

Course BCA-VI SEM

Academic Year:2021-22

Course Title: Web Programming

Course Code:602

Credits:4

Credit hour:40

Course Overview:

This subject helps Students to design and implement a basic website. Students should be able to implement different navigation strategies. Students should be able to develop simple back- end database to support a website. Students should be able to recognize and evaluate web site organizational structure and design elements. PHP is a server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites. It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server. PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.

Unit	Contents
Module 1: Introduction To PHP	Installing and configuring PHP, Building Blocks of PHP:PHP tags, variables, data types, operators, expressions. Control Structures: Conditional statements, loops, switch statement.
Module 2: Working With Functions and Arrays	Working with functions: What is function? Function declaration and definition. Calling functions, user defined functions, variable scope. Working with Arrays: creating, sorting and reordering arrays. PHP classes.
Module 3: String	Working with strings, dates and time: Formatting,

Manipulation and Forms	<p>investigation and manipulating strings with PHP, using date and time functions in php Working with forms: Creating a simple input form. File Handling: Saving data, storing and retrieving Bob's order, processing files, opening file, writing to a file, closing a file, reading from a file, uses other useful file functions</p>
Module 4: Working with cookies and sessions file handling	<p>Working with cookies: Introducing setting and deleting cookies with PHP.</p> <p>Working with session: starting a session ,working with session variables, passing session IDs in the query string ,destroying sessions and unsetting variables ,using sessions.</p>
Module 5: MYSQL	<p>Creating web database: Using MySql monitor, logging into MySql, creating data bases and users, setting users and privileges, column data types.</p> <p>Working with Mysql database: Inserting data into database, retrieving data with specific criteria, retrieving data from the multiple tables, retrieving data in particular order, grouping and aggregate data, using sub queries, updating records, deleting records from databases dropping table and database.</p>
Module 6: Accessing Mysql database from web with php	<p>Web database architecture, querying database from the web: checking and filtering input data, setting up connections, choosing database to use, querying database, retrieving the query result, disconnecting from the database.</p>

Course Objectives:

After undergoing this course, the student will be able to:

- Understand the paradigm for dealing with form-based data, both from the syntax of HTML forms, and how they are accessed inside a PHP-based script.
- Understand basic PHP syntax for variable use, and standard language constructs, such as conditionals and loops
- Be able to develop a form containing several fields and be able to process the data provided on the form by a user in a PHP-based script.
- Understand the syntax and functions available to deal with file processing for files on the server as well as processing web URLs.
- The student should be able to establish the connectivity (back- end database to support a website).

Learning Outcomes

- **L01:** Be able to develop a form containing several fields and be able to process the data provided on the form by a user in a PHP-based script.
- **L02:** Understand the paradigm for dealing with form-based data, both from the syntax of HTML forms, and how they are accessed inside a PHP-based script.
- **L03:** Be able to design online applications based and understand the concepts of client server architecture.
- **L04:** Understand the paradigm for dealing with form-based data, both from the syntax of HTML forms, and how they are accessed inside a PHP-based script.

Evaluation Criteria:

Component	Description	Weightage
First Internal Examination	First internal question paper will be based on first 3 unit of syllabus.	10 marks
Second Internal Examination	Second internal question paper will be based on last 4 unit of syllabus.	10 marks
Quiz	Test based on MCQ, one-word s.	3.33 marks
Quiz	Test based on MCQ, one-word s.	3.3marks
Quiz	Test based on MCQ, one-word s.	3.33marks

*** ALL CES ACTIVITIES ARE COMPULSORY.**

7. Recommended/ Reference Text Books and Resources:

Reference books	<ol style="list-style-type: none">1. Beginning PHP5 and MySQL: From Novice to Professional, W. Jason Gilmore, 20042. PHP, MySQL, and JavaScript: A Step-By-Step Guide to Creating Dynamic Websites by Robin Nixon O'Reilly Media; 1 edition
Text book	Teach yourself PHP,MySql and Apache by Julie.C.Meloni
Internet Resource:	https://www.w3schools.com/php/ https://www.tutorialspoint.com/php/ https://www.phptpoint.com/php-tutorial/

Session Plan:

Session	Topic	Learning Resources	Learning Outcome
1	Syllabus Discussion: Introduction and overview of Unit -1 , Revision of Basic Web Technology	Handouts	To brief about the whole syllabus and explain basic knowledge of HTML,DHTML. LO1,LO2
2	Introduction : HTML, Java Script, DHTML	Handouts	To aware about the HTML, DHTML and JavaScript in brief LO3,LO4.
3	Web Application: Introduction to Web Applications, HTML, Client Side Scripting	Handouts	To explain concept of web application. LO1,LO2
5	Web Server: Introduction to Web Server , Local Server and Remote Servers	Handouts	To explain concept of Web Server : remote server and local server LO3,LO4.
6	Web Server: Installing Web Servers WAMP, IIS(Internet information Server)	Handouts	To explain and demonstrate installation of web servers IIS. LO1,LO2
7	Website Development: Static Website vs Dynamic Website Development.	Handouts	To explain the difference between Static website and dynamic

			Website. LO3,LO4.
8	Overview of Unit-II: Introduction of PHP & concept of programming in PHP	Handouts	Overview on concept of PHP ,data types and PHP language basics. LO1,LO2
9	Introduction to PHP: what does PHP do, brief History of PHP, Installing PHP	Handouts	To explain the introduction history and installation of PHP. LO1,LO2
10	PHP Language Basics: Operators, Data Types	Handouts	To explain the concept of PHP language: data types LO3,LO4.
11	PHP Language Basics: Variables, Constants	Handouts	To explain and inculcate the PHP variable and constants in PHP scripts. LO1,LO2
12	PHP Language Basics: Expressions, Printing Data on PHP Page	Handouts	To explain and learn how to print data and apply expressions in PHP Scripts. LO3,LO4.
10	PHP Language Basics: Flow Control Statements-if, switch case	Handouts	To explain Conditional statements in PHP. LO1,LO2
11	Arrays in PHP : Identifying Elements of an Array initialization of an array, iterating through an array	Handouts	To explain introduction of array and different types of array in PHP. LO1,LO2

12	Functions in PHP: Defining a Function , Calling A function, Variable Scope(Global variable& Static Variables)	Handouts	To explain the concept of function and different types of variable scopes. LO1,LO2
13	PHP classes Classes, objects,	Handouts	To understand the object oriented features in PHP. LO3,LO4.
14	Constructors, Inheritance	Handouts	To understand the object oriented features in PHP. LO3,LO4.
15	Overview of Unit –III : String Manipulation Working with Strings Built –in functions and user defined like strlen(),strev(),count, replace etc.	Handouts	Overview of Strings manipulation. LO1,LO2
16	Printing Strings, Cleaning Strings, Comparing String	Handouts	Strings manipulation. LO1,LO2
17	Date and time functions in PHP	Handouts	Functions like date and time. LO3,LO4.
18	Forms in PHP: get &post Methods in form, get data form text boxes, password boxes, and hidden fields.	Handouts	To explain the concept of forms in php, get and post methods. LO1,LO2
19	Forms in PHP: get data from radio button, check box, from list box, from drop-down list.	Handouts	To explain and learn how can data get from radio button ,list box and etc. LO1,LO2

20	Working with Files: File Handling(file exist, creating a file ,reading a file, copying ,writing deleting a file)	Handouts	To explain concept of files and operation of files in PHP . LO3,LO4.
21	Working with Files: Getting information on Files updating files	Handouts	To explain and learn how to upload a file and image in PHP. LO3,LO4.
22	, locking files ,reading an entire file	Handouts	To explain and learn how to upload a file and image in PHP. LO3,LO4.
23	Maintaining User State: using cookies in php (setting, accessing and destroying cookies)	Handouts	To explain and understand the concept of Cookies and how can Cookies are created and end. LO3,LO4.
24	Maintaining Working with sessions : Sessions (starting, ending and session security), application states.	Handouts	To explain and understand the concept of Session and how can Session are created and end. LO3,LO4.
25	Setting and unsetting variables using sessions	Handouts	Working with sessions. LO1,LO2
26	MYSQL & PHP Database Connectivity	Handouts	Overview of MYSQL database and PHP MYadmin and PHP database connectivity LO3,LO4.

27	Introduction to MySQL: MySQL Basics, MySQL Commands, Data Types	Handouts	To explain introduction of MySQL commands and data types. LO1,LO2
28	Introduction to MySQL: Data Base Design	Handouts	to explain and learn to create database and tables in MYSQL and through PHPMysqladmin. LO3,LO4.
29	Accessing MySQL Using PHP: Querying a MySQL Databases with PHP, process, creating	Handouts	to explain and learn to create database and tables in MYSQL and through PHPMysqladmin. LO3,LO4
30	Connecting to a Database: connect to MySQL database, execute SELECT Command statements, Execute INSERT,UPDATE and DELETE statements,	Handouts	To explain and understand how to connect MySQL database and execute different commands. LO1,LO2
31	Retrieving data from multiple tables, ordering data, grouping and aggregating	Handouts	To explain and understand how to connect MySQL database and execute different commands. LO3,LO4.
32	data using sub queries	Handouts	To explain and understand how to connect MySQL database and execute

			different commands. LO3,LO4.
33	Accessing MySQL using PHP: Query a MySQL Database with PHP , the process ,creating login file, connecting to MySQL	Handouts	To understand how to access data using PHP script from MySQL database with the help of example. LO1,LO2
34	Accessing MYSQL using PHP: Delete a record ,Display the form, Querying the Database	Handouts	To understand how to delete a record a data from database. LO3,LO4.
35	Connecting to a Database: Open a Connection to the MySQL Server, Close a Connection	Handouts	To explain and understand open and close a connection using PHP script. LO1,LO2
36	Connecting to a Database: get data from result Set, product viewer application	Handouts	To understand how to get data from forms in different forms(example table form). LO3,LO4.
37	get data from result Set,	Handouts	
38	product viewer application	Working	

39	product viewer application	Working	
40	product viewer application	Working	

Study Notes

Unit-1

Introduction to PHP

Many of these development environments let you run the PHP code and see the output discussed in this chapter. I'll also show you how to embed the PHP in an HTML file so that you can see what the output looks like in a web page (the way your users will ultimately see it). But that step, as thrilling as it may be at first, isn't really important at this stage.

In production, your web pages will be a combination of PHP, HTML, and JavaScript, and some MySQL statements laid out using CSS. Furthermore, each page can lead to other pages to provide users with ways to click through links and fill out forms. We can avoid all that complexity while learning each language, though. Focus for now on just writing PHP code and making sure that you get the output you expect—or at least, that you understand the output you actually get!

Incorporating PHP Within HTML

By default, PHP documents end with the extension *.php*. When a web server encounters this extension in a requested file, it automatically passes it to the PHP processor. Of course, web servers are highly configurable, and some web developers choose to force files ending with *.htm* or *.html* to also get parsed by the PHP processor, usually because they want to hide the fact that they are using PHP.

Your PHP program is responsible for passing back a clean file suitable for display in a web browser. At its very simplest, a PHP document will output only HTML. To prove this, you can take any normal HTML document, such as an *index.html* file, and save it as *index.php*; it will display identically to the original.

Calling the PHP Parser

To trigger the PHP commands, you need to learn a new tag. The first part is:

```
<?php
```

The first thing you may notice is that the tag has not been closed. This is because entire sections of PHP can be placed inside this tag, and they finish only when the closing part, which looks like this, is encountered:

```
?>
```

A small PHP “Hello World” program might look like Example 3-1.

Example 3-1. Invoking PHP

```
<?php
```

```
echo "Hello world";
```

```
?>
```

The way you use this tag is quite flexible. Some programmers open the tag at the start of a document and close it right at the end, outputting any HTML directly from PHP commands.

Others, however, choose to insert only the smallest possible fragments of PHP within these tags wherever dynamic scripting is required, leaving the rest of the document in standard HTML.

The latter type of programmer generally argues that their style of coding results in faster code, while the former say that the speed increase is so minimal that it doesn’t justify the additional complexity of dropping in and out of PHP many times in a single document.

As you learn more, you will surely discover your preferred style of PHP development, but for the sake of making the examples in this book easier to follow, I have adopted the approach of keeping the number of transfers between PHP and HTML to a mini-mum—generally only once or twice in a document.

By the way, a slight variation to the PHP syntax exists. If you browse the Internet for PHP examples, you may also encounter code where the opening and closing syntax used is like this:

```
<?

echo "Hello world";

?>
```

Although it's not as obvious that the PHP parser is being called, this is a valid, alternative syntax that also usually works. However, it should be discouraged, as it is incompatible with XML and its use is now deprecated (meaning that it is no longer recommended and that support could be removed in future versions). If you have only PHP code in a file, you may omit the closing `?>`. This can be a good practice, as it will ensure you have no excess whitespace leaking from your PHP files (especially important when writing object-oriented code).

named_examples, in which you'll find all the examples I suggest saving using a specific filename (such as the upcoming Example 3-4, which should be saved as *test1.php*).

The Structure of PHP

We're going to cover quite a lot of ground in this section. It's not too difficult, but I recommend that you work your way through it carefully, as it sets the foundation for everything else in this book. As always, there are

some useful questions at the end of the chapter that you can use to test how much you've learned.

Using Comments

There are two ways in which you can add comments to your PHP code. The first turns a single line into a comment by preceding it with a pair of forward slashes, like this:// This is a comment

This version of the comment feature is a great way to temporarily remove a line of code from a program that is giving you errors. For example, you could use such a comment to hide a debugging line of code until you need it, like this:

```
// echo "X equals $x";
```

You can also use this type of comment directly after a line of code to describe its action like this:

```
$x += 10; // Increment $x by 10
```

When you need multiple-line comments, there's a second type of comment, which looks like Example 3-2.

Example 3-2. A multiline comment

```
<?php
```

```
/* This is a section of multiline comments that will not be interpreted */
```

```
?>
```

You can use the `/*` and `*/` pairs of characters to open and close comments almost anywhere you like inside your code. Most, if not all, programmers use this construct to temporarily comment out entire sections of code that do not work or that, for one reason or another, they do not wish to be interpreted.

A common error is to use `/*` and `*/` to comment out a large section of code that already contains a commented-out section that uses those characters. You can't nest comments this way; the PHP interpreter won't know where a comment ends and will display an error message. However, if you use a program editor or IDE with syntax highlighting, this type of error is easier to spot.

Basic Syntax

PHP is quite a simple language with roots in C and Perl, yet it looks more like Java. It is also very flexible, but there are a few rules that you need to learn about its syntax and structure.

Semicolons

You may have noticed in the previous examples that the PHP commands ended with a semicolon, like this:

```
$x += 10;
```

Probably the most common cause of errors you will encounter with PHP is forgetting this semicolon, which causes PHP to treat multiple statements like one statement, which it is unable to understand. This leads to a “Parse error” message.

The \$ symbol

The \$ symbol has come to be used in many different ways by different programming languages. For example, if you have ever written in the BASIC language, you will have used the \$ to terminate variable names to denote them as strings.

In PHP, however, you must place a \$ in front of all variables. This is required to make the PHP parser faster, as it instantly knows whenever it comes across a variable. Whether your variables are numbers, strings, or arrays, they should all look something like those in Example 3-3.

And really, that’s pretty much all the syntax that you have to remember. Unlike languages such as Python, which is very strict about how you indent and lay out our code, PHP leaves you completely free to use (or not use) all the indenting and spacing you like. In fact, sensible use of what is called *whitespace* is generally encouraged (along with comprehensive commenting) to help you understand your code when you come back to it. It also helps other programmers when they have to maintain your code.

Understanding Variables

There's a simple metaphor that will help you understand what PHP variables are all about. Just think of them as little (or big) matchboxes! That's right, matchboxes that you've painted over and written names on.

String variables

Imagine you have a matchbox on which you have written the word *username*. You then write *Fred Smith* on a piece of paper and place it into the box (see Figure 3-2). Well, that's the same process as assigning a string value to a variable, like this:

```
$username = "Fred Smith";
```

The quotation marks indicate that “Fred Smith” is a *string* of characters. You must enclose each string in either quotation marks or apostrophes (single quotes), although there is a subtle difference between the two types of quote, which is explained later. When you want to see what's in the box, you open it, take out the piece of paper, and read it. In PHP, doing so looks like this:

```
echo $username;
```

Or you can assign it to another variable (i.e., photocopy the paper and place the copy in another matchbox), like this:

```
$current_user = $username;
```

Numeric variables

Variables don't contain just strings—they can contain numbers, too. Using the match-box analogy, to store the number 17 in the variable `$count`, the

equivalent would be placing, say, 17 beads in a matchbox on which you have written the word *count*:

```
$count = 17;
```

You could also use a floating-point number (containing a decimal point); the syntax is the same:

```
$count = 17.5;
```

To examine the contents of the matchbox, you would simply open it and count the beads. In PHP, you would assign the value of `$count` to another variable or perhaps just echo it to the web browser.

Arrays

So what are arrays? Well, you can think of them as several matchboxes ~~glued~~ together. For example, let's say we want to store the player names for a five-person soccer team in an array called `$team`. To do this, we could glue five matchboxes side by side and write down the names of all the players on separate pieces of paper, placing one in each matchbox.

Across the top of the matchbox assembly, we would write the word *team* (see

Figure 3-3). The equivalent of this in PHP would be:

```
$team = array('Bill', 'Joe', 'Mike', 'Chris', 'Jim');
```

This syntax is more complicated than the instructions I've explained so far. The array-building code consists of the following construct:

```
array();
```

with five strings inside the parentheses. Each string is enclosed in single quotes.

If we then wanted to know who player 4 is, we could use this command:

```
echo $team[3]; // Displays the name Chris
```

The reason the previous statement has the number 3 and not a 4 is because the first element of a PHP array is actually the zeroth element, so the player numbers will there-fore be 0 through 4.

Two-dimensional arrays

There's a lot more you can do with arrays. For example, instead of being single-di-mensional lines of matchboxes, they can be two-dimensional matrixes or can even have three or more dimensions.

As an example of a two-dimensional array, let's say we want to keep track of a game of tic-tac-toe, which requires a data structure of nine cells arranged in a 3×3 square. Torepresent this with matchboxes, imagine nine of them glued to each other in a matrix of three rows by three columns

You can now place a piece of paper with either an "x" or an "o" in the correct matchbox for each move played. To do this in PHP code, you have to set up an array containing three more arrays, as in Example 3-5, in which the array is set up with a game already in progress.

Example 3-5. Defining a two-dimensional array

```
<?php
```

```
$oxo = array(array('x', 'o'),  
              array('o', 'o', 'x'),  
              array('x', 'o', " "));
```

?>

Once again, we've moved up a step in complexity, but it's easy to understand if you grasp the basic array syntax. There are three `array()` constructs nested inside the outer `array()` construct.

To then return the third element in the second row of this array, you would use the following PHP command, which will display an "x":

```
echo $oxo[1][2];
```

Remember that array indexes (pointers at elements within an array) start from zero, not one, so the [1] in the previous command refers to the second of the three arrays, and the [2] references the third position within that array. It will return the contents of the matchbox three along and two down from the top left.

As mentioned, arrays with even more dimensions are supported by simply creating more arrays within arrays. However, we will not be covering arrays of more than two dimensions in this book.

Don't worry if you're still having difficulty getting to grips with using arrays, as the subject is explained in detail in Chapter 6.

Variable naming rules

When creating PHP variables, you must follow these four rules:

- Variable names must start with a letter of the alphabet or the _ (underscore) character.

- Variable names can contain only the characters a-z, A-Z, 0-9, and _ (underscore).
- Variable names may not contain spaces. If a variable must comprise more than one word, the words should be separated with the _ (underscore) character (e.g., \$user_name).
- Variable names are case-sensitive. The variable \$High_Score is not the same as the variable \$high_score.

Operators

Operators are the mathematical, string, comparison, and logical commands such as plus, minus, times, and divide, which in PHP looks a lot like plain arithmetic; for in-stance, the following statement outputs 8:

```
echo 6 + 2;
```

Before moving on to learn what PHP can do for you, take a moment to learn about the various operators it provides.

Arithmetic operators

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

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

Assignment operators

These operators are used to assign values to variables. They start with the very simple

= and move on to +=, -=, and so on (see Table 3-2). The operator += adds the value on the right side to the variable on the left, instead of totally replacing the value on the left. Thus, if \$count starts with the value 5, the statement:

```
$count += 1;
```

sets \$count to 6, just like the more familiar assignment statement:

```
$count = $count + 1;
```

Table 3-2. Assignment operators

Operator	Example	Equivalent to
=	\$j = 15	\$j = 15
+=	\$j += 5	\$j = \$j + 5
-=	\$j -= 3	\$j = \$j - 3
*=	\$j *= 8	\$j = \$j * 8
/=	\$j /= 16	\$j = \$j / 16
.=	\$j .= \$k	\$j = \$j . \$k
%=	\$j %= 4	\$j = \$j % 4

Strings have their own operator, the period (.), detailed in the section “String concat-enation” on page 50 a little later in this chapter.

Comparison operators

Comparison operators are generally used inside a construct such as an if statement in which you need to compare two items. For example, you may wish to know whether a variable you have been incrementing has reached a specific value, or whether another variable is less than a set value, and so on (see Table 3-3).

Table 3-3. Comparison operators

Operator	Description	Example
==	Is equal to	\$j == 4
!=	Is not equal to	\$j != 21
>	Is greater than	\$j > 3
<	Is less than	\$j < 100
>=	Is greater than or equal to	\$j >= 15
<=	Is less than or equal to	\$j <= 8

Note the difference between = and ==. The first is an assignment operator, and the second is a comparison operator. Even more advanced programmers can sometimes transpose the two when coding hurriedly, so be careful.

Logical operators

If you haven’t used them before, logical operators may at first seem a little daunting. But just think of them the way you would use logic in English. For example, you might say to yourself, “If the time is later than 12 PM and earlier than 2 PM, then have lunch.” In PHP, the code for this might look something like the following (using military timing):

```
if ($hour > 12 && $hour < 14) dolunch();
```

Here we have moved the set of instructions for actually going to lunch into a function that we will have to create later called `dolunch`. The *then* of the statement is left out, because it is implied and therefore unnecessary.

As the previous example shows, you generally use a logical operator to combine the results of two of the comparison operators shown in the previous section. A logical operator can also be input to another logical operator (“If the time is later than 12 PM and earlier than 2 PM, or if the smell of a roast is permeating the hallway and there are plates on the table...”). As a rule, if something has a TRUE or FALSE value, it can be input to a logical operator. A logical operator takes two true-or-false inputs and produces a true-or-false result.

Table 3-4. Logical operators

Operator	Description	Example
<code>&&</code>	And	<code>\$j == 3 && \$k == 2</code>
<code>and</code>	Low-precedence and	<code>\$j == 3 and \$k == 2</code>
<code> </code>	Or	<code>\$j < 5 \$j > 10</code>
<code>or</code>	Low-precedence or	<code>\$j < 5 or \$j > 10</code>
Operator	Description	Example
<code>!</code>	Not	<code>! (\$j == \$k)</code>
<code>xor</code>	Exclusive or	<code>\$j xor \$k</code>

Note that `&&` is usually interchangeable with `and`; the same is true for `||` and `or`. But `and` and `or` have a lower precedence, so in some cases, you may need extra parentheses to force the required precedence. On the other hand, there are times when *only* `and` or `or` is acceptable, as in the following statement, which uses an `or` operator (to be explained in Chapter 10):

```
mysql_select_db($database) or die("Unable to select database");
```

The most unusual of these operators is `xor`, which stands for *exclusive or* and returns

a TRUE value if either value is TRUE, but a FALSE value if both inputs are TRUE or both inputs are FALSE. To understand this, imagine that you want to concoct your own cleaner for household items. Ammonia makes a good cleaner, and so does bleach, so you want your cleaner to contain one of these. But the cleaner must not contain both, because the combination is hazardous. In PHP, you could represent this as:

```
$ingredient = $ammonia xor $bleach;
```

In the example snippet, if either `$ammonia` or `$bleach` is TRUE, `$ingredient` will also be set to TRUE. But if both are TRUE or both are FALSE, `$ingredient` will be set to FALSE.

Variable Assignment

The syntax to assign a value to a variable is always *variable = value*. Or, to reassign the value to another variable, it is *other_variable = variable*.

There are also a couple of other assignment operators that you will find useful. For example, we've already seen:

```
$x += 10;
```

which tells the PHP parser to add the value on the right (in this instance, the value 10)

to the variable `$x`. Likewise, we could subtract as follows:

```
$y -= 10;
```

Variable incrementing and decrementing

Adding or subtracting 1 is such a common operation that PHP provides special operators for these tasks. You can use one of the following in place of the += and -= operators:

```
++$x;
```

```
—$y;
```

Variable Typing

PHP is a very loosely typed language. This means that variables do not have to be declared before they are used, and that PHP always converts variables to the type required by their context when they are accessed.

For example, you can create a multiple-digit number and extract the *n*th digit from it, simply by assuming it to be a string. In the following snippet of code (Example 3-10), the numbers 12345 and 67890 are multiplied together, returning a result of 838102050, which is then placed in the variable \$number.

