DIPLOMA IN COMPUTER ENGINEERING

A LABORATORY MANUAL

WEB PROGRAMMING (CM505E)

Prepared By: Ms. S. N. Chaudhari

DEPARTMENT OF COMPUTER ENGINEERING

GOVERNMENT POLYTECHNIC, NAGPUR

(An Autonomous Institute of Govt. of Maharashtra)

DOs and DON'Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.

2. All the students should sit according to their roll numbers starting from their left to right.

3. All the students are supposed to enter the terminal number in the log book.

4. Do not change the terminal on which you are working.

5. All the students are expected to get at least the algorithm of the program/concept to be implemented.

6. Strictly observe the instructions given by the teacher/Lab Instructor.

Instruction for Laboratory Teachers::

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.

2. Students should be taught for taking the printouts under the observation of lab teacher.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

Vision of Institute:

To become a Center of Excellence, providing quality technical education and training.

Mission of Institute:

The mission of Government Polytechnic, Nagpur is-

1. To frame institutional policies for effective implementation of teaching learning process.
2. To inculcate values and ethics for life-long learning through curricular, co-curricular and extra-curricular activities.
3. To develop facilities and services for academic excellence.
4. To enhance the skills of faculties and staff through industry institute collaboration.

Vision of Department:

To provide academic excellence in computer engineering by imparting in-depth knowledge that meets the aspirations of the global community and to serve as a valuable resource for industry

Mission of Department:

1. To develop human potential to its fullest extent so that intellectually capable and optimistic leaders can emerge in a range of profession.
2. To provide quality engineering education to the students through state of art education in Computer Engineering.
3. To produce globally competent diploma holders having creative skills and ethical values keeping pace with ever-changing technological advancements in Computer Engineering.
4. To establish Industry Institute Interaction to make students ready for the industrial environment.

Program Educational Objective's:

1. To endeavour innovations with a substantial fundamental technical skills.
2. To develop an ability to analyze the requirements of the software, understand the technical specifications design and provide novel engineering solutions and efficient product designs.
3. To design system components and processes that meets the quality criteria with appropriate consideration for the cultural, societal, and environmental considerations.
4. To apply appropriate techniques, resources, and computer engineering tools to various engineering activities with an understanding of the limitations.
5. To demonstrate knowledge and understanding of the computer engineering and management principles to manage projects in multidisciplinary environments and to promote student awareness on life-long learning.

Program Outcome's:

1. **Basic Knowledge:** Ability to apply knowledge of basic Mathematics, Science and Engineering to solve the engineering problems
2. **Discipline Knowledge:** Ability to discipline specific knowledge to solve core and/or applied engineering problems
3. **Experiments and Practice:** Ability to plan and perform experiments and practices and use the results to solve engineering problems
4. **Engineering tools:** Apply appropriate technologies and tools with an understandings of limitations
5. **The Engineer and Society:** Demonstrate knowledge to assess the societal, health, safety, legal and cultural issues, and the consequent responsibilities relevant to engineering practice.
6. **Environment and Sustainability:** Understand the impact of engineering solutions in societal and environmental context, and demonstrate knowledge and need for sustainable development.
7. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
8. **Individual and Team work:** Function effectively as an individual, and as a member or leader in diverse/ multidisciplinary teams.
9. **Communication:** An ability to communicate effectively.
10. **Life-long learning:** Recognize the need for, and have preparation and the ability to engage in independent and life-long learning in context of technological changes.

Program Specific Objectives:

1. To design and code engineering problems using programming skills.
2. To apply, design and develop principles of software engineering and data base concepts for Computer- based systems in solving engineering Problems.

PROGRAMME: DIPLOMA IN COMPUTERENGINEERING

LEVEL NAME: ELECTIVE COURSES

COURSE CODE: CM505E$

COURSE TITLE: WEB PROGRAMMING

PREREQUISITE: CM413E

TEACHING SCHEME: TH: 03; TU: 00; PR: 02(CLOCK HRs.)

TOTAL CREDITS: 04(1 TH/TU CREDIT = 1 CLOCK HR., 2 PR CREDIT = 2 CLOCK HR.)

TH. TEE EXAM: 03 HRs

PR. TEE EXAM: 02 HRs (External)

PT. EXAM: 01 HR

Course Outcomes:

After completing this course students will be able to–

1. Identify Apache Web Server, MySQL server environment to create, debug and run PHP program.
2. Identify In-Built and User defined function in PHP programming.
3. Illustrate form controls for presenting web based content.
4. Create and execute PHP scripts by applying concept of error handling techniques of PHP.
5. Create and execute a PHP script for various file commands.
6. Create dynamic website/web based applications using PHP, MySQL database.

Rubrics:

Criteria	**Unsatisfactory**	**Fair**	**Good**	**Excellent**
Writing program, Logic of the program (10)	Program/Script code is with poor logic. (0-10)	Program/Script code is with moderate logic. (10-12)	Program/Script code is clean, understandable, complex logic. (13-16)	Program/Script code is clean, understandable, and well-organized. (16-20)
Debug the program (20)	Major problems with at three or four of the readability subcategories and program cannot be interpreted.(0-2)	At least one major issue with indentation, whitespace, variable names, or organization. (3-4)	Minor issues with consistent indentation, use of whitespace, variable naming, or general organization.(5-6)	No errors. (7-10)
Execution of program, Program Output, Complexity of program (10)	Program/Script is not completed within specified time and output of program is not correct.(0-2)	Program/Script is completed within specified time and program runs successfully but output is not correct.(3-4)	Program/Script is slight complex but completed within specified time and program runs successfully but output is correct partially.(5-6)	Program/Script simple and completed within specified time and program runs successfully and output is correct without any errors.(7-10)
Viva voce (10)	No questions from external/internal are answered correctly.(0-2)	25% questions from external/internal are answered correctly.(3-4)	50% questions from external/internal are answered correctly.(5-6)	All questions from external/internal are answered correctly.(7-10)

List of Practical's:

