digit
January 2009

Fast Track to PHP

Fast Track to PHP

By Team Digit

Credits

The People Behind This Book

EDITORIAL

Editor-in-chief	**Edward Henning**
Assistant Editor	**Robert Sovereign-Smith**
Writers	**Santanu Mukherjee, Supratim Bose, Nilay Agambagis Soumita Mukherjee, Ranita Mondal**

Resource

Development	**Runa Chowdhury**

DESIGN AND LAYOUT

Layout Design	**Vijay Padaya, U Ravindranadhan**
Cover Design	**Rohit Chandwaskar**

January 2009
Free with Digit. Not to be sold separately. If you have paid separately for this book, please e-mail the editor at **editor@thinkdigit.com** along with details of location of purchase, for appropriate action.

CONTENTS

Introduction to PHP

PHP, an acronym for Hypertext Preprocessor, is an HTML embedded scripting language. As a general purpose language, it is used for web development and HTML. Apart from being one of the potent, server-side scripting languages, PHP is also used for creating interactive and dynamic web sites. The basic syntax of PHP is similar to Perl and C. Due to these similarities, PHP is often used with the Apache web server on various operating systems.

In addition to supporting the Internet Server Application Programming Interface (ISAPI), PHP is also used with IIS on Windows. Originally designed for creating dynamic web pages, it has developed as a popular medium for innovative web applications and has helped in developing some command line interface mediums. Rasmus Lerdorf is credited with creating PHP in 1995. Usually, PHP runs on a web server and is available on different operating systems and platforms. Its ability to run on most web servers and operating systems for free has added to its demand and esteem.

PHP has the following benefits:

- Creates advanced user experience based on resources already collected
- Quick solution for large and advanced web sites
- Convenient solutions for e-commerce
- Diverse scopes for creating online tools

According to recent statistical data, PHP is installed on more than 20 million web sites and around 1 million web servers. The source code for PHP is distributed under a license, thus making it easily accessible to users.

1.1. Installation of PHP

There are two ways for installation with Windows. This can be done either manually or by an installer. You require PHP, a web browser and a web server.

Normally, there are three common ways of using PHP:

- Desktop (GUI) applications
- Web sites and web applications (server-side scripting), and
- Command line scripting

Once you write the PHP script, the rest of the work is simple. Assuming that you already have a web browser, and depending on your operating system setup, you either have a web server for webspace, or you can rent this from a hosting company. After following these simple steps, you can simply upload your PHP code on your rented server and see the results in your browser. PHP can be compiled from the original source code if Microsoft Visual Studio is already installed on the local drive.

PHP installation can be categorised into two parts:

Windows installer: Here you use the MSI technology of the Wix Toolkit. In this method, PHP is installed and configured with PECL and other built-in extensions. The Windows Installer also configures many other popular web servers like Apache, Xitami and IIS.

Normal install: While running the MSI installer, follow the instructions in the installation wizard.

While in some servers PHP has a direct module interface (such as Microsoft Internet Information Server, iPlanet servers, Apache and Netscape) many other servers support ISAPI. However, PHP does not have a module support for web servers. You can host PHP application with the CGI or Fast CGI processor.

It is often used for command-line scripting with the command line executable. PHP is an advanced scripting language and can be used to write desktop GUI applications, making use of the PHP-GTK extension. Writing desktop GUI applications is different from creating web pages. In a GUI application, PHP manages windows and objects, but no HTML is produced.

Silent install: It method is helpful for system administrators. The following command is useful during the installation of PHP in silent mode:

```
msiexec.exe /i php-VERSION-win32-install.msi /q
Another useful formula to install PHP is:
```

```
msiexec.exe /i php-VERSION-win32-install.msi /q
INSTALLDIR=e: \php
```

The same command can also be used to target the 'Apache Configuration Directory' (APACHEDIR), 'the Sambar Server directory' (SAMBARDIR), or the 'Xitami Server directory' (XITAMIDIR).

Some other installation features can also be presented to install the mysqli extension and the CGI executable. See the following command:

```
msiexec.exe /i php-VERSION-win32-install.msi /q
ADDLOCAL=cgi,ext_php_mysqli
```

Some other features for installation with PHP have been developed in recent times. These are mentioned below:
MainExecutable - php.exe executable
ScriptExecutable - php-win.exe executable
ext_php_* - various extensions (e.g.: ext_php_mysql for MySQL)
apache13 - Apache 1.3 module
apache20 - Apache 2.0 module
apache22 - Apache 2, 2 module
apacheCGI - Apache CGI executable
iis4ISAPI - IIS ISAPI module
iis4CGI - IIS CGI executable
NSAPI - Sun/iPlanet/Netscape server module
Xitami - Xitami CGI executable
Sambar - Sambar Server ISAPI module
CGI - php-cgi.exe executable
PEAR - PEAR installer

Manual - PHP Manual in CHM Format

While upgrading, you need to run the installer either in a graphic pattern or perform the entire task manually. The upgraded install method offers the installer a new installation option to uninstall the old installation in lieu of the new one.

Manual installation steps: With the help of this guide, you can manually install and configure PHP with a web server on Microsoft Windows. To begin the manual installation, download the ZIP binary distribution. Before entering any server specific instructions, follow these steps:

To begin with, place the distribution file in a directory. Follow specific installation methods for installing different PHP versions. To install PHP 4, extract the files to `C:\` and then the ZIP file will expand into a folder named php-4.3.7-Win32. Similarly, if you are installing PHP 5, extract the file to C:\php as the zip file does not expand in the same way as PHP4.

Installing PHP on your operating system helps in many ways. It can perform any task a CGI program can do. Like a CGI, PHP generates dynamic page content, collect form data, and even sends and accepts cookies. Moreover, as an embedded scripting language, PHP has certain other utilities like:

- Used for server-side scripting
- Writing client-side GUI applications, and
- Used for Command line scripting

Once installed on your Windows system, you can download some extensions for additional functionality.

1.2 Basics of PHP

Apart from a simple text editor, no other additional software is required to create PHP code.

First of all in order to print the information about PHP on your server type the following code into the text editor installed on your computer.

```
<?
phpinfo();
?>
```

This single line of PHP code phpinfo, commands the server to print a standard information table. This provides information about the server setup. In the above example, the line ends with a semicolon and should not be missed, or else you will get an error.

Save this script as phpinfo.php, and upload it to the server. Access the URL of the script with your browser. If you can access the script, you will see detailed information about PHP on your server.

If the script fails to work and displays a blank page, then it indicates either the code is wrong, or the server does not support this function.

The basic features of PHP can be divided into three major sections:
- Escaping from HTML
- Instruction separation, and
- Comments

Escaping from HTML: Here, a file is analysed and simply parsed until a special tag is reached. The entire text is interpreted as PHP code.

Instruction separation: The instructions in PHP code are separated exactly the same way as in Perl or C. Each statement is terminated with a semicolon. In PHP, the closing tag suggests the end of a statement. The syntax is as follows:

```
<?php
     echo "This is used for  testing";
?>
```

Comments: In PHP, C and C++ style comments are supported along with Unix shell like comments. The comment comes at the end of a line or in a PHP block of code.

Any PHP scripting block begins with <?php and ends with ?>. PHP source code is flexible and can be placed anywhere in a document. Such source code can never be viewed by selecting the View Source option in the browser. The only thing that is apparent while running PHP is the output in HTML format that is created as the PHP scripts are executed on the server before actually sending the outcome to the browser.

You can also begin a scripting block with (<?) and end with (?>). This is just a shortened version. It is always advisable to use the standard form of (<?php) in place of the shortened form (<?) as the former is clearer and generally supported.

As in an HTML file, PHP files also have HTML tags in addition to some PHP script code. Some important examples are given as

below using the text string "Hello World" and sending it to the browser.

```
<html>
<body><?php
echo "Welcome";
?></body>
</html>
```

Here, each line of PHP code ends with a semicolon. This semicolon actually acts as a separator between two sets of instructions. Echo and Print are the two basic statements available to output text with PHP.

PHP comments: While a single line comment in PHP is made by using//, a large block comment is indicated by /* and */. The following example will make this clear:

```
<html>
<body><?php//Here is a comment/*
It is
inserted in the
block
*/?></body>
</html>
```

Since, PHP scripts are basically embedded in an HTML document, you have the freedom to shift between HTML and PHP.

PHP has several important benefits such as:

- It is not restricted to HTML output
- It provides cross-platform functionality
- PHP converses with several network protocols
- It is compatible with a wide variety of databases
- Strong text processing facilities are available
- It supports most current web servers

PHP can be used effectively on different operating systems such as Linux, Microsoft Windows, many Unix variants (like Solaris, HP-UX and OpenBSD), Mac OS X, RISC OS, and many others. As discussed earlier, in the present scenario, PHP supports most web servers. It works as a CGI processor in servers support-

ing the CGI standard. Each PHP script remains enclosed between two PHP tags commanding the server to recognise the information as PHP.

As PHP is a server-side language, its scripts only run on the operating web server. They never run in the user's browser. With PHP installed in your computer, you can use both procedural programming and object oriented programming (OOP). In some recent PHP versions, not every OOP feature is mentioned. Some code libraries and large applications have been written by using just the OOP codes.

Let's see how we can declare some PHP code:
We can declare PHP code in three different forms:

```
" <?
PHP Here we insert PHP codes
?>

" <?php
PHP Here we insert PHP codes
php?>

" <script language="php">
PHP Here we insert PHP codes
</script>
```

One of the essential features of PHP is that it supports several databases and so you can write database-enabled web pages. PHP supports the following databases:

Adabas D Ingres Oracle (OCI7 and OCI8)
dBase InterBase Ovrimos
Empress FrontBase PostgreSQL
FilePro (read-only) mySQL Solid
Hyperwave Direct MS-SQL Sybase
IBM DB2 MySQL Velocis
Informix ODBC Unix dbm

PHP also supports certain other services such as SNMP, LDAP, POP3, IMAP, HTTP, COM (on Windows), NNTP and many others. The language has certain useful text processing features from the

POSIX Extended or Perl regular expressions to analysing XML documents. Some of the important functions performed here include CyberMUT, Cybercash payment, VeriSign Payflow Pro and CCVS functions commendable to use for online financial programs.

1.3 Combining HTML and PHP

PHP and HTML are closely related to each other and often function together. PHP generates HTML and this HTML passes information to PHP. The two together are seen in the following PHP script which includes HTML:

```
1: <!DOCTYPE html PUBLIC
 2:    "-//W3C//DTD XHTML 1.0 Strict//EN"
  3:   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
 4: <html>
 5: <head>
 6: <title>Listing 3.2 A PHP Script Including
HTML</title>
 7: </head>
 8: <body>
 9: <div><b>
10: <?php
11: print "hello world";
12: ?>
13: </b></div>
14: </body>
15: </html>
```

Let's discuss how to make a form in HTML and fill the form with data using PHP. There are several methods in PHP through which you can make a form and prepare data for that form. Begin your task by making an HTML form to retrieve a user date. PHP and HTML, both are often combined to make bullets, check boxes, input fields and text fields. Both can be used for making a form and collecting personal details of users.

Input Fields:

The input fields in HTML are easily understandable. The following code would present a simple HTML form document.

Code:

```
<html>
<head>
<title>individual information</title>
</head>
<body>
<form method="post" action="<?php echo $PHP_TUTO-
RIAL;?>">
First Name:<input type="text" size="10"
maxlength="10" name="Fnaming">:<br />
Last Name:<input type="text" size="10"
maxlength="46" name="Lnaming">:<br />
```

Radios and checkboxes:

PHP also plays a significant role while making radio buttons and checkboxes in HTML. The radio buttons have associated with them value attributes and the text equated to this value is displayed by the browser when the variable is set in PHP.

The check boxes in the HTML format are made by using arrays. With PHP, the check boxes are placed in an array specified by brackets at the end.

Code:

```
...
Sex::<br />
Male:<input type="radio" value="Male"
name="Sex">:<br />
Female:<input type="radio" value="Female"
name="Sex">:<br />
Please choose type of dwelling::<br />
Steak:<input type="checkbox" value="Steak"
name="food[]">:<br />
Sandwich:<input type="checkbox" value="Sandwich"
name="food[]">:<br />
Fowl:<input type="checkbox" value="Fowl"
name="food[]">:<br />
```

Text areas:

In PHP, the text areas are input fields. These are handled according to the wrap attribute setting. The following code will illustrate this:

```
   ...
   <textarea rows="4" cols="10" name="quotation"
wrap="physic">Insert your preferred
quotation!</textarea>:<br />
```

Dropdown lists and selection lists:

Dropdown lists and selection lists are similar to radio buttons and checkbox selections. While naming a selection form, the name attributes should always be at the beginning. After this put the appropriate values in the options that are grouped under the select tag.

Code:

```
Select a Level of Learning:<br />
<select name="Learning">
<option value="Graduation">Graduation</option>
<option   value="PostGraduation">PostGraduation
</option>
<option    value="Institute">Institute</option>
</select>:<br />
Select your favourite time of Daytime:<br />
<select name="TofD" size="3">
<option value=" Day ">Dawn</option>
<option value="Daytime">Daytime</option>
<option value="Dark">Dark</option></select>:<br />
Always check the code and look for bugs and errors
while going through each of the names. The fol-
lowing code will make the indication clear.
" http://www.indusnetacademy.com/store/">
<html>
<head>
 <title>A PHP Script Including HTML</title>
  </head>
  <body>
  <div><b>
<?php
print "Welcome";
?>
 </b></div>
 </body>
 </html>
```

Fundamentals of PHP

2.1 Variables:

A variable can hold a value that can be changed during the course of the execution of the script. The values can either be explicitly changed or by performing some operation on it. They are similar to variables you used back in school Algebra.

Let's look at the syntax of a PHP variable:

```
$variable_name = Value;
Example:
<?php
$learning = "Learning Variable!";
$x_numeral = 8;
$first_name = 'John';
$lastName = 'Denver';
$nextNumeral = 16;
?>
```

Here, we have inserted a variable name and set the value as per our need. The second line of the program with '$learning' variable ends with a semicolon sign (;) to mark the closing of the statement "Learning Variable". The quotation mark (" "), inserted in the second line, is not used in the second variable ('$x_numeral) as it is an integer.

PHP is a case sensitive programming language. Here '$x_numeral' variable differs from the variable '$X_numeral' because of the use of small 'x' in the first variable and capital 'X' in the second variable. The dollar sign ($) at the beginning of the variable is important as it is exclusively used in PHP. It instructs the PHP engine that the inserted code with this dollar sign is a variable. A variable in PHP always starts with an underscore '_' or with a letter (x, X). You cannot begin a variable with a number. It is a common practice to separate variable names with underscore

(as in $first_name) when two or more words are used to name them, or by converting the first letter of each word to uppercase (as in $lastName). This is done simply to make the name more readable.

Now let us look at the various types of variables:

- **Integer:** These are whole number(s) (-9, 9, 99, 999, etc).
- **Double/Float:** These are floating point numbers or real numbers (0.99, 99.0, etc).
- **String:** These are strings of characters ("Learn", "Java and JavaScript"). This type of variable holds both words and sentences.

Boolean: This holds only two types of data: True and False.

Array: This type of variable holds a list of items.

($student = array("class", "section", "roll");

Object: This is an instance of a class.

The variable can be accessed from within the 'Variable Scope' where it was defined. A variable cannot be accessed if it was defined in a completely different scope. There are three types of Variable Scope: Superglobal, Global and Function.

Superglobal: The Superglobal variable is a type of pre-defined array in PHP. These variables can be accessed from every section of the code.

Global: Global variables can be viewed throughout the script if these are declared in it.

Function: Local variables are declared in a function scope. These variables are exclusively 'local' to the function in which they are declared. These variables cannot be accessed from outside the Function where it was defined.

In PHP, there are some in-built functions that can check the authenticity of a variable. The function 'isset()' is used to check the existence of a variable. It returns Boolean result (True or False). To remove a variable from the memory, the 'unset()' function is used. The 'empty()' function is used to check whether a variable has been defined and holds a non-empty value.

2.2 Data Types:

'Data Types' can be divided into two groups: 'Core Data Type' and 'Special Data Type'.

The 'Core data type' group includes Integer, Float/Double, String and Boolean. The 'Special data types includes Null, Array, Object and Resources.

Integers: As discussed earlier, Integers are whole numbers. It does not include precision. Negative values are also regarded as Integers.

Example: -32, 32, 986, 1245, etc.

Floating-Point Number or Double: Fractional numbers are grouped as Floating point numbers or Double data type. Simply put, Double variables hold numbers with decimal points.

Example: 123.56, 5.6, etc.

The syntax of the 'Floating-Point Number or Double is as follows:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Previously, a different syntax was used to define the 'Floating Point Number or Double. Now let's look at that syntax:

```
LNUM            [0-9]+
DNUM           ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-
9]*)
EXPONENT_DNUM (({LNUM} | {DNUM}) [eE][+-]? {LNUM})
Example:
<?php
//TRIAL 1
$numeral = 216897510871;
$original_numeral = $numeral;

$numeral *= 11123.74;
$numeral /= 11123.74;

if ($original_numeral == $numeral) {
echo "Test 1: Value Equivalent<br />";
}
else {
echo "Test 1: Value NOT Equivalent<br />";
}
```

```
//TRIAL 2
$numeral = 216897510871;
$original_numeral = $numeral;

$numeral *= 11123.75;
$numeral /= 11123.75;

if ($original_numeral == $numeral) {
echo "Test 2: Value Equivalent<br />";
}
else {
echo "Test 2: Value NOT Equivalent<br />";
}
?>
```

The output is as follows:

```
Test 1: Value Equivalent
Test 2: Value NOT Equivalent
```

String Literals:

We have already discussed that Strings hold both words and sentences. These are always inserted within quotation marks. If it starts with a single quotation mark then it must end with the same. If a single quotation is inserted at the beginning of a string then it can not be closed with a double quotation mark. If the quotation marks are inserted in a code without any characters, then it will be treated as 'Null' string. A numeric character is treated as a string if it is inserted within quotation marks. For example, if the number 9 is inserted in a PHP code, then it will be treated as a number. On the other hand, if 9 is inserted in a PHP code, then it will be treated as a string.

Example:

```
"It is an example of a string with double quotes"
'It is an example of a string with single quotes'
"It is also an example of 'a string' where the
single quote will be ignored"
'It is an example of "a string" where the double
quote will be ignored'
"4"
" " (Null string)
```

'Here-docs'

'Here-docs' or 'Here Documents' is a special type of quoting that enables you to quote a large block of text within a script. Here, multiple print statements and quotation marks are not used. The PHP engine treats this block of code as a double quoted statement. 'Here-docs' is extremely useful when a large block of HTML is used in a PHP script. 'Here-docs' usually begin and end with a delimiter word, a series of one or more characters that mark the border between various sections in a data system. Numeric characters, letters and underscores can be inserted in the delimiters. In PHP, delimiters are written in capital letters. Three less than signs (<<<) are inserted at the beginning of a delimiter (Example, <<<HEREDOCS). Look at the following example:

Example:

```
print <<<LEARNING_HEREDOC_EXAMPLE
<text here>
...
< more text>
…

LEARNING_HEREDOC_EXAMPLE
```

Let's look at the following example for better understanding:

Example:

```
<?php
$string = <<<EOD

Example of PHP heredoc stringing
across multiple lines of stringing
learning through example of using heredoc syntax.
EOD;

   /* additional compound example, with variables.
*/
   class foo
   {
        var $foo;
        var $bar;

        function foo()
```

```
        {
            $this->foo = 'Foo';
                $this->bar = array('Bar1', 'Bar2',
'Bar3');
        }
    }

    $foo = new foo();
    $forename = 'Jack';

    echo <<<EOT
    My forename is "$forename". I am printing $foo-
>foo.
    Now, I am printing {$foo->bar[1]}.
    This should print a capital 'A': \x41
    EOT;
    ?>
```

The output of the above program is as follows:
My forename is "Jack". I am printing some Foo.
Now, I am printing some Bar2.
This should print a capital 'A': A

Now-docs

Now-doc is similar to Here-doc, and the only difference is that it is a single quoted string while here-doc is a double quoted string.

Example:

```
<?php
$string = <<<'EOD'
Another example of stringing
across compound lines
by using nowdoc syntax.
EOD;

/* More compound example, with variables. */
class foo
{
    public $foo;
    public $bar;

    function foo()
```

```
        {
            $this->foo = 'Foo';
                $this->bar = array('Bar1', 'Bar2',
'Bar3');
        }
    }

    $foo = new foo();
    $forename = 'MyForename';

    echo <<<'EOT'
    My forename is "$forename". I am printing some
$foo->foo.
    Now, I am printing some {$foo->bar[1]}.
    This should not print a capital 'A': x41
    EOT;
    ?>
```

The output of the above program is as follows:
My forename is "$forename". I am printing some $foo->foo.
Now, I am printing some {$foo->bar[1]}.
This should not print a capital 'A': \x41

Escape Sequences:
In PHP, a single character, headed by a back slash (\), is an Escape character. The HTML <pre> tag is used to display escape sequences in the user's browser. The PHP engine cannot interpret the escape sequences without using the HTML<pre> tag.

Escape Sequences	Functions
\"	Used to print the next character as a double quote, not as a string closer
\'	Used to print the next character as a single quote, not a string closer
\n	Used to print a new line character (remember our print statements?)
\t	Used to print a tab character
\r	Used to print a carriage return (not used very often)
\$	Used to print the next character as a dollar, not as part of a variable

\\	Used to print the next character as a backslash, not an escape character

Example:

```
<?php
    $ExampleString = "It is an \"escpd\" string";
     $ExampleSingularString = 'It \'will\' act';
       $ExampleNonVariable = "I have \$zilch in
Example pocket";
        $ExampleNewline = "It ends with a line
return\n";
                                $ExampleFile =
"c:\\windows\\system32\\Examplefile.txt";
?>
```

Boolean literals:

Boolean literals return only two values: true and false. As discussed, PHP is a case sensitive programming language. You can only use the defined set of Boolean values like, yes/no, on/off, 1/0, etc.

Look at the syntax of the Boolean literals:

```
<?php
$foo = True; // assign the value TRUE to $foo
?>
Example:
<?php
// == is an operator which test
// equality and returns a Boolean value
if ($action == "display_version") {
echo "The version is 1.23";
}
// this is not necessary...
if ($display_dividers == TRUE) {
echo "<hr>\n";
}
// ...as you can simply type
if ($display_dividers) {
echo "<hr>\n";
}
?>
```

Null:
Null variable is assigned the 'NULL' value. A PHP engine will consider a variable as NULL if you do not set it.
Array:
The Array variables hold multiple values. We will discuss 'Array' in detail in the 'Arrays in PHP' section (Unit: 03).
Object:
Objects are used while working with the OOPs (Object Oriented Programming Language). We will discuss object in detail in the 'Object Orientation in PHP' section.
Resource:
The Resource variables hold references to another external resource like file handler, database object, etc.

