Chapter 5

Programming in PHP and MySQL

Origins and Uses of PHP

- Php was developed by Rasmus Lerdorf, a member of the Apache Group, in 1994.
- PHP is an open-source product.
- PHP is an acronym for Personal Home Page, or PHP: Hypertext Preprocessor.
- PHP is used for form handling, file processing, and database access.

Overview of PHP

- PHP is a server-side scripting language whose scripts are embedded in HTML documents - Similar to JavaScript, but on the server side .
- PHP is an alternative to CGI, Active Server Pages (ASP), and Java Server Pages (JSP) .
- The PHP processor has two modes: copy (HTML) and interpret (PHP).
- It takes a PHP document file as input and produces an HTML document file.
- When the php processor finds markup code in the input file, it simply copies it to the output file.
- When the processor encounters php script in the input file, it interprets it and sends any output of the script to the input file.
- This implies that the output from a PHP script must be HTML , either of which could include embedded client-side script.
- PHP syntax is similar to that of JavaScript.
- PHP is dynamically typed, as does JavaScript. Variables are not type declared, and they have no intrinsic type.

General Syntactic Characteristics

- PHP is a server-side scripting language i.e. embedded in HTML.
- PHP scripts also are in files that are referenced by such documents.
- PHP code is embedded in document by enclosing it between *<?php* and *?>* tags.
- The include keyword is used to reference the file where php script is stored, which takes the file name as its string parameter. eg: include("database.php").
- Thus included file can contain markup or client-side script, as well as PHP code, but any PHP script it include must be the content of *<?php* tag .
- Every variable name begins with a dollar sign ($).
- PHP variable names are case sensitive .
- But, neither reserved words nor function names are case sensitive (no difference between while, WHILE, While, and wHile).

- Single line comments to be specified either with # or with // .
- Multiple line comments are specified with /* and */ .
- PHP statements are terminated with semicolons.
- Braces are used to form compound statements for control statements.
- Some reserved words of PHP are listed below:

And	Else	Global	Require	Virtual	Break
Elseif	If	Return	Xor	Case	Extends
Include	Static	While	Class	False	List
Switch	Continue	For	New	This	Default
Foreach	Not	True	Do	Function	Or
var					

Variables:

- There are no type declarations .
- An unassigned (unbound) variable has the value, *NULL*.
- The ***unset*** function sets a variable to NULL.
- The ***IsSet*** function is used to determine whether a variable is *NULL*.
- A variable starts with the $ sign, followed by the name of the variable.
- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
- Rules for PHP variables:
  - A variable starts with the $ sign, followed by the name of the variable
  - A variable name must start with a letter or the underscore
  - character A variable name cannot start with a number
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case-sensitive ($age and $AGE are two different variables)

Output

- Output from a PHP script is HTML that is sent to the browser .
- HTML is sent to the browser through standard output.
- There are three ways to produce output: ***echo***, ***print***, and ***printf***.
- ***echo*** and ***print*** take a string, but will coerce other values to strings. *echo "whatever"; # Only one parameter*
  echo("first ", $sum) # More than one
  print "Welcome to my site!"; # Only one
- The ***printf()*** function outputs a formatted string. The arg1, arg2, ++ parameters will be inserted at percent (%) signs in the main string. This function works "step-by-step". At the first % sign, arg1 is inserted, at the second % sign, arg2 is inserted, etc.

  Example:

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

```
<body>
    <?php
        $txt = "W3Schools.com";
        echo "I love $txt!";
        $x=5;
        $y=4:
        $z=$x+$Y;
        echo "The sum is $Z"."<br>";
        echo "The sum is "."$z";
        $number = 9;
        $str = "Beijing";
        printf("There are %u million bicycles in %s.",$number,$str);
        print "Hello world!<br>";
    ?>
</body>
</html>
```

PHP Data Types

- Variables can store data of different types, and different data types can do different things.
- PHP supports the following data types:
  - String – A string is a sequence of characters, like "Hello world!". A string can be any text inside quotes. You can use single or double quotes.
  - Integer – An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.
  - Float (floating point numbers - also called double) – a float (floating point number) is a number with a decimal point or a number in exponential form.
  - Boolean – a Boolean represents two possible states: TRUE or FALSE.
  - Array – an array stores multiple values in one single variable.
  - Object – An object is a data type which stores data and information on how to process that data. An object must be explicitly declared.
  - NULL – Null is a special data type which can have only one value: *NULL*.
  - Resource – The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common example of using the resource data type is a database call.

    Example:

```
<!DOCTYPE html>
<html>
<body>
   <?php
        //String
        $x = "Hello world!";
        $y = 'Hello world!';
        echo $x;
        echo "<br>";
        echo $y;
        //Integer
```

```
$x = 5985;
var_dump($x);
//The PHP var_dump() function returns the data type and
value //Floating point
$y = 10.365;
var_dump($y);
//Array
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
//class and object
class Car {
        function Car() {
        $this->model = "VW";
        }
        }
        // create an object
        $typ = new Car();
        // show object properties
        echo $typ->model;
?>
</body>
</html>
```

String Method

- ***strlen()*** – returns the length of string.
- ***str_word_count()*** – function counts the number of words in a string.
- ***strrev()*** – reverse the string.
- ***strpos()*** – function searches for a specific text within a string.
- ***str_replace()*** – function replaces some characters with some other characters in a string.
- ***strcmp()*** – The strcmp() function compares two strings.

