UNIT – IV

Chapter 15

PHP

15.1 WHAT IS PHP?

- "PHP" is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is an extremely popular, Open Source scripting language, most commonly used on webservers to produce dynamic pages. It allows web developers to create dynamic content that interacts with databases.
- PHP is basically used for developing web based software applications.
- PHP is known as a server-side language. That's because the PHP doesn't get executed on your computer, but the scripts are executed on the Server. The results are then handed over to you, and displayed in your browser.
- PHP code may be embedded into HTML code.
- PHP is free to download and use
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code.
- PHP files have extension ".php"
- It is compatible with many types of databases.
- PHP runs on different platforms like Windows, Linux, UNIX, Mac OS X, etc.
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is free and easily downloadable from the official PHP resource: www.php.net

15.2 WHAT CAN PHP DO?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

15.3 WHAT IS MYSQL?

MySQL is the most popular relational database system used with the PHP language.

It is a database system used on Web that runs on a server.

It is freely available and easy to install, however if you have installed Wampserver it already there on your machine.

MySQL database server offers several advantages:

- MySQL is easy to use, yet extremely powerful, secure, and scalable.
- MySQL runs on a wide range of operating systems, including UNIX, Microsoft Windows, Apple Mac OS X, and others.
- MySQL supports standard SQL (Structured Query Language).
- MySQL is ideal database solution for both small and large applications.
- MySQL is developed, and distributed by Oracle Corporation.
- MySQL is very fast and secure. It includes data security layers that protect sensitive data from intruders.

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

To get access to a web server with PHP support, you can:

- Install Apache (or IIS) on your own server, install PHP, and MySQL
- Or find a web hosting plan with PHP and MySQL support.

15.4 PHP SYNTAX

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

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

A PHP script can be placed anywhere in the document.

A PHP script starts with **<?php** and ends with **?>**

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

The default file extension for PHP files is ".php"

Example : first.php

```
<html>
<body>
<h1>My First PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Output:

My first PHP page

Hello World!

Explanation:

PHP script uses a built-in PHP function "echo" to output the text.

PHP statements end with a semicolon.

Note: "echo" and "print" are used to output text.
If a file is saved with .html extension, then the PHP code will not be executed.

15.5 COMMENTS IN PHP

A comment in PHP code is a line that is not read or executed as part of the program.

Its only purpose is to be read by someone who is looking at the code.

There are two types of comments:

1. **Single Line Comment**

They are generally used for short explanations or notes relevant to the local code.

Single line comments may start with either # or //

2. **Multi Line Comment**

They are generally used to print multiple lines in a single print statement.

They are generally used to provide more detailed explanations when necessary.

Example: Single Line and Multi Line Comments

```
<html>
<body>
<?php
# This is a single line comment
// This is also a single line comment
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
"An example showcasing use of single and multi line comments";
?>
```

Output:

An example showcasing use of single and multi line comments

15.6 VARIABLES IN PHP

Variables are used to store values.

In PHP, a variable starts with the $ sign, followed by the name of the variable.

Syntax : $variable_name;

Example: $name="Sweta";

Naming Rules for Variables:

- All variables in PHP are denoted with a leading dollar sign ($), followed by the name of the variable.
- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).

- Variable names are case-sensitive ($name and $NAME are two different variables).
- A variable can have a short name (like x and y) or a more descriptive name (age, emp_name, total_marks).
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- 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.
- PHP does a good job of automatically converting types from one to another when necessary.

15.7 PHP ECHO AND PRINT STATEMENTS

echo and print statements are interchangeably used to output data on the screen.

Minor differences between echo and print

Echo	Print
echo has no return value.	print has a return value of 1 so it can be used in expressions.
echo can take multiple parameters.	print can take one argument.
echo is marginally faster than print	print is comparatively slower than echo.
The echo statement can be used with or without parentheses: echo or echo().	The print statement can be used with or without parentheses: print or print().

Example of echo statement:

```
<html>
<body>
<?php
$str1 = “Good”;
$str2 = “Day”;
$a = 5;
$b = 3;
echo "<h2>Hello World!</h2>";
echo “Have a ” . $str1 + $str2 .”<br>”;
echo “The”, “addition” ,”of” .$a. “and” .$b. “is : ”;
?>
</body>
</html>
```

Output:

Hello World!
Good day
The addition of 5 and 3 is: 8

Example of print statement

```
<html>
<body>
$str1 = "Good";
$str2 = "Day";
$a = 5;
$b = 3;
print "<h2>Hello World!</h2>";
print "Have a " . $str1 + $str2 ."<br>";
print "The", "addition" ,"of" .$a. "and" .$b. "is : ";
?>
</body>
</html>
```

Output:

Hello World!

Good day

The addition of 5 and 3 is: 8

15.8 PHP DATA TYPES

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL

15.8.1 PHP String

A string is a text which is a sequence of characters.

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

```
<html>
<body>
<?php
$str = "Hello World!";
echo $str;
?>
</body>
</html>
```

Output:

15.8.2 PHP Integer

Integers are whole numbers between -2,147,483,648 and 2,147,483,647.

Rules for integers :

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example, the PHP var_dump() function returns the data type and value:

```
<html>
<body>
<?php
$a = 456; // decimal number
var_dump($a);
echo "<br>";
$b = -456; // a negative number
var_dump($b);
echo "<br>";
$c = 0x1A; // hexadecimal number
var_dump($c);
echo "<br>";
$d = 0123; // octal number
var_dump($d);
?>
</body>
</html>
```

Output:

```
int(456)
int(-456)
int(26)
int(83)
```

15.8.3 PHP Float

Floating point numbers (also known as "floats", "doubles", or "real numbers") are decimal, fractional numbers or a number in exponential form.

Example:

```
<html>
<body>
<?php
```

```
$a = 1.234;
var_dump($a);
echo "<br>";
$b = 10.2e3;
var_dump($b);
echo "<br>";
$c = 4E-10;
var_dump($c);
?>
</body>
</html>
```

Output:

```
float(1.234)
float(10200)
float(4.0E-10)
```

15.8.4 PHP Booleans

A Boolean has only two possible values: TRUE (1) or FALSE (0). Booleans are often used in conditional testing.

Example:

```
<html>
<body>
<?php
// Assign the value TRUE to a variable
$x = True;
var_dump($x);
?>
</body>
</html>
```

Output:

```
bool(true)
```

15.8.5 PHP Arrays

An array is a variable that can hold more than one value at a time.

Example:

```
<html>
<body>
<?php
$colors = array("Red","Green","Blue");
var_dump($colors);
?>
</body>
</html>
```

Output:

array(3) { [0]=> string(3) "Red" [1]=> string(5) "Green" [2]=> string(4) "Blue" }

15.8.6 PHP Object

An object is a data type that not only allows storing data but also information on, how to process that data.

An object is an instance of a class.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods. Objects are created via the new keyword.

Example:

