Learning PHP 5
By David Sklar

Publisher: O'Reilly
Pub Date: June 2004
ISBN: 0-596-00560-1
Pages: 368

Learning PHP 5 is the ideal tutorial for graphic designers, bloggers, and other web crafters who want a thorough but non-intimidating way to understand the code that makes web sites dynamic. The book begins with an introduction to PHP, then moves to more advanced features: language basics, arrays and functions, web forms, connecting to databases, and much more. Complete with exercises to make sure the lessons stick, this book offers the ideal classroom learning experience whether you're in a classroom or on your own.
Chapter 10. Working with Files
Section 10.1. Understanding File Permissions
Section 10.2. Reading and Writing Entire Files
Section 10.3. Reading and Writing Parts of Files
Section 10.4. Working with CSV Files
Section 10.5. Inspecting File Permissions
Section 10.6. Checking for Errors
Section 10.7. Sanitizing Externally Supplied Filenames
Section 10.8. Chapter Summary
Section 10.9. Exercises

Chapter 11. Parsing and Generating XML
Section 11.1. Parsing an XML Document
Section 11.2. Generating an XML Document
Section 11.3. Chapter Summary
Section 11.4. Exercises

Chapter 12. Debugging
Section 12.1. Controlling Where Errors Appear
Section 12.2. Fixing Parse Errors
Section 12.3. Inspecting Program Data
Section 12.4. Fixing Database Errors
Section 12.5. Chapter Summary
Section 12.6. Exercises

Chapter 13. What Else Can You Do with PHP?
Section 13.1. Graphics
Section 13.2. PDF
Section 13.3. Shockwave/Flash
Section 13.4. Browser-Specific Code
Section 13.5. Sending and Receiving Mail
Section 13.6. Uploading Files in Forms
Section 13.7. The HTML_QuickForm Form-Handling Framework
Section 13.8. Classes and Objects
Section 13.9. Advanced XML Processing
Section 13.10. SQLite
Section 13.11. Running Shell Commands
Section 13.12. Advanced Math
Section 13.13. Encryption
Section 13.14. Talking to Other Languages
Section 13.15. IMAP, POP3, and NNTP
Section 13.16. Command-Line PHP
Section 13.17. PHP-GTK
Section 13.18. Even More Things You Can Do with PHP

Appendix A. Installing and Configuring the PHP Interpreter
Section A.1. Using PHP with a Web-Hosting Provider
Section A.2. Installing the PHP Interpreter
Section A.3. Installing PEAR
Section A.4. Downloading and Installing PHP's Friends
Section A.5. Modifying PHP Configuration Directives
Section A.6. Appendix Summary

Appendix B. Regular Expression Basics
Section B.1. Characters and Metacharacters
Section B.2. Quantifiers
Section B.3. Anchors
Section B.4. Character Classes
Section B.5. Greed
Section B.6. PHP's PCRE Functions
Dedication

To Jacob, who can look forward to so much learning.
Preface

Boring web sites are static. Interesting web sites are dynamic. That is, their content changes. A giant static HTML page listing the names, pictures, descriptions, and prices of all 1,000 products a company has for sale is hard to use and takes forever to load. A dynamic web product catalog that lets you search and filter those products so you see only the six items that meet your price and category criteria is more useful, faster, and much more likely to close a sale.

The PHP programming language makes it easy to build dynamic web sites. Whatever interactive excitement you want to create—such as a product catalog, a blog, a photo album, or an event calendar—PHP is up to the task. And after reading this book, you'll be up to the task of building that dynamic web site, too.
Who This Book Is For

This book is for:

- A hobbyist who wants to create an interactive web site for himself, his family, or a nonprofit organization.
- A web site builder who wants to use the PHP setup provided by an ISP or hosting provider.
- A small business owner who wants to put her company on the Web.
- A page designer who wants to communicate better with her developer co-workers.
- A JavaScript whiz who wants to build server-side programs that complement her client-side code.
- A blogger or HTML jockey who wants to easily add dynamic features to her site.
- A Perl, ASP, or ColdFusion programmer who wants to get up to speed with PHP.
- Anybody who wants a straightforward, jargon-free introduction to one of the most popular programming languages for building an interactive web site.

PHP's gentle learning curve and approachable syntax make it an ideal "gateway" language for the nontechnical web professional. Learning PHP 5 is aimed at both this interested, intelligent, but not necessarily technical individual as well as at programmers familiar with another language who want to learn PHP.

Aside from basic computer literacy (knowing how to type, moving files around, surfing the Web), the only assumption that this book makes about you is that you're acquainted with HTML. You don't need to be an HTML master, but you should be comfortable with the HTML tags that populate a basic web page such as <html>, <head>, <body>, <p>, <a>, and <br>. If you're not familiar with HTML, read HTML & XHTML: The Definitive Guide, Fifth Edition, by Bill Kennedy and Chuck Musciano (O'Reilly).
Contents of This Book

This book is designed so that you start at the beginning and work through the chapters in order. For the most part, each chapter depends on material in the previous chapters. Chapter 2, through Chapter 12 and Appendix B, each end with exercises that test your understanding of the content in the chapter.

Chapter 1, provides some general background on PHP and how it interacts with your web browser and a web server. It also shows some PHP programs and what they do to give you an idea of what PHP programs look like. Especially if you're new to programming or building dynamic web sites, it is important to read Chapter 1.

The next four chapters give you a grounding in the fundamentals of PHP. Before you can write great literature, you need to learn a little grammar and some vocabulary. That's what these chapters are for. (Don't worry—you'll learn enough PHP grammar and vocabulary right away to start writing some short programs, if not great literature.) Chapter 2 shows you how to work with different kinds of data such as pieces of text and numbers. This is important because the web pages that your PHP programs generate are just big pieces of text. Chapter 3, describes the PHP commands with which your programs can make decisions. These decisions are at the heart of the "dynamic" in "dynamic web site." The concepts in Chapter 3 are what you use, for example, to display only items in a product catalog that fall between two prices a user enters in a web form.

Chapter 4, introduces arrays, which are collections of a bunch of individual numbers or pieces of text. Many frequent activities in PHP programs, such as processing submitted web form parameters or examining information pulled out of a database, involve using arrays. As you write more complicated programs, you'll find yourself wanting to repeat similar tasks. Functions, discussed in Chapter 5, help you reuse pieces of your programs.

The three chapters after that cover three essential tasks in building a dynamic web site: dealing with forms, databases, and users. Chapter 6, supplies the details on working with web forms. These are the primary way that users interact with your web site. Chapter 7, discusses databases. A database holds the information that your web site displays, such as a product catalog or event calendar. This chapter shows you how to make your PHP programs talk to a database. With the techniques in Chapter 8, your web site can do user-specific things such as display sensitive information to authorized people only or tell someone how many new message board posts have been created since she last logged in.

Then, the next three chapters examine three other areas you're likely to encounter when building your web site. Chapter 9, highlights the steps you need to take, for example, to display a monthly calendar or to allow users to input a date or time from a web form. Chapter 10, describes the PHP commands for interacting with files on your own computer or elsewhere on the Internet. Chapter 11, supplies the basics for dealing with XML documents in your PHP programs, whether you need to generate one for another program to consume or you've been provided with one to use in your own program.

Chapter 12 and Chapter 13 each stand on their own. Chapter 12, furnishes some approaches for understanding the error messages that the PHP interpreter generates and hunting down problems in your programs. While it partially depends on earlier material, it may be worthwhile to skip ahead and peruse Chapter 12 as you're working through the book.

Chapter 13 serves a taste of many additional capabilities of PHP, such as generating images, running code written in other languages, and making Flash movies. After you've gotten comfortable with the core PHP concepts explained in Chapter 1 through Chapter 12, visit Chapter 13 for lots of new things to learn.
Other Resources

The online annotated PHP Manual (http://www.php.net/manual) is a great resource for exploring PHP's extensive function library. Plenty of user-contributed comments offer helpful advice and sample code, too. Additionally, there are many PHP mailing lists covering installation, programming, extending PHP, and various other topics. You can learn about and subscribe to these mailing lists at http://www.php.net/mailing-lists.php. A read-only web interface to the mailing lists is at http://news.php.net. Also worth exploring is the PHP Presentation System archive at http://talks.php.net. This is a collection of presentations about PHP that have been delivered at various conferences.

After you're comfortable with the material in this book, the following books about PHP are good next steps:

- Programming PHP, by Rasmus Lerdorf and Kevin Tatroe (O'Reilly). A more detailed and technical look at how to write PHP programs. Includes information on generating graphics and PDFs.

- PHP Cookbook, by David Sklar and Adam Trachtenberg (O'Reilly). A comprehensive collection of common PHP programming problems and their solutions.

- Essential PHP Tools, by David Sklar (Apress). Examples and explanations about many popular PHP add-on libraries and modules including HTML_QuickForm, SOAP, and the Smarty templating system.

- Upgrading to PHP 5, by Adam Trachtenberg (O'Reilly). A comprehensive look at the new features of PHP 5, including coverage of features for XML handling and object-oriented programming.

These books are helpful for learning about databases, SQL, and MySQL:

- Web Database Applications with PHP & MySQL, by David Lane and Hugh E. Williams (O'Reilly). How to make PHP and MySQL sing in harmony to make a robust dynamic web site.

- SQL in a Nutshell, by Kevin E. Kline (O'Reilly). The essentials you need to know to write SQL queries. Covers the SQL dialects used by Microsoft SQL Server, MySQL, Oracle, and PostgreSQL.


These books are helpful for learning about HTML and HTTP:


- Dynamic HTML: The Definitive Reference, by Danny Goodman (O'Reilly). Full of useful information you need if you're using JavaScript or Dynamic HTML as part of the web pages your PHP programs output.

- HTTP Developer's Handbook, by Chris Shiflett (Sams Publishing). With this book, you'll better understand how your web browser and a web server communicate with each other.

These books are helpful for learning about security and cryptography:


- Applied Cryptography, by Bruce Schneier (John Wiley & Sons). The nitty gritty on how different cryptographic algorithms work and why.

These books are helpful for learning about supplementary topics that this book touches on like XML processing and regular expressions:

- Learning XML, by Erik T. Ray (O'Reilly). Where to go for more in-depth information on XML than Chapter 11.

- Learning XSLT, by Michael Fitzgerald (O'Reilly). Your guide to XML stylesheets and XSL transformations.

- Mastering Regular Expressions, by Jeffrey E.F. Friedl (O'Reilly). After you've digested Appendix B, turn to this book for everything you ever wanted to know about regular expressions.
Conventions Used in This Book

The following programming and typesetting conventions are used in this book.

Programming Conventions

The code examples in this book are designed to work with PHP 5.0.0. They were tested with PHP 5.0.0RC2, which was the most up-to-date version of PHP 5 available at the time of publication. Almost all of the code in the book works with PHP 4.3 as well. The PHP 5-specific features discussed in the book are as follows:

- Chapter 7: the mysqli functions
- Chapter 10: the file_put_contents() function
- Chapter 11: the SimpleXML module
- Chapter 12: the E_STRICT error-reporting level
- Chapter 13: some new features related to classes and objects, the advanced XML processing functions, the bundled SQLite database, and the Perl extension

Typographical Conventions

The following typographical conventions are used in this book:

Italic

Indicates new terms, example URLs, example email addresses, filenames, file extensions, pathnames, and directories.

Constant width

Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

Constant width italic

Shows text that should be replaced with user-supplied values.
Using Code Examples

Typing some of the example programs in the book yourself is instructive when you are getting started. However, if your fingers get weary, you can download all of the code examples from http://www.oreilly.com/catalog/learnphp5.

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact the publisher for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "Learning PHP 5 by David Sklar Copyright 2004 O'Reilly Media, Inc., 0-596-00560-1." If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact the publisher at permissions@oreilly.com.
Comments and Questions

Please address comments and questions concerning this book to the publisher:
O'Reilly Media, Inc. 1005 Gravenstein Highway North Sebastopol, CA 95472 (800) 998-9938 (in the United States or Canada) (707) 829-0515 (international or local) (707) 829-0104 (fax)

There is a web page for this book, where we list errata, examples, and any additional information. You can access this page at:
http://www.oreilly.com/catalog/learnphp5

To comment or ask technical questions about this book, send email to:
bookquestions@oreilly.com

Or you can contact the author directly via his web site:
http://www.sklar.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:
http://www.oreilly.com
Acknowledgments

This book is the end result of the hard work of many people. Thank you to:

- The many programmers, testers, documentation writers, bug fixers, and other folks whose time, talent, and devotion have made PHP the first-class development platform that it is today. Without them, I'd have nothing to write about.

- The Apple WWPM Hardware Placement Lab for the loan of an iBook, and to Adam Trachtenberg, George Schlossnagle, and Jeremy Zawodny for advice on some code examples.

- My diligent reviewers: Griffin Cherry, Florence Leroy, Mark Oglia, and Stewart Ugelow. They caught plenty of mistakes, turned confusing explanations into clear ones, and otherwise made this book far better than it would have been without them.

- Robert Romano, who turned my blocky diagrams and rustic pencil sketches into high-quality figures and illustrations.

- Tatiana Diaz, who funneled all of my random questions to the right people, kept me on schedule, and ultimately made sure that whatever needed to get done, was done.

- Nat Torkington, whose editorial guidance and helpful suggestions improved every part of the book. Without Nat’s feedback, this book would be twice as long and half as readable as it is.

For a better fate than wisdom, thank you also to Susannah, with whom I enjoy ignoring the syntax of things.
Chapter 1. Orientation and First Steps

There are lots of great reasons to write computer programs in PHP. Maybe you want to learn PHP because you need to put together a small web site for yourself that has some interactive elements. Perhaps PHP is being used where you work and you have to get up to speed. This chapter provides context for how PHP fits into the puzzle of web site construction: what it can do and why it's so good at what it does. You'll also get your first look at the PHP language and see it in action.
1.1 PHP's Place in the Web World

PHP is a programming language that's used mostly for building web sites. Instead of a PHP program running on a desktop computer for the use of one person, it typically runs on a web server and is accessed by lots of people using web browsers on their own computers. This section explains how PHP fits into the interaction between a web browser and a web server.

When you sit down at your computer and pull up a web page using a browser such as Internet Explorer or Mozilla, you cause a little conversation to happen over the Internet between your computer and another computer. This conversation and how it makes a web page appear on your screen is illustrated in Figure 1-1.

Figure 1-1. Client and server communication without PHP

Here's what's happening in the numbered steps of the diagram:

1. You type www.example.com/catalog.html into the location bar of Internet Explorer.

2. Internet Explorer sends a message over the Internet to the computer named www.example.com asking for the /catalog.html page.

3. Apache, a program running on the www.example.com computer, gets the message and reads the catalog.html file from the disk drive.

4. Apache sends the contents of the file back to your computer over the Internet as a response to Internet Explorer's request.

5. Internet Explorer displays the page on the screen, following the instructions of the HTML tags in the page.

Every time a browser asks for http://www.example.com/catalog.html, the web server sends back the contents of the same catalog.html file. The only time the response from the web server changes is if someone edits the file on the server.
1.2 What's So Great About PHP?

You may be attracted to PHP because it's free, because it's easy to learn, or because your boss told you that you need to start working on a PHP project next week. Since you're going to use PHP, you need to know a little bit about what makes it special. The next time someone asks you "What's so great about PHP?", use this section as the basis for your answer.

1.2.1 PHP Is Free (as in Money)

You don't have to pay anyone to use PHP. Whether you run the PHP interpreter on a beat-up 10-year-old PC in your basement or in a room full of million-dollar "enterprise-class" servers, there are no licensing fees, support fees, maintenance fees, upgrade fees, or any other kind of charge.

Most Linux distributions come with PHP already installed. If yours doesn't, or you are using another operating system such as Windows, you can download PHP from [http://www.php.net/](http://www.php.net/). Appendix A has detailed instructions on how to install PHP.

1.2.2 PHP Is Free (as in Speech)

As an open source project, PHP makes its innards available for anyone to inspect. If it doesn't do what you want, or you're just curious about why a feature works the way it does, you can poke around in the guts of the PHP interpreter (written in the C programming language) to see what's what. Even if you don't have the technical expertise to do that, you can get someone who does to do the investigating for you. Most people can't fix their own cars, but it's nice to be able to take your car to a mechanic who can pop open the hood and fix it.

1.2.3 PHP Is Cross-Platform

You can use PHP with a web server computer that runs Windows, Mac OS X, Linux, Solaris, and many other versions of Unix. Plus, if you switch web server operating systems, you generally don't have to change any of your PHP programs. Just copy them from your Windows server to your Unix server, and they will still work.

While Apache is the most popular web server program used with PHP, you can also use Microsoft Internet Information Server and any other web server that supports the CGI standard. PHP also works with a large number of databases including MySQL, Oracle, Microsoft SQL Server, Sybase, and PostgreSQL. In addition, it supports the ODBC standard for database interaction.

If all the acronyms in the last paragraph freak you out, don't worry. It boils down to this: whatever system you're using, PHP probably runs on it just fine and works with whatever database you are already using.

1.2.4 PHP Is Widely Used

As of March 2004, PHP is installed on more than 15 million different web sites, from countless tiny personal home pages to giants like Yahoo!. There are many books, magazines, and web sites devoted to teaching PHP and exploring what you can do with it. There are companies that provide support and training for PHP. In short, if you are a PHP user, you are not alone.
1.3 PHP in Action

Ready for your first taste of PHP? This section contains a few program listings and explanations of what they do. If you don't understand everything going on in each listing, don't worry! That's what the rest of the book is for. Read these listings to get a sense of what PHP programs look like and an outline of how they work. Don't sweat the details yet.

When given a program to run, the PHP interpreter pays attention only to the parts of the program between PHP start and end tags. Whatever's outside those tags is printed with no modification. This makes it easy to embed small bits of PHP in pages that mostly contain HTML. The PHP interpreter runs the commands between <?php (the PHP start tag) and ?> (the PHP end tag). PHP pages typically live in files whose names end in .php. Example 1-1 shows a page with one PHP command.

