CHAPTER - 2

WEB BASED SCRIPTING USING PHP

INTRODUCTION TO PHP:

PHP stands for Hypertext Preprocessor. PHP is an open source server side scripting language that is embedded in HTML. It is used to manage dynamic content, database, session tracking, and even entire e-commerce sites. It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

In order to develop and run PHP web pages three vital components need to be installed on our computer system:

- ☑ **Server:** PHP will work with virtually all web server software, including Microsoft's Internet Server (IIS) but then most often used is freely available Apache Server.
- ☑ **Database:** PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.
- ☑ **PHP Parser:** In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser.

INSTALLATION OF WEB SERVER:

1. Download the Apache HTTPD web server from the Apache web site (download the apache_2.2.16-win32-x86-no-ssIMSI installer)
2. Save the file to windows desktop.
3. Double click the MSI saved file on windows desktop and click "Next".
4. Click the radio button "I accept the terms in the license agreement" and click "Next".
5. On the next window, again click "Next".
6. Next, fill in all the text boxes with the following information:
   - Network Domain: "localhost"
   - Server Name: "localhost"
   - Administrator's Email Address: "your email address"
7. Make sure the radio button "for all users, on port 80 as a service recommended" is selected and click "Next".
8. After we have typed in the path, click "OK" and then click "Next"
9. On the next window highlight "Apache HTTP Server" and click the "change" button.
10. We are going to install all the packages and scripts in the path C:\Server\Apache2\ (assuming C: is main hard drive). So in the text box "Folder name:" type in "C:\Server\Apache2\". The ending backslash is important.
11. After we have typed in the path, click "OK" and then click "Next".
12. Click "Install" to begin the installation.
13. Once the Apache installation software has finished installing all the files on our computer, we will see a final window letting us know the installation was a success. Click the "Finish" button.

PHP Server Configuration:

Apache uses httpd.conf file for global settings, and the .htaccess file for per-directory access settings. Older versions of Apache split up httpd.conf into three files (access.conf, httpd.conf, and srm.conf), and some users still prefer this arrangement.

Apache server has a very powerful, but slightly complex, configuration system of its own. The following section describe settings in httpd.conf that affect PHP directly and cannot be set elsewhere. If we have standard installation then httpd.conf will be found at /etc/httpd/conf:

- **Timeout:**

This value sets the default number of seconds before any HTTP request will time out. If you set PHP's max_execution_time to longer than this value, PHP will keep grinding away but the user may see a 404 error. In safe mode, this value will be ignored; you must use the timeout value in php.ini instead

- **DocumentRoot:**

DocumentRoot designates the root directory for all HTTP processes on that server. It looks something like this on Unix: DocumentRoot ./usr/local/apache_1.3.6/htdocs. We can choose any directory as document root.

- **AddType:**

The PHP MIME type needs to be set here for PHP files to be parsed. Remember that we can associate any file extension with PHP like .php3, .php5 or .htm.

```
AddType application/x-httpd-php .php
AddType application/x-httpd-phps .phps
AddType application/x-httpd-php3 .php3 .phtml
AddType application/x-httpd-php .html
```

- **Action:**

We must uncomment this line for the Windows apxs module version of Apache with shared object support: LoadModule php4_module modules/php4apache.dll or on Unix flavors: LoadModule php4_module modules/mod_php.so

- **AddModule:**

We must uncomment this line for the static module version of Apache. AddModule mod_php4.c

PHP MyAdmin:

phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. phpMyAdmin supports a wide range of operations on MySQL and MariaDB. Frequently used operations (managing databases, tables, columns, relations, indexes, users, permissions, etc) can be performed via the user interface, while you still have the ability to directly execute any SQL statement.

Tobias Ratschiller, then an IT consultant and later founder of the software company Maguma, started to work on a PHP-based web front-end to MySQL in 1998, inspired by Peter Kuppelwieser's MySQL-Webadmin. He gave up the project (and phpAdsNew, of which he was also the original author) in 2000 because of lack of time.

By that time, phpMyAdmin had already become one of the most popular PHP applications and MySQL administration tools, with a large community of users and contributors. In order to coordinate the growing number of patches, a group of three developers, Olivier Müller, Marc Delisle and Loïc Chapeaux, registered the phpMyAdmin project at SourceForge.net and took over the development in 2001. Since 2015 the development is completely based on GitHub.

Features:

- Web interface
- MySQL and MariaDB database management
- Import data from CSV and SQL
- Export data to various formats: CSV, SQL, XML, PDF (via the TCPDF library), ISO/IEC 26300 - OpenDocument Text and Spreadsheet, Word, Excel, LaTeX and others
- Administering multiple servers
- Creating PDF graphics of the database layout
- Creating complex queries using query-by-example (QBE)
- Searching globally in a database or a subset of it
- Transforming stored data into any format using a set of predefined functions, like displaying BLOB-data as image or download-link
- Live charts to monitor MySQL server activity like connections, processes, CPU/memory usage, etc.
- Working with different operating systems.

Writing a Simple PHP Program:

```
<html>
        <head>
                <title>First Program</title>
        </head>
        <body>
                <?php
                        echo 'Hello. This is my first php program';
                ?>
        </body>
</html>
```

Common Uses of PHP:

- It is used for creating dynamic website.
- It is used for sending and receiving e-mails.
- We can use PHP to find today's date, and then build a calendar for the month.

- If we host banner advertisements on our website, we can use PHP to rotate them randomly.
- Using PHP we can count our visitors on our website.
- We can use PHP to create a special area of our website for members.
- Using PHP we can create login page for our user.
- Using PHP we can add, delete, and modify elements within our database through PHP.
- Using Access cookies variables and set cookies.
- Using PHP, we can restrict users to access some pages of our website.
- It can encrypt data.
- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- It can handle forms, i.e. gather data from files, save data to a file.