Practical	**Specific Learning Outcomes (Psychomotor Domain)**	**Units**	**Hrs**	**CO's**	**Page No**
1	Create and execute PHP script to demonstrate arithmetic operators, comparison operator, and logical operator	Introduction to PHP	2	CO1	8
2	Create and execute PHP script to demonstrate Math functions	Programming Principles in PHP	2	CO2	10
3	Create and execute PHP script to demonstrate Array functions		2	CO1, CO2	13
4	Create and execute PHP script for Error-handling using exceptions		2	CO4	17
5	Create student registration form using text box, check box, radio button, select, submit button and display user inserted value in new PHP page	Working with data and forms	2	CO3	19
6	Create Registration Form using text box, check box, radio button, select, submit button and display user inserted value in new PHP page		2	CO3	22
7	Create a form and apply the Data Validation		2	CO3	22
8	Create and execute two different PHP script to demonstrate passing variables through a URL.	Sessions and Cookies	2	CO6 CO4	24
9	Create and execute two different PHP script to demonstrate passing variables with sessions		2	CO4 CO6	25
10	Create and execute PHP script to demonstrate passing variables with cookies		2	CO4 CO6	26
11	Create and execute a program to keep track of how many times a visitor has loaded the page		2	CO4 CO6	28
12	Create and execute a PHP script to connect MySQL server from your web page	Database Connectivity using MYSQL	2	CO6	29
13	Create and execute PHP script to read customer information from customer table and display all those information in table format on output screen		2	CO6	30
14	Create and execute a PHP script for creating and deleting a file	Working with the File System	2	CO5	32
15	Create and execute a PHP script for reading and writing a file		2	CO5	34
16	Mini Project (e.g. Create a dynamic web site using PHP and MySQL)		4	CO6	36
Skill Assessment			2		
Total Hrs			**32**		

Hardware/Software Requirement:

- Configuration of PHP, Apache Web Server, MySQL
- Wamp server, XAMP server/MAMP/ LAMP

Practical 1. Create and execute PHP script to demonstrate arithmetic operators, comparison operator, and logical operator.

Prerequisite: Basic Syntax, Constants, Variables, Operators

Theory:

Arithmetic operators

Arithmetic operators do what you would expect. They are used to perform mathematics. You can use them for the main four operations (plus, minus, times, and divide) as well as to find a modulus (the remainder after a division) and to increment or decrement a value (see Table 3-1).

Table 3-1. Arithmetic operators

+	Addition	$j + 1
-	Subtraction	$j - 6
*	Multiplication	$j * 11
/	Division	$j / 4
%	Modulus (division remainder)	$j % 9
++	Increment	++$j
--	Decrement	--$j

Example	**Label**	**Outcome**
++$a, $a++	Increment	Increment $a by 1
--$a, $a--	Decrement	Decrement $a by 1

These operators can be placed on either side of a variable, and the side on which they are placed provides a slightly different effect. Consider the outcomes of the following examples:

$inv = 15; // Assign integer value 15 to $inv.

$oldInv = $inv--; // Assign $oldInv the value of $inv, then decrement $inv.

$origInv = ++$inv; // Increment $inv, then assign the new $inv value to $origInv.

Comparison operators

Comparison operators (see Table 3-11), like logical operators, provide a method to direct program flow through an examination of the comparative values of two or more variables.

Comparison Operators

Example	**Label**	**Outcome**
$a < $b	Less than	True if $a is less than $b
$a > $b	Greater than	True if $a is greater than $b
$a <= $b	Less than or equal to	True if $a is less than or equal to $b
$a >= $b	Greater than or equal to	True if $a is greater than or equal to $b
($a == 12) ? 5 : -1	Ternary	If $a equals 12, return value is 5; otherwise, return value is –1

Equality Operators

Example	**Label**	**Outcome**
$a == $b	Is equal to	True if $a and $b are equivalent
$a != $b	Is not equal to	True if $a is not equal to $b
$a === $b	Is identical to	True if $a and $b are equivalent and $a and $b have the same type

Logical operators

If you haven't used them before, logical operators may at first seem a little daunting. But just think of them the way you would use logic in English. For example, you might say to yourself, "If the time is later than 12 p.m. and earlier than 2 p.m., then have lunch." In PHP, the code for this might look something like the following (using military timing): if ($hour > 12 && $hour < 14) dolunch();

&&	*And*	$j == 3 **&&** $k == 2
and	Low-precedence *and*	$j == 3 **and** $k == 2
||	*Or*	$j < 5 **||** $j > 10
or	Low-precedence *or*	$j < 5 **or** $j > 10
!	*Not*	! ($j **==** $k)
xor	*Exclusive or*	$j **xor** $k

Example	**Label**	**Outcome**
$a && $b	AND	True if both $a and $b are true
$a AND $b	AND	True if both $a and $b are true
$a || $b	OR	True if either $a or $b is true
$a OR $b	OR	True if either $a or $b is true
!$a	NOT	True if $a is not true
NOT $a	NOT	True if $a is not true
$a XOR $b	Exclusive OR	True if only $a or only $b is true

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute php script to demonstrate the use of variables with different scope.
2. Create and execute php script to demonstrate the use of Constants.
3. Create and execute php script to demonstrate the use of String in PHP
4. Create and execute php script to demonstrate the use of echo(), printf() and sprintf() function.

Course outcomes:

1. *Identify Apache Web Server, MySQL server environment to create, debug and run PHP program.*

Practical 2. Create and execute PHP script to demonstrate Math functions

Prerequisite:
Defining a function, User Defined function, argument function, variable function, Return function, default argument, variable length argument.

Theory:
The basic requirements of any programming language include somewhere to store data, a means of directing program flow, and a few bits and pieces such as expression evaluation, file management, and text output. PHP has all these, plus tools like else and elseif to make life easier. But even with all these in our toolkit, programming can be clumsy and tedious; especially if you have to rewrite portions of very similar code each time you need them.
That's where functions and objects come in. As you might guess, a *function* is a set of statements that performs a particular function and—optionally—returns a value. You can pull out a section of code that you have used more than once, place it into a function, and call the function by name when you want the code. Functions have many advantages over contiguous, inline code. For example, they:
• Involve less typing
• Reduce syntax and other programming errors
• Decrease the loading time of program files
• Decrease execution time, because each function is compiled only once, no matter how often you call it
• Accept arguments and can therefore be used for general as well as specific cases
Objects take this concept a step further. An *object* incorporates one or more functions, and the data they use, into a single structure called a *class*.