2.3 Operators

In PHP, variables and values are performed by Operators, that is, they operate on variables and values in PHP. Look at the following expression:

```
$z = $x + $y;
```

In the above expression x and y are two numbers. It is clear from the above expression that it would add x with y and the sum is z. The plus sign (+) inserted between x and y is an operator (Arithmetic Operator).

Operators used in PHP are categorically grouped in various sections:
1. Assignment Operators
2. Arithmetic Operators
3. Comparison Operators
4. String Operators
5. Combined Operators

Now let's discuss in detail.

" Assignment operators

You can use the Assignment Operator to assign a value to a variable. Often a variable is assigned a value of another variable. In this case assignment operators are used. The equal character (=) is used here. Look at the following expression:

Example:

```
$first_var = 5;
$second_var = $first_var;
```

Here the values of both '$first_var' and '$second_var' variables are assigned the same value i.e. 5.

Arithmetic operators

Look at the various Arithmetic Operators:

Operators	Name	Function
+	Addition	This operator is used to add two values
-	Subtraction	This operator is used to subtract the second value from the first one
*	Multiplication	This operator is used to multiply two values
/	Division	This operator is used to divide the first value by the second value
%	Modulus	This operator is used to divide the first value by the second value and it returns only the remainder

Example 01:

```
$adding = 2 + 4;
$minus = 6 - 2;
$multiply = 5 * 3;
$divide = 15 / 3;
$percent = 5 % 2;
echo "Result adding: 2 + 4 = ".$adding."<br />";
echo "Result minus: 6 - 2 = ".$minus."<br />";
echo "Result multiply: 5 * 3 = ".$multiply."<br />";
echo "Result divide: 15 / 3 = ".$divide."<br />";
echo "Result percent: 5 % 2 = " . $percent
```

The output of the above program is as follows:

```
Result adding: 2 + 4 = 6
Result minus: 6 - 2 = 4
Result multiply: 5 * 3 = 15
Result divide: 15 / 3 = 5
Result percent: 5 % 2 = 1.
```

Comparison operators

The 'Comparison Operators' verify the relationship between a variable and its value. These operators are usually inserted within a conditional statement and it returns boolean values like true and false. Look at the various types of Comparison Operators:

Comparison Operators	Name	Function
= =	Equal	This operator is used to check if the two variables hold equal values.
= = =	Identical	This operator is used to check whether two variables hold equal values and the data type of them are also the same.
! =	Not Equal	This operator is used to check if the two variables hold unequal values.
! = =	Not Identical	This operator is used to check for unequal values and for the different data types.
<	Less than	This operator is used to check if the value of one variable is lesser than that of another.
>	Greater than	This operator is used to check if the value of one variable is greater than that of another.
< =	Less than or Equal to	This operator is used to check if the value of one variable is less than or equal to the value of another variable.
> =	Greater than or Equal to	This operator is used to check if the value of one variable is greater than or equal to the value of another variable.

String operators

There are two types of 'String Operators': the Concatenating Operator ('.') and the Concatenating Assignment Operator ('.='). The Concatenating Operator joins the right and the left string into a single string. The Concatenating Assignment Operators add the argument that is placed on the right side of the equal operator with the argument placed on the left side of the 'equal' operator.

Example:

```
$first_string = "Welcome";
$second_string = " Jack";
$third_string = $first_string . $second_string;
echo $third_string . "!";
```

The output of the above program is as follows:

```
Welcome Jack!
```

Combined operators

As the name suggests, the Combined Operators are the combinations of different types of operators.

Look at the various types of Combined Operators:

Operator	Name	Example
+=	Plus & Equals	$a += 4;
-=	Minus & Equals	$a -= 6;
*=	Multiply & Equals	$a *= 4;
/=	Divide & Equals	$a /= 3;
%=	Modulus & Equals	$a %= 6;
.=	Concatenation & Equals	$example_str.="Welcome";

There are some other types of operators used in PHP. Let's look at those:

Logical operators:

Logical Operators	Functions
And	Checks if two or more statements are true
&&	Same as And
Or	Checks if at least one of two statements is true
||	Same as Or
Xor	Checks if only one of two statements is true
!	Checks if a statement is not true

Increment and decrement operators:

Increment / Decrement Operator	Name	Function
++value	Pre-Increment	This operator adds 1 to the value before processing the expression that can use it.
--value	Pre-Decrement	This operator subtracts 1 from the value before processing the expression that uses the value
value++	Post-Increment	This operator adds 1 to the value after processing the expression by which the value can be used
value--	Post-Decrement	This operator subtracts 1 from the value after processing the expression which uses the value

2.4 Control structure

The 'Control Structure' controls the program flow of PHP. It can also check whether a block of code is executed or not.
The syntax of the 'Control Structure' is as follows:

```
<?php
if (expression) statement
?>
```

Let's look at various types of 'Control Structure':

- if
- elseif/else if
- Alternative syntax for control structures
- while
- do-while
- for
- foreach
- switch
- if: It is used for conditional execution of code fragments. It returns Boolean values (true/false).

Look at the syntax of 'if' Control Structure:

```
if (expr)
statement

Example:
<?php
if ($x > $y) {
     echo "x is bigger than y";
     $y = $x;
}
?>
```

else: If an expression in the 'if' statement returns false, then the 'else' 'Control Structure' is used.

Example:

```
<?php
if ($x > $y) {
     echo "x is bigger than y";
} else {
```

```
    echo "x is NOT bigger than y";
}
?>
```

elseif/else if: It is a combination of 'if' and 'else' Control Structure. If the 'if' Control Structure' returns a 'false' value, then a different statement is executed by using the 'else' Control Structure'.

Example:

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

"Alternative syntax for control structures: There are some alternative syntax for some control structures like, 'if', 'while', 'for', 'foreach' and 'switch'. It changes the opening brace to a colon sign (:) and closing brace to endif;, endwhile;, endfor;, endforeach;, or endswitch.

Example:

```
<?php
if ($x == 6):
    echo "x equals 6";
    echo "...";
elseif ($x == 7):
    echo "x equals 7";
    echo "!!!";
else:
    echo "x is neither 6 nor 7";
endif;
?>
```

while: The 'while' Control Structure executes the nested statements repetitively until the 'while' statement returns a false value. The syntax of 'while' control structure is as follows:

```
while (expr)
statement
```

do-while: It is very much similar to the 'while' Control Structure. The only difference is that here the truth expression is checked at the end of every repetition. Look at the syntax of 'do-while':

```
<?php
$i = 0;
do {
     echo $i;
} while ($i > 0);
?>
```

for: This is one of the complex loops in PHP. The syntax of the 'for' control structure is as follows:

```
for (expr1; expr2; expr3)
statement
```

foreach: This Control Structure is first introduced in PHP. Have a look at its syntax:

```
foreach (array_expression as $value)
statement

foreach (array_expression as $key => $value)
statement
```

switch: This Control Structure is similar to a series of 'if' statements. We will discuss it in detail in our later section.

Arrays in PHP

3. Arrays in PHP

In PHP, arrays are ordered data maps and are used to store, manage and operate a set of variables. To put it simply, an array is a data structure that holds multiple data within a single identifier. There are two parts in an Array - Values and Keys. While Values contain information to be stored, Keys are used to identify these values. It is allocated to a single variable. It holds significant information, popularly termed as Array Elements. This information can be used for a number of times in a program. Either non negative Integers or Strings are used as Keys. The arrays that use non-negative Integers as Keys are termed as Scalar Arrays. These are Associative Arrays that use Strings as keys. An Array may contain different Array(s) popularly known as Multidimensional Arrays.
The syntax of an Array is as follows:

```
$array[key] = value;
Look at the simple example below:
Example:
$student_array[0] = "Rohit";
$student_array[1] = "Rahul";
$student_array[2] = "Sourav";
$student_array[3] = "Abdul";
```

In the above example, the names of the students (Rohit, Rahul, Sourav and Abdul) are the Values and the numeric characters (0, 1, 2 and 3) are the Keys of this array.

Scalar array:
The numeric values are used as 'Keys' in Scalar Array. To put it simply, we can use integers as index numbers in scalar arrays. In case of scalar arrays, keys start from zero (0). Look at the examples below:

Example 1:

```
<?php
$colors = array("red","brown","yellow");
print_r($colors);
```

```
?>
```

The output of this program is as follows:

```
Array ( [0] => red [1] => brown [2] => yellow )
```

Here, multiple values are simultaneously assigned to an array. It is also possible to assign values to an array one by one using keys as shown below:

Example 2:

```
$numerals = array( );
$numerals[]="3";
$numerals[]="9";
$numerals[]="11";
$numerals[]="7";
Associative array:
```

Associative arrays are indexed with strings in lieu of numbers. Look at the example below:

Example:

```
<?php
$marks["Ram"] = 80;
$marks["Raj"] = 60;
$marks["Rahul"] = 50;
$marks["Rajam"] = 0;

echo "Ram Obtained- " . $marks["Ram"] . "<br />";
echo "Raj Obtained- " . $marks["Raj"] . "<br />";
echo "Rahul Obtained- " . $marks["Rahul"] . "<br
/>";
echo "Rajam Obtained- " . $marks["Rajam"];
?>
```

The output of this program is as follows:

```
Ram Obtained- 80
Raj Obtained- 60
Rahul Obtained- 50
Rajam Obtained- 0
asort( ) - This function is used to sort an asso-
ciative array on values.
```

arsort( ) - This function is used to sort an associative array on values in reverse order.

3.1 Creating arrays

The array() language constructor is used to create an array in PHP. Look at the syntax below:

```
array( [key =>] value
    , ...
    )
    // The key may be either an integer or a string
    //The value is any reusable value
```

Let's see how we can create an array by using the array() function:

```
$countries = array ("INDIA", "PAKISTAN", "JAPAN")
```

Here we can access the second element by using the index "1". Let's see:

```
echo "$country[1]";
```

The output of this program is:

"PAKISTAN".

Now let's see how we can create an array by using an array identifier:

```
$countries[] = "INDIA";
$countries[] = "PAKISTAN";
$countries[] = "JAPAN";
```

Here, the values are inserted in the same order as the earlier one. Using the index number, we can place the data as per our requirement. You can insert these index numbers inside the square brackets. Look at the following code:

```
$countries[1] = "PAKISTAN";
$countries[2] = "JAPAN";
$countries[0] = "INDIA";
```

Example 1:

```
<?php
$learn = array(foo => "I am learning how to create an array", 12 => true);
```

```
echo $learn[foo]; // I am learning how to create
an array
echo"<br>";
echo $learn[12]; // Unit 01
?>
```

The output of this program is as follows:

I am learning how to create an array
1

While creating an array, first we denote a variable name as an array:

```
<?php
$learning = array( );
?>
```

Next, specify the array that will hold the specified value. Use a comma to separate the listed values of an array. Look at the PHP code below:

```
<?php
$learning = array(
"Basic",
"Intermediate",
"Advanced"
);
?>
```

Example 2:

```
<?php
// Learning to create array.
$array_learning = array(10, 20, 30, 40, 50);
print_r($array_learning);

// Let's delete every item and we will leave the
array itself intact:
foreach ($array_learning as $i => $value) {
unset($array_learning[$i]);
}
print_r($array_learning);

// Append an item (note that the new key is 50,
instead of 0).
$array_learning[] = 60;
```

```
print_r($array_learning);

// Re-index:
$array_learning = array_values($array_learning);
$array_learning[] = 70;
print_r($array_learning);
?>
```

The output of this program is as follows:
Array ([0] => 10 [1] => 20 [2] => 30 [3] => 40 [4] => 50) Array ( ) Array ([5] => 60) Array ([0] => 60 [1] => 70)

3.2 Multidimensional arrays

Multidimensional Arrays are the most complex arrays in PHP. As the name suggests, these are data structures that hold various other arrays. Various arrays are used as sub-array elements in a Multidimensional Array. We can easily identify the difference between a single dimensional array and a multidimensional array. In a single dimensional array, we set a value for a single key and assign a number of values to several keys. For example, we may assign values like 'class', 'section', 'roll number' to a single dimensional array that contains information on 'Student'. On the other hand, a multidimensional array 'Student' may include the break up like, 'Personal data' 'Marks', 'Attendance,' etc. Each of these sections is a single dimensional array that is treated as a separate array.

Look at the example below:

Example 1:
Here we have created a 'Multidimensional Array' by using the automatically assigned ID keys:

```
$relatives = array
(
"Robin"=>array
(
"Ram",
"Bharat",
"Adam"
),
```

```
    "Raj"=>array
    (
    "Gili"
    ),
    "Brate"=>array
    (
    "Sita",
    "Lorel",
    "Charles"
    )
    );
```

Example 2:

```
    <?php
    $vegetables = array ( "vegetables" => array ( "a"
=> "potato",
    "b" => "banana",
    "c" => "spinach"
    ),
    "numbers" => array ( 1,
    2,
    3,
    4,
    5,
    6
    ),
    "holes" => array ( "first",
    5 => "second",
    "third"
    )
    );
```

// These are some examples to address values in the array that is mentioned above

```
    echo $vegetables["holes"][5]; // prints "second"
    echo $vegetables["vegetables"]["a"]; // prints
"potato"
    unset($vegetables["holes"][0]); // remove "first"
```

// This is developing a new multi-dimensional array

```
    $juices["spinach"]["green"] = "good";
```

```
?>
```

3.3 Navigating arrays

Here we need to know the number of elements while accessing these.

```
" sizeof($arr)
```

In PHP, the sizeof($aar) function returns the number of elements in the array. This can also be used to initialise a loop counter while processing the array.

Example:

```
<?php
$data = array("yellow", "green", "red");

echo "This Array includes " . sizeof($data) . "
elements";
?>
```

The output of this program is as follows:
This Array includes 3 elements.
There is another function count() that also returns the number of elements of an array.

To access the elements of a scalar array, you can use the 'for' statement. For example, the elements of the array in the above example can be displayed by using the 'for' statement. Look at the code below:

```
for($a=0;$a<3;$a++)
echo "<br>".$data[$a];
```

The elements of an associative array cannot be accessed using the 'for' statement. In this case, we can use the 'foreach' statement. The elements of a scalar array are also accessible through this statement. The following example can also be written by using the 'foreach' statement. See the code below:

```
foreach($data as $a)
echo $a;
```

Now let us see how to access the elements of an associative array by using the 'foreach' statement. We can use the 'foreach'

statement in the following way:

```
foreach($array_name as $key=>$val)
```

Example:

```
<?php
$asso_array=array(Roll=>1,
Name=>"Tom",
Grade=>"B");
foreach($asso_array as $abc=>$xyz)
echo $abc."=".$xyz."<br>";
?>
```

The output of this program is:

```
Roll=1
Name=Tom
Grade=B
```

The each($arr) function is used to repetitively navigate to an array. When the each() function is called, it returns the current key as well as the value of the array. Here, the array cursor is also moved forward by one element. This function is popularly used in a loop.

Example:

```
$data = array("Protagonist" => "Jack", "Jam" =>
"Harry");
while (list($key, $value) = each($data)) {
echo "$key: $value \n";
}
?>
```

The output of this program is:
Protagonist: Jack
Jam: Harry

The techniques described above can be used to access the elements of multidimensional array.

3.4 Manipulating Keys

Keys play an important role in an array, especially in an associative array. There are some functions that can manipulate the keys of an array. Some of these functions are given below:

" array_keys($arr)

The array_keys($arr) function is used to recover the keys from an associative array. This function receives a PHP array and a new array is returned. This new array contains only the keys of the array. The complementary part of this function is the array_values() function.

Example:

```
$data = array("Protagonist" => "Jack", "Milliate"
=> "Jill");
print_r(array_keys($data));
?>

The output of this program is as follows:
Array
(
[0] => Protagonist
[1] => Milliate
)
```

3.5 Manipulating arrays

Now let's see how various functions are used in PHP to manipulate arrays:

" array_values($arr)

array_values($arr) function receives a PHP array. It returns a new array that contains only the values of the array and excludes the keys. The array_keys() function is used as the complementary part of the array_values($arr) function. You can use the array_values($arr) function to recover values from an associative array.

Example:

```
<?php
$data = array("Protagonist" => "Jack", "Milliate"
=> "Jill");
```

```
print_r(array_values($data));
?>
```

The output of this program is as follows:

```
Array
(
[0] => Jack
[1] => Jill
)
```

```
" array_pop($arr)
```

The array_pop($arr) function removes an element from the end of an array and returns its value.

Example:

```
<?php
$data = array("Jack", "Milliate", "Mack");
array_pop($data);
print_r($data);
?>
```

The output of this program is as follows:

```
Array
(
[0] => Jack
[1] => Milliate
)
" array_push($arr, $val)
```

The array_push($aar,$val) function inserts an element at the end of an array.

Example:

```
<?php
$data = array("Jack", "Jill", "Dick");
array_push($data, "Harry");
print_r($data);
?>
```

The output of this program is as follows:

```
Array
```

```
(
[0] => Jack
[1] => Jill
[2] => Dick
[3] => Harry
)
" array_shift($arr)
```

The array_shift($aar) function is used to remove an element from the beginning of an array.

Example:

```
<?php
$data = array("Jack", "Jam", "Dick");
array_shift($data);
print_r($data);
?>
```

The output of this program is as follows:

```
Array
(
[0] => Jam
[1] => Dick
)
" array_unshift($arr, $val)
```

The array_unshift($aar,$val) function adds an element to the beginning of an array.

Example:

```
<?php
$data = array("Jack", "Jill", "Dick");
array_unshift($data, "Sana");
print_r($data);
?>
```

The output of this program is:

```
Array
(
[0] => Sana
[1] => Jack
```

```
[2] => Jill
[3] => Dick
)
" sort($arr)
```

The sort($aar) function sorts the elements in an array in a ascending order. In the following example, the values of the elements in an array of characters are arranged in ascending alphabetical order.

Example:

```
<?php
$data = array("b", "d", "a", "c");
sort($data);
print_r($data);
?>
```

The output of this program is as follows:

```
Array
(
[0] => a
[1] => b
[2] => c
[3] => d
)
```

There are some other functions that are used to sort data in a particular order. These are rsort(), asort(), arsort(), ksort(), krsort().

rsort() - Sorts scalar array in reverse order.
asort() - Sorts associative array by values.
arsort() - Sorts associative array by values in reverse order.
ksort( ) - Sorts associative array by 'Keys'.
krsort( ) - Sorts associative array by 'Keys' in reverse order.

```
" array_flip($arr)
```

The array_flip($arr) function interchanges the keys and the values of an Associative array.

Example:

```
<?php
$data = array("x" => "Mangoes", "y" =>
```

```
"Potatoes");
   print_r(array_flip($data));
   ?>

   The output of this program is as follows:
   Array
   (
   [Mangoes] => x
   [Potatoes] => y
   )
```

" array_reverse($arr)

The array_reverse($arr) function reverses the order of the elements in an array.

Example:

```
<?php
$data = array(11, 21, 26, 61);
print_r(array_reverse($data));
?>
```

The output of this program is:

```
Array
(
[0] => 61
[1] => 26
[2] => 21
[3] => 11
)
```

" array_merge($arr)

The array_merge($aar) function merges two or more arrays. This helps to create a merged array. This function is also used to combine multiple data into a single structure.

Example:

```
<?php
$data1 = array("ram", "shyam");
$data2 = array("jack", "jam");
print_r(array_merge($data1, $data2));
?>
```

The output of this program is:

```
Array
(
[0] => ram
[1] => shyam
[2] => jack
[3] => jam
)
" array_rand($arr)
```

The array_rand($arr) function selects one or more than one random elements from an array.

Example:

```
<?php
$data = array("yellow", "pink", "green");
echo "Display the color " .
$data[array_rand($data)];
?>
```

The output of this program is:

Display the color green

```
" array_slice($arr, $offset, $length)
```

The array_slice($aar,$offset,$length) function is useful to extract the elements of an array. It extracts the elements from array offset $offset. This extracting is continued until the array slice $length element is elongated.

Example:

```
<?php
$data = array("pink", "yellow", "green", "red");
print_r(array_slice($data, 1, 2));
?>
```

The output of this program is:

```
Array
(
[0] => yellow
[1] => green
)
" array_unique($data)
```

The array_unique($data) function is used to remove all duplicate entries in the array.

Example:

```
<?php
$data = array(15,15,19,21,33,19);
print_r(array_unique($data));
?>
```

The output of this program is:

```
Array
(
[0] => 15
[3] => 21
[19] => 33
[5] => 19
)
" array_walk($arr, $func)
```

The array_walk($aar, $func) function is used while performing custom processing on the various sections of an array. This function returns an altered array.

Example:

```
<?php
function reduce(&$a, $b) {
$a -= $a * 0.1;
}

$ourarray = array(10,20,30,40);
array_walk($ourarray, 'reduce');
print_r($ourarray);
?>
```

The output of this program is:

```
Array ( [0] => 9 [1] => 18 [2] => 27 [3] => 36 )
```

3.6 Serializing arrays

The serialize() function is used to generate a representation of values that the array holds. This means that it is used to serialise some PHP values. It stores the PHP values without loosing its type and structure. These serialised values can be un-serialised by using the unserialize() function. The serialize() function can not serialise the variables of type 'Resource'.