```
<html>
<body>
<?php
class Car {
function Car(){
$this->model = "BMW";
}
}
// create an object
$herbie = new Car();
// show object properties
echo $herbie->model;
?>
</body>
</html>
```

Output:

BMW

15.8.7 PHP Null

The special NULL value is used to represent empty variables in PHP. Null is a special data type which can have only one value: NULL that has no value assigned to it.

The special NULL value is used to represent empty variables in PHP.

Note: If a variable is created without a value, it is automatically assigned a value of NULL.

Example:

```
<html>
<body>
<?php
$a = NULL;
var_dump($a);
echo "<br>";
$b = "Hello World!";
```

Output:

array(3) { [0]=> string(3) "Red" [1]=> string(5) "Green" [2]=> string(4) "Blue" }

15.8.6 PHP Object

An object is a data type that not only allows storing data but also information on, how to process that data.

An object is an instance of a class.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods. Objects are created via the new keyword.

Example:

```
<html>
<body>
<?php
class Car {
function Car(){
$this->model = "BMW";
}
}
// create an object
$herbie = new Car();
// show object properties
echo $herbie->model;
?>
</body>
</html>
```

Output:

BMW

15.8.7 PHP Null

The special NULL value is used to represent empty variables in PHP. Null is a special data type which can have only one value: NULL that has no value assigned to it.

The special NULL value is used to represent empty variables in PHP.

Note: If a variable is created without a value, it is automatically assigned a value of NULL.

Example:

```
<html>
<body>
<?php
$a = NULL;
var_dump($a);
echo "<br>";
$b = "Hello World!";
```

```
$b = NULL;
var_dump($b);
?>
</body>
</html>
```

Output:

```
NULL
NULL
```

15.9 OPERATORS

Operators are symbols that tell the PHP processor to perform certain actions. They are used to perform operations on variables and values.

15.9.1 Arithmetic Operators

The arithmetic operators are used to perform arithmetical operations i.e. addition, subtraction, multiplication, division and modulus.

Following table shows list of PHP's arithmetic operators:

Operator	Description	Example	Result
+	Addition	$a + $b	Sum of $a and $b
-	Subtraction	$a - $b	Difference of $a and $b.
*	Multiplication	$a * $b	Product of $a and $b.
/	Division	$a / $b	Quotient of $a and $b
%	Modulus	$a % $b	Remainder of $a divided by $b

Example:

```
<html>
<body>
<?php
$a = 5;
$b = 3;
echo($a + $b);
echo "<br>";
echo($a - $b);
echo "<br>";
echo($a * $b);
echo "<br>";
echo($a / $b);
echo "<br>";
echo($a % $b);
?>
</body>
html>
```

Output:

8
2
15
1.67
2

15.9.2 Assignment Operators

The PHP assignment operators are used to assign numeric values to variables.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Operator	Description	Example	Is The Same As
=	Assign	$a = $b	$a = $b
+=	Add and assign	$a += $b	$a = $a + $b
-=	Subtract and assign	$a -= $b	$a = $a - $b
*=	Multiply and assign	$a *= $b	$a = $a * $b
/=	Divide and assign quotient	$a /= $b	$a = $a / $b
%=	Divide and assign modulus	$a %= $b	$a = $a % $b

Example:

```
<html>
<body>
<?php
$a = 10;
echo $a;
echo "<br>";
$a = 20;
$a += 30;
echo $a;
echo "<br>";
$a = 50;
$a -= 20;
echo $a;
echo "<br>";
$a = 5;
$a *= 25;
echo $a;
echo "<br>";
$a = 50;
$a /= 10;
echo $a;
echo "<br>";
$a = 100;
```

```
$a %= 15;
echo $a;
?>
</body>
</html>
```

Output:

```
10
50
30
125
5
10
```

15.9.3 Comparison Operators

The comparison operators are used to compare two values (number or string) in a Boolean fashion.

Operator	Name	Example	Result
==	Equal	$a == $b	True if $a is equal to $b
===	Identical	$a === $b	True if $a is equal to $b, and they are of the same type
!=	Not equal	$a != $b	True if $a is not equal to $b
<>	Not equal	$a <> $b	True if $a is not equal to $b
!==	Not identical	$a !== $b	True if $a is not equal to $b, or they are not of the same type
<	Less than	$a < $b	True if $a is less than $b
>	Greater than	$a > $b	True if $a is greater than $b
>=	Greater than or equal to	$a >= $b	True if $a is greater than or equal to $b
<=	Less than or equal to	$a <= $b	True if $a is less than or equal to $b

Example

```
<html>
<body>
<?php
$a = 25;
$b = 35;
$c = "25";
var_dump($a == $c);
echo "<br>";
var_dump($a === $c);
echo "<br>";
var_dump($a != $b);
```

```
echo "<br>";
var_dump($a !== $c);
echo "<br>";
var_dump($a < $b);
echo "<br>";
var_dump($a > $b);
echo "<br>";
var_dump($a <= $b);
echo "<br>";
var_dump($a >= $b);
?>
</body>
</html>
```

Output:

```
bool(true)
bool(false)
bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(false)
```

15.9.4 Logical Operators

The logical operators are used to combine conditional statements.

Operator	Name	Example	Result
And	And	$a and $b	True if both $a and $b are true
Or	Or	$a or $b	True if either $a or $b is true
Xor	Xor	$a xor $b	True if either $a or $b is true, but not both
&&	And	$a && $b	True if both $a and $b are true
||	Or	$a || $b	True if either $a or $b is true
!	Not	!$a	True if $a is not true

Example:

```
<html>
<body>
<?php
$year = 2014;
// Leap years are divisible by 400 or by 4 but not 100
if(($year % 400 == 0) || (($year % 100 != 0) && ($year % 4 == 0)))
{
echo "$year is a leap year.";
} else
```

```
{
echo "$year is not a leap year.";
}
?>
</body>
</html>
```

Output:

2014 is not a leap year.

15.9.5 Increment/Decrement Operators

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

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

Example:

```
<html>
<body>
<?php
$a = 10;
echo ++$a;
echo "<br>";
echo $a;
$a = 10;
echo $a++;
echo "<br>";
echo $a;
$a = 10;
echo --$a;
echo "<br>";
echo $a;
$a = 10;
echo $a--;
echo "<br>";
echo $a;
?>
</body>
</html>
```

Output:

11
11

i = 5
j = ++i
then j = 6 & i = 6.
But

i = 5
j = i++
then j = 5
i = 6

```
10
11
9
9
10
9
```

15.9.6 String Operators

There are two operators which are specifically designed for strings.

Operator	Description	Example	Result
.	Concatenation	$str1 . $str2	Concatenation of $str1 and $str2
.=	Concatenation assignment	$str1 .= $str2	Appends the $str2 to the $str1

Example:

```
<html>
<body>
<?php
$a = "Hello";
$a = " World!";
echo $a . $b; // Outputs: Hello World!
echo "<br>";
$a .= $b;
echo $a; // Outputs: Hello World!
?>
</body>
html>
```

Output:

```
Hello World!
Hello World!
```

15.9.7 Array Operators

The array operators are used to compare arrays:

Operator	Name	Example	Result
+	Union	$a + $b	Union of $a and $b
==	Equality	$a == $b	True if $a and $b have the same key/value pairs
===	Identity	$a === $b	True if $a and $b have the same key/value pairs in the same order and of the same types
!=	Inequality	$a != $b	True if $a is not equal to $b
<>	Inequality	$a <> $b	True if $a is not equal to $b
!==	Non-identity	$a !== $b	True if $a is not identical to $b

Example:

```
<html>
<body>
<?php
$x = array("a" => "Red", "b" => "Green", "c" => "Blue");
$y = array("u" => "Yellow", "v" => "Orange", "w" => "Pink");
$z = $x + $y; // Union of $x and $y
var_dump($z);
var_dump($x == $y);
echo "<br>";
var_dump($x === $y);
echo "<br>";
var_dump($x != $y);
echo "<br>";
var_dump($x <> $y);
echo "<br>";
var_dump($x !== $y);
?>
</body>
</html>
```

Output:

array(6) { ["a"]=> string(3) "Red" ["b"]=> string(5) "Green" ["c"]=> string(4) "Blue" ["u"]=> string(6) "Yellow" ["v"]=> string(6) "Orange" ["w"]=> string(4) "Pink" }
bool(false)
bool(false)
bool(true)
bool(true)
bool(true)

15.10 PHP DECISION MAKING STATEMENTS / CONDITIONAL STATEMENTS

Conditional statements are used to perform different actions based on different conditions.

Like most programming languages, PHP also allows you to write code that perform different actions based on the results of a logical or comparative test conditions at run time. This means, you can create test conditions in the form of expressions that evaluates to either true or false and based on these results you can perform certain actions.

There are several statements in PHP that you can use to make decisions:

- The **if** statement
- The **if...else** statement
- The **if...elseif....else** statement
- The **switch...case** statement

15.10.1 The If Statement

The if statement is used to execute a block of code only if the specified condition evaluates to true.

Syntax:

```
if(condition)
{
//code to be executed
}
```

Example:

```
<html>
<body>
<?php
$marks = 65;
if ($marks > 35)
{
echo "Pass";
}
?>
</body>
</html>
```

Output:

Pass

15.10.2 The If...Else Statement

The *if...else* statement allows you to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false.

Syntax:

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

Example:

```
<html>
<body>
<?php
$marks = 25;
if ($marks < 35)
{
echo "Fail";
}
else
{
echo “Pass”;
```

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

Output:

Fail

15.10.3 The If ... Elseif ...Else Statement

The *if...elseif...else* a special statement that is used to combine multiple *if...else* statements.

Syntax:

```
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
}
```

Example:

```
<html>
<body>
<?php
$marks = 62;
if ($marks < 35)
{
echo "Fail";
}
else if (marks >=35)
{
echo “Pass”;
}
else
{
echo “Invalid Marks”;
}
?>
</body>
</html>
```

Output:
pass

15.10.4 The Switch Statement

The switch-case statement is an alternative to the if...elseif...else statement.

The switch-case statement tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

Syntax:

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

Example:

```
<html>
<body>
<?php
$favcolor = "blue";
switch ($favcolor) {
case "red":
echo "Your favorite color is red!";
break;
case "blue":
echo "Your favorite color is blue!";
break;
case "green":
echo "Your favorite color is green!";
break;
default:
echo "Your favorite color is neither red, blue, nor green!";
}
?>
</body>
</html>
```

Output:

Your favorite color is blue!

15.11 LOOPING

Loops are used to execute the same block of code again and again, until a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. PHP supports four different types of loops.

- **while** — loops through a block of code until the condition is evaluate to true.
- **do...while** — the block of code executed once and then condition is evaluated. If the condition is true the statement is repeated as long as the specified condition is true.
- **for** — loops through a block of code until the counter reaches a specified number.
- **foreach** — loops through a block of code for each element in an array.

15.11.1 While Loop

The while statement will loops through a block of code until the condition in the while statement evaluate to true.

Syntax:

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

Example:

```
<html>
<body>
<?php
$i = 1;
while($i <= 3)
{
$i++;
echo "The number is " . $i . "<br>";
}
?>
</body>
</html>
```

Output:

The number is 2
The number is 3
The number is 4

The example above defines a loop that starts with $i=1. The loop will continue to run as long as $i is less than or equal to 3. The $i will increase by 1 each time the loop runs.

15.11.2 Do...While Loop

The do-while loop is a variant of while loop, which evaluates the condition at the end of each loop iteration. With a do-while loop the block of code is executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to is true.

Syntax:

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

Example:

```
<html>
<body>
<?php
$i =1;
do{
$i++;
echo "The number is " . $i . "<br>";
}
while($i <= 3);
?>
</body>
</html>
```

Output:

The number is 2
The number is 3
The number is 4

Explanation:

The above example define a loop that starts with $i=1. It will then increase $i with 1, and print the output. Then the condition is evaluated, and the loop will continue to run as long as $i is less than, or equal to 3.

Difference between while loop and do while loop

The **while loop** differs from the do-while loop in one important way — with a while loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.

With a **do-while loop**, on the other hand, the loop will always be executed once, even if the conditional expression is false, because the condition is evaluated at the end of the loop iteration rather than the beginning.

15.11.3 For Loop

The for loop repeats a block of code until a certain condition is met. It is typically used to execute a block of code for certain number of times.

Syntax:

```
for (initialization;condition;increment)
{
```

```
// Code to be executed
}
```

The parameters of for loop have following meanings:

- Initialization - it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.
- condition - in the beginning of each iteration, condition is evaluated. If it evaluates to true, the loop continues and the nested statements are executed. If it evaluates to false, the execution of the loop ends.
- increment - it updates the loop counter with a new value. It is evaluate at the end of each iteration.

Example:

```
<html>
<body>
<?php
for($i=1; $i<=3; $i++){
echo "The number is " . $i . "<br>";
}
?>
</body>
</html>
```

Output:

```
The number is 1
The number is 2
The number is 3
```

15.11.4 For Each Loop

The foreach loop is used to iterate over arrays.

Syntax:

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

Example:

```
<html>
<body>
<?php
$colors = array("Red", "Green", "Blue");
// Loop through count array
foreach($colors as $value)
{
echo $value . "<br>";
}
?>
</body>
```

PHP

```
</html>
```

Output:

Red
Green
Blue

15.12 PHP FUNCTIONS

A function is a self-contained block of code that performs a specific task.

15.12.1 Creating an Invoking Function

Syntax:

```
function functionName()
{
//Code to be executed
}
```

The declaration of a user-defined function start with the word function, followed by the name of the function you want to create followed by parentheses i.e. () and finally place your function's code between curly brackets {}.

Example:

```
<html>
<body>
<?php
function display()
{
echo "Hello world!";
}
display();
?>
</body>
</html>
```

Output:

Hello world!

15.12.2 PHP Functions with Parameters

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

Syntax:

```
function myFunc($parameter1, parameter2,...,parameterN)
{
//Code to be executed
}
```

You can define as many parameters as you like. However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called.

Example:

```
<html>
body>
<?php
// Defining function
function add($num1, $num2){
$sum = $num1 + $num2;
echo "Sum of the two numbers $num1 and $num2 is : $sum";
}
// Calling function
add(10, 20);
?>
</body>
</html>
```

Output:

Sum of the two numbers 10 and 20 is : 30

Note:

An argument is a value that you pass to a function , and a parameter is the variable within the function that receives the argument. However, in common usage these terms are interchangeable i.e. an argument is a parameter is an argument.

15.12.3 Functions with Optional parameters and Default Values

You can also create functions with optional parameters — just insert the parameter name, followed by an equals (=) sign, followed by a default value, like this.

Example:

```
<html>
<body>
<?php
// Defining function
function customFont($font, $size=1.5){
echo "<p style=\"font-family: $font; font-size: {$size}em;\">Hello, world!</p>";
}
// Calling function
customFont("Arial", 2);
customFont("Times", 3);
customFont("Courier");
?>
</body>
</html>
```

Output:

Hello, world!

Hello, world!

Explanation:

As you can see the third call to customFont() doesn't include the second argument. This causes PHP engine to use the default value for the $size parameter which is 1.5.

15.12.4 PHP Functions Returning Values

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays and objects.

Example:

```
<html>
<body>
<?php
// Defining function
function add($num1, $num2){
$total = $num1 + $num2;
return $total;
}
// Printing returned value
echo "Sum of the two numbers 5 and 10 is :" . add(5, 10);
?>
</body>
</html>
```

Output:

Sum of the two numbers 5 and 10 is :15

15.13 PHP ARRAYS

An array can store many values at once within a single variable.

Arrays are complex variables that stores multiple or a group of values under a single variable name. Suppose you want to store colors in your PHP script. Storing the colors one by one in a variable could look something like this:

Example:

```
$color1 = "Red";
$color2 = "Green";
$color3 = "Blue";
```

But what, if you want to store all the different color codes with different shades in variables and this time this not just three may be hundred. It is quite difficult to store each color name in a separate variable.

So the solution for this is an **Array.**

Types of Arrays in PHP

There are three types of arrays that you can create. These are:

- **Numeric / Indexed array** — An array with a numeric key.
- **Associative array** — An array where each key has its own specific value.
- **Multidimensional array** — An array containing one or more arrays within itself.

15.13.1 Numeric / Indexed Array

An indexed or numeric array stores each array element with a numeric index. There are two ways to create a numeric array:

a) Index are automatically assigned.

Example: $colors = array("Red", "Green", "Blue");

Note : In an indexed or numeric array, the indexes are automatically assigned and start with 0, and the values can be of any data type.

b) Index is manually assigned.

Example:

```
$colors[0] = "Red";
$colors[1] = "Green";
$colors[2] = "Blue";
```

To access the variable values refer to the array name and index.

```
<?
$colors[0] = "Red";
$colors[1] = "Green";
$colors[2] = "Blue";
echo $colors[0]. "," .$colors[1]. "and" .$colors[2]. "are Primary Colors";
?>
```

15.13.2 Associative Array

In an associative array, the keys assigned to values can be arbitrary and user defined strings.

Example: Array uses keys instead of index numbers.

```
$marks = array("John"=>65, "Smith"=>60, "Jack"=>55);
```

The following example is equivalent to the previous example, but shows a different way of creating associative arrays:

```
<?php
$marks["John"] = "65";
$marks["Smith"] = "60";
$marks["Jack"] = "55";
echo "The Highest marks scored by John are" .$marks['John'];
?>
```

15.13.3 Multidimensional Array

The multidimensional array is an array in which each element can also be an array and each element in the sub-array can be an array or further contain array within itself and so on.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

Example:

```
// Define nested array
$contacts = array(
array(
"name" => "John Fernandes",
"email" => "johnfer@mail.com",
),
array(
"name" => "Smith Dmello",
"email" => "smithdmello@mail.com",
),
array(
"name" => "Jack Coelho",
 "email" => "jcoelho@mail.com",
)
);
// Access nested value
echo "Jack Coelho's Email-id is: " . $contacts[2]["email"];
```

Output:

Jack Coelho's Email-id is: jcoelho@mail.com

15.14 PHP GET and POST

OR

15.14 PASSING INFORMATION WITH PHP

A web browser communicates with the server typically using one of the two HTTP (Hypertext Transfer Protocol) methods - GET and POST.

The HTTP GET and POST methods are used to send information to the server.

Before the browser sends the information, it encodes it using a scheme called URL encoding.

In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

15.14.1 The Get Method

In GET method the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands **(&)**. It sends the encoded user information appended to the page request. The page and the encoded information are separated by the **(?)** character.

In general, a URL with GET data will look like this:

http://www.example.com/action.php?**name**=*john*&**age**=*22*

The bold parts in the URL are the GET parameters and the italic parts are the value of those parameters. More than one parameter = value can be embedded in the URL by concatenating with ampersands (&). One can only send simple text data via GET method.

The PHP provides **$_GET** associative array to access all the sent information using GET method.

Advantages of Using the GET Method

- Since the data sent by the GET method are displayed in the URL, it is possible to bookmark the page with specific query string values.

Disadvantages of Using the GET Method

- The GET method is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
- Because the GET method assigns data to a server environment variable, the length of the URL is limited. So, there is a limitation for the total data to be sent.
- GET can't be used to send binary data, like images or word documents, to the server.
- The GET method is restricted to send up to 1024 characters only.

Example

```
<form action = "hello.php" method="get">
Name: <input type="text" name="firstname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

The "hello.php" file can now use the $_GET variable to collect data.

Note: the names of the form fields will automatically be the keys in the $_GET array.

```
Welcome <?php echo $_GET["firstname"]; ?>.<br />
Your age is <?php echo $_GET["age"]; ?> years.
```

15.14.2 The Post Method

In POST method the data is sent to the server as a package in a separate communication with the processing script. Data sent through POST method will not visible in the URL.

The PHP provides **$_POST** associative array to access all the sent information using POST method.

Advantages of Using the POST Method

- It is more secure than GET because user-entered information is never visible in the URL query string or in the server logs.
- There is a much larger limit on the amount of data that can be passed and one can send text data as well as binary data (uploading a file) using POST.

Disadvantages of Using the POST Method

- Since the data sent by the POST method is not visible in the URL, so it is not possible to bookmark the page with specific query.

Example:

```
<form action = "hello.php" method="post">
Name: <input type="text" name="firstname" />
Age: <input type="text" name="age" />
<input type="submit" />
```

The "hello.php" file can now use the $_POST variable to collect data.

Note: the names of the form fields will automatically be the keys in the $_POST array.

```
Welcome <?php echo $_POST["firstname"]; ?>.<br />
Your age is <?php echo $_POST["age"]; ?> years.
```

15.15 PHP $_REQUEST VARIABLE

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE.