Example 3-10. Automatic conversion from a number to a string

```
<?php
```

```
$number = 12345 * 67890;
```

```
echo substr($number, 3, 1);
```

```
?>
```

At the point of the assignment, \$number is a numeric variable. But on the second line, a call is placed to the PHP function substr, which asks for one character to be returned from \$number, starting at the fourth position (remembering that PHP offsets start from zero). To do this, PHP turns \$number into a nine-character string, so that substr can access it and return the character, which in this case is 1.

The same goes for turning a string into a number, and so on. In Example 3-11, the variable \$pi is set to a string value, which is then automatically turned into a floating-point number in the third line by the equation for calculating a circle's area, which outputs the value 78.5398175.

Example 3-11. Automatically converting a string to a number

```
<?php

$pi = "3.1415927";

$radius = 5;

echo $pi * ($radius * $radius);

?>
```

In practice, what this all means is that you don't have to worry too much about your variable types. Just assign them values that make sense to you, and PHP will convert them if necessary. Then, when you want to retrieve values, just ask for them—for ex-ample, with an echo statement.

Constants

Constants are similar to variables, holding information to be accessed later, except that they are what they sound like—constant. In other words,

once you have defined one, its value is set for the remainder of the program and cannot be altered.

One example of a use for a constant might be to hold the location of your server root (the folder containing the main files of your website). You would define such a constant like this:

```
define("ROOT_LOCATION", "/usr/local/www/");
```

Then, to read the contents of the variable, you just refer to it like a regular variable (but it isn't preceded by a dollar sign):

```
$directory = ROOT_LOCATION;
```

Now, whenever you need to run your PHP code on a different server with a different folder configuration, you have only a single line of code to change.

Variable Scope

If you have a very long program, it's quite possible that you could start to run out of good variable names, but with PHP you can decide the *scope* of a variable. In other words, you can, for example, tell it that you want the variable \$temp to be used only inside a particular function and to forget it was ever used when the function returns.

In fact, this is the default scope for PHP variables.

Alternatively, you could inform PHP that a variable is global in scope and thus can be accessed by every other part of your program.

Local variables

Local variables are variables that are created within, and can be accessed only by, a function. They are generally temporary variables that are used to store partially pro-cessed results prior to the function's return.

One set of local variables is the list of arguments to a function. In the previous section, we defined a function that accepted a parameter named `$timestamp`. This is meaningful only in the body of the function; you can't get or set its value outside the function.

For another example of a local variable, take another look at the `longdate` function, which is modified slightly in Example 3-13.

Global variables

There are cases when you need a variable to have *global* scope, because you want all your code to be able to access it. Also, some data may be large and complex, and you don't want to keep passing it as arguments to functions. To declare a variable as having global scope, use the keyword `global`. Let's assume that you have a way of logging your users into your website and you want all your code to know whether it is interacting with a logged-in user or a guest. One way to do this is to create a global variable such as `$is_logged_in`:

```
global $is_logged_in;
```

Now your login function simply has to set that variable to 1 upon success of a login attempt, or 0 upon its failure. Because the scope of the variable is global, every line of code in your program can access it.

Static variables

In the section “Local variables” on page 58, I mentioned that the value of the variable is wiped out when the function ends. If a function runs many times, it starts with a fresh copy of the variable each time and the previous setting has no effect.

Here's an interesting case. What if you have a local variable inside a function that you don't want any other parts of your code to have access to, but that you would also like to keep its value for the next time the

function is called? Why? Perhaps because you want a counter to track how many times a function is called. The solution is to declare a *static variable*, as shown in Example 3-17.

Example 3-17. A function using a static variable

```
<?php

function test()

{

    static $count = 0;

    echo $count;

    $count++;

}
```

?> Here, the very first line of function test creates a static variable called \$count and initializes it to a value of 0. The next line outputs the variable's value; the final one increments it.

The next time the function is called, because \$count has already been declared, the first line of the function is skipped. Then the previously incremented value of \$count is displayed before the variable is again incremented.

If you plan to use static variables, you should note that you cannot assign the result of an expression in their definitions. They can be initialized only with predetermined values (see Example 3-18).

Example 3-18. Allowed and disallowed static variable declarations

```
<?php
```

```
static $int = 0;      // Allowed
```

```
static $int = 1+2;    // Disallowed (will produce a Parse error)
```

```
static $int = sqrt(144); // Disallowed
```

```
?>
```

Superglobal variables

Starting with PHP 4.1.0, several predefined variables are available. These are known as *superglobal variables*, which means that they are provided by the PHP environment but are global within the program, accessible absolutely everywhere.

These superglobals contain lots of useful information about the currently running program and its environment (see Table 3-6). They are structured as associative arrays, a topic discussed in Chapter 6.

Table 3-6. PHP's superglobal variables

Superglobal

name Contents

\$GLOBALS All variables that are currently defined in the global scope of the script. The variable names are the keys

of the array.

\$_SERVER Information such as headers, paths, and script locations. The web server creates the entries in this array,

and there is no guarantee that every web server will provide any or all of these.

<code>\$_GET</code>	Variables passed to the current script via the HTTP GET method.
<code>\$_POST</code>	Variables passed to the current script via the HTTP POST method.
<code>\$_FILES</code>	Items uploaded to the current script via the HTTP POST method.
<code>\$_COOKIE</code>	Variables passed to the current script via HTTP cookies.
<code>\$_SESSION</code>	Session variables available to the current script.
<code>\$_REQUEST</code>	Contents of information passed from the browser; by default, <code>\$_GET</code> , <code>\$_POST</code> , and <code>\$_COOKIE</code> .
<code>\$_ENV</code>	Variables passed to the current script via the environment method.

Expressions and Control Flow in PHP

The previous chapter introduced several topics in passing that this chapter covers more fully, such as making choices (branching) and creating complex expressions. In the previous chapter, I wanted to focus on the most basic syntax and operations in PHP, but I couldn't avoid touching on some more advanced topics. Now I can fill in the background that you need to use these powerful PHP features properly.

In this chapter, you will get a thorough grounding in how PHP programming works in practice and in how to control the flow of the program.

Expressions

Let's start with the most fundamental part of any programming language: *expressions*.

An expression is a combination of values, variables, operators, and functions that results in a value. Anyone who has taken an algebra class should recognize this sort of expression:

$$y = 3(\text{abs}(2x) + 4)$$

which in PHP would be written as:

$$\text{\$y} = 3 * (\text{abs}(2 * \text{\$x}) + 4);$$

The value returned (y or \$y in this case) can be a number, a string, or a *Boolean value* (named after George Boole, a nineteenth-century English

mathematician and philosopher). By now, you should be familiar with the first two value types, but I'll explain the third.

A basic Boolean value can be either TRUE or FALSE. For example, the expression `20 > 9` (20 is greater than 9) is TRUE, and the expression `5 == 6` (5 is equal to 6) is FALSE. (Boolean operations can be combined using operators such as AND, OR, and XOR, which are covered later in this chapter.)

Note that I am using uppercase letters for the names TRUE and FALSE. This is because they are predefined constants in PHP. You can also use the lowercase versions, if you prefer, as they are also predefined. In fact, the lowercase versions are more stable, because PHP does not allow you to redefine them; the uppercase ones may be redefined, which is something you should bear in mind if you import third-party code.

Literals and Variables

The simplest form of an expression is a *literal*, which simply means something that evaluates to itself, such as the number 73 or the string "Hello". An expression could also simply be a variable, which evaluates to the value that has been assigned to it. These are both types of expressions because they return a value. Example 4-3 shows five different literals, all of which return values, albeit of different types.

Example 4-3. Five types of literals

```
<?php
```

```
$myname = "Brian";
```

```
$myage = 37;
```

```
echo "a: " . 73 . "<br />"; // Numeric literal
```

```
echo "b: " . "Hello" . "<br />"; // String literal
```

```
echo "c: " . FALSE . "<br />"; // Constant literal
```

```
echo "d: " . $myname . "<br />"; // Variable string literal
```

```
echo "e: " . $myage . "<br />"; // Variable numeric literal
```

```
?>
```

As you'd expect, you'll see a return value from all of these with the exception of `c:`, which evaluates to `FALSE`, returning nothing in the following output:

```
a: 73
```

```
b: Hello
```

```
c:
```

```
d: Brian
```

```
e: 37
```

In conjunction with operators, it's possible to create more complex expressions that evaluate to useful results.

When you combine assignment or control-flow constructs with expressions, the result is a *statement*. Example 4-4 shows one of each. The first assigns the result of the expression `366 - $day_number` to the variable `$days_to_new_year`, and the second outputs a friendly message only if the expression `$days_to_new_year < 30` evaluates to `TRUE`.

Example 4-4. An expression and a statement

```
<?php

$days_to_new_year = 366 - $day_number;
// Expression if ($days_to_new_year < 30)
{

    echo "Not long now till new year"; // Statement

}

?>
```

Operators

PHP offers a lot of powerful operators, ranging from arithmetic, string, and logical operators to operators for assignment, comparison, and more (see Table 4-1).

Different types of operators take a different number of operands:

- *Unary* operators, such as incrementing (\$a++) or negation (-\$a), take a single operand.
- *Binary* operators, which represent the bulk of PHP operators (including addition, subtraction, multiplication, and division), take two operands.
- There is one *ternary* operator, which takes the form `x ? y : z`. It's a terse, single-line if statement that chooses between two expressions,

depending on the result of a third one. This conditional operator takes three operands.

Operator Precedence

If all operators had the same precedence, they would be processed in the order in which they are encountered. In fact, many operators do have the same precedence—Example 4-5 illustrates one such case.

Example 4-5. Three equivalent expressions

$$1 + 2 + 3 - 4 + 5$$

$$2 - 4 + 5 + 3 + 1$$

$$5 + 2 - 4 + 1 + 3$$

Here you will see that although the numbers (and their preceding operators) have been moved around, the result of each expression is the value 7, because the plus and minus operators have the same precedence. We can try the same thing with multiplication and division (see Example 4-6).

Example 4-6. Three expressions that are also equivalent

$$1 * 2 * 3 / 4 * 5$$

$$2 / 4 * 5 * 3 * 1$$

$$5 * 2 / 4 * 1 * 3$$

Here the resulting value is always 7.5. But things change when we mix operators with *different* precedences in an expression, as in Example 4-7. *Example 4-7. Three expressions using operators of mixed precedence*

$$1 + 2 * 3 - 4 * 5$$

$$2 - 4 * 5 * 3 + 1$$

$$5 + 2 - 4 + 1 * 3$$

If there were no operator precedence, these three expressions would evaluate to 25, -29, and 12, respectively. But because multiplication and division take precedence over addition and subtraction, there are implied parentheses around these parts of the expressions, which would look like Example 4-8 if they were visible.

Example 4-8. Three expressions showing implied parentheses

$$1 + (2 * 3) - (4 * 5)$$

$$2 - (4 * 5 * 3) + 1$$

$$5 + 2 - 4 + (1 * 3)$$

Clearly, PHP must evaluate the subexpressions within parentheses first to derive the semi-completed expressions in Example 4-9.

Example 4-9. After evaluating the subexpressions in parentheses

$$1 + (6) - (20)$$

$$2 - (60) + 1$$

$$5 + 2 - 4 + (3)$$

The final results of these expressions are -13 , -57 , and 6 , respectively (quite different from the results of 25 , -29 , and 12 that we would have seen had there been no operator precedence).

Of course, you can override the default operator precedence by inserting your own parentheses and force the original results that we would have seen, had there been no operator precedence (see Example 4-10).

Example 4-10. Forcing left-to-right evaluation

$$((1 + 2) * 3 - 4) * 5$$

$$(2 - 4) * 5 * 3 + 1$$

$$(5 + 2 - 4 + 1) * 3$$

With parentheses correctly inserted, we now see the values 25 , -29 , and 12 , respectively.

Table 4-2 lists PHP's operators in order of precedence from high to low.

Table 4-2. The precedence of PHP operators (high to low)

Operator(s)	Type
()	Parentheses
++ --	Increment/Decrement
!	Logical
* / %	Arithmetic
Operator(s)	Type
+ - .	Arithmetic and string
<< >>	Bitwise
< <= > >= <>	Comparison
== != === !==	Comparison
&	Bitwise (and references)
^	Bitwise
	Bitwise
&&	Logical
	Logical
? :	Ternary
= += -= *= /= .= %= &= != ^=	Assignment
<<= >>=	

and	Logical
xor	Logical
or	Logical

Associativity

We've been looking at processing expressions from left to right, except where operator precedence is in effect. But some operators can also require processing from right to left. The direction of processing is called the operator's *associativity*.

This associativity becomes important in cases in which you do not explicitly force precedence. Table 4-3 lists all the operators that have right-to-left associativity.

Table 4-3. Operators with right-to-left associativity

Operator Description	
NEW	Create a new object
!	Logical NOT
~	Bitwise NOT
++ --	Increment and decrement
+ -	Unary plus and negation
(int)	Cast to an integer

(double) Cast to a float

(string) Cast to a string

(array) Cast to an array

(object) Cast to an object

@ Inhibit error reportin

Conditionals

Conditionals alter program flow. They enable you to ask questions about certain things and respond to the s you get in different ways. Conditionals are central to dy-namic web pages—the goal of using PHP in the first place—because they make it easy to create different output each time a page is viewed.

There are three types of nonlooping conditionals: the if statement, the switch state-ment, and the ? operator. By nonlooping, I mean that the actions initiated by the state-ment take place and program flow then moves on, whereas looping conditionals (which we’ll come to shortly) execute code over and over until a condition has been met.

The if Statement

One way of thinking about program flow is to imagine it as a single-lane highway that you are driving along. It’s pretty much a straight line, but now and then you encounter various signs telling you where to go.

In the case of an if statement, you could imagine coming across a detour sign that you have to follow if a certain condition is TRUE. If so, you drive off and follow the detour until you rejoin your original route; you then continue on your way in your original direction. Or, if the condition isn’t TRUE, you ignore the detour and carry on driving

The contents of the if condition can be any valid PHP expression, including equality, comparison, tests for zero and NULL, and even the values returned by functions (either built-in functions or ones that you write).

The action to take when an if condition is TRUE are generally placed inside curly braces, {}. You can omit the braces if you have only a single statement to execute, but if you always use curly braces you'll avoid potentially difficult-to-trace bugs, such as when you add an extra line to a condition but forget to add the braces in, so it doesn't get evaluated. (Note that for reasons of layout and clarity, many of the examples in this book ignore this suggestion and omit the braces for single statements.)

In Example 4-19, imagine that it is the end of the month and all your bills have been paid, so you are performing some bank account maintenance.

Example 4-19. An if statement with curly braces

```
<?php

if ($bank_balance < 100)

{

    $money = 1000;

    $bank_balance += $money;

}

?>
```

In this example, you are checking your balance to see whether it is less than 100 dollars (or whatever your currency is). If so, you pay yourself 1000 dollars and then add it to the balance. (If only making money were that simple!)

If the bank balance is 100 dollars or greater, the conditional statements are ignored and program flow skips to the next line (not shown).

In this book, opening curly braces generally start on a new line. Some people like to place the first curly brace to the right of the conditional expression instead. Either of these approaches is fine, because PHP allows you to set out your whitespace characters (spaces, newlines, and tabs) any way you choose. However, you will find your code easier to read and debug if you indent each level of conditionals with a tab.

The else Statement

Sometimes when a conditional is not TRUE, you may not want to continue on to the main program code immediately but might wish to do something else instead. This is where the else statement comes in. With it, you can set up a second detour on your highway, as in Figure 4-2.

What happens with an if...else statement is that the first conditional statement is executed if the condition is TRUE, but if it's FALSE, the second one is executed. One of the two choices *must* be executed. Under no circumstances can both (or neither) be executed. Example 4-20 shows the use of the if...else structure.

Example 4-20. An if...else statement with curly braces

```
<?php
```

```
if ($bank_balance < 100)
```

```
{
```



```

    $money = 1000;

    $bank_balance += $money;

}


---


else

{

    $savings += 50;
    $bank_balance -= 50;

}

?>

```

In this example, having ascertained that you have over \$100 in the bank, the else statement is executed, by which you place some of this money into your savings account.

As with if statements, if your else has only one conditional statement, you can opt to leave out the curly braces. (Curly braces are always recommended, though: they make the code easier to understand, and they let you easily add more statements to the branch later.)**The elseif Statement**

There are also times when you want a number of different possibilities to occur, based upon a sequence of conditions. You can achieve this using the elseif statement. As you might imagine, it is like an else statement, except that you place a further conditional expression prior to the conditional code. In Example 4-21, you can see a complete if...elseif...else construct.

Example 4-21. An if...elseif...else statement with curly braces

```
<?php

if ($bank_balance < 100)

{

    $money = 1000;

    $bank_balance += $money;

}

elseif ($bank_balance > 200)

{

    $savings +=
    $savings 100;

    $bank_bal -=
    ance     100;

}

else

{

    $savings +=
    $savings 50;

    $bank_bal -=
    ance     50;

}
```

?>

In this example, an elseif statement has been inserted between the if and else state-ments. It checks whether your bank balance exceeds \$200 and, if so, decides that you can afford to save \$100 of it this month.

Although I'm starting to stretch the metaphor a bit too far, you can imagine this as a multiway set of detours (see Figure 4-3).An else statement closes one of the following: an if...else statement or an if...elseif...else statement. You can leave out a final else if it is not required, but you cannot have one before an elseif; neither can you have an elseif before an if statement.

You may have as many elseif statements as you like, but as the number of elseif statements increases it becomes advisable to consider a switch statement instead, if it fits your needs. We'll look at that next.

The switch Statement

The switch statement is useful in cases in which one variable or the result of an ex-pression can have multiple values, which should each trigger a different function.

For example, consider a PHP-driven menu system that passes a single string to the main menu code according to what the user requests. Let's say the options are Home, About, News, Login, and Links, and we set the variable \$page to one of these, according to the user's input.

The code for this written using if...elseif...else might look like Example 4-22.

Example 4-22. A multiple-line if...elseif...statement

<?php

```
if ($page == "Home") echo "You selected  
Home"; elseif ($page == "About") echo  
"You selected About"; elseif ($page ==  
"News") echo "You selected News"; elseif  
($page == "Login") echo "You selected  
Login"; elseif ($page == "Links") echo  
"You selected Links"; ?>
```

Example 4-23. A switch statement

```
<?php  
  
switch ($page)  
{  
  
    case "Home":  
  
        echo "You selected Home";  
  
        break;  
  
    case "About":  
  
        echo "You selected About";  
  
        break;  
  
    case "News":  
  
        echo "You selected News";  
  
        break;
```

```

case "Login":

    echo "You selected Login";

    break;

case "Links":

    echo "You selected Links";

    break;

}

?>

```

As you can see, `$page` is mentioned only once at the start of the switch statement. Thereafter, the case command checks for matches. When one occurs, the matching conditional statement is executed. Of course, in a real program you would have code here to display or jump to a page, rather than simply telling the user what was selected. One thing to note about switch statements is that you do not use curly braces inside case commands. Instead, they commence with a colon and end with the break statement. The entire list of cases in the switch statement is enclosed in a set of curly braces, though.

Breaking out

If you wish to break out of the switch statement because a condition has been fulfilled, use the break command. This command tells PHP to break out of the switch and jump to the following statement.

If you were to leave out the break commands in Example 4-23 and the case of “Home” evaluated to be TRUE, all five cases would then be executed. Or if `$page` had the value “News,” all the case commands from

then on would execute. This is deliberate and allows for some advanced programming, but generally you should always remember to issue a break command every time a set of case conditionals has finished executing. In fact, leaving out the break statement is a common error.

Default action

A typical requirement in switch statements is to fall back on a default action if none of the case conditions are met. For example, in the case of the menu code in

Example 4-23, you could add the code in Example 4-24 immediately before the final curly brace.

Example 4-24. A default statement to add to Example 4-23

```
default: echo "Unrecognized selection";  
  
break;
```

Although a break command is not required here because the default is the final sub-statement, and program flow will automatically continue to the closing curly brace, should you decide to place the default statement higher up it would definitely need a break command to prevent program flow from dropping into the following statements. Generally, the safest practice is to always include the break command.

Alternative syntax

If you prefer, you may replace the first curly brace in a switch statement with a single colon and the final curly brace with an endswitch command, as in Example 4-25. However, this approach is not commonly used and is mentioned here only in case you encounter it in third-party code.

Example 4-25. Alternate switch statement syntax

```
<?php
```

```
switch ($page):
```

```
    case "Home":
```

```
        echo "You selected Home";
```

```
        break;
```

```
    // etc...
```

```
    case "Links":
```

```
        echo "You selected Links";
```

```
        break;
```

```
endswitch;
```

```
?>
```

Looping

One of the great things about computers is that they can repeat calculating tasks quickly and tirelessly. Often you may want a program to repeat the same sequence of code again and again until something happens, such as a user inputting a value or reaching a natural end. PHP's various loop structures provide the perfect way to do this.

To picture how this works, take a look at Figure 4-4. It is much the same as the highway metaphor used to illustrate if statements, except that the detour also has a loop section that—once a vehicle has entered—can be exited only under the right program conditions.

while Loops

Let's turn the digital car dashboard in Example 4-26 into a loop that continuously checks the fuel level as you drive using a while loop (Example 4-28).

Example 4-28. A while loop

```
<?php

$fuel = 10;

while ($fuel > 1)

{

    // Keep driving ...

    echo "There's enough fuel";

}

?>
```

do...while Loops

A slight variation to the while loop is the do...while loop, used when you want a block of code to be executed at least once and made conditional only after that. Example 4-31 shows a modified version of our code for the 12 times table using such a loop.

Example 4-31. A do...while loop for printing the times table for 12

```
<?php

$count = 1;

do

    echo "$count times 12 is " . $count * 12 . "<br />";

while (++$count <= 12);

?>
```

Notice that we are back to initializing \$count to 1 (rather than 0), because the code is being executed immediately, without an opportunity to increment the variable. Other than that, though, the code looks pretty similar to Example 4-29.

for Loops

The final kind of loop statement, the for loop, is also the most powerful, as it combines the abilities to set up variables as you enter the loop, test for conditions while iterating loops, and modify variables after each iteration.

Example 4-33 shows how you could write the multiplication table program with a for loop. *Example 4-33. Outputting the 12 times table from a for loop*

```
<?php

for ($count = 1 ; $count <= 12 ; ++$count)

    echo "$count times 12 is " . $count * 12 . "<br />";

?>
```

See how the entire code has been reduced to a single for statement containing a single conditional statement? Here's what is going on. Each for statement takes three pa-rameters:

- An initialization expression
- A condition expression
- A modification expression

These are separated by semicolons, like this: `for (expr1 ; expr2 ; expr3)`. At the start of the first iteration of the loop, the initialization expression is executed. In the case of the times table code, \$count is initialized to the value 1. Then, each time around the loop, the condition expression (in this case, \$count <= 12) is tested, and the loop is entered only if the condition is TRUE. Finally, at the end of each iteration, the modification expression is executed. In the case of the times table code, the variable \$count is incremented.

Unit-2

PHP Functions and Objects ,Array

The basic requirements of any programming language include somewhere to store data, a means of directing program flow, and a few bits and pieces such as expression evaluation, file management, and text output. PHP has all these, plus tools like `else` and `elseif` to make life easier. But even with all these in your toolkit, programming can be clumsy and tedious, especially if you have to rewrite portions of very similar code each time you need them.

That's where functions and objects come in. As you might guess, a *function* is a set of statements that performs a particular function and—optionally—returns a value. You can pull out a section of code that you have used more than once, place it into a function, and call the function by name when you want the code.

Functions have many advantages over contiguous, inline code:

- Less typing is involved.
- Functions reduce syntax and other programming errors.
- They decrease the loading time of program files.
- They also decrease execution time, because each function is compiled only once, no matter how often you call it.
- Functions accept arguments and can therefore be used for general as well as specific cases.

Objects take this concept a step further. An *object* incorporates one or more functions, and the data they use, into a single structure called a *class*.

In this chapter, you'll learn all about using functions, from defining and calling them to passing arguments back and forth. With that knowledge under your belt, you'll start creating functions and using them in your own objects (where they will be referred to as *methods*). **PHP Functions**

PHP comes with hundreds of ready-made, built-in functions, making it a very rich language. To use a function, call it by name. For example, you can see the print function in action here:

```
print("print is a function");
```

The parentheses tell PHP that you're referring to a function. Otherwise, it thinks you're referring to a constant. You may see a warning such as this:

Notice: Use of undefined constant *fname* - assumed '*fname*'

followed by the text string *fname*, under the assumption that you must have wanted to put a literal string in your code. (Things are even more confusing if there is actually a constant named *fname*, in which case PHP uses its value.)

Strictly speaking, print is a pseudofunction, commonly called a *con-struct*. The difference is that you can omit the parentheses, as follows:

```
print "print doesn't require parentheses";
```

You do have to put parentheses after any other function you call, even if it's empty (that is, if you're not passing any argument to the function).