Commenting PHP Code:

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

1. **Single-Line Comments**: They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
  # This is a comment, and
  # This is the second line of the comment
  // This is a comment too. Each style comments only
  print "An example with single line comments";
?>
```

2. **Multi-Lines Comments**: They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

```
<?
  /* This is a comment with multiline
    Author: Mohammad Mohtashim
    Purpose: Multiline Comments Demo
    Subject: PHP
  */
  print "An example with multi line comments";
?>
```

Variables Type:

The main way to store information in the middle of a PHP program is by using a variable. Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.

- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables:

- **Integers** are whole numbers, without a decimal point, like 4195.
- **Doubles** are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** have only two possible values either true or false.
- **NULL** is a special type that only has one value: NULL.
- **Strings** are sequences of characters, like 'PHP supports string operations.'
- **Arrays** are named and indexed collections of other values.
- **Objects** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are **compound**. The compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

```
<?php
        $text1 = “Hello”;
        $text2 = “Kundan”;
        echo $text1.” ”.$text2;
?>
```

In PHP, a variable does not need to be declared before adding a value to it. In the example above, we see that we do not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In a strongly typed programming language, we have to declare the type and name of the variable before using it. In PHP, the variable is declared automatically when we use it.

Variable Scope:

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

1. Local Variables:

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function:

```
<?php
  $x = 4;
  function assignx () {
    $x = 0;
    print "\$x inside function is $x. <br />";
  }
  assignx();
  print "\$x outside of function is $x. <br />";
?>
```

This will produce the following result:

$x inside function is 0.
$x outside of function is 4.

2. Function Parameters:

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

```
<?php
  // multiply a value by 10 and return it to the caller
  function multiply ($value) {
    $value = $value * 10;
    return $value;
  }
  $retval = multiply (10);
  Print "Return value is $retval\n";
?>
```

This will produce the following result

Return value is 100

3. Global Variables:

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword **GLOBAL** in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name. Consider an example:

```
<?php
  $somevar = 15;
```

```
  function addit() {
    GLOBAL $somevar;
    $somevar++;
    print "Somevar is $somevar";
  }
  addit();
?>
```

This will produce the following result

Somevar is 16

4. Static Variables:

In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again. We can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?php
  function keep_track() {
    STATIC $count = 0;
    $count++;
    print $count;
    print "<br />";
  }
  keep_track();
  keep_track();
  keep_track();
?>
```

This will produce the following result

1
2
3

Constant Type:

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If we have defined a constant, it can never be changed or undefined.

To define a constant we have to use define() function and to retrieve the value of a constant, we have to simply specify its name. Unlike with variables, we do not need to have a constant with a $. We can also use the function constant() to read a constant's value if we wish to obtain the constant's name dynamically.

- **constant() function**

As indicated by the name, this function will return the value of the constant. This is useful when we want to retrieve value of a constant, but we do not know its name, i.e. it is stored in a variable or returned by a function.

```
<?php
  define("MINSIZE", 50);
  echo MINSIZE;
  echo constant("MINSIZE"); // same thing as the previous line
?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between Constants And Variables:

- There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

Valid constant names

```
define("ONE",     "first thing");
define("TWO2",    "second thing");
define("THREE_3", "third thing");
```

Invalid constant names

```
define("2TWO",    "second thing");
define("__THREE__", "third value");
```

Operator Types:

1. **Arithmetic Operators:**

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

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11

--	Decrement operator, decreases integer value by one	A-- will give 9

2. Comparison Operators:

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

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

3. Logical Operators:

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

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

4. Assignment Operators:

There are following assignment operators supported by PHP language.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C

+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
***=**	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

5. Conditional Operator:

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

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

Control Statements:

When we write code, we perform different actions for different decisions. Like other languages, PHP is built out of a series of control statements. The control statement can be an assignment, a function call, a loop, a conditional statement or even a statement that does nothing or an empty statement. In PHP we have the following conditional statements:

- **if statement**, We use this control statement to execute some code only if a specified condition is true.
- **if...else statement**, We use this control statement to execute some code if a condition is true and another code if the condition is false.
- **if...elseif....else statement,** We use this control statement to select one of several blocks of code to be executed
- **switch statement**, We use this control statement to select one of many blocks of code to be executed

1. The if Statement:

We use if statement to execute some code only if a specified condition is true. The expression is evaluated to its Boolean value. If expression evaluates to TRUE, PHP will execute statement, and if it evaluates to FALSE – it'll ignore it

Syntax

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

```
}
```

The following example would display" A is bigger than B" if $a is bigger than $b:

```
<?php
    if ($a > $b)
    echo "A is bigger than B";
?>
```

2. The if…else Statement:

Else…if, as its name suggests, is a combination of if and else. Like else, it extends if statement to execute a different statement in case the original if expression evaluates to FALSE. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to TRUE.

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

For example, the following code would display a is bigger than b, a equal to b or a is smaller than b:

```
<?php
    if ($a > $b) {
            echo "a is bigger than b";
    } elseif ($a == $b) {
            echo "a is equal to b";
    } else {
            echo "a is smaller than b";
    }
?>
```

3. The if…elseif….else Statement:

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

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

Note: Note that elseif and else if will only be considered exactly the same when using curly brackets as in the above example. When using a colon to define our if/elseif conditions, we must not separate else if into two words, or PHP will fail with a parse error.

4. **The Switch Statement:**

The switch statement is similar to IF statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the switch statement is for.