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

Example:

```
<form action = "hello.php" method="post">
Name: <input type="text" name="firstname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

The "hello.php" file can now use the $_REQUEST variable to collect data.

```
Welcome <?php echo $_REQUEST["firstname"]; ?>.<br />
Your age is <?php echo $_REQUEST["age"]; ?> years.
```

15.16 PHP STRINGS AND STRING FUNCTIONS

15.16.1 Strings

Strings are sequences of characters.

Following are valid examples of string:

```
$str1="This string is in double quotes";
$str2=""; // String with zero characters
```

15.16.2 PHP String Functions

Following table displays some commonly used functions that are used to manipulate strings.

Functions	Description
addslashes()	Returns a string with backslashes before the characters that need to be escaped. These characters are single quote ('), double quote ("), backslash (\) and NULL
count_chars()	Return information about characters used in a string
echo()	Output one or more strings
trim()	Removes whitespace (or other characters) from the beginning of a string.
parse_str()	Parses the string into variables

print()	Output a string
printf()	Output a formatted string
rtrim()	Removes whitespace (or other characters) from the end of a string
str_replace()	Replace all occurrences of the search string with the replacement string (case-sensitive)
str_split()	Splits a string into an array
str_word_count()	Counts the number of words in a string
strcmp()	Binary safe comparison of two string (case sensitive)
strlen()	Returns the length of a string
strpos()	Find the position of the first occurrence of a substring in a string
strtolower()	Converts a string to lowercase
strtoupper()	Converts a string to uppercase
substr()	Return a part of a string
trim()	Removes whitespace (or other characters) from the beginning and end of a string

Note: The first character position in a string is 0 (not 1).

Example:

```
<?php
echo strlen("Hello world!"); // outputs 12
echo str_word_count("Hello world!"); // outputs 2
echo strrev("Hello world!"); // outputs !dlrow olleH
echo strpos("Hello world!", "world"); // outputs 6
echo str_replace("world", "Everyone", "Hello world!"); // outputs Hello Everyone!
?>
```

15.17 PHP FORMS AND USER INPUT

Form: A Document that containing black fields, that the user can fill the data or user can select the data. Casually the data will store in the data base.

The PHP super globals $_GET and $_POST and $_REQUEST are used to collect user input submitted through a form.

Example:

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

```
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "test.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "test.php" looks like this:

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

Output:

Welcome Sweta
Your email address is sweta@gmail.com

The same result could also be achieved using the HTTP GET method:

Following are the changes in above example to incorporate GET method :

```
<form action="test.php" method="get">
```

test.php

```
Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
```

15.18 SUPER GLOBAL ARRAYS

Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope and it can be accessed from any function, class or file without having to do anything special.

The PHP superglobal variables are:

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

15.18.1 PHP $Globals

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

Example:

```
<?php
$x = 75;
```

```
$y = 25;
function addition() {
$GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z;
?>
```

Output: 100

Explanation: In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

15.18.2 PHP $_Server

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

Example:

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

15.18.3 PHP $_Request

Refer answer 15.15

15.18.4 PHP $_Get

Refer answer 15.14.1

15.18.5 PHP $_Post

Refer answer 15.14.2

15.19 REGULAR EXPRESSIONS

Definition: A method of specifying a search string using a number of special characters that can precisely match a substring.

Purpose: Regular expressions allow you to perform complex pattern matching on strings. A small regular expression can replace the need for a large amount of code.

Example for Email Validation:

^[A-Za-z0-9._%-]+@[A-Za-z0-9._%-]+\.[A-Za-z]{2,4}$

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression.

a) POSIX Regular Expressions
b) PERL Style Regular Expressions

15.19.1 POSIX Regular Expressions

The structure of a POSIX regular expression is similar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Character Classes are a method of matching a set of characters.

Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lower-case a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

Examples:

$regex	Matches	Doesn't Match
'/p[hu]p/'	'php', 'pup'	'phup', 'pop','PHP'
'/p[a-z1-3]p/'	'php', 'pup','pap', 'pop', 'p3p'	'PHP', 'p5p'

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.

Expression	Description	Alternate Notation
+	It matches any string containing one or more times	{1, }
*	It matches any string containing zero or more times	{0, }
?	It matches any string containing zero or one times.	{0,1}
{num}	It matches any string containing a sequence of numbers.	
{min, max}	It matches any string containing a sequence of min but not more than *max* of previous characters.	
{min, }	It matches any string containing a sequence of at least one minimum of previous characters.	

Examples:

$regex	Matches	Doesn't Match
'/ph?p/'	'pp', 'php',	'phhp', 'pap'
'/ph*p/'	'pp', 'php', 'phhhhp'	'pop', 'phhohp'
'/ph+p/'	'php', 'phhhhp'	'pp', 'phyhp'
'/ph{1,3}p/'	'php', 'phhhp'	'pp', 'phhhhp'
'/(php)+/'	'php', 'phpphp', 'phpphpphp'	'ph', 'popph'

Anchors

Description: Anchors define a specific location for the match to take place.

Expression	Description
$	It matches any string at the end of line
^	It matches any string at the beginning of line
.	Matches any single character
\b	Matches word boundary
\B	Matches non - word boundary

Examples:

$regex	Matches	Doesn't Match
'/^php$/'	'php'	'php is great', 'in php we..', 'pop'
'/p\.p/'	'p.p'	'php', 'p1p'

Predefined Character Ranges

For convenience in programming, several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set-

Expression	Description
[[:alpha:]]	It matches any string containing alphabetic characters aA through zZ.
[[:digit:]]	It matches any string containing numerical digits 0 through 9.
[[:alnum:]]	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
[[:space:]]	It matches any string containing a space.

PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions –

Function	Description
ereg()	The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.
ereg_replace()	The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.
eregi()	The eregi() function searches throughout a string specified by

	pattern for a string specified by string. The search is not case sensitive.
eregi_replace()	The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
split()	The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
spliti()	The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.
sql_regcase()	The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

15.19.2 PERL Style Regular Expressions

Perl-style regular expressions are similar to their POSIX counterparts.

Meta characters

A meta character is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

Following is the list of meta characters which can be used in PERL Style Regular Expressions.

Character	Description
.	A single character
\s	A white space character (space, tab, newline)
\S	A non white space character
\d	A digit (0-9)
\D	A non digit
\w	A word character (a-z , A-Z , 0-9, _)
\W	A non word character
[aeiou]	Matches a single character in a given set
[^aeiou]	Matches a single character outside a given set
(red|green|blue)	Matches any of the alternatives specified

Modifiers

Modifiers change the matching behaviour of the regular expression.

Character	Description
I	Ignore Case (Case Insensitive)
M	Specifies that if the string has newline or carriage return characters , the ^ and $ operators will now match against a newline boundary, instead of a string boundary
O	Evaluates the expression only once.
S	Allows use of . to match a newline character
X	Allows to use whitespace in the expression for clarity
G	Globally finds all matches
Cg	Allows a search to continue even after a global match fails

PHP's Regexp PERL Compatible Functions

PHP functions that use Perl regular expressions start with "preg_". The regular expressions are in Perl format meaning, they start and end with a slash.