Functions can take any number of arguments, including zero. For example, `phpinfo`, as shown below, displays lots of information about the current installation of PHP and requires no argument. The result of calling this function can be seen in Figure 5-1.

`phpinfo()`;The `phpinfo` function is extremely useful for obtaining information about your current PHP installation, but that information could also be very useful to potential hackers. Therefore, never leave a call to this function in any web-ready code.

Some of the built-in functions that use one or more arguments appear in Example 5-1.

Example 5-1. Three string functions

```
<?php
echo strrev(" .dlrow      olleH"); // Reverse string
echo str_repeat("Hip      ", 2);    // Repeat string
echo strtoupper("hooray!");         // String to uppercase
?>
```

This example uses three string functions to output the following text:

Hello world. Hip Hip HOORAY!

~~As you can see, the `strrev` function reversed the order of the characters in the string, `str_repeat` repeated the string “Hip ” twice (as required by a second argument), and `strtoupper` converted “hooray!” to uppercase.~~

Defining a Function

The general syntax for a function is:

```
function function_name([parameter [, ...]])
```

```
{  
  
    // Statements  
  
}
```

The first line of the syntax indicates that:

- A definition starts with the word `function`.
 - Following that is a name, which must start with a letter or underscore, followed by any number of letters, numbers, or underscores.
 - The parentheses are required.
 - One or more parameters, separated by commas, are optional (indicated by the square brackets, which are not part of the function syntax).
-

Function names are case-insensitive, so all of the following strings can refer to the print function: `PRINT`, `Print`, and `PrInT`.

The opening curly brace starts the statements that will execute when you call the function; a matching curly brace must close it. These statements may include one or more return statements, which force the function to cease execution and return to the calling code. If a value is attached to the return statement, the calling code can retrieve it, as we'll see next.

Returning a Value

Let's take a look at a simple function to convert a person's full name to lowercase and then capitalize the first letter of each part of the name.

We've already seen an example of PHP's built-in `strtoupper` function in Example 5-1.

For our current function, we'll use its counterpart, `strtolower`:

```
$lowered = strtolower("aNY # of Letters and  
Punctuation you WANT"); echo $lowered;
```

The output of this experiment is:

```
any # of letters and punctuation you want
```

We don't want names all lowercase, though; we want the first letter of each part of the name capitalized. (We're not going to deal with subtle cases such as Mary-Ann or Jo-En-Lai, for this example.) Luckily, PHP also provides a `ucfirst` function that sets the first character of a string to uppercase:

```
$ucfixed = ucfirst("any # of letters and punctuation  
you want"); echo $ucfixed;
```

The output is:

```
Any # of letters and punctuation you want
```

Now we can do our first bit of program design: to get a word with its initial letter capitalized, we call `strtolower` on a string first, and then `ucfirst`. The way to do this is to nest a call to `strtolower` within `ucfirst`. Let's see why, because it's important to understand the order in which code is evaluated.

If you make a simple call to the `print` function:

```
print(5-8);
```

The expression 5-8 is evaluated first, and the output is -3 . (As you saw in the previous chapter, PHP converts the result to a string in order to display it.) If the expression contains a function, that function is also evaluated at this point:

```
print(abs(5-8));
```

PHP is doing several things in executing that short statement:

Returning an Array

We just saw a function returning a single value. There are also ways of getting multiple values from a function.

The first method is to return them within an array. As you saw in Chapter 3, an array is like a bunch of variables stuck together in a row. Example 5-3 shows how you can use an array to return function values.

Example 5-3. Returning multiple values in an array

```
<?php
```

```
$names = fix_names("WILLIAM",  
"henry", "gatES"); echo $names[0] . " " .  
$names[1] . " " . $names[2];
```



```
function fix_names($n1, $n2, $n3){  
    $n1 = ucfirst(strtolower($n1));  
  
    $n2 = ucfirst(strtolower($n2));  
  
    $n3 = ucfirst(strtolower($n3));  
  
    return array($n1, $n2, $n3);  
  
}  
  
?>
```

This method has the benefit of keeping all three names separate, rather than concatenate them into a single string, so you can refer to any user simply by first or last name, without having to extract either name from the returned string.

Passing by Reference

In PHP, prefacing a variable name with the `&` symbol tells the parser to pass a reference to the variable's value, not the value itself. This concept can be hard to get your head around, so let's go back to the matchbox metaphor from Chapter 3.

Imagine that, instead of taking a piece of paper out of a matchbox, reading it, copying it to another piece of paper, putting the original back, and passing the copy to a function (phew!), you simply attach a piece of thread to the original piece of paper and pass one end of it to the function. Now the function can follow the thread to find the data to be accessed. This avoids all the overhead of creating a copy of the variable just for the

function's use. What's more, the function can now modify the variable's value.

This means you can rewrite Example 5-3 to pass references to all the parameters, and then the function can modify these directly (see Example 5-4).

Example 5-4. Returning values from a function by reference

```
<?php

$a1 = "WILLIAM";

$a2 = "henry";

$a3 = "gatES";

echo $a1 . " " . $a2 . " " . $a3 . "<br />";

fix_names($a1, $a2, $a3);

echo $a1 . " " . $a2 . " " . $a3;

function fix_names(&$n1, &$n2, &$n3)

{

    $n1 = ucfirst(strtolower($n1));

    $n2 = ucfirst(strtolower($n2));
```

```
$n3 = ucfirst(strtolower($n3));  
  
}  
  
?>
```

Rather than passing strings directly to the function, you first assign them to variables and print them out to see their “before” values. Then you call the function as before, but put an & symbol in front of each parameter, which tells PHP to pass the variables’ references only.

Now the variables \$n1, \$n2, and \$n3 are attached to “threads” that lead to the values of \$a1, \$a2, and \$a3. In other words, there is one group of values, but two sets of variable names are allowed to access them.

Therefore, the function fix_names only has to assign new values to \$n1, \$n2, and \$n3 to update the values of \$a1, \$a2, and \$a3. The output from this code is:

```
WILLIAM henry gatES
```

```
William Henry Gates
```

As you see, both of the echo statements use only the values of \$a1, \$a2, and \$a3. Be careful when passing values by reference. If you need to keep the original values, make copies of your variables and then pass the copies by reference.

Returning Global Variables

You can also give a function access to an externally created variable by declaring it a global variable from within the function. The global keyword followed by the variable name gives every part of your code full access to it (see Example 5-5).

Example 5-5. Returning values in global variables

```
<?php
$a1 = "WILLIAM";
$a2 = "henry";
$a3 = "gatES";
echo $a1 . " " . $a2 . " " . $a3 . "<br />";
fix_names();
echo $a1 . " " . $a2 . " " . $a3;
function fix_names()
{
    global $a1; $a1 = ucfirst(strtolower($a1));
    global $a2; $a2 = ucfirst(strtolower($a2));
    global $a3; $a3 = ucfirst(strtolower($a3));
}
?>
```

Now you don't have to pass parameters to the function, and it doesn't have to accept them. Once declared, these variables remain global and available to the rest of your program, including its functions.

If at all possible, in order to retain as much local scope as possible, you should try returning arrays or using variables by association. Otherwise, you will begin to lose some of the benefits of functions.

Recap of Variable Scope

A quick reminder of what you know from Chapter 3:

- *Local variables* are accessible just from the part of code where you define them. If they're outside of a function, they can be accessed by all code outside of functions, classes, and so on. If a variable is inside a function, only that function can access the variable, and its value is lost when the function returns.
- *Global variables* are accessible from all parts of your code.
- *Static variables* are accessible only within the function that declared them but retain their value over multiple calls.

Including and Requiring Files

As you progress in your use of PHP programming, you are likely to start building a library of functions that you think you will need again. You'll also probably start using libraries created by other programmers.

There's no need to copy and paste these functions into your code. You can save them in separate files and use commands to pull them in. There are two types of commands to perform this action: include and require.

The include Statement

Using include, you can tell PHP to fetch a particular file and load all its contents. It's as if you pasted the included file into the current file at the insertion point. Example 5-6 shows how you would include a file called *library.php*.

Example 5-6. Including a PHP file

```
<?php
```

```
include "library.php";
```

```
// Your code goes here
```

```
?>
```

Using `include_once`

Each time you issue the `include` directive, it includes the requested file again, even if you've already inserted it. For instance, suppose that *library.php* contains a lot of useful functions, so you include it in your file. Now suppose you also include another library that includes *library.php*. Through nesting, you've inadvertently included *library.php* twice. This will produce error messages, because you're trying to define the same constant or function multiple times. To avoid this problem, use `include_once` instead (see Example 5-7).

Example 5-7. Including a PHP file only once

```
<?php
```

```
include_once "library.php";
```

```
// Your code goes here
```

```
?>
```

Then, if another `include` or `include_once` for the same file is encountered, PHP will verify that it has already been loaded and, if so, will ignore it. To determine whether the file has already been executed, PHP resolves all relative paths and checks whether the absolute file path is found in your include path. In general, it's probably best to stick with `include_once` and ignore the basic `include` statement. That way you will never have the problem of files being included multiple times.

Using require and require_once

A potential problem with include and include_once is that PHP will only *attempt* to include the requested file. Program execution continues even if the file is not found.

When it is absolutely essential to include a file, require it. For the same reasons I gave for using include_once, I recommend that you generally stick with require_once when-ever you need to require . *Requiring a PHP file only once*

```
<?php
require_once "library.php";

// Your code goes here

?>
```

PHP Version Compatibility

PHP is in an ongoing process of development, and there are multiple versions. If you need to check whether a particular function is available to your code, you can use the function_exists function, which checks all predefined and user-created functions.

Example 5-9 checks for the function array_combine, which is specific to PHP version 5.

Example 5-9. Checking for a function's existence

```
<?php

if (function_exists("array_combine"))
```

```
{echo "Function exists";}  
  
else{echo "Function does not exist - better write our own";}  
  
?>
```

Using code such as this, you can identify any features available in newer versions of PHP that you will need to replicate if you want your code to still run on earlier versions. Your functions may be slower than the built-in ones, but at least your code will be much more portable. You can also use the `phpversion` function to determine which version of PHP your code is running on. The returned result will be similar to the following, depending on the version:5.2.8

PHP Objects

In much the same way that functions represent a huge increase in programming power over the early days of computing, where sometimes the best program navigation available was a very basic GOTO or GOSUB statement, object-oriented programming (OOP) takes the use of functions to a whole new level.

Once you get the hang of condensing reusable bits of code into functions, it's not that great a leap to consider bundling the functions and their data into objects. ~~Let's consider a social networking site that has many parts.~~ One handles all user functions: code to enable new users to sign up and to enable existing users to modify their details. In standard PHP, you might create a few functions to handle this and embed some calls to the MySQL database to keep track of all the users.

Imagine how much easier it would be, though, to create an object to represent the current user. To do this you could create a class, perhaps called `User`, which would contain all the code required for handling users and all the variables needed for manipulating the data within the class. Then, whenever you needed to manipulate a user's data, you could simply create a new object with the `User` class.

You could treat this new object as if it were the actual user. For example, you could pass the object a name, password, and email address; ask it

whether such a user already exists; and, if not, have it create a new user with those attributes. You could even have an instant messaging object, or one for managing whether two users are friends.

Terminology

When creating a program to use objects, you need to design a composite of data and code called a *class*. Each new object based on this class is called an *instance* (or *occurrence*) of that class.

The data associated with an object are called its *properties*; the functions it uses are called *methods*. In defining a class, you supply the names of its properties and the code for its methods. See Figure 5-3 for a jukebox metaphor for an object. Think of the CDs that it holds in the carousel as its properties; the method of playing them is to press buttons on the front panel. There is also the slot for inserting coins (the method used to activate the object), and the laser disc reader (the method used to retrieve the music, or properties, from the CDs).

When creating objects, it is best to use *encapsulation*, or writing a class in such a way that only its methods can be used to manipulate its properties. In other words, you deny outside code direct access to its data. The methods you supply are known as the object's *interface*.

This approach makes debugging easy: you have to fix faulty code only within a class. Additionally, when you want to upgrade a program, if you have used proper encapsulation and maintained the same interface, you can simply develop new replacement classes, debug them fully, and then swap them in for the old ones. If they don't work, you can swap the old ones back in to immediately fix the problem before further debugging the new classes.

Once you have created a class, you may find that you need another class that is similar to it but not quite the same. The quick and easy thing to do

is to define a new class using *inheritance*. When you do this, your new class has all the properties of the one from which it has inherited. The original class is now called the *superclass*, and the new one is the *subclass* (or *derived* class).

In our jukebox example, if you invent a new jukebox that can play a video along with the music, you can inherit all the properties and methods from the original jukebox superclass and add some new properties (videos) and new methods (a movie player).

An excellent benefit of this system is that if you improve the speed or any other aspect of the superclass, its subclasses will receive the same benefit.

Declaring a Class

Before you can use an object, you must define a class with the `class` keyword. Class definitions contain the class name (which is case-sensitive), its properties, and its methods. Example 5-10 defines the class `User` with two properties: `$name` and `$password` (indicated by the `public` keyword—see “Property and Method Scope in PHP 5” on page 112, later in this chapter). It also creates a new instance (called `$object`) of this class.

Example 5-10. Declaring a class and examining an object

```
<?php
$object = new User;


---


print_r($object);

class User{
    public $name, $password;

    function save_user()
```

```
{echo "Save User code goes here";}  
}?>
```

Here I have also used an invaluable function called `print_r`. It asks PHP to display information about a variable in human-readable form (the `_r` stands for “in human-readable format”). In the case of the new object `$object`, it prints the following:

User Object

```
([name] =>  
  
    [password] =>  
  
)
```

However, a browser compresses all the whitespace, so the output in the browser is slightly harder to read:

```
User Object ( [name] => [password] => )
```

In any case, the output says that `$object` is a user-defined object that has the properties `name` and `password`.

Creating an Object

To create an object with a specified class, use the `new` keyword, like this: `$object = new Class`. Here are a couple of ways in which we could do this:

```
$object = new User;  
  
$temp= new User('name', 'password');
```

On the first line, we simply assign an object to the `User` class. In the second, we pass parameters to the call.

A class may require or prohibit arguments; it may also allow arguments, but not require them.

Accessing Objects

Let's add a more few lines to Example 5-10 and check the results.

Example 5-11 extends the previous code by setting object properties and calling a method. *Example 5-11. Creating and interacting with an object*

```
<?php  
  
$object = new User;  
  
print_r($object); echo "<br />";  
  
$object->name           = "Joe";  
$object->password       = "mypass";  
print_r($object);      echo "<br />";  
$object->save_user();  
  
class User  
{public $name, $password;  
    function save_user()  
    {echo "Save User code goes here";}  
}  
  
?>
```

As you can see, the syntax for accessing an object's property is *\$object->property*.

Likewise, you call a method like this: *\$object->method()*.

You should note that the property and method names do not have dollar signs (\$) in front of them. If you were to preface them with a \$, the code would not work, as it would try to reference the value inside a variable. For example, the expression *\$object->\$property* would attempt to look up the value assigned to a variable named *\$property* (let's say that value is the string "brown") and then attempt to reference the property *\$object->brown*. If *\$property* is undefined, an attempt to reference *\$object->NULL* will occur and cause an error.

When looked at using a browser's view source facility, the output from Example 5-11 is:

```
User Object
```

```
([name] =>
```

```
  [password] =>)
```

```
User Object
```

```
([name]   => Joe
```

```
  [password] => mypass
```

```
)
```

```
Save User code goes here
```

Again, `print_r` shows its utility by providing the contents of `$object` before and after property assignment. From now on I'll omit `print_r` statements, ~~but if you are working along with this book on your development server,~~ you can put some in to see exactly what is happening. You can also see that the code in the method `save_user` was executed via the call to that method. It printed the string reminding us to create some code. You can place functions and class definitions anywhere in your code, before or after statements that use them. Generally, though, it is con-sidered good practice to place them toward the end of a file.

Cloning objects

Once you have created an object, it is passed by reference when you pass it as a pa-rameter. In the matchbox metaphor, this is like keeping several threads attached to an object stored in a matchbox, so that you can follow any attached thread to access it.

In other words, making object assignments does not copy objects in their entirety. You'll see how this works in Example 5-12, where we define a very simple User class with no methods and only the property name.

Example 5-12. Copying an object?

```
<?php

$object1    = new User();

$object1->name = "Alice";

$object2    = $object1;

$object2->name = "Amy";

echo "object1 name = " . $object1->name
. "<br />"; echo "object2 name = " .
$object2->name;

class User
{public $name;}

?>
```

We've created the object \$object1 and assigned the value "Alice" to the name property. Then we created \$object2, assigning it the value of \$object1, and assigned the value "Amy" just to the name property of \$object2—or so we might think. But this code outputs the following:

```
object1 name = Amy
```

```
object2 name = Amy
```

What has happened? Both \$object1 and \$object2 refer to the *same* object, so changing the name property of \$object2 to “Amy” also sets that property for \$object1.

To avoid this confusion, you can use the clone operator, which creates a new instance of the class and copies the property values from the original instance to the new in-stance. Example 5-13 illustrates this usage.

Example 5-13. Cloning an object

```
<?php

$object1    = new User();

$object1->name = "Alice";

$object2    = clone $object1;

$object2->name = "Amy";

echo "object1 name = " . $object1->name . "<br>"; echo "object2 name = " .
$object2->name;

class User
{public $name;
}??>
```

Voilà! The output from this code is what we initially wanted:

```
object1 name = Alice
object2 name = Amy
```

Constructors

When creating a new object, you can pass a list of arguments to the class being called. These are passed to a special method within the class, called the *constructor*, which initializes various properties.

In the past, you would normally give this method the same name as the class, as in Example 5-14.

Example 5-14. Creating a constructor method

```
<?php
class User
{
    function User($param1, $param2)
    {
        {Constructor statements go here public $username = "Guest";
    }
}
?>
```

However, PHP 5 provides a more logical approach to naming the constructor, which is to use the function name `__construct` (that is, `construct` preceded by two underscore characters), as in Example 5-15.

Example 5-15. Creating a constructor method in PHP 5

```
<?php
class User
{
    function __construct($param1, $param2)
    {
        {Constructor statements go here public $username = "Guest";}
    }
}
?>
```


PHP 5 destructors

Also new in PHP 5 is the ability to create *destructor* methods. This ability is useful when code has made the last reference to an object or when a script reaches the end. Example 5-16 shows how to create a destructor method.

Example 5-16. Creating a destructor method in PHP 5

```
<?php

class User

{

    function __destruct()

    {

        // Destructor code goes here

    }

}

?>
```

Writing Methods

As you have seen, declaring a method is similar to declaring a function, but there are a few differences. For example, method names beginning with a double underscore (__) are reserved and you should not create any of this form

You also have access to a special variable called `$this`, which can be used to access the current object's properties. To see how this works, take a look at Example 5-17, which contains a different method from the `User` class definition called `get_password`.

Example 5-17. Using the variable \$this in a method

```
<?php
class User
{
    public $name, $password;

    function get_password()
    {
        return $this->password;
    }
}
?>
```

What `get_password` does is use the `$this` variable to access the current object and then return the value of that object's `password` property. Note how the `$` is omitted from the property `$password` when using the `->` operator. Leaving the `$` in place is a typical error you may run into, particularly when you first use this feature.

Here's how you would use the class defined in Example 5-17:

```
$object= new User;

$object->password = "secret";

echo $object->get_password();
```

This code prints the password "secret".

Static methods in PHP 5

If you are using PHP 5, you can also define a method as *static*, which means that it is called on a class and not on an object. A static method has no access to any object properties and is created and accessed as in Example 5-18.

Example 5-18. Creating and accessing a static method

```
<?php

User::pwd_string();
```

```

class User
{
    static function pwd_string()
    {
        echo "Please enter your password";
    }
}
?>

```

Note how the class itself is called, along with the static method, using a double colon (::, also known as the *scope resolution* operator) and not ->. Static functions are useful for performing actions relating to the class itself, but not to specific instances of the class. You can see another example of a static method. If you try to access `$this->property`, or other object properties from within a static function, you will receive an error message.

Declaring Properties

It is not necessary to explicitly declare properties within classes, as they can be implicitly defined when first used. To illustrate this, in Example 5-19 the class `User` has no properties and no methods but is legal code.

Example 5-19. Defining a property implicitly

```

<?php

$object1 = new User();

$object1->name = "Alice";

echo $object1->name;

class User {}

?>

```

This code correctly outputs the string “Alice” without a problem, because PHP implicitly declares the variable `$object1->name` for you. But this kind of programming can lead to bugs that are infuriatingly difficult to discover, because `name` was declared from outside the class.

To help yourself and anyone else who will maintain your code, I advise that you get into the habit of always declaring your properties explicitly within classes. You'll be glad you did.

Also, when you declare a property within a class, you may assign a default value to it. The value you use must be a constant and not the result of a function or expression. Example 5-20 shows a few valid and invalid assignments.

Example 5-20. Valid and invalid property declarations

```
<?php
class Test
{
    public $name= "Paul Smith"; // Valid

    public $age          = 42;    // Valid

    public $time          = time(); // Invalid - calls a function

    public $score = $level * 2; // Invalid - uses an expression
}
?>
```

Declaring Constants

In the same way that you can create a global constant with the `define` function, you can define constants inside classes. The generally accepted practice is to use uppercase letters to make them stand out, as in Example 5-21.

Example 5-21. Defining constants within a class

```
<?php
Translate::lookup();

class Translate
```

```

{

    const ENGLISH = 0;

    const SPANISH = 1;

    const FRENCH    = 2;

    const GERMAN    = 3;

    // ...

    static function lookup()

    {echo self::SPANISH;

    }}

?>

```

Constants can be referenced directly, using the `self` keyword and the `double colon` operator. Note that this code calls the class directly, using the double colon operator at line 1, without creating an instance of it first. As you would expect, the value printed when you run this code is 1.

Remember that once you define a constant, you can't change it.

Property and Method Scope in PHP 5

PHP 5 provides three keywords for controlling the scope of properties and methods: `public`

These properties are the default when declaring a variable using the `var` or `public` keywords, or when a variable is implicitly declared the first time it is used. The keywords `var` and `public` are interchangeable because, although deprecated, `var` is retained for compatibility with previous versions of PHP. Methods are assumed to be `public` by default.

`protected`

These properties and methods (*members*) can be referenced only by the object's class methods and those of any subclasses.

private

These members can be referenced only by methods within the same class—not by subclasses.

Here's how to decide which you need to use:

- Use public when outside code *should* access this member and extending classes *should* also inherit it.
- Use protected when outside code *should not* access this member but extending classes *should* inherit it.
- Use private when outside code *should not* access this member and extending classes also *should not* inherit it.

Example 5-22. Changing property and method scope

```
<?php
```

```
class Example
```

```
{ var $name    = "Michael"; // Same as public but deprecated
```

```
    public $age = 23;    // Public property
```

```
    protected $usercount;    // Protected property
```

```
    private function admin() // Private method
```

```
    { // Admin code goes here } }
```

```
?>
```

Static properties and methods

Most data and methods apply to instances of a class. For example, in a User class, you want to do such things as set a particular user's password or check when the user has been registered. These facts and operations apply separately to each user and therefore use instance-specific properties and methods.

But occasionally you'll want to maintain data about a whole class. For instance, to report how many users are registered, you will store a variable that applies to the whole User class. PHP provides static properties and methods for such data.

As shown briefly in Example 5-18, declaring members of a class static makes them accessible without an instantiation of the class. A property declared static cannot be directly accessed within an instance of a class, but a static method can.

Example 5-23 defines a class called Test with a static property and a public method.

Example 5-23. Defining a class with a static property

```
<?php
$temp = new Test();

echo "Test A: " . Test::$static_property . "<br />";
echo "Test B: " . $temp->get_sp() . "<br />";
echo "Test C: " . $temp->static_property . "<br />";

class Test

{ static $static_property = "I'm static";

  function get_sp()

  { return self::$static_property; }
```

```
}
```

```
?>
```

When you run this code, it returns the following output:

Test A: I'm static

Test B: I'm static

Notice: Undefined property: Test::\$static_property

Test C:

This example shows that the property `$static_property` could be directly referenced from the class itself using the double colon operator in *Test A*. Also, *Test B* could obtain its value by calling the `get_sp` method of the object `$temp`, created from class `Test`. But *Test C* failed, because the static property `$static_property` was not accessible to the object `$temp`. Note how the method `get_sp` accesses `$static_property` using the keyword `self`. This is the way in which a static property or constant can be directly accessed within a class.

Inheritance

Once you have written a class, you can derive subclasses from it. This can save lots of painstaking code rewriting: you can take a class similar to the one you need to write, extend it to a subclass, and just modify the parts that are different. This is achieved using the `extends` operator.

In Example 5-24, the class `Subscriber` is declared a subclass of `User` by means of the `extends` operator.

