PHP for the Web

Fifth Edition

LARRY ULLMAN
Dedication

For Jessica, Gina, and Rich, with gratitude for all their love and support.
Special Thanks to:

Many, many thanks to everyone at Peachpit Press for their assistance and hard work, especially:

Robyn Thomas, for managing the project adeptly, and for knowing when to push and poke.

Liz Welch, for fine-tuning my prose with her copyediting skills.

Paul Reinheimer, for the superlative technical review, keeping me honest, and finding things to improve even in a fifth edition.

Scout Festa, for the sharp proofreading eye.

David Van Ness, who takes a bunch of disparate stuff and turns it into a book.

Thanks for doing what’s required to create, publish, distribute, market, sell, and support these books.

My sincerest thanks to the readers of the other editions of this book and my other books. Thanks for your feedback and support and for keeping me in business.

Rasmus Lerdorf (who got the PHP ball rolling), the people at PHP.net and Zend.com, those who frequent the various newsgroups and mailing lists, and the greater PHP and open source communities for developing, improving upon, and supporting such wonderfully useful technology.

Zoe and Sam, for continuing to be the kid epitome of awesomeness.

Jessica, for doing everything you do and everything you can.
# Table of Contents

Introduction ................................................. ix

## Chapter 1  Getting Started with PHP
- Basic HTML Syntax ......................................... 2
- Basic PHP Syntax ........................................... 7
- Using SFTP .................................................. 10
- Testing Your Script ......................................... 12
- Sending Text to the Browser ............................... 15
- Using the PHP Manual ..................................... 18
- Sending HTML to the Browser ............................. 21
- Adding Comments to Scripts ............................... 24
- Basic Debugging Steps .................................... 27
- Review and Pursue ......................................... 29

## Chapter 2  Variables
- What Are Variables? ........................................ 32
- Variable Syntax ............................................. 36
- Types of Variables ......................................... 38
- Variable Values ............................................ 41
- Understanding Quotation Marks ......................... 44
- Review and Pursue ......................................... 48

## Chapter 3  HTML Forms and PHP
- Creating a Simple Form .................................... 50
- Choosing a Form Method ................................... 54
- Receiving Form Data in PHP .............................. 58
- Displaying Errors .......................................... 63
- Error Reporting ............................................. 65
- Manually Sending Data to a Page ....................... 68
- Review and Pursue ......................................... 73
## Table of Contents

**Chapter 4** Using Numbers
- Creating the Form .......................... 76
- Performing Arithmetic ..................... 79
- Formatting Numbers ...................... 83
- Understanding Precedence ............... 86
- Incrementing and Decrementing a Number 88
- Review and Pursue ....................... 92

**Chapter 5** Using Strings
- Creating the HTML Form .................. 94
- Concatenating Strings .................... 97
- Handling Newlines ....................... 101
- HTML and PHP ............................ 104
- Encoding and Decoding Strings .......... 108
- Finding Substrings ...................... 113
- Replacing Parts of a String ............. 117
- Review and Pursue ....................... 120

**Chapter 6** Control Structures
- Creating the HTML Form .................. 122
- The if Conditional ....................... 125
- Validation Functions ..................... 128
- Using else ............................... 132
- More Operators ........................... 135
- Using elseif ............................. 144
- The Switch Conditional ................... 148
- The for Loop ............................. 152
- Review and Pursue ....................... 157

**Chapter 7** Using Arrays
- What Is an Array? ......................... 160
- Creating an Array ....................... 162
- Adding Items to an Array ............... 166
- Accessing Array Elements ............... 170
- Creating Multidimensional Arrays ...... 173
- Sorting Arrays ........................... 178
- Transforming Between Strings and Arrays 182
- Creating an Array from a Form .......... 186
- Review and Pursue ....................... 191
Creating Directories ........................................... 330
Reading Files Incrementally .............................. 338
Review and Pursue ........................................... 343

**Chapter 12** Intro to Databases .......................... 345
Introduction to SQL .............................. 346
Connecting to MySQL .................................. 348
MySQL Error Handling ................................. 352
Creating a Table ......................................... 355
Inserting Data into a Database ......................... 360
Securing Query Data .................................. 366
Retrieving Data from a Database ...................... 371
Deleting Data in a Database ............................ 376
Updating Data in a Database ............................ 382
Review and Pursue ........................................ 388

