



PHP

PHP and My SQL

Syllabus

Unit-I: Building blocks of PHP: Variables, Data Types, Operators and Expressions, Constants.**Flow Control Functions in PHP:** Switching Flow, Loops, Code Blocks and Browser Output. **Working with Functions:** Defining Functions, Calling functions, returning the values from User-Defined Functions, Variable Scope, Saving State between Function calls with the Static statement, more about arguments.

Unit-II: Working with Arrays: Arrays, Creating Arrays, Some Array-Related Functions.

Working with Objects: Creating Objects, Object Instance. **Working with Strings, Dates and Time:** Formatting Strings with PHP, Investigating Strings with PHP, Manipulating Strings with PHP, Using Date and Time Functions in PHP.

Unit-III: Working with Forms: Creating Forms, Accessing Form - Input with User defined Arrays, Combining HTML and PHP code on a single Page, Using Hidden Fields to save state, Redirecting the user, Sending Mail on Form Submission, Working with File Uploads. **Working with Cookies and User Sessions:** Introducing Cookies, Setting a Cookie with PHP, Session Function Overview, Starting a Session, Working with session variables, passing session IDs in the Query String, Destroying Sessions and Unsetting Variables, Using Sessions in an Environment with Registered Users.

Unit-IV: Working with Files and Directories: Including Files with include(), Validating Files, Creating and Deleting Files, Opening a File for Writing, Reading or Appending, Reading from Files, Writing or Appending to a File, Working with Directories, Open Pipes to and from Process Using popen (), Running Commands with exec(), Running Commands with system () or passthru (). **Working with Images:** Understanding the Image-Creation Process, Necessary Modifications to PHP, Drawing a New Image, Getting Fancy with Pie Charts, Modifying Existing Images, Image Creation from User Input.

Unit-V: Interacting with MySQL using PHP: MySQL Versus MySQLi Functions, Connecting to MySQL with PHP, Working with MySQL Data. **Creating an Online Address Book:** Planning and Creating Database Tables, Creating Menu, Creating Record Addition Mechanism, Viewing Records, Creating the Record Deletion Mechanism, Adding Sub-entities to a Record.

References:

1. Julie C. Meloni, PHP MySQL and Apache, SAMS Teach Yourself, Pearson Education (2007).
2. Xue Bai Michael Ekedahl, The Web Warrior Guide to Web Programming, Thomson (2006).

INDEX

UNIT – I		
1	Building Blocks of PHP	4
2	Flow Controls in PHP	13
3	Working with Functions	17
UNIT – II		
1	Working with Arrays	20
2	Working with Objects	23
3	String, Date and Time objects	26
UNIT – III		
1	Working with Forms	29
2	Cookies and Sessions	38
UNIT – IV		
1	Working with Files and Directories	42
2	Working with Images	49
UNIT – V		
1	Interacting with MySQL using PHP	53
2	Creating Online Address Book	55

UNIT – I

Chapter – 1 : Building Blocks of PHP

1. What is PHP? Explain in detail the history of PHP?

Hypertext PreProcessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based applications. This tutorial helps you to build your base with PHP.

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

PHP has both procedure programming language and OOP language features. This mainly attributes to its evolution along the way. This means that programmers from different programming language background can pick up this language within short period of time.

2. Explain the features of PhP.

The main features of php is; it is open source scripting language so you can free download this and use. PHP is a server site scripting language. It is open source scripting language. It is widely used all over the world. It is faster than other scripting language. Some important features of php are given below;

Features of php

It is most popular and frequently used world wide scripting language, the main reason of popularity is; It is open source and very simple.

- Simple
- Faster
- Interpreted
- Open Source
- Case Sensitive
- Simplicity
- Efficiency
- Platform Independent

Simple: It is very simple and easy to use, compare to other scripting language it is very simple and easy, this is widely used all over the world.

Faster: It is faster than other scripting language e.g. asp and jsp.

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

Open Source: Open source means you no need to pay for use php, you can free download and use.

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

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

3. Write about the Uses of PHP.

PHP is a scripting language generally used to make websites. PHP is mainly used for design server side applications.

Uses of php

1. It is used for create dynamic website.
2. You can use PHP to find today's date, and then build a calendar for the month.
3. If you host banner advertisements on your website, you can use PHP to rotate them randomly.
4. Using php you can count number of visitors on your website.
5. You can use PHP to create a special area of your website for members.
6. Using php you can create login page for your user. Using PHP, you can restrict users to access some pages of your website.
7. All the database actions can be performed through PHP on most popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
8. Using php you can add, delete, modify elements within your database thru PHP.
9. It can encrypt data.
10. PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
11. It is used for send and receive e-mails. PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
12. PHP access cookies variables and set cookies very easily as an client-side scripting language and also PHP is a server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
13. PHP supports many major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.

4. What are the advantages and disadvantages of

PHP? Advantages of PHP

- **Open source:** It is developed and maintained by a large group of PHP developers, this will helps in creating a support community, abundant extension library.
- **Speed:** It is relative fast since it uses much system resource.
- **Easy to use:** It uses C like syntax, so for those who are familiar with C, it's very easy for them to pick up and it is very easy to create website scripts.
- **Stable:** Since it is maintained by many developers, so when bugs are found, it can be quickly fixed.
- **Powerful library support:** You can easily find functional modules you need such as PDF, Graph etc.
- **Built-in database connection modules:** You can connect to database easily using PHP, since many websites are data/content driven, so we will use database frequently, this will largely reduce the development time of web apps.

- **Can be run on many platforms**, including Windows, Linux and Mac, it's easy for users to find hosting service providers.

Disadvantages of PHP

- **Security:** Since it is open sourced, so all people can see the source code, if there are bugs in the source code, it can be used by people to explore the weakness of PHP.
- **Not suitable for large applications:** Hard to maintain since it is not very modular.
- **Weak type:** Implicit conversion may surprise unwary programmers and lead to unexpected bugs. For example, the strings "1000" and "1e3" compare equal because they are implicitly cast to floating point numbers.

5. What is a variable? How to declare variable in PHP? Explain their scope.

An identifier of memory location whose value can be changed during program execution are called variables. Variables are "containers" for storing information. A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Creating (Declaring) PHP Variables

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

Example

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

After the execution of the statements above, the variable **\$txt** will hold the value **Hello world!**, the variable **\$x** will hold the value **5**, and the variable **\$y** will hold the value **10.5**.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Rules for PHP variables:

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (\$age and \$AGE are two different variables)

Output Variables

The PHP echo statement is often used to output data to the screen. The following example will show how to output text and a variable:

```
<?php
$txt = "Triveni College";
```

```
echo "I love $txt!";  
?>
```

6. Explain the scope of PHP Variables.

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used. PHP has three different variable scopes:

1. local
2. global
3. static

Global and Local Scope

A variable declared **outside** a function has a **GLOBAL SCOPE** and can only be accessed outside a function:

```
<?php  
$x = 5; // global scope  
function myTest() {  
    // using x inside this function will generate an error  
    echo "<p>Variable x inside function is: $x</p>";  
}  
myTest();  
echo "<p>Variable x outside function is:  
$x</p>"; ?>
```

A variable declared **within** a function has a **LOCAL SCOPE** and can only be accessed within that function:

```
<?php  
function myTest() {  
    $x = 5; // local scope  
    echo "<p>Variable x inside function is: $x</p>";  
}  
myTest();  
// using x outside the function will generate an error  
echo "<p>Variable x outside function is: $x</p>";  
?>
```

The global Keyword

The global keyword is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function):

```
<?php  
$x = 5;  
$y = 10;  
function myTest() {  
    global $x, $y; $y  
    = $x + $y;  
}  
myTest();
```

```
echo $y; // outputs 15
```

```
?>
```

PHP also stores all global variables in an array called `$GLOBALS[index]`. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly. The example above can be rewritten like this:

```
<?php
```

```
$x = 5;
```

```
$y = 10;
```

```
function myTest() {
```

```
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
```

```
}
```

```
myTest();
```

```
echo $y; // outputs 15
```

```
?>
```

The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the **static** keyword when you first declare the variable: `<?php`

```
function myTest() {
```

```
    static $x = 0;
```

```
    echo $x;
```

```
    $x++;
```

```
}
```

```
myTest();
```

```
myTest();
```

```
myTest();
```

```
?>
```

7. What are various Data Types available in PHP?

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
- Resource

String: A string is a sequence of characters, like "Hello world!". A string can be any text inside quotes. You can use single or double quotes: `<?php`

```
$x = "Hello world!";
```

```
$y = 'Hello world!';
```



```
echo $x;  
echo "<br>";  
echo $y;  
?>
```

Integer: An integer data type is a non-decimal number 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 \$x is an integer. The PHP var_dump() function returns the data type and value:

```
<?php  
$x = 5985;    var_dump($x);  
?>
```

Float: A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example \$x is a float. The PHP var_dump() function returns the data type and value:

```
<?php  
$x = 10.365;  var_dump($x);  
?>
```

Boolean: A Boolean represents two possible states: TRUE or FALSE. Booleans are often used in conditional testing.

```
$x = true;    $y = false;
```

Array: An array stores multiple values in one single variable.

```
<?php  
$cars = array("Volvo", "BMW", "Toyota");  
var_dump($cars);  
?>
```

Object: An object is a data type which stores data and information on how to process that data. 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.

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

NULL Value: Null is a special data type which can have only one value: NULL. A variable of data type NULL is a variable that has no value assigned to it.