Example 5-24. Inheriting and extending a class

```
<?php
```

```
$object      = new Subscriber;
```

```
$object->name  = "Fred";
```

```
$object->password = "pword";
```



```
$object->phone      = "012 345 6789";
```

```
$object->email       = "fred@bloggs.com";
```

```
$object->display();
```

```
class User
```

```
{public $name, $password;
```

```
    function save_user()
```

```
    {echo "Save User code goes here";
```

```
    }
```

```
}
```

```
class Subscriber extends User
```

```
{public $phone, $email;
```

```
    function display()
```

```
    {
```

```
        echo "Name:      " . $this->name . "<br />";
```

```
        echo "Pass:      " . $this->password . "<br />";
```

```
        echo "Phone: " . $this->phone . "<br />";
```

```
        echo "Email: " . $this->email;
```

```
    }
```

```
}
```

```
?>
```

The original User class has two properties, \$name and \$password, and a method to save the current user to the database. Subscriber extends this class by adding an additional two properties, \$phone and \$email, and includes a method of displaying the properties of the current object using the variable \$this, which refers to the current values of the object being accessed. The output from this code is:

Name: Fred

Pass: pword

Phone: 012 345 6789

Email: fred@bloggs.com

PHP Arrays

A very brief introduction to PHP's arrays—just enough for a little taste of their power. In this chapter, I'll show you many more things that you can do with arrays, some of which—if you have ever used a strongly typed language such as C—may surprise you with their elegance and simplicity.

Arrays are an example of what has made PHP so popular. Not only do they remove the tedium of writing code to deal with complicated data structures, but they also provide numerous ways to access data while remaining amazingly fast.

Basic Access

We've already looked at arrays as if they were clusters of matchboxes glued together. Another way to think of an array is like a string of beads, with the beads representing variables that can be numbers, strings, or even other arrays. They are like bead strings, because each element has its own location and (with the exception of the first and last ones) each has other elements on either side.

Some arrays are referenced by numeric indexes; others allow alphanumeric identifiers. Built-in functions let you sort them, add or remove sections, and walk through them to handle each item through a special kind of loop. And by placing one or more arrays inside another, you can create arrays of two, three, or any number of dimensions.

Numerically Indexed Arrays

Let's assume that you've been tasked with creating a simple website for a local office supplies company and you're currently working on the section devoted to paper. One way to manage the various items of stock in this

category would be to place them in a numeric array. You can see the simplest way of doing so in Example 6-1.

Example 6-1. Adding items to an array

```
<?php

$paper[] = "Copier";

$paper[] = "Inkjet";

$paper[] = "Laser";

$paper[] = "Photo";

print_r($paper);

?>
```

In this example, each time you assign a value to the array `$paper`, the first empty location within that array is used to store the value and a pointer internal to PHP is incremented to point to the next free location, ready for future insertions. The familiar `print_r` function (which prints out the contents of a variable, array, or object) is used to verify that the array has been correctly populated. It prints out the following:

```
Array
(
    [0] => Copier
    [1] => Inkjet
```

```
[2] => Laser  
[3] => Photo  
)
```

The previous code could equally have been written as in Example 6-2, where the exact location of each item within the array is specified. But, as you can see, that approach requires extra typing and makes your code harder to maintain if you want to insert supplies or remove supplies from the array. So, unless you wish to specify a different order, it's usually better to simply let PHP handle the actual location numbers.

Example 6-2. Adding items to an array using explicit locations

```
<?php  
  
$paper[0] = "Copier";  
  
$paper[1] = "Inkjet";  
  
$paper[2] = "Laser";  
  
$paper[3] = "Photo";  
  
print_r($paper);  
  
?>
```

The output from these examples is identical, but you are not likely to use `print_r` in a developed website, so Example 6-3 shows how you might print out the various types of paper the website offers using a for loop.

Example 6-3. Adding items to an array and retrieving them

```
<?php

$paper[] = "Copier";

$paper[] = "Inkjet";

$paper[] = "Laser";

$paper[] = "Photo";

for ($j = 0 ; $j < 4 ; ++$j)

    echo "$j: $paper[$j]<br>";
```

?>This example prints out the following:

```
0: Copier
1: Inkjet
2: Laser
3: Photo
```

So far, you've seen a couple of ways in which you can add items to an array and one way of referencing them, but PHP offers many more. We'll get to those shortly, but first, let's look at another type of array.

Associative Arrays

Keeping track of array elements by index works just fine, but it can require extra work in terms of remembering which number refers to which product. It can also make code hard for other programmers to follow.

This is where associative arrays come into their own. Using them, you can reference the items in an array by name rather than by number. Example 6-4 expands on the previous code by giving each element in the array an identifying name and a longer, more explanatory string value.

Example 6-4. Adding items to an associative array and retrieving them

```
<?php

$paper['copier'] = "Copier & Multipurpose";

$paper['inkjet'] = "Inkjet Printer";

$paper['laser'] = "Laser Printer";

$paper['photo']      = "Photographic Paper";

echo $paper['laser'];

?>
```

In place of a number (which doesn't convey any useful information, aside from the position of the item in the array), each item now has a unique name that you can use to reference it elsewhere, as with the echo statement—which simply prints out Laser Printer. The names (copier, inkjet, and so on) are called *indexes* or *keys* and the items assigned to them (such as “Laser Printer”) are called *values*.

This very powerful feature of PHP is often used when extracting information from XML and HTML. For example, an HTML parser such as those used by a search engine could place all the elements of a web page into an associative array whose names reflect the page's structure:

```
$html['title'] = "My web page";

$html['body']   = "... body of web page ...";
```


The program would also probably break out all the links found within a page into another array, and all the headings and subheadings into another. When you use as-associative rather than numeric arrays, the code to refer to all of these items is easy to write and debug.

Assignment Using the array Keyword

So far, you've seen how to assign values to arrays by just adding new items one at a time. Whether you specify keys, specify numeric identifiers, or let PHP assign numeric identifiers implicitly, this is a long-winded approach. A more compact and faster assignment method uses the array keyword. Example 6-5 shows both a numeric and an associative array assigned using this method.

Example 6-5. Adding items to an array using the array keyword

```
<?php

$p1 = array("Copier", "Inkjet", "Laser", "Photo");

echo "p1 element: " . $p1[2] . "<br>";

$p2 = array('copier' => "Copier &
            Multipurpose", 'inkjet' =>
            "Inkjet Printer", 'laser' =>
            "Laser Printer", 'photo' =>
            "Photographic Paper");

echo "p2 element: " . $p2['inkjet'] . "<br>";

?>
```

The first half of this snippet assigns the old, shortened product descriptions to the array \$p1. There are four items, so they will occupy slots 0 through 3. Therefore, the echo statement prints out the following:

p1 element: Laser

The second half assigns associative identifiers and accompanying longer product descriptions to the array \$p2 using the format *index => value*. The use of => is similar to the regular = assignment operator, except that you are assigning a value to an *index* and not to a *variable*. The index is then inextricably linked with that value, unless it is reassigned a new value. The echo command therefore prints out:

p2 element: Inkjet Printer

You can verify that \$p1 and \$p2 are different types of array, because both of the following commands, when appended to the code, will cause an “undefined index” or “undefined offset” error, as the array identifier for each is incorrect:

```
echo $p1['inkjet']; // Undefined index
```

```
echo $p2[3]; // Undefined offset
```

The foreach...as Loop

The creators of PHP have gone to great lengths to make the language easy to use. So, not content with the loop structures already provided, they added another one especially for arrays: the *foreach...as* loop. Using it, you can step through all the items in an array, one at a time, and do something with them. The process starts with the first item and ends with the last one, so you don't even have to know how many items there are in an array. Example 6-6 shows how *foreach* can be used to rewrite Example 6-3.

Example 6-6. Walking through a numeric array using foreach...as

```
<?php
```

```
$paper = array("Copier", "Inkjet", "Laser",  
"Photo"); $j = 0;
```

```
foreach ($paper as $item)
```

```
{echo "$j: $item<br>";
```

```
    ++$j;
```

```
}
```

```
?>
```

When PHP encounters a foreach statement, it takes the first item of the array and places it in the variable following the as keyword, and each time control flow returns to the foreach the next array element is placed in the as keyword. In this case, the variable \$item is set to each of the four values in turn in the array \$paper. Once all values have been used, execution of the loop ends. The output from this code is exactly the same as for Example 6-3.

Now let's see how foreach works with an associative array by taking a look at Example 6-7, which is a rewrite of the second half of Example 6-5.

Example 6-7. Walking through an associative array using foreach...as

```
<?php
```

```
$paper = array('copier' => "Copier &  
    Multipurpose", 'inkjet' =>  
    "Inkjet Printer", 'laser' =>  
    "Laser Printer", 'photo' =>  
    "Photographic Paper");
```

```
foreach ($paper as $item => $description)
```

```
    echo "$item: $description<br>";
```

```
?>
```

Remember that associative arrays do not require numeric indexes, so the variable \$j is not used in this example. Instead, each item of the array

\$paper is fed into the key/value pair of variables \$item and \$description, from where they are printed out. The result of this code is as follows:

copier: Copier & Multipurpose

inkjet: Inkjet Printer

laser: Laser Printer

photo: Photographic Paper

As an alternative syntax to foreach...as, you can use the list function in conjunction with the each function, as in Example 6-8. *Example 6-8. Walking through an associative array using each and list*

```
<?php
```

```
$paper = array('copier' => "Copier &  
    Multipurpose", 'inkjet' =>  
    "Inkjet Printer", 'laser' =>  
    "Laser Printer", 'photo' =>  
    "Photographic Paper");
```

```
while (list($item, $description) = each($paper))
```

```
    echo "$item: $description<br>";
```

```
?>
```

In this example, a while loop is set up and will continue looping until the each function returns a value of FALSE. The each function acts like foreach: it returns an array containing a key/value pair from the array

\$paper and then moves its built-in pointer to the next pair in that array. When there are no more pairs to return, each returns FALSE.

The list function takes an array as its argument (in this case, the key/value pair re-turned by function each) and then assigns the values of the array to the variables listed within parentheses.

You can see how list works a little more clearly in Example 6-9, where an array is created out of the two strings “Alice” and “Bob” and then passed to the list function, which assigns those strings as values to the variables \$a and \$b.

Example 6-9. Using the list function

```
<?php
list($a, $b) = array('Alice', 'Bob');
echo "a=$a b=$b";
?>
```

The output from this code is:

```
a=Alice b=Bob
```

You can take your pick when walking through arrays. Use foreach...as to create a loop that extracts values to the variable following the as, or use the each function and create your own looping system.

Multidimensional Arrays

A simple design feature in PHP’s array syntax makes it possible to create arrays of more than one dimension. In fact, they can be as many dimensions as you like (although it’s a rare application that goes further than three).

That feature is the ability to include an entire array as a part of another one, and to be able to keep on doing so, just like the old rhyme: “Big fleas have little fleas upon their backs to bite ’em. Little fleas have lesser fleas, add flea, ad infinitum.” Let’s look at how this works by taking the associative array in the previous example and extending it—see Example 6-10.

Example 6-10. Creating a multidimensional associative array

```
<?php

$products =
array(
    'paper'
=>      array(
        'copier'  "Copier &
=>      Multipurpose",
        'inkjet'  "Inkjet Printer",
        'laser' =>"Laser Printer",
        'photo    "Photographic
        '          =>Paper"),
    'pens' =>
array(
        'ball'   =>"Ball Point",
        'hilite' =>"Highlighters",
        'marker'
=>      "Markers"),
    'misc' =>
array(
```

```

        'tape' =>"Sticky Tape",
        'glue' =>"Adhesives",
        'clips' =>"Paperclips") );

echo "<pre>";

foreach ($products as $section => $items)

    foreach ($items as $key => $value)

        echo "$section:\t$key\t($value)<br>";

echo "</pre>";

?>

```

To make things clearer now that the code is starting to grow, I've renamed some of the elements. For example, seeing as the previous array `$paper` is now just a subsection of a larger array, the main array is now called `$products`. Within this array there are three items, paper, pens, and misc, and each of these contains another array with key/value pairs.

If necessary, these subarrays could have contained even further arrays. For example, under ball there might be an array containing all the different types and colors of ballpoint pens available in the online store. But for now, I've restricted the code to just a depth of two.

Once the array data has been assigned, I use a pair of nested `foreach...as` loops to print out the various values. The outer loop extracts the main sections from the top level of the array, and the inner loop extracts the key/value pairs for the categories within each section.

As long as you remember that each level of the array works the same way (it's a key/ value pair), you can easily write code to access any element at any level.

The echo statement makes use of the PHP escape character `\t`, which outputs a tab. Although tabs are not normally significant to the web browser, I let them be used for layout by using the `<pre>...</pre>` tags, which tell the web browser to format the text as preformatted and monospaced, and *not* to ignore whitespace characters such as tabs and line feeds. The output from this code looks like the following:

paper:	copier	(Copier & Multipurpose)
paper:	inkjet	(Inkjet Printer)
paper:	laser	(Laser Printer)
paper:	photo	(Photographic Paper)
pens:	ball	(Ball Point)
pens:	hilite	(Highlighters)
pens:	marker	(Markers)
misc:	tape	(Sticky Tape)
misc:	glue	(Adhesives)
misc:	clips	(Paperclips)

You can directly access a particular element of the array using square brackets, like this:

```
echo $products['misc']['glue'];
```

which outputs the value “Adhesives”.

You can also create numeric multidimensional arrays that are accessed directly by in-dexes rather than by alphanumeric identifiers. Example 6-11 creates the board for a chess game with the pieces in their starting positions.

Example 6-11. Creating a multidimensional numeric array

```
<?php

$chessboard = array(
    array('r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'),
    array('p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'),
    array(
        ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(
        ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(
        ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(
        ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(
        ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array('P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'),
    array('R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'));

echo "<pre>";

foreach ($chessboard as $row)
```

```

{

    foreach ($row as $piece)

        echo "$piece ";

    echo "<br />";

}

echo "</pre>";

?>

```

In this example, the lowercase letters represent black pieces and the uppercase white. The key is *r=rook*, *n=knight*, *b=bishop*, *k=king*, *q=queen*, and *p=pawn*. Again, a pair of nested `foreach...as` loops walk through the array and display its contents. The outer loop processes each row into the variable `$row`, which itself is an array, because the `$chessboard` array uses a subarray for each row. This loop has two statements within it, so curly braces enclose them.

The inner loop then processes each square in a row, outputting the character (`$piece`) stored in it, followed by a space (to square up the `printout`). This loop has a single statement, so curly braces are not required to enclose it. The `<pre>` and `</pre>` tags ensure that the output displays correctly, like this:

```

r n b q k b n r

p p p p p p p p

```

P P P P P P P P

R N B Q K B N R

You can also directly access any element within this array using square brackets, like this:

```
echo $chessboard[7][3];
```

This statement outputs the uppercase letter Q, the eighth element down and the fifth along (remembering that array indexes start at 0, not 1).

Using Array Functions

You’ve already seen the list and each functions, but PHP comes with numerous other functions for handling arrays. The full list is at <http://tinyurl.com/phparrayfuncs>. However, some of these functions are so fundamental that it’s worth taking the time to look at them here.

is_array

Arrays and variables share the same namespace. This means that you cannot have a string variable called \$fred and an array also called \$fred. If you’re in doubt and your code needs to check whether a variable is an array, you can use the `is_array` function like this:

```
echo (is_array($fred)) ? "Is an array" : "Is not an array";
```

Note that if \$fred has not yet been assigned a value, an “Undefined variable” message will be generated.

count

Although the `each` function and the `foreach...as` loop structure are excellent ways to walk through an array’s contents, sometimes you need to

know exactly how many elements there are in your array, particularly if you will be referencing them directly.

To count all the elements in the top level of an array, use a command such as the following:

```
echo count($fred);
```

Should you wish to know how many elements there are altogether in a multidimensional array, you can use a statement such as:

```
echo count($fred, 1);
```

The second parameter is optional and sets the mode to use. It should be either 0 to limit counting to only the top level, or 1 to force recursive counting of all subarray elements too.

sort

Sorting is so common that PHP provides a built-in function for this purpose. In its simplest form, you would use it like this:

```
sort($fred);
```

Unlike some other functions, `sort` will act directly on the supplied array rather than returning a new array of sorted elements. It returns `TRUE` on success and `FALSE` on error and also supports a few flags. The main two methods that you might wish to use force sorting either numerically or as strings, like this:

```
sort($fred, SORT_NUMERIC);
```

```
sort($fred, SORT_STRING);
```

You can also sort an array in reverse order using the `rsort` function, like this:

```
rsort($fred, SORT_NUMERIC);
```

```
rsort($fred, SORT_STRING);
```

shuffle

There may be times when you need the elements of an array to be put in random order, such as when creating a game of playing cards:

```
shuffle($cards);
```

Like sort, shuffle acts directly on the supplied array and returns TRUE on success or FALSE on error.

explode

explode is a very useful function that allows you to take a string containing several items separated by a single character (or string of characters) and place each of these items into an array. One handy example is to split up a sentence into an array containing all its words, as in Example 6-12. *Example 6-12. Exploding a string into an array using spaces*

```
<?php
```

```
$temp = explode(' ', "This is a sentence with seven words");
```

```
print_r($temp);
```

```
?>
```

This example prints out the following (on a single line when viewed in a browser):

```
Array(
```

```
    [0] => This
```

```
[1] => is
[2] => a
[3] => sentence
[4] => with
[5] => seven
[6] => words
)
```

The first parameter, the delimiter, need not be a space or even a single character.

Example 6-13 shows a slight variation.

*Example 6-13. Exploding a string delimited with *** into an array*

```
<?php
```

```
$temp = explode('***', "A***sentence***with***asterisks");
```

```
print_r($temp);
```

?>The code in Example 6-13 prints out the following:

```
Array(
    [0] => A
    [1] => sentence
    [2] => with
    [3] => asterisks
)
```

Sometimes it can be convenient to turn the key/value pairs from an array into PHP variables. One such time might be when processing the `$_GET` or `$_POST` variables sent to a PHP script by a form. When a form is

submitted over the Web, the web server unpacks the variables into a global array for the PHP script. If the variables were sent using the GET method, they will be placed in an associative array called `$_GET`, and if they were sent using POST, they will be placed in an associative array called `$_POST`.

You could, of course, walk through such associative arrays in the manner shown in the examples so far. However, sometimes you just want to store the values sent into variables for later use. In this case, you can have PHP do the job automatically for you: `extract($_GET);`

So, for example, if the query string parameter `q` is sent to a PHP script along with the associated value “Hi there”, a new variable called `$q` will be created and assigned that value.

Be careful with this approach, though, because if any extracted variables conflict with ones that you have already defined, your existing values will be overwritten. To avoid this possibility, you can use one of the many additional parameters available to this function, like this:

```
extract($_GET, EXTR_PREFIX_ALL, 'fromget');
```

In this case, all the new variables will begin with the given prefix string followed by an underscore, so `$q` will become `$fromget_q`. I strongly recommend that you use this version of the function when handling `$_GET` and `$_POST` arrays, or any other array whose keys could be controlled by the user, because malicious users could submit keys chosen deliberately to overwrite commonly used variable names and compromise your website.

reset

When the `foreach...as` construct or the `each` function walks through an array, it keeps an internal PHP pointer that makes a note of which element of the array it should return next. If your code ever needs to return to the start of an array, you can issue `reset`, which also returns the value of the first element. Examples of how to use this function are:

```
reset($fred);    // Throw away return value
```

```
$item = reset($fred); // Keep first element of the array in $item
```

end

Similarly, you can move PHP's internal array pointer to the final element in an array using the end function, which also returns the value of that element and can be used as in these examples:

```
end($fred);
```

```
$item = end($fred);
```

This chapter concludes your basic introduction to PHP, and you should now be able to write quite complex programs using the skills you have learned. In the next chapter, we'll look at using PHP for common, practical tasks.

Unit-3

Strings and File Handling

PHP has built-in String manipulation functions, supporting operations ranging from string repetition and reversal to comparison and search-and-replace.

Some of these important functions are

Sr	Function	What it Does
1	<code>empty()</code>	Tests if a string is empty
2	<code>strlen()</code>	Calculates the number of characters in a string
3	<code>strrev()</code>	Retrun reverse of a given string
4	<code>str_repeat()</code>	Repeats a string no. of times you want
5	<code>substr()</code>	Retrieves a section of a string
6	<code>strcmp()</code>	Compares two strings
7	<code>str_word_count()</code>	Calculates the number of words in a string
8	<code>str_replace()</code>	Replaces parts of a string
9	<code>trim()</code>	removes leading and trailing whitespaces from a string
10	<code>strtolower()</code>	Converts in Lowercases a string

11	strtoupper()	Converts in Uppercases a string
12	ucfirst()	Converts in uppercase the first character of a string
13	ucwords()	Converts in uppercases the first character of every word of a string
14	addslashes()	Escapes special characters in a string with backslashes
15	stripslashes()	Removes backslashes from a string
16	htmlentities()	Encodes HTML within a string
17	htmlspecialchars() ()	Encodes special HTML characters within a string
18	nl2br()	Replaces line breaks in a string with elements
19	html_entity_decode() ode()	Decodes HTML entities within a string
20	htmlspecialchars_decode() _decode()	Decodes special HTML characters within a string
21	strip_tags()	Removes PHP and HTML code from a string
22	md5()	The MD5 message-digest algorithm is a widely used for cryptography
23	str_split()	str_split() function splits(break) a string into an array.

1. The PHP `strlen()` function returns the length of a string.

The example below returns the length of the string "Hello world!":

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

```
<body>

<?php
echo strlen("Hello world!");
?>

</body>
</html>
```

Output 12

Count The Number of Words in a String

2. The PHP `str_word_count()` function counts the number of words in a string:

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

<?php
echo str_word_count("Hello world!");
?>

</body>
</html>
```

The output of the code above will be: 2.

3. Reverse a String

The PHP `strrev()` function reverses a string:

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

<?php
echo strrev("Hello world!");
?>
```

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

The output of the code above will be: !dlrow olleH.

4. Replace Text Within a String

The PHP `str_replace()` function replaces some characters with some other characters in a string.

The example below replaces the text "world" with "Dolly":

```
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
echo str_replace("world", "Dolly", "Hello world!");  
?>  
  
</body>  
</html>
```

Form Handling

The main way that website users interact with PHP and MySQL is through the use of HTML forms. These were introduced very early on in the development of the World Wide Web—in 1993, even before the advent of ecommerce—and have remained a mainstay ever since, due to their simplicity and ease of use.

Of course, enhancements have been made over the years to add extra functionality to HTML form handling; this chapter will bring you up to speed on the state of the art and show you the best ways to implement forms for good usability and security.

Building Forms

Handling forms is a multipart process. First a form is created, into which a user can enter the required details. This data is then sent to the web server,

where it is interpreted, often with some error checking. If the PHP code identifies one or more fields that require reentering, the form may be redisplayed with an error message. When the code is satisfied with the accuracy of the input, it takes some action that usually involves the database, such as entering details about a purchase.

To build a form, you must have at least the following elements:

- An opening `<form>` and closing `</form>` tag
- A submission type specifying either a GET or POST method
- One or more input fields
- The destination URL to which the form data is to be submitted

Example 11-1 shows a very simple form created using PHP. Type it in and save it as *formtest.php*.

Example 11-1. formtest.php—a simple PHP form handler

```
<?php // formtest.php

echo <<<_END

<html>


---


  <head>
    <title>Form Test</title>
  </head>
  <body>
    <form method="post" action="formtest.php">
      What is your name?
      <input type="text" name="name" />
```

```
<input type="submit" />

</form>

</body>

</html>

_END;

?>
```

The first thing to notice about this example is that, as you have already seen in this book, rather than dropping in and out of PHP code, I generally use the echo <<<_END..._END heredoc construct when multiline HTML must be output.

Inside this multiline output is some standard code for commencing an HTML document, displaying its title, and starting the body of the document. This is followed by the form, which is set to send its data using the POST method to the PHP program *formtest.php*, which is the name of the program itself.

The rest of the program just closes all the items it opened: the form, the body of the HTML document, and the PHP echo <<<_END statement. The result of opening this program in a web browser can be seen in Figure 11-1.

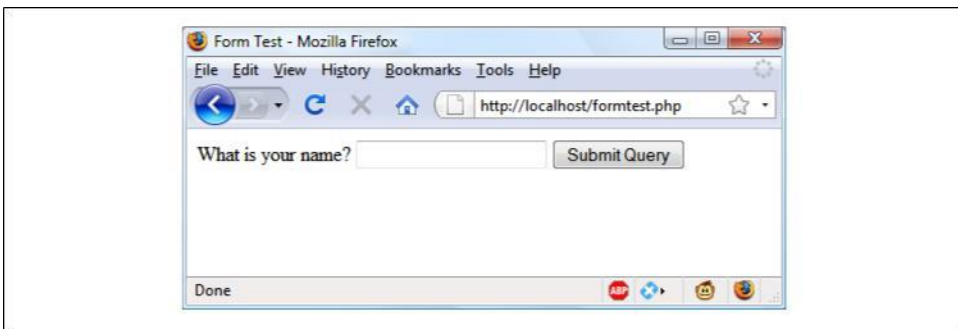


Figure 11-1. The result of opening formtest.php in a web browser

Retrieving Submitted Data

Example 11-1 is only one part of the multipart form handling process. If you enter a name and click on the Submit Query button, absolutely nothing will happen other than the form being redisplayed. So, now it's time to add some PHP code to process the data submitted by the form.