```
switch ( )
{
    case condition1
break;
    case condition2
break;
}
```

For example, the following code would display $i matched value as 0 or 1 or 2:

```
<?php
    switch ($i) {
    case 0:
        echo "i equals 0";
    case 1:
        echo "i equals 1";
    case 2:
        echo "i equals 2";
    }
?>
```

Loop Types:

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

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

We will discuss about **continue** and **break** keywords used to control the loops execution.

1. **The For Loop Statement:**

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

Syntax:

```
for (initialization; condition; increment){
  code to be executed;
}
```

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>
```

2. The While Loop Statement:

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax:

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

Example:

```
<html>
  <body>
    <?php
      $i = 0;
      $num = 50;

      while( $i < 10) {
        $num--;
        $i++;
      }
      echo ("Loop stopped at i = $i and num = $num" );
    ?>
  </body>
</html>
```

3. The Do...While Loop Statement:

The do...while statement will execute a block of code at least once it then will repeat the loop as long as a condition is true.

Syntax:

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

Example:

```
<html>
  <body>
    <?php
      $i = 0;
      $num = 0;

      do {
        $i++;
      }
      while( $i < 10 );
      echo ("Loop stopped at i = $i" );
    ?>
  </body>
</html>
```

4. The Foreach Loop Statement:

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax:

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

Example:

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

5. The Break Statement:

The PHP **break** keyword is used to terminate the execution of a loop prematurely. The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

Example:

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

6. The Continue Statement:

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop. Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.

Example:

```
<html>
  <body>

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

ARRAY:

An array is a data structure that stores one or more similar type of values in a single value. For example if we want to store 100 numbers then instead of defining 100 variables it's easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID which is called array index.

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

1. **Numeric Array:**

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

Example:

Here we have used **array()** function to create array. This function is explained in function reference.

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

2. **Associative Arrays:**

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

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE – Don't keep associative array inside double quote while printing otherwise it would not return any value.

Example:

```
<html>
  <body>
    <?php
      /* First method to associate create array. */
      $salaries = array("mohammad" => 2000, "qadir" => 1000, "zara" => 500);
      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
      echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
      echo "Salary of zara is ".  $salaries['zara']. "<br />";
      /* Second method to create array. */
      $salaries['mohammad'] = "high";
      $salaries['qadir'] = "medium";
      $salaries['zara'] = "low";
      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
      echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
      echo "Salary of zara is ".  $salaries['zara']. "<br />";
    ?>
  </body>
</html>
```

3. Multidimensional Arrays:

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

Example:

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

```
<html>
  <body>
    <?php
      $marks = array(
        "mohammad" => array (
          "physics" => 35,
          "maths" => 30,
          "chemistry" => 39
        ),
        "qadir" => array (
          "physics" => 30,
          "maths" => 32,
          "chemistry" => 29
        ),
```

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

FUNCTION:

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

In fact we hardly need to create our own PHP function because there are already more than 1000 of built-in library functions created for different area and we just need to call them according to our requirement.

- **Creating PHP Function:**

Its very easy to create our own PHP function. Suppose we want to create a PHP function which will simply write a simple message on our browser when we call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

Example:

```
<html>
  <head>
    <title>Writing PHP Function</title>
  </head>
  <body>
    <?php
      /* Defining a PHP Function */
      function writeMessage() {
        echo "You are really a nice person, Have a nice time!";
      }
      /* Calling a PHP Function */
      writeMessage();
```

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

- **PHP Functions with Parameters:**

PHP gives option to pass our parameters inside a function. We can pass as many as parameters we like. These parameters work like variables inside our function. Following example takes two integer parameters and add them together and then print them.

```
<html>
  <head>
    <title>Writing PHP Function with Parameters</title>
  </head>
    <?php
      function addFunction($num1, $num2) {
        $sum = $num1 + $num2;
        echo "Sum of the two numbers is : $sum";
      }
      addFunction(10, 20);
    ?>
  </body>
</html>
```

- **Passing Arguments by Reference:**

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value. Any changes made to an argument in these cases will change the value of the original variable. We can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

```
<html>
  <head>
    <title>Passing Argument by Reference</title>
  </head>
  <body>
    <?php
      function addFive($num) {
        $num += 5;
      }
      function addSix(&$num) {
        $num += 6;
      }
      $orignum = 10;
      addFive( $orignum );
```

```
    echo "Original Value is $orignum<br />";
    addSix( $orignum );
    echo "Original Value is $orignum<br />";
  ?>
 </body>
</html>
```

- **PHP Functions Returning Value:**

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.We can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>
 <head>
  <title>Writing PHP Function which returns value</title>
 </head>
 <body>
  <?php
   function addFunction($num1, $num2) {
    $sum = $num1 + $num2;
    return $sum;
   }
   $return_value = addFunction(10, 20);
   echo "Returned value from the function : $return_value";
  ?>
 </body>
</html>
```

- **Setting Default Values for Function Parameters:**

We can set a parameter to have a default value if the function's caller doesn't pass it. Following function prints NULL in case use does not pass any value to this function.

```
<html>
 <head>
  <title>Writing PHP Function which returns value</title>
 </head>
 <body>
  <?php
   function printMe($param = NULL) {
    print $param;
   }
   printMe("This is test");
   printMe();
```

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

- **Dynamic Function Calls:**

It is possible to assign function names as strings to variables and then treat these variables exactly as we would the function name itself. Following example depicts this behaviour.

```
<html>
  <head>
    <title>Dynamic Function Calls</title>
  </head>
  <body>
    <?php
      function sayHello() {
        echo "Hello<br />";
      }
      $function_holder = "sayHello";
      $function_holder();
    ?>
  </body>
</html>
```

URL Encoding:

URL Encoding is the process of converting string into valid URL format. Valid URL format means that the URL contains only what is termed "alpha | digit | safe | extra | escape" characters. We can read more about what and the whys of these terms on the World Wide Web Consortium site:

- http://www.w3.org/Addressing/URL/url-spec.html
- http://www.w3.org/International/francois.yergeau.html.