Note: If a variable is created without a value, it is automatically assigned a value of NULL. Variables can also be emptied by setting the value to NULL:

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

8. Explain various Operators available in PHP.

Operator is a symbol which is used to perform operations on operands, variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

Arithmetic Operators

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

Operator	Name	Example	Result
+	Addition	$\$x + \y	Sum of \$x and \$y
-	Subtraction	$\$x - \y	Difference of \$x and \$y
*	Multiplication	$\$x * \y	Product of \$x and \$y
/	Division	$\$x / \y	Quotient of \$x and \$y
%	Modulus	$\$x \% \y	Remainder of \$x divided by \$y
**	Exponentiation	$\$x ** \y	Result of raising \$x to the \$y'th power (Introduced in PHP 5.6)

Assignment Operators

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

Example:

```
$x = 50;    $x = $y;    $x+=5;
```

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

Operator	Name	Example	Result
----------	------	---------	--------

	Equal	$\$x == \y	Returns true if \$x is equal to \$y
===	Identical	$\$x === \y	Returns true if \$x is equal to \$y, and they are of the same type
!=	Not equal	$\$x != \y	Returns true if \$x is not equal to \$y
<>	Not equal	$\$x <> \y	Returns true if \$x is not equal to \$y
==	Not identical	$\$x !== \y	Returns true if \$x is not equal to \$y, or they are not of the same type
>	Greater than	$\$x > \y	Returns true if \$x is greater than \$y
<	Less than	$\$x < \y	Returns true if \$x is less than \$y
>=	Greater than or equal to	$\$x >= \y	Returns true if \$x is greater than or equal to \$y
<=	Less than or equal to	$\$x <= \y	Returns true if \$x is less than or equal to \$y

Increment / Decrement Operators: The PHP increment operators are used to increment a variable's value. The PHP decrement operators are used to decrement a variable's value.

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

Logical Operators: The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
and	And	$\$x \text{ and } \y	True if both \$x and \$y are true
or	Or	$\$x \text{ or } \y	True if either \$x or \$y is true
xor	Xor	$\$x \text{ xor } \y	True if either \$x or \$y is true, but not both
&&	And	$\$x \ \&\& \ \y	True if both \$x and \$y are true
	Or	$\$x \ \ \y	True if either \$x or \$y is true

!	Not	!\$x	True if \$x is not true
---	-----	------	-------------------------

String Operators: PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	\$txt1 . \$txt2	Concatenation of \$txt1 and \$txt2
.=	Concatenation assignment	\$txt1 .= \$txt2	Appends \$txt2 to \$txt1

Array Operators: The PHP array operators are used to compare arrays.

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

9. Define Constant. Explain different types of Constants in PHP.

Constants are like variables except that once they are defined they cannot be changed or undefined. A constant is an identifier (name) for a simple value. The value cannot be changed during the script. A valid constant name starts with a letter or underscore (no \$ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

Create a PHP Constant: To create a constant, use the define() function.

Syntax: define(*name*, *value*, *case-insensitive*)

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

The example below creates a constant with a **case-sensitive** name:

```
<?php
define("GREETING", "Welcome to Triveni!");
echo GREETING;
?>
```

The example below creates a constant with a **case-insensitive** name:

```
<?php
define("GREETING", "Welcome to Triveni!", true);
```

```
echo greeting;
```

```
?>
```

Constants are Global

Constants are automatically global and can be used across the entire script. The example below uses a constant inside a function, even if it is defined outside the function: <?php

```
define("GREETING", "Welcome to  
W3Schools.com!"); function myTest() {  
    echo GREETING;  
}  
myTest();  
?>
```

Flow Control Functions in PHP

1. Explain various Conditional Statements in PHP.

Very often when write code, we want to perform different actions for different conditions. We can use conditional statements in code to do this. In PHP we have the following conditional statements:

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif...else statement** - executes different codes for more than two conditions
- **switch statement** - selects one of many blocks of code to be executed

The if Statement: The if statement executes some code if one condition is true.

Syntax

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

Example

```
<?php  
$t = date("H");  
if ($t < "20") {  
    echo "Have a good day!";  
}  
?>
```

The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax

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

Example

```
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

Syntax

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

Example

```
<?php
$t = date("H");

if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

The switch Statement

The switch statement is used to perform different actions based on different conditions.

Use the switch statement to **select one of many blocks of code to be executed.**

Syntax

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
```

```
...
default:    code to be executed if n is different from all labels;
}
```

First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

Example

```
<?php
$favcolor = "red";

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

2. Explain various Looping statements in PHP.

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this. In PHP, we have the following looping statements:

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

The while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

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

Example

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

```
$x++;  
}  
?>
```

The do...while Loop

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

Syntax

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

Example

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

The for Loop

PHP for loops execute a block of code a specified number of times. The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter)  
    { code to be executed;  
}
```

Parameters:

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

Example

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

The foreach Loop

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

Syntax

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

For every loop iteration, the value of the current array element is assigned to \$value and the array pointer is moved by one, until it reaches the last array element.

Example

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

Working with Functions

1. What is a Function? How to create User Defined Functions in PHP?

A function is a block of statements that can be used repeatedly in a program. A function will not execute immediately when a page loads. A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function". A function name can start with a letter or underscore (not a number). Function names are NOT case-sensitive.

Syntax

```
function functionName() {  
    code to be executed;  
}
```

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php  
function writeMsg() {  
    echo "Hello world!";  
}  
writeMsg(); // call the function  
?>
```

Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with two arguments (\$fname and \$year):

Example

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}
familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument: Example

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

Returning values: To let a function return a value, use the return statement.

Example

```
<?php
function sum($x, $y) {
    $z = $x + $y;
    return $z;
}
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

2. Explain about Variables Scope in PHP.

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used. PHP has three different variable scopes:

- local
- global
- static

Global Scope: A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function.

Example

```
<?php
$x = 5; // global scope
function myTest() {
    // using x inside this function will generate an error
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is:
$x</p>"; ?>
```

Local Scope: A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function.

Example

```
<?php
function myTest() {
    $x = 5; // local scope
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

3. Explain the global and GLOBAL Keywords in PHP.

The global keyword is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function): <?php

```
$x = 5;
$y = 10;
function myTest() {
    global $x, $y;
    $y = $x + $y;
}
myTest();
echo $y; // outputs 15
?>
```

PHP also stores all global variables in an array called \$GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

Example

```
<?php
$x = 5;
$y = 10;
function myTest() {
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

```

```
myTest();  
echo $y; // outputs 15  
?>
```

4. Explain the saving state between Function calls with the Static statement. (or) PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the **static** keyword when you first declare the variable:

Example

```
<?php  
function myTest() {  
    static $x = 0;  
    echo $x;  
    $x++;  
}  
myTest();  
myTest();  
myTest();  
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

Unit-II

Working with Arrays

1. What is an Array? What are different types Arrays available in PHP?

An array is a special variable, which can hold more than one value at a time. An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP: In PHP, the array() function is used to create an array. In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

Indexed Arrays: There are two ways to create indexed arrays.

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
```

```
$cars[1] = "BMW";
```

```
$cars[2] = "Toyota";
```

The following example creates an indexed array named \$cars, assigns three elements to it, and then prints a text containing the array value

Example:

```
<?php
$scars = array("Volvo", "BMW", "Toyota");
echo "I like " . $scars[0] . ", " . $scars[1] . " and " . $scars[2] .
"."; ?>
```

Associative Arrays: Associative arrays are arrays that use named keys that you assign to them. There are two ways to create an associative array. `$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");`

Or

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script.

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old."; ?>
```

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

Multidimensional Arrays

A multidimensional array is an array containing one or more arrays. PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

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

Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays). We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15) );
```

Now the two-dimensional \$cars array contains four arrays, and it has two indices: row and column. To get access to the elements of the \$cars array we must point to the two indices (row and column):

Example

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2]."<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2]."<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold:
".$cars[2][2]."<br>"; echo $cars[3][0].": In stock: ".$cars[3][1].",
sold: ".$cars[3][2]."<br>"; ?>
```

We can also put a For loop inside another For loop to get the elements of the \$cars array (we still have to point to the two indices).

Example

```
<?php
for ($row = 0; $row < 4; $row++) {
    echo "<p><b>Row number $row</b></p>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
        echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
}
?>
```

2. Write various array related functions.

1. count() and sizeof(): These functions counts the number of elements in an array. \$colors = array("blue", "black", "red", "green");
echo "No. of elements : " . count(\$colors);

2. array_push(): This function adds one or more elements to the end of an existing array. array_push(\$arr, "element-1", "element-2", "element-2"....);

3. array_pop(): This function removes the last element of an array.
\$re = array_pop(\$arr);

4. array_shift(): This function removes the first element of an existing array. \$fe = array_shift(\$arr);

5. array_unshift(): This function adds one or more elements to the beginning of an existing array.

```
array_unshift($arr, "element-1", "element-2", ... );
```

6. array_merge(): This function combines two or more existing arrays.

```
$newarray = array_merge($array1, $array2);
```

7. array_keys(): This function returns an array containing all the key names within a given associated array.

```
$keysarray = array_keys($array);
```

8. array_values(): This function returns an array containing all the values within a given array.

```
$valuesarray = array_values($array);
```

9. shuffle(): This function randomizes the elements of a given array.

```
shuffle($array);
```

Working with Objects

1. Explain the Object Oriented Concepts.

Before we go in detail, let's define important terms related to Object Oriented Programming.

- **Class** – This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object** – An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Member Variable** – These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.
- **Member function** – These are the function defined inside a class and are used to access object data.
- **Inheritance** – When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

2. How to create classes and objects in PHP?

The general form for defining a new class in PHP is as follows:

```
<?php
```



```

class phpClass {
    var $var1;
    var $var2 = "constant string";

    function myfunc ($arg1, $arg2) {
        [..]
    }
    [..]
}
?>

```

Here is the description of each line:

- The special form **class**, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form **var**, which is followed by a conventional \$ variable name; they may also have an initial assignment to a constant value.
- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Example

An example which defines a class of Books type:

```

<?php
class Books {
    /* Member variables */
    var $price;
    var $title;

    /* Member functions */
    function setPrice($par){
        $this->price = $par;
    }
    function getPrice(){
        echo $this->price ." <br/>";
    }
    function setTitle($par){
        $this->title = $par;
    }
    function getTitle(){
        echo $this->title ." <br/>";
    }
}
?>

```

The variable **\$this** is a special variable and it refers to the same object i.e., itself.

Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using **new** operator.