Example 1-1. Hello, World!

```html
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```
1.4 Basic Rules of PHP Programs

This section lays out some ground rules about the structure of PHP programs. More foundational than the basics such as "how do I print something" or "how do I add two numbers", these proto-basics are the equivalent of someone telling you that you should read pages in this book from top to bottom and left to right, or that what's important on the page are the black squiggles, not the large white areas.

If you've had a little experience with PHP already or you're the kind of person that prefers playing with all the buttons on your new DVD player before going back and reading in the manual about how the buttons actually work, feel free to skip ahead to Chapter 2 now and flip back here later. If you forge ahead to write some PHP programs of your own, and they're behaving unexpectedly or the PHP interpreter complains of "parse errors" when it tries to run your program, revisit this section for a refresher.

1.4.1 Start and End Tags

Each of the examples you've already seen in this chapter uses <?php as the PHP start tag and ?> as the PHP end tag. The PHP interpreter ignores anything outside of those tags. Text before the start tag or after the end tag is printed with no interference from the PHP interpreter.

A PHP program can have multiple start and end tag pairs, as shown in Example 1-8.

Example 1-8. Multiple start and end tags

Five plus five is:
1.5 Chapter Summary

Chapter 1 covers:

- PHP’s usage by a web server to create a response or document to send back to the browser.

- PHP as a server-side language, meaning it runs on the web server. This is in contrast to a client-side language such as JavaScript.

- What you sign up for when you decide to use PHP: it's free (in terms of money and speech), cross-platform, popular, and designed for web programming.

- How PHP programs that print information, process forms, and talk to a database appear.

- Some basics of the structure of PHP programs, such as the PHP start and end tags (<?php and ?>), whitespace, case-sensitivity, and comments.
Chapter 2. Working with Text and Numbers

PHP can work with different types of data. In this chapter, you'll learn about individual values such as numbers and single pieces of text. You'll learn how to put text and numbers in your programs, as well as some of the limitations the PHP interpreter puts on those values and some common tricks for manipulating them.

Most PHP programs spend a lot of time handling text because they spend a lot of time generating HTML and working with information in a database. HTML is just a specially formatted kind of text, and information in a database, such as a username, a product description, or an address is a piece of text, too. Slicing and dicing text easily means you can build dynamic web pages easily.

In Chapter 1, you saw variables in action, but this chapter teaches you more about them. A variable is a named container that holds a value. The value that a variable holds can change as a program runs. When you access data submitted from a form or exchange data with a database, you use variables. In real life, a variable is something such as your checking account balance. As time goes on, the value that the phrase "checking account balance" refers to fluctuates. In a PHP program, a variable might hold the value of a submitted form parameter. Each time the program runs, the value of the submitted form parameter can be different. But whatever the value, you can always refer to it by the same name. This chapter also explains in more detail what variables are: how you create them and do things such as change their values or print them.
2.1 Text

When they're used in computer programs, pieces of text are called strings. This is because they consist of individual characters, strung together. Strings can contain letters, numbers, punctuation, spaces, tabs, or any other characters. Some examples of strings are I would like 1 bowl of soup, and "Is it too hot?" he asked, and There's no spoon!. A string can even contain the contents of a binary file such as an image or a sound. The only limit to the length of a string in a PHP program is the amount of memory your computer has.

2.1.1 Defining Text Strings

There are a few ways to indicate a string in a PHP program. The simplest is to surround the string with single quotes:

```php
print 'I would like a bowl of soup.';
```
2.2 Numbers

Numbers in PHP are expressed using familiar notation, although you can't use commas or any other characters to group thousands. You don't have to do anything special to use a number with a decimal part as compared to an integer. Example 2-15 lists some valid numbers in PHP.

Example 2-15. Numbers

```php
print 56;
```
2.3 Variables

Variables hold the data that your program manipulates while it runs, such as information about a user that you've loaded from a database or entries that have been typed into an HTML form. In PHP, variables are denoted by $ followed by the variable's name. To assign a value to a variable, use an equals sign (=). This is known as the assignment operator.

```php
$plates = 5;
```
2.4 Chapter Summary

Chapter 2 covers:

- Defining strings in your programs three different ways: with single quotes, with double quotes, and as a here document.

- Escaping: what it is and what characters need to be escaped in each kind of string.

- Validating a string by checking its length, removing leading and trailing whitespace from it, or comparing it to another string.

- Formatting a string with printf().

- Manipulating the case of a string with(strtolower(), strtoupper(), or ucwords().

- Selecting part of a string with substr().

- Changing part of a string with str_replace().

- Defining numbers in your programs.

- Doing math with numbers.

- Storing values in variables.

- Naming variables appropriately.

- Using combined operators with variables.

- Using increment and decrement operators with variables.

- Interpolating variables in strings.
2.5 Exercises

1.

Find the errors in this PHP program:

```php
<? php
```
Chapter 3. Making Decisions and Repeating Yourself

Chapter 2 covered the basics of how to represent data in PHP programs. A program full of data is only half complete, though. The other piece of the puzzle is using that data to control how the program runs, taking actions such as:

- If an administrative user is logged in, print a special menu.
- Print a different page header if it's after three o'clock.
- Notify a user if new messages have been posted since she last logged in.

All of these actions have something in common: they make decisions about whether a certain logical condition involving data is true or false. In the first action, the logical condition is "Is an administrative user logged in?" If the condition is true (yes, an administrative user is logged in), then a special menu is printed. The same kind of thing happens in the next example. If the condition "is it after three o'clock?" is true, then a different page header is printed. Likewise, if "Have new messages been posted since the user last logged in?" is true, then the user is notified.

When making decisions, the PHP interpreter boils down an expression into true or false. Section 3.1 explains how the interpreter decides which expressions and values are true and which are false.

Those true and false values are used by language constructs such as if( ) to decide whether to run certain statements in a program. The ins and outs of if( ) are detailed later in this chapter in Section 3.2. Use if( ) and similar constructs any time the outcome of a program depends on some changing conditions.

While true and false are the cornerstones of decision making, usually you want to ask more complicated questions, such as "is this user at least 21 years old?" or "does this user have a monthly subscription to the web site or enough money in their account to buy a daily pass?" Section 3.3, later in this chapter, explains PHP's comparison and logical operators. These help you express whatever kind of decision you need to make in a program, such as seeing whether numbers or strings are greater than or less than each other. You can also chain together decisions into a larger decision that depends on its pieces.

Decision making is also used in programs when you want to repeatedly execute certain statements — you need a way to indicate when the repetition should stop. Frequently, this is determined by a simple counter, such as "repeat 10 times." This is like asking the question "Have I repeated 10 times yet?" If so, then the program continues. If not, the action is repeated again. Determining when to stop can be more complicated, too — for example, "show another math question to a student until 6 questions have been answered correctly." Section 3.4, later in this chapter, introduces PHP's while( ) and for( ) constructs, with which you can implement these kinds of loops.
3.1 Understanding true and false

Every expression in a PHP program has a truth value: true or false. Sometimes that truth value is important because you use it in a calculation, but sometimes you ignore it. Understanding how expressions evaluate to true or to false is an important part of understanding PHP.

Most scalar values are true. All integers and floating-point numbers (except for 0 and 0.0) are true. All strings are true except for two: a string containing nothing at all and a string containing only the character 0. These four values are false. The special constant false also evaluates to false. Everything else is true.[1]

[1] An empty array is also false. This is discussed in Chapter 4.

A variable equal to one of the five false values, or a function that returns one of those values also evaluates to false. Every other expression evaluates to true.

Figuring out the truth value of an expression has two steps. First, figure out the actual value of the expression. Then, check whether that value is true or false. Some expressions have common sense values. The value of a mathematical expression is what you'd get by doing the math with paper and pencil. For example, 7 * 6 equals 42. Since 42 is true, the expression 7 * 6 is true. The expression 5 - 6 + 1 equals 0. Since 0 is false, the expression 5 - 6 + 1 is false.

The same is true with string concatenation. The value of an expression that concatenates two strings is the new, combined string. The expression 'jacob'.'@example.com' equals the string jacob@example.com, which is true.

The value of an assignment operation is the value being assigned. The expression $price = 5 evaluates to 5, since that's what's being assigned to $price. Because assignment produces a result, you can chain assignment operations together to assign the same value to multiple variables:

    $price = $quantity = 5;

This expression means "set $price equal to the result of setting $quantity equal to 5." When this expression is evaluated, the integer 5 is assigned to the variable $quantity. The result of that assignment expression is 5, the value being assigned. Then, that result (5) is assigned to the variable $price. Both $price and $quantity are set to 5.
3.2 Making Decisions

With the if() construct, you can have statements in your program that are only run if certain conditions are true. This lets your program take different actions depending on the circumstances. For example, you can check that a user has entered valid information in a web form before letting her see sensitive data.

The if() construct runs a block of code if its test expression is true. This is demonstrated in Example 3-1.

Example 3-1. Making a decision with if()

```php
if ($logged_in) {
}
```
3.3 Building Complicated Decisions

The comparison and logical operators in PHP help you put together more complicated expressions on which an if( ) construct can decide. These operators let you compare values, negate values, and chain together multiple expressions inside one if( ) statement.

The equality operator is ==. It returns true if the two values you test with it are equal. The values can be variables or literals. Some uses of the equality operator are shown in Example 3-6.

Example 3-6. The equality operator

```php
if ($new_messages == 10) {
    ...
}
```
3.4 Repeating Yourself

When a computer program does something repeatedly, it's called looping. This happens a lot — for example, when you want to retrieve a set of rows from a database, print rows of an HTML table, or print elements in an HTML `<select>` menu. The two looping constructs discussed in this section are while( ) and for( ). Their specifics differ but they each require you to specify the two essential attributes of any loop: what code to execute repeatedly and when to stop. The code to execute is a code block just like what goes inside the curly braces after an if( ) construct. The condition for stopping the loop is a logical expression just like an if( ) construct's test expression.

The while( ) construct is like a repeating if( ). You provide an expression to while( ), just like to if( ). If the expression is true, then a code block is executed. Unlike if( ), however, while( ) checks the expression again after executing the code block. If it's still true, then the code block is executed again (and again, and again, as long as the expression is true.) Once the expression is false, program execution continues with the lines after the code block. As you have probably guessed, your code block should do something that changes the outcome of the test expression so that the loop doesn't go on forever.

Example 3-16 uses while( ) to print out an HTML form `<select>` menu with 10 choices.

Example 3-16. Printing a `<select>` menu with while( )

```bash
$i = 1;
```
3.5 Chapter Summary

Chapter 3 covers:

- Evaluating an expression's truth value: true or false.
- Making a decision with if().
- Extending if() with else.
- Extending if() with elseif().
- Putting multiple statements inside an if(), elseif(), or else code block.
- Using the equality (==) and not-equals (!=) operators in test expressions.
- Distinguishing between assignment (=) and equality comparison (==).
- Using the less-than (<), greater-than (>), less-than-or-equal-to (<=), and greater-than-or-equal-to (>=) operators in test expressions.
- Comparing two floating-point numbers with abs().
- Comparing two strings with operators.
- Comparing two strings with strcmp() or strcasecmp().
- Using the negation operator (!) in test expressions.
- Using the logical operators (&& and ||) to build more complicated test expressions.
- Repeating a code block with while().
- Repeating a code block with for().
3.6 Exercises

1. Without using a PHP program to evaluate them, determine whether each of these expressions is true or false:
   a. 100.00 - 100
   b. "zero"
   c. "false"
   d. 0 + "true"
   e. 0.000
   f. "0.0"
   g. strcmp("false","False")

2. Without running it through the PHP interpreter, figure out what this program prints.
   $age = 12;
Chapter 4. Working with Arrays

Arrays are collections of related values, such as the data submitted from a form, the names of students in a class, or the populations of a list of cities. In Chapter 2, you learned that a variable is a named container that holds a value. An array is a container that holds multiple values, each distinct from the rest.

This chapter shows you how to work with arrays. Section 4.1, next, goes over fundamentals such as how to create arrays and manipulate their elements. Frequently, you'll want to do something with each element in an array, such as print it or inspect it for certain conditions. Section 4.2 explains how to do these things with the foreach( ) and for( ) constructs. Section 4.3 introduces the implode( ) and explode( ) functions, which turn arrays into strings and strings into arrays. Another kind of array modification is sorting, which is discussed in Section 4.4. Last, Section 4.5 explores arrays that themselves contain other arrays.

Chapter 6 shows you how to process form data, which the PHP interpreter automatically puts into an array for you. When you retrieve information from a database as described in Chapter 7, that data is often packaged into an array.
4.1 Array Basics

An array is made up of elements. Each element has a key and a value. An array holding information about the colors of vegetables has vegetable names for keys and colors for values, shown in Figure 4-1.

![Figure 4-1. Keys and values](image)

An array can only have one element with a given key. In the vegetable color array, there can't be another element with the key corn even if its value is blue. However, the same value can appear many times in one array. You can have orange carrots, orange tangerines, and orange oranges.

Any string or number value can be an array element key such as corn, 4, -36, or Salt Baked Squid. Arrays and other nonscalar values can be keys, but they can be element values. An element value can be a string, a number, true, or false; it can also be another array.

[1] Scalar describes data that has a single value: a number, a piece of text, true, or false. Complex data types such as arrays, which hold multiple values, are not scalars.

4.1.1 Creating an Array

To create an array, assign a value to a particular array key. Array keys are denoted with square brackets, as shown in Example 4-1.

**Example 4-1. Creating arrays**

```javascript
// An array called $vegetables with string keys
```
4.2 Looping Through Arrays

One of the most common things to do with an array is to consider each element in the array individually and process it somehow. This may involve incorporating it into a row of an HTML table or adding its value to a running total.

The easiest way to iterate through each element of an array is with foreach(). The foreach() construct lets you run a code block once for each element in an array. Example 4-7 uses foreach() to print an HTML table containing each element in an array.

Example 4-7. Looping with foreach()

```php
$meal = array('breakfast' => 'Walnut Bun',
              'lunch'  => 'Salmon Salad',
              'dinner' => 'Stir-Fry',
              'snacks'=> array('Lunch Box', 'Peanut Butter', 'Cheese'));

foreach($meal as $key => $value) {
    echo '<tr><td>' . $key . '</td><td>' . $value . '</td></tr>' . PHP_EOL;
}
```
4.3 Modifying Arrays

You can operate on individual array elements just like regular scalar variables, using arithmetic, logical, and other operators. Example 4-17 shows some operations on array elements.

Example 4-17. Operating on array elements

```
$dishes['Beef Chow Foon'] = 12;
```
4.4 Sorting Arrays

There are several ways to sort arrays. Which function to use depends on how you want to sort your array and what kind of array it is.

The sort() function sorts an array by its element values. It should only be used on numeric arrays, because it resets the keys of the array when it sorts. Example 4-23 shows some arrays before and after sorting.

Example 4-23. Sorting with sort()

```php
$dinner = array('Sweet Corn and Asparagus',
```
4.5 Using Multidimensional Arrays

As mentioned earlier in Section 4.1, the value of an array element can be another array. This is useful when you want to store data that has a more complicated structure than just a key and a single value. A standard key/value pair is fine for matching up a meal name (such as breakfast or lunch) with a single dish (such as Walnut Bun or Chicken with Cashew Nuts), but what about when each meal consists of more than one dish? Then, element values should be arrays, not strings.

Use the array() construct to create arrays that have more arrays as element values, as shown in Example 4-27.

**Example 4-27. Creating multidimensional arrays with array()**

```php
$meals = array('breakfast' => array('Walnut Bun', 'Coffee'),
               'lunch' => array('Roast Chicken', 'Salad'));
```
4.6 Chapter Summary

Chapter 4 covers:

- Understanding the components of an array: elements, keys, and values.
- Defining an array in your programs two ways: with array( ) and with square brackets.
- Understanding the shortcuts PHP provides for arrays with numeric keys.
- Counting the number of elements in an array.
- Visiting each element of an array with foreach( ).
- Alternating table row colors with foreach( ) and an array of color values.
- Modifying array element values inside a foreach( ) code block.
- Visiting each element of a numeric array with for( ).
- Alternating table row colors with for( ) and the modulus operator (%).
- Understanding the order in which foreach( ) and for( ) visit array elements.
- Checking for an array element with a particular key.
- Checking for an array element with a particular value.
- Interpolating array element values in strings.
- Removing an element from an array.
- Generating a string from an array with implode( ).
- Generating an array from a string with explode( ).
- Sorting an array with sort( ), asort( ), or ksort( ).
- Sorting an array in reverse.
- Defining a multidimensional array.
- Accessing individual elements of a multidimensional array.
- Visiting each element in a multidimensional array with foreach( ) or for( ).
- Interpolating multidimensional array elements in a string.
4.7 Exercises

1. According to the U.S. Census Bureau, the 10 largest American cities (by population) in 2000 were as follows:
   - New York, NY (8,008,278 people)
   - Los Angeles, CA (3,694,820)
   - Chicago, IL (2,896,016)
   - Houston, TX (1,953,631)
   - Philadelphia, PA (1,517,550)
   - Phoenix, AZ (1,321,045)
   - San Diego, CA (1,223,400)
   - Dallas, TX (1,188,580)
   - San Antonio, TX (1,144,646)
   - Detroit, MI (951,270)

Define an array (or arrays) that holds this information about locations and population. Print a table of locations and population information that includes the total population in all 10 cities.

2. Modify your solution to the previous exercise so that the rows in result table are ordered by population. Then modify your solution so that the rows are ordered by city name.

3. Modify your solution to the first exercise so that the table also contains rows that hold state population totals for each state represented in the list of cities.

4. For each of the following kinds of information, state how you would store it in an array and then give sample code that creates such an array with a few elements. For example, for the first item, you might say, "An associative array whose key is the student's name and whose value is an associative array of grade and ID number," as in the following:
   $students = array('James D. McCawley' => array('grade' => 'A+','id' => 271231),
Chapter 5. Functions

When you're writing computer programs, laziness is a virtue. Reusing code you've already written makes it easier to do as little work as possible. Functions are the key to code reuse. A function is a named set of statements that you can execute just by invoking the function name instead of retyping the statements. This saves time and prevents errors. Plus, functions make it easier to use code that other people have written (as you've discovered by using the built-in functions written by the authors of the PHP interpreter).

The basics of defining your own functions and using them are laid out in Section 5.1. When you call a function, you can hand it some values with which to operate. For example, if you write a function to check whether a user is allowed to access the current web page, you would need to provide the username and the current web page name to the function. These values are called arguments. Section 5.2 explains how to write functions that accept arguments and how to use the arguments from inside the function.

Some functions are one-way streets. You may pass them arguments, but you don't get anything back. A print_header( ) function that prints the top of an HTML page may take an argument containing the page title, but it doesn't give you any information after it executes. It just displays output. Most functions move information in two directions. The access control function mentioned above is an example of this. The function gives you back a value: true (access granted) or false (access denied). This value is called the return value. You can use the return value of a function like any other value or variable. Return values are discussed in Section 5.3.

The statements inside a function can use variables just like statements outside a function. However, the variables inside a function and outside a function live in two separate worlds. The PHP interpreter treats a variable called $name inside a function and a variable called $name outside a function as two unrelated variables. Section 5.4 explains the rules about which variables are usable in which parts of your programs. It's important to understand these rules — get them wrong and your code relies on uninitialized or incorrect variables. That's a bug that is hard to track down.
5.1 Declaring and Calling Functions

To create a new function, use the function keyword, followed by the function name and then, inside curly braces, the function body. Example 5-1 declares a new function called page_header().[1]

[1] Strictly speaking, the parentheses aren't part of the function name, but it's good practice to include them when referring to functions. Doing so helps you to distinguish functions from variables and other language constructs.

Example 5-1. Declaring a function

