All variables in PHP are denoted with a leading dollar sign ($).
- The variable name must begin with a letter or the underscore character.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- **PHP has a total of eight data types which we use to construct our variables -**

1. Integers - are whole numbers, without a decimal point, like 4195.

2. Doubles - are floating-point numbers, like 3.14159 or 49.1.

3. Booleans - have only two possible values either true or false.

4. NULL - is a special type that only has one value: NULL.

5. Strings - are sequences of characters, like 'PHP supports string operations.'

6. Arrays - are named and indexed collections of other values.

7. Objects - are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.

8. Resources - are special variables that hold references to resources external to PHP (such as database connections).

- **Assigning and Using Variables Values**
- To assign a value to a variable , use assign operator (=) symbols.
- = operator assigns a value to a variable.

Program :

```
<?php
$x = 10;
$y = 4;

echo($x + $y);

echo($x - $y);

echo($x * $y);

echo($x / $y);

echo($x % $y);
?>
```

OutPut:

```
14
6
40
2.5
2
```

3. Comparison Operators

- The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example
==	Equal	$x == $y
===	Identical	$x === $y
<>	Not equal	$x <> $y
!==	Not identical	$x !== $y
>	Greater than	$x > $y
<	Less than	$x < $y
>=	Greater than or equal to	$x >= $y
<=	Less than or equal to	$x <= $y

6. String operators Value

1.Arithmetic Operators

- The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example
+	Addition	$x + $y
-	Subtraction	$x - $y
*	Multiplication	$x * $y
/	Division	$x / $y
%	Modulus	$x % $y
**	Exponentiation	$x ** $y

2.Assignment Operators

- The PHP assignment operators are used with numeric values to write a value to a variable.
- The basic assignment operator in PHP is "=".
- It means that the left operand gets set to the value of the assignment expression on the right.

Output:

you have an A

- PHP also offer if - else(),used to defines an alternate block of code that get executed when the conditional expression in if () statement evaluates as false.

```
<?php
If(conditional test) „³ which evaluate either true or false
{
do this; „³ if true within the curly braces is executed
}
else
{
do this „³ if false within the curly braces is executed
} ?>
```

Example

```
<?php
$mark = 60;
if($mark >= 80)
{
echo "you have an A";
}
else
{
echo " you have an B";
}
?>
```

Output:

you have an B

- PHP also provide you with a way handling multiple possibilities the if-else is-else() construct.
- This construct consists of listing number of possible results, one after another, specifying the action to be taken for each.

```
if (condition1)
{
//code 1 to be executed
}
elseif(condition2)
{
//code 2 to be executed
}
else
{
//code to be executed if code 1 and code 2 are not
}
```

Example :