Example 11-2 expands on the previous program to include data processing. Type it in (or modify *formtest.php* by adding in the new lines), save it as *formtest2.php*, and try the program for yourself. The result of running this program and entering a name can be seen in Figure 11-2.

Example 11-2. Updated version of formtest.php

```
<?php // formtest2.php

if (isset($_POST['name'])) $name =
$_POST['name']; else $name = "(Not
entered)";

echo <<<_END

<html>

    <head><title>Form Test</title></head>

    <body>

        Your name is: $name<br />

        <form method="post" action="formtest2.php">

            What is your name?

            <input type="text" name="name" />

            <input type="submit" />

        </form>

    </body>

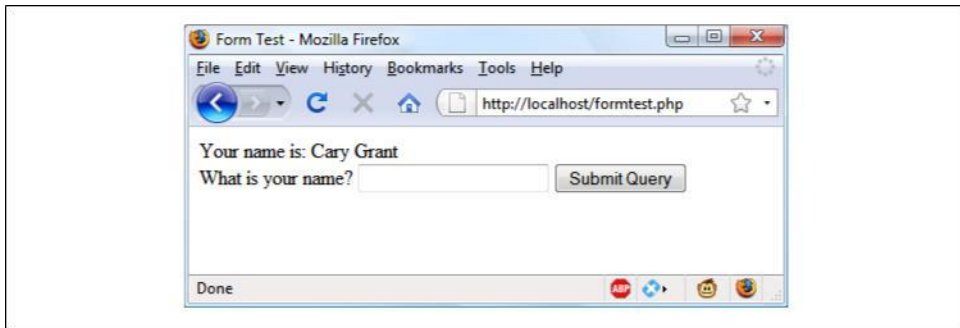
</html>
```

_END;

?>

The only changes are a couple of lines at the start that check the `$_POST` associative array for the field name submitted. The previous chapter introduced the `$_POST` associative array, which contains an element for each field in an HTML form. In Example 11-2, the input name used was `name` and the form method was `POST`, so element name of the `$_POST` array contains the value in `$_POST['name']`.

The PHP `isset` function is used to test whether `$_POST['name']` has been assigned a value. If nothing was posted, the program assigns the value “(Not entered)”; otherwise, it stores the value that was entered. Then a single line has been added after the `<body>` statement to display that value, which is stored in `$name`.



Input Types

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

Text boxes

Probably the type of input you will use most often is the text box. It accepts a wide range of alphanumeric text and other characters in a single-line box. The general format of a text box input is:

```
<input type="text" name="name" size="size" maxlength="length"  
value="value" />
```

We've already covered the name and value parameters, but two more are introduced here: size and maxlength. The size parameter specifies the width of the box, in characters of the current font, as it should appear on the screen, and maxlength specifies the maximum number of characters that a user is allowed to enter into the field.

The only required parameters are type, which tells the web browser what type of input is to be expected, and name, for providing a name to the input that is then used to process the field upon receipt of the submitted form.

Text areas

When you need to accept input of more than a single line of text, use a text area. This is similar to a text box but, because it allows multiple lines, it has some different pa-rameters. Its general format looks like this:

```
<textarea name="name" cols="width"  
rows="height" wrap="type"> </textarea>
```

The first thing to notice is that <textarea> has its own tag and is not a subtype of the <input> tag. It therefore requires a closing </textarea> to end input.

Instead of a default parameter, if you have default text to display, you must put it before the closing </textarea>, like this:

```
<textarea name="name" cols="width" rows="height" wrap="type">
```

This is some default text.

```
</textarea>
```

It will then be displayed and be editable by the user. To control the width and height, use the cols and rows parameters. Both use the character spacing of the current font to determine the size of the area. If you omit these values, a default input box will be created that will vary in dimensions depending on the browser used, so you should always define them to be certain about how your form will appear.

Lastly, you can control how the text entered into the box will wrap (and how any such wrapping will be sent to the server) using the wrap parameter. Table 11-1 shows the wrap types available. If you leave out the wrap parameter, soft wrapping is used. *Table 11-1. The wrap types available in a <textarea> input*

TypeAction

off Text does not wrap and lines appear exactly as the user types them.

soft Text wraps but is sent to the server as one long string without carriage returns and line feeds.

hard Text wraps and is sent to the server in wrapped format with soft returns and line feeds.

Checkboxes

When you want to offer a number of different options to a user, from which he can select one or more items, checkboxes are the way to go. The format to use is: `<input type="checkbox" name="name" value="value" checked="checked" />`

If you include the checked parameter, the box is already checked when the browser is displayed (the string you assign to the parameter doesn't matter; the parameter just has to be present). If you don't include the parameter, the box is shown unchecked. Here is an example of an unchecked box:

I Agree `<input type="checkbox" name="agree" />`

By default, checkboxes are square.

Radio buttons

Radio buttons are named after the push-in preset buttons found on many older radios, where any previously depressed button pops back up when another is pressed. They are used when you want only a single value to be returned from a selection of two or more options. All the buttons in a group

must use the same name and, because only a single value is returned, you do not have to pass an array.

For example, if your website offers a choice of delivery times for items purchased from your store, you might use HTML like that in Example 11-6 (see Figure 11-5 to see how it displays).

```
8am-Noon<input type="radio" name="time" value="1"
/>| Noon-4pm<input type="radio" name="time"
value="2" checked="checked"/>|
4pm-8pm<input type="radio" name="time" value="3" />
```

Hidden fields can also be useful for storing other details, such as a session ID string that you might create to identify a user, and so on. Never treat hidden fields as secure, because they are not. The HTML containing them can easily be viewed using a browser's view source feature.

Select

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

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

The parameter size is the number of lines to display. Clicking on the display causes a list to drop down showing all the options. If you use the multiple parameter, the user can select multiple options from the list by pressing the Ctrl key when clicking. So, to ask a user for his favorite vegetable from a choice of five, you might use HTML like that in Example 11-7, which offers a single selection.

Example 11-7. Using select

```
Vegetables <select name="veg" size="1">
```

```
<option value="Peas">Peas</option>

<option value="Beans">Beans</option>

<option value="Carrots">Carrots</option>

<option value="Cabbage">Cabbage</option>

<option value="Broccoli">Broccoli</option>

</select>
```

This HTML offers five choices, with the first one, *Peas*, preselected (due to it being the first item). Figure 11-6 shows the output where the list has been clicked on to drop it down, and the option *Carrots* has been highlighted.

If you want to have a different default option offered first (such as *Beans*), use the `selected` tag, like this:

```
<option selected="selected" value="Beans">Beans</option>
```

You can also allow for users to select more than one item, as in Example 11-8.

Example 11-8. Using select with the multiple parameter

```
Vegetables <select name="veg" size="5"
multiple="multiple"> <option
```

```
value="Peas">Peas</option> <option  
value="Beans">Beans</option>
```

```
<option value="Carrots">Carrots</option>
```

```
<option value="Cabbage">Cabbage</option>
```

```
<option value="Broccoli">Broccoli</option>
```

```
</select>
```

This HTML is not very different; the only changes are that the size has been changed to "5" and the parameter multiple has been added. But, as you can see from Figure 11-7, it is now possible to select more than one option by using the Ctrl key when clicking.

The text will not be underlined like a hyperlink when you do this, but as the mouse pointer passes over it it will change to an arrow instead of a text cursor, indicating that the whole item is clickable.

The submit button

To match the type of form being submitted, you can change the text of the submit button to anything you like by using the value parameter, like this:

```
<input type="submit" value="Search" />
```

You can also replace the standard text button with a graphic image of your choice, using HTML such as this:

```
<input type="image" name="submit" src="image.gif" />
```

Checking Whether a File Exists

To determine whether a file already exists, you can use the `file_exists` function, which returns either `TRUE` or `FALSE` and is used like this:

```
if (file_exists("testfile.txt")) echo "File exists";
```

File Handling

Creating a File

At this point *testfile.txt* doesn't exist, so let's create it and write a few lines to it. Type in Example 7-4 and save it as *testfile.php*.

Example 7-4. Creating a simple text file

```
<?php // testfile.php
```

```
$fh = fopen("testfile.txt", 'w') or die("Failed to create file");
```

```
$text = <<<_END
```

```
Line 1
```

```
Line 2
```

```
Line 3
```

```
_END;
```

```
fwrite($fh, $text) or die("Could not write to  
file"); fclose($fh);
```

```
echo "File 'testfile.txt' written  
successfully"; ?>
```

When you run this in a browser, all being well, you will receive the message “File ‘testfile.txt’ written successfully”. If you receive an error message, your hard disk may be full or, more likely, you may not have permission to create or write to the file, in which case you should modify the attributes of the destination folder according to your operating system. Otherwise, the file *testfile.txt* should now be residing in the same folder in which you saved the *testfile.php* program. Try opening the file in a text or program editor—the contents will look like this:

Line 1

Line 2

Line 3

This simple example shows the sequence that all file handling takes:

1. Always start by opening the file. This is done through a call to `fopen`.
2. Then you can call other functions; here we write to the file (`fwrite`), but you can also read from an existing file (`fread` or `fgets`) and do other things.
3. Finish by closing the file (`fclose`). Although the program does this for you when it ends, you should clean up yourself by closing the file when you’re finished.

Every open file requires a file resource so that PHP can access and manage it. The preceding example sets the variable `$fh` (which I chose to stand for *file handle*) to the value returned by the `fopen` function. Thereafter, each file handling function that ac-cesses the opened file, such

as `fwrite` or `fclose`, must be passed `$fh` as a parameter to identify the file being accessed. Don't worry about the content of the `$fh` variable; it's a number PHP uses to refer to internal information about the file—you just pass the variable to other functions.

Upon failure, `fopen` will return `FALSE`. The previous example shows a simple way to capture and respond to the failure: it calls the `die` function to end the program and give the user an error message. A web application would never abort in this crude way (you would create a web page with an error message instead), but this is fine for our testing purposes.

Notice the second parameter to the `fopen` call. It is simply the character `w`, which tells the function to open the file for writing. The function creates the file if it doesn't already exist. Be careful when playing around with these functions: if the file already exists, the `w` mode parameter causes the `fopen` call to delete the old contents (even if you don't write anything new!).

There are several different mode parameters that can be used here, as detailed in Table 7-5.

Table 7-5. The supported fopen modes

ModeAction	
'w'	Write from file start and truncate file
'r'	Read from file start
'w+'	Write from file start, truncate file, and allow read-ing
'r+'	Read from file start and allow writing

Description

the file pointer at the beginning of the file.
Return FALSE if the file doesn't already exist.

Open for reading only; place the file pointer at the beginning of the file. Return FALSE if the file doesn't already exist.

Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file doesn't exist, attempt to create it.

Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file doesn't exist, attempt to create it.

Open for reading and writing; place

Mode	Action	Description
'a'	Append to file end	Open for writing only; place the file pointer at the end of the file. If the file doesn't exist, attempt to create it.
'a+'	Append to file end and allow reading	Open for reading and writing; place the file pointer at the end of the file. If the file doesn't exist, attempt to create it.

Reading from Files

The easiest way to read from a text file is to grab a whole line through `fgets` (think of the final `s` as standing for “string”), as in Example 7-5.

Example 7-5. Reading a file with `fgets`

```
<?php
```

```
$fh = fopen("testfile.txt", 'r') or
```

```
    die("File does not exist or you lack permission to open it");
```

```
$line = fgets($fh);
```

```
fclose($fh);
```

```
echo $line;
```

```
?>
```

If you created the file as shown in Example 7-4, you'll get the first line:

Line 1

Or you can retrieve multiple lines or portions of lines through the `fread` function, as in Example 7-6.

Example 7-6. Reading a file with `fread`

```
<?php
```

```
$fh = fopen("testfile.txt", 'r') or
```

```
    die("File does not exist or you lack permission to open it");
```

```
$text = fread($fh, 3);
```

```
fclose($fh);
```

```
echo $text;
```

```
?>
```

I've requested three characters in the `fread` call, so the program displays the following:

Lin

The fread function is commonly used with binary data, but if you use it on text data that spans more than one line, remember to count newline characters.

Copying Files

Let's try out the PHP copy function to create a clone of *testfile.txt*. Type in Example 7-7 and save it as *copyfile.php*, then call up the program in your browser. Example 7-7. Copying a file

```
<?php // copyfile.php

copy('testfile.txt', 'testfile2.txt') or die("Could not copy
file"); echo "File successfully copied to 'testfile2.txt'";
?>
```

If you check your folder again, you'll see that you now have the new file *testfile2.txt* in it. By the way, if you don't want your programs to exit on a failed copy attempt, you could try the alternate syntax in Example 7-8.

Example 7-8. Alternate syntax for copying a file

```
<?php // copyfile2.php

if (!copy('testfile.txt', 'testfile2.txt')) echo "Could not copy
file"; else echo "File successfully copied to 'testfile2.txt'";
?>
```

Moving a File

To move a file, rename it with the rename function, as in Example 7-9.

Example 7-9. Moving a file

```
<?php // movefile.php

if (!rename('testfile2.txt', 'testfile2.new'))

    echo "Could not rename file";

else echo "File successfully renamed to
'testfile2.new'"; ?>
```

You can use the rename function on directories, too. To avoid any warning messages if the original file or directory doesn't exist, you can call the file_exists function first to check.

Deleting a File

Deleting a file is just a matter of using the unlink function to remove it from the file-system, as in Example 7-10.

Example 7-10. Deleting a file

```
<?php // deletefile.php

if (!unlink('testfile2.new')) echo "Could not
delete file"; else echo "File 'testfile2.new'
successfully deleted"; ?>
```

Whenever you access files on your hard disk directly, you must also always ensure that it is impossible for your filesystem to be compromised. For example, if you are deleting a file based on user input, you must make absolutely certain that it is a file that can be safely deleted and that the user is allowed to delete it.

As with moving a file, a warning message will be displayed if the file doesn't exist; you can avoid this by using file_exists to first check for its existence before calling unlink.

Updating Files

Often you will want to add more data to a saved file, which you can do in many ways. You can use one of the append write modes (see Table 7-5), or you can simply open a file for reading and writing with one of the other modes that supports writing, and move the file pointer to the place within the file that you wish to write to.

The *file pointer* is the position within a file at which the next file access will take place, whether it's a read or a write. It is not the same as the *file handle* (as stored in the variable `$fh` in Example 7-4), which contains details about the file being accessed.

You can see this in action by typing in Example 7-11, saving it as *update.php*, then calling it up in your browser.

Example 7-11. Updating a file

```
<?php // update.php

$fh = fopen("testfile.txt", 'r+') or die("Failed to open file");

$text = fgets($fh);

fseek($fh, 0, SEEK_END);

fwrite($fh, "$text") or die("Could not write to
file"); fclose($fh);

echo "File 'testfile.txt' successfully
updated"; ?>
```

This program opens *testfile.txt* for both reading and writing by setting the mode with 'r+', which puts the file pointer right at the start of the file. It then uses the `fgets` function to read in a single line from the file (up to the first line feed). After that, the `fseek` function is called to move the file pointer right to the file end, at which point the line of text that was extracted from the start of the file (stored in `$text`) is then appended to file's end and the file is closed. The resulting file now looks like this:

Line 1

Line 2

Line 3

Line 1

The first line has successfully been copied and then appended to the file's end.

As used here, in addition to the `$fh` file handle, the `fseek` function was passed two other parameters, 0 and `SEEK_END`. The `SEEK_END` tells the function to move the file pointer to the end of the file, and the 0 parameter tells it how many positions it should then be moved backwards from that point. In the case of Example 7-11, a value of 0 is used because the pointer is required to remain at the file's end.

There are two other seek options available to the `fseek` function: `SEEK_SET` and `SEEK_CUR`. The `SEEK_SET` option tells the function to set the file pointer to the exact position given by the preceding parameter. Thus, the following example moves the file pointer to position 18:

```
fseek($fh, 18, SEEK_SET);
```


SEEK_CUR sets the file pointer to the current position *plus* the value of the given offset. Therefore, if the file pointer is currently at position 18, the following call will move it to position 23:

```
fseek($fh, 5, SEEK_CUR);
```

Although this is not recommended unless you have very specific reasons for it, it is even possible to use text files such as this (but with fixed line lengths) as simple flat file databases. Your program can then use `fseek` to move back and forth within such a file to retrieve, update, and add new records. Records can also be deleted by overwriting them with zero characters, and so on.

Locking Files for Multiple Accesses

Web programs are often called by many users at the same time. If more than one person tries to write to a file simultaneously, it can become corrupted. And if one person writes to it while another is reading from it, the file will be all right but the person reading it may get odd results. To handle simultaneous users, it's necessary to use the file locking function `flock`. This function queues up all other requests to access a file until your program releases the lock. Whenever your programs use write access on files that may be accessed concurrently by multiple users, you should add file locking to them, as in Example 7-12, which is an updated version of Example 7-11.

Example 7-12. Updating a file with file locking

```
<?php
```

```
$fh = fopen("testfile.txt", 'r+') or die("Failed to open  
file"); $text = fgets($fh);
```

```

if (flock($fh, LOCK_EX))

{

    fseek($fh, 0, SEEK_END);

    fwrite($fh, "$text") or die("Could not write to
    file"); flock($fh, LOCK_UN);

}

fclose($fh);

echo "File 'testfile.txt' successfully
updated"; ?>

```

There is a trick to file locking to preserve the best possible response time ~~for your website visitors: perform it directly before a change you make to~~ a file, and then unlock it immediately afterwards. Having a file locked for any longer than this will slow down your application unnecessarily. This is why the calls to flock in Example 7-12 are di-rectly before and after the fwrite call.

The first call to flock sets an exclusive file lock on the file referred to by \$fh using the

LOCK_EX parameter:

```
flock($fh, LOCK_EX);
```

From this point onwards, no other processes can write to (or even read from) the file until the lock is released by using the LOCK_UN parameter, like this:

```
flock($fh, LOCK_UN);
```

As soon as the lock is released, other processes are again allowed access to the file. This is one reason why you should reseek to the point you wish to access in a file each time you need to read or write data: another process could have changed the file since the last access.

However, did you notice that the call to request an exclusive lock is nested as part of an if statement? This is because flock is not supported on all systems, and therefore it is wise to check whether you successfully secured a lock before you make your changes, just in case one could not be obtained.

Something else you must consider is that flock is what is known as an *advisory* lock. This means that it locks out only other processes that call the function. If you have any code that goes right in and modifies files without implementing flock file locking, it will always override the locking and could wreak havoc on your files.

Implementing file locking and then accidentally leaving it out in one section of code can lead to an extremely hard-to-locate bug.

flock will not work on NFS and many other networked filesystems. Also, when using a multithreaded server like ISAPI, you may not be able to rely on flock to protect files against other PHP scripts running in parallel threads of the same server instance. Additionally, flock is not supported on any system using the old FAT filesystem, such as older versions of Windows.

Reading an Entire File

A handy function for reading in an entire file without having to use file handles is `file_get_contents`. It's very easy to use, as you can see in Example 7-13.

Example 7-13. Using file_get_contents

```
<?php  
  
echo "<pre>"; // Enables display of  
line feeds echo  
file_get_contents("testfile.txt"); echo  
"</pre>"; // Terminates pre tag  
  
?>
```

But the function is actually a lot more useful than that—you can also use it to fetch a file from a server across the Internet, as in Example 7-14, which requests the HTML from the O'Reilly home page and then displays it as if the page itself had been surfed to. The result will be similar to the screen grab in Figure 7-1.

```
<?php  
  
echo file_get_contents("http://oreilly.com");  
  
?>
```

Uploading Files

Uploading files to a web server is a subject area that seems daunting to many people, but it actually couldn't be much easier. All you need to do to upload a file from a form is choose a special type of encoding called *multipart/form-data*; your browser will handle the rest. To see how this works, type in the program in Example 7-15 and save it as *upload.php*. When you run it, you'll see a form in your browser that lets you upload a file of your choice.

Example 7-15. Image uploader (upload.php)

```
<?php // upload.php  
  
echo <<<_END
```

```

<html><head><title>PHP Form Upload</title></head><body>

<form method='post' action='upload.php' enctype='multipart/form-data'>
  elect File: <input type='file' name='filename'
  size='10' /> <input type='submit' value='Upload'
  /> </form>

  _END;

if ($_FILES)

{

  $name = $_FILES['filename']['name'];

  move_uploaded_file($_FILES['filename']['tmp_
  name'], $name); echo "Uploaded image
  '$name'<br /><img src='$name' />";

}

echo "</body></html>";

?>

```

Let's examine this program a section at a time. The first line of the multiline echo statement starts an HTML document, displays the title, and then starts the document's body.

Next we come to the form that selects the POST method of form submission, sets the target for posted data to the program *upload.php* (the

program itself), and tells the web browser that the data posted should be encoded using the content type *multipart/form-data*.

With the form set up, the next lines display the prompt “Select File:” and then request two inputs. The first input being asked for is for a file, which is done by using an input type of *file* and a name of *filename*. This input field has a width of 10 characters.

The second requested input is just a Submit button that is given the label “Upload” (replacing the default button text of “Submit Query”). And then the form is closed.

This short program shows a common technique in web programming in which a single program is called twice: once when the user first visits a page, and again when the user presses the Submit button.

The PHP code to receive the uploaded data is fairly simple, because all uploaded files are placed into the associative system array `$_FILES`. Therefore, a quick check to see whether `$_FILES` has anything in it is sufficient to determine whether the user has up-loaded a file. This is done with the statement `if ($_FILES)`.

The first time the user visits the page, before uploading a file, `$_FILES` is empty, so the program skips this block of code. When the user uploads a file, the program runs again and discovers an element in the `$_FILES` array.

Once the program realizes that a file was uploaded, the actual name, as read from the uploading computer, is retrieved and placed into the variable `$name`. Now all that’s necessary is to move the uploaded file from the temporary location in which PHP stored it to a more permanent one. This is done using the `move_uploaded_file` function, passing it the original name of the file, with which it is saved to the current directory. Finally, the

uploaded image is displayed within an tag, and the result should look like the screen grab in Figure 7-2.

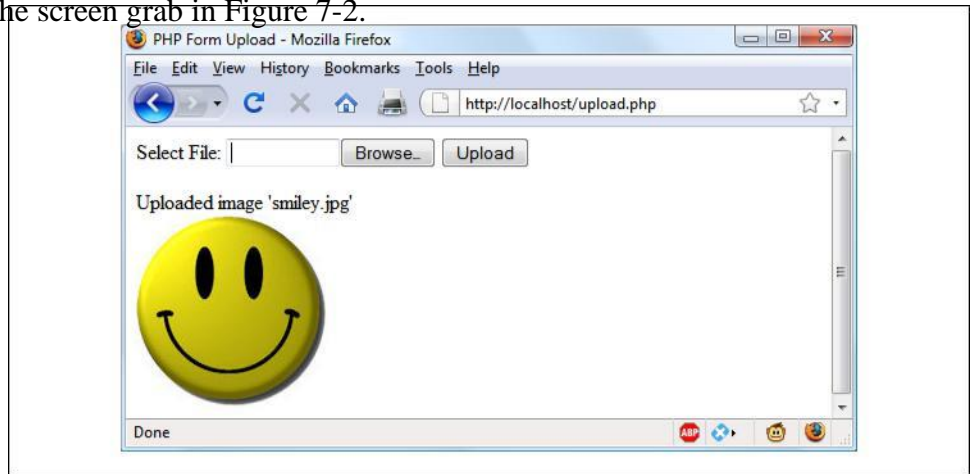


Figure 7-2. Uploading an image as form data

If you run this program and receive warning messages such as “Permis-sion denied” for the `move_uploaded_file` function call, you may not have the correct permissions set for the folder in which the program is running.

UNIT-4

Cookies, Sessions, and Authentication

As your web projects grow larger and more complicated, you will find an increasing need to keep track of your users. Even if you aren't offering logins and passwords, you will still often need to store details about a user's current session and possibly also recognize users when they return to your site.

Several technologies support this kind of interaction, ranging from simple browser cookies to session handling and HTTP authentication. Between them, they offer the opportunity for you to configure your site to your users' preferences and ensure a smooth and enjoyable transition through it.

Using Cookies in PHP

A cookie is an item of data that a web server saves to your computer's hard disk via a web browser. It can contain almost any alphanumeric information (as long as it's under 4 KB) and can be retrieved from your computer and returned to the server. Common uses include session tracking, maintaining data across multiple visits, holding shopping cart contents, storing login details, and more.

Because of their privacy implications, cookies can be read only from the issuing domain. In other words, if a cookie is issued by, for example, oreilly.com, it can be retrieved only by a web server using that domain. This prevents other websites from gaining access to details they are not authorized to have.

<a

Due to the way the Internet works, multiple elements on a web page can be embedded from multiple domains, each of which can issue its own cookies. These are referred to as *third-party* cookies. Most

commonly, they are created by advertising companies in order to track users across multiple websites.