```
$physics = new Books;  
$maths = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next we will see how to access member function and process member variables.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.

Following example shows how to set title and prices for the three books by calling member functions.

```
$chemistry->setTitle( "Advanced Chemistry" );  
$maths->setTitle( "Algebra" );
```

```
$physics->setPrice( 10 );  
$maths->setPrice( 7 );
```

Now you call another member functions to get the values set by in above example –

```
$physics->getTitle();  
$maths->getTitle();  
$physics->getPrice();  
$maths->getPrice();
```

This will produce the following result –

```
Physics for High School  
Algebra  
10  
15  
7
```

3. What is Constructor and Destructor Functions?

Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behaviour, by initializing many things through constructor functions.

PHP provides a special function called **__construct()** to define a constructor. We can pass as many as arguments you like into the constructor function.

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 ) {  
    $this->title = $par1;  
    $this->price = $par2;  
}
```

Destructor

Like a constructor function you can define a destructor function using function `__destruct()`. You can release all the resources with-in a destructor.

Example

```
<?php
class BankAccount{
private $accountNumber;
private $totalBalance;

public function __construct($accountNo, $initialAmount){
    $this->accountNumber = $accountNo; $this-
    >totalBalance = $initialAmount;
    }
}
$account = new BankAccount('1243845355',2000);
var_dump($account);
?>
```

Working with Strings, Dates and Time

1. Write about Formatting Strings with PHP.

printf() Function: The printf() function outputs a formatted string.

printf(*format, arg1, arg2, arg++*)

The arg1, arg2, ++ parameters will be inserted at percent (%) signs in the main string. This function works "step-by-step". At the first % sign, arg1 is inserted, at the second % sign, arg2 is inserted, etc.

- %% - Returns a percent sign
- %b - Binary number
- %c - The character according to the ASCII value
- %d - Signed decimal number (negative, zero or positive)
- %e - Scientific notation using a lowercase (e.g. 1.2e+2)
- %E - Scientific notation using a uppercase (e.g. 1.2E+2)
- %u - Unsigned decimal number (equal to or greather than zero)
- %f - Floating-point number (local settings aware)
- %F - Floating-point number (not local settings aware)

- %g - shorter of %e and %f
- %G - shorter of %E and %f
- %o - Octal number
- %s - String
- %x - Hexadecimal number (lowercase letters)
- %X - Hexadecimal number (uppercase letters)

2. Explain the concept of investigating Strings with PHP.

Strings can arrive from many sources, including user input, databases, files, and Web pages. Before begin to work with data from an external source, often will need to find out more about it. PHP provides many functions that enable you to acquire information about strings.

A Note About Indexing Strings

A string as an array of characters. We can access individual characters of a string as if they were elements of an array:

```
$test = "Triveni";
print $test[0]; // prints "T"
print $test[2]; // prints "i"
```

Finding the Length of a String with strlen()

We can use strlen() to determine the length of a string. strlen() requires a string and returns an integer representing the number of characters in the variable. if

```
(strlen($membership) == 4) {
    print "Thank you!";
} else {
    print "Your membership number must have 4 digits<P>";
}
```

Finding a Substring Within a String with strstr()

We can use strstr() to test whether a string exists embedded within another string. strstr() requires two arguments: a source string and the substring you want to find within it. The function returns false if the substring is absent. Otherwise, it returns the portion of the source string.

```
$membership = "pAB7";
if (strstr($membership, "AB")) {
    print "The substring exists!";
} else {
    print "No such substring exists!";
}
```

Finding the Position of a Substring with strpos()

The strpos() function tells you both whether a string exists within a larger string and where it is to be found. strpos() requires two arguments: the source string and the substring you are seeking. If the substring does not exist, strpos() returns false; otherwise, it returns the index at which the substring begins.

```
$membership = "mz00xyz";
if (strpos($membership, "mz") === 0) {
    print "hello mz";
}
```

```
}
```

Extracting Part of a String with substr()

The `substr()` function returns a portion of a string based on the start index and length of the portion. This function has arguments: a source string and the starting index an optional integer representing the length of the string you want returned.

```
$test = "scallywag";  
print substr($test,6); // prints "wag"  
print substr($test,6,2) // prints "wa"
```

Tokenizing a String with strtok()

We can parse a string word by word using `strtok()`. The `strtok()` function initially requires two arguments: the string to be tokenized and the delimiters by which to split the string.

Dividing a String into Tokens with strtok()

```
<html>  
<head>  
<title>Listing 13.3 Dividing a string into tokens with strtok()</title>  
</head>  
<body>  
<?php  
$test = "http://www.deja.com/qs.xp?";  
$test .= "OP=dnquery.xp&ST=MS&DBS=2&QRY=developer+php";  
$delims = "?&";  
$word = strtok($test, $delims);  
while (is_string($word)) {  
    if ($word) {  
        print "$word<br>";  
    }  
    $word = strtok($delims);  
}  
?>  
</body>  
</html>
```

3. Explain String Manipulation in PHP.

The **`trim()`** function removes whitespace and other predefined characters from both sides of a string.

`ltrim()` - Removes whitespace or other predefined characters from the left side of a string

`rtrim()` - Removes whitespace or other predefined characters from the right side of a string. The **`strip_tags()`** function strips a string from HTML, XML, and PHP tags. `<?php`

```
echo strip_tags("Hello <b><i>world!</i></b>","<b>"); ?>
```

The **`substr_replace()`** function replaces a part of a string with another string.

```
substr_replace(string,replacement,start,length)
```

```
<?php
echo substr_replace("Hello world","earth",6);
?>
```

The **strtoupper()** function converts a string to uppercase

```
<?php
echo strtoupper("Hello WORLD!"); ?>
```

The **strtolower()** function converts a string to lowercase.

```
<?php
echo strtolower("Hello WORLD."); ?>
```

4. Explain Date and Time Functions in PHP.

getdate(): The **getdate()** function returns date/time information of a timestamp or the current local date/time. Returns an associative array with information related to the timestamp:

- [seconds] - seconds
- [minutes] - minutes
- [hours] - hours
- [mday] - day of the month
- [wday] - day of the week
- [mon] - month
- [year] - year
- [yday] - day of the year
- [weekday] - name of the weekday
- [month] - name of the month

```
<?php
$d = getdate();
foreach($d as $key=>$val)
{
    echo "$key = $val <br>";
}
echo "Today's date : " . $d['mon'] . "/" . $d['mday'] . "/" . $d['year'];
?>
```

Unit-III

Working with Forms

1. Explain the process of create a simple PHP Contact Form.

It is a common requirement to have a form on almost any web site. We will create a PHP script that will send an email when a web form is submitted.

1. In the First step, create a form using HTML document as The form contains the fields: name, email and message. Where name and email are single-line text input fields and message is a text area field (multi-line text input).

Program

```
<html>
  <head>
    <title> Sending Mail </title>
  </head>
  <body>
    <form action="mail.php" method="POST"> <p>Name</p> <input
      type="text" name="name"> <p>Email</p> <input type="text"
      name="email"> <p>Message</p><textarea name="message"
      rows="6" cols="25"> </textarea> <br />
```

```
<input type="submit" value="Send"><input type="reset" value="Clear"> </form>
</body>
</html>
```

On hitting the submit button, the form will be submitted to “mail.php”. This form is submitted through the POST method. The browser sends the form submission data to the script mentioned in the ‘action’ attribute of the form.

2. Develop the PHP code as follows to receive the details entered from the above HTML code and send the text to the specific mail. The form submission method mentioned as POST in the form (method=’post’) we can access the form submission data through the `$_POST[]` array in the PHP script.

```
<?php $name = $_POST['name']; $email
    = $_POST['email']; $message =
    $_POST['message'];
    $formcontent="From: $name \n Message:
    $message"; $recipient = "emailaddress@here.com";
    $subject = "Contact Form";
    $mailheader = "From: $email \r\n";
    mail($recipient, $subject, $formcontent, $mailheader) or
    die("Error!"); echo "Thank You!";
?>
```

2. How to accessing form-input with user defined Arrays? (OR) How to receive multiple values send by a form?

The examples so far enable us to gather information from HTML elements that submit a single value per element name. This leaves us with a problem when working with SELECT elements. These elements make it possible for the user to choose multiple items. If we name the SELECTelement with a plain name, like so

```
<select name="products" multiple>
```

The script that receives this data has access to only a single value corresponding to this name. We can change this behavior by renaming an element of this kind so that its name ends with an empty set of square brackets.

Put these lines into a text file called list.php, and place that file in your Web server document root. Next, in the script that processes the form input, we find that input from the "products[]" form element created is available in an array called `$_POST[products]`. Because `products[]` is a SELECT element, we offer the user multiple choices using the option elements. We demonstrate that the user's choices are made available in an array in Listing.

An HTML Form Including a SELECT

Element <html>

```
<head>
  <title> An HTML form including a SELECT
  element</title> </head>
<body>
  <form action="listing.php" method="POST"> Name:
  <br> <input type="text" name="user"> <br>
  Address: <br> <textarea name="address" rows="5" cols="40"> </textarea>
  <br>
  Pick Products: <br> <select name="products[]" multiple>
    <option>Sonic Screwdriver</option>
    <option>Tricorder</option>
    <option>ORAC AI</option>
    <option>HAL 2000</option>
  </select> <br><br>
  <input type="submit" value="hit it!">
  </form>
</body>
```

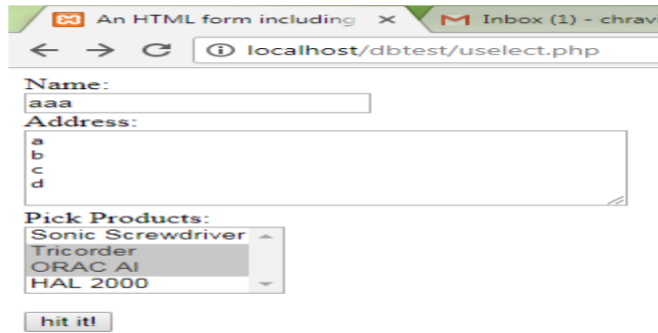
</html>

Put the following lines into a text file called **listing.php**, and place that file in your Web server document root.

Reading Input from the Form .