PHP comes with hundreds of ready-made, built-in functions, making it a very rich language. To use a function, call it by name. For example, you can see the print function in action here:
print("print is a pseudo-function"); The parentheses tell PHP that you're referring to a function.

MATH functions:
Abs
This function takes negative value as input and returns the absolute (positive) value of a integer or float number.
Syntax :
abs(number);   In this function 'number' can be float or integer.

Ceil
This function takes numeric value as argument and returns the next highest integer value by rounding up value if necessary. **Syntax :** ceil($number);

Floor
This function takes numeric value as argument and returns the next lowest integer value (as float) by rounding down value if necessary. **Syntax :** floor($number);

Round
This function takes numeric value as argument and returns the next highest integer value by rounding up value if necessary. **Syntax :** round(number, precision, mode);

In this, **number** specifies the value to round, **precision** specifies the number of decimal digits to round to (Default is 0) and, **mode(optional)** specifies one of the following constants to specify the mode in which rounding occurs :
PHP_ROUND_HALF_UP : (set by Default) Rounds number up to precision decimal, when it is half way there. Rounds 1.5 to 2 and -1.5 to -2
PHP_ROUND_HALF_DOWN : Round number down to precision decimal places, when it is half way there. Rounds 1.5 to 1 and -1.5 to -1
PHP_ROUND_HALF_EVEN : Round number to precision decimal places towards the next even value.
PHP_ROUND_HALF_ODD : Round number to precision decimal places towards the next odd value.

Fmod
This function takes two arguments as input returns the floating point remainder (modulo) of division of arguments.
Syntax : fmod(x, y);
Here, **x** is dividend and **y** is divisor. The remainder (r) is defined as: x = i * y + r, for some integer i. If y is non-zero, r has the same sign as x and a magnitude less than the magnitude of y.

Min
In this function, if the first and only parameter is an array, min() returns the lowest value in that array. If at least two parameters are provided, min() returns the smallest of these values.
Syntax : min(array_values); or min(value1,value2,...);

Max
In this function, if the first and only parameter is an array, max() returns the highest value in that array. If at least two parameters are provided, max() returns the biggest of these values.
max(array_values); or max(value1,value2,...);

Pow
pow() : This function takes base and exponent as arguments and returns base raised to the power of exponent.
Syntax : pow(base,exponent);

Sqrt
This function takes numeric value as arguments and returns the square root of value. **Syntax :** sqrt(number);

rand
If this function is called without the optional min, max arguments rand() returns a pseudo-random integer between 0 and getrandmax(). If you want a random number between 12 and 56 (inclusive). Example, use rand(12, 56). **Syntax :** rand(); or rand(min,max);

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute php script to demonstrate the use of if statement, Using the else clause with if statement, switch statement, while statement, do while statement and for statement.
2. Create and execute php script to demonstrate the use of break and continue statements.
3. Create and execute php script to define user defined function, which uses passing arguments by value.
4. Create and execute php script to define user defined function, which uses passing arguments by reference.
5. Create and execute php script to use different variable function: gettype, settype, isset, strval, floatval, intval, print_r.

Course outcomes:

2. *Identify In-Built and User defined function in PHP programming.*

Practical 3. Create and execute PHP script to demonstrate Array functions

Prerequisite: Arrays: Single-Dimensional Arrays Multidimensional Arrays, Casting Arrays

Theory:
An array is traditionally defined as a group of items that share certain characteristics, such as similarity (car models, baseball teams, types of fruit, etc.) and type (e.g., all strings or integers). Each item is distinguished by a special identifier known as a key. PHP takes this definition a step further, forgoing the requirement that the items share the same data type. For example, an array could quite possibly contain items such as state names, ZIP codes, exam scores, or playing card suits. Each item consists of two components: the aforementioned key and a value. The key serves as the lookup facility for retrieving its counterpart, the value. Keys can be numerical or associative. Numerical keys bear no real relation to the value other than the value's position in the array. As an example, the array could consist of an alphabetically sorted list of state names, with key 0 representing Alabama and key 49 representing Wyoming. Using PHP syntax, this might look like the following:

$states = array(0 => "Alabama", 1 => "Alaska"...49 => "Wyoming");
Using numerical indexing, you could reference the first state in the array (Alabama) like so: $states[0]

Like many programming languages, PHP's numerically indexed arrays begin with position 0, not 1.

An associative key logically bears a direct relation to its corresponding value. Mapping arrays associatively is particularly convenient when using numerical index values just doesn't make sense. For instance, you might want to create an array that maps state abbreviations to their names. Using PHP syntax, this might look like the following:

$states = array("OH" => "Ohio", "PA" => "Pennsylvania", "NY" => "New York")
You could then reference Ohio like this: $states["OH"]

Two-dimensional arrays
Example 3-5. Defining a two-dimensional array

```
<?php
$oxo = array(array('x', ' ', 'o'),  array('o', 'o', 'x'),        array('x', 'o', ' '));        ?>
```

To then return the third element in the second row of this array, you would use the following PHP command, which will display an x: echo $oxo[1][2];

It's also possible to create arrays of arrays, known as multidimensional arrays. For example, you could use a multidimensional array to store U.S. state information. Using PHP syntax, it might look like this:

$states = array ("Ohio" => array("population" => "11,353,140", "capital" => "Columbus"), "Nebraska" => array("population" => "1,711,263", "capital" => "Omaha"));

You could then reference Ohio's population: $states["Ohio"]["population"]
This would return the following: 11,353,140

