PHP, SQL & Word Press

Computer science Cluster paper -1

MATERIAL

Prepared by

Department of Computer Science

SRI GCSR COLLEGE

GMR NAGAR

RAJAM

INTRODUCTION TO PHP

- **What is PHP?**
  - PHP stands for **P**HP: **H**ypertext **P**reprocessor
  - PHP is a server-side scripting language, like ASP
  - PHP scripts are executed on the server
  - PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
  - PHP is an open source software
  - PHP is free to download and use
- **What is a PHP File?**
  - PHP files can contain text, HTML tags and scripts
  - PHP files are returned to the browser as plain HTML
  - PHP files have a file extension of ".php", ".php3", or ".phtml"
- **What is MySQL?**
  - MySQL is a database server
  - MySQL is ideal for both small and large applications
  - MySQL supports standard SQL
  - MySQL compiles on a number of platforms
  - MySQL is free to download and use
- **PHP + MySQL**
  - PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform?
- **Why PHP?**
  - PHP runs on different platforms (Windows, Linux, Unix, etc.)
  - PHP is compatible with almost all servers used today (Apache, IIS, etc.)
  - PHP is FREE to download from the official PHP resource: www.php.net
  - PHP is easy to learn and runs efficiently on the server side
- **What is PHP?**
  - "PHP is an open-source, server-side, HTML-embedded Web-scripting language for building dynamic and interactive Web sites. "
  - PHP stands *Personal Home Page Tools* when it was created by Rasmus Lerdort in 1994 but after few years latter it is known as a *Hypertext Preprocessor*.
  - PHP programs run on a Web server, and serve Web pages to visitors on request. One of the key features of PHP is that you can embed PHP code within HTML Web pages, making it very easy for you to create dynamic content quickly.
  - PHP stands for *PHP: Hypertext Preprocessor* (and, yes, we're aware PHP is a "recursive acronym"—probably meant to confuse the masses). Its flexibility and relatively small learning curve (especially for programmers who have a background in C, Java, or Perl) make it one of the most popular scripting languages around.
- Although PHP only started gaining popularity with Web developers around 1998, it was created by Rasmus Lerdorf way back in 1994.

- PHP started out as a set of simple tools coded in the C language to replace the Perl scripts that Rasmus was using on his personal home page (hence the original meaning of the " PHP " acronym). He released PHP to the general public in 1995, and called it PHP version 2.
- In 1997, two more developers, Zeev Suraski and Andi Gutmans, rewrote most of PHP and, along with Rasmus, released PHP version 3.0 in June 1998. By the end of

that year, PHP had already amassed tens of thousands of developers, and was being used on hundreds of thousands of Web sites.

- For the next version of PHP, Zeev and Andi set about rewriting the PHP core yet again, calling it the" Zend Engine " (basing the name " Zend " on their two names). The new version, PHP 4, was launched in May 2000. This version further improved on PHP 3, and included session handling features, output buffering, a richer core language, and support for a wider variety of Web server platforms.
- Although PHP 4 was a marked improvement over version 3, it still suffered from a relatively poor object - oriented programming (OOP) implementation. PHP 5, released in July 2004, addressed this issue, with private and protected class members; final, private, protected, and static methods; abstract classes; interfaces; and a standardized constructor/destructor syntax.

- **History of MySQL:**
  - The history of MySQL can be traced as far back as 1979, when MySQL's creator, Monty
    Widenius, worked for a Swedish IT and data consulting firm, TcX. In 1994, when TcX began working on Web data applications, chinks in the UNIREG armor, primarily having to do with application overhead, began to appear.
  - This sent Monty and his colleagues off to look for other tools.
  - One they inspected rather closely was Hughes mSQL, a light and zippy database application developed by David Hughes. mSQL possessed the distinct advantages of being inexpensive and somewhat entrenched in the market, as well as featuring a fairly well-developed client API. The 1.0 series of mSQL release lacked indexing, however, a crucial to performance with large data stores. Although the 2.0 series of mSQL would see the addition of theis feature, the particular implementation used was not compatible with UNIREG's B+-based features. At this point, MySQL, at least conceptually, was born.
  - By early 1995, TcX had a 1.0 version of this new product ready. They gave it the moniker MySQL and later that year release it under a combination open source and commercial
    licensing scheme that allowed continued development of the product while providing a revenue stream for MySQL AB, the company that evolved from TcX.
  - Over the past ten years, MySQL has truly developed into a world class product. MySQL now competes with even the most feature-rich commercial database applications such as oracle and Informix.
  - Additions in the 4.X series have included much-requested features such as transactions and foreign key support. All this has made MySQL the world's most used open source databse.

- **History of Apache Web Server:**

  - Apache, an open-source Web server created by American software developer Tobert McCool. Apache was released in 1995 and quickly gained a majority hold on the Web projects. In the early 21st century, Apache server deployed more than 50 percent of the Internet's content.

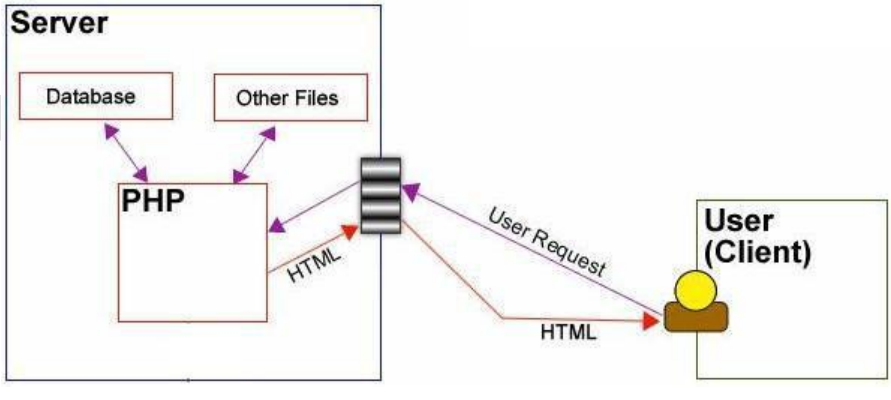
- As a Web Server, Apache is responsible for accepting directory (HTTP) requests from internet users and sending them their desired information in the form of files and
  WebPages. Much of the Web's software and code is designed to work along with Apache's features.
- Apache was originally known as the NCSA HTTPd Web server and was written by McCool when he was an undergraduate at the National Center for Supercomputing Applications (NCSA) at the University of llinois at Urbana-Champaign. Apache is maintained and developed by a large community of volunteers and developers from the Apache Software Foundation, as well ad by contributions from users worldwide.

History of Open Source:

- The term Open source was coined in 1998 after Netscape decided to publish the source code for its popular Navigator browser.
- This announcement prompted a small group of software developers who had been long-time supporters of the soon-to-be open source ideology to formally develop the Open Source Initiatives (OSI) and the Open Source Definition.
- Linux became the first operating system that could be considered open source, and many programs followed soon thereafter. Large software corporations, such as Corel, began to offer versions of their program that worked on Linux machines.
- Although there are now numerous classifications of OSI open source licenses, any software that bears the OSI Certification seal can be considered open source because it has passed the Open Source Definition test. These programs are available from a multitude of Web sites; the most popular is www.sourceforge.net, which houses more than 83,000 open source projects.

Relationship between Apache, MySQL and PHP (AMP Module):

In their scenario, you can characterize the three components of the APM module as follows:



Apache:

- This is your highly trained master of chef.
- Whatever people ask for, she prepares it without complaint.
- She is quick, flexible, and able to prepare a multitude of different types of foods.

- Apache acts in much the same way as your HTTP server, parsing files and passing on the results.
- Apache acts as your Web server.
- Its main job is to parse any file requested by a browser and display the correct results according to the code within that file.
- Apache is quite powerful and can accomplish virtually any task that you, as a Webmaster, require.
- The features and server capabilities available in this version 2.0.47 include the following: Password-protected pages for a multitude of users
  Customized error pages
  Display of code in numerous levels of HTML, and the capability to determine at what level the browser can accept the content
  Usage and error logs in multiple and customizable formats
  Virtual hosting for different IP addresses mapped to the same server Directory Index directives to multiple files
  URL aliasing or rewriting with no fixed limit

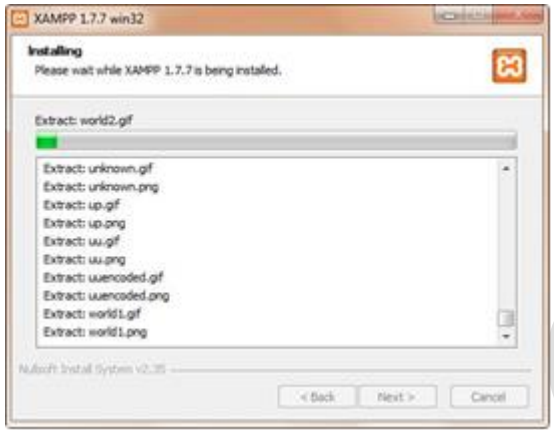
- **PHP:**
  - This is the waiter.
  - He gets requests from the patron and carries them back to the kitchen with specific instructions about how the meal should be prepared.
  - PHP is a server-side scripting language that allows your Web site to be truly dynamic.
  - PHP stands for *PHP: Hypertext Preprocessor* (and, yes, we're aware PHP is a "recursive acronym"—probably meant to confuse the masses).
  - Its flexibility and relatively small learning curve (especially for programmers who have a background in C, Java, or Perl) make it one of the most popular scripting languages around.
  - PHP's popularity continues to increase as businesses and individuals everywhere embrace it as an alternative to Microsoft's ASP language and realize that PHP's benefits most certainly outweigh the costs (three cheers for open source!).
  - According to Zend Technologies, Ltd., the central source of PHP improvements and designers of the Zend Engine, which supports PHP applications, PHP code can now be found in approximately 9 million Web sites.
- **MySQL:**
  - This is your stockroom of ingredients (or in this case, information).
  - MySQL is the database construct that enables PHP and Apache to work together to access and display data in a readable format to a browser.
  - It is a Structure Query Language server designed for heavy loads and processing of complex queries.
  - As a relational database system, MySQL allows many different tables to be joined together for maximum efficient and speed.
  - MySQL is the perfect choice for providing data via the Internet because of its ability to handle heavy loads and its advanced security measures.
  - One of the most popular features of the MySQL's are described below.
    - Multiple CPUs usable through kernel threads and Multi-platform operation.
    - Numerous column types cover virtually every type of data.
    - Group functions for mathematical calculations and sorting.

- Commands that allow information about the databases to be easily and succinctly shown to the administrator.
- Function names that do not affect table or column names.
- A password and user verification system for added security.
- Up to 32 indexes per table permitted; this feature has been successfully implemented at levels of 60,000 tables and 5,000,000,000 rows.

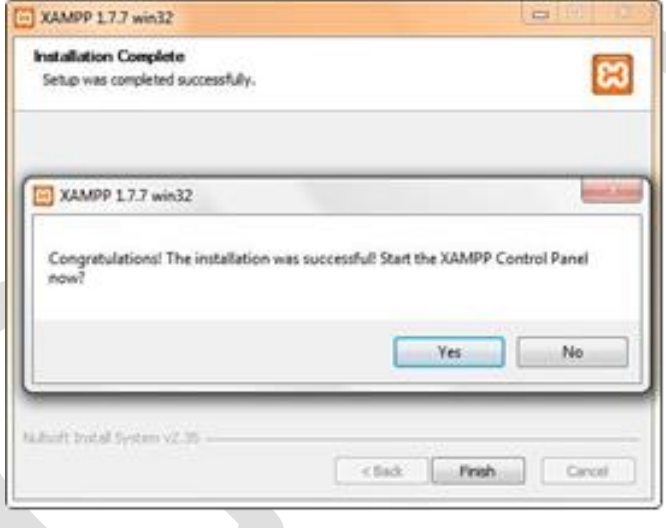
Installing XAMPP on Windows

- The XAMPP installation file included on the CD-ROM has been tested and is suitable for Windows operating systems from XP through Windows 7. Earlier versions of Windows are not supported.
- To use the XAMPP installation file from the CD-ROM, first insert the CD-ROM into your PC; it should play automatically. If it does not, double-click the drive icon for your CD-ROM under My Computer, and navigate to the directory containing the XAMPP installer files.
- Now that you have access to the XAMPP file on the CD-ROM, or if you have down-loaded the latest version from http://www.apachefriends.org/en/xampp-windows.html, double-click the file to launch the wizard-based installer program.
- You are first asked to select your language; English is the default selection. After selecting your language and clicking the OK button, you will see the welcome screen of the installer program, as shown in Figure 1.2.

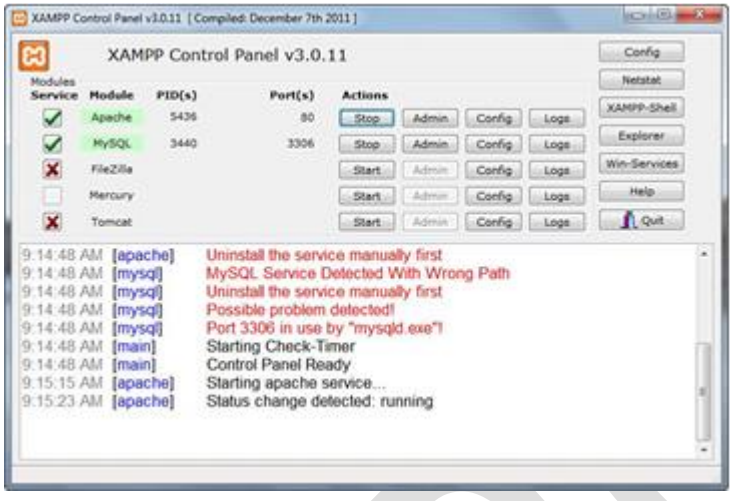
- Click the Next button to continue the installation process. As with most wizard-like installations, you are asked to select an installation location and some installation options before moving to the next step. The XAMPP installation is no different; you should leave the default installation location and the default installation options as marked and click the Next button to move on past each screen. At this point, the installation process itself happens, as shown in Figure 1.3.



- When the installation process finishes, the installer alerts you to this status; click the Finish button to complete the installation. Before the XAMPP installation process completely closes, it asks whether you want to start the Control Panel for managing the installed services, as shown in Figure 1.4.



- The XAMPP Control Panel, as shown in Figure 1.5, provides you with one-click access to starting and stopping the Apache and MySQL server processes running on your machine. If you are running these server processes on your local machine for development purposes only, you might want to turn them on only when you need them; the Control Panel allows quick access to do just that.



- To test whether the web server is running, open a web browser and enter **http://localhost/xampp/xampp.php**. The menu for the XAMPP service should display, as shown in Figure 1.6.

Installing MySQL on Windows

- The MySQL installation process on Windows uses a standard Microsoft Windows Installer (MSI) file to walk you through the installation and configuration of MySQL on your Windows XP, Windows Server 2003, Windows Vista, or Windows 7 machine.

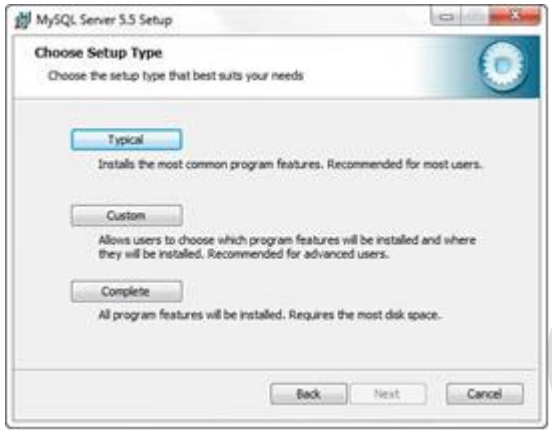
- Go to the MySQL downloads page at http://dev.mysql.com/downloads/mysql/5.5.html and select the Windows option from the drop-down menu. Download the Windows MSI Installer file for your system, either 32- or 64-bit. When this file has been downloaded

- The following steps detail the installation of MySQL 5.5.21 on Windows 7; the installation sequence follows the same steps regardless of your Windows environment.

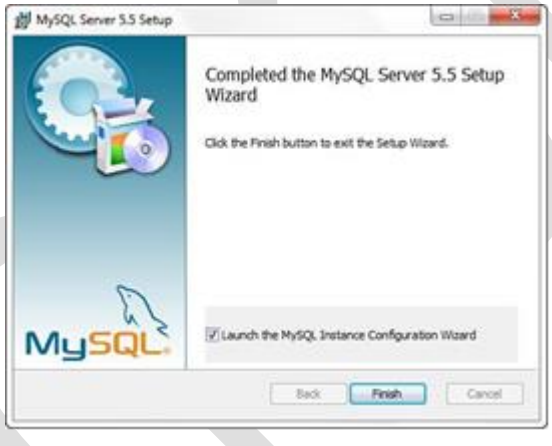
steps:

- Double-click the *.msi file to begin the installation sequence. You will see the first screen of the MySQL Setup Wizard, as shown in Figure 2.5. Click Next to continue.

- After agreeing to the terms and conditions, you are asked to choose a setup type—Typical, Custom, or Complete (see Figure 2.6).
- The Custom option allows you to pick and choose elements of MySQL to install, whereas the Complete option installs all the components of MySQL, which range from documentation to benchmarking suites.
- The Typical installation method is suitable for most users because it includes the client, server, and numerous tools for general management of your MySQL installation.
- Select Typical as the installation method and click Next to continue.



- Confirm your choice in the next screen and click the Install button to continue. The installation process takes over and installs files in their proper locations.
- When the installation is complete, you have the option of continuing to the MySQL Instance Configuration Wizard. This wizard is highly recommended because it creates a custom my.ini file tailored to your particular needs. To continue to the MySQL Instance Configuration Wizard, check the Launch the MySQL Instance Configuration Wizard check box and click the Finish button, as shown in Figure 2.7.



- When you see the MySQL Instance Configuration Wizard Welcome screen, click the Next button to go to the next step in the wizard. You will see two options for server configuration: Detailed and Standard.
- We use the Detailed Configuration option so that you can see all the options available to you. If you decide to select the Standard Configuration option, you must manually modify the resulting my.ini file to achieve the configuration you want.
- Select the Detailed Configuration radio button, and then click Next to continue.

- The next selection you must make is shown in Figure 2.8. In this step, you select the type of machine you are running: Developer Machine, Server Machine, or Dedicated MySQL Server Machine.
- Your selection on this screen determines the allotments for memory, disk, and processor usage. If you are using MySQL on your personal

machine for testing purposes, select the Developer Machine option. After making your selection, click Next to continue.