**Chapter 13** Putting It All Together ......................... 389
Getting Started ......................................... 390
Connecting to the Database ......................... 392
Writing the User-Defined Function ................. 393
Creating the Template ................................ 396
Logging In ................................................. 400
Logging Out ............................................... 404
Adding Quotes ............................................ 405
Listing Quotes ............................................ 409
Editing Quotes ............................................ 412
Deleting Quotes ........................................... 418
Creating the Home Page ............................. 422
Review and Pursue ........................................ 426

**Appendix A** Installation and Configuration ............ 427

**Appendix B** Resources and Next Steps .................. 449

Index ......................................................... 459
When I began the first edition of this book in 2000, PHP was a little-known open source project. It was adored by technical people in the know but not yet recognized as the popular choice for web development that it is today. When I taught myself PHP, very little documentation was available on the language—and that was my motivation for writing this book in the first place.

Today things are different. The Internet has gone through a boom and a bust and has righted itself. Furthermore, PHP is now the reigning king of dynamic web design tools and has expanded somewhat beyond the realm of just web development. But despite PHP’s popularity and the increase in available documentation, sample code, and examples, a good book discussing the language is still relevant. Although PHP is in the beginnings of its sixth major release, a book such as this—which teaches the language in simple but practical terms—can still be your best guide in learning the information you need to know.

This book will teach you PHP, providing both a solid understanding of the fundamentals and a sense of where to look for more advanced information. Although it isn’t a comprehensive programming reference, this book, through demonstrations and real-world examples, provides the knowledge you need to begin building dynamic websites and web applications using PHP.

What Is PHP?

PHP originally stood for Personal Home Page. It was created in 1994 by Rasmus Lerdorf to track the visitors to his online résumé. As its usefulness and capabilities grew (and as it began to be utilized in more professional situations), PHP came to mean PHP: Hypertext Preprocessor. The definition basically means that PHP handles data before it becomes HTML—which stands for Hypertext Markup Language.
According to the official PHP website, found at www.php.net, PHP is “a popular general-purpose scripting language that is especially suited to web development.” More specifically, PHP is a scripting language commonly embedded within HTML. Let’s examine what this means in more detail.

To say that PHP can be embedded into HTML means that PHP code can be written within your HTML code—HTML being the language with which all web pages are built. Therefore, programming with PHP starts off as only slightly more complicated than hand-coding HTML.

Also, PHP is a scripting language, as opposed to a compiled language. This means that PHP is designed to do something only after an event occurs—for example, when a user submits a form or goes to a URL (Uniform Resource Locator—the technical term for a web address).

Another popular example of a scripting language is JavaScript, which commonly handles events that occur within the browser. Both PHP and JavaScript can also be described as interpreted, because the code must be run through an executable, such as the PHP module or the browser’s JavaScript component. Conversely, compiled languages such as C and C++ can be used to write stand-alone applications that can act independently of any event.

As of this writing, this is the appearance of the official PHP website, located at www.php.net. Naturally, this should be the first place you look to address most of your PHP questions and curiosities.

PHP 6?

Yes, as of this writing, the current versions of PHP were 5 and 7, but not 6! There’s a long and amusing story here, but the short version is that PHP 6 was actively developed for a while. After hitting many snags, the development was halted and the created work was rolled into PHP 5.

When it became time to work on the next major version, after much debate it was decided that that version would be named PHP 7. So although there was once a beta version of PHP 6, no final release ever saw the light of day.
You should also understand that PHP is a server-side technology. This refers to the fact that everything PHP does occurs on the server (as opposed to on the client, which is the computer being used by the person viewing the website). A server is just a computer set up to provide the pages you see when you go to a web address with your browser. I’ll discuss this process in more detail later in this introduction (see “How PHP Works”).

Finally, PHP is cross-platform, meaning that it can be used on machines running Unix, Windows, Macintosh, and other operating systems. Again, we’re talking about the server’s operating system, not the client’s. Not only can PHP run on almost any operating system, but, unlike many other programming languages, it enables you to switch your work from one platform to another with few or no modifications.

As of this writing, PHP is simultaneously in versions 5.5.35, 5.6.21, and 7.0.6. (There are slight differences between versions 5.5 and 5.6, so 5.5 continues to be supported for a while.) Although I wrote this book using a stable version of PHP 7, all of the code is backward compatible, at least to PHP version 5.x. In a couple of situations where a feature requires a more current version of PHP, or where older versions might have slight variations, a note in a sidebar or a tip will indicate how you can adjust the code accordingly.