```javascript
function page_header() {
```
5.2 Passing Arguments to Functions

While some functions (such as page_header() in the previous section) always do the same thing, other functions operate on input that can change. The input values supplied to a function are called arguments. Arguments add to the power of functions because they make functions more flexible. You can modify page_header() to take an argument that holds the page color. The modified function declaration is shown in Example 5-4.

Example 5-4. Declaring a function with an argument

```php
function page_header2($color) {
}
```
5.3 Returning Values from Functions

The header-printing function you've seen already in this chapter takes action by displaying some output. In addition to an action such as printing data or saving information into a database, functions can also compute a value, called the return value, that can be used later in a program. To capture the return value of a function, assign the function call to a variable. **Example 5-10** stores the return value of the built-in function `number_format()` in the variable `$number_to_display`.

**Example 5-10. Capturing a return value**

```php
$number_to_display = number_format(285266237);
```
5.4 Understanding Variable Scope

As you saw in Example 5-9, changes inside a function to variables that hold arguments don't affect those variables outside of the function. This is because activity inside a function happens in a different scope. Variables defined outside of a function are called global variables. They exist in one scope. Variables defined inside of a function are called local variables. Each function has its own scope.

Imagine each function is one branch office of a big company, and the code outside of any function is the company headquarters. At the Philadelphia branch office, co-workers refer to each other by their first names: "Alice did great work on this report," or "Bob never puts the right amount of sugar in my coffee." These statements talk about the folks in Philadelphia (local variables of one function), and say nothing about an Alice or a Bob who works at another branch office (local variables of another function) or at company headquarters (global variables).

Local and global variables work similarly. A variable called $dinner inside a function, whether or not it's an argument to that function, is completely disconnected from a variable called $dinner outside of the function and from a variable called $dinner inside another function. Example 5-20 illustrates the unconnectedness of variables in different scopes.

Example 5-20. Variable scope

```php
$dinner = 'Curry Cuttlefish';
```
5.5 Chapter Summary

Chapter 5 covers:

- Defining your own functions and calling them in your programs.
- Defining a function with mandatory arguments.
- Defining a function with optional arguments.
- Returning a value from a function.
- Understanding variable scope.
- Using global variables inside a function.
5.6 Exercises

1. Write a function to print out an HTML `<img>` tag. The function should accept a mandatory argument of the image URL and optional arguments for alt text, height, and width.

2. Modify the function in the previous exercise so that the filename only is passed to the function in the URL argument. Inside the function, prepend a global variable to the filename to make the full URL. For example, if you pass `photo.png` to the function, and the global variable contains `/images/`, then the `src` attribute of the printed `<img>` tag would be `/images/photo.png`. A function like this is an easy way to keep your image tags correct, even if the images move to a new path or a new server. Just change the global variable — for example, from `/images/` to `http://images.example.com/`.

3. What does the following code print out?

```php
$cash_on_hand = 31;
```
Chapter 6. Making Web Forms

Form processing is an essential component of almost any web application. Forms are how users communicate with your server: signing up for a new account, searching a forum for all the posts about a particular subject, retrieving a lost password, finding a nearby restaurant or shoemaker, or buying a book.

Using a form in a PHP program is a two-step activity. Step one is to display the form. This involves constructing HTML that has tags for the appropriate user-interface elements in it, such as text boxes, checkboxes, and buttons. If you're not familiar with the HTML required to create forms, the "Forms" chapter in HTML & XHTML: The Definitive Guide, by Chuck Musciano and Bill Kennedy (O'Reilly) is a good place to start.

When a user sees a page with a form in it, she inputs the information into the form and then clicks a button or hits Enter to send the form information back to your server. Processing that submitted form information is step two of the operation.

Example 6-1 is a page that says "Hello" to a user. If a name is submitted, then the page displays a greeting. If a name is not submitted, then the page displays a form with which a user can submit her name.

Example 6-1. Saying "Hello"

```php
if (array_key_exists('my_name',$_POST)) {
```

< Day Day Up >
6.1 Useful Server Variables

Aside from PHP_SELF, the $_SERVER auto-global array contains a number of useful elements that provide information on the web server and the current request. Table 6-1 lists some of them.

Table 6-1. Entries in $_SERVER

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_STRING</td>
<td>category=kitchen&amp;price=5</td>
<td>The part of the URL after the question mark where the URL parameters live. The example query string shown is for the URL <a href="http://www.example.com/catalog/store.php?category=kitchen&amp;price=5">http://www.example.com/catalog/store.php?category=kitchen&amp;price=5</a>.</td>
</tr>
<tr>
<td>PATH_INFO</td>
<td>/browse</td>
<td>Extra path information tacked onto the end of the URL after a slash. This is a way to pass information to a script without using the query string. The example PATH_INFO shown is for the URL <a href="http://www.example.com/catalog/store.php/browse">http://www.example.com/catalog/store.php/browse</a>.</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td><a href="http://www.example.com">www.example.com</a></td>
<td>The name of the web site on which the PHP interpreter is running. If the web server hosts many different virtual domains, this is the name of the particular virtual domain that is being accessed.</td>
</tr>
<tr>
<td>DOCUMENT_ROOT</td>
<td>/usr/local/htdocs</td>
<td>The directory on the web server computer that holds the documents available on the web site. If the document root is /usr/local/htdocs for the web site <a href="http://www.example.com">http://www.example.com</a>, then a request for <a href="http://www.example.com/catalog/store.php">http://www.example.com/catalog/store.php</a> corresponds to the file /usr/local/htdocs/catalog/store.php.</td>
</tr>
<tr>
<td>REMOTE_ADDR</td>
<td>175.56.28.3</td>
<td>The IP address of the user making the request to your web server.</td>
</tr>
</tbody>
</table>

[2] The correct spelling is HTTP_REFERRER. But it was misspelled in an early Internet specification document, so you frequently see the three-R version when web programming.
6.2 Accessing Form Parameters

At the beginning of every request, the PHP interpreter sets up some auto-global arrays that contain the values of any parameters submitted in a form or passed in the URL. URL and form parameters from GET method forms are put into $_GET. Form parameters from POST method forms are put into $_POST.

The URL http://www.example.com/catalog.php?product_id=21&category=fryingpan puts two values into $_GET: $_GET['product_id'] is set to 21 and $_GET['category'] is set to fryingpan. Submitting the form in Example 6-2 causes the same values to be put into $_POST, assuming 21 is entered in the text box and Frying Pan is selected from the menu.

Example 6-2. A two-element form
<form method="POST" action="catalog.php">
6.3 Form Processing with Functions

The basic form in Example 6-1 can be made more flexible by putting the display code and the processing code in separate functions. Example 6-6 is a version of Example 6-1 with functions.

Example 6-6. Saying "Hello" with functions

// Logic to do the right thing based on
6.4 Validating Data

Some of the validation strategies discussed in this section use regular expressions, which are powerful text-matching patterns, written in a language all their own. If you're not familiar with regular expressions, Appendix B provides a quick introduction.

Data validation is one of the most important parts of a web application. Weird, wrong, and damaging data shows up where you least expect it. Users are careless, users are malicious, and users are fabulously more creative (often accidentally) than you may ever imagine when you are designing your application. Without a Clockwork Orange-style forced viewing of a filmstrip on the dangers of unvalidated data, I can't over-emphasize how crucial it is that you stringently validate any piece of data coming into your application from an external source. Some of these external sources are obvious: most of the input to your application is probably coming from a web form. But there are lots of other ways data can flow into your programs as well: databases that you share with other people or applications, web services and remote servers, even URLs and their parameters.

As mentioned earlier, Example 6-8 doesn't indicate what's wrong with the form if the check in validate_form() fails. Example 6-9 alters validate_form() and show_form() to manipulate and print an array of possible error messages.

Example 6-9. Displaying error messages with the form

```
6.5 Displaying Default Values

Sometimes, you want to display a form with a value already in a text box or with selected checkboxes, radio buttons, or `<select>` menu items. Additionally, when you redisplay a form because of an error, it is helpful to preserve any information that a user has already entered. Example 6-23 shows the code to do this. It belongs at the beginning of `show_form()` and makes `$defaults` the array of values to use with the form elements.

Example 6-23. Building an array of defaults

```php
if ($_POST['_submit_check']) {
```

```
6.6 Putting It All Together

Turning the humble web form into a feature-packed application with data validation, printing default values, and processing the submitted results might seem like an intimidating task. To ease your burden, this section contains a complete example of a program that does it all:

- Displaying a form, including default values
- Validating the submitted data
- Redisplaying the form with error messages and preserved user input if the submitted data isn't valid
- Processing the submitted data if it is valid

The do-it-all example relies on some helper functions to simplify form element display. These are listed in Example 6-29.

Example 6-29. Form element display helper functions

```bash
//print a text box
```
6.7 Chapter Summary

Chapter 6 covers:

- Understanding the conversation between the web browser and web server that displays a form, processes the submitted form parameters, and then displays a result.
- Making the connection between the `<form>` tag's action attribute and the URL to which form parameters are submitted.
- Using values from the `$_SERVER` auto-global array.
- Accessing submitted form parameters in the `$_GET` and `$_POST` auto-global arrays.
- Accessing multivalued submitted form parameters.
- Using `show_form()`, `validate_form()`, and `process_form()` functions to modularize form handling.
- Using a hidden form element to check whether a form has been submitted.
- Displaying error messages with a form.
- Validating form elements: required elements, integers, floating-point numbers, strings, date ranges, email addresses, and `<select>` menus.
- Defanging or removing submitted HTML and JavaScript before displaying it.
- Displaying default values for form elements.
- Using helper functions to display form elements.
6.8 Exercises

1. What does $_POST look like when the following form is submitted with the third option in the Braised Noodles menu selected, the first and last options in the Sweet menu selected, and 4 entered into the text box?

```html
<form method="POST" action="order.php">
```

```php
// Code here
```

```html
</form>
```
Chapter 7. Storing Information with Databases

The HTML and CSS that give your web site its pretty face reside in individual files on your web server. So does the PHP code that processes forms and performs other dynamic wizardry. There's a third kind of information necessary to a web application, though: data. And while you can store data such as user lists and product information in individual files, most people find it easier to use databases, which are the focus of this chapter.

Lots of information falls under the broad umbrella of "data":

- Who your users are, such as their names and email addresses.
- What your users do, such as message board posts and profile information.
- The "stuff" that your site is about, such as a list of record albums, a product catalog, or what's for dinner.

There are three big reasons why this kind of data belongs in a database instead of in files: convenience, simultaneous access, and security. A database program makes it much easier to search for and manipulate individual pieces of information. With a database program, you can do things such as change the email address for user Duck29 to ducky@ducks.example.com in one step. If you put usernames and email addresses in a file, changing an email address would be much more complicated: read the old file, search through each line until you find the one for Duck29, change the line, and write the file back out. If, at same time, one request updates Duck29's email address and another updates the record for user Piggy56, one update could be lost, or (worse) the data file corrupted. Database software manages the intricacies of simultaneous access for you.

In addition to searchability, database programs usually provide you with a different set of access control options compared to files. It is an exacting process to set things up properly so that your PHP programs can create, edit, and delete files on your web server without opening the door to malicious attackers who could abuse that setup to alter your PHP scripts and data files. A database program makes it easier to arrange the appropriate levels of access to your information. It can be configured so that your PHP programs can read and change some information, but only read other information. However the database access control is set up, it doesn't affect how files on the web server are accessed. Just because your PHP program can change values in the database doesn't give an attacker an opportunity to change your PHP programs and HTML files themselves.

The word database is used in a few different ways when talking about web applications. A database can be a pile of structured information, a program (such as MySQL or Oracle) that manages that structured information, or the computer on which that program runs. In this book, I use "database" to mean the pile of structured information. The software that manages the information is a database program, and the computer that the database program runs on is a database server.

Most of this chapter uses the PEAR DB database program abstraction layer. This is an add-on to PHP that simplifies communication between your PHP program and your database program. PEAR (PHP Extension and Application Repository) is a collection of useful modules and libraries for PHP. The DB module is one of the most popular PEAR modules and is bundled with recent versions of PHP. If your PHP installation doesn't have DB installed (Section 7.2, later in this chapter, shows you how to check), see Section A.3 for instructions on how to install it.
7.1 Organizing Data in a Database

Information in your database is organized in tables, which have rows and columns. (Columns are also sometimes referred to as fields.) Each column in a table is a category of information, and each row is a set of values for each column. For example, a table holding information about dishes on a menu would have columns for each dish's ID, name, price, and spiciness. Each row in the table is the group of values for a particular dish—for example, "1," "Fried Bean Curd," "5.50," and "0" (meaning not spicy).

You can think of a table organized like a simple spreadsheet, with column names across the top, as shown in Figure 7-1.

![Figure 7-1. Data organized in a grid](image)

One important difference between a spreadsheet and a database table, however, is that the rows in a database table have no inherent order. When you want to retrieve data from a table with the rows arranged in a particular way (e.g., in alphabetic order by student name), you need to explicitly specify that order when you ask the database for the data. The SQL Lesson: ORDER BY and LIMIT sidebar in this chapter describes how to do this.

SQL (Structured Query Language) is a language to ask questions of and give instructions to the database program. Your PHP program sends SQL queries to a database program. If the query retrieves data in the database (for example, "Find me all spicy dishes"), then the database program responds with the set of rows that match the query. If the query changes data in the database (for example, "Add this new dish" or "Double the prices of all nonspicy dishes"), then the database program replies with whether or not the operation succeeded.

SQL is a mixed bag when it comes to case-sensitivity. SQL keywords are not case-sensitive, but in this book they are always written as uppercase to distinguish them from the other parts of the queries. Names of tables and columns in your queries generally are case-sensitive. All of the SQL examples in this book use lowercase column and table names to help you distinguish them from the SQL keywords. Any literal values that you put in queries are case-sensitive. Telling the database program that the name of a new dish is fried bean curd is different than telling it that the new dish is called FRIED Bean Curd.

Almost all of the SQL queries that you write to use in your PHP programs rely on one of four SQL commands: INSERT, UPDATE, DELETE, or SELECT. Each of these commands is described in this chapter. Section 7.3 describes the CREATE TABLE command, which you use to make new tables in your database.

To learn more about SQL, read SQL in a Nutshell, by Kevin E. Kline (O'Reilly). It provides an overview of standard SQL as well as the SQL extensions in MySQL, Oracle, PostgreSQL, and Microsoft SQL Server. For more in-depth information about working with PHP and MySQL, read Web Database Applications with PHP & MySQL, by Hugh E. Williams and David Lane (O'Reilly). MySQL Cookbook, by Paul DuBois (O'Reilly) is also an excellent source for answers to lots of SQL and MySQL questions.
7.2 Connecting to a Database Program

To use PEAR DB in a PHP program, first you have to load the DB module. Use the require construct, as shown in Example 7-1.

Example 7-1. Loading an external file with require

```php
require 'DB.php';
```

Example 7-1 tells the PHP interpreter to execute all of the code in the file DB.php. *DB.php* is the main file of the PEAR DB package. It defines the functions that you use to talk to your database.

Similar to require is include. These constructs differ in how they handle errors. If you try to include or require a file that doesn't exist, require considers that a fatal error and your PHP program ends. The include construct is more forgiving and just reports a warning, allowing your program to continue running.

After the DB module is loaded, you need to establish a connection to the database with the DB::connect( ) function. You pass DB::connect( ) a string that describes the database you are connecting to, and it returns an object that you use in the rest of your program to exchange information with the database program.

An object is a new data type. It's a bundle of some data and functions that operate on that data. PEAR DB uses objects to provide you with a connection to the database. The double colons in the DB::connect( ) function call are a way of telling the PHP interpreter that you're calling a special function based on an object.

Example 7-2 shows a call to DB::connect( ) that connects to MySQL.

Example 7-2. Connecting with DB::connect( )

```php
require 'DB.php';
```
7.3 Creating a Table

Before you can put any data into or retrieve any data from a database table, you must create the table. This is usually a one-time operation. You tell the database program to create a new table once. Your PHP program that uses the table may read from or write to that table every time it runs. But it doesn't have to re-create the table each time. If a database table is like a spreadsheet, then creating a table is like making a new spreadsheet file. After you create the file, you open it many times to read or change it.

The SQL command to create a table is CREATE TABLE. You provide the name of the table and the names and types of all the columns in the table. Example 7-5 shows the SQL command to create the dishes table pictured in Figure 7-1.

Example 7-5. Creating the dishes table

```sql
CREATE TABLE dishes {
```
7.4 Putting Data into the Database

Assuming the connection to the database succeeds, the object returned by DB::connect() provides access to the data in your database. Calling that object's functions lets you send queries to the database program and access the results. To put some data into the database, pass an INSERT statement to the object's query( ) function, as shown in Example 7-8.

Example 7-8. Inserting data with query( )
```
require 'DB.php';
```
7.5 Inserting Form Data Safely

As Section 6.4.6 explained, printing unsanitized form data can leave you and your users vulnerable to a cross-site scripting attack. Using unsanitized form data in SQL queries can cause a similar problem, called an "SQL injection attack." Consider a form that lets a user suggest a new dish. The form contains a text element called new_dish_name into which the user can type the name of their new dish. The call to query( ) in Example 7-26 inserts the new dish into the dishes table but is vulnerable to an SQL injection attack.

Example 7-26. Unsafe insertion of form data

```php
$db->query("INSERT INTO dishes (dish_name)
```

As Section 6.4.6 explained, printing unsanitized form data can leave you and your users vulnerable to a cross-site scripting attack. Using unsanitized form data in SQL queries can cause a similar problem, called an "SQL injection attack." Consider a form that lets a user suggest a new dish. The form contains a text element called new_dish_name into which the user can type the name of their new dish. The call to query( ) in Example 7-26 inserts the new dish into the dishes table but is vulnerable to an SQL injection attack.

Example 7-26. Unsafe insertion of form data