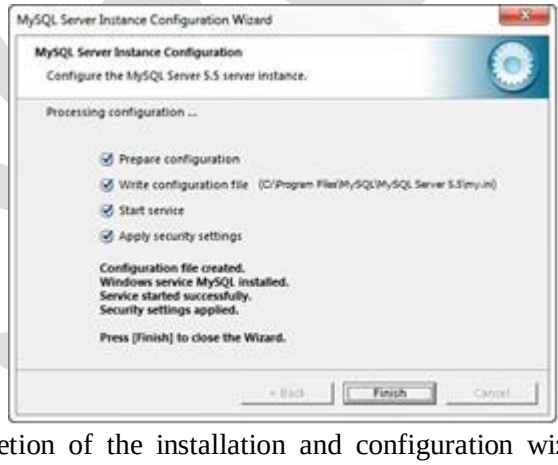
- If you have selected a database usage option that includes the InnoDB storage engine, the next step in the configuration process enables you to configure the
- disk location and storage thresholds. The defaults are shown in Figure 2.9, which you can simply confirm by clicking Next to continue, or you can change these settings and then click Next to continue with your custom set-tings in place.

- The next configuration option determines the number of concurrent connections to your MySQL server. Your setting will depend on the amount of traffic and database usage by your website or application. The default setting is Decision Support (DSS)/OLAP, which has a maximum number of 100 concurrent connections with an average of 20 assumed. The Online Transaction Processing (OLTP) option has a maximum of 500 concurrent connections, and the Manual setting allows you to select a number from a drop-down list or enter your own. Make your selection and click Next to continue.
- The Networking Options screen is next in the configuration sequence. Here you enable or disable TCP/IP networking and configure the port number used to connect to MySQL—the default is 3306, but you can use any unused port you choose. After making your selection, click Next to continue.

- It is recommended that MySQL be installed as a service. Check the Install as Windows Service check box and select a name for the service. Optionally, check the Launch the MySQL Server Automatically check box. You also have the option of adding the MySQL bin directory to your Windows PATH for easier invocation of MySQL from the cmd prompt; check the box if this is appropriate for you. When you have completed your selections, click Next to continue.

- The Security Options configuration screen is the most important screen of all. As shown in Figure 2.10, use this configuration screen to set a root password. click Next to continue.

- One step remains in the configuration sequence, and that is to click the Execute button to start the process. After the wizard has made it through the various configuration steps, you will see a confirmation screen, as shown in Figure 2.11, indicating the configuration file has been created and the MySQL service has been started. Click Finish to close the wizard.



- The completion of the installation and configuration wizards results in a running MySQL service and a custom my.ini file in the C:\Program Files\MySQL\MySQL Server 5.5\ directory.

Basic PHP Syntax

A PHP script can be placed anywhere in the document.
A PHP script starts with **<?php** and ends with **?>**:

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

The default file extension for PHP files is ".php".
A PHP file normally contains HTML tags, and some PHP scripting code.
Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:
EX:-

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

Note: PHP statements end with a semicolon (;).

Beginning and Ending a Block of PHP Statements

When writing PHP, you need to inform the PHP engine that you want it to execute your commands. If you don't do this, the code you write will be mistaken for HTML and will be output to the browser. You can designate your code as PHP with special tags that mark the beginning and end of PHP code blocks.

PHP Start and End Tags

Tag Style	Start Tag	End Tag
Standard tags	<?php	?>
Short tags	<?	?>
ASP tags	<%	%>
Script tags	<script language="php">	</script>

Note:- only the standard and script tags are guaranteed to work on any configuration. You must explici enable the short and ASP-style tags in your php.ini file. To activate recognition for short tags, you mus make sure that the short_open_tag switch is set to On in php.ini: short_open_tag = On; To activate recognition for the ASP-style tags, you must enable the asp_tags setting in php.ini: **asp_**tags **= On;**

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.
Comments can be used to:

- Let others understand what you are doing
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting:
EX:-

```
<!DOCTYPE html>
<html>
<body>
<?php
// This is a single-line comment
# This is also a single-line comment
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>
</body>
</html>
```

PHP Case Sensitivity

In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive. In the example below, all three echo statements below are legal (and equal):
EX:-

```
<!DOCTYPE html>
<html>
<body>
<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>
</body>
</html>
```

However; all variable names are case-sensitive.
In the example below, only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables):
Example

```
<!DOCTYPE html>
<html>
```

```
<body>
<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
</body>
</html>
```

Creating (Declaring) PHP Variables

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

Example

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

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

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

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

Think of variables as containers for storing data.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, car name, total volume).

Rules for PHP variables:

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

Remember that PHP variable names are case-sensitive!

Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example

```
<?php
$txt = "SGCSRC";
echo "I love $txt!";
?>
```

The following example will produce the same output as the example above:

Example

```
<?php
$txt = "SGCSRC";
echo "I love " . $txt . "!";
?>
```

The following example will output the sum of two variables:

Example

```
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
```

Note: You will learn more about the echo statement and how to output data to the screen in the next chapter.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

PHP Variables Scope

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

- local
- global
- static

Global and Local Scope

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

Example

```
<?php
$x = 5; // global scope

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

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

Example

```
<?php
function myTest() {
   $x = 5; // local scope
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
   global $x, $y;
   $y = $x + $y;
}

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

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

The example above can be rewritten like this:

Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
   $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

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

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

Example

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

myTest();
myTest();
myTest();
?>
```

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

PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen. The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

Example

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

Display Variables

The following example shows how to output text and variables with the echo statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;
echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
```

The PHP print Statement

The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

Display Variables

The following example shows how to output text and variables with the print statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;
print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>";
```

```
print $x + $y;
?>
```

PHP Data Types

Variables can store data of different types, and different data types can do different things.
PHP supports the following data types:

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

PHP String

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

Example

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

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

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.
Rules for integers:

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

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

Example

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

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.
In the following example $x is a float. The PHP var_dump() function returns the data type and value:

Example

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

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

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

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

An array stores multiple values in one single variable.
In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

Example

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

You will learn a lot more about arrays in later chapters of this tutorial.

PHP Object

An object is a data type which stores data and information on how to process that data.
In PHP, an object must be explicitly declared.
First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
  function Car() {
    $this->model = "VW";
  }
}


// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.
Tip: If a variable is created without a value, it is automatically assigned a value of NULL.
Variables can also be emptied by setting the value to NULL:

Example

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

PHP Constants

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
A valid constant name starts with a letter or underscore (no $ sign before the constant name).
Note: Unlike variables, constants are automatically global across the entire script.

Create a PHP Constant

To create a constant, use the define() function.

Syntax

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

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

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

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

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

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>
```

Constants are Global

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

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");

function myTest() {
   echo GREETING;
}
 myTest();
?>
```

PHP Operators

Operators are used to perform operations on variables and values.
PHP divides the operators in the following groups:

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

PHP Arithmetic Operators

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

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

PHP Assignment Operators

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

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x – y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

PHP Comparison Operators

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

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y,

			or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

PHP Increment / Decrement Operators

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

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

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

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

PHP String Operators

PHP has two operators that are specially designed for strings.

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

Show it »

PHP Array Operators

The PHP array operators are used to compare arrays.
Conditional statements are used to perform different actions based on different conditions.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have

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

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

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

PHP - The if Statement

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

Syntax

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

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

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

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

PHP - The if...else Statement

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

Syntax

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

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

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

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

PHP - The if...elseif....else Statement

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

Syntax

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

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

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

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

The PHP switch Statement

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

Syntax

```
switch (n) {
   case label1:
      code to be executed if n=label1;
      break;
   case label2:
      code to be executed if n=label2;
      break;
   case label3:
      code to be executed if n=label3;
      break;
   ...
   default:
      code to be executed if n is different from all labels;
}
```

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

Example

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

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

Using the? Operator

The ?: or *ternary* operator is similar to the if statement, except that it returns a value derived from one of two expressions separated by a colon. This construct provides you with three parts of the whole, hence the name *ternary*. The expression used to generate the returned value depends on the result of a test expression:

(*expression*) ? *returned_if_expression_is_true* : *returned_if_expression_is_false*;

If the test expression evaluates to true, the result of the second expression is returned; otherwise, the value of the third expression is returned. Listing 6.5 uses the ternary operator to set the value of a variable according to the value of $mood.

EX:-

```
1: <?php
2: $mood = “sad”;
3: $text = ($mood == “happy”) ? “I am in a good mood!” : “I am in a $mood mood.”;
4: echo “$text”;
5: ?>
```

Loops

So far, you’ve looked at decisions that a script can make about what code to execute. Scripts can also decide how many times to execute a block of code. Loop statements are specifically designed to enable you to perform repetitive tasks because they continue to operate until a specified condition is achieved or until you explicitly choose to exit the loop.

The while Statement

The while statement looks similar in structure to a basic if statement, but has the ability to loop:

```
while (expression) {
// do something
}
```

Unlike an if statement, a while statement executes for as long as the expression evaluates to true, over and over again if need be. Each execution of a code block within a loop is called an *iteration*. Within the block, you usually change something that affects the while statement's expression; otherwise, your loop continues indefinitely. For example, you might use a variable to count the number of iterations and act accordingly.

```
1: <?php
2: $counter = 1;
3: while ($counter <= 12) {
4: echo $counter." times 2 is ".($counter * 2)."<br />";
5: $counter++;
6: }
7: ?>
```

The do...while Statement

A do...while statement looks a little like a while statement turned on its head. The essential difference between the two is that the code block is executed *before* the truth test and not after it:

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

The test expression of a do...while statement should always end with a semicolon.
This type of statement is useful when you want the code block to be executed at least once, even if the while expression evaluates to false.

```
1: <?php
2: $num = 1;
3: do {
4: echo "The number is: ".$num."<br />";
5: $num++;
6: } while (($num > 200) && ($num < 400));
7: ?>
```

***The for* Statement**

Anything you want to do with a for statement can also be done with a while statement, but a for statement is often a more efficient method of achieving the same effect. With a for statement, you can achieve this same series of events, but in a single line of code. This allows for more compact code and makes it less likely that you might forget to increment a counter variable, thereby creating an infinite loop:

```
for (initialization expression; test expression; modification expression) {
// code to be executed
}
```

Infinite loops are, as the name suggests, loops that run without bounds. If your loop is running infinitely, your script is running for an infinite amount of time. This behavior is very stressful on your web server and renders the web page unusable. The expressions within the parentheses of the for statement are separated by semicolons. Usually, the first expression initializes a counter variable, the second expression is the test condition for the loop, and the third expression increments the counter.

```
1: <?php
2: for ($counter=1; $counter<=12; $counter++) {
```

```
3: echo $counter.” times 2 is “.($counter * 2).”<br />”;
4: }
5: ?>
```

Because the $counter variable is initialized and incremented at the beginning of the statement, the logic of the loop is clear at a glance. That is, as shown in line 2, the first expression initializes the $counter variable and assigns a value of 1, the test expression verifies that $counter contains a value that is less than or equal to 12, and the final expression increments the $counter variable. Each of these items is found in the single line of code. When the sequence of script execution reaches the for loop, the $counter variable is initialized and the test expression is evaluated. If the expression evaluates to true, the code block is executed. The $counter variable is then incremented and the test expression is evaluated again. This process continues until the test expression evaluates to false.

```
1: <?php
2: for ($counter=1; $counter <= 10; $counter++) {
3: $temp = 4000/$counter;
4: echo “4000 divided by “.$counter.” is...”.$temp.”<br />”;
5: }
6: ?>
```

This example initializes the variable $counter and assigns a value of 1. The test expression in the for statement verifies that the value of $counter is less than or equal to 10. Within the code block, 4000 is divided by $counter, printing the result to the browser.

Using the break Statement

```
1: <?php
2: $counter = -4;
3: for (; $counter <= 10; $counter++) {
4: if ($counter == 0) {
5: break;
6: } else {
7: $temp = 4000/$counter;
8: echo “4000 divided by “.$counter.” is...”.$temp.”<br />”;
9: }
10: }
11 ?>
```

Dividing a number by 0 does not cause a fatal error in PHP. Instead, PHP generates a warning and execution continues. If the value of $counter is equal to 0, the break statement immediately halts execution of the code block, and program flow continues after the for statement.

Skipping an Iteration with the continue Statement

The continue statement ends execution of the current iteration but doesn’t cause the loop as a whole to end. Instead, the next iteration begins immediately. With the continue statement you can avoid a divide-by-0 error without ending the loop completely.

```
1: <?php
2: $counter = -4;
3: for (; $counter <= 10; $counter++) {
4: if ($counter == 0) {
5: continue;
6: }
```

```
7: $temp = 4000/$counter;
8: echo “4000 divided by “.$counter.” is...”.$temp.”<br />”;
9: }
10: ?>
```

Swaps the break statement for a continue statement. If the value of the $counter variable is equivalent to 0, the iteration is skipped, and the next one starts immediately.

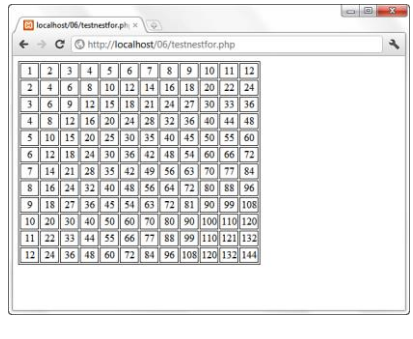
Note:- Using the break and continue statements can make code more difficult to read because they often add layers of complexity to the logic of the loop statements that contain them. Use these statements with care, or comment your code to show other programmers (or yourself) exactly what you’re trying to achieve with these statements.

Nesting Loops

Loops can contain other loop statements, as long as the logic is valid and the loops are tidy. The combination of such statements proves particularly useful when working with dynamically created HTML tables.

```
1: <?php
2: echo “<table style=\”border: 1px solid #000;\”> \n”;
3: for ($y=1; $y<=12; $y++) {
4: echo “<tr> \n”;
5: for ($x=1; $x<=12; $x++) {
6: echo “<td style=\”border: 1px solid #000; width: 25px; text-align:center;\”>”;
7. CAUTION
8. echo ($x * $y);
9: echo “</td> \n”;
10: }
11: echo “</tr> \n”;
12: }
13: echo “</table>”;
14: ?>
```

Before you examine the for loops, take a closer look at line 2 echo “<table style=\”border: 1px solid black;\”> \n”; Notice that it uses the backslash character (\) before each of the quotation marks within the string containing the style information for the table. These backslashes also appear in lines 6 and 7, in the style information for the table data cell. This is necessary because it tells the PHP engine that we want to use the quotation mark character, rather than have PHP interpret it as the beginning or end of a string. If you did not “escape” the quotation marks with the backslash character, the statement would not make sense to the engine; it would read it as a string followed by a number followed by another string. Such a construct would generate an error.




```
1: <?php
2: $display_prices = true;
3: if ($display_prices) {
4: echo “<table border=\”1\”>\n”;
5: echo “<tr><td colspan=\”3\”>”;
6: echo “today’s prices in dollars”;
7: echo “</td></tr>”;
```

8: echo “<tr><td>\$14.00</td><td>\$32.00</td><td>\$71.00</td></tr>\n”;
9: echo “</table>”;
10: }
11: ?>

If the value of $display_prices is set to true in line 2, the table is printed. For the sake of readability, we split the output into multiple echo() statements, and once again use the backslash to escape any quotation marks used in the HTML output.


```
1: <?php
2: $display_prices = true;
3: if ($display_prices) {
4: ?>
5: <table border=”1”>
6: <tr><td colspan=”3”>today’s prices in dollars</td></tr>
7: <tr><td>$14.00</td><td>$32.00</td><td>$71.00</td></tr>
8: </table>
9: <?php
10: }
11: ?>
```

Working with Functions

A *function* is a self-contained block of code that can be called by your scripts. When called, the function’s code is executed and performs a particular task. You can pass values to a function, which then uses the values appropriately—storing them, transforming them, displaying them, whatever the function is told to do. When finished, a function can also pass a value back to the original code that called it into action.

Calling Functions

Functions come in two flavors:

1. built in to the language
2. You define yourself.

PHP has hundreds of built-in functions in use like strtoupper(),strtolower(),abs()…etc.

A function call consists of the function name followed by parentheses. If you want to pass information to the function, you place it between these parentheses. A piece of information passed to a function in this way is called an *argument*. Some functions require that more than one rgument be passed to them, separated by commas:

Syntax:- *some_function*($*an_argument*, $*another_argument*);

Most functions return some information back after they’ve completed their task; they usually at least tell whether their mission was successful.

Ex:- $new_string = strtoupper(“Hello Web!”);

You may now use $new_string in your code, such as to print it to the screen:

echo $new_string;

This code results in the following text on the screen:

HELLO WEB!

```
1: <?php
2: $str=strtoupper(“sgcsrc”);
```

```
3: echo $str;
4: ?>
```

```
1: <?php
2: $num = -321;
3: $newnum = abs($num);
4: echo $newnum;
5: //prints "321"
6: ?>
```

Defining a Function

You can define your own functions using the function statement:

```
function some_function($argument1, $argument2)
{
//function code here
}
```

The name of the function follows the function statement and precedes a set of parentheses. If your function requires arguments, you must place comma-separated variable names within the parentheses. These variables are filled by the values passed to your function. Even if your function doesn't require arguments, you must nevertheless supply the parentheses.