Example 1:

```
<?php
// $session_data contains a multi-dimensional array with session
// information for the current user. We use serialize() to store
// it in a database at the end of the request.

$con = odbc_connect("webdb", "php", "bird");
$a = odbc_prepare($con,
        "UPDATE sessions SET data = ? WHERE id = ?");
$sqldata = array (serialize($session_data), $_SERVER['PHP_AUTH_USER']);
if (!odbc_execute($a, &$sqldata)) {
     $a = odbc_prepare($con,
       "INSERT INTO sessions (id, data) VALUES(?, ?)");
       if (!odbc_execute($a, &$sqldata)) {
            /* Something went wrong.. */
       }
}
?>
```

Example:

```
<?php
if($_POST[submit]) {
$ourarray = array(); // New, blank array.
foreach($_POST as $key => $a) {
if($key!="submit") { // We want to exclude the submit button
$ourarray[$key] = $a;
}
```

```
    }
    $ourarray = serialize($ourarray); // Serializes
our new array.
    $b = fopen("owndata.txt","r+");
    $write = fwrite($b,$newarray);
    if($write) { // If it works, which it will...
    echo "It worked!";
    }
    else { // In the unlikely event of the plane
crashing...
    }
    }
    ?>
    <?php
    ?>
    <input type="submit" id="submit" name="submit"
value="Update" />
    </form>
```

The output of this program is as follows:

```
    Bottom of Form
    Example (unserialize() function):

    <?php
    // Here, we use unserialize() to load session
data to the
    // $session_data array from the string selected
from a database.
    // This example complements the one described
with serialize().

    $con = odbc_connect("webdb", "php", "bird");
    $a = odbc_prepare($conn, "SELECT data FROM ses-
sions WHERE id = ?");
    $sqldata = array($_SERVER['PHP_AUTH_USER']);
    if (!odbc_execute($a, &$sqldata) ||
!odbc_fetch_into($a, &$tmp)) {
        // if the execute or fetch fails, initialize
to empty array
        $session_data = array();
    } else {
```

```
        // we should now have the serialized data in
$tmp[0].
        $session_data = unserialize($tmp[0]);
        if (!is_array($session_data)) {
             // something went wrong, initialize to
empty array
            $session_data = array();
        }
    }
    ?>
```

Functions in PHP

4. Functions in PHP

In all programming and scripting languages, a function is a block of code that is used repetitively in a program. It saves time while developing a web page. In PHP, the concept of function is the same as in other languages. There are some in-built functions in PHP. Besides that, we can define functions as per our requirements. These are called 'User Defined Functions'.

Look at the elements of a function:

function: all function declarations begin with the word 'function'.

Name of the function: names to a function are usually assigned in accordance with its utility.

Opening and Closing parentheses (()): the opening and closing parentheses are an integral part of a function and you can insert both the opening and closing parentheses together, just after the name of the function. As the dollar sign ($) indicates the existence of a variable, these parentheses indicate the existence of a function.

Opening and Closing curly braces ({}): the opening curly brace ({) indicates the beginning of the function code and the closing curly brace marks the termination of a function.

Example:

```
<html>
<body>
<?php
function DisplayTitle()
{
echo "Learning Function";
}
DisplayTitle();
?>
</body>
</html>
```

In this example, PHP codes are embedded in HTML. Here, we have used a function 'DisplayTitle()'. This function starts with the word function and indicates that the character inserted just after this word is a function. It displays the title of the tutorial. Any one who will go through this will understand the purpose of this function from its name.

Look at a more complex example:

Example:

```
<?php
$makingfoo = true;
/* Here we should note that we can't call foo()
from here
as it doesn't exist yet,
but we can call bar() */
bar();
if ($makingfoo) {
function foo ()
{
echo "This does not exist unless and until the
program execution reaches here.\n";
}
}
/* Here we can safely call foo()
since $makingfoo calculated to true */
if ($makingfoo) foo();
function bar()
{
echo "This does exist immediately upon the start-
ing of the program.\n";
}
?>
```

4.1 User defined function

Let's see how a user can define a function according to his/her need. Look at the syntax of the user defined function:

```
function function_name(){

//statements are inserted here
}
```

Here you can see that the naming convention is the same as that of a variable. The only difference is that we do not use the dollar sign ($). Also, a space is not used while naming the function. If a space is inserted between the two words (as 'function function), then these are interpreted as two different words and returns an error message. You can use an underscore (_) instead of the space between the two words. Assume you want to create a function that randomly produces a password. Let this function be randompassword(). A function cannot be completed without the use of opening and closing parentheses () and the opening and closing braces {}.

Example:

```
<?
function randompassword()
$characters = "abcdefghijklmnopqrstu-
vwxyz123456789";
$password = '';
for($i=0;$i<7;$i++)
{
$password .= $characters{rand() % 40};
}
return $password;
}
//this is to use the function
$password = randompassword();
?>
```

Here, this function creates a random password that includes both letters and numbers. Letters and passwords are inserted in the $characters variable. The randompassword() function is used here for making the password random. The $password variable returns a specific result. At last, the '$password=randompassword()' function is used to run it.

4.2 Function scope

The origin from where a function can be accessed is called the function scope. A function, once declared, can be accessed from any section of a program. A variable scope will be local to a function, if defined within a function. Use the global key word while using a variable defined in the body part of the program.

Example:

```
function MathSum ( )
{
    global $sum = 4 + 4;
 }
$sum = 0
MathSum ( )
print "4 + 4 = ".$sum
```

Here, the global keyword instructs PHP to look for a variable defined outside the function.

4.3 Function arguments and return values

In PHP, the codes are passed by both objects and arrays. We have two models to pass data in a program: 'Pass by value' and 'Pass by reference'.

Pass by value: This indicates passing the variables. Here the variables are sent as an argument to a defined function. The assignment operator is used to assign it to a different variable. The receiving function or the variable gets a copy of the value of the variable. Look at the codes below:

```
$x=5;
$y=$x;
```

Here, both the variables are assigned the value 5. You can change either of these two variables, but the other one will not be affected. The modification that we insert here, are completely local to the function it belongs to. These changes are not reflected outside the function.

Example:

```
<html>
<head>
<title>Passing an Argument tutorial</title>
</head>
<body>
```

```
<?php
function addFour( $num ) {
$num += 4;
}
$originalnum = 40;
addFour( $originalnum );
print( $originalnum );
?>
</body>
</html>
```

The output of this program is as follows:

```
40
```

Pass by Reference:

'Pass by Reference' is a unique feature of OOPs (Object Oriented Programming languages). It creates a new indicator but indicates to the same variable. As the 'Pass by Value' creates a copy of a variable, the 'Pass by Reference' creates a different name of the same variable. The ampersand sign (&) is used while passing by reference. It is always inserted just after an assignment operator. Look at the code below:

```
$x =& $y;
```

Here, the ampersand operator is used to create a reference to the variable $y. Look how this code is executed:

```
$x = (& $y);
```

In order to define arguments to pass to a function, we need to insert a list of names. These names must be inserted within parentheses in the statements of the function. Look at the code below:

```
function Passfunction ($argument1, $argument2)
```

Returning results from a function

In PHP, a function returns a specific result when a code is called. Here the 'return' keyword is used. Look at the following syntax:

```
return $abc;
```

While processing a return statement, the function is terminated. Here the value of the variable $abc is returned to a code that is called. The values must be specified in the variable. Using

one return statement, a single variable can be returned. We must define an array in case of returning more than one variable. Look at the example below:

Example:

```
<html>
<head>
<title>Returning Result from a Function </title>
</head>
<body>
<h1>Display capacity</h1>
<br />
<?php
function CalCapacity ($a, $b, $c)
{
$capacity = $a * $b * $c;
return $capacity;
}
$length = 4;
$width = 5;
$hight = 6;
print "The capacity of this object is " .
CalCapacity($length, $width, $hight) . " capacity
units.";
?>
</body>
</html>
```

In the above example, a number of variables are passed to a function as an argument. Here a result is returned. We are now defining a function 'CalCapacity'. It receives parameters like $a, $b and $c. The function CalCapacity($a,$b,$c) will calculate and determine the capacity of an object. The function will return the result in the code that is called. The variables $length, $width and $height include three values. The print statement will show the result.

Look at another example for better understanding:

```
<?php
function mySum($numA, $numB) {
$total = $numA + $numB;
return $total;
}
```

```
$numericValue = 0;
echo "First numericValue = ". $numericValue ."<br />";
$numericValue = mySum(3, 4); // Here it Stores the result of mySum in $numericValue
echo "Second numericValue = " . $numericValue ."<br />";
?>
```

The output of this program is:

First numericValue = 0

Second numericValue = 7

4.4 Internal function

To ease the development process, PHP provides a large number of in-built functions with some specific extensions. A few of these functions are described in the other sections. Here are some of these Internal Functions:

" imagecreatetruecolor()

The 'imagecreatetruecolor()' function returns an image identifier that represents a black image of the specified size. Look at the syntax of this function:

```
width
Image width
height
Image height
```

It returns an image resource identifier if it is successfully executed. The 'FALSE' value is returned if an error occurs.

Example:

```
<?php
header ("Content-type: image/png");
$im = @imagecreatetruecolor(120, 20)
        or die("Cannot Initialize new GD image stream");
$text_color = imagecolorallocate($im, 233, 14,91);
imagestring($im, 1, 5, 5, "A Simple Text String", $text_color);
imagepng($im);
```

```
imagedestroy($im);
?>
```

" mysql_connect()

The 'mysql_connect()' function is used to connect to a MySQL Database Server. It returns a MySQL link identifier if it is successfully executed. A 'FALSE' value is returned if an error occurs while executing the program.

Look at the examples below:

Example 1:

```
<?php
$con = mysql_connect('localhost','mysql_user',
'mysql_password');
if (!$con) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($con);
?>
```

Example 2:

```
<?php
// we connect to oursite.com and port 3307
$con = mysql_connect('
oursite.com:3307','mysql_user', 'mysql_password');
if (!$con) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($con);

// we connect to localhost at port 3307
$con =
mysql_connect('122.0.0.1:1107','mysql_user',
'mysql_password');
if (!$con) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($con);
?>
```

Example 3:

```
<?php

// we connect to localhost and socket e.g.
/tmp/mysql.sock

//variant 1: ommit localhost

$con = mysql_connect(':/tmp/mysql','mysql_user',
'mysql_password');
if (!$con) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($con);


// variant 2: with localhost
$ c o n
=mysql_connect('localhost:/tmp/mysql.sock','mysql_u
ser', 'mysql_password');
if (!$con) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($con);
?>
" phpinfo()
```

The phinfo() function is extremely helpful for new users. We can get a lot of information on PHP by using this function. We can get the information on the current state of PHP as the output. It returns' TRUE' if the function is executed successfully. It returns 'FALSE' if an error occurs while the execution of the program.

Function	Value	Description of the Function
INFO_GENERAL	1	This function configures line, and php.ini location. It also builds date, Web Server, System and many more.
INFO_CREDITS	2	This function displays the credits of PHP 4.
INFO_CONFIGURATION	4	This function defines the Current,

		Local and Master values for php directives.
INFO_MODULES	8	This function loads modules as well as their respective settings.
INFO_ENVIRONMENT	16	This function defines the Environment Variable information. These information are also available in $_ENV.
INFO_VARIABLES	32	This function displays all the predefined variables from EGPCS (Environment, GET, POST, Cookie and Server).
INFO_LICENSE	64	This function displays the PHP License information.
INFO_ALL	-1	This function displays all of the above characteristics. It is also regarded as the default value of PHP.

Look at the example below:
Example:

```
<?php

// Show all information, defaults to INFO_ALL
phpinfo();

// Show just the module information.
// phpinfo(8) yields identical results.
phpinfo(INFO_MODULES);

?>
" get_loaded_extensions()
```

The 'get_loaded_extensions()' function returns an array that includes the names of all modules. These modules are compiled and loaded in the interpreter of PHP.
Look at the example below:

```
Example:
<?php
print_r(get_loaded_extensions());
?>
```

The output of this program is as follows:

```
Array ( [0] => bcmath [1] => calendar [2] => com_dot-
net [3] => ctype [4] => ftp [5] => iconv [6] => odbc
[7] => pcre [8] => session [9] => SPL [10] => SQLite
[11] => standard [12] => tokenizer [13] => zlib [14]
=> libxml [15] => dom [16] => SimpleXML [17] => wddx
[18] => xml [19] => apache [20] => mbstring [21] =>
mysql [22] => mysqli )
```

" str_replace()

The 'str_replace()' function removes all the events of the search string and inserts the replacement values.

Example:

```
<?php
// Provides: <body text='black'>
$newbody = str_replace("%body%", "black","<body
text='%body%'>");
// Provides: Hll Wrld f PHP
$allvowels = array("a", "e", "i", "o", "u",
"A","E", "I", "O", "U");
$onlyconsonants = str_replace($allvowels,
"","Hello World of PHP");
// Provides: You should eat pizza, beer, and ice
cream every day
$food_msg = "You should eat fruits, vegetables,
and fiber every day.";
$healthy_food = array("fruits",
"vegetables","fiber");
$testy = array("pizza", "beer", "ice cream");
$newphrase = str_replace($healthy_food,
$testy,$food_msg);
// Use of the count parameter is available as of
PHP 5.0.0
$str_replaced = str_replace("ll", "", "good golly
miss molly!", $count);
echo $count; // 2
// Order of replacement
$str_replaced = "Line 1\nLine 2\rLine 3\r\nLine
4\n";
$order = array("\r\n", "\n", "\r");
$replace = '<br />';
```

```
// Processes \r\n's first so they aren't converted twice.
$newstr = str_replace($order, $replace, $str_replaced);
// Outputs: apearpearle pear
$letters = array('e', 'p');
$fruit = array('apple', 'pear');
$text = 'e p';
$output = str_replace($letters, $fruit, $text);
echo $output;
?>
```

" get_extension_funcs()

The 'get_extension_funcs()' function returns an array with the names of the functions that are defined in the module.

Look at the example below:

Example:

```
<?php
print_r(get_extension_funcs("xml"));
?>
```

The output of the above example is something like the following:

```
Array
(
[0] => xml_parser_create
[1] => xml_parser_create_ns
[2] => xml_set_object
[3] => xml_set_element_handler
[4] => xml_set_character_data_handler
[5] => xml_set_processing_instruction_handler
[6] => xml_set_default_handler
[7] => xml_set_unparsed_entity_decl_handler
[8] => xml_set_notation_decl_handler
[9] => xml_set_external_entity_ref_handler
[10] => xml_set_start_namespace_decl_handler
[11] => xml_set_end_namespace_decl_handler
[12] => xml_parse
[13] => xml_parse_into_struct
[14] => xml_get_error_code
[15] => xml_error_string
[16] => xml_get_current_line_number
```

```
    [17] => xml_get_current_column_number
    [18] => xml_get_current_byte_index
    [19] => xml_parser_free
    [20] => xml_parser_set_option
    [21] => xml_parser_get_option
    [22] => utf8_encode
    [23] => utf8_decode
    )
    " dl()
```

The 'dl()' function loads an extension of PHP at runtime. If it is successfully executed, then it returns the Boolean value TRUE. If there is a problem during the execution of the program, it returns FALSE.

Example:

```
<?php
// Example loading an extension based on OS
if (!extension_loaded('sqlite')) {
   if (strtoupper(substr(PHP_OS, 0, 3)) ==='WIN')
{
        dl('php_sqlite.dll');
    } else {
        dl('sqlite.so');
    }
}

// Or, the PHP_SHLIB_SUFFIX constant is available
as of PHP 4.3.0
if (!extension_loaded('sqlite')) {
   $prefix = (PHP_SHLIB_SUFFIX === 'dll') ?'php_'
: '';
    dl($prefix . 'sqlite.' . PHP_SHLIB_SUFFIX);
}
?>
```

4.5 Static variables

PHP provides some Static variables which exist only in a local function scope. The Static variables do not loose their value in case the function (where they are declared) execution is terminated and is recalled later. Static variables can be used for working with recursive functions, where a function calls itself.

Example 1:

```
<?php
$abc = 9;
function fruit () {
static $abc = 0;
$fruit_arr                                                    =
array("mango","apple","guava","bananna");
$abc++;
$abc %= count($fruit_arr);
return $fruit_arr[$abc];
}
for ($j=0; $j<10; $j++) {
print "What's for dinner - ".fruit()."?<br />";
}
print $abc; # Just to show you this is a differ-
ent abc!
?>
```

The output of this program is:

```
Which fruit you like - apple?
Which fruit you like - guava?
Which fruit you like - bananna?
Which fruit you like - mango?
Which fruit you like - apple?
Which fruit you like - guava?
Which fruit you like - bananna?
Which fruit you like - mango?
Which fruit you like - apple?
Which fruit you like - guava?
9
```

Example 2:

```
<?php
function &quick_ref() {
static $static_obj;
echo "We are using Static object: ";
var_dump($static_obj);
if (!isset($static_obj)) {
// Assign a reference to the static variable
$static_obj = &new stdClass;
}
$static_obj->property++;
return $static_obj;
}
function &quick_noref() {
static $static_obj;
echo " We are using Static object: ";
var_dump($static_obj);
if (!isset($static_obj)) {
// Assign the object to the static variable
$static_obj = new stdClass;
}
$static_obj->property++;
return $static_obj;
}
$static_obj1 = quick_ref();
$still_static_obj1 = quick_ref();
echo "\n";
$static_obj2 = quick_noref();
$still_static_obj2 = quick_noref();
?>
```

The output of this program is as follows:

```
We are using Static object: NULL
We are using Static object: NULL
We are using Static object: NULL
We are using Static object: object(stdClass)#3
(1) { ["property"]=> int(1) }
```

Strings in PHP

5.1 Introduction to string

We have already discussed the basics of strings in PHP. Here we will look at some of the advanced features of strings.

We can use the concatenation operator (.) to concatenate two strings. Look at the example below:

Example:

```
<?php
$text1 = "Welcome";
$text2 = "abcd";
echo $text1 . " " . $text2;
?>
```

The output of this program is as follows:

```
Welcome abcd
```

Here, we have used the concatenation operator twice to insert another (third) string. We have inserted a string that includes a single character just between the two string variables. Here, we have used an empty space to separate the two variables.

Variables are parsed within the strings if it is inserted within double quotation marks or in the Heredoc strings. Here we have two types of syntax:

Simple: The simple syntax is most commonly used in PHP. We cannot parse an array value or an object property in a variable by using this syntax.

Complex: PHP4 has introduced complex syntax. It is always inserted within curly braces.

Simple:

When the PHP engine executes a dollar sign($), then the parser will receive a number token to develop a valid variable name. That is why we need to insert the names of the variables in curly braces

to mark the end of the variable.

Example:

```
<?php
$fruit = 'mango';
echo "$fruit 's taste is good <br>";
echo "He eat some $fruits <br>";
echo "He eat some ${fruit}s <br>";
echo "He eat some {$fruit}s <br>";
?>
```

The output of the above program is:

```
mango's taste is good
He eat some
He eat some mangos
He eat some mangos
```

In the above example, a variable $fruit is assigned with a string 'mango'. Then its value is displayed using different methods. In the first statement, " ' " is an invalid character for a variable. Therefore, it is treated as a symbol ('). In the second statement, "s" is a valid name, so '$fruits' is treated as a variable where no value is assigned. The other two statements show how variable names can be enclosed within braces.

In the same way, you can insert an array index as well as an object property that is to be parsed. The square bracket indicates the closing of the index.

You can declare an array as:

```
$shirt = array('tshirt' => 'white', 'bushshirt'
=> 'black');
```

Then the statements given below will execute properly:

```
echo "Tshirt is $shirt[tshirt].";
echo "Tshirt is $shirt[tshirt].";
echo "Tshirt is {$shirt['tshirt']}.";
echo "Tshirt is " . $shirt['tshirt'] . ".";
```

All the statements mentioned above will give the same output:

```
Tshirt is white.
```

However, the statement

```
echo "Tshirt is $shirt['tshirt'].";
```

will produce a parse error.

Complex:

In the complex syntax, you can insert complex expressions. Any value from the namespace in strings can be inserted here. The expression can follow the same format used for inserting it outside the string. Here, insert a curly brace.

Example:

```
<?php
$name = 'John';
echo "He is { $name} <br>";
echo "He is {$name} <br>";
echo "He is ${name} <br>";
?>
```

The output of the above program is as follows:

```
He is {John}
He is John
He is John
```

In the first statement, there is a blank space between ({) and ($), so the braces ({ }) are treated as characters. In the other two statements, the curly braces ({ }) are correctly used.

Object property and array elements can also be accessed using this method.

Example 01:

```
<?php
class fruit
{
var $name="mango";
}
$fr1=new fruit();
echo "This is {$fr1->name}.";
?>
The output of the above program is as follows:
This is mango.
Example 02:
<?php
```

```
$arr=array(1,2,3,4,5);
echo "The third element is {$arr[3]}.";
?>
```

The output of the above program is as follows:

```
The third element is 4.
```

You can easily access the characters that are inserted within the strings. Here, mention the zero based offset of the expected character just after the string in curly braces.

Example:

```
<?php
$str = "This is string";
$a = $str{0};
$b = $str{2};
$c = "This is also a string";
$d = $str{strlen($str)-1};
echo $a."<br>";
echo $b."<br>";
echo $c."<br>";
echo $d."<br>";
?>
```

The output of the above program is as follows:

```
T
i
This is also a string
g
```

You can convert a value to a string by using a string cast. Here, the `strayal()` function can also be used. The `echo()` and the `print()` functions are used to automatically convert a string in the scope of an expression, if a string is needed here. This can also be done while comparing a string to a variable. A PHP engine converts a `'TRUE'` value to the `'1'` string while the `'FALSE'` value is converted to an empty string. Similarly, you can also convert an integer or a float to a string. The exponential notation is used to convert a floating point number.

Example:

```
<?php
$foo = 1 + "10.5";                    // $foo is
```

```
float (11.5)
   $foo = 1 + "-1.3e3";                    // $foo is
float (-1299)
   $foo = 1 + "bob-1.3e3";                 // $foo is
integer (1)
   $foo = 1 + "bob3";                      // $foo is
integer (1)
   $foo = 1 + "10 Small Pigs";       // $foo is inte-
ger (11)
   $foo = 4 + "10.2 Little Piggies"; // $foo is float
(14.2)
   $foo = "10.0 pigs " + 1;                // $foo is
float (11)
   $foo = "10.0 pigs " + 1.0;        // $foo is float
(11)
   ?>
```

5.2 String functions

PHP provides us various in built string functions.

" `addcslashes()`

The addcslashes() function returns a string with backslashes in front of the specified characters. This function was introduced in PHP4. Look at the example below:

Example:

```
<?php
echo addcslashes('abcd[ ]', 'A..z');
?>
```

The output of the above program will be:

```
\a\b\c\d\[ \]
```

addslashes()

The `addslashes()` function returns a string with backslashes in front of the characters that are defined previously. These characters are single quote (`'`), double quote (`"`), backslash (`\`) and `NUL` (the NULL byte). This function was introduced in PHP3. Look at the example below:

Example:

```
<?php
```

```
$hello="Are you Jack's brother?";
echo addslashes($hello);
?>
```

The output of the above program is:

```
Are you Jack\'s brother?
```

bin2hex()

The `bin2hex()` function converts a string of ASCII characters to hexadecimal values. This was introduced in PHP3.

chop()

The chop() function is the pseudonym of rtrim(). This function was introduced in PHP3. Look at the example:

Example:

```
<?php
$text = "\t\t We are using chop :) ... ";
$choped = chop($text);
echo $choped."<br>";
$choped = chop($text," \t.");
echo $choped."<br>";
?>
```

The output of the above program is as follows:

```
We are using chop :) ...
We are using chop :)
```

chr()

The `chr()` function returns a single character string from an ASCII value that is already specified. This function was introduced in PHP3. Look at the example below:

Example:

```
<?php
$string .= chr(27); /* include an escape charac-
ter at the end of $string */
/* This will often help */
$string = sprintf("The defined string will end in
escape: %c", 27);
?>
```

chunk_split()

The `chunk_split()` function divides a string into a sequence of

small fragments. This function was introduced in PHP3.