In order to prevent variable expansion, regular expressions are usually specified with single quotes.

Function	Description
preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
preg_replace()	The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
preg_split()	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
preg_quote()	Quote regular expression characters

Example:

```
<?php
$subject = "Jack and Jill went up the hill.";
$pattern = '/jack|jill/i';
if ( preg_match( $pattern, $subject ) )
{
printf( "Regular expression matched." );
}
?>
```

15.20 BASIC

OR

15.20 PHP ERROR HANDLING

Sometimes your application will not run as it supposed to do, resulting in an error.

Error handling is the process of catching errors raised by your program and then taking appropriate action.

There are a number of reasons that may cause errors, for example:

- The Web server might run out of disk space
- A user might have entered an invalid value in a form field
- The file or database record that you were trying to access may not exist
- The application might not have permission to write to a file on the disk
- A service that the application needs to access might be temporarily unavailable

These types of errors are known as runtime errors, because they occur at the time the script runs. They are distinct from syntax errors that need to be fixed before the script will run.

A professional application must have the capabilities to handle such runtime error gracefully. Usually this means informing the user about the problem more clearly and precisely.

15.20.1 Basic Error Handling Using The Die() Function

Consider the following example that simply tries to open a text file for reading only.

Example:

```
<?php
//Try to open a non-existent file
$file = fopen("sample.txt","r");
?>
```

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

Warning: fopen(sample.txt) [function.fopen]: failed to open stream: No such file or directory in C:\wamp\www\project\test.php on line 2

If we follow some simple steps we can prevent the users from getting such error message.

```
<?php
if(file_exists("sample.txt"))
{
$file = fopen("sample.txt","r");
}
else
{
die("Error: The file you are trying to access doesn't exist");
}
?>
```

Now if you run the above script you will get the error message like this:

Error: The file you are trying to access doesn't exist.

As you can see by implementing a simple check whether the file exist or not before trying to access it, we can generate an error message that is more meaningful to the user.

The die() function used above simply display the custom error message and terminate the current script if 'sample.txt' file is not found.

15.20.2 Creating A Custom Error Handler

You can create your own error handler function to deal with the run-time error generated by PHP engine. The custom error handler provides you greater flexibility and better control over the errors, it can inspect the error and decide what to do with the error, it might display a message to the user, log the error in a file or database or send by e-mail, attempt to fix the problem and carry on, exit the execution of the script or ignore the error altogether.

The custom error handler function must be able to handle at least two parameters (errno and errstr), however it can optionally accept an additional three parameters (errfile, errline, and errcontext), as described below:

Syntax: error_function(errno,errstr,errfile,errline,errcontext)

Parameter	Description
Required Fields	
Errno	Specifies the level of the error, as an integer. This corresponds to the appropriate error level constant (E_ERROR, E_WARNING, and so on)
Errstr	Specifies the error message as a string
Optional Fields	
Errfile	Specifies the filename of the script file in which the error occurred, as a string
Errline	Specifies the line number on which the error occurred, as a string
Errcontext	Specifies an array containing all the variables and their values that existed at the time the error occurred. Useful for debugging

Here's an example of a simple custom error handling function. This handler, customError() is triggered whenever an error occurred, no matter how trivial. It then outputs the details of the error to the browser and stops the execution of the script.

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

You need to tell the PHP to use your custom error handler function — just call the built-in set_error_handler() function, passing in the name of the function.

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

Output:

Error: [8] Undefined variable : test

UNIT – V

Chapter 16

ADVANCED PHP AND MYSQL

16.1 PHP/MySQL FUNCTIONS

MySQL provides various functions.
Several functions are listed below:

Functions	Description
mysql_close	Close MySQL connection
mysql_connect	Open a connection to a MySQL Server
mysql_create_db	Create a MySQL database
mysql_db_name	Retrieves database name from the call to mysql_list_dbs
mysql_db_query	Selects a database and executes a query on it
mysql_drop_db	Drop (delete) a MySQL database
mysql_errno	Returns the numerical value of the error message from previous MySQL operation
mysql_error	Returns the text of the error message from previous MySQL operation
mysql_field_name	Get the name of the specified field in a result
mysql_field_table	Get name of the table the specified field is in
mysql_query	Send a MySQL query
mysql_result	Get result data
mysql_select_db	Select a MySQL database
mysql_tablename	Get table name of field

16.2 PHP / MYSQL CONNECT TO A DATABASE

16.2.1 Open A Connection to Mysql Database Server

In order to access the data inside a MySQL database, we first need to open connection to the MySQL database server.

PHP provides **mysql_connect** function to open a database connection.

All communication between PHP and the MySQL database server takes place through this connection.

The basic syntax of the mysql_connect() function is given with:

mysql_connect(servername, username, password);

The parameters in the above syntax have the following meanings:

Parameter	Description
Optional — All the parameters are optional	
Servername	Specifies the server to connect to.
Username	The MySQL user name
Password	The MySQL password to get access

Example:

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
//some code
?>
```

Note: The default username for MySQL database server is 'root' and there is no password. However, to prevent your databases from intrusion and unauthorized access you should set password from MySQL accounts.

16.2.2 Closing The Mysql Database Server Connection

The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends. However, if you want to close it earlier you can do this by simply calling the PHP mysql_close() function.

Example:

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
//some code
mysql_close($con);
?>
```

16.3 CREATING DATABASE AND TABLES USING PHP AND MYSQL

As you know how to open a connection to the MySQL database server.

Now we will learn how to execute SQL query to create a database and tables.

Database Tables

A database can contain one or more tables

Each Table has a unique name.

Tables consists of rows and columns. Rows are called as **tuples** and columns are called as **fields.**

Example: Employee table

EmpName	EmpDept	EmpAge
Rinkal	BMS	25
Raj	Accounts	45

Queries

A **query** is used to extract data from the database in a readable format according to the user's request.

A **query** is an inquiry into the database using the SELECT statement.

16.3.1 Creating The Mysql Database

Since all tables are stored in a database, so first we have to create a database before creating tables.

The CREATE DATABASE statement is used to create a database in MySQL.

Let's make a SQL query using the CREATE DATABASE statement, after that we will execute this SQL query through passing it to the mysql_query() function to finally create our database. The following example creates a database named "demo".