```
1: <?php
2: function bighello()
3: {
4: echo "<h1>HELLO!</h1>";
5: }
6: bighello();
7: ?>
```

Declaring a Function That Requires an Argument

```
1: <?php
2: function printBR($txt)
3: {
4: echo $txt."<br/>";
5: }
6: printBR("This is a line.");
7: printBR("This is a new line.");
8: printBR("This is yet another line.");
9: ?>
```

In line 2, the printBR() function expects a string, so the variable name $txt is placed between the parentheses when the function is declared. Whatever is passed to printBR() is stored in this $txt variable. Within the body of the function, line 3 prints the $txt variable, appending a
 element to it. When you want to print a line to the browser, such as in line 6, 7, or 8, you can call printBR() instead of the built-in print(), saving you the bother of typing the
 element. Returning Values from User-Defined Functions

A function can return a value using the return statement in conjunction with a value. The return statement stops the execution of the function and sends the value back to the calling code.

A Function That Returns a Value

```
1: <?php
2: function addNums($firstnum, $secondnum)
3: {
4: $result = $firstnum + $secondnum;
5: return $result;
6: }
7: echo addNums(3,5);
8: //will print "8"
9: ?>
```

Notice in line 2 that addNums() should be called with two numeric arguments. These values are stored in the variables $firstnum and $secondnum. Predictably, addNums() adds the numbers contained in these variables and stores the result in a variable called $result. The return statement can return a value or nothing at all. How you arrive at a value passed by return can vary. The value can be hard-coded:

```
return 4;
```

It can be the result of an expression:

```
return $a/$b;
```

It can be the value returned by yet another function call:

```
return another_function($an_argument);
```

Variable Scope

A variable declared within a function remains local to that function. In other words, it is not available outside the function or within other functions. In larger projects, this can save you from accidentally overwriting the contents of a variable when you declare two variables with the same name in separate functions.

A Variable Declared Within a Function Is Unavailable Outside the Function

```
1: <?php
2: function test()
3: {
4: $testvariable = "this is a test variable";
5: }
6: echo "test variable: ".$testvariable."<br/>";
7: ?>
```

A variable declared outside a function is not automatically available within it.

Accessing Variables with the global Statement

From within one function, you cannot (by default) access a variable defined in another function or elsewhere in the script. Within a function, if you attempt to use a variable with the same name, you will only set or access a local variable.

Variables Defined Outside Functions Are Inaccessible from Within a Function by Default

```
1: <?php
2: $life = 42;
3: function meaningOfLife()
4: {
5: echo "The meaning of life is ".$life";
6: }
7: meaningOfLife();
```

```
8: ?>
```

The meaningOfLife() function does not have access to the $life variable in line 2; $life is empty when the function attempts to print it. A function can always demand an argument if it needs information about the outside world. Occasionally, you might want to access an important variable from within a function without passing it in as an argument. This is where the global statement comes into play.

Accessing Global Variables with the global Statement

```
1: <?php
2: $life=42;
3: function meaningOfLife()
4: {
5: global $life;
6: echo "The meaning of life is ".$life";
7: }
8: meaningOfLife();
9: ?>
```

When you place the global statement in front of the $life variable when it is declared in the meaningOfLife() function, it refers to the $life variable declared outside the function. You need to use the global statement within every function that needs to access a particular named global variable. You can declare more than one variable at a time with the global statement by simply separating each of the variables you want to access with commas:

```
global $var1, $var2, $var3;
```

Saving State Between Function Calls with the static Statement

Local variables within functions have a short but happy life—they come into being when the function is called and die when execution is finished, as they should. Saving State Between Function Calls with the static Statement Assume that you want a function to keep track of the number of times it has been called so that numbered headings can be created by a script.

Using the global Statement to Remember the Value of a Variable Between Function Calls

```
1: <?php
2: $num_of_calls = 0;
3: function numberedHeading($txt)
4: {
5: global $num_of_calls;
6: $num_of_calls++;
7: echo "<h1>".$num_of_calls." ".$txt."</h1>";
8: }
9: numberedHeading("Widgets");
10: echo "<p>We build a fine range of widgets.</p>";
11: numberedHeading("Doodads");
12: echo "<p>Finest in the world.</p>";
13: ?>
```

Functions that use the global statement cannot be read as standalone blocks of code. In reading or reusing them, we need to look out for the global variables that they manipulate. This is where the static statement can be useful. If you declare a variable within a function in

conjunction with the static statement, the variable remains local to the function, and the function "remembers" the value of the variable from execution to execution.

Using the static Statement to Remember the Value of a Variable Between Function Calls

```
1: <?php
2: function numberedHeading($txt)
3: {
4: static $num_of_calls = 0;
5: $num_of_calls++;
6: echo "<h1>".$num_of_calls." ". $txt."</h1>";
7: }
8: numberedHeading("Widgets");
9: echo "<p>We build a fine range of widgets.</p>";
10: numberedHeading("Doodads");
11: echo "<p>Finest in the world.</p>";
12: ?>
```

PHP provides a nifty feature to help build flexible functions. Until now, you've heard that some functions require one or more arguments. By making some arguments optional, you can render your functions a little less autocratic. The given example creates a useful little function that wraps a string in an HTML span element. To give the user of the function the chance to change the font-size style, you can demand a $fontsize argument in addition to the string.

A Function Requiring Two Arguments

```
1: <?php
2: function fontWrap($txt, $fontsize)
3: {
4: echo "<span style=\"font-size:$fontsize\">".$txt."</span>";
5: }
6: fontWrap("A Heading<br/>","24pt");
7: fontWrap("some body text<br/>","16pt");
8: fontWrap("smaller body text<br/>","12pt");
9: fontWrap("even smaller body text<br/>","10pt");
10: ?>
```

By assigning a value to an argument variable within the function definition's parentheses, you can make the $fontsize argument optional. If the function call doesn't define an argument for this argument, the value you have assigned to the argument is used instead. Listing 7.11 uses this technique to make the $fontsize argument optional.

A Function with an Optional Argument

```
1: <?php
2: function fontWrap($txt, $fontsize = "12pt")
3: {
4: echo "<span style=\"font-size:$fontsize\">".$txt."</span>";
```

LISTING 7.11 Continued

```
5: }
6: fontWrap("A Heading<br/>","24pt");
7: fontWrap("some body text<br/>");
8: fontWrap("smaller body text<br/>");
```

```
9: fontWrap("even smaller body text<br/>");
10: ?>
```

You can create as many optional arguments as you want, but when you've given an argument a default value, all subsequent arguments should also be given defaults. Passing Variable References to Functions When you pass arguments to functions, they are stored as copies in parameter variables. Any changes made to these variables in the body of the function are local to that function and are not reflected beyond it.

Passing an Argument to a Function by Value

```
1: <?php
2: function addFive($num)
3: {
4: $num += 5;
5: }
6: $orignum = 10;
7: addFive($orignum);
8: echo $orignum;
9: ?>
```

By default, variables passed to functions are passed by value. In other words, local copies of the values of the variables are made. You can change this behavior by creating a reference to your original variable. You can think of a reference as a signpost that points to a variable. In working with the reference, you are manipulating the value to which it points.
You can pass an argument by reference by adding an ampersand to the argument name in the function definition.

Using a Function Definition to Pass an Argument to a Function by Reference

```
1: <?php
2: function addFive(&$num)
3: {
4: $num += 5;
5: }
6: $orignum = 10;
7: addFive($orignum);
8: echo $orignum;
9: ?>
```

Testing for the Existence of a Function

You do not always know that a function exists before you try to invoke it. Different builds of the PHP engine might include different functionality, and if you are writing a script that may be run on multiple servers, you might want to verify that key features are available. For instance, you might want to write code that uses MySQL if MySQL-related functions are available but simply log data to a text file otherwise. You can use function_exists() to check for the availability of a function. function_ exists() requires a string representing a function name. It returns true if the function can be located, and false otherwise.

Testing for a Function's Existence

```
1: <?php
2: function tagWrap($tag, $txt, $func = "")
3: {
4: if ((!empty($txt)) && (function_exists($func))) {
```

```
5: $txt = $func($txt);
6: return “<”.$tag.”>”.$txt.”</”.$tag.”><br/>”;
7: } else {
8: return “<strong>”.$txt.”</strong><br/>”;
9: }
10: }
11:
12: function underline($txt)
13: {
14: return “<span style=\”text-decoration:underline;\”>”.$txt.”</span>”;
15: }
16: echo tagWrap(‘strong’, ‘make me bold’);
17: echo tagWrap(‘em’, ‘underline and italicize me’, “underline”);
18: echo tagWrap(‘em’, ‘make me italic and quote me’,
19: create_function(‘$txt’, ‘return “&quot;$txt&quot;”;’));
20: ?>
```

Working with Arrays

Arrays are used to store and organize data. PHP includes many functions that enable you to create, modify, and manipulate arrays, which you use often throughout the procedural programming method described in this book.

What Are Arrays?

Arrays are special types of variables that enable you to store as many values as you want. Arrays are indexed, which means that each entry is made up of a *key* and a *value*. The key is the index position, beginning with 0 and increasing incrementally by 1 with each new element in the array. The value is whatever value you associate with that position—a string, an integer, or whatever you want. Think of an array as a filing cabinet and each key/value pair as a file folder. The key is the label written on the top of the folder, and the value is what is inside.

Creating Arrays

You can create an array using either the array() function or the array operator []. The array() function is usually used when you want to create a new array and populate it with more than one element, all in one fell swoop. The array operator is often used when you want to create a new array with just one element at the outset, or when you want to add to an existing array element.

```
$rainbow = array(“red”, “orange”, “yellow”, “green”, “blue”, “indigo”, “violet”);
```

The following snippet shows the same array being created incrementally using the array operator:

```
$rainbow[] = “red”;
$rainbow[] = “orange”;
$rainbow[] = “yellow”;
$rainbow[] = “green”;
$rainbow[] = “blue”;
$rainbow[] = “indigo”;
$rainbow[] = “violet”;
```

Both snippets create a seven-element array called $rainbow, with values starting at index position 0 and ending at index position 6. If you want to be literal about it, you can specify the index positions, such as in this code:

```
$rainbow[0] = "red";
$rainbow[1] = "orange";
$rainbow[2] = "yellow";
$rainbow[3] = "green";
$rainbow[4] = "blue";
$rainbow[5] = "indigo";
$rainbow[6] = "violet";
```

Regardless of whether you initially create your array using the array() function or the array operator, you can still add to it using the array operator. In the first line of the following snippet, six elements are added to the array, and one more element is added to the end of the array in the second line:

```
$rainbow = array("red", "orange", "yellow", "green", "blue", "indigo");
$rainbow[] = "violet";
```

Creating Associative Arrays

Whereas numerically indexed arrays use an index position as the key—0, 1, 2, and so forth—associative arrays use actual named keys. The following example demonstrates this by creating an array called $character with four elements:

```
$character = array(
"name" => "Bob",
"occupation" => "superhero",
"age" => 30,
"special power" => "x-ray vision"
);
```

The four keys in the $character array are name, occupation, age, and special power. The associated values are Bob, superhero, 30, and x-ray vision, respectively. You can reference specific elements of an associative array using the specific key.

```
echo $character['occupation'];
```

As with numerically indexed arrays, you can use the array operator to add to an associative array:

```
$character['supername'] = "Mega X-Ray Guy";
```

The only difference between an associative array and a numerically indexed array is the key name. In a numerically indexed array, the key name is a number. In an associative array, the key name is a meaningful word.

Creating Multidimensional Arrays

The first two types of arrays hold strings and integers, whereas this third type holds other arrays. If each set of key/value pairs constitutes a dimension, a multidimensional array holds more than one series of these key/value pairs.

Defining a Multidimensional Array

```
1: <?php
2: $characters = array(
3: array(
```

```
4: “name” => “Bob”,
5: “occupation” => “superhero”,
6: “age” => 30,
7: “special power” => “x-ray vision”
8: ),
9: array(
10: “name” => “Sally”,
11: “occupation” => “superhero”,
12: “age” => 24,
13: “special power” => “superhuman strength”
14: ),
15: array(
16: “name” => “Jane”,
17: “occupation” => “arch villain”,
18: “age” => 45,
19: “special power” => “nanotechnology”
20: )
21: );
22: ?>
```

Each element consists of an associative array, itself containing four elements: name, occupation, age, and special power. However, if you attempt to print the master elements like so echo $characters [1]; the output will be Array because the master element indeed holds an array as its content. To really get to the content you want (that is, the specific information found within the inner array element), you need to access the master element index position plus the associative name of the value you want to view.

Take a look at this example:

```
echo $characters[1][‘occupation’];
```

It prints this:

```
superhero
```

If you add the following lines to the end of the code in Listing 8.1, it prints the information stored in each element, with an added line displayed in the browser for good measure:

```
foreach ($characters as $c) {
while (list($k, $v) = each ($c)) {
echo “$k ... $v <br/>”;
}
echo “<hr/>”;
}
```

The each() function looks at each element of the $c array and extracts the information accordingly. The echo statement simply prints each key and value ($k and $v) extracted from the $c array using the each() function and adds a line break for display purposes.

Some Array-Related Constructs and Functions

More than 70 array-related functions are built in to PHP. Some of the more common (and useful) functions are described below.

. count() and sizeof()—Each of these functions counts the number of elements in an array; they are aliases of each other.
Ex:- array $colors = array("blue", "black", "red", "green");
both count($colors); and sizeof($colors); return a value of 4.
. each() and list()—These functions (well, list() is a language construct that *looks* like a function) usually appear together, in the context of stepping through an array and returning its keys and values.
. foreach()—This control structure (that looks like a function) is used to step through an array, assigning the value of an element to a given variable.
. reset()—This function rewinds the pointer to the beginning of an array.
Ex:- reset($character);
This function proves useful when you are performing multiple manipulations on an array, such as sorting, extracting values, and so forth.
. array_push()—This function adds one or more elements to the end of an existing array.
Ex:- array_push($existingArray, "element 1", "element 2", "element 3");
. array_pop()—This function removes (and returns) the last element of an existing array.
Ex:- $last_element = array_pop($existingArray);
. array_unshift()—This function adds one or more elements to the beginning of an existing array.
Ex:- array_unshift($existingArray, "element 1", "element 2", "element 3");
. array_shift()—This function removes (and returns) the first element of an existing array
Ex:- $first_element = array_shift($existingArray);
. array_merge()—This function combines two or more existing arrays.
Ex:- $newArray = array_merge($array1, $array2);
. array_keys()—This function returns an array containing all the key names within a given array.
Ex:- $keysArray = array_keys($existingArray);
. array_values()—This function returns an array containing all the values within a given array.
Ex:- $valuesArray = array_values($existingArray);
. shuffle()—This function randomizes the elements of a given array.
Syntax:- shuffle($existingArray);

Working with Objects

Programmers use objects to store and organize data. Object-oriented programming is a type of programming in which the structure of the program (or application) is designed

around these objects and their relationships and interactions. Object-oriented programming structures are found in many programming languages, and are also evident in PHP.

In fact, many PHP programmers—especially those coming from a highly object-oriented programming background—choose to develop PHP applications in an object-oriented way. However, in PHP, it is not required that you write your scripts in an object-oriented manner. Many PHP scripts are procedural and functional rather than object-oriented. That is to say, the emphasis is on stepping through the use of variables, data and control structures, and subroutines and functions in the course of creating a program.

Creating an Object

Explaining the concept of an object is a little difficult if you've never encountered the concept before, because it is inherently abstract: It's a sort of theoretical box of things—variables, functions, and so forth—that exists in a templated structure called a *class*.

The input mechanisms are *methods*, and methods have properties. Look at how classes, methods, and properties work together to produce various outputs. An object exists in a structure called a *class*. In each class, you define a set of characteristics.

The whole purpose of using objects is to create reusable code. Because classes are so tightly structured but self-contained and independent of one another, you can reuse them from one application to another. Because a class is just a set of characteristics, you can pick up the code and use it in the second project, reaching into it with methods specific to the second application but using the inner workings of the existing code to achieve new results.

Syntax:-
```
class myClass
 {
//code will go here
}
```
you can create a new instance of an object:
```
$object1 = new myClass();
```

Ex: -
```
1: <?php
2: class myClass {
3: //code will go here
4: }
5: $object1 = new myClass();
6: echo "\$object1 is an ".gettype($object1)."."<br/>";
7:
8: if (is_object($object1)) {
9: echo "Really! I swear \$object1 is an object!";
10: }
11: ?>
```

The variables declared inside an object are called *properties*. It is standard practice to declare your variables at the top of the class. These properties can be values, arrays, or even other objects. The following example uses simple scalar variables inside the class, prefaced with the public keyword:

Ex:-
```
class myCar {
public$color = "silver";
public$make = "Mazda";
public$model = "Protege5";
                    }
```

Ex:-

```
1: <?php
2: class myCar {
3: public$color = “silver”;
4: public$make = “Mazda”;
5: public$model = “Protege5”;
6: }
7: $car = new myCar();
8: echo “I drive a: “.$car -> color.” “.$car -> make.” “.$car -> model;
9: ?>
```

Ex:- Changing Object Properties

```
1: <?php
2: class myCar {
3: public$color = “silver”;
4: public$make = “Mazda”;
5: public$model = “Protege5”;
6: }
7: $car = new myCar();
8: $car -> color = “red”;
9: $car -> make = “Porsche”;
10: $car -> model = “Boxter”;
11: echo “I drive a: “.$car -> color.” “.$car -> make.” “.$car -> model;
12: ?>
```

Ex:- A Class with a Method

```
1: <?php
2: class myClass {
3: function sayHello() {
4: echo “HELLO!”;
5: }
6: }
7: $object1 = new myClass();
8: $object1 -> sayHello();
9: ?>
```

A method looks and acts like a normal function but is defined within the framework of a class. The -> operator is used to call the object method in the context of your script. Had there been any variables stored in the object, the method would have been capable of accessing them for its own purposes.

Ex:- Accessing Class Properties Within a Method

```
1: <?php
2: class myClass {
3: public$name = “Jimbo”;
4: function sayHello() {
5: echo “HELLO! My name is “.$this->name;
6: }
7: }
```

```
8: $object1 = new myClass();
9: $object1 -> sayHello();
10: ?>
```

The special variable **$this** is used to refer to the currently instantiated object as you see on line 5. Anytime an object refers to itself, you must use the $this variable. Using the $this variable in conjunction with the -> operator enables you to access any property or method in a class, within the class itself. One final tidbit regarding the basics of working with an object's properties is how to change a property from within a method. Previously, a property's value changed outside the method in which it was contained. Listing 9.6 shows how to make the change from inside a method.

Ex:- Changing the Value of a Property from Within a Method

```
1: <?php
2: class myClass {
3: public$name = "Jimbo";
4: function setName($n) {
5: $this->name = $n;
6: }
7: function sayHello() {
8: echo "HELLO! My name is ".$this->name;
9: }
10: }
11: $object1 = new myClass();
12: $object1 -> setName("Julie");
13: $object1 -> sayHello();
14: ?>
```

Constructors

A *constructor* is a function that lives within a class and, given the same name as the class, is automatically called when a new instance of the class is created using new *class name*. Using constructors enables you to provide arguments to your class, which will then be processed immediately when the class is called.

Object Inheritance

Having learned the absolute basics of objects, properties, and methods, you can start to look at object inheritance. Inheritance with regard to classes is just what it sounds like: One class inherits functionality from its parent class.

Ex:- A Class Inheriting from Its Parent

```
1: <?php
2: class myClass {
3: public$name = "Matt";
4: function myClass($n) {
5: $this->name = $n;
6: }
7: function sayHello() {
8: echo "HELLO! My name is ".$this->name;
9: }
10: }
11: class childClass extends myClass {
```

```
12: //code goes here
13: }
14: $object1 = new childClass(“Baby Matt”);
15: $object1 -> sayHello();
16: ?>
```

Lines 4–6 make up a constructor. Notice that the name of this function is the same as the class in which it is contained: myClass. Lines 11–13 define a second class, child Class, that contains no code. That’s fine because, in this example, the class exists only to demonstrate inheritance from the parent class. The inheritance occurs through the extends clause shown in line 11. The second class inherits the elements of the first class because this clause is used.

Ex:- The Method of a Child Class Overriding That of Its Parent