URL encoding is normally performed to convert data passed via html forms, because such data may contain special character, such as "/", ".", "#", and so on, which could either:

- Have special meanings
- is not a valid character for an URL
- could be altered during transfer

For instance, the "#" character needs to be encoded because it has a special meaning of that of an html anchor. The <space> character also needs to be encoded because is not allowed on a valid URL format. Also, some characters, such as "~" might not transport properly across the internet.

Example:

One of the most common encounters with URL Encoding is when dealing with <form>s. Form methods (GET and POST) perform URL Encoding implicitly. Websites uses GET and POST

methods to pass parameters between html pages. As an example, click the form below to see the string being URL encoded.

```
<form method = “GET” action = “example.html”>
    <input type = “text” name = “var” size = “50” value = “This is a simple & short test.”>
    <input type = “submit”>
</form>
```

This sample <form> sends the data in the text field using the GET method, which means that the data will be appended as query string. If we click the button and look at the resulting URL in the browser address bar, we should see something like this (the query string portion, which is automatically URL encoded by the browser, is shown in blue):

http://www.permadi.com/tutorial/urlEncoding/example.html?var=This+is+a+simple+%26+short+test.

Here, we can see that:

- The **<space>** character has been URL encoded as "+".
- The **&** character has been URL encoded as "%26".

<space> character and **&** character are just some of the special characters that need to be encoded.

What character need to encode?

1. **ASCII Control Characters:** They are unprintable characters typically used for output control. Character ranges 00 to 1F hex (0 to 31 decimal) and 7F (127 decimal).
2. **Non-ASCII Control Characters:** These are characters beyond the ASCII characters set of 128 characters. This range is part of the ISO-Latin character set and includes the entire “top half” of the ISO-Latin set 80 to FF hex (128 to 255 decimal).
3. **Reserved Characters:** These are special characters such as the dollar sign, ampersand, plus, colon, forward-slash, semi-colon, equals sign, question marks, and at symbol. All of these can have different meanings inside a URL so need to be encoded.
4. **Unsafe Characters:** These are space, quotation marks, less than symbol, greater than symbol, pound character, percent character, left curly brace, right curly brace, pipe, backslash, caret, tilde, left and right square bracket, and grave accent. These characters present the possiblility of being misunderstood within URLs for various reasons. These characters should also always be encoded.

HTML Encoding (Character Sets):

To display an HTML page correctly, a web browser must know which character set (character encoding) to use.

What is Character Encoding?

ASCII was the first **character encoding standard** (also called character set). ASCII defined 128 different alphanumeric characters that could be used on the internet: numbers (0-9), English letters (A-Z), and some special characters like ! $ + - ( ) @ < > .

ANSI (Windows-1252) was the original Windows character set, with support for 256 different character codes.

ISO-8859-1 was the default character set for HTML 4. This character set also supported 256 different character codes.

Because ANSI and ISO-8859-1 were so limited, HTML 4 also supported UTF-8. UTF-8 (Unicode) covers almost all of the characters and symbols in the world. The default character encoding for HTML5 is UTF-8.

The HTML charset Attribute

This is specified in the <meta> tag:

For HTML4:

<meta http-equiv="Content-Type" content="text/html;charset=ISO-8859-1">

For HTML5:

<meta charset="UTF-8">

If a browser detects ISO-8859-1 in a web page, it defaults to ANSI, because ANSI is identical to ISO-8859-1 except that ANSI has 32 extra characters.

The ASCII Character Set:

- ASCII uses the values from 0 to 31 (and 127) for control characters.
- ASCII uses the values from 32 to 126 for letters, digits, and symbols.
- ASCII does not use the values from 128 to 255

The ANSI Character Set (Windows-1252):

- ANSI is identical to ASCII for the values from 0 to 127.
- ANSI has a proprietary set of characters for the values from 128 to 159.
- ANSI is identical to UTF-8 for the values from 160 to 255.

The ISO-8859-1 Character Set:

- 8859-1 is identical to ASCII for the values from 0 to 127.
- 8859-1 does not use the values from 128 to 159.
- 8859-1 is identical to UTF-8 for the values from 160 to 255.

The UTF-8 Character Set:

- UTF-8 is identical to ASCII for the values from 0 to 127.
- UTF-8 does not use the values from 128 to 159.
- UTF-8 is identical to both ANSI and 8859-1 for the values from 160 to 255.
- UTF-8 continues from the value 256 with more than 10 000 different characters.

PHP and Database Connectivity:

Need for Database:

A database is a set of data that has a regular structure and that is organized in such a way that a computer can easily find the desired information. A database can generally be looked at as being a collection of *records*, each of which contains one or more *fields* (i.e., pieces of data) about some *entity* (i.e., object), such as a person, organization, city, product, work of art, recipe, chemical, or sequence of DNA. For example, the fields for a database that is about people who work for a specific company might include the name, employee identification number, address, telephone number, date employment started, position and salary for each worker.

Advantages:

- It will help to avoid data duplication and reduced data redundancy.
- Greater data integrity and independence from applications programs.
- We can control the security like we can setup the permission on different levels where only specified users can add, update or delete the data.
- The cost of data entry, storage, and retrieval is drastically reduced.
- Data can be validated based on the business rules.
- If the data in database is growing faster than modification of data easy.
- Facilitated development of new applications program.
- Logging and tracking of user actions can be easily done using database.
- Reduced updating errors and increased consistency
- Data format of data can be easily change.
- Data can be access easily using query language and multiple peoples can edit the data simultaneously.
- Due to backup and recovery options database can survive in crashes without corrupting the data.

PHP Supported Database:

1. **MySQL:**

No points for guessing this. MySQL remains most popular database for PHP applications. The database is the most popular open-source database in the world. The enterprise version of the database is available as well and is a paid one. Many popular applications like SugarCRM, Magento, WordPress and Drupal use mySQL. Big websites like Wikipedia, Facebook and Google are also its customers. MySQL workbench gives a web based user interface to manage the database.