```php
$db->query("INSERT INTO dishes (dish_name)
```
7.6 Generating Unique IDs

As mentioned in Section 7.1, rows in a database table don't have any inherent order. In a spreadsheet, you can refer particular records such as "the first row" or "the last row" or "rows 15 to 22." A database table is different. If you want to be able to specifically identify individual records, you need to give them each a unique identifier.

To uniquely identify individual rows in a table, make a column in the table that holds an integer ID and store a different number in that column for each row. That way, even if two rows have identical values in all the other columns, you can tell them apart by using the ID column. With a dish_id column in the dishes table, you can tell apart two dishes each called "Fried Bean Curd" because the rows have different dish_id values.

PEAR DB helps you generate unique integer IDs with its support for sequences. When you ask for the next ID in a particular sequence, you get a number that you know isn't duplicated in that sequence. Even if two simultaneously executing PHP scripts ask for the next ID in a sequence at the exact same time, they each get a different ID to use.

You can have as many independent sequences as you want. To get the next value from a sequence, call the nextID( ) function. Example 7-29 gets an ID from the dishes sequence and then uses it to INSERT a row into the dishes table.

**Example 7-29. Getting an ID from a sequence**

```php
$dish_id = $db->nextID('dishes');
```
7.7 A Complete Data Insertion Form

Example 7-30 combines the database topics covered so far in this chapter with the form-handling code from Chapter 6 to build a complete program that displays a form, validates the submitted data, and then saves the data into a database table. The form displays input elements for the name of a dish, the price of a dish, and whether the dish is spicy. The information is inserted into the dishes table.

The code in Example 7-30 relies on the form helper functions defined in Example 6-29. Instead of repeating them in this example, the code assumes they have been saved into a file called formhelpers.php and then loads them with the require 'formhelpers.php' line at the top of the program.

Example 7-30. Form for inserting records into dishes
7.8 Retrieving Data from the Database

The `query()` function can also be used to retrieve information from the database. The syntax of `query()` is the same, but what you do with the object that `query()` returns is new. When it successfully completes a SELECT statement, `query()` returns an object that provides access to the retrieved rows. Each time you call the `fetchRow()` function of this object, you get the next row returned from the query. When there are no more rows left, `fetchRow()` returns a false value, making it perfect to use in a `while()` loop. This is shown in Example 7-31.

Example 7-31. Retrieving rows with `query()` and `fetchRow()`

```php
require 'DB.php';
```
7.9 Changing the Format of Retrieved Rows

So far, fetchRow(), getAll(), and getOne() have been returning rows from the database as numerically indexed arrays. This makes for concise and easy interpolation of values in double-quoted strings—but trying to remember, for example, which column from the SELECT query corresponds to element 6 in the result array can be difficult and error-prone. PEAR DB lets you specify that you'd prefer to have each result row delivered as either an array with string keys or as an object.

The fetch mode controls how result rows are formatted. The setFetchMode() function changes the fetch mode. Any queries in a page after you call setFetchMode() have their result rows formatted as specified by the argument to setFetchMode().

To get result rows as arrays with string keys, pass DB_FETCHMODE_ASSOC to setFetchMode(). Note that DB_FETCHMODE_ASSOC is a special constant defined by PEAR DB, not a string, so you shouldn't put quotes around it. The array keys in the result row arrays correspond to column names. Example 7-46 shows how to use fetchRow(), getAll(), and getRow() with string-keyed result rows.

Example 7-46. Retrieving rows as string-keyed arrays

```php
require 'DB.php';
```
7.10 Retriving Form Data Safely

It's possible to use placeholders with SELECT statements just as you do with INSERT, UPDATE, or DELETE statements. The `getAll()`, `getRow()`, and `getOne()` functions each accept a second argument of an array of values that are substituted for placeholders in a query.

However, when you use submitted form data or other external input in the WHERE clause of a SELECT, UPDATE, or DELETE statement, you must take extra care to ensure that any SQL wildcards are appropriately escaped. Consider a search form with a text element called `dish_search` into which the user can type a name of a dish he's looking for. The call to `getAll()` in Example 7-48 uses placeholders guard against confounding single-quotes in the submitted value.

Example 7-48. Using a placeholder in a SELECT statement

```php
$matches = $db->getAll('SELECT dish_name, price FROM dishes
```
7.11 A Complete Data Retrieval Form

Example 7-56 is another complete database and form program. It presents a search form and then prints an HTML table of all rows in the dishes table that match the search criteria. Like Example 7-30, it relies on the form helper functions being defined in a separate formhelpers.php file.

Example 7-56. Form for searching the dishes table
<?php
7.12 MySQL Without PEAR DB

PEAR DB smooths over a lot of the rough edges of database access in a PHP program, but there are two reasons why it's not always the right choice: PEAR DB might not be available on some systems, and a program that uses the built-in PHP functions tailored to a particular database is faster than one that uses PEAR DB. Programmers who don't anticipate switching or using more than one database program often pick those built-in functions.

The basic model of database access with the built-in functions is the same as with PEAR DB. You call a function that connects to the database. It returns a variable that represents the connection. You use that connection variable with other functions to send queries to the database program and retrieve the results.

The differences are in the details. The applicable functions and how they work differ from database to database. In general, you have to retrieve results one row at a time instead of the convenience that `getAll()` offers, and there is no unified error handling.

As an example for database access without PEAR DB, this section discusses the mysqli extension, which works with MySQL 4.1.2 or greater and with PHP 5. There are similar PHP extensions for other database programs. Table 7-4 lists the database programs that PHP supports and where in the PHP Manual you can read about the functions in the extension for each database. All of the extensions listed in Table 7-4 are not usually installed by default with the PHP interpreter, but the PHP Manual gives instructions on how to install them.

<table>
<thead>
<tr>
<th>Database program</th>
<th>PHP Manual URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adabas D</td>
<td><a href="http://www.php.net/uodbc">http://www.php.net/uodbc</a></td>
</tr>
<tr>
<td>DB2</td>
<td><a href="http://www.php.net/uodbc">http://www.php.net/uodbc</a></td>
</tr>
<tr>
<td>DB++</td>
<td><a href="http://www.php.net/dbplus">http://www.php.net/dbplus</a></td>
</tr>
<tr>
<td>Empress</td>
<td><a href="http://www.php.net/uodbc">http://www.php.net/uodbc</a></td>
</tr>
<tr>
<td>FrontBase</td>
<td><a href="http://www.php.net/fbsql">http://www.php.net/fbsql</a></td>
</tr>
<tr>
<td>Informix</td>
<td><a href="http://www.php.net/ifx">http://www.php.net/ifx</a></td>
</tr>
<tr>
<td>InterBase</td>
<td><a href="http://www.php.net/ibase">http://www.php.net/ibase</a></td>
</tr>
</tbody>
</table>

Table 7-4. Database extensions

This section doesn't explore the mysqli functions in great detail but shows how to use mysqli to do some of the things you've already seen with PEAR DB. Chapter 3 of Upgrading to PHP 5, by Adam Trachtenberg (O'Reilly) covers the ins and outs of mysqli, including advanced features such as secure connections, parameter binding, and result buffering. Examples Example 7-57 and Example 7-58 contain the necessary changes to Example 7-56 so that it uses PHP's mysqli extension instead of PEAR DB.

The two sections of the program that need to be changed are the top-level database connection code, which is shown in Example 7-57 and the `process_form()` function, which is shown in Example 7-58.
7.13 Chapter Summary

Chapter 7 covers:

- Figuring out what kinds of information belong in a database.
- Understanding how data is organized in a database.
- Loading an external file with require.
- Establishing a database connection.
- Creating a table in the database.
- Removing a table from the database.
- Using the SQL INSERT command.
- Inserting data into the database with query().
- Checking for database errors with DB::isError().
- Setting up automatic error handling with setErrorHandling().
- Using the SQL UPDATE and DELETE commands.
- Changing or deleting data with query().
- Counting the number of rows affected by a query.
- Using placeholders to insert data safely.
- Generating unique ID values with sequences.
- Using the SQL SELECT command.
- Retrieving data from the database with query() and findRow().
7.14 Exercises

The following exercises use a database table called dishes with the following structure:

```sql
CREATE TABLE dishes {
```
Chapter 8. Remembering Users with Cookies and Sessions

A web server is a lot like a clerk at a busy deli full of pushy customers. The customers at the deli shout requests: "I want a half pound of corned beef!" and "Give me a pound of pastrami, sliced thin!" The clerk scurries around slicing and wrapping to satisfy the requests. Web clients electronically shout requests ("Give me /catalog/yak.php!" or "Here's a form submission for you!"), and the server, with the PHP interpreter's help, electronically scurries around constructing responses to satisfy the requests.

The clerk has an advantage that the web server doesn't, though: a memory. She naturally ties together all the requests that come from a particular customer. The PHP interpreter and the web server can't do that without some extra steps. That's where cookies come in.

A cookie identifies a particular web client to the web server and to the PHP interpreter. Each time a web client makes a request, it sends the cookie along with the request. The interpreter reads the cookie and figures out that a particular request is coming from the same web client that made previous requests, which were accompanied by the same cookie.

If deli customers were faced with a memory-deprived clerk, they'd have to adopt the same strategy. Their requests for service would look like this:

"I'm customer 56 and I want a half-pound of corned beef."
8.1 Working with Cookies

To set a cookie, use the `setcookie()` function. This tells a web client to remember a cookie name and value and send them back to the server on subsequent requests. **Example 8-1** sets a cookie named `userid` to value `ralph`.

**Example 8-1. Setting a cookie**
```
setcookie('userid','ralph');
```

To read a previously set cookie from your PHP program, use the `$_COOKIE` auto-global array. **Example 8-2** prints the value of the `userid` cookie.

**Example 8-2. Printing a cookie value**
```
print 'Hello, ' . $_COOKIE['userid'];
```

The value for a cookie that you provide to `setcookie()` can be a string or a number. It can't be an array or more complicated data structure.

When you call `setcookie()`, the response that the PHP interpreter generates to send back to the web client includes a special header that tells the web client about the new cookie. On subsequent requests, the web client sends that cookie name and value back to the server. This two-step conversation is illustrated in **Figure 8-1**.

![Figure 8-1. Client and server communication when setting a cookie](image)

Usually, you must call `setcookie()` before the page generates any output. This means that `setcookie()` must come before any print statements. It also means that there can't be any text before the PHP `<?php` start tag in the page that comes before the `setcookie()` function. Later in this chapter, **Section 8.6** explains why this requirement exists, and how, in some cases, you can get around it.

**Example 8-3** shows the correct way to put a `setcookie()` call at the top of your page.

**Example 8-3. Starting a page with setcookie()**
```
<?php
```
8.2 Activating Sessions

Sessions use a cookie called PHPSESSID. When you start a session on a page, the PHP interpreter checks for the presence of this cookie and sets it if it doesn't exist. The value of the PHPSESSID cookie is a random alphanumeric string. Each web client gets a different session ID. The session ID in the PHPSESSID cookie identifies that web client uniquely to the server. That lets the interpreter maintain separate piles of data for each web client.

The conversation between the web client and the server when starting up a session is illustrated in Figure 8-2.

Figure 8-2. Client and server communication when starting a session

To use a session in a page, call session_start( ) at the beginning of your script. Like setcookie( ), this function must be called before any output is sent. If you want to use sessions in all your pages, set the configuration directive session.auto_start to On. (Appendix A explains how to change configuration settings.) Once you do that, there's no need to call session_start( ) in each page.
8.3 Storing and Retrieving Information

Session data is stored in the $_SESSION auto-global array. Read and change elements of that array to manipulate the session data. Example 8-9 shows a page counter that uses the $_SESSION array to keep track of how many times a user has looked at the page.

Example 8-9. Counting page accesses with a session

```php
session_start();
```
8.4 Configuring Sessions

Sessions work great with no additional tweaking. Turn them on with the session_start() function or the session.auto_start configuration directive, and the $_SESSION array is there for your enjoyment. However, if you're more particular about how you want sessions to function, there are a few helpful settings that can be changed.

Session data sticks around as long as the session is accessed at least once every 24 minutes. This is fine for most applications. Sessions aren't meant to be a permanent data store for user information—that's what the database is for. Sessions are for keeping track of recent user activity to make their browsing experience smoother.

Some situations may need a shorter session length, however. If you're developing a financial application, you may want to allow only 5 or 10 minutes of idle time to reduce the chance that an unattended computer can be used by an unauthorized person. If your application doesn't work with very critical data and you have easily distracted users, you may want to set the session length to longer than 24 minutes.

The session.gc_maxlifetime configuration directive controls how much idle time is allowed between requests to keep a session active. It's default value is 1,440—there are 1,440 seconds in 24 minutes. You can change session.gc_maxlifetime in your server configuration or by calling the ini_set() function from your program. If you use ini_set(), you must call it before session_start(). Example 8-12 shows how to use ini_set() to change the allowable session idle time to 10 minutes.

Example 8-12. Changing allowable session idle time

```php
<!-- Example 8-12 -->
```
8.6 Why setcookie( ) and session_start( ) Want to Be at the Top of the Page

When a web server sends a response to a web client, most of that response is the HTML document that the browser renders into a web page on your screen: the soup of tags and text that Internet Explorer or Mozilla formats into tables or changes the color or size of. But before that HTML is a section of the response that contains headers. These don't get displayed on your screen but are commands or information from the server for the web client. The headers say things such as "this page was generated at such-and-such a time," "please don't cache this page," or (and the one that's relevant here) "please remember that the cookie named userid has the value ralph."

All of the headers in the response from the web server to the web client have to be at the beginning of the response, before the response body, which is the HTML that controls what the browser actually displays. Once some of the body is sent—even one line—no more headers can be sent.

Functions such as setcookie( ) and session_start( ) add headers to the response. In order for the added headers to be sent properly, they must be added before any output starts. That's why they must be called before any print statements or any HTML appearing outside <?php ?> PHP tags.

If any output has been sent before setcookie( ) or session_start( ) is called, the PHP interpreter prints an error message that looks like this:

Warning: Cannot modify header information - headers already sent by
8.7 Chapter Summary

Chapter 8 covers:

- Understanding why cookies are necessary to identify a particular web browser to a web server.
- Setting a cookie in a PHP program.
- Reading a cookie value in a PHP program.
- Modifying cookie parameters such as expiration time, path, and domain.
- Deleting a cookie in a PHP program.
- Turning on sessions from a PHP program or in the PHP interpreter configuration.
- Storing information in a session.
- Reading information from a session.
- Saving form data in a session.
- Removing information from a session.
- Configuring session expiration and cleanup.
- Displaying, validating, and processing a validation form.
- Using encrypted passwords.
- Understanding why setcookie() and session_start() must be called before anything is printed.
8.8 Exercises

1. Make a web page that uses a cookie to keep track of how many times a user has viewed the page. The first time a particular user looks at the page, it should print something like "Number of views: 1." The second time the user looks at the page, it should print "Number of views: 2," and so on.

2. Modify the web page from the first exercise so that it prints out a special message on the 5th, 10th, and 15th time the user looks at the page. Also modify it so that on the 20th time the user looks at the page, it deletes the cookie and the page count starts over.

3. Write a PHP program that displays a form for a user to pick their favorite color from a list of colors. Make another page whose background color is set to the color that the user picks in the form. Store the color value in $_SESSION so that both pages can access it.

4. Write a PHP program that displays an order form. The order form should list six products. Next to each product name there should be a text box into which a user can type in how many of that product they want to order. When the form is submitted, the submitted form data should be saved into the session. Make another page that displays the contents of the saved order, a link back to the order form page, and a Check Out button. If the link back to the order form page is clicked, the order form page should be displayed with the saved order quantities from the session in the text boxes. When the Check Out button is clicked, the order should be cleared from the session.
Chapter 9. Handling Dates and Times

Dates and times are all over the place in a web application. In a shopping cart, you need to handle shipping dates of products. In a forum, you need to keep track of when messages are posted. In all sorts of applications, you need to keep track of the last time a user logged in so that you can tell them things such as "fifteen new messages were posted since you last logged in."

Handling dates and times properly in your programs is more complicated than handing strings or numbers. A date or a time is not a single value but a collection of values—month, day, and year, for example, or hour, minute, and second. Because of this, doing math with them can be tricky. Instead of just adding or subtracting entire dates and times, you have to consider their component parts and what the allowable values for each part are. Hours go up to 12 (or 24), minutes and seconds go up to 59, and not all months have the same number of days.

A programming convention that simplifies date and time calculation is to treat a particular time and date as a single value: the number of seconds that have elapsed since midnight on January 1, 1970. This value is called an epoch timestamp. The choice of January 1, 1970 is mostly arbitrary. But, as is the way with conventions, since lots of other people are doing it, you've got to do it, too. Fortunately, PHP provides plenty of functions for you to deal with epoch timestamps.

In this book, the phrase time parts (or date parts or time and date parts) means an array or group of time and date components such as day, month, year, hour, minute, and second. Formatted time string (or formatted date string, etc.) means a string that contains some particular grouping of time and date parts—for example "Wednesday, October 20, 2004" or "3:54 p.m."
9.1 Displaying the Date or Time

The simplest display of date or time is telling your users what time it is. Use the date( ) or strftime( ) function as shown in Example 9-1.

Example 9-1. What time is it?