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
echo $txt1 . $txt2;
?>
```

Output:

Hello world!

1.5 Conditional Statement

- A Conditional Statement enables you to test whether a specific condition is true or false and to perform different actions on the basis of the test result.

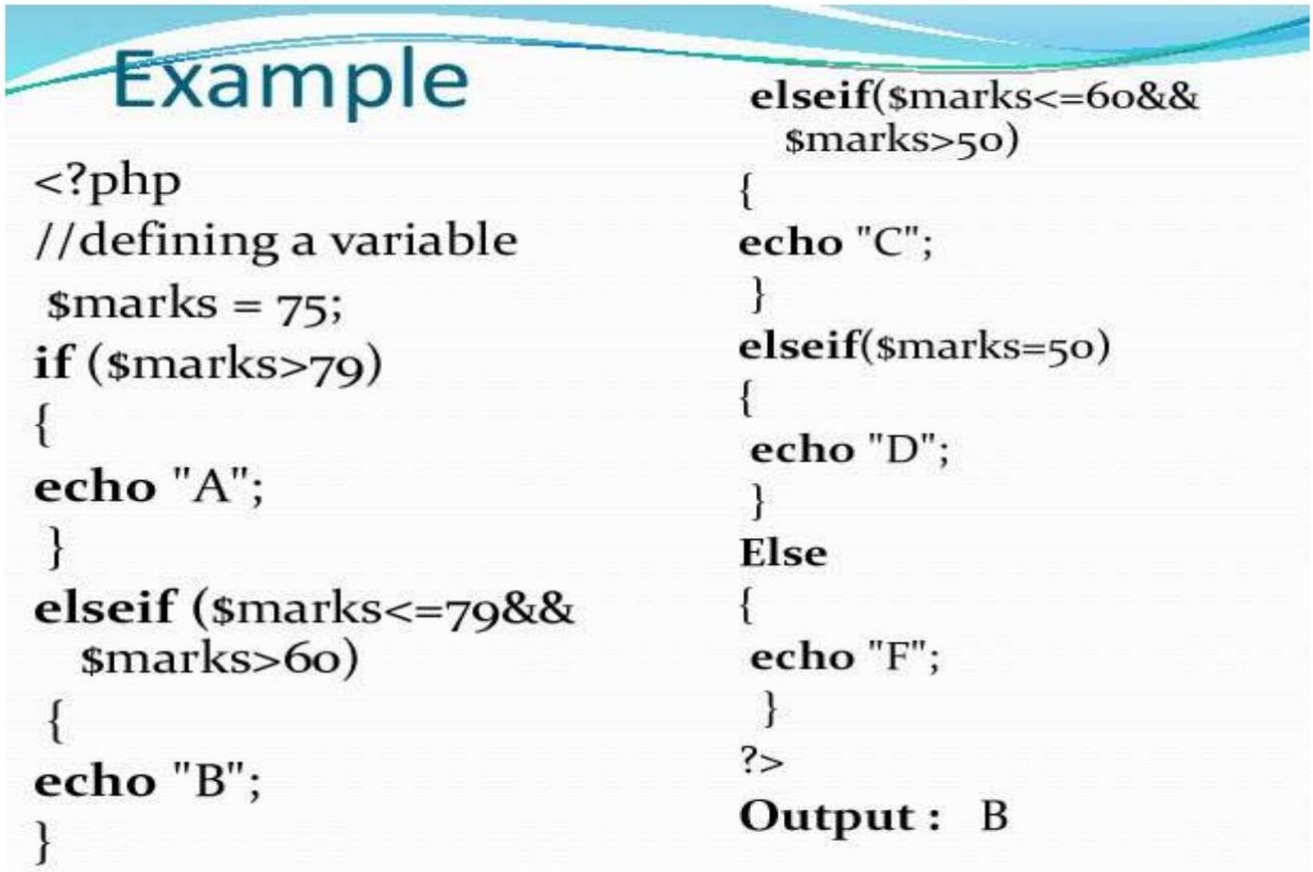
1. Using if () Statement,

In PHP, the simplest form of conditional statement is the if() statement, which looks like this:

```
<?php
If(conditional test) „³ which evaluate either true or false
{
do this; „³ if true within the curly braces is executed
}
?>
```

Example

```
<?php
$mark = 120;
if($mark >= 80)
{
echo "you have an A";
}
?>
```



2. Switch() Statement

- The **switch statement** is very similar to the **if...else statement**.
- But in the cases where your conditions are complicated like you need to check a condition with multiple constant values, a **switch statement** is preferred to an **if...else**.
- The examples below will help us better understand the **switch statements**.

```
switch (n)
{
case constant1:
// code to be executed if n is equal to constant1;
break;
case constant2:
// code to be executed if n is equal to constant2;
```

4.Increment / Decrement Operators

- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

5. Logical Operators

- The PHP logical operators are used to combine conditional statements.

Name	Example	Result
And	$x and $y (or) $x && $y	True if both $x and $y are true
Or	$x or $y (or) $x || $y	True if either $x or $y is true
Xor	$x xor $y	True if either $x or $y is true, but not both
And		True if both $x and $y are true
Not	!$x	True if $x is not true

6. String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example
.	Concatenation	$txt1 . $txt2

1.7 Merging Forms and Their Result

- Normally, when creating and processing forms in PHP, you would place the HTML form in one file and handle form processing through a separate PHP Script.
- Example you have seen so far have worked.
- With the power of conditional statement at your disposal, you can combine both pages into one.
- To do this, assign a name to the form's SUBMIT control and then check whether the special $_POST container variable contains that name when script first loads up.
- You can use a single PHP script to generate both initial form and post submission output.

Example:

```
<html>
<head><title>Merging Form</title>
</head>
<body>
<? PHP
//if the SUBMIT variable does not exist
// FORM has not been submitted
//Display initial page
If (!$_POST['submit'])
{
?>
<form action ="<?=$_SERVER['PHP_SELF']?>" method="post">
Enter Number: <input name="number" size="2">
<input type="submit" name="submit" value="Go">
</form>
```