```
1: <?php
2: class myClass {LISTING 9.8 Continued
3: public$name = “Matt”;
4: function myClass($n) {
5: $this->name = $n;
6: }
7: function sayHello() {
8: echo “HELLO! My name is “.$this->name;
9: }
10: }
11: class childClass extends myClass {
12: function sayHello() {
13: echo “I will not tell you my name.”;
14: }
15: }
16: $object1 = new childClass(“Baby Matt”);
17: $object1 -> sayHello();
18: ?>
```

Working with Strings, Dates, and Time

PHP provides many functions with which you can format and manipulate strings. Similarly, dates and times are so much a part of everyday life that it becomes easy to use them without thinking. However, because the quirks of the Gregorian calendar can be difficult to work with, PHP provides powerful tools that make date manipulation an easy task. Numerous PHP functions are available to you when it comes to the manipulations of strings, dates, and times.

Formatting Strings with PHP

PHP provides two functions that enable you first to apply formatting, whether to round doubles to a given number of decimal places, define alignment within a field, or display data according to different number systems.

The formatting options provided by printf() and sprintf().

Working with printf()

If you have any experience with a C-like programming language, you are probably familiar with the concept of the printf() function. The printf() function requires a string argument, known as a *format control string.* It also accepts additional arguments of different

types, which you learn about in a moment. The format control string contains instructions regarding the display of these additional arguments.

Example:- use printf() to output an integer as an octal (or base-8) number:

```
<?php
printf("This is my number: %o", 55);
// prints "This is my number: 67"
?>
```

Included within the format control string (the first argument) is a special code, known as a *conversion specification*. A conversion specification begins with a percent (%) symbol and defines how to treat the corresponding argument to printf(). You can include as many conversion specifications as you want within the format control string, as long as you send an equivalent number of arguments to printf().

Ex:- The following fragment outputs two floating-point numbers using printf():

```
<?php
printf("First number: %f<br/>Second number: %f<br/>", 55, 66);
// Prints:
// First number: 55.000000
// Second number: 66.000000
?>
```

The first conversion specification corresponds to the first of the additional arguments to printf(), or 55. The second conversion specification corresponds to 66. The following the percent symbol requires that the data be treated as a floating-point number.

printf() and Type Specifiers

You have already come across two type specifiers, o, which displays integers as octals, and f, which displays integers as floating-point numbers.

Type Specifiers

Specifier	Description
d	Display argument as a decimal number
b	Display an integer as a binary number
c	Display an integer as ASCII equivalent
f	Display an integer as a floating-point number (double)
o	Display an integer as an octal number (base 8)
s	Display argument as a string
x	Display an integer as a lowercase hexadecimal number (base 16)
X	Display an integer as an uppercase hexadecimal number (base 16)

These specifiers uses printf() to display a single number according to some of the type specifiers listed. Notice that the listing does not only add conversion specifications to the format control string. Any additional text included is also printed.

Ex:- Demonstrating Some Type Specifiers

```
1: <?php
2: $number = 543;
3: printf("Decimal: %d<br/>", $number);
4: printf("Binary: %b<br/>", $number);
5: printf("Double: %f<br/>", $number);
```

```
6: printf("Octal: %o<br/>", $number);
7: printf("String: %s<br/>", $number);
8: printf("Hex (lower): %x<br/>", $number);
9: printf("Hex (upper): %X<br/>", $number);
10: ?>
```

The printf() is a quick way of converting data from one number system to another and outputting the result. When specifying a color in HTML, you combine three hexadecimal numbers between 00 and FF, representing the values for red, green, and blue. You can use printf() to convert three decimal numbers between 0 and 255 to their hexadecimal equivalents:

```
<?php
$red = 204;
$green = 204;
$blue = 204;
printf("#%X%X%X", $red, $green, $blue);
// prints "#CCCCCC"
?>
```

Padding Output with a Padding Specifier

You can require that output be padded by leading characters. The padding specifier should directly follow the percent sign that begins a conversion specification. To pad output with leading zeros, the padding specifier should consist of a zero followed by the number of characters you want the output to take up. If the output occupies fewer characters than this total, zeros fill the difference:

```
<?php
printf("%04d", 36);
// prints "0036"
?>
```

To pad output with leading spaces, the padding specifier should consist of a space character followed by the number of characters that the output should occupy:

```
<?php
printf("% 4d", 36)
// prints "  36"
?>
```

You can specify any character other than a space or a zero in your padding specifier with a single quotation mark followed by the character you want to use:

```
<?php
printf ("%'x4d", 36);
// prints "xx36"
?>
```

You now have the tools you need to complete your HTML code example. Until now, you could convert three numbers to hexadecimal, but could not pad the hexadecimal values with leading zeros:

```
<?php
$red = 1;
$green = 1;
$blue = 1;
```

```
printf("#%02X%02X%02X", $red, $green, $blue);
// prints "#010101"
?>
```

Each variable is output as a hexadecimal number. If the output occupies fewer than two spaces, leading zeros are added.

Specifying a Field Width

You can specify the number of spaces within which your output should sit. A *field width specifier* is an integer that should be placed after the percent sign that begins a conversion specification (assuming that no padding specifier is defined). The following fragment outputs a list of four items, all of which sit within a field of 20 spaces. To make the spaces visible on the browser, place all the output within a pre element:

```
<?php
echo "<pre>";
printf("%20s\n", "Books");
printf("%20s\n", "CDs");
printf("%20s\n", "DVDs");
printf("%20s\n", "Games");
printf("%20s\n", "Magazines");
echo "</pre>";
?>
```

Specifying Precision

If you want to output data in floating-point format, you can specify the precision to which you want to round your data. This proves particularly useful when dealing with currency. The precision identifier should be placed directly before the type specifier. It consists of a dot followed by the number of decimal places to which you want to round. This specifier has an effect only on data that is output with the **f** type specifier:

```
<?php
printf("%.2f", 5.333333);
// prints "5.33"
?>
```

In the C language, it is possible to use a precision specifier with printf() to specify padding for decimal output. The precision specifier has no effect on decimal output in PHP. Use the padding specifier to add leading zeros to integers.

Conversion Specifications:

The below table lists the specifiers that can make up a conversion specification in the order that they would be included. Note that it is difficult to use both a padding specifier and a field width specifier. You should choose to use one or the other, but not both.

Components of Conversion Specification

Name	Description	Example
Padding specifier	Determines the number of characters that output should occupy, and the characters to add otherwise	' 4'
Field width specifier	Determines the space within which output should be formatted	'20'
Precision specifier	Determines the number of decimal places to which a double should be	'.4'

	rounded	
Type specifier	Determines the data type that should be output	'd'

Ex:- Using printf() to Format a List of Product Prices

```
1: <?php
2: $products = array("Green armchair" => "222.4",
3: "Candlestick"=> "4",
4: "Coffee table"=> "80.6");
5: echo "<pre>";
6: printf("%-20s%20s\n", "Name", "Price");
7: printf("%'-40s\n", "");
8: foreach ($products as $key=>$val) {
9: printf( "%-20s%20.2f\n", $key, $val );
10: }
11: echo "</pre>";
12: ?>
```

The first conversion specification in the format control string ("%-20s") defines a width specifier of 20 characters, with the output to be left-justified. The string type specifier is also used. The second conversion specification ("%20s") sets up a right aligned field width. This printf() call outputs our field headers.

The second printf() function call on line 7 draws a line containing - characters, 40 characters wide. You achieve this with a padding specifier, which adds padding to an empty string. The final printf() call on line 9 is part of a foreach statement that loops through the product array. The code uses two conversion specifications here—the first ("%–20s") prints the product name as a string left-justified within a 20-character field, and the second ("%20.2f") uses a field width specifier to ensure that output will be right-aligned within a 20-character field. It also uses a precision specifier to ensure that the output is rounded to two decimal places.

Storing a Formatted String

The printf() function outputs data to the browser, which means that the results are not available to your scripts. You can, however, use the function sprintf(), which is used just like printf() except that it returns a string that can be stored in a variable for later use. The following fragment uses sprintf() to round a double to two decimal places, storing the result in $cash:

```
<?php
$cash = sprintf("%.2f", 21.334454);
echo "You have \$$cash to spend.";
// Prints "You have $21.33 to spend."
?>
```

One particular use of sprintf() is to write formatted data to a file—you can call sprintf() and assign its return value to a variable that can then be printed to a file with fputs().

Investigating Strings in PHP

You do not always know everything about the data you are working with. Strings can arrive from many sources, including user input, databases, files, and web pages. Before you begin to work with data from an external source, you often need to find out more about the data. PHP provides many functions that enable you to acquire information about strings.

In fact, strings and arrays are not as different as you might imagine. You can think of a string as an array of characters, and thus you can access the individual characters of a string as if they were elements of an array:

```
<?php
$test = “phpcoder”;
echo $test[0]; // prints “p”
echo $test[4]; // prints “o”
?>
```

Finding the Length of a String with strlen() You can use the built-in strlen() function to determine the length of a string. This function requires a string as its argument and returns an integer representing the number of characters in the string you passed to it. strlen() might be used to check the length of user input, as in the following fragment, which tests a membership code to ensure that it is exactly four characters long:

```
<?php
$membership = “pAB7”;
if (strlen($membership) == 4) {
echo “<p>Thank you!</p>”;
} else {
echo “<p>Your membership number must be four characters long.</p>”;
}
?>
```

You can use the strstr() function to test whether a string exists within another string. This function requires two arguments: the source string and the substring you want to find within it. The function returns false if it cannot find the substring; otherwise, it returns the portion of the source string.

```
<?php
$membership = “pAB7”;
if (strstr($membership, “AB”)) {
echo “<p>Your membership expires soon!</p>”;
} else {
echo “<p>Thank you!</p>”;
}
?>
```

The strpos() function tells you whether a string exists within a larger string as well as where it is found. The strpos() function requires two arguments: the source string and the substring you are seeking. The function also accepts an optional third argument, an integer representing the index from which you want to start searching. If the substring does not exist, strpos() returns false; otherwise, it returns the index at which the substring begins.

```
<?php
$membership = “mz00xyz”;
if (strpos($membership, “mz”) === 0) {
echo “Hello mz!”;
}
?>
```

The substr() function returns a string based on the start index and length of the characters you are looking for. This function requires two arguments: a source string and the

starting index. Using these arguments, the function returns all the characters from the starting index to the end of the string you are searching. You can also (optionally) provide a third argument—an integer representing the length of the string you want returned. If this third argument is present, substr() returns only that number of characters, from the start index onward:

```
<?php
$test = “phpcoder”;
echo substr($test,3).”<br/>”; // prints “coder”
echo substr($test,3,2).”<br/>”; // prints “co”
?>
```

If you pass substr() a negative number as its second (starting index) argument, it will count from the end rather than the beginning of the string. The following fragment writes a specific message to people who have submitted an email address ending in.

```
<?php
$test = “pierre@wanadoo.fr”;
if ($test = substr($test, -3) == “.fr”) {
echo “<p>Bonjour! Nous avons des prix spéciaux de vous.</p>”;
} else {
echo “<p>Welcome to our store.</p>”;
}
?>
```

Tokenizing a String with strtok() You can parse a string word by word using the strtok() function. This function requires two arguments: the string to tokenize and the delimiters by which to split the string. The delimiter string can include as many characters as you want, and the function will return the first token found.

Dividing a String into Tokens with strtok()

```
1: <?php
2: $test = “http://www.google.com/search?”;
3: $test .= “hl=en&ie=UTF-8&q=php+development+books&btnG=Google+Search”;
4: $delims = “?&”;
5: $word = strtok($test, $delims);
6: while (is_string($word)) {
7: if ($word) {
8: echo $word.”<br/>”;
9: }
10: $word = strtok($delims);
11: }
12: ?>
```

Manipulating Strings with PHP

PHP provides many functions that transform a string argument, subtly or radically, as you’ll soon see. Cleaning Up a String with trim(), ltrim(), and strip_tags() When you acquire text from user input or an external file, you cannot always be sure that you haven’t also picked up whitespace at the beginning and end of your data. The trim() function shaves any whitespace characters, including newlines, tabs, and spaces, from both the start and end of a string.

```
<?php
```

```
$text = "\t\tlots of room to breathe ";
echo "<pre>$text</pre>";
// prints " lots of room to breathe ";
$text = trim($text);
echo "<pre>$text</pre>";
// prints "lots of room to breathe";
?>
```

You can use PHP's rtrim() function exactly as you would use trim(). However, rtrim() removes whitespace at only the end of the string argument:

```
<?php
$text = "\t\tlots of room to breathe ";
echo "<pre>$text</pre>";
// prints " lots of room to breathe ";
$text = rtrim($text);
echo "<pre>$text</pre>";
// prints " lots of room to breathe";
?>
```

PHP provides the ltrim() function to strip whitespace from only the beginning of a string. Once again, you call this function with the string you want to transform and it returns a new string, shorn of tabs, newlines, and spaces:

```
<?php
$text = "\t\tlots of room to breathe ";
echo "<pre>$text</pre>";
// prints " lots of room to breathe ";
$text = ltrim($text);
echo "<pre>$text</pre>";
// prints "lots of room to breathe ";
?>
```

PHP provides the strip_tags() function that accepts two arguments: the text to transform and an optional set of HTML tags that strip_tags() can leave in place. The tags in this list should not be separated by any characters:

```
<?php
$string = "<p>\"I <em>simply</em> will not have it,\" <br/>said Mr Dean.</p>
<p><strong>The end.</strong></p>";
echo strip_tags($string, "<br/><p>");
?>
```

Replacing a Portion of a String Using substr_replace()

The substr_replace() function works similarly to the substr() function, except it enables you to replace the portion of the string that you extract. The function requires three arguments: the string to transform, the text to add to it, and the starting index; it also accepts an optional length argument. The substr_replace() function finds the portion of a string specified by the starting index and length arguments, replaces this portion with the string provided, and returns the entire transformed string.

```
<?php
$membership = "mz11xyz";
```

```
$membership = substr_replace($membership, “12”, 2, 2);
echo “New membership number: $membership”;
// prints “New membership number: mz12xyz”
?>
```

Replacing Substrings Using str_replace()

The str_replace() function is used to replace all instances of a given string within another string. It requires three arguments: the search string, the replacement string, and the master string. The function returns the transformed string.

```
<?php
$string = “<h1>The 2010 Guide to All Things Good in the World</h1>”;
$string .= “<p>Site contents copyright 2010.</p>”;
echo str_replace(“2010”,”2012”,$string);
?>
```

The str_replace() function accepts arrays as well as strings for all its arguments. This enables you to perform multiple search and replace operations on a subject string, and even on more than one subject string.

```
<?php
$source = array(
“The package which is at version 4.2 was released in 2005.”,
“The year 2005 was an excellent time for PointyThing 4.2!”);
$search = array(“4.2”, “2005”);
$replace = array(“6.3”, “2012”);
$source = str_replace($search, $replace, $source);
foreach($source as $str) {
echo “$str<br>”;
}
?>
```

Converting Case

PHP provides several functions that allow you to convert the case of a string. Changing case is often useful for string comparisons. To get an uppercase version of a string, use the strtoupper() function. This function requires only the string that you want to convert and returns the converted string:

```
<?php
$membership = “mz11xyz”;
$membership = strtoupper($membership);
echo “$membership”; // prints “MZ11XYZ”
?>
```

To convert a string to lowercase characters, use the strtolower() function. Once again, this function requires the string you want to convert and returns the converted version:

```
<?php
$membership = “MZ11XYZ”;
$membership = strtolower($membership);
echo “$membership”; // prints “mz11xyz”
?>
```

PHP also provides a case function that has a useful cosmetic purpose. The ucwords() function makes the first letter of every word in a string uppercase.

```
<?php
$full_name = “violet elizabeth bott”;
$full_name = ucwords($full_name);
echo $full_name; // prints “Violet Elizabeth Bott”
?>
```

The ucfirst() function capitalizes only the first letter in a string. The following fragment capitalizes the first letter in a user-submitted string:

```
<?php
$myString = “this is my string.”;
$myString = ucfirst($myString);
echo $myString; // prints “This is my string.”
?>
```

Wrapping Text with wordwrap() and nl2br()

When you present plaintext within a web page, you are often faced with a problem in which new lines are not displayed and your text runs together into one big mess. The nl2br() function conveniently converts every new line into an HTML break.

```
<?php
$string = “one line\n”;
$string .= “another line\n”;
$string .= “a third for luck\n”;
echo nl2br($string);
?>
```

outputs

```
one line<br />
another line<br />
a third for luck<br />
```

The nl2br() function is great for maintaining newlines that are already in the text you are converting. Occasionally, you might want to add arbitrary line breaks to format a column of text. The wordwrap() function is perfect for this. wordwrap() requires one argument: the string to transform.

```
<?php
$string = “Given a long line, wordwrap() is useful as a means of “;
$string .= “breaking it into a column and thereby making it easier to read”;
echo wordwrap($string);
?>
```

The wordwrap() function does not automatically break at your line limit if a word has more characters than the limit. You can, however, use an optional fourth argument to enforce this. The argument should be a positive integer. So, using wordwrap() in conjunction with the fourth argument, you can wrap a string even when it contains words that extend beyond the limit you are setting.

```
<?php
$string = “As usual you will find me at http://www.witteringonaboutit.com/”;
$string .= “chat/eating_green_cheese/forum.php. Hope to see you there!”;
echo wordwrap($string, 24, “<br/>\n”, 1);
?>
```

Breaking Strings into Arrays with explode()

The explode() function is similar in some ways to strtok(). But explode() breaks up a string into an array, which you can then store, sort, or examine as you want. The explode() function requires two arguments: the delimiter string that you want to use to break up the source string and the source string itself. The function optionally accepts a third argument that determines the maximum number of pieces the string can be broken into.

```
<?php
$start_date = "2012-02-19";
$date_array = explode("-", $start_date);
// $date_array[0] == "2012"
// $date_array[1] == "02"
// $date_array[2] == "19"
echo $date_array[0]."-".$date_array[1]."-".$date_array[2];
//prints 2012-02-19
?>
```

Using Date and Time Functions in PHP

PHP's time() function gives you all the information you need about the current date and time. It requires no arguments and returns an integer.