```
<html>
<head>
<title> Reading input from the form</title>
</head>
<body>
<?php
  print "Welcome <b>$_POST[user]</b><p>\n\n";
  print "Your address is:<p>\n\n<b>$_POST[address]</b><p>\n\n";
  print "Your product choices are:<p>\n\n"; if
  (!empty($_POST[products])) {
    print "<ul>\n\n";
    foreach ($_POST[products] as $value) {
      print "<li>$value\n";
    }
    print "</ul>";
  }
?>
</body>
</html>
```

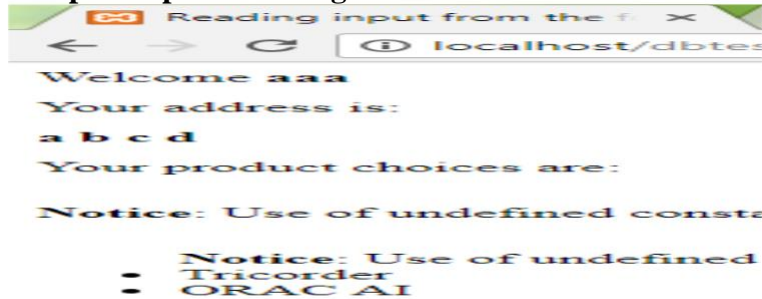
Form created in List.



The screenshot shows a web browser window with two tabs: "An HTML form including" and "Inbox (1) - chravi". The address bar shows "localhost/dbtest/uselect.php". The form contains the following elements:

- Name:** A text input field containing "aaa".
- Address:** A text area containing "a", "b", "c", and "d" on separate lines.
- Pick Products:** A dropdown menu with options: "Sonic Screwdriver", "Tricorder", "ORAC AI", and "HAL 2000". "Tricorder" is selected.
- hit it!** A submit button.

Sample output of Listing.



3. How to Combine HTML and PHP Code on a Single Page?

When building a complex page, at some point you will be faced with the need to combine PHP and HTML to achieve your needed results. At first point, this can seem complicated, since PHP and HTML are two separate languages, but this is not the case. PHP is designed to interact with HTML and PHP scripts can be included in an HTML page without a problem.

In an HTML page, PHP code is enclosed within special PHP tags. When a visitor opens the page, the server processes the PHP code and then sends the output (not the PHP code itself) to the visitor's browser. Actually it is quite simple to integrate HTML and PHP. A PHP script can be treated as an HTML page, with bits of PHP inserted here and there. Anything in a PHP script that is not contained within `<?php ?>` tags is ignored by the PHP compiler and passed directly to the web browser. The example below you can see what a full PHP script might look like:

```
<head></head>
<body>
Hello, today is <?php echo date('l, F jS, Y'); ?>.
</body>
</html>
```

4. Redirecting to Same Page (or) dynamic messaging in PHP.

In some circumstances, we may want to include form-parsing code on the same page as a hard-coded HTML form. Such a combination can be useful if we need to present the same form to the user more than once. We would have more flexibility if we were to write the entire page dynamically, of course, one of the great strengths of PHP. Where we should create functions that can be called from within the HTML code, and can be reused in other projects.

Almost all browsers will submit a form to its current page if the form element's action attribute is omitted. However, explicitly tell the browser to submit a form back to its own document by using the predefined `$PHP_SELF` variable.

```
<form action="<?php print $PHP_SELF?>">
```

We begin to build up the PHP element of the page for this process. First, we need to define the number that the user will guess. Suppose we assign '42' to the `$num_to_guess` variable. Next, we need to decide whether the form has been submitted; otherwise, we will attempt to assess variables that have not yet been made available. We can test for submission by testing for the existence of the variable `$guess`. `$guess` will have been made available as a global variable if your script has been sent a "guess" parameter. If this isn't present, we can safely assume that the user has arrived at the page without submitting a form. If the value is present, we can go ahead and test the value it contains. The test for the presence of the `$guess` variable takes place.

A PHP Number Guessing Script

```
<?php
    $num_to_guess = 42;
    $message = "";
    $guess = $_POST["guess"];
    if ( !isset( $guess ) )
        $message = "Welcome to the guessing
machine!"; elseif ( $guess > $num_to_guess )
        $message = "$guess is too big! Try a smaller
number"; elseif ( $guess < $num_to_guess )
        $message = "$guess is too small! Try a larger
number"; else // must be equivalent
        $message = "Well done!";
?>
<html>
    <head>
        <title> A PHP number guessing script</title>
    </head>
    <body>
        <h1>
            <?php print $message ?>
        </h1>
        <form method="POST">
            Type your guess here: <input type="text" name="guess">
        </form>
    </body>
</html>
```

The bulk of this script consists of an if statement that determines which string to assign to the variable `$message`. If the `$guess` variable has not been set, we assume that the user has arrived for the first time and assign a welcome string to the `$message` variable.

Otherwise, we test the `$guess` variable against the number we have stored in `$num_to_guess`, and assign advice to `$message` accordingly. We test whether `$guess` is larger than `$num_to_guess`, and whether it is smaller than `$num_to_guess`. If `$guess` is

neither larger nor smaller than \$num_to_guess, we can assume that it is equivalent and assign a congratulations message to the variable. Now all we need to do is print the \$message variable within the body of the HTML.

5. How Hidden Fields used to Save State in a web page?

A hidden field behaves exactly the same as a text field, except that the user cannot see it, unless he views the HTML source of the document that contains it. The following code adds a hidden field to the number guessing script and some PHP to work with it.

Saving State with a Hidden Field

```
<?php
    $num_to_guess = 42;
    $num_tries = $_POST["num_tries"];
    $message = "";
    $guess = $_POST["guess"];
    if ( ! isset( $guess ) )
        $message = "Welcome to the guessing
machine!"; elseif ( $guess > $num_to_guess )
        $message = "$guess is too big! Try a smaller
number"; elseif ( $guess < $num_to_guess )
        $message = "$guess is too small! Try a larger
number"; else // must be equivalent
        $message = "Well done!";
    $guess = (int) $guess;
?>
<html>
    <head>
        <title>Saving state with a hidden
field</title> </head>
    <body>
        <h1>
            <?php print $message ?>
        </h1>
        Guess number: <?php print $num_tries;
$num_tries++;?> <form method="POST">
            Type your guess here:
            <input type="text" name="guess" value="<?php print $guess?>">
            <input type="hidden" name="num_tries" value="<?php print
$num_tries?>"> </form>
        </body>
</html>
```

The hidden field given the name "num_tries". We also use PHP to write its value, so that the user can always see his last guess. This technique is useful for scripts that parse user input. If we were to reject a form submission for some reason we can at least allow our user to edit his previous query.

When you need to output the value of an expression to the browser, you can of course use print() or echo(). When you are entering PHP mode explicitly to output such a value you can also take advantage of a special extension to PHP's short opening tags. If you add an

equals (=) sign to the short PHP opening tag, the value contained will be printed to the browser. So `<? print $test;?>` is equivalent to `<?=$test?>`

Within the main PHP code, we use a ternary operator to increment the \$num_tries variable. If the \$num_tries variable is set, we add one to it and reassign this incremented value; otherwise, we initialize \$num_tries to 0. Within the body of the HTML, we can now report to the user how many guesses he has made.

6. How to Redirect the User within the application?

When a server script communicates with a client, it must first send some headers that provide information about the document to follow. PHP usually handles this automatically, but we can send our own header lines with PHP's header() function.

To call the header() function, it must be sure that no output has been sent to the browser. The first time that content is sent to the browser, PHP will send out headers and we cannot include our header. Any output from document, even a line break or a space outside of the script tags will cause headers to be sent. If we intend to use the header() function in a script we must make certain that nothing precedes the PHP code that contains the function call.

We can see headers sent in response to a request by using a telnet client. Connect to a Web host at port 80 and then type **HEAD /path/to/file.html HTTP/1.0** followed by two returns. The headers should be displayed on your client.

Redirect to another webpage or location:

By sending a "Location" header instead of PHP's default, we can cause the browser to be redirected to a new page:

```
header( "Location: congrats.html" );
```

Assuming that we have created a suitable page called "congrats.html", we can amend our number guessing script to redirect the user if the user guesses correctly (as in the previous example).

Using header() to Send Raw Headers

```
<?php
$num_to_guess = 42;
$message = "";
$guess = $_POST["guess"];
if ( !isset( $guess ) )
    $message = "Welcome to the guessing
machine!"; elseif ( $guess > $num_to_guess )
    $message = "$guess is too big! Try a smaller
number"; elseif ( $guess < $num_to_guess )
    $message = "$guess is too small! Try a larger
number"; else // must be equivalent
    header( "Location: congrats.html" );
exit;
?>
<html>
<head>
<title> A PHP User Redirect Test </title>
</head>
```

```

<body>
<h1>
<?php print $message ?>
</h1>
<form method="POST">
    Type your guess here: <input type="text" name="guess">
</form>
</body>
</html>

```

The else clause of our if statement now causes the browser to request "congrats.html". We ensure that all output from the current page is aborted with the exit statement, which immediately ends execution and output, whether HTML or PHP.

7. Write PHP script for Sending Mail on Form Submission.

It is a common requirement to have a form that will send an email when a web form is submitted. To send the mail thru PHP we use mail() function.

Before sending mail using mail() function, a few directives must be setup such as SMTP, sendmail_from options in php.ini file so that the function works properly.

1. In the First step, create a form using HTML document as The form contains the fields: name, email and message. Where name and email are single-line text input fields and message is a text area field (multi-line text input).

Program

```

<html>
<head>
<title> Sending Mail </title>
</head>
<body>
<form action="mail.php" method="POST"> <p>Name</p> <input
type="text" name="name"> <p>Email</p> <input type="text"
name="email"> <p>Message</p><textarea name="message"
rows="6" cols="25"> </textarea> <br />

<input type="submit" value="Send"><input type="reset"
value="Clear"> </form>
</body>
</html>

```

On hitting the submit button, the form will be submitted to "mail.php". This form is submitted through the POST method. The browser sends the form submission data to the script mentioned in the 'action' attribute of the form.

2. Develop the PHP code as follows to receive the details entered from the above HTML code and send the text to the specific mail. The form submission method mentioned as POST in the form (method='post') we can access the form submission data through the \$_POST[] array in the PHP script.