Example:

```
<?php
// formatting $info by using the RFC 2045 semantics
$new_strng = chunk_split(base64_encode($info));
?>
```

convert_cyr_string()

The `convert_cyr_string()` function is used to convert a string from one Cyrillic character-set to another set. It was introduced in PHP3. Look at the types that are supported by this function:

```
" k - koi8-r
" w - windows-1251
" i - iso8859-5
" a - x-cp866
" d - x-cp866
" m - x-mac-cyrillic
```

str_ireplace()

The `str_ireplace()` function is used to replace some case insensitive characters in a string. This function was introduced in PHP5. Look at the example below:

Example:

```
<?php
$bdtag = str_ireplace("%bd%", "blue", "<body text=%BD%>");
?>
```

str_repeat()

The `str_repeat()` function is used to repeat a string, a specified number of times. This function was introduced in PHP4. Look at the example below:

Example:

```
<?php
echo str_repeat("*-", 8);
?>
The output of the above program is as follows:
*-*-*-*-*-*-*-*-
```

str_replace()

The `str_replace()` function is used to replace some case sensitive characters in a string. This function was introduced in PHP3.

Example:

```
<?php
$replace = array("a", "e");
$aft_replace = str_replace($replace, "i", "We are learning php");
echo $aft_replace."<br>";
$team = "Sourav, Rahul,Anil all are great cricketers.";
echo $team."<br>";
$present = array("Sourav", "Rahul","Anil");
$future = array("Dhoni", "Rohit", "Sehbag");
$newteam = str_replace($present, $future, $team,$num);
echo $newteam."<br>";
echo "there are {$num} change in team";
?>
```

The output of the above program is as follows:

```
Wi iri liirning php
Sourav, Rahul, Anil all are great cricketers.
Dhoni, Rohit, Sehwag all are great cricketers.
there are 3 change in team
```

str_split()

The `str_split()` function is used to split a string into an array. This function was introduced in PHP5. Look at the example below:

Example:

```
<?php
$text = "How are you";
$split1 = str_split($text);
$split2 = str_split($text, 3);
print_r($split1);
print_r($split2);
?>
```

The output of the above program is as follows:

```
Array
(
```

```
[0] => H
[1] => o
[2] => w
[3] =>
[4] => a
[5] => r
[6] => e
[7] =>
[8] => y
[9] => o
[10] => u
)
Array
(
[0] => How
[1] => ar
[2] => e y
[3] => ou
)
```

str_word_count()

The `str_word_count()` function is used to count the number of words in a string. This function was introduced in PHP4. Look at the example below:

Example:

```
<?php
$text = "Good morning friends! have nice day";
$a=str_word_count($text,1);
$b=str_word_count($text,2);
$c=str_word_count($text);
print_r($a);
print_r($b);
print $c;
?>
```

The output of the above program is as follows:

```
Array
(
[0] => Good
[1] => morning
[2] => friends
[3] => have
```

```
[4] => nice
[5] => day
)
Array
(
[0] => Good
[5] => morning
[13] => friends
[22] => have
[27] => nice
[32] => day
)
6
```

strcasecmp()

The `strcasecmp()` function is used to compare two case sensitive strings. This function was introduced in PHP3. Look at the example below:

Example:

```
<?php
$text1 = "Good morning";
$text2 = "Good morning";
if (strcasecmp($text1, $text2) == 0) {
echo '$text1 is equal to $text2 in a case-insensitive string comparison';
}
?>
```

The output of the above program is as follows:

```
$text1 is equal to $text2 in a case-insensitive string comparison
```

strchr()

The strchr() function is used to find the first appearance of a string inside another string. It is the pseudonym of the `strstr()` function. This function was introduced in PHP3.

strcmp()

The `strcmp()` function is used to compare two case sensitive strings. This function was introduced in PHP3.

stripcslashes()

The `stripcslashes()` function is used to unquote a string that is quoted by using the addcslashes() function. This function was introduced in PHP4.

stripslashes()

The `stripslashes()` function is used to unquote a string that is quoted by using the addslashes() function. This function was introduced in PHP3. Look at the following example:

Example:

```
<?php
$str = "Are you Jack?";
// Outputs: Are you Jack?
echo stripslashes($str);
?>
```

strlen()

The `strlen()` function is used to return the length of a specific string. This function was introduced in PHP3.

Example:

```
<?php
$text = 'aeiou';
echo strlen($text)."<br>";
$str = ' ab cd ';
echo strlen($str);
?>
The output of the above program is:
5
7
```

strncasecmp()

The `strncasecmp()` function is used to compare a case sensitive string of the first 'n' character. This function was introduced in PHP4.

substr()

The `substr()` function is used to return a part of a string. This function was introduced in PHP3.

substr_compare()

The `substr_compare()` function is used to compare two strings

from a specific starting position. This function was introduced in PHP5.

substr_count()

The `substr_count()` function is used to count the number of times a substring appears in a string. This function was introduced in PHP4.

Example:

```
<?php
print substr_count("Hello how are you and what are you doing now?", "are");
?>
Output:
2
```

substr_replace()

The `substr_replace()` function is used to replace a part of a string with another string of the program. This function was introduced in PHP4.

Example:

```
<?php
$str = 'INDIA:/DELHI/';
echo "Before using function: $str<hr>\n";
echo substr_replace($str, 'KOLKATA', 0) . "<br>\n";
echo substr_replace($str, 'MUMBAI', 0, strlen($str)) . "<br>\n";
echo substr_replace($str, 'BANGALURU', 0, 0) . "<br>\n";
echo substr_replace($str, 'HYDRABAD', 6, -1) . "<br>\n";
echo substr_replace($str, 'CHENNI', -7, -1) . "<br>\n";
echo substr_replace($str, '', 13, -1) . "<br>\n";
?>
```

The output of the above program is as follows:

```
Before using function: INDIA:/DELHI/

KOLKATA
MUMBAI
BANGALURUINDIA:/DELHI/
```

```
INDIA:HYDRABAD/
INDIA:CHENNI/
INDIA:/DELHI/
```

trim()

The `trim()` function is used to remove the white spaces from both sides of a string. This function was introduced in PHP3.

Example:

```
<?php
$text = "\t\t We are using trim :) ... ";
$trimed = trim($text);
echo $trimed."<br>";
$trimed = trim($text," \t.");
echo $trimed."<br>";
?>
```

The output of the above program is as follows:

```
We are using trim :) ...
We are using trim :)
```

Object Orientation in PHP

6.1 Getting started

We will now go through the basic concepts of the Object Oriented Programming (OOP). Here, variables and methods are grouped together to form a class. These classes are instantiated by creating objects. The members of a class are accessed through the object of a class.

6.2 Class and object

A class implements the concept of ADT (Abstract Data Type). It is a compilation of methods and objects. A class definition always begins with the keyword 'class', followed by an identifier that represents the name of the class. These are followed by a pair of curly braces ({ }) that contains the members of the class. In the following example, we have defined a class Wonderbox.

Example:

```
<?php
class Wonderbox
{
var $val;
function getvalue()
{
$this->val=10;
}
function showvalue()
{
echo "You have entered ".$this->val;
}
}
$ourobj = new Wonderbox();
$ourobj->getvalue();
```

```
$ourobj->showvalue();
?>
```

The output of the above program is as follows:

```
You have entered 10
value is changed to 20
```

This is the basic structure on which we have constructed our Wonderbox class. It includes the variable '$val' and the two following functions: 'getvalue()' and 'showvalue()'. The keyword 'var' is used to declare a variable within a class. By nature this type of variables are public and can be accessed by the methods of the class. They can also be accessed outside a class as shown in the above example. You can also use the keyword 'public' instead of 'var'. '$this' is a pseudo variable and '->' is an operator. Using this combination, you can access any member (property or value) within the class itself.

A class is a template. To use this template an object must be instantiated. The keyword 'new' is used to instantiate an object. In the above example, $ourobj represents an object of the class Wonderbox. You can access the members of the class using the object and the '->' operator. Any member declared with the keyword "private" or "protected" cannot be accessed outside the method of the class.

Example:

```
<?php
class ourclass
{
private $val;
function getvalue()
{
$this->val=10;
}
function showvalue()
{
echo "within method value is ".$this->val;
}
}
$ourobj=new ourclass();
$ourobj->getvalue();
```

```
    $ourobj->showvalue();
    $ourobj->val=20;
    echo "<br>value outside the method ".$ourobj->val;
    ?>
```

The above example will display an error message, since any private data member cannot be accessed outside the method of the class.

6.3 Classes as namespaces

The 'Namespace' solves the problem of scoping in the huge PHP library. Class definitions are global in PHP. It helps to manage naming scope without using a long name, while referring to a class. It solves the problem of sharing a global space as well. We can avoid making the codes unreadable. While declaring a namespace, the keyword 'namespace' is used at the beginning of the file. A single namespace can be used for more than one file. The namespace includes class, constants and function definitions. It never includes free codes. See the following example:

Example:

```
<?php
    namespace ourNameSpace::DB;

    const CONNECTION = 1;

    class Connection { /* ... */ }

    function connect() { /* ... */  }

?>
```

All classes, functions and constant names remain automatically prefixed inside the namespace. The namespace name and constant name together develop the constants. The namespace constants always include static values.

In the absence of a namespace definition, insert the class and the function definition into the global space. To specify a global space, insert the double colon sign (::) into the name of the namespace.

Example:

```
<?php
     namespace X::Y::Z;

  /* This is the function X::Y::Z::fopen */
     function fopen() {
          /* ... */
           $ab = ::fopen(...); // calling global
fopen
          return $ab;
     }
?>
```

The constant named '__NAMESPACE__' holds the value of the current namespace as a string. In the Global Scope or inside the Code without a namespace, the value of the Constant '__NAMESPACE__' is an empty String. While composing a full name for local namespace, we use this constant. See the example below:

Example:

```
<?php
namespace X::Y::Z;

function foo() {
// doing stuff
}

set_error_handler(__NAMESPACE__ . "::foo");
?>
```

6.4 Objects as References

'Reference' is an exclusive feature of PHP. Using 'Reference' we can have multiple names for the same Variable.

Example:

```
<?php
$x = 10;
$y = & $x;
$x++;
echo 'The value of $y is ' . $y;
?>
```

The output of the above program is as follows:

```
The value of $y is 11;
```

In the above example, the $x and the $y are the same variable with two different names. Therefore, increasing the value of $x also affects the value of the variable.

Let's see how we can unset a reference:

While unsetting a reference, we need to separate the variable name from the variable content. Here, the variable content will not be ruined.

Example:

```
<?php
$x = 1;
$y =& $x;
unset ($x);
?>
```

Here, it will not unset the `$y` variable, rather it will remove the association of the name `$x` from the variable. The variable named `$y` still holds the same Variable and has the same content.

The example given below will show how an object can be referred by other names.

Example:

```
<?php
class ourclass
{
private $val;
function getValue()
{
$this->val=10;
}
function showValue()
{
      echo "within method value is ".$this->val;
}
}
$ourObj = new ourclass();
$refObj = & $ourObj;
$refObj->getValue();
```

```
$refObj->showValue();
?>
```

Here the object `$ourObj` will be referred as `$refObj`. We can easily spot a reference here. There are various PHP constructs that are executed by reference. Look at some of those constructs:

Global references:

When we declare a `'global $var'` in a program, it indicates that we are developing a reference to a 'global variable'. See the code below:

Example:

```
<?php
$var = 100;
$varRef = & $GLOBALS['var'];

echo "\nVar = " . $var;
echo "\nVarRef = " . $varRef;

unset($var);
echo "\nVar = " . $var;
echo "\nVarRef = " . $varRef;
?>
```

The output of the above program will be as follows:

```
If we unset the $var, then it does not unset the
global variable.
" $this
```

Inside a class, the '$this' variable always refers to the caller objects. Basically, the '$this' variable is used to specify a local variable. It instructs PHP to point to the particular object with which you are presently working. Look at the example below:

Example:

```
function buzzing() {
print "{$this->Name} says Woof!\n";
}
```

6.5 Implementing inheritance

Inheritance is a concept by which members (property and method) of one class can be used by another class.

Example:

```
<?php
class pclass
{
var $p_val;
function p_getval()
{
$this->p_val = "parent";
}
function p_showval()
{
echo "We are in $this->p_val class method";
}
}
class cclass extends pclass
{
var $c_val;
function c_getval()
{
$this->c_val = "child";
}
function c_showval()
{
echo "We are in $this->c_val class method";
}
}
$cobj = new cclass();
$cobj->p_getval();
$cobj->p_showval();
$cobj->c_getval();
echo "<br>";
$cobj->c_showval();
?>
```

The output of the above program is as follows:

```
We are in parent class method
We are in child class method
```

In the above example, the 'extends' keyword is used to tell the PHP parser that the cclass class inherits the pclass class. Here pclass' is called the parent class and 'cclass' is called the child or the derived class. All the members of 'pclass' are now available in 'cclass'. Therefore, the members of 'pclass' can be accessed by the object of 'cclass' as shown in the above example. A new member can also be created in the child class.

6.6 Method overriding

We can easily redefine functions of parent class in the derived or child class. Look at the examples below:

Example 1:

```
<?php
class pclass
{
var $p_val = "parent";
function showval()
{
echo "We are in $this->p_val class";
}
}
class cclass extends pclass
{
var $c_val = "child";
function showval()
{
echo "We are in $this->c_val class";
}
}
$cobj = new cclass();
$cobj->showval();
?>
```

In the above example, we have redefined the function `showval()` inside the `'cclass'`. Now when you access the function `showval()` using `'cclass'` object, the definition of `'cclass'` is executed. If you want to use the definition of `'pclass'`, then you have to create the object of `'pclass'` or use the method that is used in following example.

Example 02:

```
<?php
class pclass
{
var $p_val="parent";
function showval()
{
echo "We are in $this->p_val class";
}
}
class cclass extends pclass
{
var $c_val="child";
function showval()
{
echo "We are in $this->c_val class";
pclass::showval();
}
}
$cobj=new cclass();
$cobj->showval();
?>
```

6.7 Magic functions

Magic Functions are those with a double underscore sign (__). We do not declare these functions and are reserved by PHP. Look at the various 'Magic Functions':

- __autoload()
- __get()
- __set()
- __call()
- __toString()

Now let's go through them in detail:

__autoload()

The __autolad() function is a magic function which is called when you try to create an instant of a class which has not been declared. So we can implement our own version of this function to try to include the file which declares the class before an error is displayed. Look at the example below:

Example:

```
<?php
function __autoload($c_name) {
print "Bad class name: $c_name!\n";
include "abcclass.php";
}
$obj = new abc;
$obj->show();
?>
```

In the above example, we have tried to create an object of 'abc' class that is not defined previously. So the function __autoload() is called automatically by the PHP Engine. Inside this, we have included the File (abcclass.php) where the Class abc is defined. This will create the object as per our need.

__get()

The `__get()` function is used to specify the action if an unknown class variable is read from within the script.

Example:

```
<?php
class student{
var $Name;
var $roll;
// public $address;
public function __get($val) {
print "Attempted to retrieve $val and failed...\n";
}
}
$std1 = new student;
print $std1->address;
?>
```

In the above example, the student class includes the commented declaration of the variable '$address'. Here, the __get() function will be called to display the following output.

The output of the above program is as follows:

Attempted to retrieve address and failed...

__set()

The __set() magic function is used to complement the __get() function. This function is called when you set an undefined class variable in a program. Here we have given an example where we have used the '__set()' magic function to develop a database table class. Assuming itself as the member of the class, it performs an unplanned enquiry.

Example:

```
<?php
//...[snip - insert the MySQL connection code here]...
class newtable {
public $Naming;
// public $AdministrativeEmail;
public function __construct($Naming)
{
$this->Naming = $Naming;
}
public function __set($var, $val) {
mysql_query("UPDATE {$this->Naming} SET $var = '$val';");
}
// public $AdministrativeEmail = 'foo@bar.com';
}
$systemvars = new newtable("systemvars");
$systemvars->AdministrativeEmail = 'telrev@some-site.net';
?>
```

In the above example, $AdministrativeEmail is commented out. It is not available in the newtable class. $AdministrativeEmail is set on the last line and __set() is called here. It includes the name of the variable that is being set.

__call()

Another important magic function is the __call() function. Example:

```
<?php
class Bee {
public $Naming;
public function buzz() {
```

```
print "Woof!\n";
}
// public function meow() {
// print "Bees don't meow!\n";
// }
public function __call($function, $args) {
$args = implode(', ', $args);
print "Call to $function() with args '$args'
failed!\n";
}
}
$honey = new Bee;
$honey->meow("foo", "bar", "baz");
?>
```

In the above example, the `meow()` function is commented out. You can remove the comments from the `meow()` function by ensuring that the `__call()` function is not used in case the function already exists.

__toString()

The `__toString()` magic function is used to set a string value for an object. This function will only be used if this object is used as a string. Look at the following example:

Example:

```
<?php
class dog {
public function __toString() {
return "This is a dog\n";
}
}
$tommy = new dog;
print $tommy;
?>
```

The output of the above program is as follows:

```
This is a dog
```

Here the object `$tommy` is used as string with the help of `__toString()` function.

Working with forms

One of the remarkable features of PHP is the handling of HTML forms. All the form elements of an HTML page are available in PHP. HTML forms on the World Wide Web are important for transferring substantial standards of information from the user to the server. In PHP, it is easy to acquire and work with the information submitted by HTML forms.

7.1. Global and environmental variable:

Before making a form for acquiring data from the users, you need to make a small diversion and check the global variables. Usually global variables are declared in the initial part of a script and outside a function. The functions are available in a global associative array. See the example below:

Example:

```
<html>
<head>
<title>Getting information from the $GLOBALS array</title>
</head>
<body>
<?php
$student1 = "Sachin";
$student2 = "Sourav";
$student3 = "Rahul";
foreach ( $GLOBALS as $key => $value )
{
print "\$GLOBALS[\"$key\"] == $value<br>";
}
?>
</body>
</html>
```

From the above example, we can understand that all the three declared variables will represent the keys of the $GLOBAL associative array and their values will be displayed. Apart from this, PHP

automatically provides the description of the PHP global variables related to the server and client environments. Hence, these variables are called environment variables. The variables in PHP vary depending on system, server and configuration. See the examples of environment variables in the following table:

Example:

Name of the variable	Description	Example
$HTTP_USER_AGENT	Client's Name and version	Mozilla
$REMOTE_ADDR	The IP address of the client	155.148.65.33
$REQUEST_METHOD	Whether the request was GET or POST	POST/GET
$QUERY_STRING	For GET requests, the encoded data send appended to the URL	id=I001t&product=abcd
$REQUEST_URI	The full address of the request including query string	/php-learing book/unit_07/newpage.html?id=I001
$HTTP_REFERER	The address of the page from which the request was made	http://www.ourpage.com/newpage.html

Apart from header-oriented variables, PHP also offers certain other global variables. You can directly access this variable as the global variable `$PHP_SELF`. You can also use it as a string in the HTML forms action argument. This saves time in hard coding the page name. Both the global and environmental arrays in PHP are useful in different ways.

7.2. Script to accept user input:

The HTML form can be separated from PHP code. See the example below:

Example:

```
<html>
<head>
```

```
<title>Our HTML form</title>
</head>
<body>
<form action="ourpage.php">
<b>USER ID:</b>
<input type="text" name="userid">
<br>
<b>PASSWORD:</b>
<input type="password" name="password">
<br>
<input type="submit" value="login">
</form>
</body>
</html>
```

Here, we have created a form containing a text field with the name 'userid', a text field to accept a password with the name 'password' and a submit button named 'login'. Since nothing more than a file name has been added to the action argument, it is assumed that both the PHP file (ourpage.php), and the HTML document are on the same directory of the server. Here we have used the GET method discussed later.

The following code receives the user input (the code of ourpage.php):

```
<html>
<head>
<title>Check User Validation</title>
</head>
<body>
<?php
print "Welcome <b>$_GET[userid]</b><P>\n\n";
print "Your password is:<P>\n\n<b>$_GET[password]</b>";
?>
</body>
</html>
```

In the above code, two variables `$_GET[userid]` and `$_GET[password]` have been used to access the data entered by the user in the above HTML form. Every detail that a user submits

is available in the global variables having the same names as the form elements on an HTML page.

7.3. Accessing input from various elements of a form

In all the examples mentioned earlier, each HTML element submits a single value. Let us see what happens if an HTML element accepts more than one value from users. This can be done by using the selected elements in the HTML form. With the availability of these elements, the user can choose multiple items. Look at the following code:

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

Any script with this data can only access a single value corresponding to the given name. This activity can be changed by renaming the elements. Here the elements end with an empty set of square brackets. See the example below:

Example:

```
<html>
<head>
<title>Fill up the Form</title>
</head>
<body>
<form action="ourpage1.php" method="POST">
<b>USER ID:</b>
<input type="text" name="userid">
<br>
<b>CITY:</b>
<select name="city[]" multiple>
<option>Kolkata
<option>Delhi
<option>Mumbai
<option>Bangaluru
</select>
<br>
<input type="submit" value="submit">
</form>
</body>
</html>
```

The script that processes the form input, has that input in the `city [ ]` element available in an array called `$city`.

```
<html>
<head>
<title>WELCOME PAGE</title>
</head>
<body>
<?php
print "Welcome <b>$_POST[userid]</b><p>\n\n";
print "Your city choices are:<p>\n\n";
if ( ! empty( $_POST[city] ) ) {
print "<ul>\n\n";
foreach ( $_POST[city] as $val ) {
print "<li>$val<br>\n";
}
print "</ul>";
}
?>
</body>
</html>
```

The SELECT element is not the only one to offer multiple values. By assigning similar name to numerous check boxes, you can select different values in a single field name. When the chosen name ends with empty square brackets, PHP assembles the user input for this field into an array. The SELECT element can be replaced with a string of check boxes to bring the exact result:

```
<input type="checkbox" name="qualification[]"
value="bca">BCA<br>
<input type="checkbox" name=" qualification[]"
value="bba">BBA<br>
<input type="checkbox" name=" qualification[]"
value="mca">MCA<br>
<input type="checkbox" name=" qualification[]"
value="mba">MBA<br>
```

7.4. Accessing inputs in an associative array

Sometimes, techniques may mess up the scripts with global variables. To confine the number of globals in a script, the features can be disabled for every form field by setting the 'register_globals' command to off in the php.ini file.