```
break;
. . . default:
// code to be executed if n doesn't match any
constant
}
```

Example:

```
<?php
//variable definition
$gender = 'M';
switch ($gender)
{
 case 'F':
echo 'F is FEMALE';
break;
case 'M':
 echo 'M is MALE';
 break;
default:
echo 'Invalid choice';
}
 ?>
```

Output :

M is MALE

1.6 Nesting Conditional Statement

- To handle multiple conditions, you can "nest" conditional statements include each other.
- It structure will look like

```
if (expression 1 )
{
if (expression 2 )
{
if (expression 3 )
{
// statements 1
}
}
}
```

Example

```
<?PHP

$country = "India";

$state = "Tamilnadu";

$city = "Kovilpatti";

 if ($country == "India")

{

    if ($state == "Tamilnadu")

    {

            if ($city == "Kovilpatti")

       {

      echo "nice country";
}
else
{
echo "city not match";
}

        }
else
        {
        echo "state not match";
        }

}
else
{
echo "country not match";
}
?>
```

Syntax:

```
<?php
While (condition is true)
{
Do this;
}
?>
```

Example:

```
<?php
  $x = 1;

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

Output:

The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

2. Using the do() Loop

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

Syntax:

<?php

do

{

code to be executed;

} while (*condition is true*);

?>

Example

<?php

```
$x = 1;
do {
echo "The number is: $x <br>";
$x++;
} while ($x <= 5);
?>
```

Output:

The number is: 1

The number is: 2

The number is: 3

The number is: 4

The number is: 5

3. **Using the for() Loop**

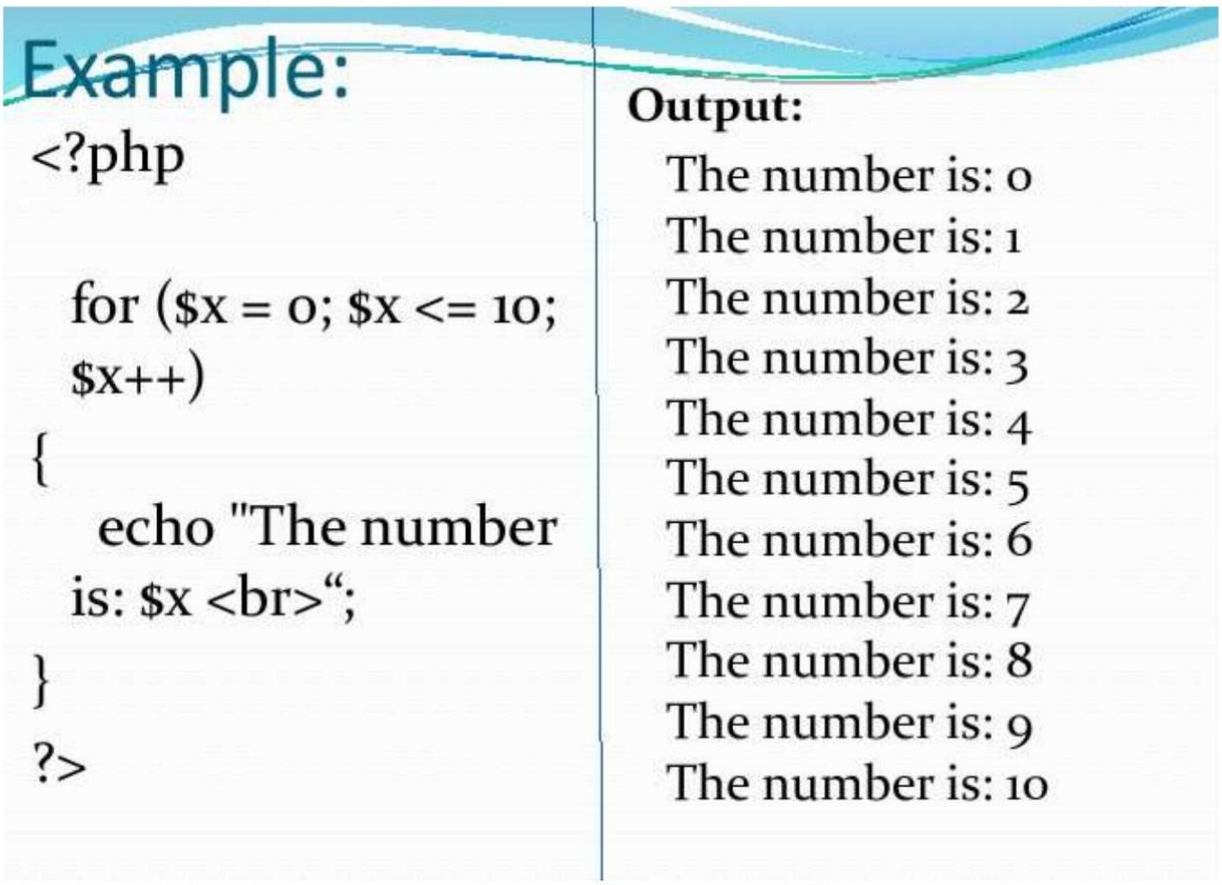
- The for loop - Loops through a block of code a specified number of times.