More information can be found at PHP.net and Zend (www.zend.com), a key company involved with PHP development. This Zend website contains useful software as well as a code gallery and well-written tutorials.

### What PHP Is Not

The thing about PHP that confuses most new learners is what PHP can’t do. Although you can use the language for an amazing array of tasks, its main limitation is that PHP cannot be used for client-side features found in some websites.

Using a client-side technology like JavaScript, you can create a new browser window, make pop-up dialogs, dynamically generate and alter forms, and much more. None of these tasks can be accomplished using PHP because PHP is server-side, whereas those are client-side issues. But you can use PHP to create JavaScript, just as you can use PHP to create HTML.

When it comes time to develop your own PHP projects, remember that you can use PHP only to send information (HTML and such) to the browser. You can’t do anything else within the browser until another request from the server has been made (a form has been submitted or a link has been clicked).
Why Use PHP?

Put simply, PHP is better, faster, and easier to learn than the alternatives. All websites must begin with just HTML, and you can create an entire site using a number of static HTML pages. But basic HTML is a limited approach that does not allow for flexibility or dynamic behavior. Visitors accessing HTML-only sites see simple pages with no level of customization or dynamic behavior. With PHP, you can create exciting and original pages based on whatever factors you want to consider. PHP can also interact with databases and files, handle email, and do many other things that HTML alone cannot.

Web developers learned a long time ago that HTML alone won’t produce enticing and lasting websites. Toward this end, server-side technologies such as PHP have become the norm. These technologies allow developers to create web applications that are dynamically generated, taking into account whichever elements the programmer desires. Often database-driven, these advanced sites can be updated and maintained more readily than static HTML pages.

When it comes to choosing a server-side technology, the primary alternatives to PHP are: ASP.NET (Active Server Pages), JSP (JavaServer Pages), Ruby (through the Rails or Sinatra frameworks), and some newer server-side JavaScript options such as Node.js.

A The Web Technology Surveys site says that PHP is running on 82 percent of all websites (http://w3techs.com/technologies/overview/programming_language/all).
So the question is, why should a web developer use PHP instead of ASP.NET, Node.js, or whatever else to make a dynamic website?

- **PHP is much easier to learn and use.** People—perhaps like you—without any formal programming training can write PHP scripts with ease after reading this one book. In comparison, ASP.NET requires an understanding of Visual Basic, C#, or another language; Node.js requires JavaScript. These are more complex languages and are much more difficult to learn.

- **PHP was written specifically for dynamic web page creation.** Perl, VBScript, Java, and Ruby were not, and this fact suggests that, by its very intent, PHP can do certain tasks faster and more easily than the alternatives. I’d like to make it clear, however, that although I’m suggesting that PHP is better for certain things—specifically those it was created to do, PHP isn’t a “better” programming language than JavaScript or C#—they can do things PHP can’t.

- **PHP is both free and cross-platform.** Therefore, you can learn and use PHP on nearly any computer and at no cost. Furthermore, its open source nature means that PHP’s users are driving its development, not some corporate entity.

- **PHP is the most popular tool available for developing dynamic websites.** As of this writing, PHP is in use on over 82 percent of all websites and is the sixth most popular programming language overall. Many of the biggest websites—Yahoo, Wikipedia, and Facebook, just to name three—and content management tools, such as WordPress, Drupal, Moodle, and Joomla, use PHP. By learning this one language, you’ll provide yourself with either a usable hobby or a lucrative skill.

![table]

<table>
<thead>
<tr>
<th>May 2016</th>
<th>May 2015</th>
<th>Change</th>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>Java</td>
<td>20.958%</td>
<td>+4.00%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>C</td>
<td>13.223%</td>
<td>-3.62%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>C++</td>
<td>6.698%</td>
<td>-1.18%</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td>C#</td>
<td>4.481%</td>
<td>-0.76%</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td>Python</td>
<td>3.789%</td>
<td>+0.06%</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td></td>
<td>PHP</td>
<td>2.992%</td>
<td>+0.27%</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
<td>JavaScript</td>
<td>2.340%</td>
<td>-0.78%</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td></td>
<td>Ruby</td>
<td>2.338%</td>
<td>+1.07%</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td></td>
<td>Perl</td>
<td>2.326%</td>
<td>+0.51%</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td></td>
<td>Visual Basic .NET</td>
<td>2.325%</td>
<td>-0.64%</td>
</tr>
</tbody>
</table>