PHP4 provides a solution to this problem. Basically, two methods are used to submit a form, the Get or Post method. This method is declared in the HTML form with the attribute named 'method'. The values submitted in a form can be accessed through the '$HTTP_GET_VARS' variable, if we use the Get method. While using the Post method the values are accessed through the '$HTTP_POST_VARS' variable. The example below gives a clear idea how to read from any form with the help of $HTTP_GET_VARS array:

Example:

```
<html>
<head>
<title>List of the values inserted in a form</title>
</head>
<body>
<?php
foreach ( $HTTP_GET_VARS as $key=>$val )
{
print "$key => $val<BR>\n";
}
?>
</body>
</html>
```

The above code displays the names and values of different parameters passed through the GET transaction. The above code will yield an output if each form element accepts a single value. If the form element accepts multiple values, then the above code will not be able to show those values. This problem can be solved by testing the data type of each element in `$HTTP_GET_VARS` or `$HTTP_POST_VARS` and by treating them accordingly. The code below tests the data type of every element in `$HTTP_GET_VARS`. It changes the output accordingly.

Example:

```
<html>
```

```
<head>
<title>Reading values inserted in a form</title>
</head>
<body>
<?php
foreach($HTTP_GET_VARS as $key => $val)
{
If (gettype($val) ==”array”)
{
print “$key => <br>\n”;
echo “<ul>”;
foreach($val as $individualValuel)
{
echo “<li>”;
print “$individualValue”;
echo “</li>”;
}
echo “</ul>”;
}
else
{
print “$key => $val<br>\n”;
}
}
?>
</body>
</html>
```

Here, we have again used the ‘foreach()’ function to navigate through the ‘$HTTP_GET_VARS’ array. The ‘gettype()’ function is used to confirm that the values inserted within the function are arrays. Since the arrays are used as function values, another ‘foreach’ statement is created to navigate through it. The values are displayed in the web browser as the output of the program.

The above code displays the values inserted in the HTML form in section 7.3 and the output is as follows:

```
userid==ascascs
city ==
“ Kolkata
“ Mumbai
```

7.5. Get and Post method:

A flexible script has the ability to decide whether to read the $HTTP_GET_VARS or $HTTP_POST_VARS arrays. Usually, in all systems one can trace whether the user is working with a GET or POST method in the predefined variable $REQUEST_METHOD. This variable should contain the string POST or GET. You can use the isset() function to check the type of array that is to be read. The above example can be rewritten using this concept as given below:

Example:

```
<html>
<head>
<title>Reading values inserted in a form after checking the array</title>
</head>
<body>
<?php
$check = (isset($HTTP_POST_VARS)) ? $HTTP_POST_VARS : $HTTP_GET_VARS;
echo $check;
foreach($check as $key => $val)
{
If(gettype($val) ==”array”)
{
print “$key => <br>\n”;
echo “<ul>”;
foreach($val as $multi_val)
{
echo “<li>”;
print “$multi_val”;
echo “</li>”;
}
echo “</ul>”;
}
else
{
print “$key => $val<br>\n”;
}
}
?>
</body>
</html>
```

In the above example, the `$check` variable has been used by applying the ternary operator. By using the inbuilt `isset()` function, you can check if the $HTTP_POST_VARS array comprises elements.

Basically, both `GET` and `POST` methods are used to send data to the data processing page. Though both methods are used in form data handling, there are differences in their working method. Some remarkable differences are:

- The data remains visible in the address bar since contents are passed as part of the URL and as a query string in the GET method. In the POST method, data is not visible as contents are passed to the script as an input file.
- Using the GET method, you can insert a bookmark link whereas a page link can never be bookmarked with the POST method.
- You can only transfer 1KB of data through the GET method. Large amounts of data can be transferred through POST method, which is determined by the 'post_max_size' directive in php.ini.
- In the GET method, data is submitted as a part of a URL, while in the POST method data is submitted as part of an http request.
- In the GET method, data is swift but not secure. On the other hand, POST data is secure and slow as compared to GET.

7.6. File upload

In the sections above, we have seen simple form input. Now we will discuss about the features that PHP creates to work with inputs. Here, we will learn about file uploading in PHP. PHP makes it possible to upload files to the server. To begin with, we should first create HTML forms including file upload fields and an ENCTYPE argument:

```
ENCTYPE="multipart/form-data"
```

Look at the HTML form below, used for uploading files:

```
<html>
<body><form action="upload1.php" method="post"
enctype="multipart/form-data">
<label for="file">Filename:</label>
<input type="file" name="uploadedfile" id="file"
/>
<br />
<input type="submit" name="submit" value="Upload"
```

```
/>
   </form></body>
   </html>
```

In the above code, the enctype feature of the <form> tag focuses on the content category to be used for submitting the form. To insert a binary data in an HTML form, say, the content of a file, the "multipart/form-data" should be used. The type= "file" in the above code implies that the input should be processed as a file.

See the upload script below:

The file name upload_file.php includes the code for uploading a file:

Example:

```
<?php
if ($_FILES["file"]["error"] > 0)
{
echo "Error: " . $_FILES["uploadedfile"]["error"]
. "<br />";
}
else
{
echo "Uploaded File: " . $_FILES["uploaded-
file"]["name"] . "<br />";
echo "Uploaded File Type: " . $_FILES["uploaded-
file"]["type"] . "<br />";
echo "Uploaded File Size: " . ($_FILES["uploaded-
file"]["size"] / 1024) . " Kb<br />";
echo "Uploaded File Stored in: " .
$_FILES["uploadedfile"]["tmp_name"];
}
?>
```

With the use of global PHP $_FILES array, a file can be successfully uploaded from a client computer to the remote server. The form's first parameter often remains input name and the second index varies from name, type, size, tmp_name or error. Look at the following code:

```
$_FILES["uploadedfile"]["name"] - uploaded file
name
```

```
   $_FILES["uploadedfile"]["type"] - uploaded file
type
   $_FILES["uploadedfile"]["size"] - uploaded file
size in bytes
   $_FILES["uploadedfile"]["tmp_name"] - the name of
the temporary copy of the uploaded file stored on
the server
   $_FILES["uploadedfile"]["error"] - the error code
resulting from the file upload
```

Using the above mentioned code, you can easily upload files. To keep them secured, you can also add certain limitations of uploading.

Uploading restrictions:

The script below shows certain restrictions to be followed while uploading a file. A user may only upload JPEG or GIF files and the file size must be less than or equal to 40KB. See the example below:

Example:

```
   <?php
   if     ((($_FILES["uploadedfile"]["type"]     ==
"image/gif")
   ||      ($_FILES["uploadedfile"]["type"]      ==
"image/jpeg")
   ||      ($_FILES["uploadedfile"]["type"]      ==
"image/pjpeg"))
   && ($_FILES["uploadedfile"]["size"] < 50000))
   {
   if ($_FILES["file"]["error"] > 0)
   {
   echo "Error:" . $_FILES["uploadedfile"]["error"]
. "<br />";
   }
   else
   {
   echo "Upload File Name: " . $_FILES["uploaded-
file"]["name"] . "<br />";
   echo "Upload File Type: " . $_FILES["uploaded-
file"]["type"] . "<br />";
   echo "Upload File Size: " . ($_FILES["uploaded-
file"]["size"] / 1024) . " Kb<br />";
```

```
   echo "Upload File Stored in: " . $_FILES["upload-
edfile"]["tmp_name"];
   }
   }
   else
   {
   echo "File not valid";
   }?>
```

The code above describes the creation of a temporary copy of the uploaded files in the PHP temp folder on the server. These temporary files often disappear when the script ends. Hence, to keep them safe, you need to secure the uploaded files. Follow the example below:

Example:

```
   <?php
   if     (((($_FILES["uploadedfile"]["type"]     ==
"image/gif")
   ||      ($_FILES["uploadedfile"]["type"]      ==
"image/jpeg")
   ||      ($_FILES["uploadedfile"]["type"]      ==
"image/pjpeg"))
   && ($_FILES["uploadedfile"]["size"] < 50000))
   {
   if ($_FILES["uploadedfile"]["error"] > 0)
   {
   echo        "Error        Code:        "        .
$_FILES["uploadedfile"]["error"] . "<br />";
   }
   else
   {
   echo "Upload: " . $_FILES["uploadedfile"]["name"]
. "<br />";
   echo "Type: " . $_FILES["uploadedfile"]["type"] .
"<br />";
   echo "Size: " . ($_FILES["uploadedfile"]["size"]
/ 1024) . " Kb<br />";
   echo        "Temp        file:        "        .
$_FILES["uploadedfile"]["tmp_name"] . "<br />";
   if            (file_exists("upload/"            .
$_FILES["file"]["name"]))
```

```
    {
    echo $_FILES["uploadedfile"]["name"] . " already
exists. ";
    }
    else
    {
    move_uploaded_file($_FILES["uploadedfile"]["tmp_
name"],
    "temp/" . $_FILES["uploadedfile"]["name"]);
    echo "Stored in: " . "temp/" . $_FILES["upload-
edfile"]["name"];
    }
    }
    }
    else
    {
    echo "File not valid ";
    }
    ?>
```

In this way, you can check whether the same file name already exists or not. The file gets copied to a new folder if it does not exist.

File manipulation

The most fundamental advantage that any programming language offers is the process to create and manipulate with data structures. The structures that we create in PHP are sometimes difficult to memorise. Usually, these include variables, like arrays and objects or some disk units like files and database tables. In spite of these disadvantages, the frequency, ease and consistency with which a file is created, modified and erased, is more important.

File manipulation is one of the most basic requirements for professional programmers. PHP offers multiple options to create, upload and edit files. In PHP, as with many other programming languages, you can read from a file and also write into it. PHP also offers complete scope for file and directory manipulation. With PHP installed on your local drive, you can read and record directory contents, recover documents into different data styles and view and transform file features. You can also change file permissions and search for special files.

File manipulation is a distinguishing feature of PHP. While manipulating a file, you must take immense care, as even a minor mistake on your part can cause substantial damage. Some of the common errors committed by most programmers, while manipulating a file include:

- Editing a wrong file
- Filling a hard-drive with unnecessary data
- Accidentally deleting contents from a file.

8.1 Testing Files

PHP makes testing, reading and writing, simple and compact. Before you start work with a file or a directory, always collect detailed information. In PHP4, there are multiple options that help in assorting information about files on your personal computer. In this section of PHP, we will discuss some of the useful facts about files. Suppose you have saved a file assigning a name to

it and now you want to check whether the same file name exists or not. You can use a simple function, file_ exists(). It takes the name (of a file or a directory) with the relative path as an argument. If the file exists, it returns true. If it returns 'false' then it indicates that the file did not exist.

Sometimes, as per our requirements, we need to check the existence of a directory. You can do this by using is_dir () function. This function also requires a name with a relative path as an argument and also returns a Boolean value. Look at the example:

```
if ( is_dir( "ABC" ) )
print "ABC is a directory";
```

The above program will print "ABC is a directory" if a directory ABC exists in the working folder/directory.

Once you are sure of the existence of the file, you can manipulate it in multiple ways. Now, you can easily read, write on or execute this file. PHP supports all these mechanisms. Often on UNIX Systems, you can see a particular file, but cannot read its contents. The ' is_readable()' function briefs whether a particular file is readable or not. This function accepts the file path in a string and returns a Boolean value. Look at the example:

```
<?php
Is_readable("XYZ.txt");
print "XYZ.txt is readable";
?>
```

The above program will print "XYZ.txt is readable" if the file XYZ.txt is found readable.

To confirm whether a file is writable or not, there is a separate function named is_writable(). This function also requires a file path. It returns a Boolean value: true if the file is writable and false otherwise. Similarly, there is another function is_executable() that tells whether a file is executable or not. To verify if a file XYZ.txt is executable, you can use, 'is_executable ("XYZ.txt")'. This function returns true if the file is executable and false otherwise.

Ascertaining file size with file_size()

The filesize() function takes name of a file or path as an argument and returns the file size in bytes. While working, if the path faces any problem, it returns false. Look at the syntax to determine the file size

```
<?php
print filesize( "XYZ.txt" );
?>
```

The above program will print the size of XYZ.txt in bytes.

Determining date information about a file:

Sometimes we need to know the time a particular file was last written or accessed. There are certain paths like 'fileatime()' through which you can trace the last accessibility time of a file. Accessing a file means, to read or write on it. The accessibility time and dates are in the UNIX epoch format.

The information about modification of date and time of a file can be collected using the function filemtime(). To determine date information, it requires file path. Here, it returns the date in the UNIX epoch format. Modification of a file suggests altering its contents.

```
$mod_time = filemtime( "XYZ.txt" );
print "XYZ.txt was last modified on ";
print date("D d M Y g:i A", $mod_time);
// Sample output: Thu 13 Jan 2000 2:26 PM]
```

With PHP, you can change the test time of a document using the function 'filectime()'. The UNIX system supplies information about file modifications or changes. In other systems, the function filectime() provides the creation date.

```
$cng_time = filectime( "XYZ.txt" );
print " XYZ.txt was last changed on ";
print date("D d M Y g:i A", $cng_time);
// Sample output: Thu 13 Jan 2000 2:26 PM]
```

Mentioned below is an example focusing the use of a function to output various file tests:

```
<html>
<head>
<title> use of a function to output various file
tests </title>
</head>
<body>
<?php
$file_name = " XYZ.txt";
outputFileTestInfo( $file_name );
function outputFileTestInfo( $ file_name )
{
182
if ( ! file_exists( $file_name ) )
{
print "$file_name does not exist<BR>";
return;
}
print "$ file_name is ".(is_file( $ file_name
)?"":"not ")."a file<br>";
print "$file_name is ".(is_dir( $ file_name
)?"":"not ")."a directory<br>";
print "$file_name is ".(is_readable( $file_name
)?"":"not ")."readable<br>";
print "$file_name is ".(is_writable( $file_name
)?"":"not ")."writable<br>";
print "$file_name is ".(is_executable( $file_name
)?"":"not ")."executable<br>";
print "$file_name is ".(filesize($file_name))."
bytes<br>";
print "$file_name was accessed on ".date( "D d M
Y g:i A", fileatime( $file_name ) )."<br>";
print "$file_name was modified on ".date( "D d M
Y g:i A", filemtime( $file_name ) )."<br>";
print "$file_name was changed on ".date( "D d M
Y g:i A", filectime( $file_name ) )."<br>";
}
?>
</body>
</html>
```

8.2 Opening files:

Before beginning to work with a file, it is indispensable to know how to open it for different purposes. This includes reading, writing or sometimes both. PHP has the 'fopen()' function for this purpose. The function used for opening a file should include a string that contains a file path. It should also include another string with the mode in which the file is expected to be opened.

Some of the common modes used in the function for opening a file include read ('r'), write ('w'), and append ('a'). Certain special functions are applied to open files for different purposes. The following examples would provide the clear idea:

- File to open for reading: $fp = fopen("XYZ.txt", 'r');
- File to open for writing: $fp = fopen(" XYZ.txt", 'w');
- File to open for appending: $fp = fopen(" XYZ.txt", 'a');

If the file fails to open for any reason the function fopen() returns false, generating a message. The following example illustrates this point.

```
<html>
<body>
<?php
$file_ptr = fopen("welcomefile.txt","r") or exit("Unable to open file!");
?>
</body>
</html>
```

As we have seen earlier, there are three standard methods of opening a file in PHP. However, there are certain other methods, also, through which you can open a file so that both reading and writing can easily be done. This is possible by inserting a plus symbol (+) after the file mode.

To open a file, both for reading and writing `r+` (the file pointer should be at the beginning of the file)

To delete all information from the file when the file is opened `w+` (the file pointer should be at the beginning of the file)

To append `a+` (the file pointer should be at the end of the file)

8.3. Closing files:

Consider you have opened a file and have worked on it. Now you want to close the file as an open file may disturb the server by capturing the resources and causing unwanted turbulence. PHP has functions for both opening and closing files.

It offers an easy method of closing a file. Even if you forget to close file that is open, the server automatically closes all files when the PHP execution is completed. It is a good practice to close all files once you are done.

PHP provides the 'fclose() function to close file that is open. The 'fclose()' function requires the file handle to close it. Once a file is closed using the fclose() function, it cannot be read, written into or further appended. You can reopen the file with the same 'fopen()' function.

Look at the example:

```
1.      <?php
$file_name = fopen("XYZ.txt","r");
//some code to be executed
fclose ($file_name);
?>
```

```
2.      $ourFileName = "testFile.txt";
$ourFileHandle = fopen($ourFileName, 'w') or
die("can't open file");
fclose($ourFileHandle);
```

While closing a file, it is necessary to check, if the file has reached its end. The following syntax supports this argument:

The `feof()` function checks if the "end-of-file" (EOF) has been reached.

8.4. Reading from a file

To read information from a file, it is necessary to know the basic function used to open a file. Look at the example below:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'r');
```

This function will open the file and you can read it. The 'fread()' function reads data from a file. This function requires a file handle along with an integer to command how much data in bytes should read at a time.

Basically, the fgets() function reads each line from a file. Read the example below:

```
<?php
$file_ptr = fopen("welcomefile.txt", "r") or exit("Unable to open file!");
//Output a line of the file until the end is reached

while(!feof($file_ptr))
{
echo fgets($file_ptr). "<br />";
}
fclose($file_ptr);
?>
```

The above example describes how a function can be used to read a file line by line. The example below would focus on how a file can be read character by character:

Look at the code below:

```
<?php
$file_ptr = fopen("welcomefile.txt","r") or exit("Unable to open file!");
while (!feof($file_ptr))
{
echo fgetc($file_ptr);
}
fclose($file_ptr);
?>
```

8.5. Writing to a file

Writing to a file is a major part to know about in file manipulation. Basically, the fwrite() function is used to write. To write information into a text file, first open it by using the fopen() function. See the code below:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w');
```

After opening the file, you can use the fwrite() command to add data to your file. Look at the example below:

```
<?php
$file_name = "ourFile.txt";
$file_Handle = fopen($file_name, 'w');
$file_Data = "Jane Doe\n";
fwrite($file_Handle, $file_Data);
$file_Data = "Bilbo Jones\n";
fwrite($file_Handle, $file_Data);
print "Data Written";
fclose($file_Handle);
?>
```

After working on the file, close the file using the fclose function to keep the document safe. In the above example you can notice that at the end of every data string, we have used `\n`.

8.6. Locking files

If you are through with the input and output operations, lock it immediately. This should be done each time you do an input and output operation if you have opened a file to read and write data. Now, unless you lock this file, the authenticity of the file cannot be maintained. Mere knowledge of reading and learning files can only help when a script is presented to a single user. However, if you want to make your file accessible to multiple users at the same time, PHP4 has provision for this. It provides the flock() function with which you can lock your existing document.

`flock()` not only locks an existing document, but also prevents the file from further writing or reading. The `flock()` function always requires a file pointer. Therefore, sometimes you have to use one special lock file to prevent file access. A file locking system requires a modern approach. In PHP, the flock system is used virtually at different stages.

In PHP, a complete file can be locked in an advisory pattern. This suggests that all other programs are entitled to use the similar locking manner unless they do not work. A file locked using flock can be released by `fclose()`.

flock considers the file handle as the first parameter, while the entire lock operation follows next. The operations applied in PHP include LOCK_SH (requests a shared lock), LOCK_EX (requests an exclusive lock), and LOCK_UN (releases a lock). In a PHP file, flock can be used in the following manner:

```
<?php
$file_ptr = fopen("XYZ.txt", "w");
if (flock($file_ptr, LOCK_EX)) {
print "Got lock!\n";
sleep(10);
flock($file_ptr, LOCK_UN);
}
?>
```

File locking in PHP requires a complete unconventional approach. It cannot be done in the original Microsoft's FAT file system's version used on Windows 95 and 98. Both NTFS and FAT32 can work for file locking in PHP. All the processes themselves get locked in PHP by default. The example below can focus on the concept:

```
<?php
$file_ptr = fopen("XYZ.txt", "w");
if (flock($file_ptr, LOCK_EX)) {
print "Got lock!\n";
sleep(10);
flock($file_ptr, LOCK_UN);
}
?>
```

8.7. Miscellaneous shortcuts

With a comprehensive knowledge on opening and writing of files, you must now be confident enough to attempt different activities with a file. Here are some miscellaneous shortcuts for PHP and the related concepts.

Using these shortcut methods, you can perform different file manipulation activities in PHP. One thing you should necessarily know is that all the shortcut methods require PHP5.

Instead of `fopen()` and `fread()`, you can simply use file_get_contents to read and open a file. You only require the file name. By giving a file name, you will receive a series of related contents. See the code below:

```
<?php
$file_contents                                          =
file_get_contents(&apos;XYZ.txt&apos;);
?>
```

PHP also offers a shortcut for writing on a file. Instead of following the detailed process, you can apply the shortcut method for `fwrite` mentioned below:

```
<?php
file_put_contents('XYZ.txt', 'Hello World!');
```

Saving state in PHP

HTTP is considered a stateless protocol. Hence, whenever you download a page from your server, it presents a separate connection. There must be some process by which information stored in one page can be accessed by subsequent pages. Here we will see some of these processes in brief.

9.1. Setting a cookie

In PHP, a cookie is set in the following ways:

setcookie() function: As the name suggests, setcookie() outputs a header and is used before sending any content to the browser. The function has certain optional and essential attributes. Cookie value, expiry date in UNIX epoch format, domain, path and integer are the optional attributes. Only the cookie name is the essential attribute of the setcookie() function.

Example:

```
<?php
$cookie_val ="We are testing cookies";
setcookie("OurCookie", $cookie_val);
setcookie("OurCookie", $cookie_val, time()+3600);
if (isset($_COOKIE['OurCookie']))
{
echo $_COOKIE["OurCookie"];
}
?>
```

Here, either statement in line 2 or line 3 can be used to set the cookie. The value assigned to a cookie variable can also be accessed from other pages. The example given below displays the cookie created in the above example:

Example:

```
<?php
echo $_COOKIE["OurCookie"];
echo $HTTP_COOKIE_VARS["OurCookie"];
?>
```

Cookies can also be set in array as given below:

```
<?php
setcookie("owncookie[a]", "First Cookie");
setcookie("owncookie[b]", "Second Cookie");
setcookie("owncookie[c]", "Third Cookie");
if (isset($_COOKIE['owncookie'])) {
foreach ($_COOKIE['owncookie'] as $cookiename =>
$Ourcookie) {
echo "$cookiename : $Ourcookie <br />\n";
}
}
?>
```

`print_r($_COOKIE);` can be used to display all the available cookies

9.2. Deleting a cookie

For deleting a cookie, `setcookie()` is called with the name argument like `setcookie("Ourcookie")`. However, this argument does not work always. Hence, it is advised to set the cookie with an already expired date. Look at the delete example below:

Example:

```
<?php
setcookie("Ourcookie ", "", time()-3600);
?>
```

9.3. Creating session cookie

For creating a cookie that is valid till the user runs his/her browser, you can pass setcookie() with an expiry argument of 0. In such a case, your browser continues to run and cookies are returned to the server. Once the browser quits and restarts, it does not remember them.