2. **PostGreSQL:**

PostGreSQL was launched in 1995 and it laid stress on compliance and standards. As compared to mySQL it is feature rich and the database is currently under its release 9.3 as compare to release 5.6 of mySQL. PostGrey has advanced optimization and compression features which put it in parallel with Oracle.

Popular applications using PostGreSQL are OpenBravo and PostERP among others. However in comparison to mySQL PostGreSQL is believed to be a fraction slower.

3. SYBASE:

Founded in Berkley in 1984, SYBASE is now part of SAP AG. The solution is a popular choice for enterprise database management. It is also proven to perform under extreme load and is suitable where an organization needs a big database. It connects with PHP using sasql_connect() command similar to MySQL.

4. IBM-DB2:

This came into existence in 1983 though the foundation work for this started in 1970s.This was used extensively in conjunction with mainframe systems. The database is now being used in many large scale ERP and E-Commerce implementations. For connecting it with PHP you would need PECL extension.

5. Oracle Database:

One of the most prevalent databases of our times, Oracle database is equally popular for Windows and Linux based servers. It is difficult to come across organizations which do not use this database. For connecting with PHP applications, OCI8 extension is needed. Once the extension is in place oci_connect function can be used to connect to the database. Function oci_close() can be used to disconnect the connection.

6. Other Supported Databases:

Other databases which can be connected with PHP are listed below. Some of them are open source and others are proprietary. We can get more information about them by going their home page. This list has been arranged in alphabetic order.

- Cubrid
- DB++
- dBase
- filePro
- FireBird/InterBase
- FrontBase
- Informix
- Ingres
- MaxDB
- Mongo
- mSQL
- Ovrimos SQL
- Paradox
- SQLite
- SQLite3
- SQLSRV
- Tokyo Tyrant

Introduction to MySQL:

MySQL is the most popular Open Source Relational SQL database management system. MySQL is one of the best RDBMS being used for developing web-based software applications. MySQL is a fast, easy to use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons:

- MySQL is released under an open-source license. So we have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data languages.
- MySQL works on many operating systems and with many languages including PHP, PERL, Java, etc.
- MySQL works very quickly and even well with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default size limit for a table is 4 GB. But we can increase this if our operating system can handle it to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

Opening Database Connection:

PHP provides **mysql_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

Syntax: connection mysql_connect(server,user,passwd,new_link,client_flag);

S.N.	Parameter & Description
1	**server** Optional – The host name running database server. If not specified then default value is **localhost:3306**.
2	**user** Optional – The username accessing the database. If not specified then default is the name of the user that owns the server process.
3	**password** Optional – The password of the user accessing the database. If not specified then default is an empty password.
4	**new_link** Optional – If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned.
5	**client_flags** Optional – A combination of the following constants – • **MYSQL_CLIENT_SSL** – Use SSL encryption • **MYSQL_CLIENT_COMPRESS** – Use compression protocol • **MYSQL_CLIENT_IGNORE_SPACE** – Allow space after function names

	• **MYSQL_CLIENT_INTERACTIVE** – Allow interactive timeout seconds of inactivity before closing the connection

Closing Database Connection:

Its simplest function **mysql_close** PHP provides to close a database connection. This function takes connection resource returned by mysql_connect function. It returns TRUE on success or FALSE on failure.

Syntax: bool mysql_close (resource $link_identifier);

If a resource is not specified then last opend database is closed.

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'guest';
  $dbpass = 'guest123';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
  echo 'Connected successfully';
  mysql_close($conn);
?>
```

Creating a Database:

To create and delete a database we should have admin privilege. It's very easy to create a new MySQL database. PHP uses **mysql_query** function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

Syntax: bool mysql_query(sql, connection);

S.N.	Parameter & Description
1	**sql** Required - SQL query to create a database
2	**connection** Optional - if not specified then last opend connection by mysql_connect will be used.

Example:

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
```

```
  }
  echo 'Connected successfully';
  $sql = 'CREATE Database test_db';
  $retval = mysql_query( $sql, $conn );
  if(! $retval ) {
    die('Could not create database: ' . mysql_error());
  }
  echo "Database test_db created successfully\n";
  mysql_close($conn);
?>
```

Selecting a Database:

Once we establish a connection with a database server then it is required to select a particular database where our all the tables are associated. This is required because there may be multiple databases residing on a single server and we can do work with a single database at a time. PHP provides function **mysql_select_db** to select a database. It returns TRUE on success or FALSE on failure.

Syntax: bool mysql_select_db(db_name, connection);

S.N.	Parameter & Description
1	**db_name** Required - Database name to be selected
2	**connection** Optional - if not specified then last opened connection by mysql_connect will be used.

Example:

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'guest';
  $dbpass = 'guest123';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
  echo 'Connected successfully';
  mysql_select_db( 'test_db' );
  mysql_close($conn);
?>
```

Creating Database Tables:

To create tables in the new database we need to do the same thing as creating the database. First create the SQL query to create the tables then execute the query using mysql_query() function.

Example:

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
  echo 'Connected successfully';
  $sql = 'CREATE TABLE employee( '.
    'emp_id INT NOT NULL AUTO_INCREMENT, '.
    'emp_name VARCHAR(20) NOT NULL, '.
    'emp_address  VARCHAR(20) NOT NULL, '.
    'emp_salary   INT NOT NULL, '.
    'join_date    timestamp(14) NOT NULL, '.
    'primary key ( emp_id ))';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );
  if(! $retval ) {
    die('Could not create table: ' . mysql_error());
  }
  echo "Table employee created successfully\n";
  mysql_close($conn);
?>
```

In case we need to create many tables then it's better to create a text file first and put all the SQL commands in that text file and then load that file into $sql variable and excute those commands.