Most browsers allow users to turn off cookies for either the current server's domain, third-party servers, or both. Fortunately, most people who disable cookies do so only for third-party websites. Cookies are exchanged during the transfer of headers, before the actual HTML of a web page is sent, and it is impossible to send a cookie once any HTML has been transferred. Therefore, careful planning of cookie usage is important. Figure 12-1 illustrates a typical request and response dialog between a web browser and web server passing cookies.

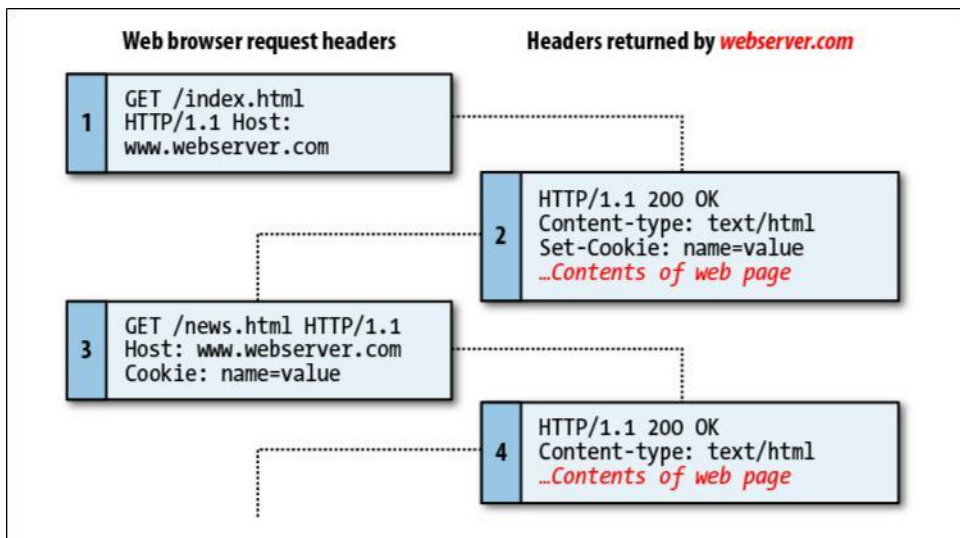


Figure 12-1. A browser/server request/response dialog with cookies

This exchange shows a browser receiving two pages:

1. The browser issues a request to retrieve the main page, *index.html*, at the website *http://www.webserver.com*. The first header specifies the file and the second header specifies the server.
2. When the web server at *webserver.com* receives this pair of headers, it returns some of its own. The second header defines the type of content to be sent (*text/html*) and the third one sends a cookie with the name

name and the value *value*. Only then are the contents of the web page transferred.

3. Once the browser has received the cookie, it will then return it with every future request made to the issuing server until the cookie expires or is deleted. So, when the browser requests the new page */news.html*, it also returns the cookie *name* with the value *value*.
4. Because the cookie has already been set, when the server receives the request to send */news.html*, it does not have to resend the cookie, but ~~just returns the requested page.~~

Setting a Cookie

Setting a cookie in PHP is a simple matter. As long as no HTML has yet been transferred, you can call the `setcookie` function, which has the following syntax (see Table 12-1):

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

Table 12-1. The setcookie parameters

Parameter	Description	Example
name	The name of the cookie. This is the name that your server will use to access the cookie on subsequent browser requests.	username
value	The value of the cookie, or the cookie's contents. This can contain up to 4 KB of alphanumeric text.	Hannah
expire	(Optional) The Unix timestamp of the cookie's expiration date. Generally, you will use <code>time()</code> plus or minus a number of seconds. If not set, the cookie expires when the browser closes.	<code>time() + 2592000</code>
path	(Optional) The path of the cookie on the server. If this is a / (forward	/

slash), the cookie is

available over the entire domain, such as `www.webserver.com`. If it is a subdirectory, the

cookie is available only within that subdirectory. The default is the current directory that

the cookie is being set in, and this is the setting you will normally use.

domain	(Optional) The Internet domain of the cookie. If this is <code>.webserver.com</code> , the cookie is available to all of <code>webserver.com</code> and its subdomains, such as <code>www.webserver.com</code> and <code>images.webserver.com</code> . If it is <code>images.webserver.com</code> , the cookie is available only to <code>images.webserver.com</code> and its subdomains, such as <code>sub.images.webserver.com</code> , but not, say, <code>www.webserver.com</code> .	<code>.webserver.com</code>
secure	(Optional) Whether the cookie must use a secure connection (<code>https://</code>). If this value is TRUE, the cookie can be transferred only across a secure connection. The default is FALSE.	FALSE
httponly	(Optional; implemented since PHP version 5.2.0) Whether the cookie must use the HTTP protocol. If this value is TRUE, scripting languages such as JavaScript cannot access the cookie. (Not supported in all browsers.) The default is FALSE.	FALSE

So, to create a cookie with the name `username` and the value “Hannah” that is accessible across the entire web server on the current domain, and

will be removed from the browser's cache in seven days, use the following:

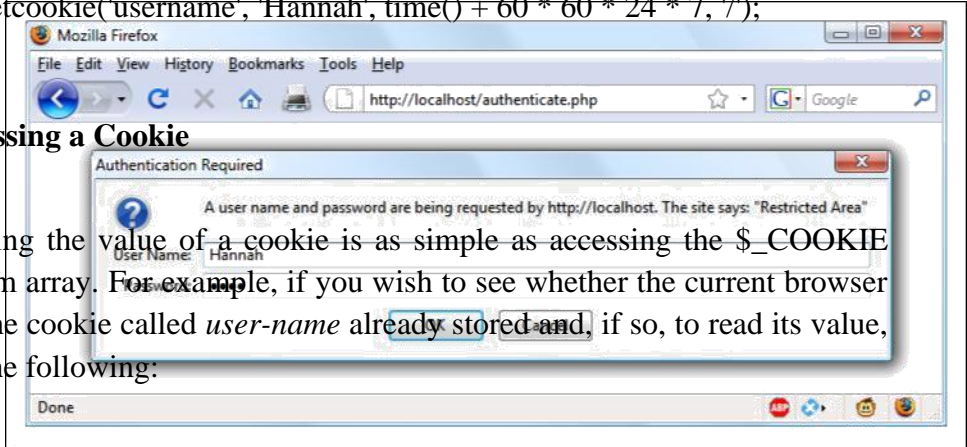
```
setcookie('username', 'Hannah', time() + 60 * 60 * 24 * 7, '/');
```

Accessing a Cookie

Reading the value of a cookie is as simple as accessing the `$_COOKIE` system array. For example, if you wish to see whether the current browser has the cookie called *user-name* already stored and, if so, to read its value, use the following:

```
if (isset($_COOKIE['username']))
```

```
    $username = $_COOKIE['username'];
```



Note that you can read a cookie back only after it has been sent to a web browser. This means that when you issue a cookie, you cannot read it in again until the browser reloads the page (or another with access to the cookie) from your website and passes the cookie back to the server in the process.

Destroying a Cookie

To delete a cookie, you must issue it again and set a date in the past. It is important for all parameters in your new `setcookie` call except the timestamp to be identical to the parameters used when the cookie was first issued; otherwise, the deletion will fail. Therefore, to delete the cookie created earlier, you would use the following

```
setcookie('username', 'Hannah', time() - 2592000, '/');
```

As long as the time given is in the past, the cookie should be deleted. However, I have used a time of 2,592,000 seconds (one month) in the past in this example, in case the client computer's date and time are not set correctly.

Unit-5

Introduction to MySQL

With well over ten million installations, MySQL is probably the most popular database management system for web servers. Developed in the mid-1990s, it's now a mature technology that powers many of today's most-visited Internet destinations.

One reason for its success must be the fact that, like PHP, it's free to use. But it's also extremely powerful and exceptionally fast—it can run on even the most basic of hard-ware, and it hardly puts a dent in system resources.

MySQL is also highly scalable, which means that it can grow with your website. In fact, in a comparison of several databases by *eWEEK*, MySQL and Oracle tied for both best performance and greatest scalability.

MySQL Basics

A database is a structured collection of records or data stored in a computer system and organized in such a way that it can be searched quickly and information can be retrieved rapidly.

The SQL in MySQL stands for Structured Query Language. This language is loosely based on English and is also used on other databases, such as Oracle and Microsoft SQL Server. It is designed to allow simple requests from a database via commands such as:

```
SELECT title FROM publications WHERE author = 'Charles Dickens';
```

A MySQL database contains one or more *tables*, each of which contains *records* or *rows*. Within these rows are various *columns* or *fields* that contain the data itself. Table 8-1 shows the contents of an example database of five publications detailing the author, title, type, and year of publication. *Table 8-1. Example of a simple database*

Author	Title	Type	Year
Mark Twain	The Adventures of Tom Sawyer	Fiction	1876
Jane Austen	Pride and Prejudice	Fiction	1811
Charles Darwin	The Origin of Species	Non-Fiction	1856
Charles Dickens	The Old Curiosity Shop	Fiction	1841
William Shakespeare	Romeo and Juliet	Play	1594

Each row in the table is the same as a row in a MySQL table, and each element within a row is the same as a MySQL field.

To uniquely identify this database, I'll refer to it as the publications database in the examples that follow. And, as you will have observed, all these publications are considered to be classics of literature, so I'll call the table within the database that holds the details classics.

Summary of Database Terms

The main terms you need to acquaint yourself with for now are:

Database

The overall container for a collection of MySQL data.

Table

A subcontainer within a database that stores the actual data.

Row

A single record within a table, which may contain several fields.

Column

The name of a field within a row.

I should note that I'm not trying to reproduce the precise terminology used in academic literature about relational databases, but just to provide simple, everyday terms to help you quickly grasp basic concepts and get started with a database.

Accessing MySQL via the Command Line

There are three main ways in which you can interact with MySQL: using a command line, via a web interface such as phpMyAdmin, and through a programming language like PHP. We'll start doing the third of these in [Chapter 10](#), but for now, let's look at the first two.

Starting the Command-Line Interface

The following sections describe relevant instructions for Windows, OS X, and Linux.

Windows users

If you installed the Zend Server CE WAMP as explained in Chapter 2, you will be able to access the MySQL executable from one of the following directories (the first on 32-bit computers, and the second on 64-bit machines):

C:\Program Files\Zend\MySQL51\bin

C:\Program Files (x86)\Zend\MySQL51\bin

If you installed Zend Server CE in a place other than
\Program Files (or *\Program Files (x86)*), you will
need to use that directory instead.

By default, the initial MySQL user will be *root* and will not have had a password set. Seeing as this is a development server that only you should be able to access, we won't worry about creating one yet.

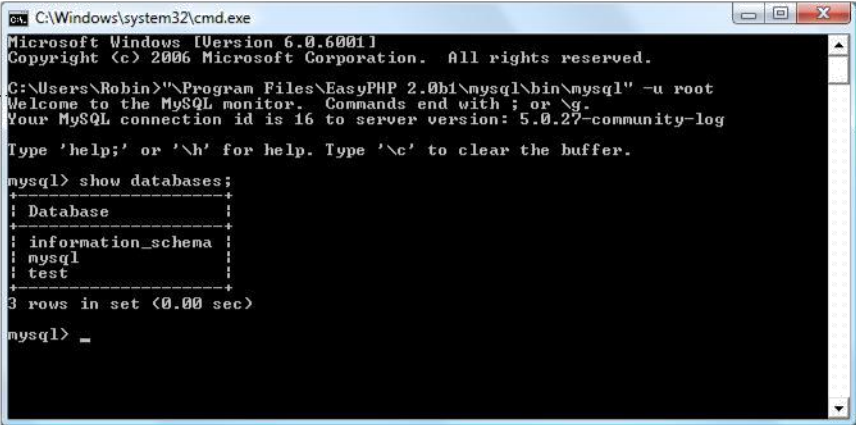
So, to enter MySQL's command-line interface, select Start→Run and enter CMD into the Run box, then press Return. This will call up a Windows Command prompt. From there, enter one of the following (making any appropriate changes as just discussed):

```
"C:\Program Files\Zend\MySQL51\bin"  
-u root "C:\Program Files  
(x86)\Zend\MySQL51\bin" -u root
```

Note the quotation marks surrounding the path and filename. These are present because the name contains spaces, which the Command prompt doesn't correctly interpret; the quotation marks group the parts of the filename into a single string for the Command program to understand.

This command tells MySQL to log you in as the user *root*, without a password. You will now be logged in to MySQL and can start entering commands. To be sure every-thing is working as it should be, enter the following—the results should be similar to Figure 8-1:

SHOW databases;



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Users\Robin>"\Program Files\EasyPHP 2.0b1\mysql\bin\mysql" -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16 to server version: 5.0.27-community-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| test       |
+-----+
3 rows in set (0.00 sec)

mysql> _
```

Figure 8-1. Accessing MySQL from a Windows Command prompt

SHOW databases;

If you receive an error such as “Can’t connect to local MySQL server through socket,” you haven’t started up the MySQL server, so make sure you followed the advice in Chapter 2 about configuring MySQL to start when OS X starts.

You should now be ready to move on to the next section, “Using the Command-Line Interface” on page 166.

MySQL Commands

You've already seen the SHOW command, which lists tables, databases, and many other items. The commands you'll use most often are listed in Table 8-3.

Table 8-3. A selection of common MySQL commands

Command	Parameter(s)	Meaning
ALTER	<i>database, table</i>	Alter <i>database</i> or <i>table</i>
BACKUP	<i>table</i>	Back up <i>table</i>
\c		Cancel input
CREATE	<i>database, table</i>	Create <i>database</i> or <i>table</i>
DELETE	Expression with <i>table</i> and <i>row</i>	Delete <i>row</i> from <i>table</i>
DESCRIBE	<i>table</i>	Describe the <i>table</i> 's columns
DROP	<i>database, table</i>	Delete <i>database</i> or <i>table</i>
EXIT (Ctrl-C)		Exit
GRANT	<i>user</i> details	Change <i>user</i> privileges
HELP (\h, \?)	<i>item</i>	Display help on <i>item</i>
INSERT	Expression with <i>data</i>	Insert <i>data</i>
LOCK	<i>table(s)</i>	Lock <i>table(s)</i>
QUIT (\q)		Same as EXIT
RENAME	<i>table</i>	Rename <i>table</i>
SHOW	Too many <i>items</i> to list	List <i>item</i> 's details
SOURCE	<i>filename</i>	Execute commands from <i>filename</i>

STATUS (\s)		Display current status
TRUNCATE	<i>table</i>	Empty <i>table</i>
UNLOCK	<i>table(s)</i>	Unlock <i>table(s)</i>
UPDATE	Expression with <i>data</i>	Update an existing record
USE	<i>database</i>	Use <i>database</i>

I'll cover most of these as we proceed, but first, you need to remember a couple of points about MySQL commands:

- SQL commands and keywords are case-insensitive. CREATE, create, and CrEaTe all mean the same thing. However, for the sake of clarity, the recommended style is to use uppercase.
 - Table names are case-insensitive on Windows, but case-sensitive on Linux and OS X. So, for portability purposes, you should always choose a case and stick to it. The recommended style is to use lowercase or mixed upper- and lowercase for table names.
-

Treated as a string with a character set

Treated as a string with a character set

Treated as a string with a character set

Treated as a string with a character set

The BLOB data type

The term BLOB stands for *Binary Large Object*, and therefore, as you would think, the BLOB data type is most useful for binary data in excess

of 65,536 bytes in size. The main other difference between the BLOB and BINARY data types is that BLOBs cannot have default values (see Table 8-9).

Table 8-9. MySQL's BLOB data types

Attributes		
Data type	Bytes used	Treated as binary data—no character set
TINYBLOB(<i>n</i>) <i>n</i> (<= 255)	Up to	Treated as binary data—no character set
BLOB(<i>n</i>)	Up to <i>n</i> (<= 65535)	Treated as binary data—no character set
MEDIUMBLOB(<i>n</i>)	Up to <i>n</i> (<= 16777215)	Treated as binary data—no character set
LOB(<i>n</i>)	Up to <i>n</i> (<= 4294967295)	Treated as binary data—no character set

Numeric data types

MySQL supports various numeric data types, from a single byte up to double-precision floating-point numbers. Although the most memory that a numeric field can use up is eight bytes, you are well advised to choose the smallest data type that will adequately handle the largest value you expect. This will help keep your databases small and quickly accessible.

Table 8-10 lists the numeric data types supported by MySQL and the ranges of values they can contain. In case you are not acquainted with the

terms, a *signed* number is one with a possible range from a minus value, through zero, to a positive value, and an *unsigned* number has a value ranging from zero to some positive number. They can both hold the same number of values—just picture a signed number as being shifted halfway to the left so that half its values are negative and half are positive. Note that floating-point values (of any precision) may only be signed.

To specify whether a data type is signed or unsigned, use the **UNSIGNED** qualifier. The following example creates a table called `tablename` with a field in it called `fieldname` of the data type **UNSIGNED INTEGER**:

```
CREATE TABLE tablename (fieldname INT UNSIGNED);
```

When creating a numeric field, you can also pass an optional number as a parameter, like this:

```
CREATE TABLE tablename (fieldname INT(4));
```

But you must remember that, unlike with **BINARY** and **CHAR** data types, this parameter does not indicate the number of bytes of storage to use. It may seem counterintuitive, but what the number actually represents is the display width of the data in the field when it is retrieved. It is commonly used with the **ZEROFILL** qualifier, like this:

```
CREATE TABLE tablename (fieldname INT(4) ZEROFILL);
```

What this does is cause any numbers with a width of less than four characters to be padded with one or more zeros, sufficient to make the display width of the field four characters long. When a field is already of the specified width or greater, no padding takes place.

DATE and TIME

The main remaining data types supported by MySQL relate to the date and time and can be seen in Table 8-11.

Table 8-11. MySQL's DATE and TIME data types

Data type		Time/date format
DATETIME		'0000-00-00 00:00:00'
DATE		'0000-00-00'
TIMESTAMP		'0000-00-00 00:00:00'
TIME		'00:00:00'
YEAR		0000 (Only years 0000 and 1901 - 2155)

The DATETIME and TIMESTAMP data types display the same way. The main difference is that TIMESTAMP has a very narrow range (the years 1970 through 2037), whereas DATE TIME will hold just about any date you're likely to specify, unless you're interested in ancient history or science fiction.

TIMESTAMP is useful, however, because you can let MySQL set the value for you. If you don't specify the value when adding a row, the current time is automatically inserted. You can also have MySQL update a TIMESTAMP column each time you change a row.

Each entry in the column id will now have a unique number, with the first starting at 1 and the others counting upwards from there. And whenever a

new row is inserted, its id column will automatically be given the next number in the sequence.

Rather than applying the column retroactively, you could have included it by issuing the CREATE command in slightly different format. In that case, the command in Example 8-3 would be replaced with Example 8-6. Check the final line in particular.

Example 8-6. Adding the autoincrementing id column at table creation

```
CREATE TABLE classics (  
  
  author VARCHAR(128),  
  
  title VARCHAR(128),  
  
  type VARCHAR(16),  
  
  year CHAR(4),  
  
  id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE  
  MyISAM;
```

If you wish to check whether the column has been added, use the following command to view the table's columns and data types:

```
DESCRIBE classics;
```

Now that we've finished with it, the id column is no longer needed, so if you created it using Example 8-5, you should now remove the column using the command in Example 8-7. *Example 8-7. Removing the id column*

```
ALTER TABLE classics DROP id;
```

Adding data to a table

To add data to a table, use the INSERT command. Let's see this in action by populating the table classics with the data from Table 8-1, using one form of the INSERT command repeatedly (Example 8-8).

Example 8-8. Populating the classics table

```
INSERT INTO classics(author, title, type, year)
```

```
VALUES('Mark Twain','The Adventures of Tom  
Sawyer','Fiction','1876'); INSERT INTO classics(author,  
title, type, year)
```

```
VALUES('Jane Austen','Pride and  
Prejudice','Fiction','1811'); INSERT INTO  
classics(author, title, type, year)
```

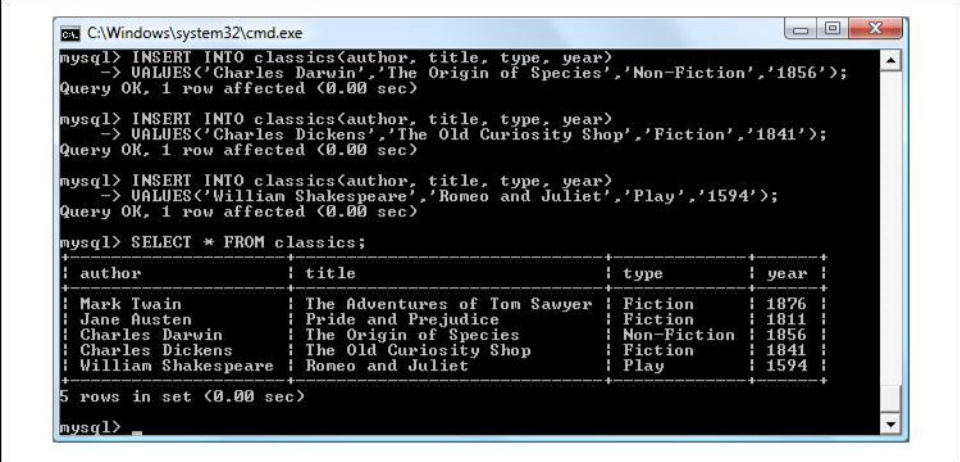
```
VALUES('Charles Darwin','The Origin of Species','Non-  
Fiction','1856'); INSERT INTO classics(author, title, type,  
year)
```

```
VALUES('Charles Dickens','The Old Curiosity  
Shop','Fiction','1841'); INSERT INTO classics(author,  
title, type, year)
```

```
VALUES('William Shakespeare','Romeo and Juliet','Play','1594');
```

After every second line, you should see a “Query OK” message. Once all lines have been entered, type the following command, which will display the table's contents—the result should look like Figure 8-4:

```
SELECT * FROM classics;
```



```
C:\Windows\system32\cmd.exe
mysql> INSERT INTO classics(author, title, type, year)
-> VALUES('Charles Darwin','The Origin of Species','Non-Fiction','1856');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO classics(author, title, type, year)
-> VALUES('Charles Dickens','The Old Curiosity Shop','Fiction','1841');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO classics(author, title, type, year)
-> VALUES('William Shakespeare','Romeo and Juliet','Play','1594');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM classics;
+-----+-----+-----+-----+
| author          | title                                | type       | year  |
+-----+-----+-----+-----+
| Mark Twain      | The Adventures of Tom Sawyer        | Fiction    | 1876  |
| Jane Austen     | Pride and Prejudice                 | Fiction    | 1811  |
| Charles Darwin  | The Origin of Species               | Non-Fiction | 1856  |
| Charles Dickens | The Old Curiosity Shop              | Fiction    | 1841  |
| William Shakespeare | Romeo and Juliet                  | Play       | 1594  |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Figure 8-4. Populating the classics table and viewing its contents

Don't worry about the `SELECT` command for now—we'll come to it in the upcoming section "Querying a MySQL Database" on page 187. Suffice it to say that as typed, it will display all the data you just entered. Let's go back and look at how we used the `INSERT` command. The first part, `INSERT INTO classics`, tells MySQL where to insert the following data. Then, within parentheses, the four column names are listed—author, title, type, and year—all separated by commas. This tells MySQL that these are the fields into which the data is to be inserted.

The second line of each `INSERT` command contains the keyword `VALUES` followed by four strings within parentheses, separated by commas. This supplies MySQL with the four values to be inserted into the four columns previously specified. (As always, my choice of where to break the lines was arbitrary.)

Each item of data will be inserted into the corresponding column, in a one-to-one correspondence. If you accidentally listed the columns in a

different order from the data, the data would go into the wrong columns. The number of columns must match the number of data items.

Renaming a table

Renaming a table, like any other change to the structure or metainformation of a table, is achieved via the ALTER command. So, for example, to change the name of the table classics to pre1900, you would use the following command:

```
ALTER TABLE classics RENAME pre1900;
```

If you tried that command, you should rename the table back again by entering the following, so that later examples in this chapter will work as printed:

```
ALTER TABLE pre1900 RENAME classics;
```

Changing the data type of a column

Changing a column's data type also makes use of the ALTER command, this time in conjunction with the MODIFY keyword. So, to change the data type of the column year from CHAR(4) to SMALLINT (which requires only two bytes of storage and so will save disk space), enter the following:

```
ALTER TABLE classics MODIFY year SMALLINT;
```

When you do this, if the data type conversion makes sense to MySQL, it will automatically change the data while keeping its meaning. In this case, it will change each string to a comparable integer, and so on, as the string is recognizable as referring to an integer.

Adding a new column

Let's suppose that you have created a table and populated it with plenty of data, only to discover you need an additional column. Not to worry. Here's how to add the new column pages, which will be used to store the number of pages in a publication:

```
ALTER TABLE classics ADD pages SMALLINT UNSIGNED;;
```

Primary keys

So far you've created the table classics and ensured that MySQL can search it quickly by adding indexes, but there's still something missing. All the publications in the table can be searched, but there is no single unique key for each publication to enable instant accessing of a row. The importance of having a key with a unique value for each row (known as the *primary key*) will become clear when we start to combine data from different tables (see the section "Primary Keys: The Keys to Relational Databases" on page 206 in Chapter 9).