This is useful for the scripts validating a user with a cookie. This also enables you to have regular access to personal information on different pages, once the password is submitted.

See the syntax below:

```
setcookie( "session_id" , "66343" , 0 ) ;
```

9.4. Working with query string

A query string plays a pivotal role in making web applications. A cookie file is completely dependent on the client when the complete function depends on frequent accessibility of users. When a form is submitted using the GET method, the fields and values are encoded with a URL and the filled form is sent. A form with two fields including user_id and name ends up like, http://www.oursite.co.in/qstring.php?name=abcd&user_id=xyz+pqr.

Here, every name and value is divided by an equal (=) operator. The names and value pair are separated by an ampersand sign (&). In PHP, the strings are decoded and pairs are available in the $HTTP_GET_VARS array. You access the user_id using the GET parameter. You can use the GET array as:

```
$HTTP_GET_VARS[user_id];
```

Creating a query string:

A query string can be created with a URL for encoding the keys and values. Suppose you have to pass a URL to another page as a query string, then a forward slash and the colon sign appear ambiguous to the parser. Therefore, convert the URL into hexadecimal characters. To do this, use PHP's urlencode() function. This accepts a string as an argument and returns an encoded copy such as `print urlencode(http://www.oursite.co.in);`

```
// prints http%3A%2F%2Fwww. oursite.co.in
```

Look at the example on query string made from two variables:

Example:

```
<?php
$name = "john";
$ourpage = "http://www.oursite.co.in";
$query_string = "ourpage=".urlencode( $ourpage );
$query_string .= "&name=".urlencode( $name );
?>
<a href="ourpage.php?<?php echo $query_string ?>">Go</a>
```

To dynamically make a query string, use the http_build_query () function.

Function to create string:

```
<html>
```

```
<head>
<title>We are learning Query String</title>
</head>
<body>
<?php
$arr_query = array (
'prod_id' => "P001",
'prod_name' => "New Produst",
'homepage' => "http://www.oursite.co.in/"
);
$query_string = http_build_query( $arr_query );
print $query_string;
?>
<p>
<a href="ourpage.php?<?php print $query_string
?>">Move!</a>
</p>
</body>
</html>
```

9.5 Session function

Session functions provide a unique identifier. This can be used to acquire information from accessing points. The cookies are also used in the session function by default. Usually the session states are stored in a temporary file.

The `session_start()` function creates session or resumes the current one based on the current session id that's being passed via a request, such as `GET`, `POST`, or a cookie. Session id can be accessed using the session_id() function.

Usually `session_start()` function returns true. While using cookie-based sessions, we can use the session_start() before outputting anything to the browser. Look at the following example:

Example:

```
<?php
session_start();
echo 'We are in ourpage';
$_SESSION['name'] = 'John';
$_SESSION['add'] = 'Kolkata';
```

```
$_SESSION['time'] = time();
echo '<br /><a href="ourpage.php">page 2</a>';
echo '<br /><a href="ourpage.php?' . SID .
'">page 2</a>';
?>
```

The page `ourpage.php` contains the session data when this page will be navigated through the above example.

9.6. Session variables

Sessions in PHP are like the server side cookie files. It stores variables. PHP scripts can be read as well as these stored variables can be written. Session files are created as per user requests. These files can only be accessed on the request of the same user. For example, take an HTML form with a user name and occupation. This HTML form transmits the data to other pages with a session file.

The new data page comprises of one HTML form requesting a user to enter his/her name and occupation. The details are then passed as name-value pairs, called $name and $job to a PHP page. The PHP pages store all the information as session variables. The first part of the code on the HTML page and all other pages is required for accessing the following variables: `<?php session_start(); ?>`

The code mentioned above, basically has two functions and performs as per your request. If you do not have a session, it creates a new session. It connects to the existing session file, if you have a session. After a new session is created, a session identifier is generated by the PHP session management. This session identifier is a string comprising of 32 hex digits. It creates an empty session file on the server as 'sess_' followed by the session identifier. In turn, it creates a set-cookie and a session cookie in the browser according to the session identifier value.

Any request by the user to the server includes the session identifier with PHP connecting to the exact session file. Going back to the HTML form page, an HTML form is created on the user's browser. The user fills in the required details and clicks on the send button. The filled form with variables is sent to the PHP page storing the variables. Now the code appears as follows:

```
<?php
session_start(); // It connects to the existing session or starts a new session
session_register ("id"); // It creates a session variable called id
session_register ("prod"); // It creates a session variable called prod
$HTTP_SESSION_VARS ["id"] = $id; // It sets value of id by variable $id
$HTTP_SESSION_VARS ["prod"] = $prod; // It sets value of job by variable $prod
?>
```

The code mentioned above first connects to the session that is already present. It now creates two session variables with values set from the HTML form. As a PHP variable is added to a session file, it uses session_register() function. Only the variable name is written and the code appears $HTTP_SESSION_VARS("id") = $id is used.

When PHP variables are referred in a session file, the dollar ($) sign is not used. It is only used when the variables are used in the script. Once a session variable is registered, it is used like a normal PHP variable. A script with an updated session variable does not require an updated session file. Session management automatically does this with the end of the script. Therefore the script would appear as:

```
$name = "myname";
```

The new value is automatically written to the session file. To create a new session variable, the following code can be used:

```
session_register ("variable_name");
$variable_name = "newvar";
```

The above mentioned variable is available to all PHP pages connecting to the session that uses session_start(). Since the values of the form variables are stored, any PHP page that connects to the session, can read the variables.

To destroy a session file, use the following function:

```
session_destroy();
```

The following example tells you how to access registered variables:

Example:

```
<?php
session_start();
?>
<html><head>
<title>Accessing session variable</title>
</head>
<body>
<?php
echo "Value of session variable:\n\n";
echo $_SESSION[variable_name];
?>
</body>
</html>
```

Advanced PHP

Let's go through some of the advanced features of PHP like Date, Secure E-mail, Include, E-mail, Error, PHP Filter and PHP Exception:

10.1. Date

In PHP the `date()` function is used to format a timestamp or a date. It arranges a timestamp into a readable date and time. Using the date/ time functions, you can format date and time on the server. However, these functions entirely depend on the server settings. Here you can use the syntax, date (format, timestamp). Look at the table below:

Parameter	Description
Format	This parameter is essential. It assigns the timestamp format.
Timestamp	This Parameter is optional. This takes the Date or/and time that you want to format. If no value is provided then the current time is used for formatting.

In PHP, timestamp is the number of seconds since January 1, 1970 at 00:00:00. This is also termed as, 'Unix Timestamp'.

In the date function, the first parameter specifies about formatting date and time. Several letters are used to represent date and time formats. Some commonly used letters are given below:

- d - Represents day of a month (01-31)
- D - Represents day in three letter text format
- m - Represents month, as a number (01-12)
- M - Represents month in three letter text format
- Y - Represents year in four digits
- y - Represents year in two digits

Some other frequently used characters are "/", ".", or "-" etc. Basically, these characters are inserted between letters to add additional formatting. Look at the code below:

Example:

```
<?php
print date("y/m/d");
print "<br />";
print date("Y.M.D");
print "<br />";
print date("d-m-y");
?>
The output of this code is as follows:
08/11/14
2008.Nov.Fri
14-11-08
```

A timestamp can be added in PHP date. In the date() function, the second parameter specifies the timestamp. Since this is an optional parameter, even if you do not supply a time stamp, the current time is automatically used.

Let us now discuss about the mktime() function. It is used to create a timestamp for the following day. The mktime() function returns the Unix timestamp for a particular date. The syntax is as follows:

```
mktime(hour,minute,second,month,day,year,is_dst)
```

Here all the arguments are integers.

To count a day in the future, add one to the day argument of mktime(). Look at the code below:

Example:

```
<?php
$nextweek = mktime(0,0,0,date("m"),date("d")+7,date("Y"));
echo "Next ".date(l)." will be on ".date("d/m/Y", $nextweek);
?>
```

The output of the above code is:

```
'Next Friday will be on 21/11/2008'.
```

Since date/time functions are the parts of PHP core, no installation is required to use this function.

10.2. Include

'Server Side Includes' are used to create functions, headers, footers and certain other elements which can be reused on multiple pages. You can insert file content into a PHP file by using either the include() or require() function. Both the include() and require() functions are identical. The only difference lies in the way they handle errors. While the include() function generates a warning, the require() function generates a fatal error.

This feature of 'include' is beneficial on the part of the developer, as it saves considerable amount of time. You can also create a standard header or menu file to include in all web pages. If you want to update the header, simply update one include file. You can also change the menu file to add a new page to your site.

Suppose you have a standard PHP file named "firstpage.php". You can include this file in a page by using the example below:

Example:

```
firstpage.php
<?php
$name = "John";
$address = "kolkata";
?>
secondpage.php
<?php
include("firstpage.php");
echo "Your name is ".$name." and you live in ".$address;
?>
```

In the above example, the "firstpage.php" file with all its contents is included in "secondpage.php". To use a standard menu file in all pages, you can use the include() function. Let the code for creating menu be written in a file named "menu.php". Now you have to include this file in all the pages using include() function.

require():

When a file is included with the include() function, it returns an error message as shown in the following example:

Example:

```
<html>
<body>
```

```
<?php
include("notexist.php");
echo "Why are you doing this!";
?>
</body>
</html>
```

Error message:

```
Warning: main(notexist.php) [function.main]: failed to open stream: No such file or directory inc:\wamp\www\php_check_bs\adv6.php on line 4
Warning: main() [function.include]: Failed opening 'notexist.php' for inclusion (include_path='.;C:\php5\pear') in c:\wamp\www\php_check_bs\adv6.php on line 4
Why are you doing this!
```

Let us use the above example with the require() function:

Example:

```
<html>
<body>
<?php
require("notexist.php");
echo "Why are you doing this!";
?>
</body>
</html>
```

Error Message:

```
Warning: main(notexist.php) [function.main]: failed to open stream: No such file or directory in c:\wamp\www\php_check_bs\adv6.php on line 4
Fatal error: main() [function.require]: Failed opening required 'notexist.php' (include_path='.;C:\php5\pear') in c:\wamp\www\php_check_bs\adv6.php on line 4
```

The echo statement has not been executed here as the script execution has been stopped after the fatal error. It is advised to use the require() function in place of include() because the scripts cannot be repeatedly executed if the files are missed or miscalled.

10.3. E-mail

With PHP you can send e-mails directly from a script. With mail(), the messages are automatically mailed to the specified receiver. Similar mails can be forwarded to multiple recipients by putting a comma in the 'to' column between the addresses. This function can be applied to send some special content types as well as emails with attachments.

This function takes receiver(s) email-id, subject, content of mail, additional headers like Cc, Bcc, etc. and additional parameters as arguments. The last two arguments are optional.

On successful delivery of the mail, mail() returns TRUE, otherwise a FALSE value is returned.

Example:

```
<?php
mail("abc@ourmail.com", "urgent", "Hello\nHow are you");
?>
```

To pass a fourth string argument, you can insert it at the end of the header. You can also add extra headers by using this function. Multiple headers can also be added. Here we need to use a carriage return and a new line to separate each header from the other. While sending the mails we must separate the headers using \r\n.

Following code can be used to send mail with extra headers:

```
<?php
$receiver = "myfriend@ourmail.co.in";
$subject = "wish";
$content = "Hi! My dear friend how are you.";
$sender = "myself@ ourmail.co.in ";
$headers = "From: $sender";
mail($receiver,$subject, $content,$headers);
echo "Mail has been sent successfully.";
?>
```

To send mail with extra headers and additional command line parameter use the code below:

```
<?php
mail("myfriend@ourmail.co.in ", "$subject", $content,
```

```
$header, "- fwebmaster@ourmail.com");
?>
```

To create some complex email messages you can also use simple string building techniques:

```
<?php
$receiver = "mary@example.com" . ", " ;
$receiver .= "kelly@example.com";
$subject = "Important day";
$content = "
<html>
<head>
<title> important days </title>
</head>
<body>
<p> Some important days!</p>
<table>
<tr>
<th>Event</th><th>Day</th><th>Month</th>
</tr>
<tr>
<td>Republic day</td><td>26th</td><td>January</td>
</tr>
<tr>
<td>May day</td><td>1st</td><td>May</td>
</tr>
<tr>
<td>Independence day</td><td>15th</td><td>August</td>
</tr>
</table>
</body>
</html>
";
$headers = "MIME-Version: 1.0\r\n";
$headers .= "Content-type: text/html; charset=iso-8859-1\r\n";
$headers .= "To: raja<raja@ourmail.com>, rahim<rahim@ ourmail.com >\r\n";
$headers .= "From: Event Reminder <remind@our-
```

```
mail.com>\r\n";
   $headers .= "Cc: john@ourmail.com\r\n";
   $headers .= "Bcc: karan@ourmail.com\r\n";
   mail($receiver, $subject, $content, $headers);
   ?>
```

While using the code be sure you do not use any new line characters in the 'to' or subject field, else the mail may not be sent.

10.4. Secure email

Secure email is another characteristic of PHP. You can keep your mails secure from others for opening and inserting data. This can be done by validating inputs. See the code below:

```
<html>
<body>
<?php
if (isset($_REQUEST['mail']))
{
$mail = $_REQUEST['mail'] ;
$sub = $_REQUEST['sub'] ;
$msg = $_REQUEST['msg'] ;
mail("myfriend@ourmail.co.in", "Subject: $sub",
$msg, "From: $mail" );
echo "mail is ready";
}
else
{
echo "<form method='post' action='ourmail.php'>
mail-id: <input name='mail' type='text' /><br />
Subject: <input name='sub' type='text' /><br />
Content:<br />
<textarea name='msg' rows='10' cols='50'>
</textarea><br />
<input type='submit' />
</form>";
}
?>
</body>
</html>
```

The above code is not secure, as even an unauthorised user can insert data into the mail headers through the input form. Suppose the user adds the text given below into the email input field within a form:

```
myfriend@ourmail.co.in%0ACc:newfriend1@ ourmail.co.in
%0ABcc: newfriend2@ourmail.co.in, newfriend3@ourmail.co.in,
anotherfriend1@ourmail.co.in, anotherfriend2@ourmail.co.in
%0ABTo:anotherfriend3@ourmail.co.in
```

The above text is put into the mail headers by the mail() function. Now the new header comprises of extra Cc:, Bcc:, and To: field. As the user clicks on the submit button, the email is sent to the supposed addresses.

The following is a piece of code which checks whether the email address that was provided is a valid email address or not:

```
<html>
<body>
<?php
function mailcheck($check_mail)
{
$check_mail = filter_var($check_mail, FILTER_SANITIZE_EMAIL);
if (filter_var ($check_mail, FILTER_VALIDATE_EMAIL))
{
return TRUE;
}
else
{
return FALSE;
}
}
if (isset($_REQUEST['mail']))
$validate_mail = mailcheck($_REQUEST['mail']);
if ($validate_mail ==FALSE)
{
```

```
echo "Can't send ";
}
else
{
$mail = $_REQUEST['mail'] ;
$sub = $_REQUEST['sub'] ;
$msg= $_REQUEST['msg'] ;
mail("myfriend@ourmail.co.in", "Subject: $sub",
$msg, "From: $mail" );
echo "mail is ready";
}
}
else
{
echo "<form method='post' action='ourmail.php'>
Email: <input name='mail' type='text' /><br />
Subject: <input name='sub' type='text' /><br />
Message:<br />
<textarea name='msg' rows='10' cols='50'>
</textarea><br />
<input type='submit' />
</form>";
}
?>
</body>
</html>
```

The PHP filters have been used to validate input in the above example:

- All illegal e-mail characters from a string are removed by the FILTER_SANITIZE_EMAIL.
- The FILTER_VALIDATE_EMAIL filter validates value as an e-mail address.

10.5. Error

In PHP, default error handling is very simple. It returns an error message with line number, filename and a message indicating that the error is sent to the browser. Error handling plays an important role when creating scripts and web applications. A program with no error checking code appears very unprofessional.

The program also remains open to security risks.

There are three different error handling methods which include:

- Simple "die()" statements
- Custom errors and error triggers
- Error reporting

Simple "die()" statements:

The example below shows a simple script for opening a text file:

```
<?php
$fptr=fopen("newfile.txt","r");
?>
```

When a file does not exist, it returns an error message like:

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

To avoid the error message, test the existence of the file before accessing it. Use the code below:

```
<?php
if(!file_exists("newfile.txt"))
{
die("File not exists");
}
else
{
$fptr=fopen("newfile.txt","r");
}
?>
```

It returns an error message 'File does not exist' if the file does not exist. In the above code, a simple error handling mechanism has been used, which stops the script after the error occurs. PHP offers some alternative error handling functions such as:

Creating a custom error handler:

It is easy to create a custom error handler. You can simply create a special function and use it just as an error occurs in PHP. This func-

tion basically handles two parameters including error level and error message. It accepts around five parameters optionally including file, line-number, and error context. Look at the syntax below:

```
error_handler(error_level,error_message,error_file,error_line,error_context)
```

Parameter	Description
error_level	Essentially required to specify the error report level for the user-defined error.
error_message	Essential for indicating the error message for the user- specific error.
error_file	Completely elective. It indicates the filename in which the error occurred.
error_line	Elective option. It targets the line number where the error occurred.
error_context	Also an elective option. It indicates to an array that contains variable and their values. The array is used at the time when an error occurs.

Error report levels:

The error report levels are the collections of different types of errors. The user defined error handlers are used for the following purposes:

Value	Constant	Description
2	E_WARNING	Non-fatal run-time errors. Script execution is not halted.
8	E_NOTICE	Run-time notices. The script found something that might be an error. It could also happen while running a script normally.
256	E_USER_ERROR	Serious user-generated error. This is like an E_ERROR arranged by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Simple user generated warning like E_WARNING arranged by the programmer with the help of PHP function trigger_error()

1024	E_USER_NOTICE	User-generated notice like E_RROR. It can be traced by a user defined handle
4096	E_RECOVERABLE_ERROR	Traceable serious error like an E_ERROR. It can be traced by a user defined handle.
8191	E_ALL	All errors and warnings leaving E_STRICT.

Let us create a function to handle errors:

```
function error_handler($error_number, $error_msg)
{
echo "<b>Error:</b> [$error_number] $error_msg
<br />";
echo "End of error message";
die();
}
```

The above mentioned code is a simple error handling function. As it is activated, it gets the error number and an error message. It terminates the script. As you create an error handling function, you need to decide when to trigger it.

Set error handler:

In PHP the default error handler is the built in error handler. These can be changed for applying it to some errors. This way, the script can handle different errors in various ways. Look at the code below:

```
set_error_handler("customError");
```

In the above example, the custom error handler has been used for the errors. The set_error_handler() requires one parameter. However, to specify an error level a second parameter can be added.

Example:

```
<?php
function error_handling($error_number,
$error_msg)
{
echo "<b>Error:</b> [$error_number] $error_msg ";
}
set_error_handler("error_handling ");
```

```
echo($error_test);
?>
```

'Error: [8] Undefined variable: error_test' will be the output of the above code.

Trigger an error:

To input an error in a data, it is necessary to trigger errors just as an illegal input occurs. This is done by the trigger_error() function in PHP. See the example below:

Example:

```
<?php
$val=110;
if ($val>100)
{
trigger_error("Number is greater than 100");
}
?>
```

The output of the above code is:

```
Notice: Number is greater than 100
```

An error can be traced anywhere in a script. With the help of a second parameter you can specify the error level. Some of the common error types are discussed below:

- E_USER_ERROR - Serious user-generated run-time error. It can not be easily recovered. The script execution is halted.
- E_USER_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted
- E_USER_NOTICE - Default. User-generated run-time notice.

The example below is relevant to our discussion above:

Example:

```
<?php
function error_handling($error_number, $error_msg)
{
echo "<b>Error:</b> [$error_number] $error_msg<br />";
echo "We are in error handler";
die();
```

```
    }
    set_error_handler("error_handling",E_USER_WARN-
ING);
    $val=220;
    if ($test<200)
    {
    trigger_error("Number is greater than
200",E_USER_WARNING);
    }
    ?>
```

The output of this code is:

```
Error: [512] Number is greater than 200
We are in error handler
```

Error Logging:

We discussed about creating errors and triggering them. Let us now discuss about error logging. In PHP, an error log is passed by default to the servers logging system or a file. This depends on the manner in which the error_log configuration is set in the php.ini file. With the help of error_log() function error logs can be sent to a specific file or destination.

Send an error message by email:

You can also send error messages by e-mail. In fact this is one feasible way of notifying some specific errors. See the example below:

Example:

```
    <?php
    function error_handling($error_number,
$error_msg)
    {
    echo "<b>Error:</b> [$error_number] $error_msg<br
/>";
    echo "We are in error handler";
    error_log("Error: [$error_number] $$error_msg",1,
    "myfriend@ourmail.co.in","From: allfriend@our-
mail.co.in");
    }
    set_error_handler("error_handling",E_USER_WARN-
ING);
    $val=220;
```

```
    if ($val>200)
    {
    trigger_error("Number is greater than
200",E_USER_WARNING);
    }
    ?>
```

Caution: *The error_log function should not be used with all errors. Frequent errors should be logged on the server using the default PHP logging system.*

10.6 PHP exception

'PHP Exception' was introduced in PHP5. By using the 'Exception' model, PHP changes the normal flow of a script if a specific error (mismatched condition) takes place. This feature of PHP saves the current code state. Here, it stops the program execution and executes a custom exception handler function. It can also terminate the program execution if necessary. In this case, it starts to go on with the script from a different location in the code.

Example:

```
<?php
function test($val) {
     if (!$val) {
         throw new Exception('Division by zero.');
     }
     else return 1/$val;
}
 try {
     echo test(12) . "\n";
     echo test(0) . "\n";
} catch (Exception $e) {
     echo 'Throw Exception: ', $e->getMessage(),
"\n";
    }

    echo 'LearningException';
    ?>
```

The output of the above program is:

```
0.0833333333333 Throwexception: Division by zero.
LearningException
```

Let's look at some of the in built functions of PHP that are used in the 'Exception handling':

- Exception::__construct: This function is used to construct various exceptions.
- Exception::getMessage: This function is used to get the Exception message.
- Exception::getCode: This function provides the Exception code.
- Exception::getFile: This function is used to get the file where the exception occurred.
- Exception::getLine: This function is used to get the line where the exception occurred.
- Exception::getTrace: This function is used to get the stack trace.
- Exception::getTraceAsString: This function is used to get the stack trace as a string.
- Exception::__toString: This function is used for the string representation of the exception.
- Exception::__clone: This function is used to get the clone of the exception.