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        echo strlen("Hello world!");
        echo str_word_count("Hello world!");
        echo strrev("Hello world!");
        echo strpos("Hello world!", "world");
        echo str_replace("world", "Dolly", "Hello world!");
        echo strcmp("Hello world!","Hello world!");
?>
        <p>If strcmp() function returns 0, the two strings are equal.</p>
</body>
</html>
```

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).
- To create a constant, use the ***define()*** function.

Syntax:

define(name, value, case-insensitive); Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
// case-sensitive constant name
        define("GREETING", "Welcome to PHP class!");
        echo GREETING;
// case-insensitive constant name
        define("WPT", "This is web technology class.", true);
        echo wpt;
?>
</body>
</html>
```

Control/Conditional statement

- Conditional statements are used to perform different actions based on different conditions.
- In PHP we have the following conditional statements:
  - ***if*** statement - executes some code if one condition is true

    Syntax:

    ```
    if(condition){
      //code to be executed if condition is true
    }
    ```
  - ***if...else*** statement - executes some code if a condition is true and another code if that condition is false.

    Syntax:

    ```
    if(condition){
      //code to be executed if condition is true
    }
    else{
      //code to be executed if condition is false
    }
    ```
  - ***if...elseif...else*** statement - executes different codes for more than two conditions.

    Syntax:

    ```
    if(condition){
    ```

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

- ***switch*** statement - selects one of many blocks of code to be executed.

Syntax:

```
switch(n){
  case label1:
        //code to be executed if label1=n;
        break;
  case label2:
        //code to be executed if label2=n;
        break;
 ... ... .....
default:
  //code to executed if n is different from all labels;
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        $t = date("H");
        echo "<p>The hour (of the server) is " . $t;
        echo ", and will give the following message:</p>";
        if ($t < "10") {
         echo "Have a good morning!";
        } elseif ($t < "20") {
        echo "Have a good day!";
        } else {
        echo "Have a good night!";
        }
?>
</body>
</html>
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        $favcolor = "red";
        switch ($favcolor) {
        case "red":
                echo "Your favorite color is red!";
                break;
```

```
        case "blue":
                echo "Your favorite color is blue!";
                break;
        case "green":
                echo "Your favorite color is green!";
                break;
        default:
                echo "Your favorite color is neither red, blue, nor green!";
}
?>
</body>
</html>
```

Loop Statements

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

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Flowchart:

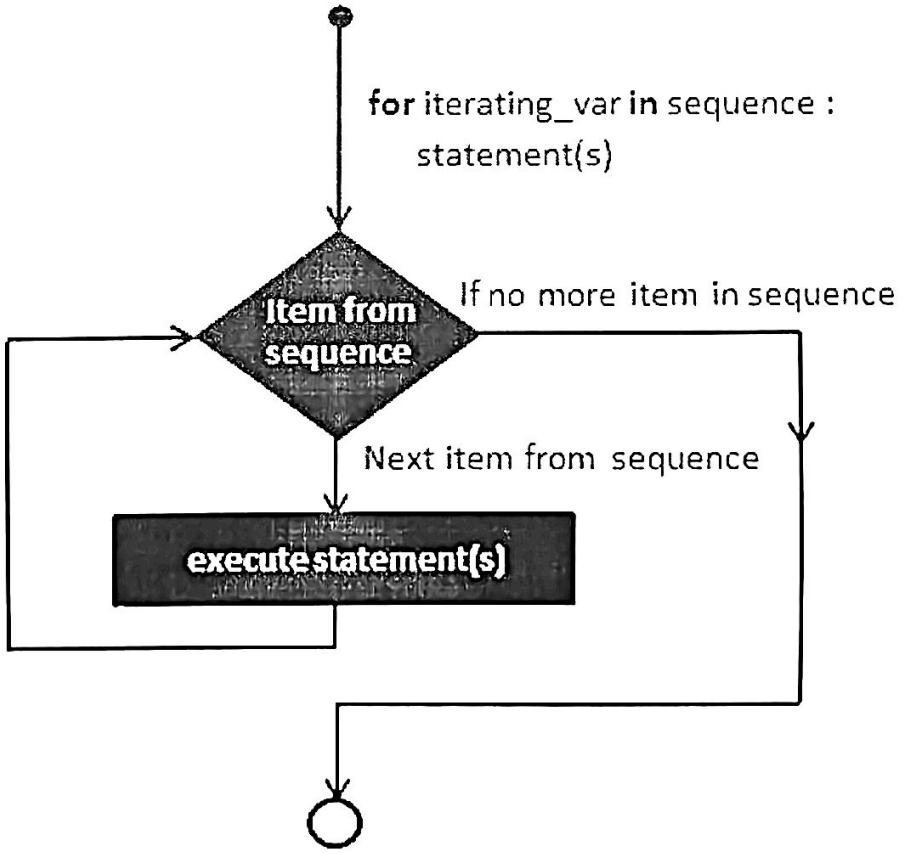


figure – flowchart of for loop

Syntax:

```
for(initialization; test-condition; iteration-statement){
    //statement(s) to be executed if expression is true
}
```

Example:

```
<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>
```

The While loop statement

Syntax:

while(condition is true){

// code to be executed

}

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
    $x = 1;
    while($x <= 5) {
        echo "The number is: $x <br>";
        $x++;
    }
?>
</body>
</html>
```

The PHP do...while Loop

Syntax:

do{

// code to be executed

} while(condition is true)

Example:

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

The PHP foreach Loop

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

Syntax:

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

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        $colors = array("red", "green", "blue", "yellow");
        foreach ($colors as $value) {
        echo "$value <br>";
        }
?>
</body>
</html>
```