1 The Tiobe Index (www.tiobe.com/tiobe_index) uses a combination of factors to rank the popularity of programming languages.
How PHP Works

PHP is a server-side language, which means the code you write in PHP resides on a host computer that serves web pages to browsers. When you go to a website (www.LarryUllman.com, for example), your Internet service provider (ISP) directs your request to the server that holds the www.LarryUllman.com information. That server reads the PHP code and processes it according to its scripted directions. In this example, the PHP code tells the server to send the appropriate web page data to your browser in the form of HTML A. In short, PHP creates an HTML page on the fly based on parameters of your choosing.

This differs from an HTML-generated site in that when a request is made, the server merely sends the HTML data to the browser—no server-side interpretation occurs B. Hence, to the end user's browser, there may or may not be an obvious difference between what home.html and home.php look like, but how you arrive at that point is critically altered. The major difference is that by using PHP, you can have the server dynamically generate the HTML code. For example, different information could be presented if it’s Monday as opposed to Tuesday or if the user has visited the page before. Dynamic web page creation sets apart the less appealing, static sites from the more interesting, and therefore more visited, interactive ones.

The central difference between using PHP and using straight HTML is that PHP does everything on the server and then sends the appropriate information to the browser. This book covers how to use PHP to send the right data to the browser.
What You’ll Need

The most important requirement for working with PHP—because it’s a server-side scripting language—is access to a PHP-enabled server. Considering PHP’s popularity, your web host most likely has this option available to you on their servers. You’ll need to contact them to see what technology they support.

Your other option is to install PHP and a web server application (like Apache) on your own computer. Users of Windows, Mac OS X, or Linux can easily install and use PHP for no cost. Directions for installing PHP are available in Appendix A, “Installation and Configuration.” If you’re up to the task of using your own PHP-installed server, you can take some consolation in knowing that PHP is available for free from the PHP website (www.php.net) and comes in easy-to-install packages. If you take this approach, and I recommend that you do, then your computer will act as both the client and the server.

The second requirement is almost a given: You must have a text editor on your computer. Atom, Notepad++, UltraEdit, and similar freeware applications are all sufficient for your purposes, and TextMate, SublimeText, and other commercial applications offer more features that you may appreciate. If you’re accustomed to using a graphical interface (also referred to as WYSIWYG—What You See Is What You Get) such as Adobe Dreamweaver A or Aptana Studio, you can consult that application’s manual to see how to program within it.

continues on next page
Third, you need a method of getting the scripts you write to the server. If you’ve installed PHP on your own computer, you can save the scripts to the appropriate directory. However, if you’re using a remote server with a web host, you’ll need an SFTP (Secure File Transfer Protocol) program to send the script to the server. There are plenty of SFTP applications available; for example, in Chapter 1, “Getting Started with PHP,” I use the free FileZilla (http://filezilla-project.org)

Finally, if you want to follow the examples in Chapter 12, “Intro to Databases,” you need access to MySQL (www.mysql.com). MySQL is available in a free version that you can install on your own computer.

This book assumes only a basic knowledge of HTML, although the more comfortable you are handling raw HTML code without the aid of a WYSIWYG application such as Dreamweaver, the easier the transition to using PHP will be. Every programmer will eventually turn to an HTML reference at some time or other, regardless of how much you know, so I encourage you to keep a good HTML book by your side. One such introduction to HTML is Elizabeth Castro and Bruce Hyslop’s HTML, XHTML, and CSS: Visual QuickStart Guide (Peachpit Press, 2014).

Previous programming experience is certainly not required. However, it may expedite your learning because you’ll quickly see numerous similarities between, for example, Perl and PHP or JavaScript and PHP.
About This Book

This book attempts to convey the fundamentals of programming with PHP while hinting at some of the more advanced features you may want to consider in the future, without going into overwhelming detail. It uses the following conventions to do so.

The step-by-step instructions indicate what coding you’re to add to your scripts and where. The specific text you should type is printed in a unique type style to separate it from the main body text. For example:

```php
<?php print "Hello, World!"; ?>
```

The PHP code is also written as its own complete script and is numbered by line for reference (Script i.1). You shouldn’t insert these line numbers yourself, because doing so will render your work inoperable.

I recommend using a text editor that automatically displays the line numbers for you—the numbers will help when you’re debugging your work. In the scripts, you’ll sometimes see particular lines highlighted in bold, in order to draw attention to new or relevant material.