Unlike other languages, PHP doesn't require that you assign a size to an array at creation time. In fact, because it's a loosely typed language, PHP doesn't even require that you declare the array before using it, although you're free to do so. Each approach is introduced in this section, beginning with the informal variety. Individual elements of a PHP array are referenced by denoting the element between a pair of square brackets. Because there is no size limitation on the array, you can create the array simply by making reference to it, like this:
$state[0] = "Delaware"; You can then display the first element of the array $state, like this: echo $state[0];
Additional values can be added by mapping each new value to an array index, like this:
$state[1] = "Pennsylvania"; $state[2] = "New Jersey"; ... $state[49] = "Hawaii";
Interestingly, if you intend for the index value to be numerical and ascending, you can omit the index value at creation time:
$state[] = "Pennsylvania"; $state[] = "New Jersey"; ... $state[] = "Hawaii";

Testing for an Array
The **is_array() function** determines whether variable is an array, returning TRUE if it is and FALSE otherwise. Note that even an array consisting of a single value will still be considered an array. An example follows:

$states = array("Florida"); $state = "Ohio"; printf("\$states is an array: %s <br />", (is_array($states) ? "TRUE" : "FALSE")); printf("\$state is an array: %s <br />", (is_array($state) ? "TRUE" : "FALSE"));

Executing this example produces the following:
$states is an array: TRUE $state is an array: FALSE

Outputting an Array
The most common way to output an array's contents is by iterating over each key and echoing the corresponding value. For instance, a foreach statement does the trick nicely:

$states = array("Ohio", "Florida", "Texas"); foreach ($states AS $state) { echo "{$state}<br />"; }

Adding and Removing Array Elements

Adding a Value to the Front of an Array
The **array_unshift()** function adds elements to the front of the array. All preexisting numerical keys are modified to reflect their new position in the array, but associative keys aren't affected. Its prototype follows:

Int **array_unshift**(array array, mixed variable [, mixed variable...])

The following example adds two states to the front of the $states array:
$states = array("Ohio", "New York"); array_unshift($states, "California", "Texas"); // $states = array("California", "Texas", "Ohio", "New York");

Adding a Value to the End of an Array
The **array_push()** function adds a value to the end of an array, returning the total count of elements in the array after the new value has been added. You can push multiple variables onto the array simultaneously by passing these variables into the function as input parameters. Its prototype follows:

intarray_push(array array, mixed variable [, mixed variable...])

The following example adds two more states onto the $states array:

$states = array("Ohio", "New York"); array_push($states, "California", "Texas"); // $states = array("Ohio", "New York", "California", "Texas");

Removing a Value from the Front of an Array
The **array_shift()** function removes and returns the first item found in an array. If numerical keys are used, all corresponding values will be shifted down, whereas arrays using associative keys will not be affected. Its prototype follows:
mixedarray_shift(array array)
The following example removes the first state from the $states array:
$states = array("Ohio", "New York", "California", "Texas"); $state = array_shift($states); // $states = array("New York", "California", "Texas") // $state = "Ohio"

Removing a Value from the End of an Array
The **array_pop()** function removes and returns the last element from an array. Its prototype follows:
mixedarray_pop(array array)
The following example removes the last state from the $states array:
$states = array("Ohio", "New York", "California", "Texas"); $state = array_pop($states); // $states = array("Ohio", "New York", "California" // $state = "Texas"

Searching an Array
The **in_array()** function searches an array for a specific value, returning TRUE if the value is found and FALSE otherwise. Its prototype follows:
booleanin_array(mixed needle, array haystack [, boolean strict])
In the following example, a message is output if a specified state (Ohio) is found in an array consisting of states having statewide smoking bans:
$state = "Ohio"; $states = array("California", "Hawaii", "Ohio", "New York"); if(in_array($state, $states)) echo "Not to worry, $state is smoke-free!";
The optional third parameter, strict, forces in_array() to also consider type.

Searching Associative Array Keys
The function **array_key_exists()** returns TRUE if a specified key is found in an array and FALSE otherwise. Its prototype follows: booleanarray_key_exists(mixed key, array array)
The following example will search an array's keys for Ohio, and if found, will output information about its entrance into the Union:
$state["Delaware"] = "December 7, 1787"; $state["Pennsylvania"] = "December 12, 1787"; $state["Ohio"] = "March 1, 1803"; if (array_key_exists("Ohio", $state)) printf("Ohio joined the Union on %s", $state["Ohio"]);

Searching Associative Array Values

The **array_search()** function searches an array for a specified value, returning its key if located and FALSE otherwise. Its prototype follows: mixedarray_search(mixed needle, array haystack [, boolean strict])

The following example searches $state for a particular date (December 7), returning information about the corresponding state if located:

$state["Ohio"] = "March 1"; $state["Delaware"] = "December 7"; $state["Pennsylvania"] = "December 12"; $founded = array_search("December 7", $state); if ($founded) printf("%s was founded on %s.", $founded, $state[$founded]);

The output follows: Delaware was founded on December 7.

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute php script to define constructor and destructor in a class.

Course outcomes:

1. *Identify Apache Web Server, MySQL server environment to create, debug and run PHP program.*
2. *Identify In-Built and User defined function in PHP programming.*

Practical 4. Create and execute PHP script for Error-handling using exceptions

Prerequisite:
Error types: Parse error, Run time errors, Logic errors, Debugging Methodology, Error levels

Theory:
Exceptions are used to change the normal flow of a script if a specified error occurs.
What is an Exception
With PHP 5 came a new object oriented way of dealing with errors.
Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.
This is what normally happens when an exception is triggered:

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

We will show different error handling methods:

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

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

Basic Use of Exceptions
When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.
If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

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

Try, throw and catch

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

- 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

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

Multiple Exceptions

- It is possible for a script to use multiple exceptions to check for multiple conditions.
- It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

Rules for exceptions

- Code may be surrounded in a try block, to help catch potential exceptions
- Each try block or "throw" must have at least one corresponding catch block
- Multiple catch blocks can be used to catch different classes of exceptions
- Exceptions can be thrown (or re-thrown) in a catch block within a try block

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Define Notice Error, Fatal Error, Waring Error, Parse Error.
2. Create and execute PHP scripts to demonstrate the use of Notice Error, Fatal Error, Warning Error, Parse Error.
3. Create and execute PHP scripts to handle multiple exceptions.