```python
print 'strftime( ) says: ';
```
9.2 Parsing a Date or Time

To work with date or time values in your program as epoch timestamps, you need to convert other time representations to epoch timestamps. If you have discrete date or time parts (for example, from different form parameters), then use mktime(). It accepts an hour, minute, second, month, day, and year, and returns the corresponding epoch timestamp. Example 9-6 shows mktime() at work.

Example 9-6. Making an epoch timestamp

// get the values from a form
9.3 Dates and Times in Forms

When you need a user to input a date in a form, the best thing to do is to use \(<select>\) menus. This generally restricts the possible input to whatever you display in the menus. The specific date or time information you need controls what you populate the \(<select>\) menus with.

9.3.1 A Single Menu with One Choice Per Day

If there are a small number of choices, you can have just one menu that lists all of them. Example 9-9 prints a \(<select>\) menu that lets a user pick one day in the coming week. The value for each option in the menu is an epoch timestamp corresponding to midnight on the displayed day.

Example 9-9. A day choice \(<select>\) menu

```perl
$midnight_today = mktime(0,0,0);
```
9.4 Displaying a Calendar

This section puts the date and time functions to work in displaying a calendar. The show_form( ) function in Example 9-17 displays a form that asks for a month and year. The process_form( ) function hands those values off to the show_calendar( ) function, which does the real work of printing a calendar grid for a particular month.

The structure of the if( ) statement that controls show_form( ), validate_form( ), and process_form( ) is different in Example 9-17 than in previous form examples. That's because we want to display the form above the calendar. Usually, if the form data is valid, show_form( ) is not called—only process_form( ) is. But here, show_form( ) is called before process_form( ) so that the form is displayed above the calendar and the user can pick another month and year to view.

Similarly, the call to show_form( ) that happens when the form has not been submitted (when there is no $POST['submit_check'] parameter) is followed by a call to show_calendar( ) to display the calendar for the current month the first time the page is loaded.

Example 9-17. Printing a calendar

```php
<?php
```
9.5 Chapter Summary

Chapter 9 covers:

• Defining some time- and date-handling vocabulary such as epoch timestamp, time and date parts, and formatted time and date string.

• Printing formatted time and date strings with strftime( ) and date( ).

• Making an epoch timestamp with mktime( ).

• Making an epoch timestamp with strtotime( ).

• Displaying form elements to allow for date or time input.

• Doing calculations with a date or time submitted in a form.

• Displaying a calendar.
9.6 Exercises

1. Use strftime() to print a formatted time and date string that looks like this: 

   Today is day 20 of October and day 294 of the year 2004. The time is 07:45 PM
Chapter 10. Working with Files

The data storage destination of choice for a web application is a database. That doesn't mean that you're completely off the hook from dealing with regular old files, though. Plain text files are still a handy, universal way to exchange some kinds of information.

You can do some easy customization of your web site by storing HTML templates in text files. When it's time to generate a specialized page, load the text file, substitute real data for the template elements, and print it. Example 10-1 shows you how to do this.

Files are also good for importing or exporting tabular data between your program and a spreadsheet. In your PHP programs, you can easily read and write the CSV ("comma-separated value") files with which spreadsheet programs work.

Working with files in PHP also means working with remote web pages. A great thing about file handling in PHP is you can open a remote file on another computer as easily as you can open a file that sits on your web server. Most file-handling functions in PHP understand URLs as well as local filenames. However, for this feature to work, the allow_url_fopen configuration directive must be enabled. It is enabled by default, but if you're having problems loading a remote file, check this setting.
10.1 Understanding File Permissions

To read or write a file with any of the functions you'll learn about in this chapter, the PHP interpreter must have permission from the operating system to do so. Every program that runs on a computer, including the PHP interpreter, runs with the privileges of a particular user account. Most of the user accounts correspond to people. When you log in to your computer and start up your word processor, that word processor runs with the privileges that correspond to your account: it can read files that you are allowed to see and write files that you are allowed to change.

Some user accounts on a computer, however, aren't for people, but for system processes such as web servers. When the PHP interpreter runs inside of a web server, it has the privileges that the web server's "account" has. So if the web server is allowed to read a certain file or directory, then the PHP interpreter (and therefore your PHP program) can read that file or directory. If the web server is allowed to change a certain file or write new files in a particular directory, then so can the PHP interpreter and your PHP program.

Usually, the privileges extended to a web server's account are more limited than the privileges that go along with a real person's account. The web server (and the PHP interpreter) need to be able to read all of the PHP program files that make up your web site, but they shouldn't be able to change them. If a bug in the web server or an insecure PHP program lets an attacker break in, the PHP program files should be protected against being changed by that attacker.

In practice, what this means is that your PHP programs shouldn't have too much trouble reading most files that you need to read. (Of course, if you try to read another user's private files, you may run into a problem—but that's as it should be!) However, the files that your PHP program can change and the directories into which your program can write new files are limited. If you need to create lots of new files in your PHP programs, work with your system administrator to make a special directory that you can write to but that doesn't compromise system security. Section 10.5, later in this chapter, shows you how to determine what files and directories your programs are allowed to read and write.
10.2 Reading and Writing Entire Files

This section shows you how to work with an entire file at once, as opposed to manipulating just a few lines of a file. PHP provides special functions for reading or writing a whole file in a single step.

10.2.1 Reading a File

To read the contents of a file into a string, use `file_get_contents()` . Pass it a filename, and it returns a string containing everything in the file. Example 10-1 reads the file in Example 10-2 with `file_get_contents()` , modifies it with `str_replace()` , and then prints the result.

**Example 10-1. Using file_get_contents() with a page template**

```php
// Load the file from Example 10.2
```

Example 10-1. Using file_get_contents() with a page template

```php
// Load the file from Example 10.2
```
10.3 Reading and Writing Parts of Files

The `file_get_contents()` and `file_put_contents()` functions are fine when you want to work with an entire file at once. But when it's time for precision work, use other functions to deal with a file a line at a time. **Example 10-6** reads a file in which each line contains a name and an email address and then prints an HTML-formatted list of that information.

**Example 10-6. Reading a file a line at a time**

```php
$fh = fopen('people.txt','rb');
```
10.4 Working with CSV Files

One type of text file gets special treatment in PHP: the CSV file. It can't handle graphs or charts, but excels for sharing tables of data among different programs. To read a line of a CSV file, use fgetcsv( ) instead of fgets( ). It reads a line from the CSV file and returns an array containing each field in the line. Example 10-10 is a CSV file of information about restaurant dishes. Example 10-11 uses fgetcsv( ) to read the file and insert the information in it into the dishes database table from Chapter 7.

Example 10-10. dishes.csv for Example 10-11
"Fish Ball with Vegetables",4.25,0
10.5 Inspecting File Permissions

As mentioned at the beginning of the chapter, your programs can only read and write files when the PHP interpreter has permission to do so. You don't have to cast about blindly and rely on error messages to figure out what those permissions are, however. PHP gives you functions with which you can determine what your program is allowed to do.

To check whether a file or directory exists, use `file_exists()`.

Example 10-16 uses this function to report whether a directory's index file has been created.

Example 10-16. Checking the existence of a file
```php
if (file_exists('/usr/local/htdocs/index.html')) {
```
10.6 Checking for Errors

So far, the examples in this chapter have been shown without any error checking in them. This keeps them shorter, so you can focus on the file manipulation functions such as file_get_contents(), fopen(), and fgetcsv(). It also makes them somewhat incomplete. Just like talking to a database program, working with files means interacting with resources external to your program. This means you have to worry about all sorts of things that can cause problems, such as operating system file permissions or a disk running out of free space.

In practice, to write robust file-handling code, you should check the return value of each file-related function. They each generate a warning message and return false if there is a problem. If the configuration directive track_errors is on, the text of the error message is available in the global variable $php_errormsg.

Example 10-19 shows how to check whether fopen() or fclose() encounters an error.

Example 10-19. Checking for an error from fopen() or fclose()

```php
require 'DB.php';
```
10.7 Sanitizing Externally Supplied Filenames

Just like data submitted in a form or URL can cause problems when it is displayed (cross-site scripting attack) or put in an SQL query (SQL injection attack), it can also cause problems when it is used as a filename or as part of a filename. It doesn't have a fancy name like those other attacks, but it can be just as devastating.

The cause of the problem is the same: there are special characters that must be escaped so they lose their special meaning. In filenames, the special characters are / (which separates parts of filenames), and the two-character sequence .. (which means "go up one directory" in a filename).

For example, the funny-looking filename /usr/local/data/../../../etc/passwd doesn't point to a file under the /usr/local/data directory but instead to the file /etc/passwd, which, on most Unix systems, contains a list of user accounts. The filename /usr/local/data/../../../etc/passwd means "from the directory /usr/local/data, go up one level (to /usr/local), then go up another level (to /usr), then go up another level (to /, the top level of the filesystem), then down into /etc, then stop at the file passwd."

How could this be a problem in your PHP programs? When you use data from a form in a filename, you are vulnerable to this sort of attack unless you sanitize that submitted form data. Example 10-23 takes the approach of removing all forward slashes and .. sequences from a submitted form parameter before incorporating the parameter into a filename.

Example 10-23. Cleaning up a form parameter that goes in a filename

```php
// Remove slashes from user
```
10.8 Chapter Summary

Chapter 10 covers:

- Understanding where the PHP interpreter's file access permissions come from.
- Reading entire local and remote files with file_get_contents().
- Writing entire local and remote files with file_put_contents().
- Opening and closing files with fopen() and fclose().
- Reading a line of a file with fgets().
- Using feof() and a for() loop to read each line in a file.
- Using forward slashes in filenames with all operating systems.
- Providing different file modes to fopen().
- Writing data to a file with fwrite().
- Reading a line of a CSV file with fgetcsv().
- Determining whether a file exists with file_exists().
- Inspecting file permissions with is_readable() and is_writeable().
- Checking for errors returned from file access functions.
- Understanding when to check a return value with the identical operator (===).
- Removing potentially dangerous parts of externally supplied filenames.
10.9 Exercises

1. Outside of the PHP interpreter, create a new template file in the style of Example 10-2. Use file_get_contents() and file_put_contents() to read an HTML template file, substitute values for the template variables, and save the new page to a separate file.

2. Outside of the PHP interpreter, create a file that contains some email addresses, one per line. Make sure a few of the addresses appear more than once in the file. Call that file addresses.txt. Then, write a PHP program that reads each line in addresses.txt and counts how many times each address appears. For each distinct address in addresses.txt, your program should write a line to another file, addresses-count.txt. Each line in addresses-count.txt should consist of the number of times an address appears in addresses.txt, a comma, and the email address. Write the lines to addresses-count.txt in sorted order from the address that occurs the most times in addresses.txt to the address that occurs the fewest times in addresses.txt.

3. Display a CSV file as an HTML table. If you don't have a CSV file (or spreadsheet program) handy, use the data from Example 10-10.

4. Write a PHP program that displays a form that asks a user for the name of a file underneath the web server's document root directory. If that file exists on the server, is readable, and is underneath the web server's document root directory, then display the contents of the file. For example, if the user enters article.html, display the file article.html in the document root directory. If the user enters catalog/show.php, display the file show.php in the directory catalog under the document root directory. Table 6-1 tells you how to find the web server's document root directory.

5. Modify your solution to the previous exercise so that the program displays only files whose names end in .html. Letting users look at the PHP source code of any page on your site can be dangerous if those pages have sensitive information in them such as database usernames and passwords.
Chapter 11. Parsing and Generating XML

With XML, you can effortlessly exchange data between programs written in different languages, running on different operating systems, located on computers anywhere in the world. At least, that's what enthusiastic computer programmers and salespeople who work for companies that sell XML tools will tell you. They're sort of telling the truth. XML does make it easier to trade structured information between two programs. But you still have to do some work to herd your data into the right structure. This chapter shows you how to do that work with PHP.

XML is a markup language that looks a lot like HTML. An XML document is plain text and contains tags delimited by < and >. There are two big differences between XML and HTML:

- XML doesn't define a specific set of tags you must use.
- XML is extremely picky about document structure.

In one sense, XML gives you a lot more freedom than HTML. HTML has a certain set of tags: the <a></a> tags surround a link, the <ul></ul> tags denote an unordered list, the <li></li> tags indicate a list element, and so on. An XML document, however, can use any tags you want. Put <rating></rating> tags around a movie rating, <height></height> tags around someone's height, or <favoritecolor></favoritecolor> tags around someone's favorite color—XML doesn't care. Of course, whomever (or whatever program) you're sharing the XML document with also needs to agree to use and understand the same set of tags.

While you get more freedom in the tag-choice department, XML clamps down much harder than HTML when it comes to document structure. HTML lets you play fast and loose with some opening and closing tags. The HTML list in Example 11-1 renders just fine in a web browser.

Example 11-1. HTML list that's not valid XML

```xml
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```
11.1 Parsing an XML Document

PHP 5's new SimpleXML module makes parsing an XML document, well, simple. It turns an XML document into an object that provides structured access to the XML.

To create a SimpleXML object from an XML document stored in a string, pass the string to simplexml_load_string(). It returns a SimpleXML object. In Example 11-3, $channel holds XML that represents the <channel> part of an RSS 0.91 feed.

Example 11-3. Parsing XML in a string

```php
$channel =<<<XML
< Day Day Up >
```
11.2 Generating an XML Document

SimpleXML is good for parsing existing XML documents, but you can't use it to create a new one from scratch. For many XML documents, the easiest way to generate them is to build a PHP array whose structure mirrors that of the XML document and then to iterate through the array, printing each element with appropriate formatting.

Example 11-17 generates the XML for the channel part of an RSS feed using the information in the $channel array.

Example 11-17. Generating XML from an array

```php
$channel = array('title' => "What's For Dinner",
```
11.3 Chapter Summary

Chapter 11 covers:

- Understanding the basic differences between XML and HTML.
- Creating a SimpleXML object from a string that contains XML.
- Printing XML element contents with a SimpleXML object.
- Printing XML element attributes with a SimpleXML object.
- Accessing identically named elements with a SimpleXML object.
- Looping through a SimpleXML object with foreach().
- Changing elements and attributes in a SimpleXML object.
- Printing a SimpleXML object as an XML document.
- Sending a Content-Type header to indicate an XML document.
- Creating a SimpleXML object from a local or remote file that contains XML.
- Saving a SimpleXML object to a file as an XML document.
- Generating an XML document from a PHP array.
- Generating an XML document from information in a database table.
11.4 Exercises

1. Using the XML document in the $menu variable defined in Example 11-5, print an HTML <ul> list in which each list element is the <title> of one <item> in the XML document, and that <title> is hyperlinked to the URL listed in the <link> element of the item. For example, if one of the items were:

```xml
<item>
  <title>Day Day Up</title>
  <link>http://example.com</link>
</item>
```
Chapter 12. Debugging

Programs rarely work correctly the first time. This chapter shows you some techniques for finding and fixing the problems in your programs. When you're just learning PHP, your programs are probably simpler than the programs that PHP wizards write. The errors you get, however, generally aren't much simpler, and you have to use the same tools and techniques to find and fix those errors.
12.1 Controlling Where Errors Appear

Many things can go wrong in your program that cause the PHP interpreter to generate an error message. You have a choice about where those error messages go. The messages can be sent along with other program output to the web browser. They can also be included in the web server error log.

A useful way to configure an error message display is to have the errors displayed on screen when you're developing a PHP program, and then sent to the error log once you're done development and people are actually using the program. While you're working on a program, it's helpful to see immediately that there was a parse error on a particular line, for example. But once the program is (supposedly) working so that your coworkers or customers can use it, such an error message would be confusing to them.

To make error messages display in the browser, set the `display_errors` configuration directive to On. To send errors to the web server error log, set `log_errors` to On. You can set them both to On if you want error messages in both places.

An error message that the PHP interpreter generates falls into one of five different categories:

**Parse error**

A problem with the syntax of your program, such as leaving a semicolon off of the end of a statement. The interpreter stops running your program when it encounters a parse error.

**Fatal error**

A severe problem with the content of your program, such as calling a function that hasn't been defined. The interpreter stops running your program when it encounters a fatal error.

**Warning**

An advisory from the interpreter that something is fishy in your program, but the interpreter can keep going. Using the wrong number of arguments when you call a function causes a warning.

**Notice**

A tip from the PHP interpreter playing the role of Miss Manners. For example, printing a variable without first initializing it to some value generates a notice.

**Strict notices**

An admonishment from the PHP interpreter about your coding style. Most of these have to do with esoteric features that were available in PHP 4 and PHP 5, so you're unlikely to encounter them any more.

Appendix A explains what the & and ~ mean in configuration directive values.

PHP defines some constants you can use to set the value of error_reporting such that only errors of certain types get reported: `E_ALL` (for all errors except strict notices), `E_PARSE` (parse errors), `E_ERROR` (fatal errors), `E_WARNING` (warnings), `E_NOTICE` (notices), and `E_STRICT` (strict notices).

Because strict notices are rare (and new to PHP 5), they are not included in `E_ALL`. To tell the PHP interpreter that you want to hear about everything that could possibly be an error, set `error_reporting` to `E_ALL | E_STRICT`. 
12.2 Fixing Parse Errors

The PHP interpreter is really picky but not very chatty. If you leave out a necessary semicolon, or start a string with a single quote but end it with a double quote, the interpreter doesn't run your program. It throws up its (virtual) hands, complains about a "parse error," and leaves you stuck in the debugging wilderness.

This can be one of the most frustrating things about programming when you're getting started. Everything has to be phrased and punctuated just so in order for the PHP interpreter to accept it. One thing that helps this process along is writing your programs in an editor that is PHP-aware. This is a program that, when you tell it you are editing a PHP program, turns on some special features that make programming easier.

One of these special features is syntax highlighting. It changes the color of different parts of your program based on what those parts are. For example, strings are pink, keywords such as if and while are blue, comments are grey, and variables are black. Syntax highlighting makes it easier to detect things such as a string that's missing its closing quote: the pink text continues past the line that the string is on, all the way to the end of the file (or the next quote that appears later in the program).

Another feature is quote and bracket matching, which helps to make sure that your quotes and brackets are balanced. When you type a closing delimiter such as }, the editor highlights the opening { that it matches. Different editors do this in different ways, but typical methods are to flash the cursor at the location of the opening {, or to bold the { } pair for a short time. This behavior is helpful for pairs of punctuation that go together: single and double quotes that delimit strings, parentheses, square brackets, and curly braces.

These editors also show the line numbers of your program files. When you get an error message from the PHP interpreter complaining about a parse error in line 35 in your program, you can focus on the right place to look for your error.

Table 12-1 lists seven PHP-aware editors. Some of them go beyond the basics of syntax highlighting and bracket matching and provide more advanced features to help your coding. These features are listed in the "Comments" column of the table.

Table 12-1. PHP-aware text editors

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform(s)</th>
<th>URL</th>
<th>Cost</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emacs and XEmacs</td>
<td>All</td>
<td><a href="http://www.gnu.org/software/emacs/">http://www.gnu.org/software/emacs/</a></td>
<td>Free</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://www.xemacs.org">http://www.xemacs.org</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
12.3 Inspecting Program Data

Once you clear the parse error hurdle, you still may have some work to do before you reach the finish line. A program can be syntactically correct but logically flawed. Just as the sentence "The tugboat chewed apoplectically with six subtle buffaloes" is grammatically correct but meaningless nonsense, you can write a program that the PHP interpreter doesn't find any problems with but doesn't do what you expect.

If your program is acting funny, add some checkpoints that display the values of variables. That way, you can see where the program's behavior diverges from your expectations. Example 12-3 shows a program that incorrectly attempts to calculate the total cost of a few items.

Example 12-3. A broken program without debugging output

```php
$prices = array(5.95, 3.00, 12.50);
```
12.4 Fixing Database Errors

When your program involves talking to a database, you have to deal with an additional universe of errors. Just as the PHP interpreter expects your programs to adhere to a particular grammar, the database program expects your SQL statements to adhere to the grammar of SQL.

The setErrorHandling( ) function introduced in Section 7.4 has an additional mode of operation that gives you increased control over how database errors are handled in your PHP programs. Instead of having a terse error message printed or your program exit when a database error happens, you can have a custom function called. That function can do whatever you want, such as print a more detailed error message or write to the web server error log.

To enable this mode, call setErrorHandling( ) with the PEAR_ERROR_CALLBACK constant and the name of your error-handling function. Example 12-8 says that when there is a database error, the database_error( ) function should be called.