```
echo time();
// sample output: 1326853185
// this represents January 17, 2012 at 09:19PM
```

Converting a Timestamp with getdate()

Now that you have a timestamp to work with, you must convert it before you present it to the user. getdate() optionally accepts a timestamp and returns an associative array containing information about the date.

The Associative Array Returned by getdate()

Key	Description	Example
Seconds	Seconds past the minute (0–59)	53
Minutes	Minutes past the hour (0–59)	44
hours	Hours of the day (0–23)	17
mday	Day of the month (1–31)	3
wday	Day of the week (0–6)	0
mon	Month of the year (1–12)	2
year	Year (four digits)	2008
yday	Day of year (0–365)	33
weekday	Day of the week (name)	Sunday
month	Month of the year (name)	February

Acquiring Date Information with getdate()

```
1: <?php
2: $date_array = getdate(); // no argument passed so today's date will be used
3: foreach ($date_array as $key => $val) {
4: echo "$key = $val<br>";
5: }
6: ?>
7: <hr/>
8: <?php
```

```
9: echo "<p>Today's date: ".$date_array['mon']."/".$date_array['mday']."/".
10: $date_array['year']."</p>";
11: ?>
```

Converting a Timestamp with date()

You can use getdate() when you want to work with the elements that it outputs. Sometimes, though, you want to display the date as a string. The date() function returns a formatted string that represents a date. You can exercise an enormous amount of control over the format that date() returns with a string argument that you must pass to it. In addition to the format string, date() optionally accepts a timestamp.

Some Format Codes for Use with date()

Format	Description	Example
a	am or pm (lowercase)	am
A	AM or PM (uppercase)	AM
d	Day of month (number with leading zeros)	01
D	Day of week (three letters)	Tue
e	Timezone identifier	America/New_York
F	Month name	January
h	Hour (12-hour format—leading zeros)	09
H	Hour (24-hour format—leading zeros)	21
g	Hour (12-hour format—no leading zeros)	9
G	Hour (24-hour format—no leading zeros)	21
i	Minutes	21
j	Day of the month (no leading zeros)	17
l	Day of the week (name)	Tuesday
L	Leap year (1 for yes, 0 for no)	1
m	Month of year (number—leading zeros)	01
M	Month of year (three letters)	Jan
n	Month of year (number—no leading zeros)	1
s	Seconds of hour	11
S	Ordinal suffix for the day of the month	th
r	Full date standardized to RFC 822	Tue, 17 Jan 2012
U	Timestamp	1326853271
y	Year (two digits)	12
Y	Year (four digits)	2012
z	Day of year (0–365)	17
Z	Offset in seconds from GMT -	18000

Formatting a Date with date()

```
1: <?php
2: $time = time(); //stores the exact timestamp to use in this script
3: echo(“m/d/y G:i:s e”, $time);
4: echo “<br/>”;
5: echo “Today is “;
```

```
6: echo date("jS \of F Y, \a\\t g:ia \i\\n e", $time);
7: ?>
```

Creating Timestamps with mktime()

You can already get information about the current time, but you cannot yet work with arbitrary dates. mktime() returns a timestamp that you can then use with date() or getdate(). mktime() accepts up to six integer arguments in the following order:

Hour
Minute
Second
Month
Day of month
Year

Creating a Timestamp with mktime()

```
1: <?php
2: // make a timestamp for Jan 17 2012 at 9:34 pm
3: $ts = mktime(21, 34, 0, 1, 17, 2012);
4: echo date("m/d/y G:i:s e", $ts);
5: echo "<br/>";
6: echo "The date is ";
7: echo date("jS \of F Y, \a\\t g:ia \i\\n e", $ts );
8: ?>
```

Testing a Date with checkdate()

You might need to accept date information from user input. Before you work with a user-entered date or store it in a database, make sure that the date is valid. checkdate() accepts three integers: month, day, and year. checkdate() returns true if the month is between 1 and 12, the day is acceptable for the given month and year (accounting for leap years), and the year is between 0 and 32767. Be careful, though: A date might be valid but not acceptable to other date functions.

For example, the following line returns true:

```
checkdate(4, 4, 1066)
```

Working with Forms

HTML forms are the principal means by which substantial amounts of information pass from the user to the server.

Creating a Simple Input Form

A Simple HTML Form

```
<!DOCTYPE html>

<html>
<head>
<title>A simple HTML form</title>
</head>
<body>
<form method="post" action="send_simpleform.php">
```

```
<p><label for="user">Name:</label><br/>
<input type="text" id="user" name="user"></p>
<p><label for="message">Message:</label><br/>
<textarea id="message" name="message" rows="5" cols="40"></textarea></p>
<button type="submit" name="submit" value="send">Send Message</button>
</form>
</body>
</html>
```

Put these lines into a text file called simpleform.html and place that file in your web server document root. This listing defines a form that contains a text field with the name "user" on line 9, a text area with the name "message" on line 11, and a submit button on line 12. The FORM element's ACTION argument points to a file called send_simpleform.php, which processes the form information. The method of this form is POST, so the variables are stored in the $_POST superglobal.

Reading Input from a Form

```
<!DOCTYPE html>
<html>
<head>
<title>A simple response</title>
</head>
<body>
<p>Welcome, <strong><?php echo $_POST['user']; ?></strong>!</p>
<p>Your message is:
<strong><?php echo $_POST['message']; ?></strong></p>
</body>
</html>
```

Put these lines into a text file called send_simpleform.php and place that file in your web server document root. Now access the form itself (simpleform.html) with your web browser.

Accessing Form Input with User-Defined Arrays

The script that receives this data has access to only a single value corresponding to this name ($_POST['products']) (and therefore only the first checkbox in the list that the user selected). You can change this behavior by renaming an element of this kind so that its name ends with an empty set of square brackets.

An HTML Form Including Multiple Check Boxes

```
<!DOCTYPE html>
<html>
<head>
<title>An HTML form with checkboxes</title>
</head>
<body>
<form action="send_formwithcb.php" method="POST">
<p><label>Name:</label><br/>
<input type="text" name="user" /></p>
```

```
<fieldset>
<legend>Select Some Products:</legend><br/>
<input type="checkbox" id="tricorder"
name="products[]" value="Tricorder">
<label for="tricorder">Tricorder</label><br/>
<input type="checkbox" id="ORAC_AI"
name="products[]" value="ORAC AI">
<label for="ORAC_AI">ORAC AI</label><br/>
<input type="checkbox" id="HAL_2000"
name="products[]" value="HAL 2000">
<label for="HAL_2000">HAL 2000</label>
</fieldset>
<button type="submit" name="submit" value="submit">Submit
Form</button>
</form>
</body>
</html>
```

Put these lines into a text file called formwithcb.html and place that file in your web server document root. Next, in the script that processes the form input, you find that the value of all checked checkboxes with the "products[]" name are available in an array called $_POST['products']. Each checkbox is indicated in lines 12 through 22.That the user's choices are made available is demonstrated in an array.

Reading Input from the Form the above file

```
<!DOCTYPE html>
<html>
<head>
<title>Reading checkboxes</title>
</head>
<body>
<p>Welcome, <strong><?php echo $_POST['user']; ?></strong>!</p>
<p>Your product choices are:
<?php
if (!empty($_POST['products'])) {
echo "<ul>";
foreach ($_POST['products'] as $value) {
echo "<li>$value</li>";
}
echo "</ul>";
} else {
echo "None";
}
?>
</body>
</html>
```

Put these lines into a text file called send_formwithcb.php and place that file in your web server document root. Now access the form (formwithcb.html) with your web browser and check some checkboxes.

Combining HTML and PHP Code on a Single Page

In some circumstances, you might want to include the form-parsing PHP code on the same page as a hard-coded HTML form. Such a combination can prove useful if you need to present the same form to the user more than once. You would have more flexibility if you were to write the entire page dynamically, of course, but you would miss out on one of the great strengths of PHP, which is that it mingles well with standard HTML. The more standard HTML you can include in your pages, the easier they are for designers and page builders to amend without asking the programmer.

For the following examples, imagine that you're creating a site that teaches basic math to preschool children and have been asked to create a script that takes a number from form input and tells the user whether it's larger or smaller than a predefined integer.

Listing 11.5 creates the HTML. For this example, you need only a single text field, but even so, the code listing includes a little PHP.

An HTML Form That Calls Itself

```
<!DOCTYPE html>
<html>
<head>
<title>An HTML form that calls itself</title>
</head>
<body>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<p><label for="guess">Type your guess here:</label> <br/>
<input type="text" id="guess" name="guess" /></p>
<button type="submit" name="submit" value="submit">Submit</button>
</form>
</body>
</html>
```

The action of this script is $_SERVER['PHP_SELF'], as shown in line 7. This global variable represents the name of the current script. In other words, the action tells the script to reload itself. The script does not produce any output, but if you upload the script to your web server, access the page, and view the source of the page, you will notice that the form action now contains the name of the script itself.

you begin to build up the PHP element of the page.

A PHP Number-Guessing Script

```
<?php
$num_to_guess = 42;
if (!isset($_POST['guess'])) {
$message = "Welcome to the guessing machine!";
```

```
} elseif (!is_numeric($_POST[‘guess’])) { // is not numeric
$message = “I don’t understand that response.”;
} elseif ($_POST[‘guess’] == $num_to_guess) { // matches!
$message = “Well done!”;
} elseif ($_POST[‘guess’] > $num_to_guess) {
$message = $_POST[‘guess’].” is too big! Try a smaller number.”;
} elseif ($_POST[‘guess’] < $num_to_guess) {
$message = $_POST[‘guess’].” is too small! Try a larger number.”;
} else { // some other condition
$message = “I am terribly confused.”;
}
?>
```

First, you must define the number that the user guesses, and this is done in line 2 when 42 is assigned to the $num_to_guess variable. Next, you must determine whether the form has been submitted. You can test for submission by looking for the existence of the variable $_POST[‘guess’], which is available only if the form script has been submitted (with or without a value in the field). If a value for $_POST[‘guess’] isn’t present, you can safely assume that the user arrived at the page without submitting a form. If the value *is* present, you can test the value it contains. The test for the presence of the $_POST[‘guess’] variable takes place on line 3. Lines 3 through 15 represent an if...elseif...else control structure. Only one of these conditions will be true at any given time, depending on what (if anything) was submitted from the form. Depending on the condition, a different value is assigned to the $message variable. That variable is then printed to the screen in line 23 which is part of the HTML portion of the script.

A PHP Number-Guessing Script (Continued)

```
<!DOCTYPE html>
<html>
<head>
<title>A PHP number guessing script</title>
</head>
<body>
<h1><?php echo $message; ?></h1>
<form action=”<?php echo $_SERVER[‘PHP_SELF’]; ?>” method=”POST”>
<p><label for=”guess”>Type your guess here:</label><br/>
<input type=”text” is=”guess” name=”guess” /></p>
<button type=”submit” name=”submit” value=”submit”>Submit</button>
</form>
</body>
</html>
```

Place the PHP and HTML code into a text file called numguess.php and put this file in your web server document root. Now access the script with your web browser.

Sending Mail on Form Submission

You’ve already seen how to take form responses and print the results to the screen, so you’re only one step away from sending those responses in an email message.

System Configuration for the mail() **Function**

- Before you can use the mail() function to send mail, you need to set up a few directives in the php.ini file so that the function works properly. Open php.ini with a text editor and look for these lines:

- For Win32 only.
- http://php.net/smtp SMTP = localhost
- http://php.net/smtp-port smtp_port = 25
- For Win32 only.
- http://php.net/sendmail-from;sendmail_from = me@example.com

If you're using Windows as your web server platform, the first two directives apply to you. For the mail() function to send mail, it must be able to access a valid outgoing mail server. If you plan to use the outgoing mail server of your choosing, the entry in php.ini could look like this:

SMTP = smtp.yourisp.net

The second configuration directive is sendmail_from, which is the email address used in the From header of the outgoing email. It can be overwritten in the mail script itself but normally operates as the default value, as in this example:

sendmail_from = youraddress@yourdomain.com

Creating the Form

The basic HTML for creating a simple feedback form named feedback.html. This form has an action of sendmail.php, which you create in the next section. The fields in feedback.html are simple: Lines 8 and 9 create a name field and label, lines 10 and 11 create the return email address field and label, and lines 12 and 13 contain the text area and label for the user's message.

Creating a Simple Feedback Form

```
<!DOCTYPE html>
<html>
<head>
<title>E-Mail Form</title>
</head>
<body>
<form action="sendmail.php" method="POST">
<p><label for="name">Name:</label><br/>
<input type="text" size="25" id="name" name="name"/></p>
<p><label for="email">E-Mail Address:</label><br/>
<input type="text" size="25" id="email" name="email"/></p>
<p><label for="msg">Message:</label><br/>
<textarea id="msg" name="msg" cols="30" rows="5"></textarea></p>
<button type="submit" name="submit" value="send">Send Message</button>
</form>
```

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

Put all the lines into a text file called feedback.html and place this file in your web server document root. Now access the script with your web browser.

Creating the Script to Send the Mail

This script differs only slightly in concept from the script, which simply printed form responses to the screen. In the script, in addition to printing the responses to the screen, you send them to an email address.

Sending the Simple Feedback Form

```
<?php
//start building the mail string
$msg  = "Name:".$_POST['name']."\n";
$msg .= "E-Mail:  ".$_POST['email']."\n";
$msg .= "Message: ".$_POST['message']."\n";
//set up the mail
$recipient = "you@yourdomain.com";
$subject = "Form Submission Results";
$mailheaders  = "From: My Web Site <defaultaddress@yourdomain.com> \n";
$mailheaders .= "Reply-To: ".$_POST['email'];
//send the mail
mail($recipient, $subject, $msg, $mailheaders);
?>
<!DOCTYPE html>
<html>
<head>
<title>Sending mail from the form in Listing 11.10</title>
</head>
<body>
<p>Thanks, <strong><?php echo $_POST['name']; ?></strong>, for your
message.</p>
<p>Your e-mail address:<strong><?php echo $_POST['email']; ?></strong></p>
<p>Your message: <br/> <?php echo $_POST['message']; ?> </p>
</body>
</html>
```

The variables printed to the screen in lines 22–26 are $_POST['name'], $_POST['email'], and $_POST['message']—the names of the fields in the form, their values saved as part of the $_POST superglobal. That's all well and good for printing the information to the screen, but in this script, you also want to create a string that's sent in email. For this task, you essentially build the email by concatenating strings to form one long message string, using the newline (\n) character to add line breaks where appropriate.

Lines 3 through 5 create the $msg variable, a string containing the values typed by the user in the form fields (and some label text for good measure). This string forms the body of the email. Note the use of the concatenation operator (.=) when adding to the $msg variable in lines 4 and 5.

Lines 8 and 9 are hard-coded variables for the email recipient and the subject of the email message. Replace you@yourdomain.com with your own email address, obviously. If you want to change the subject, feel free to do that, too!

Lines 10 and 11 set up some mail headers, namely the From: and Reply to: headers. You could put any value in the From: header; this is the information that displays in the From or Sender column of your email application when you receive this mail.

Put these lines into a text file called sendmail.php and place that file in your web server document root. Use your web browser and go back to the form, enter some information, and click the submission button.

If you then check your email, you should have a message waiting for you.

PHP Cookies

A cookie is often used to identify a user.

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

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

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Note: The setcookie() function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>
<?php
echo "Cookie 'user' is deleted.";
?>
</body>
</html>
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>
<?php
if(count($_COOKIE) > 0) {
   echo "Cookies are enabled.";
```

```
} else {
   echo "Cookies are disabled.";
}
?>
</body>
</html>
```

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

Note: The session_start() function must be the very first thing in your document. Before any HTML tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body>
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
print_r($_SESSION);
?>
</body>
</html>
```

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>
</body>
</html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
?>
</body>
</html>
```

6. IMAGES WITH PHP

- **Working with GD Library**
  - **What is the GD Library?**
    - The GD library is used for dynamic image creation. From PHP we use with the GD library to create GIF, PNG or JPG images instantly from our code. This allows us to do things such as create charts on the fly, created an an anti-robot security image, create thumbnail images, or even build images from other images.
    - If you are unsure if you have GD library, you can run phpinfo() to check that GD Support is enabled. If you don't have it, you can download it for free

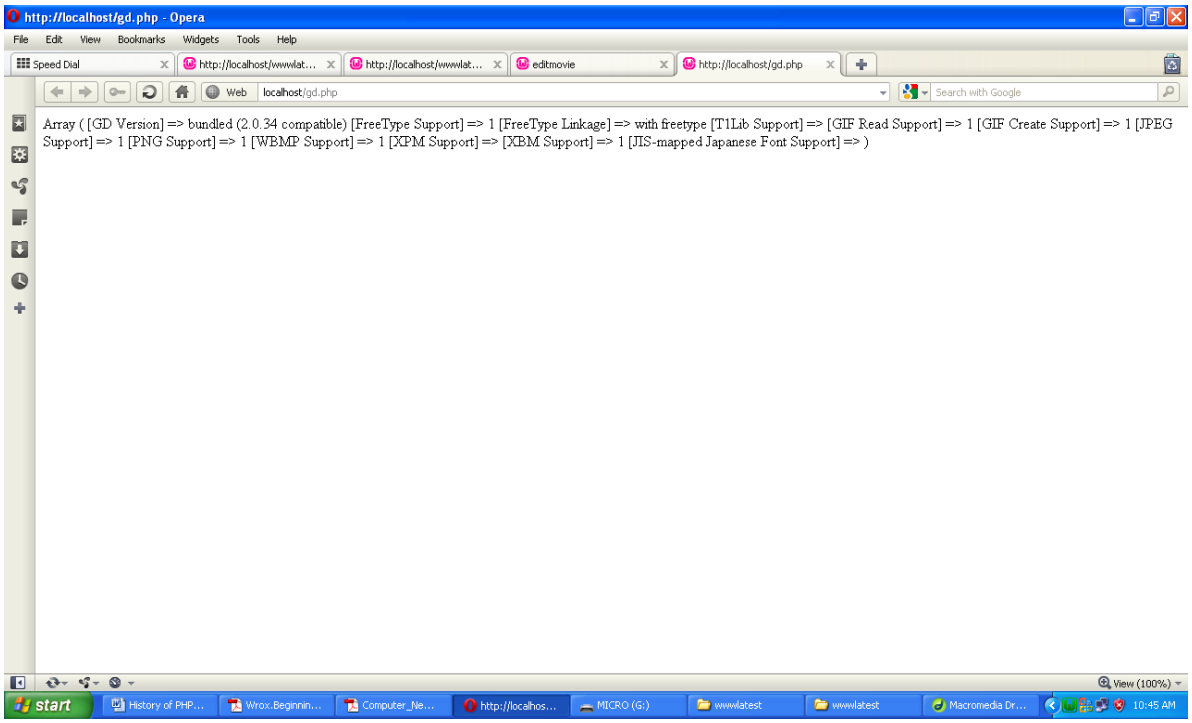
- **File types with GD and PHP**
  - GD itself can work with a multitude of images, but when you use it with PHP you can get information about any GIF, JPG, PNG, SWF, SWC, PSD, TIFF, BMP, IFF, JP2, JPX, JB2, JPC, XBM, or WBMP image format.
  - You can also manipulate and create images in GIF, JPG, PNG, WBMP, and XBM image formats.
  - GD can also allow PHP to create shapes such as squares, polygons, and ellipses, as well as text boxes using True Type Fonts.
  - Depending on your version of GD, GIF support may or may not be enabled. You can tell if GIF support is enabled with the use of the gd_info function described in the "Try It Out - Testing Your GD" section that follow

- **Compiling PHP with GD**
  - If you are using a Web host, chances are that they have already enabled GD on their installation of PHP.
  - If you are running your own server, enabling the GD functions in PHP is really not so hard. In Windows, simply find the following line in your php.ini file:

    ;extension=php_gd2.dll
  - Uncomment that line like this:

    extension=php_gd2.dll
  - You need to restart Apache for your changes to take effect.
  - In Linux, you need to enable GD with the --with-gd configure option.
  - Again, because the bundled version of GD is recommended for use with PHP, you do not need to identify GD's installation directory— it will find the bundled version by default.