Deleting a Database:

If a database is no longer required then it can be deleted forever. We can use pass an SQL command to **mysql_query** to delete a database.

Example:

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
  $sql = 'DROP DATABASE test_db';
  $retval = mysql_query( $sql, $conn );
  if(! $retval ) {
```

```
    die('Could not delete database db_test: ' . mysql_error());
  }
  echo "Database deleted successfully\n";
  mysql_close($conn);
?>
```

WARNING: it's very dangerous to delete a database and any table. So before deleting any table or database we should make sure we are doing everything intentionally.

Deleting a Table:

It's again a matter of issuing one SQL command through **mysql_query** function to delete any database table. But be very careful while using this command because by doing so we can delete some important information we have in our table.

Example:

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
  $sql = 'DROP TABLE employee';
  $retval = mysql_query( $sql, $conn );
  if(! $retval ) {
    die('Could not delete table employee: ' . mysql_error());
  }
  echo "Table deleted successfully\n";
  mysql_close($conn);
?>
```

Insert Data into MySQL Database:

Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function **mysql_query**. Below a simple example to insert a record into **employee** table.

Example:

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
```

```
  }
  $sql = 'INSERT INTO employee '.
    '(emp_name,emp_address, emp_salary, join_date) '.
    'VALUES ( "guest", "XYZ", 2000, NOW() )';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );
  if(! $retval ) {
    die('Could not enter data: ' . mysql_error());
  }
  echo "Entered data successfully\n";
  mysql_close($conn);
?>
```

In real application, all the values will be taken using HTML form and then those values will be captured using PHP script and finally they will be inserted into MySQL tables. While doing data insert its best practice to use function **get_magic_quotes_gpc()** to check if current configuration for magic quote is set or not. If this function returns false then use function **addslashes()** to add slashes before quotes.

Example:

Try out this example by putting this code into add_employee.php, this will take input using HTML Form and then it will create records into database.

```
<html>

  <head>
    <title>Add New Record in MySQL Database</title>
  </head>
  <body>
    <?php
      if(isset($_POST['add'])) {
        $dbhost = 'localhost:3036';
        $dbuser = 'root';
        $dbpass = 'rootpassword';
        $conn = mysql_connect($dbhost, $dbuser, $dbpass);
        if(! $conn ) {
          die('Could not connect: ' . mysql_error());
        }
        if(! get_magic_quotes_gpc() ) {
          $emp_name = addslashes ($_POST['emp_name']);
          $emp_address = addslashes ($_POST['emp_address']);
        }else {
          $emp_name = $_POST['emp_name'];
          $emp_address = $_POST['emp_address'];
        }
        $emp_salary = $_POST['emp_salary'];
        $sql = "INSERT INTO employee ". "(emp_name,emp_address, emp_salary,
```

```
    join_date) ". "VALUES('$emp_name','$emp_address',$emp_salary, NOW())";
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );
   if(! $retval ) {
     die('Could not enter data: ' . mysql_error());
   }
   echo "Entered data successfully\n";
   mysql_close($conn);
 }else {
   ?>
     <form method = "post" action = "<?php $_PHP_SELF ?>">
       <table width = "400" border = "0" cellspacing = "1"
         cellpadding = "2">

         <tr>
           <td width = "100">Employee Name</td>
           <td><input name = "emp_name" type = "text"
             id = "emp_name"></td>
         </tr>

         <tr>
           <td width = "100">Employee Address</td>
           <td><input name = "emp_address" type = "text"
             id = "emp_address"></td>
         </tr>

         <tr>
           <td width = "100">Employee Salary</td>
           <td><input name = "emp_salary" type = "text"
             id = "emp_salary"></td>
         </tr>

         <tr>
           <td width = "100"> </td>
           <td> </td>
         </tr>

         <tr>
           <td width = "100"> </td>
           <td>
             <input name = "add" type = "submit" id = "add"
               value = "Add Employee">
           </td>
         </tr>

       </table>
```

```
        </form>
      <?php
    }
  ?>
 </body>
</html>
```

Getting Data from MySQL Database:

Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function mysql_query. We have several options to fetch data from MySQL. The most frequently used option is to use function **mysql_fetch_array()**. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows. Below is a simple example to fetch records from **employee** table.

Example

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not get data: ' . mysql_error());
  }
  while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
    echo "EMP ID :{$row['emp_id']}  <br> ".
      "EMP NAME : {$row['emp_name']} <br> ".
      "EMP SALARY : {$row['emp_salary']} <br> ".
      "--------------------------------<br>";
  }
  echo "Fetched data successfully\n";
  mysql_close($conn);
?>
```

The content of the rows are assigned to the variable $row and the values in row are then printed.

NOTE: Always remember to put curly brackets when we want to insert an array value directly into a string.

In above example the constant **MYSQL_ASSOC** is used as the second argument to mysql_fetch_array(), so that it returns the row as an associative array. With an associative array we can access the field by using their name instead of using the index.

PHP provides another function called **mysql_fetch_assoc()**which also returns the row as an associative array.