Example 12-8. Setting up a custom database error handling function

```php
$db->setErrorHandling(PEAR_ERROR_CALLBACK,'database_error');
```

You also have to write the custom error-handling function whose name is passed to setErrorHandling( ). This function must accept one argument. When DB invokes the function, it passes an object to the function that contains the error information. You can use the getDebugInfo( ) method of that object to get more detailed error information. Example 12-9 is a sample custom error-handling function.

Example 12-9. A custom database error handling function

```php
function database_error($error_object) {
}```
12.5 Chapter Summary

Chapter 12 covers:

- Configuring error display for a web browser, a web server error log, or both.
- Configuring the PHP interpreter's error-reporting level.
- Getting the benefits of a PHP-aware text editor.
- Deciphering parse error messages.
- Finding and fixing parse errors.
- Printing debugging information with print, var_dump() and error_log().
- Sending var_dump() output to the error log with output buffering functions.
- Writing a custom database error-handling function.
12.6 Exercises

1.

This program has a syntax error in it:

```php
<?php
```
Chapter 13. What Else Can You Do with PHP?

This book covers the fundamental PHP topics that you need for everyday dynamic web site development, such as handling forms, working with a database, and remembering users with sessions. Beyond that core, though, PHP can do much more. Here are a few paragraphs, an example or two, and links to more info about many other capabilities of PHP.
13.1 Graphics

Your PHP programs can produce more than just HTML web pages. With the GD extension, they can also dynamically generate graphics—for example, you can create custom buttons. Example 13-1 draws a rudimentary button whose text comes from the button URL variable.

Example 13-1. Drawing a button image

```php
<?php
```
13.2 PDF

Another kind of non-HTML document that your PHP programs can produce is a PDF file, as shown in Example 13-2. This is handy for making an invoice that incorporates information from your database or providing printable versions of pages that meet exacting layout standards.

Example 13-2. Generating a PDF document

```php
// These values are in points (1/72nd of an inch)
```
13.3 Shockwave/Flash

You can also create full-featured SWF-format Flash movies with the Ming extension. Example 13-3 produces a movie with a blue circle in it that you can drag around.

Example 13-3. Generating a Flash movie

// Use SWF Version 6 to enable Actionscript
13.4 Browser-Specific Code

The get_browser( ) function gives you information about the characteristics and capabilities of a user's browser. It makes it easy to dynamically determine what kind of page to output based on what a browser can do, what kind of browser it is, or on what operating system it's running. Example 13-5 prints a message that depends on the operating system of the user's browser.

Example 13-5. Using get_browser( )

```php
$browser = get_browser( );
```

Example 13-5. Using get_browser( )
13.5 Sending and Receiving Mail

The mail() function (which you saw briefly in Example 6-30) sends an email message. To use mail(), pass it a destination address, a message subject, and a message body. Example 13-6 sends a message with mail().

Example 13-6. Sending a message with mail()

```bash
$mail_body=<<<_TXT_

< Day Day Up >
13.6 Uploading Files in Forms

The `<input type="file">` form element lets a user upload the entire contents of a file to your server. When a form that includes a file element is submitted, the PHP interpreter provides access to the uploaded file through the `$_FILES` auto-global array. Example 13-8 shows a form-processing program whose validate_form( ) and process_form( ) functions use `$_FILES`.

Example 13-8. A file upload form

```php
if (isset($_POST['_stage'])) {
```
13.7 The HTML_QuickForm Form-Handling Framework

Chapter 6 provides all the building blocks of robust form handling. A PEAR module, HTML_QuickForm, takes things a step further. It makes it easy to use common validation rules and simplifies default processing and encoding user input with htmlentities(). With HTML_QuickForm, the entire form is an object. You call methods on that object to add elements and validation rules to the form. Example 13-9 uses HTML_QuickForm to build the form in Example 6-30.

Example 13-9. Building a form with QuickForm

<?php
13.8 Classes and Objects

PHP 5 provides comprehensive and robust support for object-oriented programming. If you've never heard of object-oriented programming, then you don't need to use any of these fancy features. But if you're coming to PHP from a language such as Java, you can structure your code in familiar ways. You can create interfaces; abstract classes; public, private, and protected properties and methods; constructors and destructors; overloaded property accessors and method dispatchers; and plenty of other OO goodies.

Chapter 2 of Upgrading to PHP 5 by Adam Trachtenberg (O'Reilly), lays out the many object-related changes in PHP 5. The PHP Manual covers classes and objects at http://www.php.net/manual/language.oop.php.

13.8.1 Object Basics

An object, in the programming world, is a structure that combines data about a thing (such as the ingredients in an entree) with actions on that thing (such as preparing the entree). Using objects in a program provides an organizational structure for grouping related variables and functions together.

Some words to know when working with objects are defined in the following list:

**Class**

A template or recipe that describes the variables and functions for a kind of object. For example, an Entree class would contain variables that hold its name and ingredients. The functions in an Entree class would be for things such as cooking the entree, serving it, and determining whether a particular ingredient is in it.

**Method**

A function defined in a class is called a method.

**Property**

A variable defined in a class is called a property.

**Instance**

An individual usage of a class. If you are serving three entrees for dinner in your program, you would create three instances of the Entree class. While each of these instances is based on the same class, they differ internally with different properties. The methods in each instance contain the same instructions, but probably produce different results because they each rely on the particular property values in each instance. Creating a new instance of a class is called "instantiating an object."

**Constructor**

A special method that is automatically run when an object is instantiated. Usually, constructors set up object properties and do other housekeeping that makes the object ready for use.

**Static method**

A special kind of method that can be called without instantiating a class. Static methods don't depend on the property values of a particular instance. PEAR DB uses a static method to create a database connection.

13.8.2 Creating a New Object

PEAR DB uses a static method to create a new object instance for you to use:

```
$db = DB::connect($dsn);
```

This calls the connect( ) method defined in the DB class. The connect( ) method is a static method: nothing in connect( ) depends on a specific instance of the DB class. The classname::method( ) syntax is how you call a static method. When you see two colons in a function name like that in a PHP program, think "static method call."

The other way to create a new object is with the new operator:

```
$dinner = new Entree( );
```

This makes the variable $dinner an instance of the class Entree. To pass arguments to a class's constructor, put them in the parentheses:

```
$dinner = new Entree('Chinese','spicy');
```

13.8.3 Accessing Properties and Methods

The -> ("arrow") operator, composed of a hyphen and a greater-than sign, is your road to the properties (variables) and methods (functions) inside an object. To access a property, put the arrow after the object's name and put the property after the arrow:

```
print $dinner->price;
```
13.9 Advanced XML Processing

SimpleXML is just the tip of PHP 5’s new XML processing capabilities. The DOM functions give you exacting control over all aspects of an XML document, and you can also do XSL transformations, XPath queries, and XInclude processing, as well as execute an extravagant, exhaustive exaltation of other exciting and exotic XML exercises.

Example 13-10 shows an RSS feed-handling class based on the built-in DomDocument class. The addItem() method of the RSS class is used to add a new item to the feed.

Example 13-10. Extending DomDocument to handle an RSS feed

class RSS extends DomDocument {

```php
Item
```
13.10 SQLite

The SQLite embedded database engine comes bundled with PHP 5. An SQLite database is a single file. Inside that file are all the tables in a database. You don't need a separate database program running on your server to access an SQLite database—when your PHP program connects to the database, it opens the file, reads from it, and writes to it. For heavily trafficked sites, SQLite isn't as fast as a regular database program such as MySQL, but it is packed with features and is capable for small projects. Example 13-13 shows the answer to Exercise 7.1 using SQLite.

Example 13-13. Using the SQLite database

```php
require 'DB.php';
```
13.11 Running Shell Commands

While you can do almost anything in PHP, you can't do everything. If you need to run an external program from inside a PHP script, you have a few options. These are described in the "Program Execution" section of the PHP Manual (http://www.php.net/exec). Example 13-14 demonstrates the shell_exec() command, which runs a program and returns its output. In Example 13-14, shell_exec() runs the df command, which (on Unix) produces information about disk usage.

**Example 13-14. Running a program with shell_exec()**

```php
// Run "df" and divide up its output into individual lines
```

< Day Day Up >
13.12 Advanced Math

On most systems, the PHP interpreter can handle integers between -2147483648 and 2147483647 (that's 2 billion), and floating-point numbers between $-10^{308}$ and $10^{308}$. If you're writing scientific or other math-intensive applications, such as figuring out each citizen's portion of the U.S. National Debt, that might not be good enough. The BCMath and GMP extensions provide more advanced mathematical capabilities. The GMP extension is more capable, but not available on Windows. Example 13-15 uses the BCMath extension to compute the hypotenuse of a really big right triangle.

Example 13-15. Doing math with the BCMath extension

```php
// Figure out hypotenuse of a giant right triangle
```
13.13 Encryption

With the mcrypt extension, you can encrypt and decrypt data using a variety of popular algorithms such as Blowfish, Triple DES, and Twofish. Example 13-17 encrypts and decrypts a string with Blowfish.

Example 13-17. Encrypting and decrypting with mcrypt

```c
// The string to encrypt
```
13.14 Talking to Other Languages

With various extensions, the PHP interpreter can run programs written in other languages such as Java and Perl. On Windows, the PHP interpreter can access COM objects.

The Perl extension is for PHP 5 only. Example 13-18 demonstrates a very simple program that uses the Perl extension to print a message. Typically, you'd use the Perl extension to access some existing Perl libraries that you have.

Example 13-18. Using Perl from PHP

```php
$perl = new Perl();
```

```bash
13.14 Talking to Other Languages

With various extensions, the PHP interpreter can run programs written in other languages such as Java and Perl. On Windows, the PHP interpreter can access COM objects.

The Perl extension is for PHP 5 only. Example 13-18 demonstrates a very simple program that uses the Perl extension to print a message. Typically, you'd use the Perl extension to access some existing Perl libraries that you have.

Example 13-18. Using Perl from PHP

```php
$perl = new Perl();
```
13.15 IMAP, POP3, and NNTP

You can write a full-featured mail or news client in PHP. (In fact, some people already have—check out http://www.horde.org/imp/ and http://www.squirrelmail.org/). The imap extension gives your PHP programs the ability to talk with IMAP, POP3, and NNTP servers. Example 13-20 uses some of the imap extension functions to connect to the news.php.net news server and retrieve information about 10 most recent messages from the php.announce newsgroup.

Example 13-20. Connecting to an NNTP server

```php
$server = '{news.php.net/nntp:119}';
```
13.16 Command-Line PHP

PHP isn't just for web applications. Your PHP installation can include a CLI (Command-Line Interface) version of the PHP interpreter that lets you run PHP scripts as standalone programs. This can be useful for running a PHP program at certain times of day or just reusing code that you wrote for a web application in a different context.

Read about the CLI version of the PHP interpreter in Section 1.4.5 of O'Reilly's Programming PHP, PHP Cookbook (O'Reilly), Section 20.0 and Recipes 20.1-20.4; and the PHP Manual (http://www.php.net/features.commandline). The PEAR installation instructions in Appendix A use the CLI version of the PHP interpreter.
13.17 PHP-GTK

One advanced use of the CLI PHP interpreter is to use it along with the PHP-GTK functions, which let you write full-featured GUI applications. The existing version of PHP-GTK (1.0.0) works with PHP 4. A new version of PHP-GTK is in the works for PHP 5.

Example 13-21 uses PHP-GTK to display a window with a button in it.

Example 13-21. Displaying a button with PHP-GTK

```php
$window =& new GtkWindow();
```
13.18 Even More Things You Can Do with PHP

There are even more extensions and built-in functions available than what's discussed in this chapter. Three good places to look to learn about PHP's function library, extensions, and add-ons are:

The PHP Manual (http://www.php.net/manual/)

Available in 24 languages, the online PHP Manual has information about all of PHP's built-in functions and lots of user-contributed comments.

The PEAR Package List (http://pear.php.net/packages.php)

PEAR is a collection of hundreds of add-on packages to PHP. The DB package covered in Chapter 7 is probably the most popular one. This chapter highlights some others. When you need to solve a new problem with PHP, check out PEAR before you start to write your code. Someone may have already solved it for you.

The PECL Package List (http://pecl.php.net/packages.php)

PECL is another location for finding extensions to PHP. While the packages in PEAR are themselves written in PHP, PECL packages are written in C and provide access to external libraries or other resources.
Appendix A. Installing and Configuring the PHP Interpreter

If you want to write some PHP programs, you need a PHP interpreter to turn them from punctuation-studded text files into actual interactive web pages. The easiest way to get up and running with PHP is to sign up for a cheap or free web-hosting provider that offers PHP—but you can run the PHP interpreter on your own computer, too.
A.1 Using PHP with a Web-Hosting Provider

If you already have an account with a web-hosting provider, you probably have access to a PHP-enabled server. These days, it is the odd web-hosting provider that doesn't have PHP support. Usually, hosting providers configure their servers so that files whose names end in .php are treated as PHP programs. To see whether your hosted web site supports PHP, first save the file in Example A-1 on your server as phptest.php.

Example A-1. PHP test program

```php
<?php print "PHP enabled"; ?>
```

Load the file in your browser by visiting the right URL for your site (e.g., http://www.example.com/phptest.php). If you see just the message PHP enabled, then your hosted web site supports PHP. If you see the entire contents of the page (<?php print "PHP enabled"; ?>), then your hosting provider probably doesn't support PHP. Check with them, however, to make sure that they haven't turned on PHP for a different file extension or made some other nonstandard configuration choice.

If you can't use PHP with your web hosting provider (or you don't have one), the links at http://www.php.net/links.php#hosts are a good place to start when looking for a web-hosting provider that supports PHP.
A.2 Installing the PHP Interpreter

Installing the PHP interpreter on your own computer is a good idea if you don't have an account with a hosting provider, or you just want to experiment with PHP without exposing your programs to the entire Internet. If you're not using a hosting provider and want to install the PHP interpreter on your own computer, follow the instructions in this section. After you've installed the interpreter, you'll be able to run your own PHP programs.

Installing the PHP interpreter is a matter of downloading some files and putting them in the right places on your computer. You must also configure your web server so that it knows about PHP. This section contains instructions on how to do this for computers running Windows, Linux, Unix, and OS X. If you get stuck, check out the installation FAQ at http://www.php.net/manual/faq.installation.

As this section is being written, the final version of PHP 5 is not yet released. The instructions here are for PHP 4 but should be almost identical for PHP 5. The only difference may be in the names of some files or packages—for example, a php5 Debian package instead of php4. For the latest information, see http://www.oreilly.com/catalog/0596005601.

A.2.1 Installing on Windows

You can install PHP after downloading it from the PHP web site, or you can download a third-party package that integrates PHP, Apache, and MySQL. Installing PHP is a good idea if you already have Apache or MySQL installed, or you want more control over your setup. The integrated packages are a convenient way to get everything up and running in one step.

A.2.1.1 Installing PHP

Download the PHP installation package from http://www.php.net/downloads.php. There are two versions of the Windows download available: the installer download and the zip download. Use the installer download. It is an installation program that you run after downloading. This program copies the PHP interpreter program and supporting files to the right places and helps you configure your web server program to work with the PHP interpreter. The zip version contains the PHP interpreter and a number of PHP extensions but no installation program. If you use the zip version, then you must copy the PHP interpreter program and other files to the right places. The installer download is easier to deal with.

Your web server should be installed before you run the PHP installer. If you want to use Apache, follow the instructions in the later section Section A.4.1.1. However, Apache should not be running when you install PHP. Bring up the Apache monitor by double-clicking on the Apache Monitor icon in the System Tray, or go to to Start ➔ All Programs ➔ Apache HTTP Server 2.0.49 ➔ Control Apache Server ➔ Monitor Apache Servers. This displays the window in Figure A-1. Select Apache2 in the Service Status window and click Stop to stop Apache. If Apache is correctly stopped, the Service Monitor looks like Figure A-2.
A.3 Installing PEAR

Many PEAR modules, such as the DB module discussed in Chapter 7, make your PHP programming life easier. They are high-quality code libraries that help you do all sorts of common tasks in PHP programs such as interacting with a database or generating an HTML form. I recommend always having the PEAR libraries available.

Depending on how you have installed PHP (or how your hosting provider has installed PHP), you may need to take extra steps to also install the PEAR base libraries (including DB) and its package management tool. To see whether you have PEAR installed properly, make a short PHP program that just attempts to include `DB.php`, as shown in Example A-2.

Example A-2. Testing for PEAR installation