10.7 PHP filter

As the name suggests, 'PHP filters' are used to filter the data collected from unauthentic sources. The user inputs and the data collected from different web services are often harmful for web application. These external data are not secure. Besides filtering, the PHP filters validate and test the custom data and the user inputs.

The external data filtered by the PHP Filters includes cookies, web-server variables, database query results, form elements and the web service data.

In PHP we have two types of Filters:

- Validating Filters: These are used to validate the user inputs. It returns the desired category as true value. A 'False' value is returned if it is not successful.
- Sanitizing Filters: These are used either to allow or disallow a specific character in a strings. This function always returns the string.

PHP provides us various in-built filter functions, such as:

- filter_var()
- filter_var_array()
- filter_input
- filter_input_array
- filter_has_var()
- filter_id()
- filter_list()

filter_var()

The filter_var() function is used to filter a single variable with a specific filter. Look at the syntax of filter_var() below:

```
filter_var(variable, filter, options)
```

Example:

```
<?php
if(!filter_var("myfriend@ourmail...co.in",   FIL-
TER_VALIDATE_EMAIL))
{
echo("E-mail is not valid");
}
else
{
echo("E-mail is valid");
}
?>
```

The output of the above program is:

```
E-mail is not valid
```

filter_var_array()

The 'filter_var_array()' function is used to filter various variables by using the same or some other filters. Look at the syntax of the function:

```
filter_var_array(array, args)
See the following example:
Example:
<?php
$our_array = array
(
"prod_id" => "p001",
"price" => "500",
"comp_mail" => "newcompany@ourmail.com",
```

```
);
$array_filter = array
(
"prod_id"=> array
(
"filter"=>FILTER_CALLBACK,
"flags"=>FILTER_FORCE_ARRAY,
"options"=>"ourwords"
),
"price" => array
(
"filter"=>FILTER_VALIDATE_INT,
"options"=>array
(
"low_price"=>200,
"high_price"=>1000
)
),
"comp_mail"=> FILTER_VALIDATE_EMAIL,
);
print_r(filter_var_array$our_array, $array_fil-
ter));
?>
The output of the above program is:
Array
(
[name] => p001
[age] => 500
[email] => newcompany@ourmail.com
)
```

filter_input

The 'filter_input()' function is used to get and filter one input variable. Look at the syntax of this function:

```
filter_input(input_type, variable, filter,
options)
```

Example:

```
<?php
if (!filter_input(INPUT_POST, 'email',
FILTER_VALIDATE_EMAIL))
{
```

```
echo "Invalid mail";
}
else
{
echo "valid mail ";
}
?>
```

filter_input_array

The 'filter_input_array()' function is used to get and filter different variables of input by using various filters. Look at the syntax of this function:

```
filter_input(input_type, args)
```

filter_has_var()

The 'filter_has_var()' function is used to verify the existence of a specified variable. Look at the syntax of this function:

```
filter_has_var(type, variable)
```

Look at the following example:

Example:

```
<?php
if(!filter_has_var(INPUT_GET, "roll"))
{
echo("Roll does not exist");
}
else
{
echo("Roll exists");
}
?>
```

filter_id()

The 'filter_id()' function is used to return the filter ID of a named filter. Look at the syntax of this function:

```
filter_id(filter_name)
```

Note the following example:

Example:

```
<?php
echo(filter_id("validate_email"));
?>
```

filter_list()

The 'filter_list()' function is used to return a catalog of various supported filters. Look at the syntax of this program:

```
filter_list()
```

Example:

```
<?php
print_r(filter_list());
?>
```

Supplementary filtering options can be added by using 'Options' and 'Flags'.

PHP and databases

11. PHP and databases

PHP has become widely popular due to its capacity to use varied and powerful database systems. Website content is made dynamic, interactive and flexible with the help of a database.

Databases are collection of data, stored separately in such a manner that you can easily recover it. Database information is stored in table format. The tables are divided into rows and columns, separating the data uniformly. Each row in a table represents a single record like name and address. In each column of a table, a uniform record is maintained like the first name of an individual and his/her contact number. A database in such format arranges a record according to the values available in the column exactly like a spreadsheet program.

In this way, you can easily retrieve a record from a database without keeping in mind how the data has been arranged into a table. Basically, most of the database systems use the SQL. There are several Database Management Systems available in the Market. Mysql, Microsoft SQL Server, Oracle are among others. MySQL is the most popular Open Source Database Management System and for this 'Open Source' nature it is widely used among the web applications that use PHP.

PHP performs various inbuilt functions with certain databases such as MySQL database, SQL Server, Oracle and others. PHP supports databases in multiple ways. It supports various databases as:

- Informix
- DBM (Berkeley)
- MSSQL (Microsoft)
- Sybase
- Oracle 8
- PostgreSQL (Berkeley, open source)
- MySQL (Open source)

There are certain databases which are supported by PHP through protocol-based functions. These include:

- ODBC
- LDAP
- DBM style

Let us discuss about each of these databases in detail:

Informix: the Informix driver is employed for Informix (IDS) 7.x, SE 7.x, Universal Server (IUS) 9.x and IDS 2000 in "ifx.ec" and "php_informix.h" in the informix extension directory. IDS 7.x support is finished with a good support for BYTE and TEXT columns. IUS 9.x support is not completed fully. Here we have some new data types except SLOB and CLOB as these supports are not yet completed.

DBM (Berkeley): DBM functions provide certain high performance implementations and source codes relevance, for different interface applications. Apart from this, they cannot be used for any other functions. The DBM applications can be compiled by replacing the application's #include of the DBM or NDBM include file. The lines mentioned below focus on this concept:

```
#define DB_DBM_HSEARCH 1
#include <db.h>
```

'dbm datum typedef' basically describes two objects including key and content. A dbm data focuses on a series of dsize bytes presented by the dptr. It also includes binary data as well as text strings. A dbm database can be opened by the dbminit function. This helps in opening and creating a database file.db. The open database can be closed by calling dbmclose.

MSSQL (Microsoft): PHP uses MySQL server in a stack of software such as LAMP or WAMP stacks. In some occasions, PHP uses MSSQL databases in the back-end. With the PHP scripts certain web applications are also made, exposing the MSSQL Databases. The MSSQL, a relational database management system, was produced by Microsoft. MS-SQL and T-SQL are the two query languages of MSSQL.

Sybase: Sybase provides a detail introduction about Sybase SQL Server. It also offers the system model and some tools and components of the Sybase System 11. It is a client server database engine.

With Sybase installed on your local drive, you can focus more on writing application than writing data access and security code.

Oracle 8: Oracle Corporation released 'Oracle 8' in 1999, aiming to provide a much advanced database system. The main function of Oracle 8 is to design and develop different database objects like synonyms, indexes, views and tables. With Oracle 8, databases are installed into any Oracle 8 environment without any modifications. The release of this version of Oracle 8 offered a database supporting different multimedia applications and object oriented development.

PostgreSQL(Berkeley, open source): PostgreSQL is an important database with some potent characteristics. It helps in transaction support, presentation, performance, and industrial-strength monitoring. PostgreSQL is an open source database and is often arranged with Linux. It does not offer support for Java procedures. At present, more than 21 percent of PostgreSQL users combine it with Windows or Cygwin environments.

Advanced training is not required to learn PostgreSQL. With PostgreSQL, you can collect a composite knowledge about commands and other basic features. Manipulate and update databases, application of joins, customize queries, consider SQL aggregates, exercise PostgreSQL query tools, combine SELECTs with sub queries, work with triggers and transactions, import and export data, can also be done here.

MySQL (Open source): MySQL was released in the middle of 1996. It is a famous, open source database management system used for Unix and Linux. While surfing the MySQL database, you can find an assortment of power and functionality. Simple command sets used for inserting, recovering, updating and deleting a data can be used to develop some complex databases and tables.

This database system supports different connection methods including Unix Sockets, TCP/IP sockets and named pipes for Windows NT/2000. MySQL can be downloaded free of cost and server passwords can be allocated by it. It comprises of all the tools required to get started. It is one of the most stable database management systems in the market. One of its ISAM table format created during the late eighties is one of the important table formats in MySQL. MySQL is easily accessible and can be manipulated from

a variety of popular programming languages. When MySQL was written, it was composed in C and C++, and is wholly optimized for both the Unix and Win32 platforms. MySQL also uses memory hash tables, kernel threads, certain high optimized individual collected class libraries and thread based memory portions.

MySQL supports different field types including CHAR, FLOAT, DOUBLE, DATE, VARCHAR, TEXT, SET, BLOB and ENUM. Certain advanced querying and grouping functions such as, COUNT(), GROUP BY and ORDER BY, STD(),AVG(),MAX(),MIN() and SUM() are also supported here.

11.1 Database concept

The term 'data' suggests a record with certain necessary information. A database in computer is a structured record collection. They are stored in computer systems. A database structure is prepared by arranging a data in a database model patterns. There are three types of databases that are frequently used:

- Relational model
- Hierarchical model
- Network model

A database is organised in a computer with a database management system. Using this software, a computer performs different related functions. These include recovering data, storing, adding, deleting and modifying data.

An extensive part of a database depends on different managing factors like integrity, presentation, concurrency and recovery from hardware failures. We can divide a database management system into two categories - desktop databases and server databases. The desktop databases are generally targeted towards individual user application. The server databases target towards the authenticity and uniformity of a data.

A database is basically of two types - flat file and relational database.

Flat file: Flat file databases usually store small amounts of data. They are easily read and edited. Basically, they are arranged in a series and are accordingly analysed. Flat files are best to store sim-

ple data types. They may become complicated if you store complex data structures.

Relational database: The relational model is the most commonly used database in the present scenario. MySQL, Microsoft SQL Server and Oracle are best examples of relational databases. In a relational database, tables are used to represent some interlinked objects. In the relational model, databases are arranged for maintaining integrity.

11.2 Database connection:

A database connection is a system that establishes a connection between the client and database software. This connection helps in sending commands and getting answers in a result set. Basically, there are two sources of database connection - data source and driver manager.

PHP MySQL database connection:
In the following code fragment, you can see mysql_connnect() connects to the MySQL database server.

```
$ con=mysql_connect("localhost", "root", "password" );
   If (! $con )
   Die( "Couldn't connect database" );
```

Persistent database connection is also a connection method designed to represent some regular connections. A persistent connection is often used as a substitute for a non persistent connection. A non-persistent connection may bring change in the script efficiency but not in its performance.

You can also establish a database connection by using PEAR DB:

```
   <?php
   require_once 'DB.php';
   $dbh                                                    =
DB::connect("mysql://test@localhost/test");
   if (DB::isError($dbh)) {
   print "Connect failed!\n";
   print "Error message: " . $dbh->getMessage() .
"\n";
```

```
   print "Error details: " . $dbh->getUserInfo() .
"\n";
   exit(1);
   }
   print "Connect ok!\n
```

Once you are connected to the database server, select the database to be used. This database should be accessible from your username.

11.3 Creating tables

In order to create tables at first we have to create database. The following code will create a database named "ourdatabase".

```
   <?php
   $con=mysql_connect("localhost","root");
   $create_db=" CREATE DATABASE ourdatabase;";
   mysql_query($create_db,$con) or die ("can not
create database or database already exist");
   ?>
```

After creating the database, select the database and create table(s) in the database as per requirement. The example below will create a table named "ourtable" having five fields:

```
   <?php
   $con=mysql_connect("localhost","root");
   mysql_select_db("ourdatabase",$con) or die("can
not select database or database does not exist");
   $create_table="CREATE TABLE ourtable (
   roll INT NOT NULL ,
   name VARCHAR(50) NOT NULL ,
   address VARCHAR(100) NOT NULL ,
   ph_no VARCHAR(12) NOT NULL ,
   grade VARCHAR(2) NOT NULL ,
   PRIMARY KEY (roll)
   ) ;";
   mysql_query($create_table,$con) or die ("can not
create table or table already exist");
   ?>
```

11.4 Getting information on database:

We all know that a database is a collection of information. With the help of the mysql_list_dbs() function, you can derive a list of different databases available from the recent database connection. Usually all databases begin from 0. The code written below will provide the entire information about the databases present in a connection including the names of the databases, names of the tables present in the database and the list of fields in the corresponding tables:

```
<?php
$con=mysql_connect('localhost','root')        or
die('Can not connect database');
$db_list=mysql_list_dbs($con);
$num_db=mysql_num_rows($db_list);
while($select_db=mysql_fetch_object($db_list))
{
echo '<b>'."Database Name :".' '.'</b>';
echo '<b>'.$select_db->Database.'</b>'.'<br>';
$db_name=$select_db->Database;
mysql_select_db($db_name,$con) or die('Cant find
database');
$show_tab = "show tables from $db_name";
$tab_list = mysql_query($show_tab);
$num_tab=mysql_num_rows($tab_list);
echo     '<b>'."$select_db->Database    contains
$num_tab Tables".'</b>'.'<br>';
$x=1;
while($tab_name=mysql_fetch_row($tab_list))
{
echo "Table $j : ".' ';
echo "$tab_name[0] <br>"."</a>";
$show_field     =     "show     fields     from
".$tab_name[0].'"';
$field_list    =    mysql_query($show_field)    or
die(mysql_error());
$num_field=mysql_num_fields($field_list);
echo "<table border='1'>";
echo "<tr align='center'>";
for($i=0;$i<$num_field;$i++)
{
echo                                          "<th
```

```
bordercolor='#000000'>".mysql_field_name($field_lis
t,$i)."</th>";
    }
    echo "</tr>";
    while($field_name=mysql_fetch_row($field_list))
    {
    echo "<tr>";
    for($i=0;$i<$num_field;$i++)
    {
    echo "<td bordercolor='#000000'>";
    echo "$field_name[$i]";
    echo "</td>\n";
    }
    echo "</tr>\n";
    }
    echo "</table>";
    $x++;
    }
    echo "<br>";
    }
    ?>
```

11.5 Inserting data to a table

In order to add, change and remove information from a database, you need to use an HTML form. The HTML forms are used to accept user input. Then using `mysql_query()`, the operations on database are performed. Basically, we add data to a database using `SQL INSERT` command. This command can be sent to the database either by using the `Query` method or by `Prepare and Execute` method. Let's see how a data can be inserted using Prepare and Execute method.

In the following code, you can see an HTML page with textboxes and a submit button to accept user input:

```
<html>
<head>
<title>
Add New Record
</title>
</head>
```

```
<body bgcolor="#0099CC">
<form action="newrecord.php" method="post">
<table align="center" bgcolor="#00CC99">
<caption align="center">ADD NEW RECORD</caption>
<tr>
<td align="center">Roll:</td>
<td align="center"><input type="text"
name="roll"></td>
</tr>
<tr>
<td align="center">Name:</td>
<td align="center"><input type="text" name="
name"></td>
</tr>
<tr>
<td align="center">Address:</td>
<td align="center"><input type="text"
name="address"></td>
</tr>
<tr>
<td align="center">Phone:</td>
<td align="center"><input type="text"
name="phone"></td>
</tr>
<tr>
<td align="center">Grade:</td>
<td align="center"><input type="text"
name="grade"></td>
</tr>
<tr align="center">
<table align="center">
<tr>
<td align="center"><input type="Submit"
name="submit" value=Add></td>
</tr>
</table>
</tr>
</table>
</form>
</body>
</html>
```

The above HTML code will receive the information in text boxes. After clicking the submit button, the information will be submitted to the newrecord.php file. This file will then insert the record in ourtable in ourdatabase using the following code:

```
newrecord.php
<?php
$con=mysql_connect("localhost","root");
mysql_select_db("ourdatabase",$con) or die("can
not select database or database does exist");
$roll=stripslashes(trim($_POST['roll']));
$select_record="select * from ourtable where
roll='".$roll."'";
$select_query=mysql_query($select_record) or
die(mysql_error());
if(mysql_num_rows($select_query)==0)
{
$name=stripslashes(trim($_POST['name']));
$address=stripslashes(trim($_POST['address']));
$phone=stripslashes(trim($_POST['phone']));
$grade=stripslashes(trim($_POST['grade']));
$add_new="insert into
ourtable(roll,name,address,ph_no,grade) values
($roll,'$name','$address','$phone','$grade')";
$add_query=mysql_query($add_new) or
die(mysql_error());
if($add_query)
{
echo "New Record Added !!";
}
}
else
echo "Duplicate Roll Not Allowed";
?>
```

It is also possible to create a single file using both HTML and PHP codes to fulfill the above requirements instead of two separate files.

11.6. Retrieving data from a table

We have learnt about adding information to a database. There are certain set strategies to recover inserted information from a database. You can use 'mysql_query ()' to make a select query. Suppose you have a record in your database and you want to output it. The first command for this will be:

```
SELECT * FROM TABLE NAME;
```

Here, TABLE NAME represents the name of the table from where the data will be retrieved.

This is a basic command in MySQL that directs the script to select all the records in the table. As this command outputs a data, it should be placed with the results given to a variable.

```
$query="SELECT * FROM TABLE NAME";
$result=mysql_query($query);
```

The entire content of the table will be stored in the array `$result`.

Retrieving data is also recognised with the term SELECT. The SELECT statement indicates selecting data from a database. See the following syntax:

```
SELECT column_name(s) FROM TABLE NAME;
```

Reconciling the statement above, we can use the `mysql_query()` function. With this function, a query or a command can be sent to a MySQL connection.

Example:

```
<?php
$con=mysql_connect("localhost","root");
mysql_select_db("ourdatabase",$con) or die("can
not select database or database does exist");
$select_query="SELECT * FROM ourtable";
$list = mysql_query($select_query);
while($record = mysql_fetch_array($list))
{
echo $record['roll'] . " " . $record['name']."
".$record['address'] . " " .$record['ph_no'] . " "
.$record['grade'];
echo "<br />";
```

```
}
mysql_close($con);
?>
```

In the following example, the above data is selected and displayed in an HTML table format.

Example:

```
<?php
$con=mysql_connect("localhost","root");
mysql_select_db("ourdatabase",$con) or die("can
not select database or database does exist");
$select_query="SELECT * FROM ourtable";
$list = mysql_query($select_query);
echo "<table border='1'>
<tr>
<th>ROLL</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>PHONE</th>
<th>GRADE</th>
</tr>";
while($record = mysql_fetch_array($list))
{
e c h o
"<tr><td>".$record['roll']."</td><td>".$record['nam
e']."</td><td>".$record['address']."</td><td>"
.$record['ph_no']."</td><td>".$record['grade']."</t
d><td></tr>";
}
echo "</table>";
mysql_close($con);
?>
```

11. 7. Changing data of a table

PHP has provisions for changing information in a table. Changing data and updating data are similar terms. The 'Update' statement intends modifying or changing the present records in a table. Following syntax can be used to change or update a table record:

```
UPDATE TABLE_NAME SET column1=value,
column2=value2,...
```

```
WHERE some_column=some_value ;
```

In the above code the 'WHERE' clause indicates the record to be changed. Look at the example below to gather a practical idea on updating or changing data in an existing table. Let the table `ourtable` contain the following records:

Example:

Roll	Name	Address	Phone	Grade
1	John	Kolkata	9123456789	A
2	Peter	Delhi	9987654321	B
3	Tom	Mumbai	9991234567	A
4	Jimi	Kolkata	9999121345	C
5	Robin	Delhi	9123456999	B

The code given below will change the data of an existing record:

```
<?php
$con=mysql_connect("localhost","root");
mysql_select_db("ourdatabase",$con) or die("can not select database or database does exist");
$res=mysql_query("UPDATE ourtable SET name = 'Andrew' WHERE roll = 2");
if(!$res)
echo "Record not changed";
else
echo "Record changed";
mysql_close($con);
?>
```

The updated table will look like this:

Roll	Name	Address	Phone	Grade
1	John	Kolkata	9123456789	A
2	Peter	Delhi	9987654321	B
3	Andrew	Mumbai	9991234567	A
4	Jimi	Kolkata	9999123456	C
5	Robin	Delhi	9331234567	B

11.8. Deleting data from a table

Sometimes we need to delete existing information, from a database. Using the 'DELETE' statement we can delete a record from a table. The 'DELETE FROM' statement is used to delete records from a database table. This is almost similar to updating a page.

Following syntax is used to delete information from a table:

```
DELETE FROM table_name
WHERE some_column = some_value;
```

You can see the use of the `WHERE` clause in the above syntax. It indicates the record to be deleted from a database. Using the `mysql_query()` function a query or a command is sent to a MySQL connection. Look at the example:

Example:

See, how the records of `ourtable` are deleted in the example below:

```
<?php
$con=mysql_connect("localhost","root");
mysql_select_db("ourdatabase",$con) or die("can
not select database or database does exist");
$res=mysql_query("DELETE from ourtable WHERE roll
= 5");
if(!$res)
echo "Record not deleted";
else
echo "Record deleted";
mysql_close($con);
?>
```

After the information are deleted the fresh table looks like this:

Roll	Name	Address	Phone	Grade
1	John	Kolkata	9282828221	A
2	Peter	Delhi	9876221112	B
3	Andrew	Mumbai	9764646311	A
5	Robin	Delhi	9576564322	B

PHP project

Here we have given a small project in which we will see how a database is manipulated using PHP. At first we have to create a database. In this project the name of the database is "ourdb". We can create the database using the following program:

createdb.php

```
<?php
$host="localhost";
$username="root";
$password="";
mysql_connect($host,$username,$password) or
die("Could not connect database");
if(!mysql_select_db("ourdb"))
{
$createdb="CREATE DATABASE ourdb";
mysql_query($createdb) or die("Can't create
database");
}
else
echo "Database already exists";
?>
```

The above database will be required almost in every file in this project. So we have included a special file in all our project files. The special file is given below:

connection.php

```
<?php
$host="localhost";
$username="root";
$password="";
$databasename="ourdb";
connect=mysql_connect($host,$username,$password)
or die("Cannot Caonnect to database");
mysql_select_db($databasename,$connect) or die
("Cannot find database");
?>
```

In this project we will use two tables. One table will be used to store category details and another to store details of products. The table structures are given below:

tblcatagory

Field	Type	Null	Primary Key
cat_id	int(11)	Yes	Yes
cat_name	varchar(40)	Yes	No

tblproduct

Field	Type	Null	Primary Key
prod_id	int(11)	Yes	Yes
prod_name	varchar(255)	Yes	No
cat_id	int(11)	Yes	No
prod_img_path	varchar(255)	Yes	No
comp_name	varchar(40)	Yes	No
model	varchar(20)	Yes	No
prod_desc	varchar(255)	Yes	No
rate	Float	Yes	No

To create the tables we can use the following program:

createtable.php

```
<?php
include_once "connection.php";
$tblcatagory="CREATE TABLE `tblcatagory` (
`cat_id` int(11) NOT NULL,
`cat_name` varchar(40) NOT NULL,
PRIMARY KEY (`cat_id`)
);";
mysql_query($tblcatagory) or die(mysql_error());
$tblproduct="CREATE TABLE `tblproduct` (
`prod_id` int(11) NOT NULL,
`prod_name` varchar(255) NOT NULL,
`cat_id` int(11) NOT NULL,
`prod_img_path` varchar(255) NOT NULL,
`comp_name` varchar(40) NOT NULL,
`model` varchar(20) NOT NULL,
`prod_desc` varchar(255) NOT NULL,
`rate` float NOT NULL,
PRIMARY KEY (`prod_id`)
```

```
);";
mysql_query($tblproduct) or die(mysql_error());
?>
```

In our project the first page will provide the user some options like add new category, display category details, add new product and display product details. Since this will be the home page its name will be index.php.