Syntax

```
<?php
for (init counter; test counter; increment counter)
{
code to be executed for each iteration;
}
?>
```

Parameters:

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



1.9 Controlling Loop Iteration With Break And Continue

1. Break

- The break statement can be used to jump out of a loop.

Example of Continue:

```
<?php
for ($x = 0; $x < 10; $x++)
  {
  if ($x == 4) {
    continue;
  }
  echo "The number is: $x
   <br>";
}
?>
```

Output:

The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9

</body>

</html>

1.8 Repeating Action with LOOPS

- A LOOP is a control structure that enables you to repeat the same set of statement or commands over and over again.
- The actual number of repetitions may be dependent on a number you specify or fulfillment of a certain condition or set of conditions.

1. Using While() Loop:

- The first and simplest to loop learn in PHP is called While() Loop.
- With this loop type, so long as conditional expression specified evaluates to true, the loop will continue to execute.
- When Condition become false, the loop broken and the statement will executed.

Example of Break:

```
<?php
for ($x = 0; $x < 10; $x++)
 {
  if ($x == 4)
{
    break;
  }
  echo "The number is: $x
   <br>";
}
?>
```

Output:

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
```

➢

2. Continue

➢ The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Unit – II

2.1 Arrays

- An array is a data structure that stores one or more similar type of values in a single variables.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

- An array can be created using the array() language construct.
- There are three different kind of arrays:

1. Indexed arrays - These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

ex: $car[0]="Nano",
$car[1]="Audi",
$car[2]="Hundai"

2. Associative arrays - Associative array will have their index as string so that you can establish a strong association between key and values.

ex: $car[N]="Nano", $car[A]="Audi", $car[H]="Hundai"

3. **Multidimensional arrays** - Arrays containing one or more arrays

ex:
```
$myarray = array(
array("Ankit", "Ram", "Shyam"),
array("Unnao", "Trichy", "Kanpur")
);
```

- For example, say you have this array:

$fruits[0] = "pineapple";

$fruits[1] = "pomegranate";

$fruits[2] = " apple";

- Now we want to change the value

of $fruits[1] to "watermelon“ then give as follow:

$fruits[0] = "pineapple";

$fruits[1] = "pomegranate";

$fruits[2] = " apple ";

$fruits[1] = "watermelon";

- If you want to add a new element, "grapes", to the end

of the array as follows:

$fruits[0] = "pineapple";

$fruits[1] = "pomegranate";

$fruits[2] = " apple ";

$fruits[1] = "watermelon";

$fruits[] = "grapes";

- Example program as follows:

```
echo "<li>$item";
}
?>
```

OUTPUT:

. eye

. wing

. tail

. leg

2.5 Grouping Form Selections with Arrays

- Uses arrays and loops also come in handy when processing form in PHP.
- If you have a group of related checkboxes and a multiselect list, you can use an array to capture all the selected form values in a single variable.

2.2 Creating an array

- To define an array variables,name it using standard PHP variables rules and populate it with elements using array() function as follow:

```
<?php
//define an array
$flavors =
array('strawberry','grape','vanilla','chocolate')
?>
```

- An alternative way to defines an array is by specifying values for each element using index notation like this:

```
<?php
//define an array
$flavors[0] = 'strawberry',
$flavors[1] ='grape'
$flavors[2] ='vanilla'
$flavors[3] ='chocolate'
?>
```

- To create an associative array, use keys instead of numeric indices:

```
<?php
//define an array
$flavors[S] = 'strawberry',
$flavors[G] ='grape'
$flavors[V] ='vanilla'
$flavors[C] ='chocolate'
?>
```

2.3 Modifying Arrays

- You can modify the values in arrays as easily as other variables. One way is to access an element in an array simply by referring to it by index.

Example program :

```
<?php
        $shoppinglist=array('eye','wing','tail','leg');
        for ($x=o;$x<sizeof($shoppinglist);$x++)
                {
                        echo "<li>$shoppinglist[$x]";
                }
?>
```

OUTPUT:

- Eye
- Wing
- Tail
- leg

1.The foreach() Loop

- The foreach() loop runs once for each element of array, moving forward through the array on each iteration.
- On each run, the statements within curly braces are executed and the currently selected array element is made available through a loop variable.
- Foreach() loop doesn't need a counter or call to sizeof().
- It keeps track of its position in the array automatically.

Example:

```
<?php
$shoppinglist=array('eye','wing','tail','leg');
foreach ($shoppinglist as $item)
{
```

Example

```
<?php
If (1$_POST['submit'])
{

}
Else
{
If(is_array($_POST['option']))
{
Foreach($_POST['option']
  as $o)
{
Echo "<i>$o</i>";

}
}
 Else
{
Echo "nothing selected";
}
}
```

Using Array Functions

- The array_keys() and array_values() functions come in handy to get a list of all keys and values within array.
- The **print_r**() function prints the information about a variable in a more human-readable way.

The following examples

```
<?php
//define an array
$a=array("Volvo"=>"XC90","BMW"=>"X5","Toyota"=>"Highlander");
echo "array key <br>";
print_r(array_keys($a));
echo "<br>array values<br>";
print_r(array_values($a));
```

```
<HTML>
<HEAD>
    <TITLE>
      Modifying an array
    </TITLE>
  </HEAD>

  <BODY>
    <H1>
      Modifying an array
    </H1>
<?php
  $fruits[0] = "pineapple";
  $fruits[1] = "pomegranate";
  $fruits[2] = "tangerine";
  for ($index = 0;
      $index < count($fruits);
      $index++)
{

  echo $fruits[$index], "<BR>";
}
      ?>
</BODY>
</HTML>
```

Output:

Modifying an array

pineapple
pomegranate
watermelon
grapes

2.4 Processing Arrays with Loops

• The process the data in PHP array with loop over it using any loop constructs.

• The for () loop is used through the array, extract the elements from it using index and display them one after other in an unordered list.

• The sizeof () function is to return the size of array.

OUTPUT:

BMW

- The extract() function imports variables into the local symbol table from an array.
- This function uses array keys as variable names and values as variable values.
- **Syntax**

  extract(array)

  Array --> Required. Specifies the array to use

Example:

```
<?php
$my_array = array("a" => "Cat","b" =>
"Dog", "c" => "Horse");
extract($my_array);
echo "\$a = $a; \$b = $b; \$c = $c";
?>
```

Output: $a = Cat; $b = Dog; $c = Horse

2.6 Creating User Defined Functions

- A **Function** is nothing but a 'block of statements' which generally performs a specific task and can be used repeatedly in our program.
- This 'block of statements' is also given a **name** so that whenever we want to use it in our program/script, we can call it by its **name**.
- In PHP there are thousands of built-in functions which we can directly use in our program/script.
- PHP also supports **user defined functions**, where we can define our own functions.
- we can define our own functions in our program and use those functions.
- **Syntax:**

?>

OUTPUT:

array key

Array ([0] => Volvo [1] => BMW [2] => Toyota)

array values

Array ([0] => XC90 [1] => X5 [2] => Highlander)

- The is_array() function checks whether a variable is an array or not.
- This function returns true (1) if the variable is an array, otherwise it returns false/nothing.

Syntax

is_array(*variable*);

Variable „³Required. Specifies the variable to check

Example

```
<?php
$a=array("Volvo"=>"XC90","BMW"=>"X5","To
yota"=>"Highlander");
echo is_array($a);
?>
```

OUTPUT:

1

- The list() function assigns array elements to variables.
- **Example** of list() function

```
<?php
$a=array("Volvo","BMW","Toyota");
list($a1,$a2,$a3)=$a;
echo $a2;
?>
```

```
function function_name()
{
// function code statements
}
```

- **Few Rules to name Functions**

1. A **function name** can only contain alphabets, numbers and underscores. No other special character is allowed.
2. The name should start with either an alphabet or an underscore. It should not start with a number.
3. And last but not least, function names are not case-sensitive.
4. The opening curly brace { after the function name marks the start of the function code, and the closing curly brace } marks the end of function code.