PHP Function

The real power of PHP comes from its functions; it has more than 1000 built-in functions. But here you study about user defined function. 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.
- A user-defined function declaration starts with the word ***function***:

Syntax:

```
function function_Name(){
        //code to be executed
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
    function writeMsg() {
    echo "Hello world!";
    }
    writeMsg();
    //function having arguments/parameters
    function familyName($fname, $year) {
    echo "$fname was Born in $year <br>";
    }
    familyName("Ram", "1975");
    familyName("Shyam", "1978");
    //Default argument value
    function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
```

```
}
setHeight(350);
setHeight(); // will use the default value of 50
//Function – returning values
function sum($x, $y) {
$z = $x + $y;
return $z;
}
echo "The sum of 5 + 10 = " . sum(5, 10) . "<br>";
echo "The sum of 7 + 13 = " . sum(7, 13) . "<br>";
?>
</body>
</html>
```

Array in PHP

- An array is a special variable, which can hold more than one value at a time.
- An array can hold many values under a single name, and you can access the values by referring to an index number.
- In PHP, the ***array()*** function is used to create an array.
- In PHP, there are three types of arrays:
  - **Indexed arrays** - Arrays with a numeric index
  - **Associative arrays** - Arrays with named keys
  - **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

- There are two ways to create indexed arrays:
- The index can be assigned automatically (index always starts at 0), like this.

Example:

```
<?php
  $cars = array("Volvo", "BMW", "Toyota");
  echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] .
  "."; echo count($cars); //to count length of an array //using
  for loop for indexed array
  $vehicle= array("car", "bike", "bus");
  $arrlength = count($vehicle);
  for($x = 0; $x < $arrlength; $x++) {
    echo $vehicle[$x];
    echo "<br>";
  }
?>
```

PHP Associative Arrays

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

There are two ways to create an associative array:

```
$age=array("Ram"=>34, "Shyam"=>54,"Hari"=>50); OR
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
        echo "Peter is " . $age['Peter'] . " years old.";
        //using foreach loop
        foreach($age as $x => $x_value) {
        echo "Key=" . $x . ", Value=" . $x_value;
        echo "<br>";
}
?>
</body>
</html>
```

Multidimensional Arrays

An array containing one or more arrays and values are accessed using multiple indices. 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:

```
<?php
        $cars = array
        (
         array("Volvo",22,18),
        array("BMW",15,13),
        array("Saab",5,2),
        array("Land Rover",17,15)
        );
        for ($row = 0; $row < 4; $row++) {
        echo "<p><b>Row number $row</b></p>";
        echo "<ul>";
        for ($col = 0; $col < 3; $col++) {
        echo "<li>".$cars[$row][$col]."</li>";
        }
         echo "</ul>";
        }
 ?>
```

Array sorting

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending. The sort function , which takes an array as a parameter , sorts the values in the array, replacing the keys with the numeric keys 0,1,2.. .

PHP - Sort Functions For Arrays:

- ***sort()*** – sort arrays in ascending order
- ***rsort()*** – sort arrays in descending order
- ***asort()*** – sort associative arrays in ascending order, according to the value

- ***ksort()*** – sort associative arrays in ascending order, according to the key
- ***arsort()*** – sort associative arrays in descending order, according to the value
- ***krsort()*** – sort associative arrays in descending order, according to the key

Example: Sort Array in Ascending Order - sort():

```
<?php
        $cars = array("Volvo", "BMW", "Toyota");
        sort($cars);
        $clength = count($cars);
        for($x = 0; $x < $clength; $x++) {
        echo $cars[$x];
        echo "<br>";
        }
?>
</body>
</html>
```

Example: Sort Array (Ascending Order), According to Value - asort():

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

Example: Sort Array (Ascending Order), According to Value - ksort():

```
<!DOCTYPE html>
<html>
<body>
<?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>";
        }
?>
</body>
</html>
```

Example: Sort Array (Descending Order), According to Key - krsort():

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

```
<body>
<?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>";
        }
?>
</body>
</html>
```

each() Function

- is used to returns a two -element array consisting of the key and value of the current element.

Syntax:

each(array)

Example:

```
<?php
        $salaries=array("Shyam"=>4444,"Ram"=>55555,"Hari"=>6666);
        while($employee=each($salaries)){
                $name=$employee["key"];
                $salary=$employee["value"];
                print("The salary of $name is $salary <br/>");
}
?>
```

PHP array_slice() Function

The array_slice() function returns selected parts of an array.