Index.php

```
<html>
<head>
<title>Select Your Choice</title>
</head>
<body bgcolor="#BBE8B0">
<h2 align="center">Select Any Option</h2>
<hr align="center" size="4" color="#660060"
width="100%" />
<br />
<p align="center"><a href="AddCatagory.php">Add
New Catagory</a></p>
<br />
<p align="center"><a
href="DisplayCatagories.php">Display all
catagories</a></p>
<br />
<p align="center"><a href="AddProduct.php">Add
New Product</a></p>
<br />
<p align="center"><a
href="DisplayProducts.php">Display all products
details</a></p>
<br />
</body>
</html>
```

This is simple HTML page.

There are some php files used in anchor tag of the above code. These files have special purposes.

The first file that used in anchor tag is AddCatagory.php. This file is used to enter new category. The code is given below:

AddCatagory.php

```
<?php
include_once "connection.php";
if(isset($_POST['Add']) && $_POST['Add']=='Add
Catagory')
{
if($_POST['cat_id']!='' &&
$_POST['cat_name']!='')
{
$cat_id=stripslashes($_POST['cat_id']);
$cat_name=stripslashes($_POST['cat_name']);
$select_catagory="select * from tblcatagory
where cat_name='".$cat_name."' or
cat_id='".$cat_id."'";
$select_query=mysql_query($select_catagory) or
die(mysql_error());
if(mysql_num_rows($select_query)==0)
{
$insert_catagory="insert into tblcatagory
(cat_id, cat_name) values($cat_id,'$cat_name')";
$insert_query=mysql_query($insert_catagory) or
die (mysql_error());
if($insert_query)
{
$msg="New catagory has been added!!";
}
}
else {$msg="Catagory Id or Catagory name already
exists!!";}
}
}
?>
<html>
<head>
<title>Enter New Catagory</title>
</head>
<body bgcolor="#FFCE9D">
<h3 align="center">Add New Catagory</h3>
<hr align="center" size="4" color="#990000"
width="100%" />
<p align="center"><?php echo $msg;?></p>
```

```
<table width="100%" border="0">
<tr>
<td width="28%">
<table width="98%" border="0">
<tr>
<td align="center"><a href="AddCatagory.php">Add
New Catagory</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayCatagories.php">Display
catagories</a></td>
</tr>
<tr>
<td align="center"><a href="AddProduct.php">Add
New Product</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayProducts.php">Display
products</a></td>
</tr>
<tr>
<td align="center"><a href="index1.php"
style="color:#FF0000">Main Page</a></td>
</tr>
</table>
</td>
<td width="72%">
<form method="post" name="frm1">
<table width="50%" border="0" cellspacing="0"
cellpadding="0" align="center">
<tr>
<td width="50%" style="font-size:20px">Catagory
Id:</td>
<td width="50%" align="left"><input type="text"
name="cat_id" maxlength="50" /></td>
</tr>
<tr>
<td width="50%" style="font-size:20px">Catagory
Name:</td>
```

```
<td width="50%" align="left"><input type="text"
name="cat_name" maxlength="50" /></td>
</tr>
</table>
<br /><br />
<table align="center">
<tr>
<td align="center"><input type="submit"
name="Add" value="Add Catagory" /></td>
</tr>
</table>
</form>
</td>
</tr>
</table>
</body>
</html>
```

Next file is used to display the list of categories stored in tblcatagory.

DisplayCatagories.php

```
<?php
include_once "connection.php";
if(isset($_GET['delete']) &&
$_GET['delete']=='yes')
{
$delete_qury1="delete from tblproduct where
cat_id='".$_GET['cat_id']."'";
mysql_query($delete_qury1) or
die(mysql_error());
$delete_qury2="delete from tblcatagory where
cat_id='".$_GET['cat_id']."'";
mysql_query($delete_qury2) or
die(mysql_error());
}
$select1="select * from tblcatagory";
$select_query1=mysql_query($select1) or
die(mysql_error());
?>
<html>
```

```
<head>
<title>Display all catagories</title>
</head>
<body bgcolor="#FFC4C4">
<h2 align="center">Display all Catagories </h2>
<hr align="center" size="4" color="#990000"
width="100%" />
<p align="center"><?php echo $msg;?></p>
<table width="100%" border="0">
<tr>
<td width="28%">
<table width="98%" border="0">
<tr>
<td align="center"><a href="AddCatagory.php">Add
New Catagory</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayCatagories.php">Display
catagories</a></td>
</tr>
<tr>
<td align="center"><a href="AddProduct.php">Add
New Product</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayProducts.php">Display
products</a></td>
</tr>
<tr>
<td align="center"><a href="index1.php">Main
Page</a></td>
</tr>
</table>
</td>
<td width="72%">
<table width="100%" border="1">
<tr>
<td width="12%">Catagory name</td>
<td width="6%">Edit</td>
```

```
<td width="11%">Delete</td>
</tr>
</table>
<table width="100%" border="1">
<?php
while($data1=mysql_fetch_object($select_query1))
{ ?>
<tr>
<td width="12%"><?php echo $data1->cat_name
?></td>
<td width="6%"><a
href="EditCatagories.php?cat_id=<?php echo
$data1->cat_id?>">Edit</a></td>
<td width="11%"><a
href="DisplayCatagories.php?delete=yes&cat_id=<?
php echo $data1->cat_id ?>">Delete</a></td>
</tr>
<?php } ?>
</table>
</td>
</tr>
</table>
</body>
</html>
```

Within another file "EditCatagories.php" is called. This file is written for editing the existing records of tblproduct. User can also delete any record from tblcatagory using the link "Delete".

EditCatagories.php

```
<?php
include_once "connection.php";
if(isset($_GET['delete']) &&
$_GET['delete']=='yes')
{
$delete_qury1="delete from tblproduct where
cat_id='".$_GET['cat_id']."'";
mysql_query($delete_qury1) or
die(mysql_error());
$delete_qury2="delete from tblcatagory where
cat_id='".$_GET['cat_id']."'";
```

```
mysql_query($delete_qury2) or
die(mysql_error());
}
$select1="select * from tblcatagory";
$select_query1=mysql_query($select1) or
die(mysql_error());
?>
<html>
<head>
<title>Display all catagories</title>
</head>
<body bgcolor="#FFC4C4">
<h2 align="center">Display all Catagories </h2>
<hr align="center" size="4" color="#990000"
width="100%" />
<p align="center"><?php echo $msg;?></p>
<table width="100%" border="0">
<tr>
<td width="28%">
<table width="98%" border="0">
<tr>
<td align="center"><a href="AddCatagory.php">Add
New Catagory</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayCatagories.php">Display
catagories</a></td>
</tr>
<tr>
<td align="center"><a href="AddProduct.php">Add
New Product</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayProducts.php">Display
products</a></td>
</tr>
<tr>
<td align="center"><a href="index1.php">Main
Page</a></td>
```

```
</tr>
</table>
</td>
<td width="72%">
<table width="100%" border="1">
<tr>
<td width="12%">Catagory name</td>
<td width="6%">Edit</td>
<td width="11%">Delete</td>
</tr>
</table>
<table width="100%" border="1">
<?php
while($data1=mysql_fetch_object($select_query1))
{ ?>
<tr>
<td width="12%"><?php echo $data1->cat_name
?></td>
<td width="6%"><a
href="EditCatagories.php?cat_id=<?php echo
$data1->cat_id?>">Edit</a></td>
<td width="11%"><a
href="DisplayCatagories.php?delete=yes&cat_id=<?
php echo $data1->cat_id ?>">Delete</a></td>
</tr>
<?php } ?>
</table>
</td>
</tr>
</table>
</body>
</html>
To add new product the following file is used:
AddProduct.php
<?php
include_once "connection.php";
$select_catagory="select * from tblcatagory";
$select_query=mysql_query($select_catagory) or
die(mysql_error());
if(isset($_POST['Add']) && ($_POST['Add']=='Add
Product'))
```

```
{
if($_POST['cat_name']!='select' &&
$_POST['comp_name']!='' && $_POST['model']!=''
&& $_POST['prod_desc']!='' &&
$_POST['rate']!='')
{
$cat_name=stripslashes(trim($_POST['cat_name']))
;
$prod_id=stripslashes(trim($_POST['prod_id']));
$prod_name=stripslashes(trim($_POST['prod_name']
));
$comp_name=stripslashes(trim($_POST['comp_name']
));
$model=stripslashes(trim($_POST['model']));
$prod_desc=stripslashes(trim($_POST['prod_desc']
));
$rate=stripslashes(trim($_POST['rate']));
$select_catagory2="select * from tblcatagory
where cat_name='".$cat_name."'";
$select_query2=mysql_query($select_catagory2) or
die(mysql_error());
$result_cat_id=mysql_fetch_object($select_query2
);
$cat_id=$result_cat_id->cat_id;
$select_product="select * from tblproduct where
model='".$model."'";
$select_query3=mysql_query($select_product) or
die(mysql_error());
if(is_dir("img"))
mkdir("img");
if(mysql_num_rows($select_query3)==0)
{
$img_name=$_FILES['file']['name'];
$img_tmp_name=$_FILES['file']['tmp_name'];
$prod_img_path=time().$img_name.'.jpg';
if($img_name!='')
{
move_uploaded_file($img_tmp_name,"img/".$prod_im
g_path);
$insert_product="insert into
tblproduct(prod_id,prod_name,cat_id,prod_img_pat
```

```
h,comp_name,model,prod_desc,rate) values
($prod_id,'$prod_name',$cat_id,'$prod_img_path',
'$comp_name','$model','$prod_desc',$rate)";
$insert_query=mysql_query($insert_product) or
die(mysql_error());
if($insert_query)
{
$msg="New Product Details Added !!";
}
}
}
else {$msg="Model no can't be unique !!";}
}
}
?>
<html>
<head>
<title>Add Product</title>
</head>
<body bgcolor="#E0C6F0">
<h3 align="center">Add New Product Details</h3>
<hr align="center" size="4" color="#990000"
width="768" />
<h4 align="center"><?php echo $msg ?></h4>
<table width="100%" border="0">
<tr>
<td width="25%">
<table width="98%" border="0">
<tr>
<td align="center"><a href="AddCatagory.php"
>Add New Catagory</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayCatagories.php">Display
catagories</a></td>
</tr>
<tr>
<td align="center"><a href="AddProduct.php">Add
New Product</a></td>
</tr>
```

```
<tr>
<td align="center"><a
href="DisplayProducts.php">Display
products</a></td>
</tr>
<tr>
<td align="center"><a href="index1.php">Main
Page</a></span></td>
</tr>
</table>
</td>
<td width="75%">
<form name="frm" method="post" enctype="multi-
part/form-data">
<table width="606" border="0" align="center"
cellspacing="5">
<tr>
<td>Product Id:</td>
<td>
<input type="text" name="prod_id" />
</td>
</tr>
<tr>
<td>Product Name:</td>
<td>
<input type="text" name="prod_name" />
</td>
</tr>
<tr>
<td width="305">Image path of the Product:</td>
<td width="282"><input type="file" name="file"
/></td>
</tr>
<tr>
<td>Catagory:</td>
<td>
<select name="cat_name">
<?php
while($catagory_dls=mysql_fetch_object($select_q
uery)) { ?>
<option value="<?php echo $catagory_dls-
```

```
>cat_name?>"><?php echo $catagory_dls-
>cat_name?></option>
<?php } ?>
</select>
</td>
</tr>
<tr>
<td>Company Name:</td>
<td>
<input type="text" name="comp_name" />
</td>
</tr>
<tr>
<td>Model:</td>
<td><input type="text" name="model" /></td>
</tr>
<tr>
<td>Description:</td>
<td><textarea name="prod_desc" rows="5"
cols="40"></textarea></td>
</tr>
<tr>
<td>Rate:</td>
<td><input type="text" name="rate" /></td>
</tr>
</table>
<table width="564" border="0" align="center"
cellspacing="5">
<tr>
<td align="center"><input type="submit"
name="Add" value="Add Product" /></td>
</tr>
</table>
</form>
</td>
</tr>
</table>
</body>
</html>
```

The following will be used to see the records stored in the table "tblproduct ".

DisplayProducts.php

```
<?php
include_once "connection.php";
if(isset($_GET['delete']) &&
$_GET['delete']=='yes')
{
$delete_qury="delete from tblproduct where
prod_id='".$_GET['prod_id']."'";
mysql_query($delete_qury) or die(mysql_error());
}
$select1="select * from tblproduct";
$select_query1=mysql_query($select1) or
die(mysql_error());
?>
<html>
<head>
<title>Display All Products</title>
</head>
<body bgcolor="#FFFFBF">
<table width="120%" border="0">
<tr>
<td width="20%">
<table width="83%" border="0">
<tr>
<td align="center"><a href="AddCatagory.php"
>Add New Catagory</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayCatagories.php">Display
catagories</a></td>
</tr>
<tr>
<td align="center"><a href="AddProduct.php">Add
New Product</a></td>
</tr>
<tr>
<td align="center"><a
```

```
href="DisplayProducts.php">Display
products</a></td>
</tr>
<tr>
<td align="center"><a href="index1.php">Main
Page</a></td>
</tr>
</table>
</td>
<td width="100%">
<table width="100%" border="1">
<caption align="center">
<h2 align="center">Display All Products</h2>
<hr align="center" size="4" color="#990000"
width="768" />
<tr>
<td width="10%">Product Name</td>
<td width="10%">Product Image</td>
<td width="12%">Catagory</td>
<td width="25%">Product description</td>
<td width="9%">Price</td>
<td width="6%">Edit</td>
<td width="11%">Delete</td>
</tr>
<?php
while($data1=mysql_fetch_object($select_query1))
{ ?>
<input type="hidden" name="prod_id" value="" />
<tr>
<td width="12%"><?php echo $data1->prod_name
?></td>
<td width="23%" height="77"><img src="img\<?php
echo $data1->prod_img_path ?>" height="60"
width="60" /><br />
<p><?php echo $data1->comp_name.' '. $data1-
>model; ?></P></td>
<?php
$select_catagory2="select * from tblcatagory
where cat_id='".$data1->cat_id."'";
$select_query2=mysql_query($select_catagory2) or
die(mysql_error());
```

```
$result_cat_name=mysql_fetch_object($select_quer
y2);
$cat_name=$result_cat_name->cat_name;
?>
<td width="12%"><?php echo $cat_name ?></td>
<td width="25%"><?php echo $data1->prod_desc
?></td>
<td width="9%"><?php echo $data1->rate ?></td>
<td width="6%"><a
href="EditProducts.php?prod_id=<?php echo
$data1->prod_id ?>&cat_id=<?php echo $data1-
>cat_id?>" style="color:#FF0000">Edit</a></td>
<td width="11%"><a
href="DisplayProducts.php?delete=yes&prod_id=<?p
hp echo $data1->prod_id ?>">Delete</a></td>
</tr>
<?php } ?>
</table>
</td>
</tr>
</table>
</body>
</html>
```

In this file there is a link "Edit". If user clicks this link the file "EditProducts.php" will be called. The record of the table "tblproduct" can be edited by this file. User can also delete any record from tblproduct using the link "Delete".

EditProducts.php

```
<?php
include_once "connection.php";
$cat_id=$_REQUEST['cat_id'];
if(isset($_POST['update_data']) &&
($_POST['update_data']=='update_data'))
{
if(isset($_POST['Edit']) &&
($_POST['Edit']=='Edit Product'))
{
if($_POST['cat_name']!='' &&
$_POST['comp_name']!='' && $_POST['model']!=''
```

```
&& $_POST['prod_desc']!='' &&
$_POST['rate']!='')
{
$cat_name=$_POST['cat_name'];
$prod_name=stripslashes(trim($_POST['prod_name']
));
$select_catagory2="select * from tblcatagory
where cat_name='".$cat_name."'";
$select_query2=mysql_query($select_catagory2) or
die(mysql_error());
$result_cat_id=mysql_fetch_object($select_query2
);
$cat_id=$result_cat_id->cat_id;
$comp_name=stripslashes(trim($_POST['comp_name']
));
$model=stripslashes(trim($_POST['model']));
$prod_desc=stripslashes(trim($_POST['prod_desc']
));
$rate=stripslashes(trim($_POST['rate']));
$insert_query="update tblproduct set
cat_id='$cat_id',comp_name='$comp_name',model='$
model',prod_desc='$prod_desc',rate=$rate where
prod_id='".$_REQUEST['prod_id']."'";
$result=mysql_query($insert_query) or
die(mysql_error());
if($result)
{
$msg1="Product Info Updated !!";
}
}
}
}
if(isset($_POST['update_picture']) &&
($_POST['update_picture']=='update_picture'))
{
if(isset($_POST['Edit_Pic']) &&
($_POST['Edit_Pic']=='Change Picture'))
{
$img_name=$_FILES['file']['name'];
$img_tmp_name=$_FILES['file']['tmp_name'];
$prod_img_path=time().$img_name.'.jpg';
```

```
if($img_name!='')
{
$select_img="select * from tblproduct where
prod_id='".$_REQUEST['prod_id']."'";
$qry=mysql_query($select_img) or die
(mysql_error());
$res=mysql_fetch_object($qry);
if(file_exists("img/".$res->prod_img_path))
{
unlink("img/".$res->prod_img_path);
}
move_uploaded_file($img_tmp_name,"img/".$prod_im
g_path);
$insert_query="update tblproduct set
prod_img_path='$prod_img_path' where
prod_id='".$_REQUEST['prod_id']."'";
$result=mysql_query($insert_query) or
die(mysql_error());
if($result)
{
$msg2="Picture Updated !!";
}
}
}
}
$query="select * from tblproduct where
prod_id='".$_REQUEST['prod_id']."'";
$result=mysql_query($query) or
die(mysql_error());
$data=mysql_fetch_object($result);
$select_catagory="select * from tblcatagory";
$select_query_catagory=mysql_query($select_catag
ory) or die(mysql_error());
?>
<html>
<head>
<title>Update Products</title>
</head>
<body bgcolor="#FFD3A8">
<h2 align="center">Edit Product Details</h2>
<hr align="center" size="4" color="#990000"
```

```
width="328" />
<h3 align="center"><span><?php echo $msg1
?></span></h3>
<table width="100%" border="0">
<tr>
<td width="22%">
<table width="100%" border="0">
<tr>
<td align="center"><a href="AddCatagory.php"
>Add New Catagory</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayCatagories.php">Display
catagories</a></td>
</tr>
<tr>
<td align="center"><a href="AddProduct.php">Add
New Product</a></td>
</tr>
<tr>
<td align="center"><a
href="DisplayProducts.php">Display
products</a></td>
</tr>
<tr>
<td align="center"><a href="index1.php">Main
Page</a></td>
</tr>
</table>
</td>
<td width="78%">
<form name="frm" method="post" enctype="multi-
part/form-data">
<input type="hidden" name="update_data"
value="update_data" />
<table width="623" border="0" align="center"
cellspacing="5">
<tr>
<td>Product Name:</td>
<td>
```

```
<input type="text" name="prod_name" value="<?php
echo $data->prod_name ?>"/></td>
</tr>
<tr>
<td>Catagory:</td>
<td>
<select name="cat_name">
<?php
while($data_catagory=mysql_fetch_object($select_
query_catagory)) { ?>
<option value="<?php echo $data_catagory-
>cat_name?>" <?php if($data_catagory-
>cat_name==$cat_name) {echo 'selected';
}?>><?php echo $data_catagory-
>cat_name?></option>
<?php } ?>
</select>
</td>
</tr>
<tr>
<td>Company Name:</td>
<td>
<input type="text" name="comp_name" value="<?php
echo $data->comp_name ?>"/></td>
</tr>
<tr>
<td>Model:</td>
<td><input type="text" name="model" value="<?php
echo $data->model ?>" /></td>
</tr>
<td>Description:</td>
<td><textarea name="prod_desc" rows="5"
cols="40"><?php echo $data->prod_desc
?></textarea></td>
</tr>
<tr>
<td>Rate:</td>
<td><input type="text" name="rate" value="<?php
echo $data->rate ?>" /></td>
</tr>
</table>
```

```
<table width="564" border="0" align="center"
cellspacing="5">
<tr>
<td align="center"><input type="submit"
name="Edit" value="Edit Product" /></td>
</tr>
</table>
</form>
<br /><br />
<h2 align="center">Change Product Picture</h2>
<hr align="center" size="4" color="#990000"
width="368" />
<h3 align="center"><span><?php echo $msg2
?></span></h3>
<form method="post" name="frm2" enctype="multi-
part/form-data">
<input type="hidden" name="update_picture"
value="update_picture" />
<table width="623" border="0" align="center"
cellspacing="5">
<tr>
<td width="218">Change Picture:</td>
<td width="380"><input type="file" name="file"
/></td>
</tr>
</table>
<br />
<div align="center"><input type="submit"
name="Edit_Pic" value="Change Picture" /></div>
</form>
</td>
</tr>
</table>
</body>
</html>
```

To run this project at first you have to create the database and the tables using the programs described above. Then only you can run this project successfully. If you have "wamp" installed in your system, then you can create all these files described above in a folder within "www" folder. Then you will be able to run this project from "localhost".

Are you drowning in Tech Problems?

Specialist lifeguards are just a click away...

If you have a technology related problem, just log on to thinkdigit.com, visit our Tech Q&A section and let one of our experts solve it for you.

Troubleshooting has never been easier

www.thinkdigit.com

Presents

TECH Q&A