```

<?php $name = $_POST['name'];

```

```

$email = $_POST['email'];
$message = $_POST['message'];
$formcontent="From: $name \n Message:
$message"; $recipient = "emailaddress@here.com";
$subject = "Contact Form";
$mailheader = "From: $email \r\n";
mail($recipient, $subject, $formcontent, $mailheader) or
die("Error!"); echo "Thank You!";

```

?>

The mail() function requires four parameters: the recipient, the subject, the message, and any additional mail headers. The order of the parameters should be as in the above program.

8. How to upload a File in PHP?

Web browsers support file uploads, and so, of course, does PHP. Information about the uploaded file becomes available in the \$_FILES super global, which is indexed by the name of the uploaded field (or fields) in the form. The corresponding value for each of these keys is an associative array used in the fileupload() function.

Steps:

First, we need to create the HTML forms that include file upload fields must include an ENCTYPE argument: **ENCTYPE="multipart/form-data"**.

An optional hidden field MAX_FILE_SIZE that can be inserted before the file upload field have a value representing the maximum size in bytes of the file that is willing to accept. This size cannot override the maximum size set in the upload_max_filesize field in your php.ini file that defaults to 2 megabytes.

After the MAX_FILE_SIZE field has been entered, add the upload field itself. This is simply an INPUT element with a TYPE argument of "file".

The \$file_dir and \$file_url variables to store path information. The foreach statement to loop through every element in the \$_FILES array. Once the form has been submitted the \$_FILES array will be populated. The foreach loop stores the upload file's name in the \$file_name variable and the file information in the \$file_ array variable. We can then output the information about the upload.

The is_uploaded_file() function accepts a path to an uploaded file and returns true only if the file is a valid upload file. This function therefore enhances the security of scripts.

Now copy the file from its temporary home to a new directory using move_uploaded_file() function from one place to another. move_uploaded_file() requires a path to the source file and a path to the destination. It will return true if the move was successful and false if the file was not a valid upload file, or if it could not be found.

File Upload Global Variables

<i>Element</i>	<i>Contains</i>	<i>Example</i>
----------------	-----------------	----------------



<code>\$_FILES['fupload']['name']</code>	Name of uploaded file	test.gif
<code>\$_FILES['fupload']['tmp_name']</code>	Path to temporary file	/tmp
<code>\$_FILES['fupload']['size']</code>	Size (in bytes) of uploaded file	6835
<code>\$_FILES['fupload']['type']</code>	Mime type of uploaded file(when given by client)	image/gif

A Simple File Upload Form

```

<html>
  <head>
    <title> A file upload script</title>
  </head>
  <?php
    $file_dir = "C:/xampp";
    $file_url = "C:/xampp";
    foreach( $_FILES as $file_name => $file_array ) { print
      "path: ".$file_array['tmp_name']."<br>\n"; print
      "name: ".$file_array['name']."<br>\n"; print
      "type: ".$file_array['type']."<br>\n"; print "size:
      ".$file_array['size']."<br>\n";
      if ( is_uploaded_file( $file_array['tmp_name'] ) ) {
        move_uploaded_file( $file_array['tmp_name'], "$file_dir/$file_name" )
        or die ("Couldn't copy");
      print "<img
      src=\"$file_url/$file_name\"><p>\n\n"; }
    }
  ?>
  <body>
    <form enctype="multipart/form-data" method="POST">
      <input type="hidden" name="MAX_FILE_SIZE" value="51200">
      <input type="file" name="fupload"><br> <input type="submit"
      value="Send file!">
    </form>
  </body>
</html>

```


Working with Cookies and User Session

1. What is a Cookie? How to store and remove cookies in PHP?

A cookie is a small amount of data stored by the user's browser in compliance with a request from a server or script. A single host can request that up to 20 cookies be stored by a user's browser. Each cookie consists of a name, value, and expiration date, as well as host and path information. The size of an individual cookie is limited to 4 KB.

After a cookie is set, only the originating host can read the data, ensuring that the user's privacy is respected. Furthermore, the user either configure the browser to notify the receipt of all cookies or even refuse all cookie requests. So cookies cannot be an essential element of the environment.

Typical Server Headers Sent from a PHP Script

```
HEAD /devdemos/cootest.php HTTP/1.0
HTTP/1.1 200 OK
Date: Wed, 18 Oct 2017 14:32:28 GMT
Server: Apache/2.2.8 (Unix) PHP/5.2.5
X-Powered-By: PHP/5.2.5
Set-Cookie: myCookie
Connection: close
Content-Type: text/html
```

Setting a Cookie: We can set a cookie in PHP in two ways

First, use the `header()` function to set the Set-Cookie header. The `header()` function requires a string that will be included in the header section of the server response. Because the headers are sent automatically `header()` must be called before any output at all is sent to the browser:

```
header ("Set-Cookie: myCookie=triveni; expires=Thu, 19-Oct-17 15:30:59 GMT; path=/; domain=mydomain.com");
```

Second, using the `setcookie()` function. It should be called before any other content is sent to the browser. The function accepts the cookie name, cookie value, expiration date in UNIX format, path, domain and integer that should be set to 1 if the cookie is only to be sent over a secure connection. All arguments are optional except the first one(cookie name).

```
setcookie("myCookie", "triveni", time()+3600, "/", "mydomain.com", 0);
```

Deleting a cookie:

To delete a cookie, call `setcookie()` function with the name argument only as: `setcookie("myCookie");`. The safest way to remove a cookie is to set the cookie with a date that has already expired:

```
setcookie("myCookie", "", time()-60, "/", "mydomain.com");
```

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

```
<?php
```

```
$value = "Hello world!";
```

```
// cookie will expire when the browser close
```

```
setcookie("myCookie", $value);
```

```
// cookie will expire in 1 hour
```

```
setcookie("myCookie", $value, time() + 3600);
```

```
// cookie will expire in 1 hour, and will only be available
```

```

// within the php directory + all sub-directories of php
setcookie("myCookie", $value, time() + 3600, "/php/");

echo "<p> The stored cookie value = " . $_COOKIE["myCookie"] . "</p>";
?>
<html>
  <body>...some code...
  </body>
</html>

```

2. Write short note on cookies.

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

There are three steps involved in identifying returning users –

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.
- The setcookie() function defines a cookie to be sent along with the rest of the HTTP headers.
- A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, we can both create and retrieve cookie values.
- The name of the cookie is automatically assigned to a variable of the same name. For example, if a cookie was sent with the name "user", a variable is automatically created called \$user, containing the cookie value.
- **Note: The setcookie() function must appear BEFORE the <html> tag.**

3. What is a Session? How to handle Sessions in PHP?

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session. A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

Session functions provide a unique identifier to a user, which can then be used to store and acquire information linked to that ID, when a visitor accesses a session-enabled page, either a new identifier is allocated or the user is reassociated with one that was already visited. Any variables that have been associated with the session become available in the code through the \$_SESSION super global.

A session ends when the user loses the browser or after leaving the site, or the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function. This function first checks if a session is already started and if none is started then it starts one. The session_start() function must be the very first thing in your document. Before any HTML tags.

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

The following example starts a session then register a variable called **counter** that is incremented each time the page is visited during the session. Make use of **isset()** function to check if session variable is already set or not.

```
<?php
session_start();

if( isset( $_SESSION['counter'] ) ) {
    $_SESSION['counter'] += 1;
}else {
    $_SESSION['counter'] = 1;
}
$msg = "You have visited this page ". $_SESSION['counter'];
$msg .= "in this session.";
?>

<html>

    <head>
        <title>Setting up a PHP session</title>
    </head>
    <body>
        <?php echo ( $msg ); ?>
    </body>
</html>
```

It will produce the following result – **You have visited this page 1 in this session.** And increases the count for each time it refreshed.

Destroying a PHP Session

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

Here is the example to unset a single variable

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

Here is the call which will destroy all the session variables

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

4. How to Pass Session IDs in/to the Query String?

So far, you have relied on a cookie to save the session ID between script requests. On its own, this is not the most reliable way of saving state because you cannot be sure that the browser will accept cookies. You can build in a failsafe, however, by passing the session ID from script to script embedded in a query string. PHP makes a name/value pair available in a

constant called SID if a cookie value for a session ID cannot be found. You can add this string to any HTML links in session-enabled pages:

```
<a href="anotherpage.html"?<?php print SID; ?>">Another  
page</a> will reach the browser as
```