  - Make sure everything is working properly before you go any further:
  - **Open your favorite text/HTML editor and enter the following code:**
    - <?php
      - print_r(gd_info());
    - ?>
  - Save this file as gdtest.php (and upload it to your Web server if needed).
  - Load your favorite browser and view the page that you have just uploaded. Your screen should look like the one shown in Figure.



- **How It Works**
  - The gd_info function is quite useful, because it tells you what capabilities you have with the current version of GD that was bundled with your version of PHP.
  - Its purpose is to put all the information about the GD version into an array that you can then view.
  - This not only serves as a test to make sure that GD and PHP are playing nice with each other, but it lets you see what your limitations are for using the GD functions in PHP.
  - For the purposes of the examples in this chapter, you will need to have JPG, GIF, and PNG support.
  - If your version of GD doesn't support any of these image types, you will need to upgrade. You can find full upgrade instructions and source files at http://www.boutell.com/gd.
  - The print_r() function takes all the information stored in a variable (including arrays) and outputs it to the browser so you can see it.
  - Now that you know that your GD library is working well, and what image types are supported in your version, let's move along.

❖ **Creating the image table**

- First, you need to create a table that will hold information about your images.
- You are going to store basic information about each image, such as the user's name and the title of the image.
- Then, give your users a form where they can submit an image for display on your site.
- You will ask some basic information about the image, then you will let users upload the file directly from the comfort of their own browser, without the aid of any FTP software.
- **Step:-1**
  - If you do not have a directory to house your images, you will need to create one. In this exercise, the images will be stored in /images/.
  - **Open your text editor and type the following:**

```
<?php
        $con = mysql_connect("localhost","root")
        if(!$con)
```

```
{
        die(mysql_error());
}
mysql_select_db("test", $con);
$sql = "CREATE TABLE images(
image_id INT(11) NOT NULL AUTO_INCREMENT,
image_caption VARCHAR(255) NOT NULL,
image_username VARCHAR(255) NOT NULL,
PRIMARY KEY (image_id)
)";
$a = mysql_query($sql,$con)
if($a)
{
        echo "Image table successfully created.";
}
else
{
        die(mysql_error());
}
?>
```

- **Step:-2**
  - Save this file as test.php. Open this file in your browser and you should see the message "Image table successfully created."

❖ **Uploading the image**

- There is some debate about whether or not actual images can be efficiently stored in a database using the blob MySQL column type.
- My personal preference is not to store the actual image, but to store only information about the image, and if needed, a link to the image.
- The images themselves are then stored in a regular old directory of your choosing. That being said, create a table in your movie review database that will store the links to the images your users upload.
- **Step:-1**

```
create database test
```

- **Step:-2**

```
CREATE TABLE photos (
        id int(11) NOT NULL AUTO_INCREMENT,
        location varchar(100) NOT NULL,
        caption varchar(100) NOT NULL,
        PRIMARY KEY (id)
) ;
```

- **Step:-3**

```
<html>
<body>
<form action="upload.php" method="post" enctype="multipart/form-data">
        Select Image: <br />
        <input type="file" name="image" ><br />
        Caption<br />
```

```
        <input name="caption" type="text"   />
        <br />
         <input type="submit" name="Submit" value="Upload"  />
</form>
</body>
</html>
```

NOTE:- The first parameter is the form's input name and the second index can be either "name", "type", "size", "tmp_name" or "error". Like this:

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

- **move_uploaded_file()**
  - he move_uploaded_file() function moves an uploaded file to a new location.
  - This function returns TRUE on success, or FALSE on failure.
  - Syntax
    - move_uploaded_file(file,newloc)

Parameter	Description
file	Required. Specifies the file to be moved
newloc	Required. Specifies the new location for the file

  - Tips and Notes
    - **Note:** This function only works on files uploaded via HTTP POST.
    - **Note:** If the destination file already exists, it will be overwritten.

Step:-4

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        if (!isset($_FILES['image']['tmp_name']))
        {
                echo "error";
        }
        else
        {
                move_uploaded_file($_FILES["image"]["tmp_name"] , "photos/" .
                $_FILES["image"]["name"]);
                $location="photos/" . $_FILES["image"]["name"];
                $caption=$_POST['caption'];
                $save=mysql_query("INSERT INTO photos (location, caption) VALUES
                ('$location','$caption')");
                header("location: photoview.php");
```

```
                exit();
        }
?>

Step:-5
        <?php
                $con = mysql_connect("localhost","root","");
                mysql_select_db("test",$con);
                $sql="SELECT * FROM photos";
                $a = mysql_query($sql,$con);
                while($row = mysql_fetch_array($a))
                {
                        echo '<div>';
                        echo '<p><img src="'.$row['location'].'"></p>';
                        echo '<p id="caption">'.$row['caption'].' </p>';
                        echo '</div>';
                }
        ?>
```

- **Drawing Lines**
  - To draw a line in an image use the imageline() function.
  - A line has both start and end points, so you must give imageline() two sets of coordinates.
    - Line color can be modified using the imagecolorallocate() function, which we learned about before. It should be stored in a variable to be used later.
    - Line thickness can be modified using the imagesetthickness() function, which requires two parameters: imagesetthickness(image, thickness)
  - The imageline() function itself requires 6 parameters.
  - The syntax is:
    - **imageline(image, x1, y1, x2, y2, color)**
    - image = Refers to the Image Resource That the Line Will Be Applied to
      - x1 = x-coordinate For First Point
      - y1 = y-coordinate For First Point
      - x2 = x-coordinate For Second Point
      - y2 = y-coordinate For Second Point
      - color = Refers to the Line Color Identifier Created With imagecolorallocate()
    - **Example:-**

```
<?php
    header('Content-type: image/png');
    $png_image = imagecreate(150, 150);
    imagecolorallocate($png_image, 15, 142, 210);
    $black = imagecolorallocate($png_image, 0, 0, 0);
    imageline($png_image, 0, 0, 150, 150, $black);
    imagepng($png_image);
    imagedestroy($png_image);
?>
```

❖ **Drawing Rectangles**

- Draw Rectangle use for imagerectangle() fiunction
- The imageline() function itself requires 6 parameters.
- The syntax is:
  - **imagerectangle(image, x1, y1, x2, y2, color)**
    - image = Refers to the Image Resource That the Line Will Be Applied to
    - x1 = x-coordinate For First Point
    - y1 = y-coordinate For First Point
    - x2 = x-coordinate For Second Point
    - y2 = y-coordinate For Second Point
    - color = Refers to the Line Color Identifier Created With imagecolorallocate()
- **Example:-**

```
<?php
    // Create a 200 x 200 image
    $canvas = imagecreatetruecolor(200, 200);

    // Allocate colors
    $pink = imagecolorallocate($canvas, 255, 105, 180);
    $green = imagecolorallocate($canvas, 132, 135, 28);

    // Draw three rectangles each with its own color
    imagerectangle($canvas, 50, 50, 150, 150, $pink);
    imagerectangle($canvas, 100, 120, 75, 160, $green);

    // Output and free from memory
    header('Content-Type: image/jpeg');
    imagejpeg($canvas);
    imagedestroy($canvas);
?>
```

❖ **Drawing Circles and Ellipses**

- **Draw circle and ellipses using imageellipse() function**
- **The syntax is:**
- **imageellipse ($image ,$cx ,$cy ,$width ,$height ,$color)**
  - image:-An image resource, returned by one of the image creation functions, such as imagecreatetruecolor().
  - Cx:- x-coordinate of the center.
  - Cy:- y-coordinate of the center.
  - Width:- The ellipse width.
  - Height:- The ellipse height.
  - Color:- The color of the ellipse. A color identifier created with imagecolorallocate().
- **Example:-**

```
<?php

// Create a blank image.
$image = imagecreatetruecolor(400, 300);

// Select the background color.
$bg = imagecolorallocate($image, 0, 0, 0);
```

```
// Fill the background with the color selected above.
imagefill($image, 0, 0, $bg);

// Choose a color for the ellipse.
$col_ellipse = imagecolorallocate($image, 255, 255, 255);

// Draw the ellipse.
imageellipse($image, 200, 150, 300, 200, $col_ellipse);

// Output the image.
header("Content-type: image/png");
imagepng($image);

?>
```

- **Drawing an Arc**
  - **Draw Arc using imagearc() function**
  - Imagearc — Draws an arc
  - **imagearc ($image ,$cx ,$cy ,$width ,$height, $start , $end ,$color)**
    - image:-An image resource, returned by one of the image creation functions, such as imagecreatetruecolor().
    - Cx:- x-coordinate of the center.
    - Cy:- y-coordinate of the center.
    - Width:- The ellipse width.
    - Height:- The ellipse height.
    - Start:-The arc start angle, in degrees.
    - End:-The arc end angle, in degrees. 0° is located at the three-o'clock position, and the arc is drawn clockwise.
    - Color:- The color of the ellipse. A color identifier created with imagecolorallocate().
  - **Example:-**

```
<?php

// create a 200*200 image
$img = imagecreatetruecolor(200, 200);

// allocate some colors
$white = imagecolorallocate($img, 255, 255, 255);
$red   = imagecolorallocate($img, 255,   0,   0);
$green = imagecolorallocate($img,   0, 255,   0);
$blue  = imagecolorallocate($img,   0,   0, 255);

// draw the head
imagearc($img, 100, 100, 200, 200,  0, 360, $white);
// mouth
imagearc($img, 100, 100, 150, 150, 25, 155, $red);
// left and then the right eye
imagearc($img,  60,  75,  50,  50,  0, 360, $green);
imagearc($img, 140,  75,  50,  50,  0, 360, $blue);
```

```
// output image in the browser
header("Content-type: image/png");
imagepng($img);

// free memory
imagedestroy($img);

?>
```

- **Drawing Polygons**
  - imagepolygon — Draws a polygon
  - **Draw Polygons using imagepolygon () function**
  - **Syntax:-**
    - **imagepolygon** ($image ,$points , $num_points ,$color)
    - **image:-** An image resource, returned by one of the image creation functions, such as imagecreatetruecolor().
    - **Points:-** An array containing the polygon's vertices, e.g.:

points[0]	= x0
points[1]	= y0
points[2]	= x1
points[3]	= y1

    - **num_points:-** Total number of points (vertices).
    - **Color:-** A color identifier created with imagecolorallocate().
  - **Example:-**

```
<?php
// Create a blank image
$image = imagecreatetruecolor(400, 300);

// Allocate a color for the polygon
$col_poly = imagecolorallocate($image, 255, 255, 255);

// Draw the polygon
imagepolygon($image, array( 0,  0,  100, 200,  300, 200  ),  3,  $col_poly);

// Output the picture to the browser
header('Content-type: image/png');

imagepng($image);
imagedestroy($image);
?>
```

7. INTRODUCTION TO MYSQL

❖ **What is MySQL?**

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

- The data in MySQL is stored in tables. A table is a collection of related data, and it consists of columns and rows.
- Databases are useful when storing information categorically. A company may have a database with the following tables:
  - Employees
  - Products
  - Customers
  - Orders

❖ **MySQL structure and syntax**

- **MySQL Structure:**
  - Because MySQL is a relational database management system, it allows you to separate information into tables or areas of pertinent information.
  - In non relational database systems, all the information is stored in one big area, which makes it much more difficult and cumbersome to sort and extract only the data you want.
  - In MySQL, each table consists of separate fields, which represent each bit of information.
  - For example, one field could contain a **customer's** first name, and another field could contain his last name.
  - Fields can hold different types of data, such as text, numbers, dates, and so on.
  - You create database tables based on what type of information you want to store in them.
  - The separate tables of MySQL are then linked together with some common denominator, where the values of the common field are the same.
  - For an example of this structure, imagine a table that includes a customer's name, address, and ID number, and another table that includes the customer's ID number and past orders he has placed.
  - The common field is the customer's ID number, and the information stored in the two separate tables would be linked together via fields where the ID number is equal.
  - This enables you to see all the information related to this customer at one time.

- **MySQL Syntax:**
  - PHP provides various functions to access MySQL database and to manipulate data records inside MySQL database. You would require to call PHP functions in the same way you call any other PHP function.
  - The PHP functions for use with MySQL have the following general format:
    - **mysql_function(value,value,...);**

- The second part of the function name is specific to the function, usually a word that describes what the function does. The following are two of the functions which we will use in our tutorial
- **mysql_connect($connect);**
- **mysql_query($connect,"SQL statement");**
- Following example shows a generic syntax of PHP to call any MySQL function.

```
<html>
<head>
    <title>PHP with MySQL</title>
</head>
<body>
    <?php
      $a = mysql_function (value, [value,...]);
      if( !$a)
      {
         die ( "Error: a related error message" );
      }
      // Otherwise MySQL  or PHP Statements
    ?>
</body>
</html>
```

❖ **MySQL Data types (Field Types)**

➢ **Numeric Data Types:**

- **INT**
  - **Enter to the integer value.**
  - A normal-sized integer that can be signed or unsigned.
  - If signed, the allowable range is from -2147483648 to 2147483647.
  - If unsigned, the allowable range is from 0 to 4294967295.
  - You can specify a width of up to 11 digits.

- **TINYINT**
  - **Enter to the integer value**
  - A very small integer that can be signed or unsigned.
  - If signed, the allowable range is from -128 to 127.
  - If unsigned, the allowable range is from 0 to 255.
  - You can specify a width of up to 4 digits.

- **SMALLINT**
  - **Enter to the integer value**
  - A small integer that can be signed or unsigned.
  - If signed, the allowable range is from -32768 to 32767.
  - If unsigned, the allowable range is from 0 to 65535.
  - You can specify a width of up to 5 digits.

- **MEDIUMINT**
  - **Enter to the integer value**
  - A medium-sized integer that can be signed or unsigned.
  - If signed, the allowable range is from -8388608 to 8388607.
  - If unsigned, the allowable range is from 0 to 16777215.
  - You can specify a width of up to 9 digits.

- **BIGINT**
  - **Enter to the integer value**
  - A large integer that can be signed or unsigned.
  - If signed, the allowable range is from -9223372036854775808 to 9223372036854775807.
  - If unsigned, the allowable range is from 0 to 18446744073709551615.
  - You can specify a width of up to 11 digits.

- **FLOAT(M,D)**
  - **Enter to the Float value**
  - A floating-point number that cannot be unsigned.
  - You can define the display length (M) and the number of decimals (D).
  - This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals).
  - Decimal precision can go to 24 places for a FLOAT.

- **DOUBLE(M,D)**
  - A double precision floating-point number that cannot be unsigned.
  - You can define the display length (M) and the number of decimals (D).
  - This is not required and will default to 16,4, where 4 is the number of decimals.
  - Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.

- **DECIMAL(M,D)**
  - An unpacked floating-point number that cannot be unsigned.
  - In unpacked decimals, each decimal corresponds to one byte.
  - Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

➢ **Date and Time Types:**

- **DATE**
  - A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31.
  - For example, December 30th, 1973 would be stored as 1973-12-30.

- **DATETIME**
  - A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59.
  - For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.

- **TIMESTAMP**
  - A timestamp between midnight, January 1, 1970 and sometime in 2037.
  - This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).

- **TIME**
  - Stores the time in HH:MM:SS format.

- **YEAR(M)**
  - Stores a year in 2-digit or 4-digit format.
  - If the length is specified as 2 (for example YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155. The default length is 4.

➢ **String Types:**

- **CHAR(M)**
  - A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored.
  - Defining a length is not required, but the default is 1.

- **VARCHAR(M)**
  - A variable-length string between 1 and 255 characters in length; for example VARCHAR(25).
  - You must define a length when creating a VARCHAR field.

- **BLOB or TEXT**
  - A field with a maximum length of 65535 characters.
  - BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files.
  - Fields defined as TEXT also hold large amounts of data; the difference between the two is that sorts and comparisons on stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields.
  - You do not specify a length with BLOB or TEXT.

- **TINYBLOB or TINYTEXT**
  - A BLOB or TEXT column with a maximum length of 255 characters.
  - You do not specify a length with TINYBLOB or TINYTEXT.

- **MEDIUMBLOB or MEDIUMTEXT**
  - A BLOB or TEXT column with a maximum length of 16777215 characters.
  - You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.

- **LONGBLOB or LONGTEXT**
  - A BLOB or TEXT column with a maximum length of 4294967295 characters.
  - You do not specify a length with LONGBLOB or LONGTEXT.

- **ENUM**
  - An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL).
  - For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

❖ **Field Modifier in MySQL**

➢ **NOT NULL or NULL**

- When a column is defined as NOT NULL, it means that a value must be entered into the column if the record is to be accepted for storage in table
- Syntax: columnname datatype(size) NOT NULL
- Syntax: columnname datatype(size) NULL

- **Example:**

```
Create table student
        (id varchar(6) NOT NULL,
                Name varchar(20) NOT NULL,
                City varchar(15) NULL  )
```

- In the above query the user has to specify values for the Id, name fields otherwise the record will not be inserted into the table.
- In the above query the user has not specify values in city.

➢ **DEFAULT**

- The DEFAULT constraint is used to insert a default value into a column.
- The default value will be added to all new records, if no other value is specified.
- **Example:**

```
Create table student
        (id varchar(6) NOT NULL,
                Name varchar(20) NOT NULL,
                City varchar(15) DEFAULT 'rahul'  )
```

➢ **AUTO_INCREMENT**

- Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.
- We would like to create an auto-increment field in a table.
- MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

- **Example:**