The earlier section "The AUTO_INCREMENT data type" on page 176 briefly introduced the idea of a primary key when creating the autoincrementing column id, which could have been used as a primary key for this table. However, I wanted to reserve that task for a more appropriate column: the internationally recognized ISBN number.

So, let's go ahead and create a new column for this key. Now, bearing in mind that ISBN numbers are 13 characters long, you might think that the following command would do the job:

```
ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;
```

But it doesn't. If you try it, you'll get the error "Duplicate entry" for key 1. The reason is that the table is already populated with some data and this command is trying to add a column with the value NULL to each row, which is not allowed, as all columns using a primary key index must be unique. However, if there were no data already in the table, this command would work just fine, as would adding the primary key index upon table creation.

In our current situation, we have to be a bit sneaky and create the new column without an index, populate it with data, and then add the index using the commands in Example 8-13. Luckily, each of the years is unique in the current set of data, so we can use

the year column to identify each row for updating. Note that this example uses the UPDATE and WHERE keywords, which are explained in more detail in the upcoming section "Querying a MySQL Database" on page 187.

Example 8-13. Populating the isbn column with data and using a primary key

```
ALTER TABLE classics ADD isbn CHAR(13);
```

```
UPDATE classics SET isbn='9781598184891' WHERE year='1876';
```

```
UPDATE classics SET isbn='9780582506206' WHERE year='1811';
```

```
UPDATE classics SET isbn='9780517123201' WHERE year='1856';
```

```
UPDATE classics SET isbn='9780099533474' WHERE year='1841';
```

```
UPDATE classics SET isbn='9780192814968' WHERE year='1594';
```

```
ALTER TABLE classics ADD PRIMARY KEY(isbn);
```

```
DESCRIBE classics;
```

Once you have typed in these commands, the results should look like the screen grab in Figure 8-8. Note that the keywords **PRIMARY KEY** replace the keyword **INDEX** in the **ALTER TABLE** syntax (compare Example 8-10 and Example 8-13).

To create a primary key when you created the table *classics*, you could have used the commands in Example 8-14. Again, rename *classics* in line 1 to something else if you wish to try this example for yourself, and then delete the test table afterwards.

Example 8-14. Creating the table classics with indexes

```
CREATE TABLE classics (author VARCHAR(128),title  
VARCHAR(128),  
category VARCHAR(16),year SMALLINT,isbn CHAR(13),  
INDEX(author(20)),INDEX(title(20)),INDEX(category(4)),INDEX(year  
,  
  
PRIMARY KEY (isbn)) ENGINE MyISAM;
```

Querying a MySQL Database

So far we've created a MySQL database and tables, populated them with data, and added indexes to make them fast to search. Now it's time to look at how these searches are performed, and the various commands and qualifiers available.

SELECT

As you saw in Figure 8-4, the `SELECT` command is used to extract data from a table. In that section, I used its simplest form to select all the data and display it—something you will never want to do on anything but the smallest tables, because the data will scroll by at an unreadable pace. Let’s now examine `SELECT` in more detail.

The basic syntax is:

```
SELECT something FROM tablename;
```

The *something* can be an `*` (asterisk), as you saw before, to indicate “every column,” or you can choose to select only certain columns. For instance, Example 8-16 shows how to select just the author and title columns, and just the title and isbn. The result of typing these commands can be seen in Figure 8-9.

Example 8-16. Two different SELECT statements

```
SELECT author,title FROM classics;
```

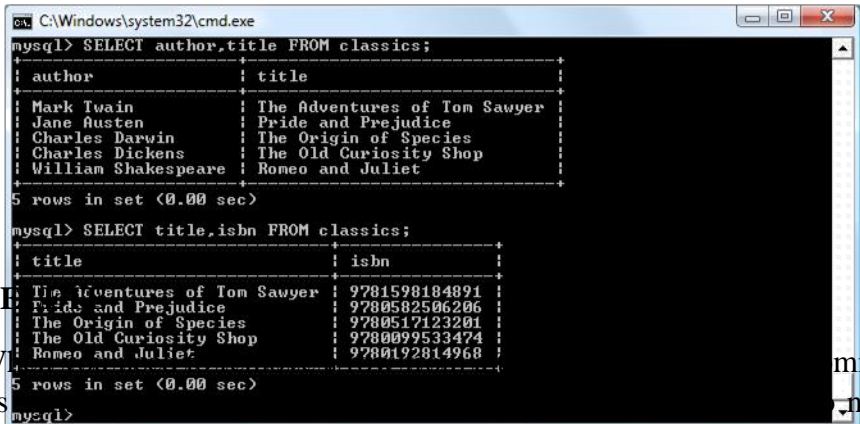
```
SELECT title,isbn FROM classics;
```

SELECT COUNT

Another option for the *something* parameter is `COUNT`, which can be used in many ways. In Example 8-17, it displays the number of rows in the table by passing `*` as a parameter, which means “all rows.” As you’d expect, the result returned is 5, as there are five publications in the table.

Example 8-17. Counting rows

SELECT COUNT(*) FROM classics;



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe" with a MySQL command prompt inside. The first query is "mysql> SELECT author,title FROM classics;" which returns a table with 5 rows. The second query is "mysql> SELECT title,isbn FROM classics;" which returns a table with 5 rows. The text "DELETE FROM classics WHERE title='Little Dorrit';" is partially visible on the left, and "command." and "narrow" are visible on the right.

author	title
Mark Twain	The Adventures of Tom Sawyer
Jane Austen	Pride and Prejudice
Charles Darwin	The Origin of Species
Charles Dickens	The Old Curiosity Shop
William Shakespeare	Romeo and Juliet

5 rows in set (0.00 sec)

title	isbn
The Adventures of Tom Sawyer	9781598184891
Pride and Prejudice	9780582506206
The Origin of Species	9780517123201
The Old Curiosity Shop	9780099533474
Romeo and Juliet	9780192814968

5 rows in set (0.00 sec)

mysql>

DELETE FROM classics WHERE title='Little Dorrit';

command.

narrow

Now that you've seen the effects of the DISTINCT qualifier, if you typed in Exam-ple 8-18, you should remove *Little Dorrit* by entering the commands in Example 8-20.

DELETE FROM classics WHERE title='Little Dorrit';

This example issues a DELETE command for all rows whose title column contains the string 'Little Dorrit'.

The WHERE keyword is very powerful, and it's important to enter it correctly; an error could lead a command to the wrong rows (or have no

effect in cases where nothing matches the WHERE clause). So now we'll spend some time on that clause, which is the heart and soul of SQL.

WHERE

The WHERE keyword enables you to narrow down queries by returning only those where a certain expression is true. Example 8-20 returns only the rows where the title column exactly matches the string 'Little Dorrit', using the equality operator =. Example 8-21 shows a couple more examples of using WHERE with =. *Example 8-21. Using the WHERE keyword*

```
SELECT author,title FROM classics WHERE  
author="Mark Twain"; SELECT author,title  
FROM classics WHERE isbn="9781598184891 ";
```

Given our current table, the two commands in Example 8-21 display the same results. But we could easily add more books by Mark Twain, in which case the first line would display all titles he wrote and the second line would continue (because we know the ISBN is unique) to display only *The Adventures of Tom Sawyer*. In other words, searches using a unique key are more predictable. You'll see further evidence later of the value of unique and primary keys.

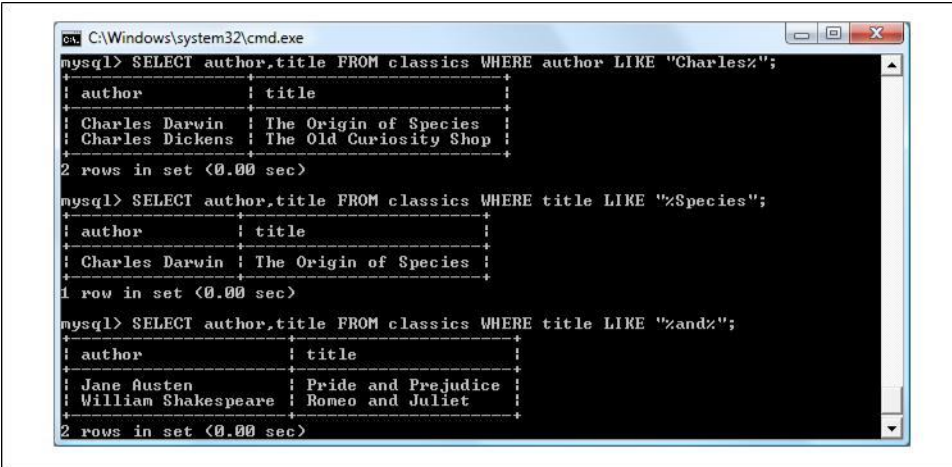
You can also do pattern matching for your searches using the LIKE qualifier, which allows searches on parts of strings. This qualifier should be used with a % character before or after some text. When placed before a keyword % means "anything before," and after a keyword it means "anything after." Example 8-22 performs three different queries, one for the start of a string, one for the end, and one for anywhere in a string. You can see the results of these commands in Figure 8-11.

Example 8-22. Using the LIKE qualifier

```
SELECT author,title FROM classics WHERE author LIKE "Charles%";
```

```
SELECT author,title FROM classics WHERE title LIKE "%Species";
```

```
SELECT author,title FROM classics WHERE title LIKE "%and%";
```



The screenshot shows a Windows command prompt window titled "cmd: C:\Windows\system32\cmd.exe". It contains three MySQL queries and their results. The first query filters by author, the second by title, and the third by a substring in the title.

```
mysql> SELECT author,title FROM classics WHERE author LIKE "Charles%";
```

author	title
Charles Darwin	The Origin of Species
Charles Dickens	The Old Curiosity Shop

2 rows in set (0.00 sec)

```
mysql> SELECT author,title FROM classics WHERE title LIKE "%Species";
```

author	title
Charles Darwin	The Origin of Species

1 row in set (0.00 sec)

```
mysql> SELECT author,title FROM classics WHERE title LIKE "%and%";
```

author	title
Jane Austen	Pride and Prejudice
William Shakespeare	Romeo and Juliet

2 rows in set (0.00 sec)

Figure 8-11. Using WHERE with the LIKE qualifier

The first command outputs the publications by both Charles Darwin and Charles Dickens, because the LIKE qualifier was set to return anything matching the string “Charles” followed by any other text. Then just *The Origin of Species* is returned, because it’s the only row whose column ends with the string “Species”. Lastly, both *Pride and Prejudice* and *Romeo and Juliet* are returned, because they both matched the string “and” anywhere in the column. The % will also match if there is nothing in the position it occupies; in other words, it can match an empty string.

UPDATE...SET

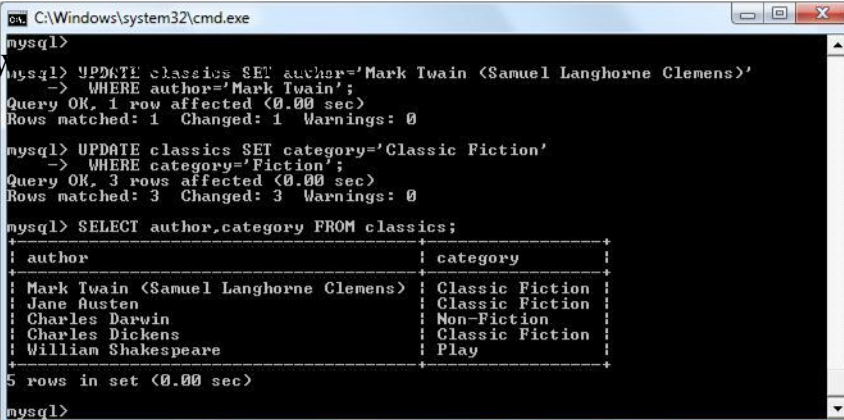
This construct allows you to update the contents of a field. If you wish to change the contents of one or more fields, you need to first narrow in on just the field or fields to be changed, in much the same way you use the SELECT command. Example 8-26 shows the use of UPDATE...SET in two different ways. You can see a screen grab of the results in Figure 8-15.

Example 8-26. Using UPDATE...SET

UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)'

WHERE author='Mark Twain';

UPDATE classics SET category='Classic Fiction'



```
mysql> UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)';
-> WHERE author='Mark Twain';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE classics SET category='Classic Fiction';
-> WHERE category='Fiction';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT author,category FROM classics;
+-----+-----+
| author                                     | category |
+-----+-----+
| Mark Twain (Samuel Langhorne Clemens)     | Classic Fiction |
| Jane Austen                               | Classic Fiction |
| Charles Darwin                             | Non-Fiction   |
| Charles Dickens                           | Classic Fiction |
| William Shakespeare                       | Play         |
+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Figure 8-15. Updating columns in the classics table

In the first query, Mark Twain's real name (Samuel Langhorne Clemens) was appended to his pen name in parens, which affected only one row. The second query, however, affected three rows, because it changed all occurrences of the word *Fiction* in the category column to the term *Classic Fiction*.

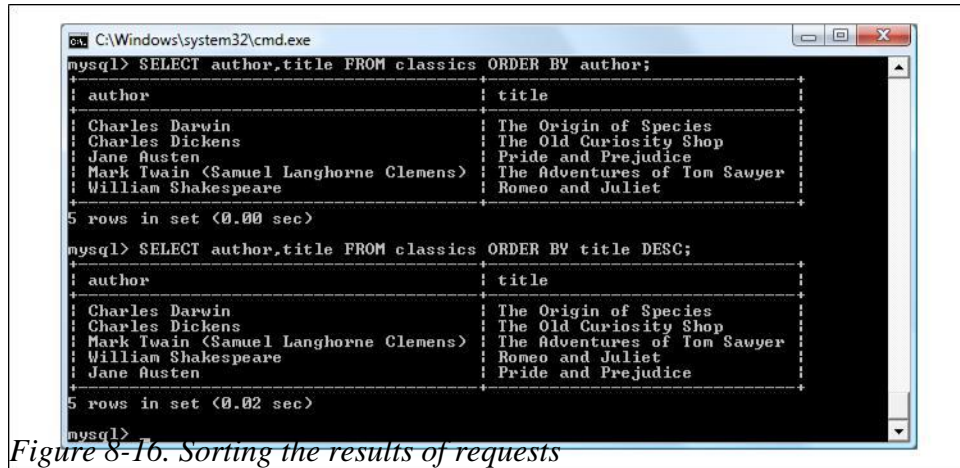
When performing an update you can also make use of the qualifiers you have already seen, such as LIMIT, and the ORDER BY and GROUP BY keywords, discussed next.

ORDER BY

ORDER BY sorts returned results by one or more columns, in ascending or descending order. Example 8-27 shows two such queries, the results of which can be seen in Figure 8-16.

Example 8-27. Using ORDER BY

```
SELECT author,title FROM classics ORDER BY  
author; SELECT author,title FROM classics ORDER  
BY title DESC;
```



```
cmd.exe
mysql> SELECT author,title FROM classics ORDER BY author;
+-----+-----+
| author                    | title                    |
+-----+-----+
| Charles Darwin            | The Origin of Species   |
| Charles Dickens           | The Old Curiosity Shop  |
| Jane Austen               | Pride and Prejudice     |
| Mark Twain (Samuel Langhorne Clemens) | The Adventures of Tom Sawyer |
| William Shakespeare       | Romeo and Juliet        |
+-----+-----+
5 rows in set (0.00 sec)

mysql> SELECT author,title FROM classics ORDER BY title DESC;
+-----+-----+
| author                    | title                    |
+-----+-----+
| Charles Darwin            | The Origin of Species   |
| Charles Dickens           | The Old Curiosity Shop  |
| Mark Twain (Samuel Langhorne Clemens) | The Adventures of Tom Sawyer |
| William Shakespeare       | Romeo and Juliet        |
| Jane Austen               | Pride and Prejudice     |
+-----+-----+
5 rows in set (0.02 sec)

mysql>
```

Figure 8-16. Sorting the results of requests

As you can see, the first query returns the publications by author in ascending alpha-betical order (the default), and the second returns them by title in descending order.

If you wanted to sort all the rows by author and then by descending year of publication (to view the most recent first), you would issue the following query:

```
SELECT author,title,year FROM classics ORDER BY author,year DESC;
```

This shows that each ascending and descending qualifier applies to a single column. The DESC keyword applies only to the preceding column, year. Because you allow author to use the default sort order, it is sorted in ascending order. You

could also have explicitly specified ascending order for that column, with the same results:

```
SELECT author,title,year FROM classics ORDER BY author ASC,year DESC;
```

UNIT-6

Accessing MySQL Using PHP

If you worked through the previous chapters, you'll be comfortable using both MySQL and PHP. In this chapter, you will learn how to integrate the two by using PHP's built-in functions to access MySQL.

Querying a MySQL Database with PHP

The reason for using PHP as an interface to MySQL is to format the results of SQL queries in a form visible in a web page. As long as you can log in to your MySQL installation using your username and password, you can also do so from PHP. However, instead of using MySQL's command line to enter instructions and view output, you will create query strings that are passed to MySQL. When MySQL returns its response, it will come as a data structure that PHP can recognize instead of the formatted output you see when you work on the command line. Further PHP commands can retrieve the data and format it for the web page.

The Process

The process of using MySQL with PHP is:

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

7. Disconnect from MySQL.

We'll work through these sections in turn, but first it's important to set up your login details in a secure manner so people snooping around on your system have trouble getting access to your database.

Creating a Login File

Most websites developed with PHP contain multiple program files that will require access to MySQL and will therefore need your login and password details. So, it's sensible to create a single file to store these and then include that file wherever it's needed. Example 10-1 shows such a file, which I've called *login.php*. Type it in, replacing the *username* and *password* values with the actual values you use for your MySQL database, and save it to the web development directory you set up in Chapter 2. We'll be making use of the file shortly. The hostname *localhost* should work as long as you're using a MySQL database on your local system, and the database publications should work if you're typing in the examples we've used so far.

Example 10-1. The login.php file

```
<?php // login.php

$db_hostname = 'localhost';

$db_database = 'publications';

$db_username = 'username';

$db_password = 'password';

?>
```


The enclosing `<?php` and `?>` tags are especially important for the *login.php* file in Ex-ample 10-1, because they mean that the lines between can be interpreted *only* as PHP code. If you were to leave them out and someone were to call up the file directly from your website, it would display as text and reveal your secrets. However, with the tags in place, all they will see is a blank page. The file will correctly include in your other PHP files.

The database we'll be using, `$db_database`, is the one called publications, which you probably created in Chapter 8. Alternatively, you can use the one you were provided with by your server administrator (in which case you'll have to modify *login.php* accordingly).

The variables `$db_username` and `$db_password` should be set to the username and pass-word that you have been using with MySQL.

Connecting to MySQL

Now that you have the *login.php* file saved, you can include it in any PHP files that will need to access the database by using the `require_once` statement. This has been chosen in preference to an `include` statement, as it will generate a fatal error if the file is not found. And believe me, not finding the file containing the login details to your database *is* a fatal error.

Also, using `require_once` instead of `require` means that the file will be read in only when it has not previously been included, which prevents wasteful duplicate disk accesses. Example 10-2 shows the code to use.

Example 10-2. Connecting to a MySQL server

```
<?php
```

```
require_once 'login.php';
```

```
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
```

```
if (!$db_server) die("Unable to connect to MySQL: " .  
mysql_error()); ?>
```

This example runs PHP's `mysql_connect` function, which requires three parameters, the *hostname*, *username*, and *password* of a MySQL server. Upon success it returns an *identifier* to the server; otherwise, `FALSE` is returned. Notice that the second line uses an `if` statement with the `die` function, which does what it sounds like and quits from PHP with an error message if `$db_server` is not `TRUE`.

The `die` message explains that it was not possible to connect to the MySQL database, and—to help identify why this happened—includes a call to the `mysql_error` function. This function outputs the error text from the last called MySQL function.

The database server pointer `$db_server` will be used in some of the following examples to identify the MySQL server to be queried. Using identifiers this way, it is possible to connect to and access multiple MySQL servers from a single PHP program. The `die` function is great for when you are developing PHP code, but of course you will want more user-friendly error messages on a production server. In this case, you won't abort your PHP program, but rather will format a message that will be displayed when the program exits normally, such as:

```
function mysql_fatal_error($msg)  
  
{  
  
    $msg2 = mysql_error();  
  
    echo <<< _END
```

We are sorry, but it was not possible to complete

the requested task. The error message we got was:

<p>\$msg: \$msg2</p>

Please click the back button on your browser

and try again. If you are still having problems,please email

our administrator. Thank you.

_END;

}

MULTIPLE QUESTIONS

UNIT-1

- 1. Which of the following variables is not a predefined variable?**
 - a. A.\$get**
 - b. B.\$ask**
 - c. C.\$request**
 - d. D.\$post**

- 2. When you need to obtain the ASCII value of a character which of the following function you apply in PHP?**
 - a. A.chr();**
 - b. B.asc();**
 - c. C.ord();**
 - d. D.val();**

- 3. Which of the following method sends input to a script via a URL?**
 - A.Get**
 - B.Post**
 - C.Both**
 - D.None**

Which of the following function returns a text in title case from a variable?

- A.ucwords(\$var)**
- B.upper(\$var)**
- C.toupper(\$var)**
- D.ucword(\$var)**

Which of the following function returns the number of characters in a string variable?

A.count(\$variable)

B.len(\$variable)

C.strcount(\$variable)

D.strlen(\$variable)

PHP Stands for?

A.PHP Hypertex ProcessorB.PHP Hyper Markup Processor

C.PHP Hyper Markup PreprocessorD.PHP Hypertext Preprocessor

PHP is an example of _____ scripting language.

A.Server-side

B.Client-side

C.Browser-side

D.In-side

Who is known as the father of PHP?

A.Rasmus Lerdorf

B.Willam Makepiece

C.Drek Kolkevi

D.List Barely

Which of the following is not true?

A.PHP can be used to develop web applications.

B.PHP makes a website dynamic

C.PHP applications can not be compile

D.PHP can not be embedded into html.

PHP scripts are enclosed within _____

A.<php> . . . </php>

B.<?php . . . ?>

C.?php . . . ?php

D.<p> . . . </p>

PHP files have extensions as

A..html

B..xml

C..php

D..ph

Unit-2

PHP's numerically indexed array begin with position _____

- a) 1
- b) 2
- c) 0
- d) -1

2. Which of the following are correct ways of creating an array?

- i) `state[0] = "karnataka";`
- ii) `$state[] = array("karnataka");`
- iii) `$state[0] = "karnataka";`
- iv) `$state = array("karnataka");`

- a) iii) and iv)
- b) ii) and iii)
- c) Only i)
- d) ii), iii) and iv)

Which of the following PHP function will return true if a variable is an array or false if it is not an array?

- a) `this_array()`
- b) `is_array()`
- c) `do_array()`
- d) `in_array()`

5. Which in-built function will add a value to the end of an array?

- a) `array_unshift()`
- b) `into_array()`
- c) `inend_array()`
- d) `array_push()`

6. What will be the output of the following PHP code?

```
<?php  
$state = array ("Karnataka", "Goa", "Tamil Nadu",  
"Andhra Pradesh");  
echo (array_search ("Tamil Nadu", $state) );  
?>
```

- a) True
- b) 1
- c) False
- d) 2

7. What will be the output of the following PHP code?

```
<?php  
$fruits = array ("apple", "orange", "banana");  
echo (next($fruits));  
echo (next($fruits));  
?>
```

- a) orangebanana
- b) appleorange
- c) orangeorange
- d) appleapple

8. Which of the following function is used to get the value of the previous element in an array?

- a) last()
- b) before()
- c) prev()
- d) previous()

. What will be the output of the following PHP code?

```
<?php  
  
$fruits = array ("apple", "orange", array ("pear", "mango"),  
"banana");  
  
echo (count($fruits, 1));  
  
?>
```

- a) 3
- b) 4
- c) 5
- d) 6

10. Which function returns an array consisting of associative key/value pairs?

- a) count()
- b) array_count()
- c) array_count_values()
- d) count_values()

What will be the output of the following PHP code?

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

- a) I like Volvo, Toyota and BMW
- b) I like Volvo, BMW and Toyota
- c) I like BMW, Volvo and Toyota
- d) I like Toyota, BMW and Volvo

2. What will be the output of the following PHP code?