```
<a href="anotherpage.html? PHPSESSID=08ecedf79fe34561fa82591401a01da1">
```

```
Another page </a>
```

The session ID passed ...

5. What is the purpose of Sessions in an Environment with Registered Users?

Working with Registered Users

Suppose we have created an online community, or a portal, or some other type of application that users can join. The process usually involves a registration form, where the user creates a username and password and completes an identification profile. From that point forward, each time a registered user logs into the system. We can grab the user's identification information and store it in the user's session.

Working with User Preferences

In some applications such as design phase of a user-based application, we might build a system in which registered users can set specific preferences that affect the way we view our site. For example, we might allow users to select from a predetermined color scheme, font type and size and so forth. Or, we might allow users to turn off or on the visibility of certain content groupings.

We can store each of those functional elements in a session. When the user logs in, the application loads all relevant values into the user's session and reacts accordingly for each subsequently requested page. If the user changes any preferences while logged in, simply replace the value stored in the `$_SESSION` superglobal with the new selection.

Unit-IV

Working with Files and Directories

1. Write about the including external files with include().

The include() function enables to incorporate other files into PHP document. When we have the same thing (data, headings etc.) in more pages we have to paste it into every document that needs to use it. The include() statement save us from such a chore.

The include() statement requires a single argument, a relative path to the file to include. The include () statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website. One of the most popular features of PHP is the ability to include files. This allows you to separate your code into multiple files, and re-use them in various places. For instance, you can create a file with the functions you use a lot, and then include it when needed, or you can use it to keep your static markup in one place, for easy updating of your site.

File1.php

```
<?php
```

```
Echo "This is File-1. I am including another file below.  
<br>"; include("file2.php");
```

```
?>
```

File2.php

```
<?php
```

```
Echo "I have been included!!!;
```

```
?>
```

Returning a value from an included document.

Included file in PHP can return a value in the same way that functions do. As in a function, using the return statement ends the execution of code within the included file.

File1.php

```
<?php
```

```
Echo "This is File-1. I am including another file below.  
<br>"; $res = include("file2.php");
```

```
?>
```

File2.php

```
<?php
```

```
$rval = (4+5);  
Return $rval;
```

```
?>
```

2. Write about the include_once() function in PHP.

One problem caused by using several different libraries of code within your code is the danger of calling include() twice on the same file. This sometimes happens in larger

projects when different library files call include() on a common file. Including the same file twice often results in repeated declarations of functions and classes, thereby causing the PHP engine great problem.

This can be remedied by using the include_once() function in place of include() function. It requires the path to an include file and otherwise behaves the same way as the include() function the first time it's called.

3. Explain the process of Validating Files.

Before working with a file or directory within our code, it's often good practice to check whether the file or directory exists or not. If exists check its status etc. PHP provides many functions to help us to discover the information about files on the system.

Checking the existence

file_exists():

The file_exists() function checks whether or not a file or directory exists. This function returns TRUE if the file or directory exists, otherwise it returns FALSE.

Syntax: file_exists(path)

```
<?php
echo file_exists("test.txt");
?>
```

is_file(): The is_file() function checks whether the specified entity is a regular file. This function returns TRUE if it is a file.

Syntax: is_file(file)

```
<?php
$file = "test.txt";
if(is_file($file))
{
    echo ("$file is a regular file");
}
?>
```

is_dir(): The is_dir() function checks whether the specified entity is a directory or folder. This function returns TRUE if it is a directory.

Syntax: is_dir(file)

```
<?php
$d = "bcom3";
if(is_dir($d))
{
    echo ("$d is a directory ");
}
?>
```

Checking the status of a file

is_readable(): The is_readable() function checks whether the specified file is readable. This function returns TRUE if the file is readable.

Syntax: is_readable(file)

```
<?php
$file = "test.txt";
if(is_readable($file))
{
    echo ("$file is readable");
}
else
{
    echo ("$file is not readable");
}
?>
```

file_size(): The filesize() function returns the size of the specified file. This function returns the file size in bytes on success or FALSE on failure.

Syntax: filesize(filename)

```
<?php
echo filesize("test.txt");
?>
```

4. Write about various functions available in PHP for creating and deleting Files.**Creating File:**

If a file does not exist, we can create an empty file with the touch() function by giving a string representing a file path. If the file already exists, its contents won't be disturbed, but the modification date will be updated to reflect the time at which the function executed.

```
touch("myfile.txt");
```

We can remove an existing file with the unlink() function by specifying the file path.

```
unlink("myfile.txt");
```

Create files: fopen() is used for creating files. We need to pass the name of file it needs to open and the permission (read, write to that file).

Syntax: \$fp = fopen (string \$filename, string \$ mode);

Modes	Description
R	Open a file for read only. File pointer starts at the beginning of the file
W	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
A	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
X	Creates a new file for write only. Returns FALSE and an error if file already exists

r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

Note: We should remember to close the opened file when it finish by calling `fclose()` which requires the file resource returned from `fopen()` as follows: `fclose($fp);`

Example: `$sample = fopen($sample, 'w') or die("can't open file");`

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.

Syntax: `unlink (string $filename);`

Example: `unlink ($samplefile);`

5. Write short notes to Reading Files.

fread(): The `fread()` function reads from an open file. The first parameter of `fread()` contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

```
fread($myfile,filesize("webdictionary.txt"));
```

fgets(): The `fgets()` function is used to read a single line from a file. The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Check End-Of-File - feof(): The `feof()` function checks if the "end-of-file" (EOF) has been reached. The `feof()` function is useful for looping through data of unknown length. The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

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



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

Read Single Character - fgetc(): The fgetc() function is used to read a single character from a file. The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

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

6. Explain about Writing to a file with fwrite() or fputs():

To write to a file, we need to open the file in write mode ("w") or in append mode ("a").

```
fopen("test.txt", "w");  
fopen("test.txt", "a");
```

The **fwrite()** function accepts a file resource and a string, and then writes the string to the file. The **fputs()** function also work as same as fwrite().

```
fwrite($fp, "Hello World");  
fpus($fp, "Hellow World");
```

```
<?php  
$fname = "test.txt";  
echo "<p> Writing to " . $fname . " ...</p>";  
$fp = fopen($fname, "w") or die("Couldn't open $fname");  
fwrite($fp, "Hello World");  
fclose($fp);  
  
echo "<p> Appending to " . $fname . " ....</p>";  
$fp = fopen($fname, "a") or die("Couldn't open $fname");  
fputs($fp, "And another string");  
fclose($fp);  
?>
```

7. Explain about working with directories in PHP.

PHP provides many functions to work with directories such as create, remove and read from.

Creating directory: The **mkdir()** function enables you to create a directory. The mkdir() function requires a string that represents the path to the directory to create and an octal number integer that represents the mode you want to set for the directory.

```
mkdir("testdir", 0777); // global read/write/execute permissions
```

Removing directory: The `rmdir()` function enables to remove a directory from the file system if the directory is empty. The `rmdir()` function requires only a string representing the path to the directory you want to delete.

```
rmdir("testdir");
```

Opening a directory: Before to read the contents of a directory, we must first open a directory. The `opendir()` function with a string represents the path to the directory do this.

```
$dn = opendir("testdir");
```

Reading the contents of a directory: The `readdir()` function is used to read the file or directory name from a directory. The `readdir()` function requires a directory to handle and returns a string containing the file(or) directory name.

```
<?php
$dn = ".";
$dh = opendir($dn) or die("Couldn't open
directory"); while(!($file=readdir($dh)) == false)
{ if(is_dir("$dn/$file")){
    echo "(D)";
}
echo $file."<br>";
}
closedir($dh);
?>
```

8. Explain the Opening Pipes to and from Processes in PHP. (or) Write about `popen()` function in PHP.

The `popen()` function opens a pipe to the program specified in the command parameter. You can open a pipe to a process using the `popen()` function. The `popen()` function is used like this: `$fp = popen("command", mode);`

The mode is either r (read) or w (write).

```
<?php
$file = popen("/bin/ls","r");
//some code to be executed
pclose($file);
?>
```

9. How to run System Commands from PHP?

Despite PHP being a powerful language with many extensions available to handle specialist libraries, you will likely find it helpful to be able to run external programs when necessary, particularly if you run Unix where the OS comes with many more built-in programs.

In PHP there are two important methods to execute programs, and these are `exec()` and `passthru()`. Both of these two take a minimum of one parameter, which is the

name of the program you want to run, but the difference between them is that *exec()* runs the program then sends back the last line outputted from that program as its return value.

The *passthru()* function, on the other hand, runs the program specified and prints out all the output that program generates. Calling *exec()* is usually preferred when the output of your program is irrelevant, whereas *passthru()* automatically prints your output. Here is both in action:

```
<?php
    print exec("uptime");
    passthru("who");
?>
```

Both of those commands are available on standard Unix boxes. Note that as uptime usually returns just one line, it does not matter whether print *exec()* or *passthru()* is used.

If you pass a second and third parameter to *exec()*, the output of the command will be put into parameter two as an array with one line per element, and the return value of the command will be put into parameter three. Similarly, if you pass a second parameter to *passthru()* it will be filled with the return value of the command.

For example:

```
<?php
    exec("dir", $output, $return);
    echo "Dir returned $return, and output:\n";
    var_dump($output);
?>
```

The System() function: The *system()* function is similar to *exec()* function in that it launches an external application, and it utilizes a scalar variable for storing a return value.

`system("/path/to/somecommand", $return_val);`

The *system()* function differs from *exec()* in that it outputs information directly to the browser, without programmatic intervention.

Working with Images

1. How to Draw a New Image?

The basic PHP function used to create a new image is called **ImageCreate()**. Creating an image is not as simple as just calling the function. It is a stepwise process and includes the use of several different PHP functions.

The ImageCreate() function set a canvas for the new image. The following line creates a drawing area with 150 pixels wide by 150 pixels high.

```
$img = ImageCreate(150, 150);
```

With the canvas defined, next define a few colours for use in new image. The colours can be created using ImageColorAllocate() function and RGB values.