Example:

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not get data: ' . mysql_error());
  }
  while($row = mysql_fetch_assoc($retval)) {
    echo "EMP ID :{$row['emp_id']}  <br> ".
      "EMP NAME : {$row['emp_name']} <br> ".
      "EMP SALARY : {$row['emp_salary']} <br> ".
      "--------------------------------<br>";
  }
  echo "Fetched data successfully\n";
  mysql_close($conn);
?>
```

We can also use the constant **MYSQL_NUM**, as the second argument to mysql_fetch_array(). This will cause the function to return an array with numeric index.

Example

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);
  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
```

```
  $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not get data: ' . mysql_error());
  }
  while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
    echo "EMP ID :{$row[0]}  <br> ".
      "EMP NAME : {$row[1]} <br> ".
      "EMP SALARY : {$row[2]} <br> ".
      "--------------------------------<br>";
  }
  echo "Fetched data successfully\n";
  mysql_close($conn);
?>
```

Updating Data into MySQL Database:

Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function **mysql_query**. Below is a simple example to update records into **employee** table. To update a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example:

```
<html>
  <head>
    <title>Update a Record in MySQL Database</title>
  </head>
  <body>
    <?php
      if(isset($_POST['update'])) {
        $dbhost = 'localhost:3036';
        $dbuser = 'root';
        $dbpass = 'rootpassword';
        $conn = mysql_connect($dbhost, $dbuser, $dbpass);

        if(! $conn ) {
          die('Could not connect: ' . mysql_error());
        }
        $emp_id = $_POST['emp_id'];
        $emp_salary = $_POST['emp_salary'];

        $sql = "UPDATE employee ". "SET emp_salary = $emp_salary ".
          "WHERE emp_id = $emp_id" ;
        mysql_select_db('test_db');
```

```
      $retval = mysql_query( $sql, $conn );

      if(! $retval ) {
        die('Could not update data: ' . mysql_error());
      }
      echo "Updated data successfully\n";
      mysql_close($conn);
    }else {
      ?>
        <form method = "post" action = "<?php $_PHP_SELF ?>">
          <table width = "400" border =" 0" cellspacing = "1"
            cellpadding = "2">

            <tr>
              <td width = "100">Employee ID</td>
              <td><input name = "emp_id" type = "text"
                id = "emp_id"></td>
            </tr>

            <tr>
              <td width = "100">Employee Salary</td>
              <td><input name = "emp_salary" type = "text"
                id = "emp_salary"></td>
            </tr>

            <tr>
              <td width = "100"> </td>
              <td> </td>
            </tr>

            <tr>
              <td width = "100"> </td>
              <td>
                <input name = "update" type = "submit"
                  id = "update" value = "Update">
              </td>
            </tr>

          </table>
        </form>
      <?php
    }
  ?>
 </body>
</html>
```

Deleting Data from MySQL Database

Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function **mysql_query**. Below is a simple example to delete records into **employee** table. To delete a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example:

```
<html>
  <head>
    <title>Delete a Record from MySQL Database</title>
  </head>
  <body>
    <?php
      if(isset($_POST['delete'])) {
        $dbhost = 'localhost:3036';
        $dbuser = 'root';
        $dbpass = 'rootpassword';
        $conn = mysql_connect($dbhost, $dbuser, $dbpass);

        if(! $conn ) {
          die('Could not connect: ' . mysql_error());
        }
        $emp_id = $_POST['emp_id'];

        $sql = "DELETE FROM employee WHERE emp_id = $emp_id" ;
        mysql_select_db('test_db');
        $retval = mysql_query( $sql, $conn );

        if(! $retval ) {
          die('Could not delete data: ' . mysql_error());
        }
        echo "Deleted data successfully\n";
        mysql_close($conn);
      }else {
        ?>
          <form method = "post" action = "<?php $_PHP_SELF ?>">
            <table width = "400" border = "0" cellspacing = "1"
              cellpadding = "2">

              <tr>
                <td width = "100">Employee ID</td>
                <td><input name = "emp_id" type = "text"
                  id = "emp_id"></td>
              </tr>

              <tr>
```

```
            <td width = "100"> </td>
            <td> </td>
          </tr>

          <tr>
            <td width = "100"> </td>
            <td>
              <input name = "delete" type = "submit"
                id = "delete" value = "Delete">
            </td>
          </tr>
        </table>
      </form>
    <?php
  }
?>

  </body>
</html>
```

File Handling in PHP:

File handling is an important part of any web application. We often need to open and process a file for different tasks. PHP has several functions for creating, reading, uploading, and editing files.

Create File:

The fopen() function is also used to create a file. Maybe a little confusing but in PHP, a file is created using the same function used to open file. Example: $myfile = fopen("testfile.txt","w")

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens 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.
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.
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.
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.

Read 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. So, here are the steps required to read a file with PHP:

- Open a file using fopen() function.
- Get the file's length using filesize() function.
- Read the file's content using fread() function.
- Close the file with fclose() function.

Example:

```
<html>
    <head>
        <title>Read file</title>
    </head>
    <body>
        <?php
            $filename = "testfile.txt";
            $file = fopen($filename,"r");
            If($file==false){
                echo "error in opening file";
                exit();
            }
            $filesize = filesize($filename);
            $filetext = fread($file,$filesize);
            fclose($file);
            echo "File Size= ".$filesize."bytes";
            echo "<pre>".$filetext."</pre>";
        ?>
    </body>
</html>
```

Write 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. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included writing would stop after the specified length has been reached. The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using file_exist() function which takes file name as an argument.

Example:

```
<html>
```

```
<head>
        <title>Write file</title>
</head>
<body>
        <?php
                $filename = “testfile.txt”;
                $file = fopen($filename,”r”);
                If($file==false){
                        echo “error in opening file”;
                        exit();
                }
                fwrite($file,”This is a sample text to write\n”);
        ?>
</body>
</html>
```

Close File:

The fclose() function is used to close an open file. It’s a good programming practice to close all files after we have finished with them. We don’t want an open file running around on our server taking up resources.

Example:

```
<?php
    //some code to be executed…
    fclose($myfile);
?>
```

PHP Read Single Line:

The fgets() function is used to read a single line from a file. The example below outputs the first line of the “testfile.txt” file.

Example:

```
<?php
    $myfile = fopen(“testfile.txt”,”r”) or die(“unable to open file”);
    echo fgets($myfile);
    fclose($myfile);
?>
```

PHP Read Single Character:

The fgetc() function is used to read a single character from a file. The example below reads the “testfile” file character by character, until end-of-file is reached.