```
<?php
// defining the function
function greetings()
{
echo "Merry Christmas and a Very Happy New Year";
}
echo "Hey Martha <br/>";
// invoking the function
greetings();
echo "Hey Jon <br/>";
// invoking the function again
greetings();
?>
```

Example program to defining and invoking functions

Output:

Hey Martha

Merry Christmas and a Very Happy New Year

Hey Jon

Merry Christmas and a Very Happy New Year

2.7 Advantages of User-defined Functions

- **Reuseable Code**: As it's clear from the example above, you write a function once and can use it for a thousand times in your program.
- **Less Code Repetition**: Rather than repeating all those lines of code again and again, we can just create a function for them and simply call the function.

Easy to Understand: Using functions in your program, makes the code more readable and easy to understand.

Using Arguments and Return Values

- We can even pass data to a function, which can be used inside the function block.
- This is done using arguments.
- An argument is nothing but a variable.
- Arguments are specified after the function name, in parentheses, separated by comma.
- When we define a function, we must define the number of arguments it will accept and only that much arguments can be passed while calling the function.

Syntax:

```
<?php
/* we can have as many arguments as we
want to have in a function */
function function_name(argument1,
argument2)
```

```
{
// function code statements
}
?>
```

Example

```
<?php
// defining the function with argument
function greetings($festival)
{
echo "Wish you a very Happy $festival";
}
echo "Hey Jai <br/>";
// invoking the function
greetings("Diwali");
// next line echo "<br/>";
echo "Hey Jon <br/>";
// invoking the function again
greetings("New Year");
?>
```

Output:

Hey Jai

Wish you a very Happy Diwali

Hey Jon

Wish you a very Happy New Year

2.8 Default Function Arguments

- Sometimes function arguments play an important role in the function code execution.

- In such cases, if a user forgets to provide the argument while calling the function, it might lead to some error.
- To avoid such errors, we can provide a default value for the arguments which is used when no value is provided for the argument when the function is called.
- **Example**

```
<?php
// defining the function with default argument function
greetings($festival = "Life")
{
echo "Wish you a very Happy $festival";
}
echo "Hey Jai <br/>";
greetings("Diwali");
echo "<br/>";
echo "Hey Jon <br/>";
greetings();
?>
```

2.9 Function Overloading

- Function overloading allows you to have multiple different variants of one function, differentiated by the number and type of arguments they take.
- For example, we defined the function add() which takes two arguments, and return the sum of those two. What if we wish to provide support for adding 3 numbers.

Example

```
<?php
// add function with 2 arguments
function add($a, $b)
```

- **Mode** - Required. Specifies the type of access you require to the file/stream.There are different types of mode are listed below:

Mode	Description
r	• Read file from beginning. • Returns false if the file doesn't exist. • Read only
r+	• Read file from beginning • Returns false if the file doesn't exist. • Read and write
w	• Write to file at beginning • truncate file to zero length • If the file doesn't exist attempt to create it. • Write only
w+	• Write to file at beginning, truncate file to zero length • If the file doesn't exist attempt to create it. • Read and Write
a	• Append to file at end • If the file doesn't exist attempt to create it. • Write only
a+	• Php append to file at end • If the file doesn't exist attempt to create it • Read and write

ii. fread()

Syntax

fread (*file*, *length*)

- **File** - Required. Specifies the open file to read from
- **length** - Required. Specifies the maximum number of bytes to read

Ex: fread ($file,"10");

```
<?php
$myfile =
fopen("webdictionary.txt", "r")
echo fread($myfile,filesize("webdictionary.txt"));
```

```
{
$sum = $a + $b;
return $sum;
}
function add1($a, $b, $c)
{
$sum1 = $a + $b + $c;
return $sum1;
}
echo "5 + 10 = " . add(5, 10) . "<br/>";
// calling add with 3 arguments
echo "5 + 10 + 15 = " . add1(5, 10, 15) . "<br/>";
?>
```

Output:

5 + 10 = 15

5 + 10 + 15 = 30

2.10 Using Files

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

  i. fopen() function is unable to open the specified file.

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

  iii.fread() function reads from an open file.

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

i. fopen():

Syntax:

fopen(filename, mode)

- **Filename**- Required. Specifies the file or URL to open