```
$white = ImageColorAllocate($img, 255, 255,  
255); $black = ImageColorAllocate($img, 0, 0, 0);  
$red = ImageColorAllocate($img, 255, 0, 0);
```

Drawing Shapes and Lines

ImageEllipse() is used to draw an ellipse.

ImageArc() is used to draw partial ellipse.

ImagePolygon() is used to draw a polygon.

ImageRectangle() is used to draw a rectangle.

ImageLine() is used to draw a line.

For all the above functions, the parameters should be in order of : the canvas, starting x-axis, y-axis coordinates, ending x-axis, y-axis coordinates and finally the colour to be applied.

Example:

```
<?php  
//create the canvas  
$img = ImageCreate(150, 150);  
  
//set up some colors  
$white = ImageColorAllocate($img, 255, 255,  
255); $black = ImageColorAllocate($img, 0, 0, 0);  
$red = ImageColorAllocate($img, 255, 0, 0);  
$green = ImageColorAllocate($img, 0, 255, 0);  
$blue = ImageColorAllocate($img, 0, 0, 255);  
  
//draw some rectangles  
ImageRectangle($img, 15, 15, 55, 85, $red);  
ImageRectangle($img, 55, 85, 125, 135, $white);  
  
// output the image to the browser  
header("Content-Type: image/png");
```

```
ImagePng($img);
```

```
//clean up after yourself  
ImageDestroy($img);  
?>
```

2. How to draw a Pie Chart?

As the process of creating images—define the canvas, define the colors, and then draw and fill. You can use this same sequence of events to expand your scripts to create charts and graphs, using either static or dynamic data for the data points.

First define the canvas, the color of the canvas will be white. We use the **ImageFilledArc()** function, which has several attributes:

- The image identifier
- The partial ellipse centered at x
- The partial ellipse centered at y
- The partial ellipse width
- The partial ellipse height
- The partial ellipse start point
- The partial ellipse end point
- Color
- Style

The arc should be filled with the defined color and should use the **IMG_ARC_PIE** style. The **IMG_ARC_PIE** style is one of several built-in styles used in the display; this one says to create a rounded edge.

Example

```
<?php  
//create the canvas  
$myImage = ImageCreate(300,300);  
  
//set up some colors  
$white = ImageColorAllocate($myImage, 255, 255, 255);  
$red = ImageColorAllocate($myImage, 255, 0, 0);  
$green = ImageColorAllocate($myImage, 0, 255, 0);  
$blue = ImageColorAllocate($myImage, 0, 0, 255);  
  
//draw a pie  
ImageFilledArc($myImage, 100, 100, 200, 150, 0, 90, $red, IMG_ARC_PIE);  
ImageFilledArc($myImage, 100, 100, 200, 150, 90, 180, $green, IMG_ARC_PIE);  
ImageFilledArc($myImage, 100, 100, 200, 150, 180, 360, $blue, IMG_ARC_PIE);  
  
// output the image to the browser  
header("Content-Type: image/png");  
ImagePng($myImage);  
  
//clean up after yourself
```

```
ImageDestroy($myImage);
```

```
?>
```

3. Write the PHP Code for Image creation from User Input.

In addition to creating images from other images and drawing images on our own, we can also create images based on the user input.

The following program creates a form and script that asks for user input for a variety of attributes ranging from image size to text and background colors, as well as message string. The `imagestring()` function is used to write a string onto an image.

```
<?php
if(!$_POST){
    //show form
?>

<html>
    <head>
        <title> Image Creation Form </title>
    </head>
    <body>
        <h1> Create an Image </h1>
        <form method="post" >
            <p> <strong> Image Size: </strong> <br>
                Width: <input type="text" name="W" size="5" maxlength="5">
                Height: <input type="text" name="H" size="5" maxlength="5"> </p>
            <p> <strong> Background Color: </strong>
                R: <input type="text" name="b_r" size="3">
                G: <input type="text" name="b_g" size="3">
                B: <input type="text" name="b_b" size="3"> </p>

            <p> <strong> Text Color: </strong>
                R: <input type="text" name="t_r" size="3">
                G: <input type="text" name="t_g" size="3">
                B: <input type="text" name="t_b" size="3"> </p>
            <p> <strong> Text String: </strong>
                <input type="text" name="string" size="35">
            </p> <p> <strong> Font Size: </strong>
                <select name="font_size">
                    <option value="1"> 1 </option>
                    <option value="2"> 2 </option>
                    <option value="3"> 3 </option>
                    <option value="4"> 4 </option>
                    <option value="5"> 5 </option>
                </select> </p>
            <p> <strong> Text Starting Position: </strong>
                X: <input type="text" name="X" size="3">
                Y: <input type="text" name="Y" size="3"> </p>
            <p> <input type="submit" name="submit" value="Create Image"> </p>
```

```
        </form>
    </body>
</html>
<?php
}
else
{
    // create canvas
    $img = ImageCreate($_POST["W"], $_POST["H"]);

    // set up colors
    $background = ImageColorAllocate($img, $_POST["b_r"],
$_POST["b_g"], $_POST["b_b"]);
    $text = ImageColorAllocate($img, $_POST["t_r"], $_POST["t_g"], $_POST["t_b"]);

    //write the string at the top left
    ImageString($img, $_POST["font_size"], $_POST["X"],
$_POST["Y"], $_POST["string"], $text);

    //output the image to the browser
    header("Content-Type: image/png");
    ImagePNG($img);

    //cleanup after
    ImageDestroy($img);
}
?>
```

Unit-V

Interacting with MySQL using PHP

1. Differentiate MySQL and MySQLi.

MySQL and MySQLi are PHP database extensions implemented by using PHP extension framework. PHP database extensions are used to write PHP code for accessing the database. They expose database API to provide interfaces to use database functions.

MySQL extension is deprecated and will not be available in future PHP versions. It is recommended to use the MySQLi extension with PHP 5.5 and above.

MySQL MySQLi Difference

There are too many differences between these PHP database extensions. These differences are based on some factors like performance, library functions, features, benefits, and others.

MySQL	MySQLi
MySQL extension added in PHP version 2.0. and deprecated as of PHP 5.5.0.	MySQLi extension added in PHP 5.5 and will work on MySQL 4.1.3 or above.
Does not support prepared statements.	MySQLi supports prepared statements.
MySQL provides the procedural interface.	MySQLi provides both procedural and object-oriented interface.
MySQL extension does not support stored procedure.	MySQLi supports store procedure.
MySQL extension lags in security and other special features, comparatively.	MySQLi extension is with enhanced security and improved debugging.
Transactions are handled by SQL queries only.	MySQLi supports transactions through API.
Extension directory: ext/mysql.	Extension directory: ext/mysqli.

Note:

Though MySQL extension is deprecated, for backward compatibility it will be available. But do not use if you are starting something new and recommend to migrate the older from mysql to mysqli extension.

Other MySQLi Advantages

- MySQLi function `mysqli_query()` allows to enforce error prone queries and prevents bugs like SQL injection.
- Using MySQLi data fetch, we can get buffered or unbuffered based on server resource size.
- MySQLi API allows executing multiple queries with single expression using `multi_query()` function.

2. How to Connect to MySQL in PHP?

In order to manage the database from PHP we should establish a connection to the MySQL database. This is an extremely important step because if script cannot connect to its database, queries to the database will fail.

A good practice when using databases is to set the username, the password and the database name values at the beginning of the script code. If you need to change them later, it will be an easy task.

```
$username="your_username";$password="your_password";$database="your_database";
```

Next you should connect your PHP script to the database. This can be done with the `mysqli_connect()` function:

```
$ms = mysqli_connect("hostname", "username", "password", "database");
```

Ex: \$ms = mysqli_connect("localhost", "root", "admin", "tdb");

The following example creates a new connection and then tests to see whether an error occurred. If an error occurred prints an error message and uses `mysqli_connect_error()` function to print the message. If no error occurs, prints a message including the host information. Although the connection closes when the script finishes its execution. A good practice is to close the connection explicitly using `mysqli_close()` function.

Example Program

```
<?php
$ms = new mysqli("localhost", "root", "admin", "ksr1");
if(mysqli_connect_errno())
{
    printf("Connection failed : %s\n",
    mysqli_connect_error()); exit();
}
else
    printf("Host information : %s\n", mysqli_get_host_info($ms));
mysqli_close($ms);
?>
```

Output:

Host information : localhost via TCP/IP

3. Explain working with MySQL data in PHP (or) How to insert and retrieve data from database thru PHP.

The `mysqli_query()` function is used for basic inserting, updating, deleting and retrieving data in PHP on the database. For insert, update and delete queries, no additional

scripting is required after the query has been executed because we need not display any result. When using select queries, we have a few options for displaying the data retrieved by the query.

Inserting data with PHP

The INSERT INTO statement is used to add new records to a MySQL table. Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

Sample Table: student

```
create table student(  
rno int,  
sname text,  
course text );
```

```
<?php  
$conn = mysqli_connect("localhost", "root", "admin",  
"tdb"); if ($conn->connect_error) {  
    die("Connection failed: " . $conn->connect_error);  
}  
$sql = "INSERT INTO student VALUES (1, 'abc', 'bsc')";  
if ($conn->query($sql) === TRUE) {  
    echo "New record created successfully";  
} else {  
    echo "Error: " . $sql . "<br>" . $conn->error;  
}  
$conn->close();  
?>
```

Retrieving data with PHP

Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function `mysql_query()`. We have several options to fetch data from MySQL.

The most frequently used option is to use function `mysqli_fetch_array()`. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

The constant `MYSQLI_ASSOC` is used as the second argument to `mysqli_fetch_array()`, so that it returns the row as an associative array. With an associative array you can access the field by using their name instead of using the index.

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

```
<?php  
$ms = mysqli_connect("localhost", "root", "admin",  
"tdb"); if ($ms->connect_error) {  
    die("Connection failed: " . $ms->connect_error);  
}
```

```

$sql = "select * from student";
$res = mysqli_query($ms, $sql);
while($row = mysqli_fetch_array($res, MYSQLI_ASSOC))
{
    echo $row['rno'] . " -- " . $row['sname'] . " -- " . $row['course'];
}
echo "<br> Total No. of rows : " . mysqli_num_rows($res);

$ms->close();
?>

```

Creating an Online Address Book

1. Describe various tables used to create an online address book.

When think of an address book, the obvious fields come to mind: name, address, telephone number, email address. However it may have several entries for one person. May be that person has three phone numbers, or two email addresses. While designing online address book, a set of related tables will alleviate the redundancy and repetition of information.

For our application, we may create the following tables and fields.

TABLE NAME	FIELD NAMES
master_name	Id, date_added, date_modified, f_name, l_name
address	Id, master_id, date_added, date_modified, address, city, state, zipcode, type
telephone	Id, master_id, date_added, date_modified, tel_number, type
fax	Id, master_id, date_added, date_modified, fax_number, type
email	Id, master_id, date_added, date_modified, email, type
personal_notes	Id, master_id, date_added, date_modified, note

Notice the use of date-related fields; each table has a date_added and date_modified field in it. The fields will help maintain data, at some point want to issue a query that removes all records older than a certain number of months or years, or removes all records that haven't been updated within a certain period.