Example:

```
<?php
        $a=array("red","green","blue","yellow","brown");
        print_r(array_slice($a,2,3));
?>
```

Output: Array ([0] => blue [1] => yellow [2] => brown)

Setting true and false:

```
<?php
// Preserve parameter set to true:
        $a=array("red","green","blue","yellow","brown");
        print_r(array_slice($a,2,2,true));
        echo"<br>";
// Preserve parameter set to false (default):
        $a=array("red","green","blue","yellow","brown");
        print_r(array_slice($a,1,2,false));
?>
```

Output:

Array ([2] => blue [3] => yellow)
Array ([0] => green [1] => blue)

With both string and integer keys:

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow","e"=>"brown");
print_r(array_slice($a,1,2));

echo"<br>";
$a=array("0"=>"red","1"=>"green","2"=>"blue","3"=>"yellow","4"=>"brown");
print_r(array_slice($a,1,2));
?>
```

Output:

Array ([b] => green [c] => blue)
Array ([0] => green [1] => blue)

PHP array_udiff() Function:

The *array_udiff()* function compares the values of two or more arrays, and returns the differences.

Note: This function uses a user-defined function to compare the values!

This function compares the values of two (or more) arrays, and return an array that contains the entries from *array1* that are not present in *array2* or *array3*, etc.

Example:

```
<?php
function myfunction($a,$b)
{
if ($b===$a)
 {
 return 0;
 }
 return ($b>$a)?-1:1;
}
$a1=array("a"=>"red","b"=>"green","c"=>"blue");
$a2=array("a"=>"blue","b"=>"black","e"=>"blue");
$result=array_udiff($a2,$a1,"myfunction");
print_r($result);
?>
```

Output:

Array ([b] => black)

PHP | print_r() Function

The print_r() function is a built-in function in PHP and is used to print or display information stored in a variable.

Syntax:

print_r($variable, $isStore)

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

1. **$variable**: This parameter specifies the variable to be printed and is a mandatory parameter.
2. **$isStore**: This an option parameter. This parameter is of boolean type whose default value is FALSE and is used to store the output of the **print_r()** function in a variable rather than printing it. If this parameter is set to TRUE then the **print_r()** function will return the output which it is supposed to print.

Form Handling

Forms could be handled by the same document that creates the form, but that may be confusing. It does not matter whether GET or POST method is used to transmit the form data. PHP builds a variable for each form element with the same name as the element.

Example: Form containing html source code ,file name *form.php*

```
<!DOCTYPE HTML>
<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>
```

The php file that response to above through action attribute is *welcome.php*. I.e. given below:

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

File Handling:

File handling is an important part of any web application. Here we will explain following functions related to files:

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP *fopen()* function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate. Files modes can be specified as one of the six options in this table.

Sr.No	Mode & Purpose
1	r Opens the file for reading only. Places the file pointer at the beginning of the file.
2	r+ Opens the file for reading and writing. Places the file pointer at the beginning of the file.

3	w	Opens the file for writing only. Places the file pointer at the beginning of the file and truncates the file to zero length. If files does not exist then it attempts to create a file.
4	w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
5	a	Opens the file for writing only. Places the file pointer at the end of the file.If files does not exist then it attempts to create a file.
6	a+	Opens the file for reading and writing only.Places the file pointer at the end of the file.If files does not exist then it attempts to create a file.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called ***fread()***. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files length can be found using the ***filesize()*** function which takes the file name as its argument and returns the size of the file expressed in bytes.

Example:

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

Example: Reading single line – *fgets()* function

```
//The fgetc() function is used to read single line from a file , named student.txt
<!DOCTYPE html>
<html>
<body>
<?php
        $myfile = fopen("student.txt", "r") or die("Unable to open file!");
        echo fgets($myfile);
        fclose($myfile);
?>

</body>
</html>
```

Example: Reading single character – *fgetc()* function

```
//The fgetc() function is used to read single character from a file , named student.txt
<!DOCTYPE html>
<html>
<body>
```

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

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.

Example:

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

Writing a File

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written.

Example:

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

Cookies in PHP

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies. There are three steps involved in identifying returning users:

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

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser).

Setting Cookies with PHP

PHP provided ***setcookie()*** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

- **Name** – This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** – This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Example:

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

```
    $cookie_name = "user";
    $cookie_value = "John Doe";
    setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
    // 86400 = 1 day
?>
<html>
<body>
<?php
    if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
    } else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
    }
?>
    <p><strong>Note:</strong> You might have to reload the page to see the
    value of the cookie.</p>
</body>
</html>
```

Delete cookies

To delete a cookie, use the ***setcookie()*** function with an expiration date in the past:

Example:

```
<!DOCTYPE html>
<?php
    // set the expiration date to one hour
    ago setcookie("user", "", time() - 3600);
?>
<html>
<body>
<?php
    echo "Cookie 'user' is deleted.";
?>
</body>
</html>
```

Check if Cookies are Enabled

Example:

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

```
        setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>
<?php
        if(count($_COOKIE) > 0) {
                echo "Cookies are enabled.";
        } else {
                echo "Cookies are disabled.";
        }
?>
</body>
</html>
```

Example: ***The isset() function is used to check if a cookie is set or not .***

```
<!DOCTYPE html>
<html>
<body>
<?php
        if(isset($_COOKIE["name"])) {
                echo "Welcome".$_COOKIE["name"]."<br>";
        } else {
                echo "Sorry ...Not recognized"."<br>";
        }
?>
</body>
</html>
```

Note: cookies can be modified by changing their name and values.

Session

- PHP Session is an alternative way to make data accessible across the various pages of an entire website.
- A session creates a file in a temporary directory on the server where registered session variables and their values are stored.
- This data will be available to all pages on the site during that visit.
- The location of the temporary file is determined by a setting in the ***php.ini*** file called ***session.save_path***.

  When a session is started following things happen :

  - PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
  - A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.

- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

- When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file.
- A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the ***session_start()*** function. This function first checks if a session is already started and if none is started then it starts one. Session variables are stored in associative array called ***$_SESSION[]***. These variables can be accessed during lifetime of a session. Make use of **isset()** function to check if session variable is already set or not.

Example: *The following example starts a session then register a variable called* ***counter*** *that is incremented each time the page is visited during the session.*