Example:

```
<?php
    $myfile = fopen("testfile.txt","r") or die("unable to open file");
    while(!feof($myfile)){
        echo fgetc($myfile);
    }
    fclose($myfile);
?>
```

PHP File System:

The file system function allow us to access and manipulate the file system. The file system functions are part of the PHP core. There is no installation needed to use these functions.

Function	Description
basename()	Returns the filename component of a path
chgrp()	Changes the file group
chmod()	Changes the file mode
chown()	Changes the file owner
clearstatcache()	Clears the file status cache
copy()	Copies a file
fclose()	Closes an openfile
dirname()	Returns the directory name component of a path
disk_free_space()	Returns the free space of a directory

Example:

```
<?php
    echo copy("source.txt","target.txt");
?>
<?php
    echo disk_free_space("C:");
?>
```

PHP Directory Functions:

The directory functions allows us to retrieve information about directories and their contents. The PHP directory functions are part of the PHP core. No installation is required to use these functions.

Function	Description
chdir()	Change current directory
chroot()	Change the root directory
dir()	Opens a directory handle and returns an object.
closedir()	Closes a directory
getcwd()	Gets the current working directory
opendir()	Open directory handle
readdir()	Read entry from directory handle

rewinddir()	Rewind directory handle
scandir()	List files and directories inside the specified path

Example:

```
<?php
    $d = dir(getcwd());
    echo "Handle: ".$d -> handle."<br>";
    echo "Path: "$d -> path."<br>";
    while(($file = $d->read())!==false){
            echo "filename: ".$file."<br>";
    }
    $d->close();
?>
```

More Features of PHP:

PHP Cookies:

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

There are three steps involved in identifying returning users:

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

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.

Syntax: setcookie(name, value, expire, path, domain, security);

Here is the detail of all the arguments:

- **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.

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

```
<?php
  setcookie("name", "John Watkin", time()+3600, "/","", 0);
  setcookie("age", "36", time()+3600, "/", "",  0);
?>
<html>
  <head>
    <title>Setting Cookies with PHP</title>
  </head>
  <body>
    <?php echo "Set Cookies"?>
  </body>
</html>
```

Accessing Cookies with PHP:

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```
<html>
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
      echo $_COOKIE["name"]. "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"]. "<br />";
      echo $_COOKIE["age"] . "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["age"] . "<br />";
    ?>
  </body>
</html>
```

We can use isset() function to check if a cookie is set or not.

```
<html>
```

```
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
      if( isset($_COOKIE["name"]))
        echo "Welcome " . $_COOKIE["name"] . "<br />";
      else
        echo "Sorry... Not recognized" . "<br />";
    ?>
  </body>
</html>
```

Deleting Cookie with PHP:

Officially, to delete a cookie we should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on. It is safest to set the cookie with a date that has already expired:

```
<?php
  setcookie( "name", "", time()- 60, "/","", 0);
  setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>

  <head>
    <title>Deleting Cookies with PHP</title>
  </head>
  <body>
    <?php echo "Deleted Cookies" ?>
  </body>
</html>
```

PHP Session:

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session. 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**. Before using any session variable make sure you have setup this 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 bearing that name and a validation can be done by comparing both values.

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. It is recommended to put the call to **session_start()** at the beginning of the page.

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

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

Make use of **isset()** function to check if session variable is already set or not.

```
<?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 variable then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable:

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

Here is the call which will destroy all the session variables:

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

Sending Emails using PHP:

PHP must be configured correctly in the **php.ini** file with the details of how our system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines our email server address. The second is called sendmail_from which defines our own email address.

The configuration for Windows should look something like this:

```
[mail function]
; For Win32 only.
SMTP = smtp.secureserver.net

; For win32 only
sendmail_from = webmaster@tutorialspoint.com
```

Linux users simply need to let PHP know the location of their **sendmail** application. The path and any desired switches should be specified to the sendmail_path directive.

The configuration for Linux should look something like this:

```
[mail function]
; For Win32 only.
SMTP =
; For win32 only
sendmail_from =
; For Unix only
sendmail_path = /usr/sbin/sendmail -t -i
```

Sending Plain Text Email

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

Syntax: mail(to, subject, message, headers, parameters);

Here is the description for each parameters.

S.N.	Parameter & Description
1	**to** Required. Specifies the receiver / receivers of the email
2	**subject** Required. Specifies the subject of the email. This parameter cannot contain any newline characters
3	**message** Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters
4	**headers** Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n)
5	**parameters** Optional. Specifies an additional parameter to the send mail program

As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed. Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

Sending HTML Email:

When we send a text message using PHP then all the content will be treated as simple text. Even if we will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.

While sending an email message we can specify a MIME version, content type and character set to send an HTML email.

Example:

Following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. We can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<html>
  <head>
    <title>Sending HTML email using PHP</title>
  </head>
  <body>
```

```
    <?php
      $to = "xyz@somedomain.com";
      $subject = "This is subject";

      $message = "<b>This is HTML message.</b>";
      $message .= "<h1>This is headline.</h1>";

      $header = "From:abc@somedomain.com \r\n";
      $header .= "Cc:afgh@somedomain.com \r\n";
      $header .= "MIME-Version: 1.0\r\n";
      $header .= "Content-type: text/html\r\n";

      $retval = mail ($to,$subject,$message,$header);
      if( $retval == true ) {
        echo "Message sent successfully...";
      }else {
        echo "Message could not be sent...";
      }
    ?>
  </body>
</html>
```

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.

Example: http://www.test.com/index.htm?name1=value1&name2=value2

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

```
<?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" />
    </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.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides $_POST associative array to access all the sent information using POST method.

Try out following example by putting the source code in test.php script.

```
<?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. We will discuss $_COOKIE variable when we will explain about cookies.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in test.php script.

```
Here $_P<?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>
```

HP_SELF variable contains the name of self-script in which it is being called.