```

1. create table master_name(
id int not null primary key auto_increment,
date_added datetime,
date_modified datetime,
f_name varchar(50),
l_name varchar(50) );

```

```

2. Create table address (

```

```
id int not null primary key auto_increment,  
master_id int not null,  
date_added datetime,  
date_modified datetime,  
address varchar(255),  
city varchar(30),  
state char(2),  
zipcode varchar(10),  
type enum('home', 'work', 'other') );
```

3. Create table **telephone** (
id int not null primary key auto_increment,
master_id int not null,
date_added datetime,
date_modified datetime,
tel_number varchar(25),
type enum('home', 'work', 'other'));

4. Create table **fax**(
id int not null primary key auto_increment,
master_id int not null,
date_added datetime,
date_modified datetime,
fax_number varchar(25),
type enum('home', 'work', 'other'));

5. Create table **email**(
id int not null primary key auto_increment,
master_id int not null,
date_added datetime,
date_modified datetime,
email varchar(25),
type enum('home', 'work', 'other'));

6. Create table **personal_notes**(
id int not null primary key auto_increment,
master_id int not null,
date_added datetime,
date_modified datetime,
note text);

2. Create an include file for common functions for Online Address Book.

The only content in the common functions file is the database connection function, this process serves two purposes: to make the scripts more concise and to eliminate the need to modify the database connection information throughout multiple files should that information change.

dbconnection.php

```

<?php
function doDB() {
    Global $mysqli;
    // connect to server and select database; you may need it
    $mysqli = mysqli_connect("localhost", "root", "admin", "testDB");
    // if connection fails, stop script execution
    if( mysqli_connect_errno() ) {
        printf("Connection failed : %s\n", mysqli_connect_error());
        exit();
    }
}
?>

```

3.Creating a Menu

Our online address book will contain several actions, so it makes sense to create a menu for our links. Create a simple menu for all the scripts as follows: <html>

```

<head>
    <title> Address Book </title>
</head>
<body>
    <h1> My Address Book </h1>
    <p> <strong> Management </strong> </p>
    <ul>
        <li> <a href="addentry.php"> Add an Entry </a> </li>
        <li> <a href="delentry.php"> Delete an Entry </a> </li>
    </ul>
    <p> <strong> Viewing </strong> </p>
    <ul>
        <li> <a href="selentry.php"> Select a Record </a> </li>
    </ul>
</body>
</html>

```

4. Write the steps for record addition in your application Online Address Book.

1. Create a file which has two parts:
 - (a) What to do if the form should be displayed.
 - (b) What actions to take if the form is being submitted.
2. In the file include the common functions i.e., for connecting to the database.
3. If there is no value in the \$_POST super global, the user has not submitted the form and therefore needs to see the form to accept the details to be store in address book.
4. If there is a value in \$_POST, means that the user has submitted the form.

Check whether the user has entered the f_name and l_name and redirect the user back to the form if either value is missing.
5. After check for required fields, connects to the database.
6. Write the SQL statements to insert the data into various tables.
7. After successful insertions, display the confirmation message to the user.

```

<?php
include("dbconnection.php");
if(!$_POST){
%>

<form method="post" action="$_SERVER["PHP_SELF"]">
<p> <strong> First/Last Names: </strong> <br>
    <input type="text" name="f_name" size="30" maxlength="75"> <input
    type="text" name="L_name" size="30" maxlength="75"> </p>
<p> <strong> Address: </strong> <br>
    <input type="text" name="address" size="30"> </p>
<p> <strong> City/State/Zip: </strong> <br>
    <input type="text" name="city" size="30" maxlength="50">
    <input type="text" name="state" size="5" maxlength="2">
    <input type="text" name="zipcode" size="10" maxlength="10">
</p> <p> <strong> Address Type: </strong> <br>
    <input type="radio" name="add_type" value="home" checked>
    Home <input type="radio" name="add_type" value="word" > Work
    <input type="radio" name="add_type" value="other" > Other </p>

<p> <strong> Telephone Number: </strong> <br>
    <input type="text" name="tel_number" size="30" maxlength="25">
    <input type="radio" name="tel_type" value="home" checked>
    Home <input type="radio" name="tel_type" value="word" > Work
    <input type="radio" name="tel_type" value="other" > Other </p>

<p> <strong> Fax Number: </strong> <br>
    <input type="text" name="fax_number" size="30" maxlength="25">
    <input type="radio" name="fax_type" value="home" checked>
    Home <input type="radio" name="fax_type" value="word" > Work
    <input type="radio" name="fax_type" value="other" > Other </p>

<p> <strong> Email Address: </strong> <br>
    <input type="text" name="email" size="30" maxlength="150">
    <input type="radio" name="email_type" value="home" checked>
    Home <input type="radio" name="email_type" value="word" > Work
    <input type="radio" name="email_type" value="other" > Other </p>

<p> <strong> Personal Note: </strong> <br>
    <textarea name="note" cols="35" rows="3" wrap="virtual"> </textarea>
</p> <input type="submit" name="submit" value="Add Entry"> </p>
</form>
<%php
}
else if($_POST) {
    if( ($_POST["f_name"]=="") || ($_POST["l_name"]=="") ) {
        header("Location: addentry.php");
        exit;
    }
}

```

```

    }
doDB();

// add to master_name table
$add_master_sql = "insert into master_name(date_added, date_modified, f_name, l_name)
values (now(), now(), '". $_POST["f_name"] . "', '". $_POST["l_name"] . "')";
$add_master_res = mysqli_query($mysqli, $add_master_sql); or die (mysqli_error($mysqli));

// get master_id for use with other tables
$master_id = mysqli_insert_id($mysqli);
// something relevant, so add to the address table
if( ( $_POST["address"] ) || ( $_POST["city"] ) || ( $_POST["state"] ) || ( $_POST["zipcode"] ) ){
$add_sql = "insert into address (master_id, add_added, date_modified, address, city, state,
zipcode, type) values (". $master_id . ", now(), now(), '". $_POST["address"] . "', '".
$_POST["city"] . "', '". $_POST["state"] . "', '". $_POST["zipcode"] . "', '".
$_POST["add_type"] . "')";
$add_res = mysqli_query($mysqli, $add_sql) or
die(mysqli_error($mysqli)); }

// something relevant, so add to the telephone
table if( ( $_POST["tel_number"] ) ){
$tel_sql = "insert into telephone (master_id, add_added, date_modified, tel_number, type)
values (". $master_id . ", now(), now(), '". $_POST["tel_number"] . "', '".
$_POST["tel_type"] . "')";
$add_res = mysqli_query($mysqli, $tel_sql) or die(mysqli_error($mysqli));
}

// something relevant, so add to the fax table
if( ( $_POST["fax_number"] ) ){
$fax_sql = "insert into fax (master_id, add_added, date_modified, fax_number, type) values
( ". $master_id . ", now(), now(), '". $_POST["fax_number"] . "', '".
$_POST["fax_type"] . "')";
$add_res = mysqli_query($mysqli, $fax_sql) or
die(mysqli_error($mysqli)); }

// something relevant, so add to the email
table if( ( $_POST["email"] ) ){
$emai_sql = "insert into email (master_id, add_added, date_modified, email, type) values (".
$master_id . ", now(), now(), '". $_POST["email"] . "', '". $_POST["email_type"] . "')";
$add_res = mysqli_query($mysqli, $emai_sql) or die(mysqli_error($mysqli));
}

// something relevant, so add to the notes table
if( ( $_POST["note"] ) ){
$notes_sql = "insert into personal_notes (master_id, add_added, date_modified, note) values
( ". $master_id . ", now(), now(), '". $_POST["note"] . "')";
$add_res = mysqli_query($mysqli, $notes_sql) or
die(mysqli_error($mysqli)); }

```

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

```
<html>  
  <head>  
    <title> Add an Entry </title>  
  </head>  
  <body>  
    <h1> Add an Entry </h1>  
    <?php echo ""?>  
  </body>  
</html>
```

5. Write the steps for display the details of selected person from Online Address Book.

1. Create a select-and-view script to view the records.
2. The script has two parts:
 - a) the record selection form for which the details to display and
 - b) code to display the record contents.
3. In the code, include the database connectivity file and call the connection establishment function.
4. If \$_POST has no value, load the first, last names from master_name table into a selection drop-down list.
5. If the form submitted,
 - a) First receive the value from the form.
 - b) Match the value with the ID in the master_table.
 - c) If value does not exist, the user is redirected back to the selection form.
 - d) If the ID present, write the queries to obtain the details from various tables and display them.

6. Write the steps to delete the details of selected person from Online Address Book.

1. Create a select-and-view and delete script to view the records.
2. The script has two parts:
 - a) the record selection form for which the details to delete and
 - b) code to delete the records.
3. In the code, include the database connectivity file and call the connection establishment function.
4. If \$_POST has no value, load the first, last names from master_name table into a selection drop-down list.
5. If the form submitted,
 - a) First receive the value from the form.
 - b) Match the value with the ID in the master_table.
 - c) If value does not exist, the user is redirected back to the selection form.
 - d) If the ID present, write the queries to delete the records from various tables.