```
<?php
```

```
$fname = array("Peter", "Ben", "Joe");
```

```
$age = array("35", "37", "43");
```

```
$c = array_combine($age, $fname);
```

```
print_r($c);
```

```
1. ?>
```

- a) Array (Peter Ben Joe)
- b) Array ([Peter] => 35 [Ben] => 37 [Joe] => 43)
- c) Array (35 37 43)
- d) Array ([35] => Peter [37] => Ben [43] => Joe)

unit-3

The filesize() function returns the file size in _____

- a) bits
- b) bytes
- c) kilobytes
- d) gigabytes

2. Which one of the following PHP function is used to determine a file's last access time?

- a) filemtime()
- b) filectime()
- c) fileatime()
- d) filetype()

0

Which one of the following function is capable of reading a file into a string variable?

- a) file_contents()

- b) file_get_contents()
- c) file_content()
- d) file_get_content()

5. Which one of the following function is capable of reading a specific number of characters from a file?

- a) fgets()
- b) fget()
- c) fileget()
- d) filegets()

6. Which one of the following function operates similarly to fgets(), except that it also strips any HTML and PHP tags from the input?

- a) fgetsh()
- b) fgetsp()
- c) fgetsa()
- d) fgetss()

Which one of the following function outputs the contents of a string variable to the specified resource?

- a) filewrite()
- b) fwrite()
- c) filewrites()
- d) fwrites()

8. Which function sets the file filename last-modified and last-accessed times?

- a) sets()
- b) set()
- c) touch()
- d) touched()

9. Which function is useful when you want to output the executed command result?

- a) out_cmm()
- b) out_system()

- c) cmm()
- d) system()

10. Which one of the following function reads a directory into an Array?

- a) scandir()
- b) readdir()
- c) scandirectory()
- d) readdirectory()

Which two predefined variables are used to retrieve information from forms?

- a) \$GET & \$SET
- b) \$_GET & \$_SET
- c) \$__GET & \$__SET
- d) GET & SET

2. The attack which involves the insertion of malicious code into a page frequented by other users is known as _____

- a) basic sql injection
- b) advanced sql injection
- c) cross-site scripting
- d) scripting

3. When you use the \$_GET variable to collect data, the data is visible to _____

- a) none
- b) only you
- c) everyone
- d) selected few

4. When you use the \$_POST variable to collect data, the data is visible to _____

- a) none
- b) only you

- c) everyone
- d) selected few

5. Which variable is used to collect form data sent with both the GET and POST methods?

- a) \$BOTH
- b) \$_BOTH
- c) \$REQUEST
- d) \$_REQUEST

6. Which one of the following should not be used while sending passwords or other sensitive information?

- a) GET
- b) POST
- c) REQUEST
- d) NEXT

7. Which function is used to remove all HTML tags from a string passed to a form?

- a) remove_tags()
- b) strip_tags()
- c) tags_strip()
- d) tags_remove()

To validate an email address, which flag is to be passed to the function filter_var()?

- a) FILTER_VALIDATE_EMAIL
- b) FILTER_VALIDATE_MAIL
- c) VALIDATE_EMAIL
- d) VALIDATE_MAIL

10. How many validation filters like FILTER_VALIDATE_EMAIL are currently available?

- a) 5
- b) 6
- c) 7
- d) 8

Which directive determines whether PHP scripts on the server can accept file uploads?

- a) file_uploads
- b) file_upload
- c) file_input
- d) file_intake

2. Which of the following directive determines the maximum amount of time that a PHP script will spend attempting to parse input before registering a fatal error?

- a) max_take_time
- b) max_intake_time
- c) max_input_time
- d) max_parse_time

3. What is the default value of max_input_time directive?

- a) 30 seconds
- b) 60 seconds
- c) 120 seconds
- d) 1 second

4. Since which version of PHP was the directive max_file_limit available.

- a) PHP 5.2.1
- b) PHP 5.2.2
- c) PHP 5.2.12
- d) PHP 5.2.21

UNIT-4

Which one of the following is the very first task executed by a session enabled page?

- a) Delete the previous session
- b) Start a new session
- c) Check whether a valid session exists
- d) Handle the session

2. How many ways can a session data be stored?

- a) 3
- b) 4
- c) 5
- d) 6

3. Which directive determines how the session information will be stored?

- a) save_data
- b) session.save
- c) session.save_data
- d) session.save_handler

Which one of the following is the default PHP session name?

- a) PHPSESSID
- b) PHPSESID
- c) PHPSESSIONID
- d) PHPIDSESS

5. If session.use_cookie is set to 0, this results in use of _____

- a) Session

- b) Cookie
- c) URL rewriting
- d) Nothing happens

6. If the directive `session.cookie_lifetime` is set to 3600, the cookie will live until

-
- a) 3600 sec
 - b) 3600 min
 - c) 3600 hrs
 - d) the browser is restarted

7. Neglecting to set which of the following cookie will result in the cookie's domain being set to the host name of the server which generated it.

- a) `session.domain`
- b) `session.path`
- c) `session.cookie_path`
- d) `session.cookie_domain`

What is the default number of seconds that cached session pages are made available before the new pages are created?

- a) 360
- b) 180
- c) 3600
- d) 1800

9. What is the default time(in seconds) for which session data is considered valid?

- a) 1800
- b) 3600
- c) 1440
- d) 1540

10. Which one of the following function is used to start a session?

- a) `start_session()`
- b) `session_start()`

- c) session_begin()
- d) begin_session()

Which function is used to erase all session variables stored in the current session?

- a) session_destroy()
- b) session_change()
- c) session_remove()
- d) session_unset()

2. What will the function session_id() return is no parameter is passed?

- a) Current Session Identification Number
- b) Previous Session Identification Number
- c) Last Session Identification Number
- d) Error

3. Which one of the following statements should you use to set the session username to Nachi?

- a) \$_SESSION['username'] = "Nachi";
- b) \$_SESSION['username'] = "Nachi";
- c) session_start("nachi");
- d) \$_SESSION_START["username"] = "Nachi";

An attacker somehow obtains an unsuspecting user's SID and then using it to impersonate the user in order to gain potentially sensitive information. This attack is known as _____

- a) session-fixation
- b) session-fixing
- c) session-hijack
- d) session-copy

6. Which parameter determines whether the old session file will also be deleted when the session ID is regenerated?

- a) delete_old_file

- b) delete_old_session
- c) delete_old_session_file
- d) delete_session_file

7. Which function effectively deletes all sessions that have expired?

- a) session_delete()
- b) session_destroy()
- c) session_garbage_collect()
- d) SessionHandler::gc

UNIT-5

Which one of the following databases has PHP supported almost since the beginning?

- a) Oracle Database
- b) SQL
- c) SQL+
- d) MySQL

2. The updated MySQL extension released with PHP 5 is typically referred to as

-
- a) MySQL
 - b) mysql
 - c) mysqli
 - d) mysqlly

3. Which one of the following lines need to be uncommented or added in the php.ini file so as to enable mysqli extension?

- a) extension=php_mysqli.dll
- b) extension=mysqli.dll
- c) extension=php_mysqli.dll
- d) extension=mysqli.dll

4. In which version of PHP was MySQL Native Driver(also known as mysqlnd) introduced?

- a) PHP 5.0
- b) PHP 5.1
- c) PHP 5.2
- d) PHP 5.3

Which one of the following statements is used to create a table?

- a) CREATE TABLE table_name (column_name column_type);
- b) CREATE table_name (column_type column_name);
- c) CREATE table_name (column_name column_type);
- d) CREATE TABLE table_name (column_type column_name);

6. Which one of the following statements instantiates the mysqli class?

- a) mysqli = new mysqli()
- b) \$mysqli = new mysqli()
- c) \$mysqli->new(mysqli())
- d) mysqli->new(mysqli())

7. Which one of the following statements can be used to select the database?

- a) \$mysqli=select_db('databasename');
- b) mysqli=select_db('databasename');
- c) mysqli->select_db('databasename');
- d) \$mysqli->select_db('databasename');

8. Which one of the following methods can be used to diagnose and display information about a MySQL connection error?

- a) connect_errno()
- b) connect_error()
- c) mysqli_connect_errno()
- d) mysqli_connect_error()

Which method returns the error code generated from the execution of the last MySQL function?

- a) errno()
- b) errnumber()
- c) errorno()
- d) errornumber()

10. If there is no error, then what will the error() method return?

- a) TRUE
- b) FALSE
- c) Empty String
- d) 0

Which one of the following statements should be used to include a file?

- a) #include 'filename';
- b) include 'filename';
- c) @include 'filename';
- d) #include <filename>;

2. Which one of the following methods is responsible for sending the query to the database?

- a) query()
- b) send_query()
- c) sendquery()
- d) mysqli_query()

3. Which one of the following methods recuperates any memory consumed by a result set?

- a) destroy()
- b) mysqli_free_result()

- c) alloc()
- d) free()

4. Which of the methods are used to manage result sets using both associative and indexed arrays?

- a) get_array() and get_row()
- b) get_array() and get_column()
- c) fetch_array() and fetch_row()
- d) mysqli_fetch_array() and mysqli_fetch_row()

UNIT-6

Which function is used to connect to a MySQL database in PHP?

- ☐ A) database_connect()
- ☐ B) mysql_connect()
- ☐ C) connect_database()
- ☐ D) connect_mysql()

What is use of mysql_pconnect() function in PHP?

- ☐ A) the connection will be close when the the PHP script ends
- ☐ B) the connection do not close when the the PHP script ends
- ☐ C) just to connect database
- ☐ D) None of the above

How can we know the number of rows returned in result set in PHP?

- ☐ A) using mysql_rows()

- ☐ B) using `mysql_num_rows()`
- ☐ C) using `mysql_resultset()`
- ☐ D) using `mysql_row_count()`

How do I find out all databases starting with 'test'?

A `SHOW DATABASES LIKE '%test%';`

B `SHOW DATABASES LIKE '%test;`

C `SHOW DATABASES LIKE 'test%';`

D `SHOW DATABASES LIKE 'test%';`

PHP's numerically indexed array begin with position _____

- A. 1
- B. 2
- C. 0
- D. -1

Which of the following ways are the correct way to get the current date?

A `SELECT CURTIME();`

B `SELECT CURDATE();`

C SELECT CURRENT_TIME();

D All of the above

Which of the following statement prints in PHP

A. echo

B. out

C. write

D. display

27 Which of the following is not true?

A. PHP can be used to develop web applications.

B. PHP makes a website dynamic

C. PHP applications can not be compile

D. PHP can not be embedded into html.

When do we use a HAVING clause?

Options

A To limit the output of a query

B To limit the output of a query using an aggregate function only

C When GROUP by is used

D both b and c

Which of the following method sends input to a script via a URL?

- A. Get
- B. Post
- C. Both
- D. None

The database schema is written in

- A HLL
- B DML
- C DDL
- D DCL

Can DISTINCT command be used for more than one column?

- A No
- B Yes
- C TWO
- D MANY

How can we get the number of records or rows in a table?

- A Using COUNT
- B Using NUM
- C Using NUMBER
- D Both a and c

An outer join requires each record in the two joined tables to have a matching record.

Options

A True

B False

C UNANIMOUS

D INDEFINITE

USE keyword is used to select a _____

A Table

B Column

C Database

D All of above

University
Question Paper

B.C.A. SEM-VI (2014 Course) CBCS : SUMMER - 2019

SUBJECT : WEBSITE DEVELOPMENT

Day : Monday

Date : 22/04/2019

S-2019-2081

Time : 10.00 AM TO 01.00 PM

Max. Marks :100

N.B.

- 1) Attempt any **FOUR** questions from Section - I and any **TWO** questions from Section - II.
- 2) Answers to both the sections should be written in **SAME** answer book.
- 3) Figures to the right indicate **FULL** marks.

SECTION - I

- Q.1 What are different data types and constraints used in PHP. (15)
- Q.2 a) Explain Bob's order for storing and retrieving data. (08)
b) Explain switch statement with suitable example. (07)
- Q.3 Explain date and time functions in PHP. (15)
- Q.4 Explain setting and deleting cookies with PHP. (15)
- Q.5 How to check and filter input data using MySQL? (15)
- Q.6 Explain PHP classes with example. (15)
- Q.7 Write short notes on: (15)
a) Arrays in PHP
b) Grouping and Aggregate data
c) Web database architecture

SECTION - II

- Q.8 Write user defined functions for the following using PHP: (20)
a) To print concatenated string
b) To print reverse of a given number
- Q.9 Create an application in PHP to design an input form for online library. (20)
- Q.10 Write a PHP script to generate student's database for an institute. (20)
Assume: registration no., student name, course, address, mobile number, email in an information file.
* * *

Subject : Website Development

Day : Saturday
Date : 15/04/2017



Time : 10.00 AM TO 01.00 PM
Max Marks : 100 Total Pages : 1

N.B.:

- 1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from Section – II.
- 2) Answers to both the sections should be written in **SEPARATE** answer books.
- 3) Figures to the right indicate **FULL** marks.

SECTION – I

- Q.1 a) What is meant by session? Explain the procedure to start and destroy a session in PHP. [08]
- b) Explain any seven PHP tags. [07]
- Q.2 a) Explain date and time functions in PHP. [08]
- b) Describe file handling functions in PHP. [07]
- Q.3 Explain any two control structures in PHP with suitable examples. [15]
- Q.4 Explain DDL and DML clauses related with MYSQL database accessing in PHP. [15]
- Q.5 Differentiate between: [15]
- a) Radio button and checkbox
 - b) Label and textbox
 - c) Combo box and list box
- Q.6 a) Describe arrays in PHP. [08]
- b) What are different data types and constants in PHP? [07]
- Q.7 Write short notes on **ANY TWO** of the following: [15]
- a) Cookies
 - b) Web database architecture
 - c) PHP classes

SECTION – II

- Q.8 Create an application in PHP to design an input form for online examinations system. [20]
- Q.9 Write user-defined functions in PHP: [20]
- a) To display reverse string.
 - b) To print sum of digits of a given number.
[e.g. Input : 234; Output: (2 + 3 + 4) = 9]
- Q.10 Write a PHP script to generate student marksheet for 10 different students. [20]
Assume – Roll no., Name, Course, Marks of 5 subject in student information

B.C.A. SEM-VI (2014 COURSE) CBCS : SUMMER - 2018
SUBJECT : WEBSITE DEVELOPMENT

Day : Wednesday
Date : 02/05/2018

S-2018-1716

Time : 10.00 AM TO 01.00 PM
Max. Marks : 100

N.B.:

- 1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from Section – II.
- 2) Answers to both the sections should be written in **SEPARATE** answer books.
- 3) Figures to the right indicate **FULL** marks.

SECTION – I

- Q.1** a) Explain PHP classes with appropriate examples. [08]
b) Explain various operators used in PHP. [07]
- Q.2** Explain date and time functions in PHP. [15]
- Q.3** How items in the array can be reordered? Explain with the example. [15]
- Q.4** Explain working of sessions in reference with: [15]
a) Start and destroy session c) Passing session ID in queries
b) Session variables d) Unsetting variables
- Q.5** Differentiate between following controls of forms: [15]
a) Radio button and checkbox
b) Label and textbox
c) Combo box and list box
- Q.6** Explain steps to connect and disconnect database in MYSQL with PHP. [15]
- Q.7** Write short notes on **ANY TWO** of the following: [15]
a) Cookies
b) Grouping and Aggregate clauses
c) Web database architecture

SECTION – II

- Q.8** Book-master (book_Id, booktitle, author_Id, price)
Author (author_Id, author_name, publication)
Write PHP script: [20]
a) Create database 'Library' having book-master and author tables. Create tables using appropriate constraints.
b) Write a sub query to display all information of book title, author and publication.
- Q.9** Create an application in PHP to design an input form for creating registration form for new e-mail Id. [20]
- Q.10** Write user defined functions for: [20]
a) To print GCD of two numbers.
b) To print factorial of a given number.

B.C.A. SEM-VI (2014 COURSE) CBCS : WINTER - 2017
SUBJECT : WEBSITE DEVELOPMENT

Day : Tuesday
Date : 14/11/2017

Time : 10.00 AM TO 01.00 PM
Max. Marks : 100

W-2017-1624

N.B.:

- 1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from Section – II.
- 2) Answers to both the sections should be written in **SEPARATE** answer books.
- 3) Figures to the right indicate **FULL** marks.

SECTION – I

- Q.1 a) Explain various data types used in PHP. [08]
b) What are various features of MYSQL? [07]
- Q.2 Explain various controls used while creating forms in PHP. [15]
- Q.3 Define the term “Cookies”. Explain how to set and delete cookies with PHP. [15]
- Q.4 How array can be used for sorting items? Explain with appropriate example. [15]
- Q.5 Explain various string manipulation functions used in PHP. [15]
- Q.6 Explain steps to connect and disconnect MYSQL database with PHP. [15]
- Q.7 Write short notes on **ANY TWO** of the following: [15]
a) Sessions
b) PHP classes
c) Web database architecture

SECTION – II

- Q.8 a) Write a PHP script to find whether given number is prime or not. [10]
b) Write a PHP script to display a month of a year corresponding to number of a month. Use ‘switch case’ structure. (When number ‘3’ is input, it should display ‘March’.) [10]

Subject : Website Development

Day : Saturday

Date : 15/04/2017



Time : 10.00 AM TO 01.00 PM

Max Marks : 100 Total Pages : 1

N.B.:

- 1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from Section – II.
- 2) Answers to both the sections should be written in **SEPARATE** answer books.
- 3) Figures to the right indicate **FULL** marks.

SECTION – I

- Q.1 a) What is meant by session? Explain the procedure to start and destroy a session in PHP. [08]
- b) Explain any seven PHP tags. [07]
- Q.2 a) Explain date and time functions in PHP. [08]
- b) Describe file handling functions in PHP. [07]
- Q.3 Explain any two control structures in PHP with suitable examples. [15]
- Q.4 Explain DDL and DML clauses related with MYSQL database accessing in PHP. [15]
- Q.5 Differentiate between: [15]
- a) Radio button and checkbox
 - b) Label and textbox
 - c) Combo box and list box
- Q.6 a) Describe arrays in PHP. [08]
- b) What are different data types and constants in PHP? [07]
- Q.7 Write short notes on **ANY TWO** of the following: [15]
- a) Cookies
 - b) Web database architecture
 - c) PHP classes

SECTION – II

- Q.8 Create an application in PHP to design an input form for online examinations system. [20]
- Q.9 Write user-defined functions in PHP: [20]
- a) To display reverse string.
 - b) To print sum of digits of a given number.
[e.g. Input : 234; Output: (2 + 3 + 4) = 9]

B.C.A. SEM-VI (2014 COURSE) CBCS : SUMMER - 2018
SUBJECT : WEBSITE DEVELOPMENT

Day : Wednesday
Date : 02/05/2018

S-2018-1716

Time : 10.00 AM TO 01.00 PM
Max. Marks : 100

N.B.:

- 1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from Section – II.
- 2) Answers to both the sections should be written in **SEPARATE** answer books.
- 3) Figures to the right indicate **FULL** marks.

SECTION – I

- Q.1** a) Explain PHP classes with appropriate examples. [08]
b) Explain various operators used in PHP. [07]
- Q.2** Explain date and time functions in PHP. [15]
- Q.3** How items in the array can be reordered? Explain with the example. [15]
- Q.4** Explain working of sessions in reference with: [15]
a) Start and destroy session c) Passing session ID in queries
b) Session variables d) Unsetting variables
- Q.5** Differentiate between following controls of forms: [15]
a) Radio button and checkbox
b) Label and textbox
c) Combo box and list box
- Q.6** Explain steps to connect and disconnect database in MYSQL with PHP. [15]
- Q.7** Write short notes on **ANY TWO** of the following: [15]
a) Cookies
b) Grouping and Aggregate clauses
c) Web database architecture

SECTION – II

- Q.8** Book-master (book_Id, booktitle, author_Id, price)
Author (author_Id, author_name, publication) [20]
Write PHP script:
a) Create database 'Library' having book-master and author tables. Create tables using appropriate constraints.
b) Write a sub query to display all information of book title, author and publication.
- Q.9** Create an application in PHP to design an input form for creating registration form for new e-mail Id. [20]
- Q.10** Write user defined functions for: [20]
a) To print GCD of two numbers.

Subject : Website Development

Day : Saturday
Date : 15/04/2017



35645

Time : 10.00 AM TO 01.00 PM
Max Marks : 100 Total Pages : 1

N.B.:

- 1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from Section – II.
- 2) Answers to both the sections should be written in **SEPARATE** answer books.
- 3) Figures to the right indicate **FULL** marks.

SECTION – I

- Q.1** a) What is meant by session? Explain the procedure to start and destroy a session in PHP. [08]
b) Explain any seven PHP tags. [07]
- Q.2** a) Explain date and time functions in PHP. [08]
b) Describe file handling functions in PHP. [07]
- Q.3** Explain any two control structures in PHP with suitable examples. [15]
- Q.4** Explain DDL and DML clauses related with MYSQL database accessing in PHP. [15]
- Q.5** Differentiate between: [15]
a) Radio button and checkbox
b) Label and textbox
c) Combo box and list box
- Q.6** a) Describe arrays in PHP. [08]
b) What are different data types and constants in PHP? [07]
- Q.7** Write short notes on **ANY TWO** of the following: [15]
a) Cookies
b) Web database architecture
c) PHP classes

SECTION – II

- Q.8** Create an application in PHP to design an input form for online examinations system. [20]
- Q.9** Write user-defined functions in PHP: [20]
a) To display reverse string.
b) To print sum of digits of a given number.
[e.g. Input : 234; Output: (2 + 3 + 4) = 9]
- Q.10** Write a PHP script to generate student marksheet for 10 different students. [20]
Assume – Roll no., Name, Course, Marks of 5 subject in student information file.

Internal Question Papers



**Bharati Vidyapeeth Deemed University,
Institute of Management and Research (BVIMR), New Delhi**

1st Internal Examination

Course :BCA

Semester :VI

Subject:Website Development

Course Code: 603

Max. Marks: 40

Max. Time: 2 Hours

Instructions (if any): Give examples & Diagrammatic Representations

whenever as possible.

Question No 1 is compulsory. Attempt any two questions from Q2 to Q5.

Attempt any two question from section 2.

Each Question in Section 1 carries 6marks & Each Qestion in Section 2 carries 11 marks.

Section 1

in 400 words. Each question carry 06 marks.

Q1.

Q2.

Q3.

Q4

Q5.

a)

b)

c)

in 800 words. Attempt any 2 questions. Each carry 11 marks.

Q6.

Q7.

Q8. a)

b)

c)



Bharati Vidyapeeth Deemed University,



Institute of Management and Research (BVIMR), New Delhi

2nd Internal Examination

Course :BCA

Semester :VI

Subject:Website Development

Course Code: 603

Max. Marks: 40

Max. Time: 2 Hours

Instructions (if any): Give examples & Diagrammatic Representations

whenever as possible.

Question No 1 is compulsory. Attempt any two questions from Q2 to Q5.

Attempt any two question from section 2.

Each Question in Section 1 carries 6marks & Each Qestion in Section 2 carries 11 marks.

Section 1

in 400 words. Each question carry 06 marks.

Q1.

Q2.

Q3.

Q4

Q5. a)

b)

c)

in 800 words. Attempt any 2 questions. Each carry 11 marks.

Q6.

Q7.

Q8. a)

b)

c)

- Design an HTML website for displaying different courses offered by the university using various types of lists.
- Create a Table to display the following table with items in the Décor using **ordered lists and Hyperlinks**. On the click of the Décor items, suitable pages with their images and information should be displayed.

HOUSE TYPES	DECOR	ROOMS	PRICE/SQ FT.
Villas	1. <u>Dining Room</u>	4BHK	Rs.5000
	2. <u>Master Bed Room</u>		
	3. <u>Kid's Room</u>		
	4. <u>Kitchen</u>		
Mansion	1. <u>Dining Room</u>	5BHK	Rs 7000
	2. <u>Master Bed Room</u>		
	3. <u>Kid's Room</u>		
	4. <u>Kitchen</u>		
Flats	1. <u>Single Bedroom</u>	1BHK	Rs500
	2. <u>Double Bedroom</u>	2BHK	Rs. 700

- Create a web page, which prompts a user to enter a colour, then sets the background Colour to that value.
- Insert an image to a web page and display its co-ordinates on a mouse move.
- To Display Biodata.
- To calculate Area of a Circle.
- An electric power distribution company charges its domestic consumer as

follows

Consumption units

Rate of charge

0-200

Rs. 0.50 / unit

201-400

Rs. 100 + .65/unit (excess of 200)

401-600

Rs. 230 + .80/unit (excess of 400)

601 and above

Rs 390 + Rs 1.00 /unit (excess of 600)

WAP to calculate Amount to be paid after accepting the consumption units.

- To display days of week.
- To sort n numbers using Bubble sort
- To Search an element using Binary search
- To merge two sorted arrays.
- Convert Binary number into decimal
- To check whether a year is leap year or not.
- Write a Procedure space (x) that can be used to provide x spaces.
- To Display capital of a state.
- Form Validation
- To create a student file-containing student records.(RollNo, Name, Marks1,Marks2,Marks3)
- To Copy the contents of previous file into new file & add total marks field into it.
- To upload a file
- PHP Sessions
 - Starting a PHP Session
 - Storing a Session Variable
 - Destroying a Session
- PHP Cookies
 - Create a Cookie
 - Retrieve a Cookie Value
 - Delete a Cookie
- What is MySQL
 - Variable, Constraints, Datatypes of MySQL