```
<?php
        session_start();
        if( isset( $_SESSION['counter'] ) ) {
                $_SESSION['counter'] += 1;
        }else {
                $_SESSION['counter'] = 1;
        }
                $msg = "You have visited this page ". $_SESSION['counter'];
                $msg .= "in this session.";
?>
<html>
 <head>
  <title>Setting up a PHP session</title>
 </head>
 <body>
        <?php echo ( $msg ); ?>
 </body>
</html>
```

Destroying a PHP Session

A PHP session can be destroyed by ***session_destroy()*** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session use variable then you can ***unset()*** function to unset a session variable.

Example: ***To unset a single variable***

```
<?php
        unset($_SESSION['counter']);
?>
```

Example: ***To destroy all session variables***

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

Error Handling

Error handling is the process of catching errors raised by your program and then taking appropriate action. If you would handle errors properly then it may lead to many unforeseen consequences.

Example: ***Using die() function***

```
<?php
    if(!file_exists("/tmp/test.txt")) {
        die("File not found");
    }else {
        $file = fopen("/tmp/test.txt","r");
        print "Opend file sucessfully";
    }
 // Test of the code here.
?>
```

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)
;

Example:

```
<?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);
} ?>
```

Note: A *trigger_error()* function is useful to trigger errors when an illegal input occurs.

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.

Exception Handling

Exceptions are important and provides a better control over error handling. Lets explain there new keyword related to exceptions.

- **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".
- **Throw** – This is how you trigger an exception. Each "throw" must have at least one "catch".
- **Catch** – A "catch" block retrieves an exception and creates an object containing the exception information.

Example:

```
<?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 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
        catch(Exception $e) {
                echo 'Message: ' .$e->getMessage();
        }
?>
```

In the above example $e->getMessage function is used to get error message.

Creating a Custom Exception Class

To create a custom exception handler you must 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.

Example:

```
<?php
class customException extends Exception {
        public function errorMessage() {
                //error message
                $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
                .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
                return $errorMsg;
        }
}
        $email = "someone@example.com";

        try {
        //check if
                if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
        {//throw exception if email is not valid
         throw new customException($email);
        }
 //check for "example" in mail address
        if(strpos($email, "example") !== FALSE) {
                throw new Exception("$email is an example e-mail");
        }
        }
        catch (customException $e) {
                echo $e->errorMessage();
        }

        catch(Exception $e) {
                echo $e->getMessage();
        }
?>
```

Example Explained:

The code above tests two conditions and throws an exception if any of the conditions are not met:

1. The ***customException()*** class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class.
2. The ***errorMessage()*** function is created. This function returns an error message if an e-mail address is invalid.
3. The ***$email*** variable is set to a string that is a valid e-mail address, but contains the string "example".
4. The ***"try"*** block is executed and an exception is not thrown on the first condition.
5. The second condition triggers an exception since the e-mail contains the string "example".
6. The ***"catch"*** block catches the exception and displays the correct error message.

MySQL Database

- MySQL is the most popular database system used with PHP.
- MySQL is a database system used on the web

- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

vii. MySQL connect

Before we can access data in the MySQL database, we need to be able to connect to the server:

Example:

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "";
    // Create connection
    $conn = new mysqli($servername, $username,
    $password); // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    echo "Connected successfully";
?>
```

viii. MySQL create Database

Example: The following example creates database named db_student.

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "password";
// Create connection
    $conn = mysqli_connect($servername, $username, $password);
// Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
// Create database
    $sql = "CREATE DATABASE db_student";
    if (mysqli_query($conn, $sql)) {
        echo "Database created successfully";
    } else {
        echo "Error creating database: " . mysqli_error($conn);
    }
mysqli_close($conn);
?>
```

ix. MySQL create Table

Example: The following example create table named tbl_student with five column : "*id*", "*firstName*", "*lastName*", "*email*" and "*address*".

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "";
    $dbname = "db_student";
// Create connection
    $conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
    if (!$conn) {
            die("Connection failed: " . mysqli_connect_error());
    }
// sql to create table
    $sql = "CREATE TABLE tbl_student(id INT(6) UNSIGNED AUTO_INCREMENT
    PRIMARY KEY, firstName VARCHAR(30) NOT NULL, lastName VARCHAR(30)
    NOT NULL, email VARCHAR(50),address VARCHAR(30) NOT NULL)";

if (mysqli_query($conn, $sql)) {
    echo "Table tbl_student created successfully";
} else {
  echo "Error creating table: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

x. MySQL Insert Data

Example: The following example add a new record in table *tbl_student*.

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "";
    $dbname = "db_student";
// Create connection
    $conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
    if (!$conn) {
            die("Connection failed: " . mysqli_connect_error());
    }
    $sql = "INSERT INTO tbl_student (firstname, lastname, email)VALUES
    ('John', 'Doe', 'john@example.com', 'kathmandu')";

    if (mysqli_query($conn, $sql)) {
            echo "New record created successfully";
    } else {
            echo "Error: " . $sql . "<br>" . mysqli_error($conn);
    }
```

```
mysqli_close($conn);
?>
```

xi. MySQL Select Data

Example: The following example selects the *id*, *firstname* and *lastname* columns from the *tbl_student* table and displays it on the page:

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "";
    $dbname = "db_student";
// Create connection
    $conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
    $sql = "SELECT id, firstname, lastname FROM
    tbl_student"; $result = mysqli_query($conn, $sql);
    if (mysqli_num_rows($result) > 0) {
  // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. "
        " .$row["lastname"]."<br>";
    }
    } else {
    echo "0 results";
    }
mysqli_close($conn);
?>
```

Note: The function ***num_rows()*** checks if there are more than zero rows returned. If there are more than zero rows returned, the function ***fetch_assoc()*** puts all the results into an associative array that we can loop through. The ***while()*** loop loops through the result set and outputs the data from the id, firstname and lastname columns.

[* is used to select all data from table.(SELECT * FROM tbl_student)]

xii. MySQL Update Data

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

Example: The following examples update the record with id=2 in the "tbl_student" table:

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "";
    $dbname = " db_student ";
// Create connection
    $conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
    if (!$conn) {
```

```
        die("Connection failed: " . mysqli_connect_error());
    }
        $sql = "UPDATE tbl_student SET lastname='Ram' WHERE id=2";
    if (mysqli_query($conn, $sql)) {
        echo "Record updated successfully";
    } else {
        echo "Error updating record: " . mysqli_error($conn);
    }
mysqli_close($conn);
?>
```

Note: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause , all record will be updated.

xiii. MySQL Delete Data

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

Example: The following examples delete the record with id=3 in the "tbl_student" table:

```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "";
    $dbname = "db_student";
// Create connection
    $conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
// sql to delete a record
    $sql = "DELETE FROM tbl_student WHERE id=3";
    if (mysqli_query($conn, $sql)) {
        echo "Record deleted successfully";
    } else {
        echo "Error deleting record: " . mysqli_error($conn);
    }
mysqli_close($conn);
?>
```

xiv. MySQL Insert Multiple Data

Multiple SQL statements must be executed with the *mysqli_multi_query* function.

Example: The following examples add three new records to the "tbl_student" table:

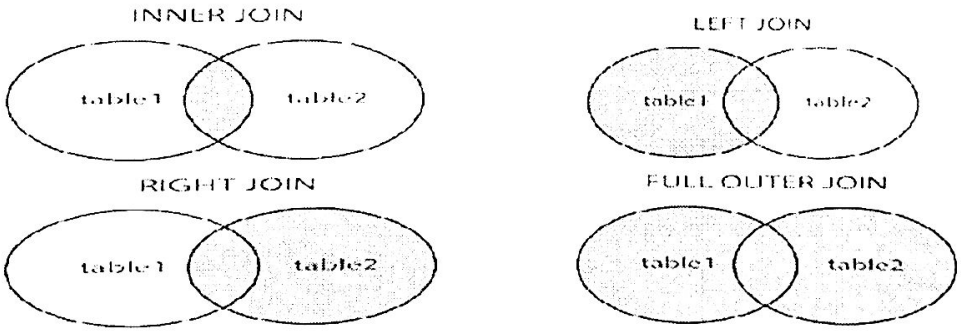
```
<?php
    $servername = "localhost";
    $username = "username";
    $password = "";
    $dbname = "db_student";
// Create connection
    $conn = mysqli_connect($servername, $username, $password,
$dbname); // Check connection
```

```
if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
}
$sql = "INSERT INTO tbl_student (firstname, lastname, email)
        VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO tbl_student (firstname, lastname, email)
        VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO tbl_student (firstname, lastname, email)
        VALUES ('Julie', 'Dooley', 'julie@example.com')";
if (mysqli_multi_query($conn, $sql)) {
        echo "New records created successfully";
} else {
        echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

SQL JOIN command

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



(INNER) JOIN

The INNER JOIN keyword selects records that have matching values in both tables.

Syntax:

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name = table2.column_name*;

SQL Statement:

SELECT Orders.OrderID, Customers.CustomerName FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

Syntax:

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name = table2.column_name*;

SQL Statement:

SELECT Orders.OrderID, Customers.CustomerName FROM Orders LEFT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

Syntax:

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name = table2.column_name*;

SQL Statement:

SELECT Orders.OrderID, Customers.CustomerName FROM Orders RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

FULL JOIN

The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.

Syntax:

SELECT *column_name(s)*
FROM *table1*
FULL JOIN *table2*
ON *table1.column_name = table2.column_name*;

SQL Statement:

SELECT Orders.OrderID, Customers.CustomerName FROM Orders FULL JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

Let , we have two tables , customers and orders as shown in figure below:

CustomerID	CustomerName	Address
1	Ram	Kailali
2	Shyam	Kanchanpur
3	Hari	Kathmandu
4	Sabin	Kailali
5	Ramesh	Kanchanpur
6	Mohan	Kathmandu

7	Tej	Morang

figure: Table 'Customers'

OrderID	CustomerID	OrderDate
1001	2	2075-2-2
1002	3	2075-2-3
1003	5	2075-2-4
1004	6	2075-2-2
1005	1	2075-2-4
1006	10	2075-2-8
1007	11	2075-2-9

figure: Table 'Orders'

GET and POST

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

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. Both GET and POST are treated as $_GET and $_POST. 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.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character. $_GET is an array of variables passed to the current script via the URL parameters. 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. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. GET may be used for sending non-sensitive data.

Note: *GET* should never be used for sending passwords or other sensitive information!

Example:

```
<?php
  if($_GET['name'] || $_GET['age']) {
    echo "Welcome ". $_GET['name']. "<br />";
    echo "You are ". $_GET['age']. " years old.";
    exit();
  }
?>
<html>
  <body>
```

```
    <form action = "<?php $_PHP_SELF ?>" method =
      "GET"> Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" value="submit" />
    </form>
  </body>
</html>
```

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING. $_POST is an array of variables passed to the current script via the HTTP POST method. 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. Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

Example:

```
<?php
  if( $_POST["name"] || $_POST["age"] ) {
    if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
      die ("invalid name and name should be alpha");
    }
    echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years old.";
    exit();
  }
?>
<html>
  <body>
    <form action = "<?php $_PHP_SELF ?>" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>
  </body>
</html>
```

The $_REQUEST variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Example:

```
<?php
  if( $_REQUEST["name"] || $_REQUEST["age"] ) {
    echo "Welcome ". $_REQUEST['name']. "<br />";
    echo "You are ". $_REQUEST['age']. " years old.";
    exit();
  }
```

```
?>
<html>
  <body>
    <form action = "<?php $_PHP_SELF ?>" method =
      "POST"> Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>
  </body>
</html>
```

Pattern Matching

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality. Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks. About Regular Expression , discussed in **chapter 4**, here we only discuss on some functions that are used for pattern matching.

These expression often used with SQL(Structured Query Language) to create, modify and manage data in databases and data-driven application.

Some examples:

^[0-9][a-z]$ = match any string that DOES begin with a number and has a lowercase letters.

^[^0-9][A-Z]$ = match any string that DOES NOT begin with a number and has a Capital letters.

^Z{1,3}$ = matches Z,ZZ and ZZZ

^Z{3,5}$ = matches ZZZ,ZZZZ and ZZZZZ

^\-?[0-9]{0,}\.?[0-9]{0,}$ = In this expression everything is between the ^ and $, so we are looking for an exact pattern match from beginning to end . The first part of the expression, "\-?" means look for a minus sign and uses the \ since it means a literal . The **?** means that the minus sign is optional. The second part of the expression is "**[0-9]{0,}**" and means look for numbers with **[0-9]** and find 0 or more of them with **{0,}**. Other part are also similar where "." is optional.

^[0-9]{1,}$ = match any string that begins with one or more numbers. To say this

+

^.+@.+\..+$ = match any string that begins with one or more non-new-line characters, has an "@" , is then followed by one or more other non-new-line characters, then has a literal "." and other non-new-line characters.

^(L|Th)ink$ = matches either **Link** OR **Think Some**

PHP's Regexp PERL Compatible Functions

preg_match() Function:

The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.

Example:

```
<?php
        $a="hello world";
        if(preg_match('/hi/',$a))
        {
                echo "Pattern found";
        }
        else{
                echo "Pattern not found";
        }
?>
```

Example:

```
<?php
        $email="santoshbist69@gmail.com";

        if(preg_match("/^.+@.+\..+/",$email)==true)
        {
                echo("that's true id");
        }
        else{
                echo("that's fake id");
        }
?>
```

preg_replace() function

The preg_replace() function operates just like POSIX function ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.

Example:

```
<?php
  $copy_date = "Copyright 1999";
  $copy_date = preg_replace("([0-9]+)", "2000",
  $copy_date); print $copy_date;
?>
```

Some Questions:

Q1. Write necessary php scripts to make connection with the database and insert/save form data into database . Form given below:

Enter your first name:

Enter second name:

Enter address:

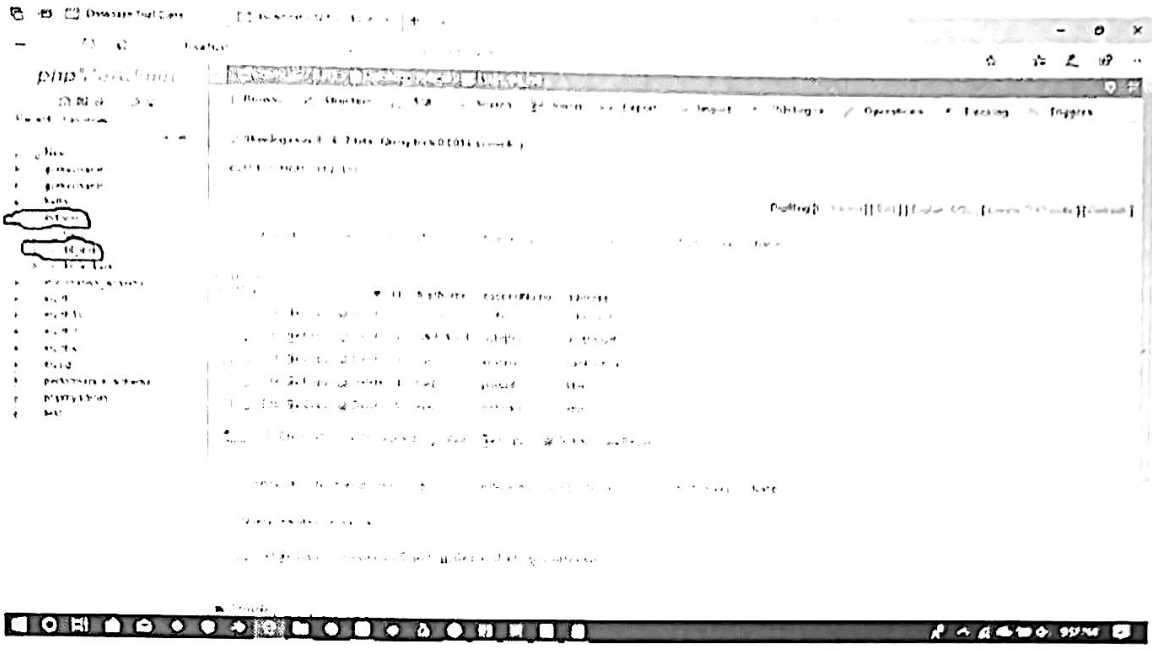
Save

Answer:

```
<!DOCTYPE html>
```

```
<html>
<head>
 <title>Database Trial Class</title>
</head>
<body>
 <form method="post">
  Enter your first name:<input type="text" name="fname"><br><br>
  Enter second name:<input type="text" name="lname"><br> <br>

  Enter address:<input type="text"
 name="address"><br><br> <input type="submit"
 name="imply" value="Save"> </form>
</body>
</html>
<?php
          if(isset($_POST['imply']))
          {
          $fname=$_POST["fname"];
          $lname=$_POST["lname"];
          $address=$_POST["address"];

          $host='localhost';
          $user='root';
          $pass='';
          $db='indreni';

          $con=mysqli_connect($host,$user,$pass,$db);

          if($con)
                    echo 'database connected successfully';
          echo'<br>';
                    $sql="insert into tbl_ind(firstName,secondName,address) values
                    ('".$fname."', '".$lname."','".$address."')";
          $query=mysqli_query($con,$sql);
          if($query)
                    echo 'Data is inserted';
          }
?>
```

Output: Data will be inserted when clicked on Save button. Here database base name is ***indreni*** and table name is ***tbl_ind.***



Q2. Write JavaScript code to find the factorial of a number requesting a number from user prompt.

Answer:

```
<!doctype html>
<html>
<head>
<script>
        function show(){
        var i, numb, fact;
        fact=1;
        numb=prompt("Enter the number","");
        for(i=1; i<=numb; i++)
        {
                fact= fact*i;
        }
        document.write("The factorial of "+numb+" is ",fact);
        }
        show();//function calling
</script>
</head>
<body>
<p>Calculating Factorial of number through prompt.</p>
</body>
</html>
```

OR if not using Prompt

```
<!doctype html>
<html>
<head>
<script>
        function show(){
        var i, numb, fact;
        fact=1;
        numb=Number(document.getElementById("num").value);
        for(i=1; i<=numb; i++)
        {
                fact= fact*i;
        }
```

```
        document.getElementById("answer").value= fact;
        }
</script>
</head>
<body>
Enter Num: <input id="num">
<button onclick="show()">Factorial</button><br><br>
<p>The factorial of a given number is <input id="answer">
</body>
</html>
```

Q3. Validation of number and character:

Answer:

```
<!DOCTYPE html>
<html>
<head>
        <title>validate number and characters</title>
</head>
<script>
//this function is used to validate character input
        function myFunction() {
//for character input
        var correct=/^[A-Za-z]+$/;
        var b=document.getElementById("user_name").value;
        if (b=="") {
                document.getElementById("msg").innerHTML="**Must
                fill"; return false;
        }
        if (b.length<3) {
                document.getElementById("msg").innerHTML="**Character must be
                greater  than 3";
                return false;
        }
        if (b.length>20) {
                document.getElementById("msg").innerHTML="**Character less than
                25";
                return false;
        }
        if (b.match(correct))
                true;
        else
        {
                document.getElementById("msg").innerHTML="**Only
                character allows";
        return false;
}
}
//This function is used to validate number input
        function myFun(){
        var x, text;
// Get the value of the input field with id="numb"
        x = document.getElementById("numb").value;
// If x is Not a Number or less than one or greater than 10
        if (isNaN(x) || x < 1 || x > 10) {
```

```
                text = "Input not valid";
        } else {
                text = "Input OK";
        }
                document.getElementById("demo").innerHTML = text;
        }
</script>
<body>
        <p>Please input a number between 1 and 10:</p>
        <input id="numb">
        <button type="button" onclick="myFun()">click me</button>
        <p id="demo"></p>
        <form onsubmit="return myFunction()">
                user name:<input type="text" id="user_name">
                <span id="msg"></span>
                <input type="submit" value="submit">
        </form>
</body>
</html>
```

Q4. Validate the Form below.

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Submit

Your Input:

Answer:

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

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

```
  } else {
   $name = test_input($_POST["name"]);
   // check if name only contains letters and whitespace
   if (!preg_match("/^[a-zA-Z ]*$/",$name)) { $nameErr
   = "Only letters and white space allowed";
   }
  }

  if (empty($_POST["email"])) {
   $emailErr = "Email is required";
  } else {
   $email = test_input($_POST["email"]); //
   check if e-mail address is well-formed
   if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $emailErr = "Invalid email format";
   }
  }

  if (empty($_POST["website"])) {
   $website = "";
  } else {
   $website = test_input($_POST["website"]);
   // check if URL address syntax is valid
   if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
    $websiteErr = "Invalid URL";
   }
  }

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

  if (empty($_POST["gender"])) {
   $genderErr = "Gender is required";
  } else {
   $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>

<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
```

```
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender"
  value="female">Female <input type="radio"
  name="gender" value="male">Male <input type="radio"
  name="gender" value="other">Other <span class="error">*
  <?php echo $genderErr;?></span> <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

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