Course outcomes:

4. *Create and execute PHP scripts by applying concept of error handling techniques of PHP.*

Practical 5. Create student registration form using text box, check box, radio button, select, submit button and display user inserted value in new PHP page

Prerequisite:
Reading data using Form Controls Text Fields, Text Areas, Checkboxes, Radio Buttons etc.
Submitting form values, using $_*Get* and $_*Post* Methods, $_REQUEST
Retrieving Submitted Data

Theory:
Handling forms is a multipart process. First a form is created, into which a user can enter the required details. This data is then sent to the web server, where it is interpreted, often with some error checking. If the PHP code identifies one or more fields that require reentering, the form may be redisplayed with an error message. When the code is satisfied with the accuracy of the input, it takes some action that usually involves the database, such as entering details about a purchase.
To build a form, you must have at least the following elements:
• An opening <form> and closing </form> tag
• A submission type specifying either a Get or Post method
• One or more input fields
• The destination URL to which the form data is to be submitted

Input Types
HTML forms are very versatile and allow you to submit a wide range of input types, from text boxes and text areas to checkboxes, radio buttons, and more.

Text boxes
The input type you will probably use most often is the text box. It accepts a wide range of alphanumeric text and other characters in a single-line box. The general format of a text box input is as follows:
<input type="text" name="*name*" size="*size*" maxlength="*length*" value="*value*">
Text areas
When you need to accept input of more than a short line of text, use a text area. This is similar to a text box, but, because it allows multiple lines, it has some different attributes. Its general format looks like this:
<textarea name="*name*" cols="*width*" rows="*height*" wrap="*type*"> </textarea>

TYPE	**ACTION**
off	Text does not wrap, and lines appear exactly as the user types them.
soft	Text wraps but is sent to the server as one long string without carriage returns and line feeds.
hard	Text wraps and is sent to the server in wrapped format with soft returns and line feeds.

Checkboxes
When you want to offer a number of different options to a user, from which he can select one or more items, checkboxes are the way to go. Here is the format to use:

<input type="checkbox" name="*name*" value="*value*" checked="checked">

Radio buttons
Radio buttons are named after the push-in preset buttons found on many older radios, where any previously depressed button pops back up when another is pressed. They are used when you want only a single value to be returned from a selection of two or more options. All the buttons in a group must use the same name and, because only a single value is returned, you do not have to pass an array.
Hidden fields
Sometimes it is convenient to have hidden form fields so that you can keep track of the state of form entry. For example, you might wish to know whether a form has already been submitted. You can achieve this by adding some HTML in your PHP code, such as the following:

echo '<input type="hidden" name="submitted" value="yes">'

<select>
The <select> tag lets you create a drop-down list of options, offering either single or multiple selections. It conforms to the following syntax:
<select name="*name*" size="*size*" multiple="multiple">

Labels
You can provide an even better user experience by utilizing the <label> tag. With it, you can surround a form element, making it selectable by clicking any visible part contained between the opening and closing <label> tags.
For example, going back to the example of choosing a delivery time, you could allow the user to click the radio button itself *and* the associated text, like this:
<label>8am-Noon<input type="radio" name="time" value="1"></label>

The submit button

To match the type of form being submitted, you can change the text of the submit button to anything you like by using the value attribute, like this:
<input type="submit" value="Search">
You can also replace the standard text button with a graphic image of your choice, using HTML such as this:
<input type="image" name="submit" src="image.gif">

Retrieving Submitted Data
The $_POST Array
PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.
When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. We can use the super global variable $_POST to collect the value of the input field: eg: `$name = $_POST['fname'];`

$_REQUEST

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

When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. We can use the super global variable $_REQUEST to collect the value of the input field: eg: $name = htmlspecialchars($_REQUEST['fname']);

$_GET (PREVIOUS EXAMPLE USING GET)

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

When to use GET?
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. This can be useful in some cases.
GET may be used for sending non-sensitive data.
Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?
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.

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute PHP scripts to demonstrate the use of Image Maps.
2. Create and execute PHP scripts to escaping Shell commands and SQL queries.
3. Create and execute PHP scripts to upload Files.
4. Create and execute PHP scripts to handle required fields on form.

Course outcomes:

3. *Illustrate form controls for presenting web based content.*

Practical 6.
Practical 7. Create a form and apply the Data Validation

Prerequisite:
Form controls, Required Fields

Theory:
Required Fields:
PHP - Display The Error Messages
Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

```
Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
```

PHP - Validate Name
The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

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

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

PHP - Validate E-mail
The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.
In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format"; }
```

PHP - Validate URL
The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

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

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute PHP scripts by using $_SERVER["PHP_SELF"] variable and htmlspecialchars().
2. Create and execute PHP scripts for employee registration with Form Data Validation.
3. Create and execute PHP scripts for user registration by adding different input type like Color, Date, datetime-local, week, month , time and validate it by using email, , number, range, search, tel, , url.

Course outcomes:

4. *Create and execute PHP scripts by applying concept of error handling techniques of PHP.*

Practical 8. Create and execute two different PHP script to demonstrate passing variables through a URL.

Prerequisite:
$_GET, Cookies, Session, URL rewriting, Setting a cookie with PHP, Creating session cookie, Accessing a Cookie, Session-Handling Process, session variables

Theory:
We can pass variable values between pages through the URL (in address bar of the browser). Here values are visible to the user and others as they appear in the address bar and also in the browser history. This is not a secure way to transfer sensitive data like password, etc.
Three symbols are used to define a string of parameters to pass:

```
  ?   concatenates the URL and the string of parameters.
  &   separates multiple parameters.
  =   assigns a value to the variable.
```

Ex: `https://www.xul.fr/demo.html?login="me"&password="1234"`
<a href='page2.php?id=2489&user=tom'>link to page2 </a>

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute PHP scripts for setting a Cookie, accessing a Cookie and destroying a Cookie.

Course outcomes:
4. Create and execute PHP scripts by applying concept of error handling techniques of PHP.
6. Create dynamic website/web based applications using PHP, MySQL database.

Practical 9. Create and execute two different PHP script to demonstrate passing variables with sessions

Prerequisite:

Cookies, Session, URL rewriting, Setting a cookie with PHP, Creating session cookie, Accessing a Cookie, Session-Handling Process, session variables

Theory:

Why and when to use Sessions?

- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL
- You are developing an application such as a shopping cart that has to temporary store information with a capacity larger than 4KB.

This is one of the secured ways the variables are passed between pages. The best example of such a system is when we see our user-id inside a member area after we logged in. In a member login system the user details are verified and once found correct, a new session is created and with user id of the member and this value is stored at server end. Every time a new page is opened by the browser, the server checks the associated session value and display the user id. This system is more secure and the member doesn't get any chance to change the values.

Creating a Session

In order to create a session, you must first call the PHP session_start function and then store your values in the $_SESSION array variable.

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute PHP scripts to start a Session, set different session variables, destroy a Session and erase all session variables.

Course outcomes:

4. Create and execute PHP scripts by applying concept of error handling techniques of PHP.

6. Create dynamic website/web based applications using PHP, MySQL database.

Practical 10. Create and execute PHP script to demonstrate passing variables with cookies

Prerequisite:
Cookies, Session, URL rewriting, Setting a cookie with PHP, Creating session cookie, Accessing a Cookie, Session-Handling Process, session variables

Theory:

What is Cookie?

A cookie is a small file with the maximum size of 4KB that the web server stores on the client computer.
Once a cookie has been set, all page requests that follow return the cookie name and value.
A cookie can only be read from the domain that it has been issued from. For example, a cookie set using the domain www.guru99.com can not be read from the domain career.guru99.com.
Most of the websites on the internet display elements from other domains such as advertising. The domains serving these elements can also set their own cookies. These are known as third party cookies.
A cookie created by a user can only be visible to them. Other users cannot see its value.
Most web browsers have options for disabling cookies, third party cookies or both.
If this is the case then PHP responds by passing the cookie token in the URL.

Why and when to use Cookies?

- Http is a stateless protocol; cookies allow us to track the state of the application using small files stored on the user's computer.
- The path were the cookies are stored depends on the browser.
- Internet Explorer usually stores them in Temporal Internet Files folder.
- Personalizing the user experience – this is achieved by allowing users to select their preferences.
- The page requested that follow are personalized based on the set preferences in the cookies.
- Tracking the pages visited by a user

Creating Cookies

Let's now look at the basic syntax used to create a cookie.