```
fclose($myfile);
?>
```

iii. fwrite()

Syntax:

fwrite(*file*, *string*, *length*)

- ***File*** - Required. Specifies the open file to write to
- ***string*** - Required. Specifies the string to write to the open file
- ***length*** -Optional. Specifies the maximum number of bytes to write
- Ex: echo fwrite($file,"Hello World. Testing!");

iv. fclose()

Syntax:

fclose(*file*)

- ***file* -** Required. Specifies the file to close

Ex: fclose($file);

Example:

```
<?php
$file = fopen("test.txt","w");
echo fwrite($file,"HelloWorld. Testing! ");
fclose($file);
?>
```

2.11 Session

- When you work with an application, you open it, do some changes, and then you close it.
- This is much like a Session. The computer knows who you are. It knows when you start the application and when you end.
- But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

they are retrieved from the session we open at the beginning of each page (session_start()).

- All session variable values are stored in the global $_SESSION variable.

Example

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

Destroy a PHP Session

- To remove all global session variables and destroy the session, use session_unset() and session_destroy():

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// remove all session variables
```

- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc).
- By default, session variables last until the user closes the browser.
- Session variables hold information about one single user, and are available to all pages in one application.

Start a PHP Session

- A session is started with the session_start() function.
- Session variables are set with the PHP global variable: $_SESSION.
- The session_start() function must be the very first thing in your document. Before any HTML tags.

Example

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

Get PHP Session Variable Values

- Session variables are not passed individually to each new page, instead

```
session_unset();
// destroy the session
session_destroy();
?>
</body>
</html>
```

2.12 Cookie

- A cookie is often used to identify a user.
- A cookie is a small file that the server embeds on the user's computer.

„«Each time the same computer requests a page with a browser, it will send the cookie too.

- With PHP, you can use both create and retrieve cookie values.
- **Create Cookies With PHP**
- A cookie is created with the setcookie() function.
- **Syntax**:
- **setcookie**(*name, value, expire, path, domain, secure*);
- Only the *name* parameter is required. All other parameters are optional.
- Here is the detail of all the arguments -
- **Name -** Set of name and values of the cookie
- **Value** - Sets the value of the named variable and is the content that you actually want to store.
- **Expiry** - Sets the date and time at which the cookie expries.
- **Path** - This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** - This can be used to specify the domain name in very large domains and must contain at least two periods to be valid.

- **Security** - This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.
- Example

```
<?php
setcookie("name", "John Watkin", time()+3600, "/","", 0);
setcookie("age", "36", time()+86400, "/", "", 0);
?>
```

OUTPUT:

Time: 60 sec * 60 mins = 3600

Time: 60sec * 60 mins* 24 hours = 86,400

Retrieving Cookie Data

- Once cookie has been sent for a domain, it becomes available in the special $_COOKIE array, and its value may be accessed using array notation.

Example:

```
<? Php
If($_COOKIE[‘name’])
{
Echo “Welcome to ” . $_COOKIE[‘name’];
}
Else
{
Echo ‘Cookie not found”;
}
```

Deleting Cookie

- To delete a cookie, simply use setcookie() with its name to set the cookie’s expiry date to a value in the past.

```
<?php
Setcookie(`name`,``, time()-10000,'/');
?>
```

2.13 Dealing with Dates and Times

- The date/time functions allow you to get the date and time from the server where your PHP script runs.
- You can then use the date/time functions to format the date and time in several ways.
- The PHP date() function is used to format a date and/or a time.
- The PHP date() function formats a timestamp to a more readable date and time.
- **Syntax** :

  (*format*,*timestamp*)

format - Required. Specifies the format of the timestamp

timestamp - Optional. Specifies a timestamp. Default is the current date and time.

- A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

Get a Date

- 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)

  l (lowercase 'L') - 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:**

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

Output:

Today is 2020/10/07

Today is 2020.10.07

Today is 2020-10-07

Today is Wednesday

Create a Date With mktime()

- The optional *timestamp* parameter in the date() function specifies a timestamp. If omitted, the current date and time will be used.
- The PHP mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.
- **Syntax**

  mktime(*hour, minute, second, month, day, year*)
- The example below creates a date and time with the date() function from a number of parameters in the mktime() function:

➢ **Example**

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