Syntax: CREATE DATABASE database_name

Example:

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
if(mysql_query("CREATE DATABASE demo",$con))
{
echo "Database created";
}
else
{
 echo "Error creating database : " . mysql_error();
 }
 mysql_close($con);
 ?>
```

16.3.2 Adding Tables To Mysql Database / Creating Tables

Since our database is created now it's time to add some tables to it. The CREATE TABLE statement is used to create a table in MySQL database.

So let's make a SQL query using the CREATE TABLE statement, after that we will execute this SQL query through passing it to the mysql_query() function to finally create our table.

Syntax:

```
CREATE TABLE table_name
{
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
...
}
```

Example:

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
if(mysql_query("CREATE DATABASE demo",$con))
{
echo "Database created";
}
else
{
echo "Error creating database : " . mysql_error();
}
//Creating Table
mysql_select_db("demo",$con);
$sql="CREATE TABLE Employee
{
EmpName varchar(15),
EmpDept varchar(20),
EmpAge int
}";

//Execute Query
mysql_query($sql,$con);
Close Connection
mysql_close($con);
?>
```

Note: A database must be selected before a table can be created.
The database is selected with the mysql_select_db() function.

MySQL Data types

MySQL supports a number of different data types, the most important ones are summarized below.

Field Type	Description
INT	A numeric type that can accept values in the range of -2147483648 to 2147483647
DECIMAL	A numeric type with support for floating-point or decimal numbers
CHAR	A string type with a maximum size of 255 characters and a fixed length
VARCHAR	A string type with a maximum size of 255 characters and a variable length
TEXT	A string type with a maximum size of 65535 characters
DATE	A date field in the YYYY-MM-DD format
TIME	A time field in the HH:MM:SS format
DATETIME	A combined date/time type in the YYYY-MM-DD HH:MM:SS format

16.4 INSERTING DATA INTO A MYSQL DATABASE TABLE

Let's make a SQL query using the INSERT INTO statement with appropriate values, after that we will execute this SQL query through passing it to the mysql_query() function to insert data in table. Here's an example, which adds a record to the Employee table by specifying values for the 'EmpName', 'EmpDept' and 'EmpAge' fields:

Syntax:

The first form doesn't specify the column names where the data will be inserted , only their values.

1. INSERT INTO table_name VALUES (value1, value2,value3...)

   The second form specifies both the column names and the values to be inserted.

2. INSERT INTO table_name (col1, col2,col3...) VALUES (value1, value2,value3...)

Example:

```
php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
//Creating Table
mysql_select_db("demo",$con);
mysql_query("INSERT INTO Employee (EmpName, EmpDept, EmpAge)
VALUES ('Rinkal','BMS','25')");
mysql_query("INSERT INTO Employee (EmpName, EmpDept, EmpAge)
VALUES ('Raj','Accounts','35')");
//Close Connection
mysql_close($con);
?>
```

16.5 INSERT DATA INTO A DATABASE FROM AN HTML FORM

OR

16.5 BUILDING FORMS FROM QUERIES

Let's create an HTML form that can be used to insert new records to Employee table.

16.5.1 Creating The Html Form

Here's a simple HTML form that has three text <input> fields and a submit button.

```
<html>
<body>
<form action="insert.php" method="post">
EmpName : <input type="text" name="empname" />
EmpDept : <input type="text" name="empdept" />
EmpAge : <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

16.5.2 Retrieving And Inserting The Form Data

When a user clicks the submit button of the add record HTML form, in the example above, the form data is sent to 'insert.php' file. The 'insert.php' file connects to the MySQL database server, retrieves forms fields using the PHP $_POST variables and finally execute the insert query to add the records.

Here is the complete code of our 'insert.php' file:

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
mysql_select_db("demo",$con);
$sql = "INSERT INTO Employee (EmpName, EmpDept,EmpAge) VALUES
('$_POST[empname]','$_POST[empdept]','$_POST[age]')";
if(!mysql_query($sql,$con))
{
die('Error: ' . mysql_error());
}
echo "One record added";
//Close Connection
mysql_close($con);
?>
```

16.6 SELECTING DATA FROM DATABASE TABLES

The SELECT statement is used to select the records from MySQL database tables.

Let's make a SQL query using the SELECT statement, after that we will execute this SQL query through passing it to the mysql_query() function to retrieve the table data.

Syntax

SELECT column_name(s) FROM table_name

Example:

The following table selects all the data stored in the "Employee" table.

The * character selects all the data in the table.

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
mysql_select_db("demo",$con);
$result = mysql_query("SELECT * FROM Employee");
while($row = mysql_fetch_array($result))
{
echo $row['EmpName'] . " " . $row['EmpDept'] . " " . $row['EmpAge'];
echo "<br />";
//Close Connection
mysql_close($con);
?>
```

Explanation:

The above example stores the data returned by the mysql_query() function in the $result variable. Next, we use the mysql_fetch_array() function to return the first row from the recordset as an array. Each call to mysql_fetch_array() returns the next row in the recordset. the while loop loops through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['EmpName'] ,$row['EmpDept'] and $row['EmpAge']).

Output:

Rinkal BMS 26
Raj Account 35

16.7 DISPLAYING THE RESULT IN HTML TABLE

The following example will display the same data in HTML Table.

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
mysql_select_db("demo",$con);
$result = mysql_query("SELECT * FROM Employee");
```

```
echo "<table border = '2'>
<tr>
<th>EmpName</th>
<th>EmpDept</th>
<th>EmpAge</th>
</tr>";
while($row = mysql_fetch_array($result))
{
echo "<tr>";
echo "<td>" . $row['EmpName'] . "</td>";
echo "<td>" . $row['EmpDept'] . "</td>";
echo "<td>" . $row['EmpAge]' . "</td>";
echo "</tr>";
}
echo "</table>";
//Close Connection
mysql_close($con);
?>
```

Output

EmpName	EmpDept	EmpAge
Rinkal	BMS	25
Raj	Accounts	45

16.8 UPDATING DATABASE TABLE DATA

The UPDATE statement is used to change or modify the existing records in a database table. It is typically used in conjugation with the WHERE clause to apply the changes to only those records that matches specific criteria.

Syntax:

UPDATE table_name SET column1=value, column2=value,... WHERE some_column=some_value

Example:

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
mysql_select_db("demo",$con);
mysql_query("UPDATE Employee SET EmpAge = '30' WHERE EmpName = 'Rinkal' AND EmpDept = 'BMS'");
mysql_close($con);
?>
```

Output:

EmpName	EmpDept	EmpAge
Rinkal	BMS	30
Raj	Accounts	45

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

16.9 DELETING DATABASE TABLE DATA

Just as you insert records into tables, you can delete records from table using the DELETE statement. It is typically used in conjugation with the WHERE clause to delete only those records that matches specific criteria.

Syntax:

DELETE FROM table_name WHERE some_column=some_value

Example:

```
<?php
$con=mysql_connect("localhost","sweta","sweta123");
if(!$con)
{
die('Could not connect:' .mysql_error());
}
mysql_select_db("demo",$con);
mysql_query("DELETE FROM Employee WHERE EmpName = 'Raj'");
mysql_close($con);
?>
```

Output:

EmpName	EmpDept	EmpAge
Rinkal	BMS	30

Note: The WHERE clause in the DELETE statement specifies which record or records should be deleted. If you omit the WHERE clause, all records will be deleted.

16.10 STRINGS

For Answer – Refer Unit IV - 15.16

16.11 REGULAR EXPRESSIONS

For Answer – Refer Unit IV - 15.19

16.12 SESSIONS

A PHP session is used to store certain data on the server on a temporary basis.

16.12.1 What Is A Session

- Although you can store data using cookies but it has some security issues. Since cookies are stored on user's computer it is possible for an attacker to easily modify a cookie content to insert potentially harmful data in your application that might break your application.
- Also every time the browser requests a URL to the server, all the cookie data for a Web site is automatically sent to the server within the request. It means if you have sto d 5 cookies on user's system, each having 4KB in size, the browser needs to upload 20KB of data each time the user views a page, which can affect your site's performance.
- You can solve both of these issues by using the PHP session. A PHP session stores data on the server rather than user's computer. In a session based environment, every user is identified through a unique number called session identifier or SID. This unique session ID is used to link each user with their own information on the server like emails, posts, etc.

Note: The session IDs are randomly generated by the PHP engine which is almost impossible to guess. Furthermore, because the session data is stored on the server, it doesn't have to be sent with every browser request.

16.12.2 Starting A PHP Session

Before you can store any information in session variables, you must first start up the session. To begin a new session, simply call the session_start() function. It will create a new session and generate a unique session ID for the user.

Example:

```
<?php
//Starting session
session_start();
?>
```

The session_start() function first checks for an existing session ID. If it finds one, i.e. if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID.

Note: You must call the session_start() function at the beginning of the page i.e. before any output generated by your script I the browser, much like you do while setting the cookies with setcookie() function.

16.12.3 Storing And Accessing Session Data

You can store all your session data as key-value pairs in the $_SESSION[] superglobal array. The stored data can be accessed during lifetime of a session.

The following script creates a new session and registers two session variables.

Example:

```
<?php
// Starting Session
session_start();
//Storing session data
$_SESSION["firstname"] = "Sweta";
```

```
$SESSION["lastname"] = "Chheda";
?>
```

To access the session data from our previous example from any other page on the same web domain, we simply recreate the session by calling session_start() and then pass the corresponding key to the $_SESSION associative array.

```
<?php
//Starting Session
session_start();
//Accessing Session Data
echo 'Welcome,' . $_SESSION["firstname"] . ' ' . $_SESSION["lastname"];
?>
```

Output:

Welcome Sweta Chheda

Note: To access the session data in the same page there is no need to recreate the session since it has been already started on the top of the page.

16.12.4 Destroying A Session

To remove certain session data, simply unset the corresponding key of the $_SESSION associative array as shown below:

```
<?php
//Starting Session
session_start();
//Removing session data
if(isset($_SESSION["lastname"]))
{
unset($_SESSION["lastname"]);
}
?>
```

To destroy a session completely, call the session_destroy() function.

This function does not need any argument and a single call destroys all the session data.

```
<?php
//Starting session
session_start();
//Destroying session
session_destroy();
?>
```

Note:

Before destroying a session with session_destroy(), first you need to recreate the session environment if it is not already there using the session_start() function, so that there is something to destroy.

16.13 COOKIES AND HTTP

A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. They are typically used for keeping track of

information such as username that the site can retrieve to personalize the page when user visit the website next time.

A **cookie** allows you store a small amount of data within the user's browser itself.

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.

Note: Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.

16.13.1 Setting a Cookie in PHP

The setcookie() function is used to set a cookie in PHP.

Make sure you call the setcookie()function before any output generated by your script otherwise cookie will not set.

The basic syntax of this function can be given with:

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

The parameters of the setcookie() function have the following meanings:

Parameter	Description
Name	The name of the cookie.
Value	The value of the cookie. Do not store sensitive information since this value is stored on the user's computer.
Expires	The expiry date in UNIX timestamp format. After this time cookie will become inaccessible. The default value is 0.
Path	Specify the path on the server for which the cookie will be available. If set to'/', the cookie will be available within the entire domain.
Domain	Specify the domain for which the cookie is available to e.g www.example.com.
Secure	This field, if present, indicates that the cookie should be sent only if a secure HTTPS connection exists.

Note: If the expiration time of the cookie is set to 0, or omitted, the cookie will expire at the end of the session i.e. when the browser closes.

Example:

It uses setcookie() function to create a cookie named "username" and assign the value value "Sweta Chheda" to it. It also specify that the cookie will expire after 30 days (30 days * 24 hours * 60 min * 60 sec).

```
<?php
//Setting a cookie
setcookie("username","Sweta Chheda",time()+30*24*60*60);
?>
```

Note: All the arguments except the name are optional. You may also replace an argument with an empty string ("") in order to skip that argument, however to skip the expire argument use a zero(0) instead, since it is an integer.

16.13.2 Accessing Cookies Values

The PHP $_COOKIE superglobal variable is used to retrieve a cookie value.

Example:

```
<?php
// Accessing an individual cookie value
echo $_COOKIE["username"];
?>
```

Output:

Chheda

It's a good practice to check whether a cookie is set or not before accessing its value. To do this you can use the PHP isset() function, like this:

Example:

```
<?php
//Verifying whether a cookie is set or not
if(isset($_COOKIE["username"]))
{
echo "Hi" . $_COOKIE["username"];
}
else
{
echo "Welcome Guest!";
}
?>
```

16.13.3 Removing/ Deleting Cookies

You can delete a cookie by calling the same setcookie() function with the cookie name and any value (such as an empty string) however this time you need the set the expiration date in the past.

Example:

```
<?php
//Deleting a cookie
setcookie("username","",time()-3600);
?>
```

Note: You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

16.14 EMAIL

Sending email messages is very common for a web application, for example: sending a welcome email when a user create a new account on your website, sending newsletters to your registered users, or getting user feedback or comment through the website's contact from and many more. You can use the PHP built-in mail() function for creating and sending email messages to one or

more recipients dynamically from your PHP application either in a plain-text form or formatted HTML.

The PHP mail() function is used to send emails directly from the script.

Syntax:

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

The following table summarizes the parameters of this function.

Parameter	Description
Required —	The following parameters are required
To	The recipient's email address.
subject	Subject of the email to be sent. This parameter cannot contain any newline characters.
message	Defines the message to be sent. Each line should be separated with a line feed-LF (\n). Lines should not exceed 70 characters.
Optional —	The following parameters are optional
headers	This is typically used to add extra headers such as "From", "Cc", "Bcc". The additional headers should be separated with a carriage return plus a line feed-CRLF (\r\n).
parameters	Used to pass additional parameters.

Sending Plain Text Emails

The simplest way to send an email with PHP is to send a text email.

```
<?php
$to = 'swetachheda@gmail.com';
$subject = 'Test Mail';
$message = 'Hi Sweta, This is an email testing';
$from = 'rinkalgangar@yahoo.com';
//Sending Email
if(mail($to, $subject, $message))
{
echo 'Your mail has been sent successfully!!';
}
else
{
echo 'Unable to send email. Please try again';
}
?>
```

Explanation:

In the above example, we first declare the variables — recipient's email address, subject line and message body — then we pass these variables to the mail() function to send the email.