```
<?php
setcookie(cookie_name, cookie_value, [expiry_time], [cookie_path], [domain], [secure], [httponly]);
?>
```

HERE,

- Php"setcookie" is the PHP function used to create the cookie.
- "cookie_name" is the name of the cookie that the server will use when retrieving its value from the $_COOKIE array variable. It's mandatory.
- "cookie_value" is the value of the cookie and its mandatory
- "[expiry_time]" is optional; it can be used to set the expiry time for the cookie such as 1 hour. The time is set using the PHP time() functions plus or minus a number of seconds greater than 0 i.e. time() + 3600 for 1 hour.

- “[cookie_path]” is optional; it can be used to set the cookie path on the server. The forward slash “/” means that the cookie will be made available on the entire domain. Sub directories limit the cookie access to the subdomain.
- “[domain]” is optional, it can be used to define the cookie access hierarchy i.e. www.cookiedomain.com means entire domain while www.sub.cookiedomain.com limits the cookie access to www.sub.cookiedomain.com and its sub domains. *Note it’s possible to have a subdomain of a subdomain as long as the total characters do not exceed 253 characters.*
- “[secure]” is optional, the default is false. It is used to determine whether the cookie is sent via https if it is set to true or http if it is set to false.
- “[Httponly]” is optional. If it is set to true, then only client side scripting languages i.e. JavaScript cannot access them.

Retrieving the Cookie value

- $_COOKIE is a PHP built in super global variable.
- It contains the names and values of all the set cookies.
- The number of values that the $_COOKIE array can contain depends on the memory size set in php.ini.
- The default value is 1GB.

Cookies are stored at the user or the client end and values can be passed between pages using PHP. But here the client browser can reject accepting cookies by changing the security settings of the browser. So this system can fail to pass values between pages if user or client end settings are changed. But cookies are quit useful in handling user entered values and passing them between pages.

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Why and when to use Sessions?
2. Compare Cookies and Sessions.

Course outcomes:
4. Create and execute PHP scripts by applying concept of error handling techniques of PHP.
6. Create dynamic website/web based applications using PHP, MySQL database.

Practical 11. Create and execute a program to keep track of how many times a visitor has loaded the page

Prerequisite:

Cookies, Session, URL rewriting, Setting a cookie with PHP, Creating session cookie, Accessing a Cookie, Session-Handling Process, session variables

Theory:

What is a Session?

- A session is a global variable stored on the server.
- Each session is assigned a unique id which is used to retrieve stored values.
- Whenever a session is created, a cookie containing the unique session id is stored on the user's computer and returned with every request to the server. If the client browser does not support cookies, the unique php session id is displayed in the URL
- Sessions have the capacity to store relatively large data compared to cookies.
- The session values are automatically deleted when the browser is closed. If you want to store the values permanently, then you should store them in the database.
- Just like the $_COOKIE array variable, session variables are stored in the $_SESSION array variable. Just like cookies, the session must be started before any HTML tags.

Why and when to use Sessions?

- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL
- You are developing an application such as a shopping cart that has to temporary store information with a capacity larger than 4KB.

Creating a Session

- In order to create a session, you must first call the PHP session_start function and then store your values in the $_SESSION array variable.

  Destroying Session Variables
- The session_destroy() function is used to destroy the whole Php session variables.
- If you want to destroy only a session single item, you use the unset() function.

Procedure/Methodology/SourceCode:

Output:

Course outcomes:

4. Create and execute PHP scripts by applying concept of error handling techniques of PHP.

6. Create dynamic website/web based applications using PHP, MySQL database.

Practical 12. Create and execute a PHP script to connect MySQL server from your web page.

Prerequisite:
Concepts and Installation of MySQL, Integration of PHP with MySQL, Connection to the MySQL Database, Structure and syntaxes from Mysql.

Theory:
Connection to the MySQL Database
The Process
The process of using MySQL with PHP is as follows:

1. Connect to MySQL and select the database to use.
2. Build a query string.
3. Perform the query.
4. Retrieve the results and output them to a web page.
5. Repeat steps 2 to 4 until all desired data has been retrieved.
6. Disconnect from MySQL.

$conn = new mysqli($hn, $un, $pw, $db);
Creates a new object called $conn by calling a new instance of the mysqli method, passing all the values.
Error checking is achieved by referencing the $conn->connect_error property.
The -> operator indicates that the item on the right is a property or method of the object on the left. In this case, if connect_error has a value, then there was an error, so we call the die function and display that property, which details the connection error.

Building and executing a query
Sending a query to MySQL from PHP is as simple as issuing it using the query method of a connection object.
$result = $conn->query($query);
if (!$result) die($conn->error);

Closing a connection
PHP will eventually return the memory it has allocated for objects after you have finished with the script, so in small scripts, you don’t usually need to worry about releasing memory yourself. However, if you’re allocating a lot of result objects or fetching large amounts of data, it can be a good idea to free the memory you have been using to prevent problems later in your script. Therefore, note the calls to the close methods of the objects $result and $conn in the preceding scripts, as soon as each object is no longer needed, like this:
$result->close();
$conn->close();

Procedure/Methodology/SourceCode:
Output:
Questions:

1. Create and execute PHP script to insert student information into student table and display all those information on output screen.

Course outcomes: *6. Create dynamic website/web based applications using PHP, MySQL database.*

Practical 13. Create and execute PHP script to read customer information from customer table and display all those information in table format on output screen

Prerequisite:
Concepts and Installation of MySQL, Integration of PHP with MySQL, Connection to the MySQL Database, Structure and syntaxes from Mysql.

Theory:
Building and executing a query
Sending a query to MySQL from PHP is as simple as issuing it using the query method of a connection object.
$query = "SELECT * FROM classics";
$result = $conn->query($query);

Here the variable $query is assigned a string containing the query to be made, and then passed to the query method of the $conn object, which returns a result that we place in the object $result. If $result is FALSE, there was a problem and the error property of the connection object will contain the details, so the die function is called to display that error. All the data returned by MySQL is now stored in an easily interrogatable format in the $result object.

Fetching a result
Once you have an object returned in $result, you can use it to extract the data you want, one item at a time, using the fetch_assoc method of the object.
echo 'Author: ' . $result->fetch_assoc()['author']

Fetching a row

To fetch one row at a time, use function fetch_array(MYSQLI_ASSOC);
The fetch_array method can return three types of array according to the value passed to it:
$result->data_seek($j);
$row = $result->fetch_array(MYSQLI_ASSOC);
echo 'Author: ' . $row['author']

MYSQLI_NUM
Numeric array. Each column appears in the array in the order in which you defined it when you created (or altered) the table. In our case, the zeroth element of the array contains the Author column, element 1 contains the Title, and so on.

MYSQLI_ASSOC
Associative array. Each key is the name of a column. Because items of data are referenced by column name (rather than index number), use this option where possible in your code to make debugging easier and help other programmers better manage your code.

MYSQLI_BOTH
Associative and numeric array.

Associative arrays are usually more useful than numeric ones because you can refer to each column by name, such as $row['author'], instead of trying to remember where it is in the column order. So this script uses an associative array, leading us to pass MYSQLI_ASSOC.

Closing a connection
PHP will eventually return the memory it has allocated for objects after you have finished with the script, so in small scripts, you don’t usually need to worry about releasing memory yourself. However, if you’re allocating a lot of result objects or fetching large amounts of data, it can be a good idea to free the memory you have been using to prevent problems later in your script. This becomes particularly important on higher-traffic pages, because the amount of memory consumed in a session can rapidly grow. Therefore, note the calls to the close methods of the objects $result and $conn in the preceding scripts, as soon as each object is no longer needed, like this:

```
$result->close();
$conn->close();
```

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute PHP script to update specific record from student information from student table and display updated record on output screen.
2. Create and execute PHP script to delete specific record from student information from student table and display appropriate message.

Course outcomes:
6. Create dynamic website/web based applications using PHP, MySQL database.

Practical 14. Create and execute a PHP script for creating and deleting a file.

Prerequisite:

Working with directories, Creating and deleting a file, Opening a file for writing, reading, or appending.

Theory:

Opening a File

The fopen() function binds a file to a handle. Once bound, the script can interact with this file via the handle. Its prototype follows:

resource fopen(string *resource*, string *mode* [, int *use_include_path* [, *resource context*]])

While fopen() is most commonly used to open files for reading and manipulation, it's also capable of opening resources via a number of protocols, including HTTP, HTTPS, and FTP.

The *mode*, assigned at the time a resource is opened, determines the level of access available to that resource. The various modes are defined in below table.

Mode	**Description**
R	Read-only. The file pointer is placed at the beginning of the file.
r+	Read and write. The file pointer is placed at the beginning of the file.
W	Write only. Before writing, delete the file contents and return the file pointer to the beginning of the file. If the file does not exist, attempt to create it.
w+	Read and write. Before reading or writing, delete the file contents and return the file pointer to the beginning of the file. If the file does not exist, attempt to create it.
A	Write only. The file pointer is placed at the end of the file. If the file does not exist, attempt to create it. This mode is better known as Append.
a+	Read and write. The file pointer is placed at the end of the file. If the file does not exist, attempt to create it. This process is known as *appending to the file*.
x	Create and open the file for writing only. If the file exists, fopen() will fail and an error of level E_WARNING will be generated.
x+	Create and open the file for writing and writing. If the file exists, fopen() will fail and an error of level E_WARNING will be generated.

Closing a File

Good programming practice dictates that you should destroy pointers to any resources once you're finished with them. The fclose() function handles this for you, closing the previously opened file pointer specified by a file handle, returning TRUE on success and FALSE otherwise. Its prototype follows:

boolean fclose(resource *filehandle*)

The filehandle must be an existing file pointer opened using fopen() or fsockopen().

Deleting files: unlink() function is used to delete files. The term "unlink" is used because one can think of these filenames as links that join the files to the directory you are currently viewing. The function returns true on success or false on failure.

Syntax: unlink (string $filename); example : unlink ($samplefile);

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute a PHP script for checking for existence of a given file or directory.

Course outcomes:

5. *Create and execute a PHP script for various file commands.*

Practical 15. Create and execute a PHP script for reading and writing a file.

Prerequisite:
Working with directories, Creating and deleting a file, Opening a file for writing, reading, or appending.

Theory:
Reading from a File
PHP offers numerous methods for reading data from a file, ranging from reading in just one character at a time to reading in the entire file with a single operation. Many of the most useful functions are introduced in this section.

Reading a File into an Array
The file() function is capable of reading a file into an array, separating each element by the newline character, with the newline still attached to the end of each element. Its prototype follows:
array file(string *filename* [int *use_include_path* [, resource *context*]])
Reading File Contents into a String Variable
The file_get_contents() function reads the contents of a file into a string. Its prototype follows:
string file_get_contents(string *filename* [, int *use_include_path* [, resource *context* [, int *offset* [, int *maxlen*]]]])

Reading a Specific Number of Characters
The fgets() function returns a certain number of characters read in through the opened resource handle, or everything it has read up to the point when a newline or an EOF character is encountered. Its prototype follows:
string fgets(resource *handle* [, int *length*])
If the optional *length* parameter is omitted, 1,024 characters is assumed. In most situations, this means that fgets() will encounter a newline character before reading 1,024 characters, thereby returning the next line with each successive call.
Reading a File One Character at a Time
The fgetc() function reads a single character from the open resource stream specified by handle. If the EOF is encountered, a value of FALSE is returned. Its prototype follows:
string fgetc(resource *handle*)

Ignoring Newline Characters
The fread() function reads length characters from the resource specified by handle. Reading stops when the EOF is reached or when length characters have been read. Its prototype follows:
string fread(resource *handle*, int *length*)

Reading in an Entire File
The readfile() function reads an entire file specified by filename and immediately outputs it to the output buffer, returning the number of bytes read. Its prototype follows:
int readfile(string *filename* [, int *use_include_path*])

Writing a String to a File
The fwrite() function outputs the contents of a string variable to the specified resource. Its prototype follows:

int fwrite(resource *handle*, string *string* [, int *length*])
If the optional length parameter is provided, fwrite() will stop writing when length characters have been written. Otherwise, writing will stop when the end of the string is found.

Procedure/Methodology/SourceCode:

Output:

Questions:

1. Create and execute a PHP script for closing a file
2. Create and execute a PHP script for deleting a file.

Course outcomes:

1. *Create and execute a PHP script for various file commands.*

Practical 16. Mini Project (e.g. Create a dynamic web site using PHP and MySQL)

Prerequisite:
Integration of PHP with MySQL, Connection to the MySQL Database, Structure and syntaxes from Mysql.

Problem definition:

Objectives:

Procedure/Methodology/SourceCode:

Output:

Course outcomes:

6. Create dynamic website/web based applications using PHP, MySQL database.

ASSESSMENT AND EVALUATION SCHEME:

	What		To Whom	Frequency	Max Marks	Min Marks	Evidence Collected	Course Outcomes
Direct Assessment Theory	**CA** (Continuous Assessment)	PT	Students	Two PT (average of two tests will be computed)	20	--	Theory Answer Scripts	1, 2, 3, 4
		Class Room Assignments		Assignments	10	--	Assignments Books	1, 2, 3, 4
	TEE (Term End Examination)	End Exam	Students	End Of the Course	**70**	**28**	Theory Answer Scripts	1, 2, 3, 4
				Total	**100**	**40**		
Direct Assessment Practical	**CA** (Continuous Assessment)	ST	Students	One skill test at end of term	20	--	Practical Answer Scripts/ Print Outs	1, 2, 3, 4
		Journal Writing		Assignments	05	--	Journal	1, 2, 3, 4
				TOTAL	**25**	**10**		
	TEE (Term End Examination)	End Exam	Students	End Of the Course	**50**	**20**	Practical Answer Scripts	1, 2, 3, 4
Indirect Assessment	Student Feedback on course		Students	After First PT	Student Feedback Form			1, 2, 3, 4
	End Of Course			End Of The Course	Questionnaires			

SCHEME OF PRACTICAL EVALUATION:

S.N.	**Description**	**Max. Marks**
1	Writing program, Logic of the program	10
2	Debug the program	20
3	Execution of program, Program Output, Complexity of program	10
4	Viva voce	10
	TOTAL	**50**

REFERENCE & TEXT BOOKS:

S.N.	**Title**	**Author, Publisher, Edition and Year Of publication**	**ISBN Number**
1.	Learning PHP, MySQL & JavaScript with j Query, CSS & HTML5	Robin Nixon, Shroff Publishers & Distributers Private Limited Fourth edition, 2015	10: 9352130154
2.	Beginning PHP and MySQL	W. Jason Gilmore, Apress,4thEdition, 2010	10: 1430231149
3.	PHP: The Complete Reference	Steven Holzner , McGraw-Hill, 1st Edition, 2007	10: 0070223629
4.	Learning PHP, MySQL, JavaScript, CSS & HTML5	Robin Nixon, O'reilly Media , 3rd Edition, 2014	10:9352130154
5	Web Technologies: Black Book	Kogent Learning Solutions Inc.,Dreamtech Press,2009	10: 8177229974