```php
require 'DB.php';
```

```codepretty
< Day Day Up >  NEXT
```

```codepretty
< Day Day Up >  NEXT
```
A.4 Downloading and Installing PHP's Friends

To build a web site with PHP, you need a web server. Apache is the most popular web server in the world. It's free, powerful, stable, and secure. What more could you ask for? You probably want a database program to use with your web site. One of the most common choices for a database program to go along with PHP is MySQL. This section shows you how to install Apache and MySQL on your computer.

The instructions in this section are only for people who are installing PHP on their own computers. If you are using a web-hosting provider's PHP setup, then don't install Apache and MySQL yourself. Your hosting provider has taken care of that for you.

A.4.1 Installing Apache

How you install Apache depends on what operating system you're using. Follow the appropriate instructions for your platform.

A.4.1.1 Apache on Windows

Take the following steps to install Apache on Windows:

1. Go to http://httpd.apache.org/download.cgi and download the most recent version of the "Win32 Binary (MSI Installer)" for Apache 2. This is in a section of the page titled something like "Apache 2.0.49 is the best available version," and has a filename such as apache_2.0.49-win32-x86-no_ssl.msi. (As new versions of Apache are released, the 2.0.49 becomes 2.0.50 or 2.1.0 and so on.)

2. After the Installer downloads, double-click on it to run it. You should see a window like the one in Figure A-15. Click the Next button to begin the installation procedure.
A.5 Modifying PHP Configuration Directives

Earlier chapters in the book mention various PHP configuration directives. These are settings that affect the behavior of the PHP interpreter, such as how errors are reported, where the PHP interpreter looks for included files and extensions, and much more.

Read this section when you encounter a configuration directive you want to alter or are curious as to how you can tweak the PHP interpreter's settings (whether you are using PHP on your own computer or with a hosting provider). For example, changing the output_buffering directive (as discussed in Section 8.6) makes your life much easier if you are working with cookies and sessions.

The values of configuration directives can be changed in a few places: in the PHP interpreter's php.ini configuration file, in Apache's httpd.conf or .htaccess configuration files, and in your PHP programs. Not all configuration directives can be changed in all places. If you can edit your php.ini or httpd.conf file, it's easiest to set PHP configuration directives there. But if you can't change those files because of server permissions, then you can still change some settings in your PHP programs.

The php.ini file holds system-wide configuration for the PHP interpreter. When the web server process starts up, the PHP interpreter reads the php.ini file and adjusts its configuration accordingly. To find the location of your system's php.ini file, examine the output from the phpinfo() function. This function prints a report of the PHP interpreter's configuration. The tiny program in Example A-3 produces a page that looks like the one in Figure A-21.

Figure A-21. Output of phpinfo()
A.6 Appendix Summary

This appendix covers:

• Using PHP with a web-hosting provider.
• Installing the PHP interpreter on Windows, Linux, or OS X.
• Installing PEAR.
• Installing Apache on Windows, Linux, or OS X.
• Installing MySQL on Windows, Linux, or OS X.
• Using phpinfo( ) to see the PHP interpreter's configuration.
• Understanding the structure of the *php.ini* configuration file.
• Configuring the PHP interpreter in the *httpd.conf* configuration file.
• Reading and writing configuration directive values with ini_get( ) and ini_set( ).
• Using common configuration directives.
Appendix B. Regular Expression Basics

Behind the innocuous and generic phrase regular expression lives an intricate and powerful world of text pattern matching. With regular expressions, you can make sure that a user really entered a ZIP Code or an email address in a form field, or find all the HTML <a> tags in a page. If your web site relies on data feeds that come in text files, such as sports scores, news articles, or frequently updated headlines, regular expressions can help you make sense of these.

This appendix provides an overview of the most useful and commonly encountered parts of the regular expression menagerie. By learning the special meanings of 5 or 10 symbols and 2 or 3 PHP functions, you can use regular expressions to solve most of the text-processing problems you run into when building a web site with PHP. There are some dark corners and steep ravines of the regular expression landscape that are not covered here, however, such as locale support, lookahead and assertions, and conditional subpatterns. To learn more about regular expressions, see the PCRE section of the PHP Manual, at http://www.php.net/pcre, or read the comprehensive Mastering Regular Expressions by Jeffrey E.F. Friedl (O'Reilly).

To work with regular expressions in PHP, use the functions in the PCRE (Perl-compatible regular expressions) extension.[1] These functions are included with PHP by default and are described in the online manual at http://www.php.net/pcre. Section B.6, later in this appendix, gives an overview of the PCRE functions. If you're already familiar with regular expression basics, read that section to learn the language-specific details of using regular expressions in PHP.

[1] Generally, it's best to avoid the POSIX regular expression functions: ereg( ) and friends. They are not as capable as the PCRE functions.

A regular expression is a string. That string defines a pattern that matches other strings. For example, the regular expression \d{5}(-\d{4})? matches U.S. ZIP or ZIP+4 Codes:

```
\d
```

A digit (0-9)

```
{5}
```

A total of five of the previous item (a digit)

```
-
```

A literal - character

```
\d
```

A digit (0-9)
B.1 Characters and Metacharacters

In a regular expression, some characters match themselves, such as the hyphen in the ZIP Code regex or the < in the HTML tag regex. Some characters have special meanings, such as the ? that makes something optional or the square brackets that mean "one character from the list inside the square brackets." The characters that match themselves are called literals. The characters that have special meanings are called metacharacters.

A pattern containing only literals matches strings that contain the sequence of literals in the pattern. For example, the pattern href= matches the strings <a href="/">Home</a>, schref=, and set href=12.

The metacharacter . (dot) matches any character.\[2\] So, the pattern d.g matches dog, d7g, adagio, digdug, and *d*g*, among other possibilities. It also matches d.g, since dot (the metacharacter) matches a literal . character. Without a quantifier (introduced in Section B.2), dot matches exactly one character. This means that d.g doesn't match ridge (it has no characters between the d and the g) or doug (it has more than one character between the d and the g).

[2] This isn't entirely true. By default, dot doesn't match a newline character. Turning on the s pattern modifier makes dot match newline, however. This and other pattern modifiers are explained later in this appendix in Section B.6.

The metacharacter | (bar) is for alternation. Use alternation to construct a pattern that matches more than one set of characters. For example, dog|cat matches strings that contain dog or cat, such as dog, cathode, redogame, and hotdog stand. The pattern dog|cat does not mean "match do, then either g or c, then at." The alternation text generally includes everything back to the beginning of the pattern or forward to the end of the pattern. However, you can restrict the reach of alternation by enclosing the choices in parentheses. For example, s(cr|in)ew means "match s, then either cr or in, then ew"—it matches screw, sinew, and my screwdriver, but not screen or deminews. Without the parentheses, the pattern scr|inew means "match scr or inew." This still matches screw and sinew, but it also matches screen and deminews. Alternation can also be used with more than just two choices. For example, s(cr|in|tr|ch)ew matches screw, sinew, strew, and eschew.

Using parentheses to group together characters for alternation is called grouping. (Some things about regular expressions are straightforward.) Grouping also applies to quantifiers, as discussed in the next section. Parentheses also capture the text inside them for subsequent use. The characters that match the part of the pattern inside a set of parentheses are stored in a special variable so you can retrieve them later. Capturing is explained later in this appendix in more detail in Section B.6.1 and Section B.6.2.
B.2 Quantifiers

A quantifier is a metacharacter that tells "how many." You put a quantifier after an item to indicate you want to match that item a certain number of times. Quantifiers are listed in Table B-1.

Table B-1. Quantifiers

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>How many times</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Zero or more</td>
</tr>
<tr>
<td>+</td>
<td>One or more</td>
</tr>
<tr>
<td>?</td>
<td>Optional (zero or one)</td>
</tr>
<tr>
<td>{x}</td>
<td>Exactly x</td>
</tr>
<tr>
<td>{x,}</td>
<td>At least x</td>
</tr>
<tr>
<td>{x,y}</td>
<td>At least x, but no more than y</td>
</tr>
</tbody>
</table>

To use a quantifier, put it immediately after the item you want to quantify. Table B-2 shows some regular expressions with quantifiers.

Table B-2. Quantifier examples

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Meaning</th>
<th>Matches</th>
<th>Doesn't match</th>
</tr>
</thead>
<tbody>
<tr>
<td>ba+</td>
<td>b, then at least one a</td>
<td>ba, baa, baaa, rhumba, babar</td>
<td>b, abs, taaa-daaa, celeste</td>
</tr>
<tr>
<td>ba+na*s</td>
<td>b, at least one a, n, zero or more a, s</td>
<td>turbans, baanas, rhumbanas!</td>
<td>banana, bananas</td>
</tr>
<tr>
<td>ba(na){2}</td>
<td>ba, then na twice</td>
<td>banana, bananas, semi-banana, bananarama</td>
<td>cabana, banarama</td>
</tr>
</tbody>
</table>
B.3 Anchors

Anchors align a pattern for more specific matching. A pattern such as ba(na)+ matches banana but also cabana or bananarama. As long as text matching ba(na)+ is somewhere in a string, the pattern matches. An anchor, however, matches a pattern at the beginning or end of a string. The ^ anchor matches the beginning of a string and the $ anchor matches the end of a string. For example, this pattern matches strings that begin with Gre:

^Gre

The pattern matches Green, Grey Lantern, and Grep is my favorite, but not GGreen VVegetables, gre, or InGres.

This pattern matches strings that end with an exclamation point:

!$

It matches "Zip!," "Zoom!," and "Pow! Kablam!," but not "Kerflooie," "! is the negation operator," "Pow! Oh," or "!!!!!!!!??"

You can use both anchors in a single pattern to match an entire string. The pattern ^ba(na)+ matches banana and bananarama but not cabana. Similarly, ba(na)+$ matches banana and cabana but not bananarama. Anchored on both ends, however, ^ba(na)+$ matches only banana (and bananana, banananana, and so on.) This pattern matches various nicknames for the name William:

^(w|W|b|B)illy?$

It matches Will, will, Bill, bill, Willy, willy, Billy, and billy, but not Willa, billo, twill, handbill, or William.

In addition to the ^ and $ anchors, there are anchor metacharacters that deal with word boundaries. The \b anchor matches at a word boundary and \B matches everywhere that isn't a word boundary. A word boundary is between one character that is a letter, digit, or underscore and another character that is none of those.[3] So, in the phrase It's not a tumor., the word boundaries are before the I, before and after the apostrophe, before and after each space, and before and after the period.

[3] More specifically, a word boundary is between a place where something matches \w and something does not match \w. This includes the beginning of strings that start with word characters and the end of strings that end with word characters. The \w metacharacter is discussed in Section B.4.

The word boundary anchors are useful for matching a string that could occur as part of another word. For example, this pattern matches fish only when it's not part of a compound word:

\b[fF]ish

The pattern matches fish, Go fish!, and Hamilton Fish High School, but not bluefish, sportfishing, or swordfish. However, it also matches sport-fishing, since a word boundary is between - and f.
B.4 Character Classes

A character class lets you represent a bunch of characters (a "class") as a single item in a regular expression. Put characters in square brackets to make a character class. A character class matches any one of the characters in the class. This pattern matches a person's name or a bird's name:

```
^D[ao]ve$
```

The pattern matches Dave or Dove. The character class [ao] matches either a or o.

To put a whole range of characters in a character class, just put the first and last characters in, separated by a hyphen. For instance, to match all English alphabetic characters:

```
[a-zA-Z]
```

When you use a hyphen in a character class to represent a range, the character class includes all the characters whose ASCII values are between the first and last character (and the first and last character). If you want a literal hyphen inside a character class, you must backslash-escape it. The character class [a-z] is the same as [abcdefgijklmnopqrstuvwxyz], but the character class [a-zA-Z] matches only three characters: a, -, and z.

You can also create a negated character class, which matches any character that is not in the class. To create a negated character class, begin the character class with `^`:

```
// Match everything but letters
```

```
```

B.5 Greed

Quantifiers in the PHP interpreter’s regular expression engine are greedy. This means they match as much as they can. The pattern `<b>.*</b>` means "the string `<b>`, then zero or more characters, then the string `</b>`." The "more" in "zero or more" matches as many characters as possible. When the pattern is applied to the string `<b>Look Out!</b> <i>Caution!</i> <b>Uh-Oh!</b>`, the `.*` matches `<b>Look Out!</b> <i>Caution!</i> <b>Uh-Oh!</b>`. The greediness of the quantifier causes it to skip over the first `<b>` it sees and gobble up characters to the last `<b>` in the string.

To turn a quantifier from greedy to nongreedy, put a question mark after it. The pattern `<b>.*?</b>` still matches "the string `<b>`, then zero or more characters, then the string `</b>`", but now the "more" in "zero or more" matches as few characters as possible. Example B-1 shows the difference between greedy and nongreedy matching with `preg_match_all()` (Example B-5 details how `preg_match_all()` works, including the meaning of the @ characters at the start and end of the pattern.)

**Example B-1. Greedy and nongreedy matching**

```php
$meats = "<b>Chicken</b>, <b>Beef</b>, <b>Duck</b>";
```
B.6 PHP's PCRE Functions

Use the functions in PHP's PCRE extension to work with regular expressions in your programs. These functions allow you to match a string against a pattern and to alter a string based on how it matches a pattern. When you pass a pattern to one of the PCRE functions, it must be enclosed in delimiters. Traditionally, the delimiters are slashes, but you can use any character that's not a letter, number, or backslash as a delimiter. If the character you choose as a delimiter appears in the pattern, it must be backslash-escaped in the pattern, so you should only use a nonslash delimiter when a slash is in your pattern.

After the closing delimiter, you can add one or more pattern modifiers to change how the pattern is interpreted. These modifiers are listed at http://www.php.net/pcre.pattern.modifiers. One handy modifier is i, which makes the pattern matching case-insensitive. For example, the patterns (with delimiters) /\[a-zA-Z]+/ and /\[a-z]+/i produce the same results.

Another useful modifier is s, which makes the dot metacharacter match newlines. The pattern (with delimiters) @<b>.*?</b>@ matches a set of <b></b> tags and the text between them, but only if that text is all on one line. To match text that may include newlines, use the s modifier:

@<b>.*?</b>@s

B.6.1 Matching

The preg_match() function tests whether a string matches a pattern. Pass it the pattern and the string to test as arguments. It returns 1 if the string matches the pattern and 0 if it doesn't. Example B-2 demonstrates preg_match().

Example B-2. Matching with preg_match()

```php
// Test the value of $_POST['zip'] against the
```
B.7 Appendix Summary

Appendix B covers:

- Thinking about what you can use a regular expression for.
- Understanding the difference between literals and metacharacters.
- Using the metacharacters . (dot) and | (bar).
- Using the quantifiers *, +, ?, {x}, {x,}, and {x,y}.
- Anchoring a regular expression with ^ or $.
- Anchoring a regular expression with \b or \B.
- Using a character class.
- Using a negated character class.
- Using character class metacharacters such as \d, \D, \w, \W, \s, and \S.
- Understanding greed (in a regular expression context, at least).
- Making quantifiers greedy or nongreedy.
- Matching with preg_match().
- Capturing with preg_match().
- Matching and capturing with preg_match_all().
- Using backreferences in a regular expression.
- Replacing with preg_replace().
- Using backreferences in replacement.
B.8 Exercises

1. Write a regular expression that flexibly matches a U.S. phone number whether or not it has parentheses around the area code and has its parts separated by spaces, hyphens, or periods. The regular expression should match phone numbers written like this:
   - (718) 498-1043
   - (718) 498 1043
   - 718 498 1043
   - 718 498-1043
   - 718.498.1043

2. What would you add to a validate_form( ) function to check that a submitted form field named username contains only letters and numbers? Use if( ), preg_match( ), and a regular expression.

3. Starting with the code from Example 10-3, write a program that retrieves the weather page for your ZIP Code and parses that page with a regular expression to get the current temperature.

4. Write a program that retrieves a remote web page and prints a list of the hyperlinks in that page. Just look for links that look like this: <a href="http://www.example.com">The Example Page</a>. Don't worry about links with other attributes in the <a> tag.
Appendix C. Answers To Exercises

Section C.1. Chapter 2
Section C.2. Chapter 3
Section C.3. Chapter 4
Section C.4. Chapter 5
Section C.5. Chapter 6
Section C.6. Chapter 7
Section C.7. Chapter 8
Section C.8. Chapter 9
Section C.9. Chapter 10
Section C.10. Chapter 11
Section C.11. Chapter 12
Section C.12. Appendix B
C.1 Chapter 2

C.1.1 Exercise 1:

1. The opening PHP tag should be <?php. There should not be a space between <? and php.

2. The string 'I'm fine' should either be enclosed in double quotes ("I'm fine") or the apostrophe should be escaped (I\'m fine).

3. The closing PHP tag should be ?>, not ??>.

C.1.2 Exercise 2:

$hamburger = 4.95;
C.2 Chapter 3

C.2.1 Exercise 1:

a. false
b. true
c. true
d. true
e. false
f. false
g. true

C.2.2 Exercise 2:


C.2.3 Exercise 3:

$fahr = -50;$
C.3 Chapter 4

C.3.1 Exercise 1:

$population = array('New York, NY' => 8008278,
C.4 Chapter 5

C.4.1 Exercise 1:

```php
function html_img($url, $alt = '', $height = 0, $width = 0) {
```

C.5 Chapter 6

C.5.1 Exercise 1:

```
var_dump($_POST) prints:
array(4) {
```
C.6 Chapter 7

C.6.1 Exercise 1:

```php
<?php

```
C.7 Chapter 8

C.7.1 Exercise 1:

```php
<?php
```
C.8 Chapter 9

C.8.1 Exercise 1:

$stamp = mktime(19, 45, 0, 10, 20, 2004);
C.9 Chapter 10

C.9.1 Exercise 1:

Here's a sample template file, article.html:

```html
<html>
</html>
```
C.10 Chapter 11

C.10.1 Exercise 1:

$menu=<<<XML_

< Day Day Up >
C.11 Chapter 12

C.11.1 Exercise 1:

The error message looks like:

```
Parse error: parse error, unexpected T_GLOBAL in exercise-12-1.php on line 6
```

The global declaration has to be on a line by itself, not inside the print statement. To fix the program, separate the two:

```php
<?php
```
C.12 Appendix B

C.12.1 Exercise 1:

The regular expression `^\(?\d{3}\)?[-\.\ ]\d{3}[-\.\ ]\d{4}$` matches "an optional literal (, then three digits, then an optional literal ), then either a hyphen, space, or period, then three digits, then either a hyphen, space, or period, then four digits." The ^ and $ anchors make the expression match only phone numbers, not larger strings that contain phone numbers.

C.12.2 Exercise 2:

```php
if (! preg_match('/^[a-z0-9]$/', $_POST['username'])) {
```

< Day Day Up >
Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of Learning PHP 5 is an eagle. Eagles fall into the category of bird known as "raptors," a category that also includes falcons and hawks. There are two types of raptor: grasping killers, with beaks shaped for tearing and cutting and short toes with curved claws designed for killing; and grasping holders, with beaks shaped for tearing and biting, and longer toes designed for holding. Eagles are grasping killers. Sea eagles have special adaptations to their toes that enable them to grasp smooth prey such as fish. Their excellent vision enables all eagles to spot prey from the air or a high perch. The eagle then swoops down, grabs its prey, and takes off in flight again, in one graceful movement. Eagles often eat their victims while still flying, breaking them apart and discarding the nonedible parts to lighten their load. Eagles, like most raptors, often dine on sick or wounded animals.

There are more than 50 species of eagle spread throughout the world, with the exception of New Zealand and Antarctica. All species of eagles build nests, or aeries, high above the ground, in trees or on rocky ledges. A pair of eagles will use the same nest year after year, lining it with green leaves and grass, fur, turf, or soft materials. The eagle will add to its nest each year. The largest eagle nest ever found was 20 feet deep and 10 feet across.

Hunting, increased use of pesticides, and the diminishment of their natural environment, with the attendant reduction in food sources, have endangered many species of eagle.

Mary Brady was the production editor and the copyeditor for Learning PHP 5. Leanne Soylemez was the proofreader. Mary Anne Weeks Mayo and Claire Cloutier provided quality control. Judy Hoer wrote the index.

Hanna Dyer designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Joe Wizda to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Mary Brady.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Ellie Cutler) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, Ellie Cutler, and Jeff Liggett.
/ (forward slash)
   division operator
   (period) string concatenation operator  2nd
!= not-equal operator
# (hash mark), comments in PHP programs
$ (dollar sign) anchor
$_COOKIE auto-global array
$_FILES auto-global array
$_GET auto-global array
$_POST auto-global array  2nd  3rd
   changing values in
   default values for forms, displaying
   encrypted passwords and
   hidden parameters in
   validating numeric and string elements
$_SERVER auto-global array
   elements in
$_SESSION auto-global array
   saving form data in a session
   unset( ) and
$GLOBALS array
$php_errormsg global variable
% (percent sign)
   modulus division operator  2nd
   SQL wildcard
& (ampersand) logical AND operator
&& (two ampersands) logical AND operator
&amp; (ampersand) HTML entity
&amp;gt; (greater than) HTML entity
&amp;lt; (less than) HTML entity
&amp;quot; (double quote) HTML entity
> (greater than)
   comparing numbers and strings
   WHERE clause operator
>= (greater than or equal to)
   comparing numbers and strings
   WHERE clause operator
< (less than)
   comparing numbers and strings
   WHERE clause operator
<<< (here document syntax)
<= (less than or equal to)
   comparing numbers and strings
   WHERE clause operator
<? start tags  2nd
<?php start tags  2nd  3rd  4th
<![ ]> (not equal to) WHERE clause operator
<select> menu
   displaying in show_form( )
   multiple menus
      for date input
      for time input
   processing date/time input from forms
   setting default values in
   single menu with one choice
ab and ab+ modes for fopen()
abs()
activating sessions
Adabas D PHP extension
addresses (email), validating
affectedRows()
allow_url_fopen configuration directive
anchors and regular expressions
AND WHERE clause operator
answers to exercises
  Appendix B
  Chapter 10
  Chapter 11
  Chapter 12
  Chapter 2
  Chapter 3
  Chapter 4
  Chapter 5
  Chapter 6
  Chapter 7
  Chapter 8
  Chapter 9
Apache
  configuring
  installing on
    Linux
    OS X
    Windows
  stopping
Applied Cryptography 2nd
arguments
  changing values of
  default values for
  mandatory vs. optional
  multiple, in functions
  passing to functions
arithmetic operators in PHP
array variables in PHP
array( )
  arrays, creating
  multidimensional arrays, creating
  numeric arrays, creating
array_key_exists( ) 2nd
array_search( )
arrays
  checking for
    elements with particular keys
    elements with particular values
  choosing names for
create
  using explode( )
  using preg_split( )
finding elements of, using preg_grep( )
generating XML from
including in debugging output
backreferences in regular expressions
backslashes, escaping with 2nd 3rd
BBEdit text editor
BCMath extension for PHP
BLOB column type
bracket matching (debugging feature)
browscap configuration directive
browsers
get_browser() 
PHP and
sending error messages to
calendars, displaying
calling functions
   with multiple arguments
capturing return values of functions
capturing text
   
   preg_match() and
   preg_match_all() and
   preg_replace() and

case of strings, manipulating
case sensitivity
   comparing strings and
   of variables
   in PHP programs
   in SQL
character classes and regular expressions
characters and regular expressions
checkboxes, setting default values in
checkpoints (debugging feature)
classes, support for in PHP 5
CLI (Command-Line Interface) version of PHP interpreter
CLibPDF extension
client-side languages
columns
   creating database tables
   inserting values in
   ordering by multiple
   retrieving data from
   returning one
   updating data in
COM extension for PHP
command-line PHP
comments in PHP programs 2nd 3rd
configuration directives, modifying
confirmation-message strategy
connect() [See DB::connect()]
constructors
cookies
   activating sessions
   default lifetime of
   domain, setting
   expiration times for, setting 2nd
   setting
   setting paths for
correct passwords, results of entering
count() 2nd
CREATE TABLE command
cross-platform feature of PHP
cross-site scripting attacks
   preventing 2nd
crypt() CSV files
curly braces
   interpolating with 2nd 3rd
   making decisions with if() usefulness of
\d metacharacter
\D metacharacter
Data Source Names (DSNs)
database extensions
database tables
   adding rows to
column types for
creating
displaying information from
errors in, fixing
form data
   inserting safely
   retrieving safely
   information from, formatting as XML
inserting CSV data into
inserting data into
organizing data in
retrieving data from
date parts
date ( )
   format characters for
   show_form ( ) and
   vs. strftime ( )
dates and times
displaying
in forms
   testing number ranges
   parsing
DATETIME column type
DB module [See PEAR DB]
DB++ PHP extension
DB::connect ( )
   creating new objects
   inserting data into databases
   mysqli functions and
DB::isError ( )
   checking query success
DB_FETCHMODE_ASSOC constant
DB_FETCHMODE_OBJECT constant
db_program options
DB2 PHP extension
dbase (db_program option)
debugging programs
   [See also errors]
   inspecting program data
   PHP-aware text editors
   syntax highlighting
DECIMAL column type
declaring functions
decrementing variables
decrypting data with mcrypt extension
default values
   for arguments, specifying
   in forms, displaying
DELETE command
   using wildcards with
E_ALL constant
E_ERROR constant
E_NOTICE constant
E_PARSE constant
E_STRICT constant
E_WARNING constant
EasyPHP package
elements of arrays
elements, XML
   accessing identically named
   changing
   generating XML from arrays
   printing attributes of
   printing contents of
else clause, using with if()
different forms
endif()
Emacs text editor
email messages
   sending
   sending confirmation messages for verification
   sending confirmation messages for verification
   validating addresses
Empress PHP extension
empty arrays
encrypting
data with mcrypt extension
passwords
end tags (?>)
entities, HTML
   [See htmlentities()]
epoch timestamps
   number ranges in forms
   printing formatted time strings
   processing date/time <select> menus
   working with date/time values as
error_log()
error_reporting configuration directive
ErrorLog Apache configuration setting
errors
   checking query success
   checkpoints, adding
   connecting to database programs
   controlling where they appear
in databases, fixing
   debugging programs
   displaying error messages in forms
   error handling in mysqli extension
in files, checking for
   PHP-aware text editors
   sending output before setcookie() or session_start() is called
   syntax highlighting
escapeshellargs()
escaping
   escape character
   shell metacharacters
   single quotes
special characters
false (truth value)
- negation operator and
- return values of functions
- validating form elements
- fatal errors
- fbsql (db_program option)
- fclose()
  - checking for errors from
  feof()
- fetch mode
- fetchRow()
  - changing format of retrieved rows
  - retrieving data from database
- fgetcsv()
  - checking for errors from
- fgets()
  - checking for errors from
- __FILE__ special constant
- file_exists()
- file_get_contents() 2nd
  - checking for errors from
  - sanitizing externally supplied filenames
- file_put_contents()
  - checking for errors from
  - return values for
- file_uploads configuration directive 2nd
- files
  - CSV
    - error checking in
    - escaping special characters
  - permissions
    - inspecting
    - understanding
- reading
  - entire file
  - parts of
- sanitizing externally supplied names
- writing
  - entire file
  - parts of
- Fitzgerald, Michael
- Flash movies in PHP programs
- floating-point numbers
  - arithmetic operators and
  - checking for, in forms
  - comparing
  - formatting rules for
  - truth values of
- floatval()
- fopen()
  - checking for errors from
  - modes for
- for() loop 2nd
  - looping through multidimensional arrays
  - multidimensional numeric arrays and
Garfinkel, Simson  2nd
GD extension
get_browser()    getAll()
    changing format of retrieved rows  retrieving rows
getDebugInfo()  getMessage()  getOne()  2nd
ggetRow()      global keyword
    accessing from inside functions
global variables
GMP extension for PHP
Goodman, Danny
graphics in PHP programs
greedy quantifiers
grouping together characters in regular expressions
header()  2nd
Headers already sent error message
headers in HTML documents
Hello World! example
helper functions for simplifying form element display  2nd
here documents
  assignment and
  interpolating variables into
hidden parameters in forms
HTML
  form example
  transforming XML to, using XSL
  validating submitted form data
  vs. XML
HTML & XHTML: The Definitive Guide  2nd
HTML_Common package
HTML_QuickForm module
  installing
htmlentities()  2nd
  generating XML documents
  HTML_QuickForm module
  preventing cross-site scripting attacks  2nd
HTTP Developer's Handbook
HTTP_REFERER element in $_SERVER auto-global array
HTTP_USER_AGENT element in $_SERVER auto-global array  2nd
httpd.conf file
i pattern modifier
ibase (db_program option)
identifying rows in tables uniquely
idle times of sessions, changing 2nd
if() 2nd
  assignment vs. comparison
  equality operator and
  extending with else clause
  extending with elseif() 
  negation operator and
  not-equal operator and 
  return values in 2nd
  validating number ranges in forms
ifx (db_program option)
imap extension
implode() 2nd
in_array()
include construct
include_path configuration directive
incorrect passwords, results of entering
incrementing variables
Informix PHP extension
Ingres II PHP extension
ini_get()
ini_set()
  changing session idle times
initialization expressions
input_radiocheck() 
input_select() 2nd 3rd
input_submit()
input_text()
input_textarea() 
INSERT command
instances and classes
INT column type
incorrect passwords, checking for, in forms
InterBase PHP extension
interpolating
  array element values in double-quoted strings
  with curly braces 2nd 3rd
  inserting form data 
  variables into strings
intval()
is_readable()
is_writeable()
iteration expressions
Java extension for PHP
JavaScript in submitted form data, validating
Kennedy, Bill
keys of array elements
Kline, Kevin E.
Knight, Jeff
Komodo text editor
krsort()
ksort()
Lane, David
Learning XML
Learning XSLT
Lerdorf, Rasmus 2nd
LIKE operator
LIMIT clause
line numbers in program files (debugging feature)
Linux
installing Apache on
installing PHP interpreter on
literals
default values for arguments
in regular expressions
local variables
localhost, connecting to
log_errors configuration directive 2nd
logging out users
logical operators
combining multiple expressions inside if() statement
setting error_reporting configuration directive
login identification for sessions
looping constructs
for() loop  [See for()]
foreach() loop  [See foreach()]
while() loop  2nd  3rd
Macromedia Dreamweaver MX 2004 text editor
OMagic QuotesÓ feature in PHP
magic_quotes_gpc configuration directive  2nd
magic_quotes_runtime configuration directive
mail( )
Mail/Mail_Mime modules
make_csv_line( )
mandatory vs. optional arguments
Mastering Regular Expressions  2nd
matching patterns with preg_match( )
mathematics
  arithmetic operators in PHP
    BCMATH and GMP extensions
mcrypt extension for PHP
metacharacters
  escaping shell metacharacters
  regular expressions and
methods
  accessing
Microsoft SQL Server PHP extension
Ming extension
mktime( )
  calculating epoch timestamps
  cookie expiration times, creating
  making epoch timestamps with
move_uploaded_file( )
msql (db_program option)
MySQL PHP extension
msql (db_program option)
mssql (db_program option)
multidimensional arrays
  forms and
multiline text areas, setting default values in
Musciano, Chuck
MySQL
  installing on Windows/OS X/Linux
    without PEAR DB
    PHP extension
mysql (db_program option)
MySQL Cookbook  2nd
MySQL Reference Manual
mysqli (db_program option)
mysqli extension
mysqli functions vs. PEAR DB functions
mysqli_affected_rows( )
mysqli_connect( )
mysqli_connect_error( )
mysqli_error( )
mysqli_fetch_assoc( )
mysqli_fetch_object( )  2nd
mysqli_fetch_row( )
mysqli_num_rows( )  2nd
mysqli_query( )  2nd
mysqli_real_escape_string( )
special character
negated character classes
negation operator, using in test expressions
nextID() 2nd
NNTP servers and PHP programs
nongreedy quantifiers
notices from PHP interpreter
number_format( )
numbers
  comparing
  comparing strings and
  validating in forms
numeric arrays, creating
numrows( ) 2nd
NuSphere PHPEd text editor
ob_end_clean()
ob_get_contents()
ob_start()
objects
- connecting to database programs
- creating new
- putting data into databases
- retrieving data from databases
- retrieving rows as
oci8 (db_program option)
odbc (db_program option)
ODBC PHP extension
one-dimensional arrays
open source project, PHP as
optional vs. mandatory arguments
OR WHERE clause operator
Oracle PHP extension
ORDER BY clause
ordering rows returned from SELECT query
OS X
- installing Apache on
- installing PHP interpreter on
output_buffering configuration directive 2nd 3rd
Ovrimos SQL PHP extension
padding characters
parameters in forms  
  accessing  
  hidden
parse errors
  fixing
passing return values to other functions
passwords
  encrypted
    retrieving from database
    using
    results of entering correct and incorrect
PATH_INFO element in $_SERVER auto-global array
paths, setting for cookies
pattern matching  [See regular expressions]
pattern modifiers
PCRE (Perl-compatible regular expressions) extension
  functions working with regular expressions
PDF documents, generated by PHP
PDFLib library
PEAR DB  2nd
  changing format of retrieved rows
  connecting to database programs
  creating new objects
db_program options
  functions vs. mysqli functions
  generating unique IDs
  installing
  Mail/Mail_Mime modules
  using MySQL without
  placeholders feature
PEAR_ERROR_CALLBACK function
PEAR_ERROR_DIE constant
PECL packages
Perl extension for PHP
permissions, file
  inspecting
  understanding
pgsql (db_program option)
PHP
  advantages of
  basic rules of programs
  database extensions
  graphics generated by
  PDF documents generated by
  Shockwave/Flash in
  SimpleXML module
  usage statistics for
  variables in
  web browsers, web servers, and
  web-hosting providers and
  XML, parsing/generating
PHP Cookbook  2nd  3rd  4th  5th
PHP Extension and Application Repository  [See PEAR DB]
PHP interpreter
quantifiers and regular expressions
  greedy/nongreedy

query()
  changing data in databases
  changing format of retrieved rows
  creating tables
  deleting data from databases
  inserting data into databases
  placeholders
    in UPDATE commands
    inputting form data
  retrieving data from databases
  safely inserting form data

QUERY_STRING element in $SERVER auto-global array

quote()

quotes
  double  [See double-quoted strings]
  [matching and balancing (debugging feature)]
  single  [See single quotes]
  turning straight into curly

quoteSmart()  2nd
radio buttons, setting default values in
Ray, Erik T. 2nd
rb and rb+ modes for fopen()
read permission, testing for
reading
  entire files
  parts of files
realpath()
register_globals configuration directive
regular expressions
  anchors and
  character classes and
  characters and metacharacters
  email addresses, verifying with
  grouping together characters
  PCRE extension functions
  quantifiers and
    greedy/nongreedy
  screen scraping and
  validation strategies and
remote files
  reading
  writing
REMOTE_ADDR element in $_SERVER auto-global array
REMOTE_HOST element in $_SERVER auto-global array
replacing matching parts of strings
require construct
required elements in forms, checking length of
resources, PHP
response body in HTML documents
return keyword
return values 2nd
  assigning to variables
  capturing
  passing to other functions
return values of functions
reverse-sorting functions for arrays
rows
  adding to database tables
  affectedRows()
  alternating colors of
  counting, using numRows()
  fetchRow()
  removing from tables
retrieved
  changing format of
    as objects
  returned from SELECT query, ordering
  uniquely identifying in tables
  updating all or some
rsort()
RSS (XML format)
  extending DomDocument to handle RSS feed
  generating XML documents
\s metacharacter
\S metacharacter
s pattern modifier
sanitizing
  externally supplied filenames
  externally supplied form input
  form data  2nd
SAP DB/MaxDB PHP extension
Schneier, Bruce  2nd
Schwartz, Alan  2nd
scope of variables
screen scraping
SELECT command
  using wildcards with
<select> menu
  printing with for()
  printing with while()
semicolons (;), ending PHP programs
sequences and unique integer IDs
server-side languages
SERVER_NAME element in $_SERVER auto-global array
servers
  PHP and
  sending error messages to error logs  2nd
  useful variables for
session IDs
  session.auto_start configuration directive  2nd  3rd
  session.gc_maxlifetime configuration directive  2nd
  session.gc_probability configuration directive  2nd
session_start( )
  required to be at top of page
  storing session data
sessions
  activating
  configuring
  idle times of, changing  2nd
  login and user identification
  printing session data
  retrieving information
  saving form data in
  storing data
setcookie( )
  cookie domain, setting
  deleting
  expiration time for cookies, setting  2nd
  required to be at top of page
  setting paths for
  starting a page with
setErrorHandling( )  2nd
setFetchMode( )  2nd
shell_exec()  2nd
Shiflett, Chris
Shockwave/Flash in PHP programs
short open tags
short_open_tag configuration directive
special character
T_VARIABLE tokens
Tatroe, Kevin 2nd
test expressions
  assignment vs. comparison
  for() loops and
  negation operator and
  return values of functions and
text boxes, setting default values in
text in PHP
time parts
time()
  cookie expiration times, creating
times [See dates and times]
timestamps [See epoch timestamps]
tokens used by PHP interpreter
Trachtenberg, Adam 2nd
track_errors configuration directive 2nd
trim()
  combining with strlen()
  removing newlines
true (truth value)
  equality operator and
  negation operator and
  return values of functions
  validating form elements
while() and
truncating strings with substr()
ucwords()
unencrypted passwords, avoid using
Unix, installing PHP interpreter on
unset() 2nd
UPDATE command
using wildcards with
Upgrading to PHP 5 2nd 3rd
upload_max_filesize configuration directive 2nd 3rd
URLs
reading remote files
writing remote files
usage statistics for PHP
users
accounts and file permissions
identifying before logging in
logging out
names of, retrieving from database
validate_form() 2nd
\checking submitted value for <select> menu
changing values in $_POST
displaying calendars
displaying error messages 2nd
encrypted passwords, using
processing date/time <select> menus
retrieving usernames/passwords from database
saving form data in a session
username/password acceptability, checking
validating
  email addresses
  form data
  HTML/JavaScript
  number ranges
  strings
values of array elements
var_dump()
\w metacharacter
\W metacharacter

warnings from PHP interpreter
wb and wb+ modes for fopen()

web browsers
  get_browser()
  PHP and
  sending error messages to
Web Database Applications with PHP & MySQL  2nd
web pages, retrieving with file_get_contents()
Web Security, Privacy & Commerce

web servers
  PHP and
  sending error messages to error logs  2nd
  useful variables for
web-hosting providers and PHP
WHERE clause
  removing some rows from tables
  SQL operators
    updating some rows
while() loop  2nd  3rd

whitespace
  in PHP programs
    in single-quoted strings
wildcards in SQL
Williams, Hugh E.
Windows
  EasyPHP package
    installing Apache on
      installing PHP interpreter on
word boundary anchors
write permission, testing for
writing
  entire files
  parts of files
xb and xb+ modes for fopen()
XEmacs text editor
XHTML (XML tag set)
XML documents
accessing elements in
advanced processing
generating
in existing files, processing
on remote servers, loading
parsing
printing
saving
transforming to HTML, using XSL
vs. HTML
XSLTProcessor class
Zend IDE text editor