```
Create table student
        (id INT(5) AUTO_INCREMENT ,
                Name varchar(20),
                City varchar(15) ,PRIMARY KEY (Id))
```

❖ **Types of MySQL tables and storages engines**

❖ **Storages engines**

➢ **What is a storage engine**
- A storage engine is a software which a DataBase management System uses to create, read, update and delete data from a database.
- Storage engines are also called as DataBase Engines.

➢ **The current version of MySQL uses five main types of storage engines to store and update data in those tables:**
- **MyISAM**
- **MERGE**
- **MEMORY (formerly known as HEAP)**
- **InnoDB**
- **BDB**

➢ **MyISAM**

- Default storage engine, manages non transactional tables, provides high-speed storage and retrieval, supports full text searching.
- Each MyISAM table is stored on disk in three files.
  - The files have names that begin with the table name and have an extension to indicate the file type.
  - **An .frm file stores the table format.**
  - **The data file has an .MYD (MYData) extension.**
  - **The index file has an .MYI (MYIndex) extension.**
- To specify explicitly that you want a MyISAM table, indicate that with an ENGINE table option:
  - **CREATE TABLE t (i INT) ENGINE = MYISAM;**
- MyISAM tables have the following characteristics:
  - All data values are stored with the low byte first.
  - All numeric key values are stored with the high byte first to permit better index compression.
  - Large files (up to 63-bit file length) are supported on file systems and operating systems that support large files.
  - There is a limit of 2^{32} (~4.295E+09) rows in a MyISAM table
  - The maximum number of indexes per MyISAM table is 64.
  - The maximum number of columns per index is 16.
  - The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.
  - BLOB and TEXT columns can be indexed.
  - NULL values are permitted in indexed columns. This takes 0 to 1 bytes per key.
- Each character column can have a different character MyISAM also supports the following features:
  - Support for a true VARCHAR type; a VARCHAR column starts with a length stored in one or two bytes.
  - Tables with VARCHAR columns may have fixed or dynamic row length.
  - The sum of the lengths of the VARCHAR and CHAR columns in a table may be up to 64KB.

- Arbitrary length UNIQUE constraints

➢ **MERGE**

- The MERGE storage engine, introduced in MySQL version 3.23.25, provides a way to combine many identical MyISAM tables into one logical table.
- These tables are often referred to as MERGE tables.
- It's useful in many ways.
- One advantage it has relates to table size limits. Since the data for each MyISAM table is stored in a seperate file on disk, it allows administrators to overcome the maximum file size constraints imposed by certain filesystems.
- You cannot merge tables in which the columns are listed in a different order, don't have exactly the same columns or have the indexes in different order.
- When you create a MERGE table, MySQL creates two files on disk.
- The files have names that begin with the table name and have an extension to indicate the file type.
- An .frm file stores the table definition, and an .MRG file contains the names of the tables that should be used as one.
- Originally, all tables used had to be in the same database as the MERGE table itself. This restriction has been lifted as of MySQL 4.1.1.
- You can use SELECT, DELETE, UPDATE, and (as of MySQL 4.0) INSERT statements on the collection of tables. For the moment, you must have SELECT, UPDATE, and DELETE privileges on the tables to which you map a MERGE table.
- If you DROP the MERGE table, you are deleting only the MERGE specification. The underlying tables are not affected.
- Example shows how to create a MERGE table:

```
CREATE TABLE Student1 (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name CHAR(20)
    );

CREATE TABLE Student2 (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name CHAR(20)
    );

INSERT INTO Student1 (name)
    VALUES (' rahul '),('kumar'),('patel');

INSERT INTO Student2 (name)
    VALUES ('ravi'),('kumar'),('patel');

CREATE TABLE total (
    id INT NOT NULL AUTO_INCREMENT,
    name CHAR(20), INDEX(id)
    )

TYPE=MERGE UNION=( Student1, Student2) INSERT_METHOD=LAST;
```

- Note that the a column is indexed in the MERGE table, but is not declared as a PRIMARY KEY as it is in the underlying MyISAM tables.
- This is necessary because a MERGE table cannot enforce uniqueness over the set of underlying tables.
- After creating the MERGE table, you can do things like this:
- **SELECT * FROM total;**

id	name
1	rahul
2	kumar
3	patel
1	ravi
2	kumar
3	patel

➢ **MEMORY (formerly known as HEAP)**

- The MEMORY storage engine creates tables with contents that are stored in memory.
- Formerly, these were known as HEAP tables. MEMORY is the preferred term, although HEAP remains supported for backward compatibility.
- The MEMORY storage engine associates each table with one disk file.
- The file name begins with the table name and has an extension of .frm to indicate that it stores the table definition.
- To specify that you want to create a MEMORY table, indicate that with an ENGINE table option:

  CREATE TABLE t (i INT) ENGINE = MEMORY;
- The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.
- As indicated by the engine name, MEMORY tables are stored in memory.
- They use hash indexes by default, which makes them very fast, and very useful for creating temporary tables.
- However, when the server shuts down, all rows stored in MEMORY tables are lost.
- The tables themselves continue to exist because their definitions are stored in .frm files on disk, but they are empty when the server restarts.
- Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts
- MEMORY tables can have up to 64 indexes per table, 16 columns per index and a maximum key length of 3072 bytes.
- The MEMORY storage engine supports both HASH and BTREE indexes. You can specify one or the other for a given index by adding a USING clause as shown here:

  CREATE TABLE lookup

  (id INT, INDEX USING HASH (id))ENGINE = MEMORY;

  CREATE TABLE lookup

  (id INT, INDEX USING BTREE (id))ENGINE = MEMORY;
- MEMORY tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)
- Columns that are indexed can contain NULL values.

- MEMORY tables use a fixed-length row-storage format. Variable-length types such as VARCHAR are stored using a fixed length.
- MEMORY tables cannot contain BLOB or TEXT columns.
- MEMORY includes support for AUTO_INCREMENT columns.

➢ **InnoDB**
- **InnoDB Limitation**
  - No full text indexing
  - Cannot be compressed for fast, read-only
- ACID compliant and hence fully transactional with ROLLBACK and COMMIT and support for Foreign Keys
- Rackspace Cloud Default Engine
- Auto recovery from crash via replay of logs
- Dirty pages converted from random to sequential before commit and flush to disk
- Row data stored in pages in PK order
- Row level locking
- To specify explicitly that you want a **InnoDB** table, indicate that with an ENGINE table option:
  - **CREATE TABLE t (i INT) ENGINE = InnoDB;**

❖ **MyISAM vs Innodb - Quick comparison Table:**

MyISAM	Innodb
Not *ACID compliant and non-transactional	*ACID compliant and hence fully transactional with ROLLBACK and COMMIT and support for Foreign Keys
MySQL 5.0 Default Engine	Rackspace Cloud Default Engine
Offers Compression	Offers Compression
Requires full repair/rebuild of indexes/tables	Auto recovery from crash via replay of logs
Changed Db pages written to disk instantly	Dirty pages converted from random to sequential before commit and flush to disk
No ordering in storage of data	Row data stored in pages in PK order
Table level locking	Row level locking

➢ **BDB**
- Each BDB table is stored on disk in two files.
- The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format, and a .db file contains the table data and indexes.
- To specify explicitly that you want a BDB table, indicate that with an ENGINE table option:
  - **CREATE TABLE t (i INT) ENGINE = BDB;**
- The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.
- BerkeleyDB is a synonym for BDB in the ENGINE table option.
- The BDB storage engine provides transactional tables. The way you use these tables depends on the autocommit mode:
  - If you are running with autocommit enabled (which is the default), changes to BDB tables are committed immediately and cannot be rolled back.

- If you are running with autocommit disabled, changes do not become permanent until you execute a COMMIT statement. Instead of committing, you can execute ROLLBACK to forget the changes.

- You can start a transaction with the START TRANSACTION or BEGIN statement to suspend autocommit, or with SET autocommit = 0 to disable autocommit explicitly.
- The BDB storage engine has the following characteristics:
  - BDB tables can have up to 31 indexes per table, 16 columns per index, and a maximum key size of 1024 bytes.
  - MySQL requires a primary key in each BDB table so that each row can be uniquely identified.
  - The primary key is faster than any other index, because it is stored together with the row data.
  - This behavior is similar to that of InnoDB, where shorter primary keys save space not only in the primary index but in secondary indexes as well.
  - SELECT COUNT(*) FROM tbl_name is slow for BDB tables, because no row count is maintained in the table.

❖ **MySQL commands**

➢ **What is DDL, DML, DCL**

- Data Definition Language deals with database schemas and descriptions of how the data should reside in the database, therefore language statements like CREATE TABLE or ALTER TABLE belong to DDL. DML deals with data manipulation, and therefore includes most common SQL statements such SELECT, INSERT, etc. Data Control Language includes commands such as GRANT, and mostly concerns with rights, permissions and other controls of the database system.

➢ **DDL**

- Data Definition Language (DDL) statements are used to define the database structure or schema.
  - **CREATE - to create objects in the database**
    - ♦ For creating a database CREATE DATABASE statement is used in MySQL.
    - ♦ **Syntax:-**

      CREATE DATABASE database_name;
    - ♦ **Example:**

```
<?php
    $con = mysql_connect("localhost","root","");
    $sql=mysql_query("CREATE DATABASE test",$con);
    if ($sql)
    {
            echo "Database created";
    }
    else
    {
            echo "Error creating database: " . mysql_error();
    }
?>
```

- **Create table in database**
  - The CREATE TABLE statement is used to create a table in MySQL
  - **Syntax:-**

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

  - **Example:**

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "CREATE TABLE student
        (id INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
        name VARCHAR( 20 ) NOT NULL ,
        city TEXT NOT NULL )";
        $a=mysql_query($sql,$con);
        if($a)
        {
                echo "Table created successfully";
        }
        else
        {
                 die('Could not create table: ' . mysql_error());
        }
        mysql_close($con);
?>
```

- **ALTER**
  - Alters the structure of the database
  - **Syntax:-**
    1. **ALTER TABLE table_name DROP column _name;**
    2. **ALTER TABLE table_name ADD column _name Datatype(Size);**
    3. **ALTER TABLE table_name MODIFY column _name Datatype(Size);**
  - MySQL **ALTER** command is very useful when you want to change a name of your table, any table field or if you want to add or delete an existing column in a table.
  - Let's begin with creation of a table called **user**

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "create table user(id INT,name CHAR(1))";
        $a=mysql_query($sql,$con);
        if($a)
        {
                echo "Table created successfully";
        }
        else
```

```
{
        die('Could not create table: ' . mysql_error());
}
mysql_close($con);
?>
```

- **SHOW COLUMNS FROM user;**

Field	Type	Null	Key	Default	Extra
id	int(11)	YES		*NULL*	
name	char(1)	YES		*NULL*	

- **Dropping, Adding, or Repositioning a Column:**
  - Suppose you want to drop an existing column **name** from above MySQL table then you will use DROP clause along with ALTER command as follows
  - Example:

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "ALTER TABLE user  DROP name";
        mysql_query($sql,$con);
        mysql_close($con);
?>
```

  - A DROP will not work if the column is the only one left in the table.
  - **SHOW COLUMNS FROM user;**

Field	Type	Null	Key	Default	Extra
id	int(11)	YES		*NULL*	

  - To add a column, use ADD and specify the column definition. The following statement restores the **city column to user**
  - **Example:**

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "ALTER TABLE user ADD city varchar(15)";
        // Execute query
        mysql_query($sql,$con);
        mysql_close($con);
?>
```

- After issuing this statement, user alter will contain the same two columns that it had when you first created the table, but will not have quite the same structure. That's because new columns are added to the end of the table by default. So even though city originally was the first column in mytbl, now it is the last one:
- **SHOW COLUMNS FROM user;**

Field	Type	Null	Key	Default	Extra
id	int(11)	YES		*NULL*	
city	varchar(15)	YES		*NULL*	

- **Changing a Column Definition or Name:**
  - To change a column's definition, use **MODIFY** or **CHANGE** clause along with ALTER command. For example, to change column **city** from **varchar(15)** to CHAR(10), do this:

- **Example:-**

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "ALTER TABLE user MODIFY city CHAR(10);"
        mysql_query($sql,$con);
        mysql_close($con);
?>
```

- **DROP**
  - delete objects from the database
  - **Syntax:-**
    - **DROP DATABASE database_name;**
    - **DROP TABLE table_name;**
  - **Drop to the database**

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "DROP DATABASE test";
        mysql_query($sql,$con);
        mysql_close($con);
?>
```

  - Above MySQL code to drop the database
  - **Drop to the table**

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "DROP TABLE student";
        mysql_query($sql,$con);
        mysql_close($con);
?>
```

- **TRUNCATE**
  - remove all records from a table, including all spaces allocated for the records are removed
  - **Syntax:-**
    - **TRUNCATE TABLE table_name";**
  - Example:-

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "TRUNCATE TABLE employee";
        mysql_query($sql,$con);
        mysql_close($con);
?>
```

- **RENAME**
  - **rename an object**
  - To rename a table, use the **RENAME** option of the ALTER TABLE statement.
  - Try out following example to rename user to employee
  - **Example:-**

```
<?php
        $con = mysql_connect("localhost","root","");
        mysql_select_db("test",$con);
        $sql = "ALTER TABLE user RENAME TO employee";
        mysql_query($sql,$con);
        mysql_close($con);
?>
```

- **DML**
  - **INSERT**
    - To insert data into MySQL table you would need to use SQL **INSERT INTO** command.
    - You can insert data into MySQL table by using mysql> prompt or by using any script like PHP.
    - Syntax:
      - Here is generic SQL syntax of INSERT INTO command to insert data into MySQL table:

```
INSERT INTO table_name ( field1, field2,...fieldN )
    VALUES( value1, value2,...valueN );
```

      - To insert string data types it is required to keep all the values into double or single quote, for example:- **"value"**.
    - **Example:**

```
<?php
        $con = mysql_connect("localhost","root","");
        if(!$con )
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_select_db("test",$con);
        $sql = "INSERT INTO student(name, city) VALUES ('rahul', 'unjha')";
        $a= mysql_query( $sql, $con );
        if(!$a)
        {
                die('Could not enter data: '.mysql_error());
        }
        else
        {
                echo "Entered data successfully\n";
        }
        mysql_close($con);
?>
```

  - **SELECT**
    - The SQL **SELECT** command is used to fetch data from MySQL database. You can use this command at mysql> prompt as well as in any script like PHP.

- **Syntax:**
  - Here is generic SQL syntax of SELECT command to fetch data from MySQL table:

    SELECT field1, field2,...fieldN table_name1, table_name2...
    [WHERE Clause]
    [OFFSET M][LIMIT N]
- You can use one or more tables separated by comma to include various condition using a WHERE clause. But WHERE clause is an optional part of SELECT command.
- You can fetch one or more fields in a single SELECT command.
- You can specify star (*) in place of fields. In this case SELECT will return all the fields
- You can specify any condition using WHERE clause.
- You can specify an offset using **OFFSET** from where SELECT will start returning records. By default offset is zero
- You can limit the number of returned using **LIMIT** attribute.
- **Example:-**

```
<?php
        $con = mysql_connect("localhost","root","");
        if(!$con )
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_select_db("test",$con);
        $sql = "select * from student";
        $a = mysql_query( $sql, $con );
        if(!$a)
        {
                die('Could not read data: '.mysql_error());
        }
        else
        {
                while($row = mysql_fetch_array($a))
                {
                        echo $row[0]."  ".$row[1]."  ".$row[2]."</br>";
                }
                echo "Fetched data successfully\n";

        }
        mysql_close($con);
?>
```

- **UPDATE**
  - There may be a requirement where existing data in a MySQL table need to be modified.
  - You can do so by using SQL **UPDATE** command. This will modify any field value of any MySQL table.
  - **Syntax:**
    - Here is generic SQL syntax of UPDATE command to modify data into MySQL table:

UPDATE table_name
SET field1=new-value1, field2=new-value2
[WHERE Clause]

- You can update one or more field all together.
- You can specify any condition using WHERE clause.
- You can update values in a single table at a time.
- The WHERE clause is very useful when you want to update selected rows in a table
- **Example:-**

```
<?php
        $con = mysql_connect("localhost","root","");
        if(!$con )
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_select_db("test",$con);
        $sql = "update student set name='ravi' where city='unjha' ";
        $a = mysql_query( $sql, $con );
        if(!$a)
        {
                die('Could not Update data: '.mysql_error());
        }
        else
        {
                echo "data updated successfully";

        }
        mysql_close($con);
?>
```

- **DELETE**
  - If you want to delete a record from any MySQL table then you can use SQL command **DELETE FROM**. You can use this command at mysql> prompt as well as in any script like PHP.
  - Syntax:
    - Here is generic SQL syntax of DELETE command to delete data from a MySQL table:

      DELETE FROM table_name [WHERE Clause]
  - If WHERE clause is not specified then all the records will be deleted from the given MySQL table.
  - You can specify any condition using WHERE clause.
  - You can delete records in a single table at a time.
  - The WHERE clause is very useful when you want to delete selected rows in a table.
  - **Example:-**

```
<?php
        $con = mysql_connect("localhost","root","");
        if(!$con )
        {
                die('Could not connect: ' . mysql_error());
        }
```

```
mysql_select_db("test",$con);
$sql = "delete from student where city='rahul'";
$a = mysql_query( $sql, $con );
if(!$a)
{
        die('Could not Delete data: '.mysql_error());
}
else
{
        echo "data deleted successfully";

}
mysql_close($con);
?>
```

- **DCL**
  - **Grant**
    - Grants a privilege to a user
    - It means that giving authority to other user by administrator
    - If you are administrator then only you have authority for grating the other authority to other user
    - Can grant privilege only if you have been granted that privilege
    - **Syntax:**
      - **GRANT < Object Privileges > ON <ObjectName> TO <UserName> [WITH GRANT OPTION];**
      - OBJECT PRIVILEGES
      - Each object privilege that is granted authorizes the grantee to perform some operation on the object. A user can grant all the privileges or grant only specific object privileges.
      - The list of object privileges is as follows:
        - ALTER : Allows the grantee to change the table definition with the ALTER TABLE command
        - DELETE : Allows the grantee to remove the records from the table with the DELETE command
        - INDEX : Allows the grantee to create an index on the table with the CREATE INDEX command
        - INSERT : Allows the grantee to add records to the table with the INSERT command
        - SELECT : Allows the grantee to query the table with the SELECT command
        - UPDATE : Allows the grantee to modify the records in the tables with the UPDATE command
      - **Example 1:** Give the user root permission to only view and modify records in the table student**.**

```
<?php
        $con = mysql_connect("localhost","root","");
        if(!$con )
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_select_db("test",$con);
        $sql = "GRANT SELECT, UPDATE ON student TO root";
        $a = mysql_query( $sql, $con );
```

```
        if(!$a)
        {
                die('Could not GRANT To user: '.mysql_error());
        }
        else
        {
                echo "GRANT  successfully";
        }
        mysql_close($con);
?>
```

- **Revoke**
  - **The REVOKE** statement is used to deny the grant given on an object.
  - Revokes a privilege from a user
  - It is use to taking off or remove of authority or say getting back authority from user
  - **Syntax:**
    - **REVOKE < Object Privileges > ON <Object Name> FROM <Username>;**
  - **Example 1:**
    - All privileges on the table student have been granted to root. Take back the SELECT, UPDATE privilege on the table.

```
<?php
        $con = mysql_connect("localhost","root","");
        if(!$con )
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_select_db("test",$con);
        $sql = "REVOKE SELECT, UPDATE ON student FROM root";
        $a = mysql_query( $sql, $con );
        if(!$a)
        {
                die('Could not GRANT To user: '.mysql_error());
        }
        else
        {
                echo "REVOKE  successfully";
        }
        mysql_close($con);
?>
```

❖ **Integration of PHP with MySQL**

➢ PHP MySQL Functions

Function	Description
mysql_affected_rows()	Returns the number of affected rows in the previous MySQL operation
mysql_change_user()	Deprecated. Changes the user of the current MySQL connection
mysql_client_encoding()	Returns the name of the character set for the current connection
mysql_close()	Closes a non-persistent MySQL connection
mysql_connect()	Opens a non-persistent MySQL connection
mysql_create_db()	Deprecated. Creates a new MySQL database. Use mysql_query() instead
mysql_data_seek()	Moves the record pointer
mysql_db_name()	Returns a database name from a call to mysql_list_dbs()
mysql_db_query()	Deprecated. Sends a MySQL query. Use mysql_select_db() and mysql_query() instead
mysql_drop_db()	Deprecated. Deletes a MySQL database. Use mysql_query() instead
mysql_errno()	Returns the error number of the last MySQL operation
mysql_error()	Returns the error description of the last MySQL operation
mysql_escape_string()	Deprecated. Escapes a string for use in a mysql_query. Use mysql_real_escape_string() instead
mysql_fetch_array()	Returns a row from a recordset as an associative array and/or a numeric array
mysql_fetch_assoc()	Returns a row from a recordset as an associative array
mysql_fetch_field()	Returns column info from a recordset as an object
mysql_fetch_lengths()	Returns the length of the contents of each field in a result row
mysql_fetch_object()	Returns a row from a recordset as an object
mysql_fetch_row()	Returns a row from a recordset as a numeric array
mysql_field_flags()	Returns the flags associated with a field in a recordset
mysql_field_len()	Returns the maximum length of a field in a recordset
mysql_field_name()	Returns the name of a field in a recordset
mysql_field_seek()	Moves the result pointer to a specified field
mysql_field_table()	Returns the name of the table the specified field is in
mysql_field_type()	Returns the type of a field in a recordset
mysql_free_result()	Free result memory
mysql_get_client_info()	Returns MySQL client info
mysql_get_host_info()	Returns MySQL host info
mysql_get_proto_info()	Returns MySQL protocol info
mysql_get_server_info()	Returns MySQL server info
mysql_info()	Returns information about the last query
mysql_insert_id()	Returns the AUTO_INCREMENT ID generated from the previous INSERT operation

mysql_list_dbs()	Lists available databases on a MySQL server
mysql_list_fields()	Deprecated. Lists MySQL table fields. Use mysql_query() instead
mysql_list_processes()	Lists MySQL processes
mysql_list_tables()	Deprecated. Lists tables in a MySQL database. Use mysql_query() instead
mysql_num_fields()	Returns the number of fields in a recordset
mysql_num_rows()	Returns the number of rows in a recordset
mysql_pconnect()	Opens a persistent MySQL connection
mysql_ping()	Pings a server connection or reconnects if there is no connection
mysql_query()	Executes a query on a MySQL database
mysql_real_escape_string()	Escapes a string for use in SQL statements
mysql_result()	Returns the value of a field in a recordset
mysql_select_db()	Sets the active MySQL database
mysql_stat()	Returns the current system status of the MySQL server
mysql_tablename()	Deprecated. Returns the table name of field. Use mysql_query() instead
mysql_thread_id()	Returns the current thread ID
mysql_unbuffered_query()	Executes a query on a MySQL database (without fetching / buffering the result)

- **mysql_connect()**

  - The mysql_connect() function opens a non-persistent MySQL connection.
  - This function returns the connection on success, or FALSE and an error on failure. You can hide the error output by adding an '@' in front of the function name.
  - Syntax**:-** **mysql_connect(server, username, password)**

Parameter	**Description**
server	Optional. Specifies the server to connect to (can also include a port number. e.g. "hostname:port" or a path to a local socket for the localhost). Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

  - Tips and Notes
    - **Tip:** The connection will be closed as soon as the script ends. To close the connection before, use mysql_close().
    - **Tip:** To establish a persistent MySQL connection, use mysql_pconnect() instead.

  - **Example**

```
<?php
        $con = mysql_connect ("localhost", "root", "");
        if ($con)
        {
```

```
        echo "Connection Sucessfully";
    }
    else
    {
        die('Could not connect: ' . mysql_error());
    }
    // some code
    mysql_close($con);
?>
```

- **mysql_select_db()**

  - Definition and Usage
    - The mysql_select_db() function sets the active MySQL database.
    - This function returns TRUE on success, or FALSE on failure.
  - Syntax:- mysql_select_db(database, connection)

Parameter	Description
database	Required. Specifies the database to select.
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

  - **Example**

```
<?php
    $con = mysql_connect("localhost", "root", "");
    if (!$con)
    {
        die('Could not connect: ' . mysql_error());
    }
    $a = mysql_select_db("test", $con);
    if ($a)
    {
        echo "Database Selected  successfully";
    }
    else
    {
        die ("Can\'t use test_db : " . mysql_error());
    }
    mysql_close($con);
?>
```

- **mysql_close()**
  - Definition and Usage
    - The mysql_close() function closes a non-persistent MySQL connection.
    - This function returns TRUE on success, or FALSE on failure.
    - **Syntax:- mysql_close(connection)**

Parameter	Description
connection	Optional. Specifies the MySQL connection to close. If not specified, the last connection opened by mysql_connect() is used.

- **Tips and Notes**
  - **Tip:** Using mysql_close() isn't required, as non-persistent connections are automatically closed at the end of the script.
- **Example :-**

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

- **mysql_query()**
  - Definition and Usage
    - The mysql_query() function executes a query on a MySQL database.
    - This function returns the query handle for SELECT queries, TRUE/FALSE for other queries, or FALSE on failure.
  - **Syntax:- mysql_query(query, connection)**

Parameter	Description
query	Required. Specifies the SQL query to send (should not end with a semicolon)
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

- **Example:-**

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

- **mysql_fetch_row()**
  - Definition and Usage
    - The mysql_fetch_row() function returns a row from a recordset as a numeric array.
    - This function gets a row from the mysql_query() function and returns an array on success, or FALSE on failure or when there are no more rows.
    - **Syntax:- mysql_fetch_row(data)**

Parameter	Description
data	Required. Specifies which data pointer to use. The data pointer is the result from the mysql_query() function

- **Example**

```
<?php
        $con = mysql_connect("localhost", "root", "");
        if (!$con)
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_select_db("test",$con);
        $sql = "SELECT * from student1";
        $a = mysql_query($sql,$con);
        print_r(mysql_fetch_row($a));
        mysql_close($con);
?>
```

- **mysql_fetch_array()**
  - Definition and Usage
    - The mysql_fetch_array() function returns a row from a recordset as an associative array and/or a numeric array.
    - This function gets a row from the mysql_query() function and returns an array on success, or FALSE on failure or when there are no more rows.
    - **Syntax:- mysql_fetch_array(data, array_type)**

Parameter	Description
data	Required. Specifies which data pointer to use. The data pointer is the result from the mysql_query() function
array_type	Optional. Specifies what kind of array to return. Possible values: • MYSQL_ASSOC - Associative array • MYSQL_NUM - Numeric array • MYSQL_BOTH - Default. Both associative and numeric array

- **Example**

```
<?php
        $con = mysql_connect("localhost", "root", "");
        if (!$con)
```

```
{
        die('Could not connect: ' . mysql_error());
}
mysql_select_db("test",$con);
$sql = "SELECT * from student1";
$a = mysql_query($sql,$con);
print_r(mysql_fetch_array($a));
mysql_close($con);
?>
```

- **PHP mysql_error()**
  - Definition and Usage
    - The mysql_error() function returns the error description of the last MySQL operation.
    - This function returns an empty string ("") if no error occurs.
  - **Syntax:- mysql_error(connection)**

Parameter	Description
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

  - **Example**
    - In this example we will try to log on to a MySQL server with the wrong username and password:

```
<?php
        $con = mysql_connect("localhost", "rahul", "rahul");
        if (!$con)
        {
                die(mysql_error());
        }
        mysql_select_db("test",$con);
?>
```

- **mysql_num_rows()**
  - Definition and Usage
    - The mysql_num_rows() function returns the number of rows in a recordset.
    - This function returns FALSE on failure.
  - **Syntax:- mysql_num_rows(data)**

Parameter	Description
data	Required. Specifies which data pointer to use. The data pointer is the result from the mysql_query() function

  - **Example:-**

```
<?php
        $con = mysql_connect("localhost", "root", "");
        if (!$con)
        {
                die('Could not connect: ' . mysql_error());
        }
        mysql_select_db("test",$con);
```

```
$sql = "SELECT * FROM student1";
$a = mysql_query($sql,$con);
echo mysql_num_rows($a);
mysql_close($con);
?>
```

- **mysql_info()**
  - Definition and Usage
    - The mysql_info() function returns information about the last query.
    - This function returns information about the statement on success, or FALSE on failure.
  - **Syntax:- mysql_info(connection)**

Parameter	Description
connection	Optional. Specifies the MySQL connection. If not specified, the last connection opened by mysql_connect() or mysql_pconnect() is used.

  - **Example**

```
<?php
    $con = mysql_connect("localhost", "peter", "abc123");
    if (!$con)
    {
        die('Could not connect: ' . mysql_error());
    }
    mysql_select_db("test",$con);
    $sql = "INSERT INTO user VALUES ('dolly','patel')";
    $result = mysql_query($sql,$con);
    $info = mysql_info($con);
    echo $info;
    mysql_close($con);
?>
```

❖ **Connection to the MySQL server**
- **Explain to the mysql_connect() function in above.**

❖ **Working with PHP and arrays of data**
- **Explain to the mysql_fetch_row() and mysql_fetch_array().**
- **Example:-2**

```
<html>
<body>
    <table border='3'>
        <tr style='background-color: #CCCAF0;'>
            <th>ID</th>
            <th>Name</th>
            <th>City</th>
        </tr>

        <?php
            $con = mysql_connect("localhost","root","");
            if(!$con )
            {
```

```
                    die('Could not connect: ' . mysql_error());
                }
                mysql_select_db("test",$con);
                $sql = "select * from student";
                $a = mysql_query( $sql, $con );
                while($rows=mysql_fetch_row($a))
                {
                    echo "<tr>";
                    echo "<td>" . $rows[0] . "</td>";
                    echo "<td>" . $rows[1] . "</td>";
                    echo "<td>" . $rows[2] . "</td>";
                    echo "</tr>";
                }
                mysql_close($con);
            ?>
        </table>
    </body>
    </html>
```

- **Referencing two tables**
  - First Create to the two table
  - Then insert two the data in the table
  - Where condition relation between tables based on common filed
  - We like to show examples and code before we explain anything in detail, so here is how you would combine two tables into one using MySQL.
    - Table name: **Client**

Column name	Datatype	Size
Name	Varchar	20
Id	int	10

    - Table name: **Salesman**

Column name	Datatype	Size
Id	int	10
City	Varchar	20

- **Client(Insert to the data)**

id	name
10	rahul
20	ravi
30	pratik

- **Salesman (Insert to the data)**

id	city
10	unjha
20	kherva
40	visnagar

- **Syntax:-**

  SELECT TableName1.FiledName 1, TableName1.Filedname 2, ……. TableName1.Filedname n, TableName2.FiledName 1, TableName2.Filedname 2, ……. TableName2.Filedname n
  FROM Tablename1,Tablename2
  WHERE Tablename1.CommonFiledName = Tablename2.CommonFiledName;

- **Query**

  select client.name, salesman.city from client, salesman where client.id = salesman.id

- **Example:-**

```
<html>
<body>
    <table border='3'>
        <tr style='background-color: green;'>
            <th>Name</th>
            <th>City</th>
        </tr>

        <?php
            $con = mysql_connect("localhost","root","");
            if(!$con )
            {
                die('Could not connect: ' . mysql_error());
            }
            mysql_select_db("test",$con);
            $sql = "select client.name, salesman.city from client, salesman
            where client.id=salesman.id";
            $a = mysql_query( $sql, $con );
            while($rows=mysql_fetch_array($a))
            {
                echo "<tr>";
                echo "<td>" . $rows[0] . "</td>";
                echo "<td>" . $rows[1] . "</td>";
                echo "</tr>";
            }
            mysql_close($con);
        ?>
    </table>
</body>
</html>
```

❖ Joining two tables

- **Types of join**
  - **INNER JOIN**
    - Default join
    - Two table joined using INNER JOIN.
    - Two table common filed data will be read in INNER JOIN
    - **Syntax:-**

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

- **Query:-**

```
select * from client INNER JOIN salesman ON client.id=salesman.id
```

- **Example:-**

```
<html>
<body>
    <table border='3'>
        <tr style='background-color: green;'>
            <th>ID</th>
            <th>Name</th>
            <th>ID</th>
            <th>City</th>
        </tr>

        <?php
            $con = mysql_connect("localhost","root","");
            if(!$con )
            {
                die('Could not connect: ' . mysql_error());
            }
            mysql_select_db("test",$con);
            $sql = "select * from client INNER JOIN salesman ON
            client.id=salesman.id";
            $a = mysql_query( $sql, $con );
            while($rows=mysql_fetch_array($a))
            {
                echo "<tr>";
                echo "<td>" . $rows[0] . "</td>";
                echo "<td>" . $rows[1] . "</td>";
                echo "<td>" . $rows[2] . "</td>";
                echo "<td>" . $rows[3] . "</td>";
                echo "</tr>";
            }
            mysql_close($con);
        ?>
    </table>
</body>
</html>
```

- **OUTER JOIN**
  - **TWO types OUTER JOIN**
    - **LEFT OUTER JOIN**
    - **RIGHT OUTER JOIN**

- **LEFT OUTER JOIN**
  - The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).
  - Syntax:-

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

  - **Query:-**

```
Select * from client LEFT OUTER JOIN salesman ON client.id=salesman.id
```

  - **Example:-**

```
<html>
<body>
    <table border='3'>
        <tr style='background-color: green;'>
            <th>ID</th>
            <th>Name</th>
            <th>ID</th>
            <th>City</th>
        </tr>

        <?php
            $con = mysql_connect("localhost","root","");
            if(!$con )
            {
                die('Could not connect: ' . mysql_error());
            }
            mysql_select_db("test",$con);
            sql = "select * from client LEFT OUTER JOIN salesman ON
            client.id=salesman.id";
            $a = mysql_query( $sql, $con );
            while($rows=mysql_fetch_array($a))
            {
                echo "<tr>";
                echo "<td>" . $rows[0] . "</td>";
                echo "<td>" . $rows[1] . "</td>";
                echo "<td>" . $rows[2] . "</td>";
                echo "<td>" . $rows[3] . "</td>";
                echo "</tr>";
            }
            mysql_close($con);
        ?>
    </table>
</body>
</html>
```

- **RIGHT OUTER JOIN**
  - The RIGHT JOIN keyword returns all rows from the RIGHT table (table_name2), even if there are no matches in the LEFT table (table_name1).
  - **Syntax:-**

```
SELECT column_name(s)
FROM table_name1
RIGHT OUTER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

  - **Query:-**

```
Select * from client RIGHT OUTER JOIN salesman ON client.id=salesman.id
```

  - **Example:-**

```
<html>
<body>
    <table border='3'>
        <tr style='background-color: CYAN;'>
            <th>ID</th>
            <th>Name</th>
            <th>ID</th>
            <th>City</th>
        </tr>

        <?php
            $con = mysql_connect("localhost","root","");
            if(!$con )
            {
                die('Could not connect: ' . mysql_error());
            }
            mysql_select_db("test",$con);
            sql = "select * from client RIGHT OUTER JOIN salesman ON
            client.id=salesman.id";
            $a = mysql_query( $sql, $con );
            while($rows=mysql_fetch_array($a))
            {
                echo "<tr>";
                echo "<td>" . $rows[0] . "</td>";
                echo "<td>" . $rows[1] . "</td>";
                echo "<td>" . $rows[2] . "</td>";
                echo "<td>" . $rows[3] . "</td>";
                echo "</tr>";
            }
            mysql_close($con);
        ?>
    </table>
</body>
</html>
```

- **CROSS JOIN**
  - CROSS join works as a Cartesian product of rows for both left and right table specified to the both side of the CROSS JOIn.
  - **Syntax:-**

```
SELECT column_name(s)
FROM table_name1
CROSS JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

  - **Query**

```
select * from client CROSS JOIN salesman ON client.id=salesman.id
```

  - **Example:-**

```
<html>
<body>
    <table border='3'>
        <tr style='background-color: CYAN;'>
            <th>ID</th>
            <th>Name</th>
            <th>ID</th>
            <th>City</th>
        </tr>

        <?php
            $con = mysql_connect("localhost","root","");
            if(!$con )
            {
                die('Could not connect: ' . mysql_error());
            }
            mysql_select_db("test",$con);
            $sql = "select * from client CROSS JOIN salesman ON
            client.id=salesman.id";
            $a = mysql_query( $sql, $con );
            while($rows=mysql_fetch_array($a))
            {
                echo "<tr>";
                echo "<td>" . $rows[0] . "</td>";
                echo "<td>" . $rows[1] . "</td>";
                echo "<td>" . $rows[2] . "</td>";
                echo "<td>" . $rows[3] . "</td>";
                echo "</tr>";
            }
            mysql_close($con);
        ?>
    </table>
</body>
</html>
```

WORDPRESS

WordPress is an open source Content Management System (CMS), which allows the users to build dynamic websites and blogs. Wordpress is the most popular blogging system on the web and allows updating, customizing and managing the website from its back-end CMS and components. What is Content Management System (CMS)? The Content Management System (CMS) is a software which stores all the data such as text, photos, music, documents, etc. and is made available on your website. It helps in editing, publishing and modifying the content of the website. WordPress was initially released on 27th May, 2003 by Matt Mullenweg and Mike Little. WordPress was announced as open source in October 2009.

Features

- User Management: It allows managing the user information such as changing the role of the users to (subscriber, contributor, author, editor or administrator), create or delete the user, change the password and user information. The main role of the user manager is Authentication.
- Media Management: It is the tool for managing the media files and folder, in which you can easily upload, organize and manage the media files on your website.
- Theme System: It allows modifying the site view and functionality. It includes images, stylesheet, template files and custom pages.
- Extend with Plugins: Several plugins are available which provides custom functions and features according to the users need.
- Search Engine Optimization: It provides several search engine optimization (SEO) tools which makes on-site SEO simple.
- Multilingual: It allows translating the entire content into the language preferred by the user. .
- Importers: It allows importing data in the form of posts. It imports custom files, comments, post pages and tags.

Advantages

- It is an open source platform and available for free.
- CSS files can be modified according to the design as per users need.
- There are many plugins and templates available for free. Users can customize the various plugins as per their need.
- It is very easy to edit the content as it uses WYSIWYG editor (What You See Is What You Get is a user interface that allows the user to directly manipulate the layout of document without having a layout command).
- Media files can be uploaded easily and quickly.
- It offers several SEO tools which makes on-site SEO simple.
- Customization is easy according to the user's needs.
- It allows creating different roles for users for website such as admin, author, editor and contributor.

Disadvantages

- Using several plugins can make the website heavy to load and run.
- PHP knowledge is required to make modifications or changes in the WordPress website.
- Sometimes software needs to be updated to keep the WordPress up-to-date with the current browsers and mobile devices.
- Updating WordPress version leads to loss of data, so it a backup copy of the website is required. Modifying and formatting the graphic images and tables is difficult.