What’s New in This Book?

I would consider this fifth edition to be a modest revision of an already solid book. The biggest changes are

- All examples now use HTML5.
- The MySQL code uses the most current version of PHP’s MySQL extension.
- We cover PHP 7, as applicable.

Finally, I tweaked some of the examples mostly to satisfy my own drive for perfection. No content from the previous edition has been removed.
Because of the nature of how PHP works, you need to understand that there are essentially three views of every script: the PHP code (e.g., Script 1.1), the code that’s sent to the browser (primarily HTML), and what the browser displays to the end user. Where appropriate, sections of, or all of, the browser window are revealed, showing the result of the exercise A. Occasionally, you’ll also see an image displaying the HTML source that the browser received B. You can normally access this view by choosing View Source or View Page Source from the appropriate browser menu. To summarize, B displays the HTML the browser receives, and A demonstrates how the browser interprets that HTML. Using PHP, you’ll create the HTML that’s sent to the browser.

A This is a sample view you’ll see of the browser window. For the purposes of this book, it won’t make any difference which browser or operating system you use.

B By viewing the source code received by the browser, you can see the HTML created by PHP and sent by the server.
Because the columns in this book are narrower than the common text editor screen, sometimes lines of PHP code printed in the steps have to be broken where they would not otherwise break in your editor. A small gray arrow indicates when this kind of break occurs. For example:

```
print "This is going to be a longer line of code.";
```

You should continue to use one line in your scripts, or else you'll encounter errors when executing them. (The gray arrow isn't used in scripts that are numbered.)

While demonstrating new features and techniques, I'll do my best to explain the why's and how's of them as I go. Between reading about and using a function, you should clearly comprehend it. Should something remain confusing, though, this book contains a number of references where you can find answers to any questions (see Appendix B, “Resources and Next Steps”). If you’re confused by a particular function or example, your best bet will be to check the online PHP manual or the book’s supporting website (and its user support forum).
Companion Website

While you're reading this book, you may also find it helpful to visit the *PHP for the Web: Visual QuickStart Guide, 5th Edition* website, found within www.LarryUllman.com. There you'll find every script in this book available in a downloadable form. However, I strongly encourage you to type the scripts yourself in order to become more familiar with the structure and syntax of PHP. The site also provides an errata page listing any mistakes made in this text.

What many users find most helpful, though, is the book's supporting forum, found through the website or more directly at www.LarryUllman.com/forums/. Using the forum, you can

- Find answers to problems you’re having
- Receive advice on how to approach an idea you have
- Get debugging help
- See how changes in the technologies have affected the examples in the book
- Learn what other people are doing with PHP
- Confirm the answers to review questions
- Receive a faster reply from me than if you send me a direct email

Which Book Is Right for You?

This is the fifth edition of my first book on PHP. Like the original, it's written with the beginner or nonprogrammer in mind. If you have little or no programming experience, prefer a gentler pace, or like to learn things in bite-sized pieces, this is the book for you. Make no mistake: This book covers what you need to know to begin developing dynamic websites and uses practical examples, but it does so without any in-depth theory or advanced applications.

Conversely, if you pick up new technologies really quickly or already have some experience developing websites, you may find this to be too basic. In that case, you should consider my *PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide* instead (Peachpit Press, 2012). It discusses SQL and MySQL in much greater detail and goes through several more complex examples, but it does so at a quick jog.
Questions, comments, or suggestions?

If you have a PHP-specific question, there are newsgroups, mailing lists, and question-and-answer sections available on PHP-related websites for you to turn to. These are discussed in more detail in Appendix B. Browsing through these references or searching the Internet will almost always provide you with the fastest answer.

You can also direct your questions, comments, and suggestions to me. You’ll get the fastest reply using the book’s corresponding forum; I always answer those questions first. If you’d rather email me, you can do so through the contact page on the website. I do try to answer every email I receive, but it will probably take a week or two (whereas you’ll likely get a reply in the forum within a couple of days).

For more tips and an enlightening read, see the sidebar on this page and Eric Steven Raymond’s “How to Ask Questions the Smart Way,” at www.catb.org/~esr/faqs/smart-questions.html. The 10 minutes you spend on it will save you hours in the future. Those people who will answer your questions, like myself, will be most appreciative!
This page intentionally left blank
Chapter 2, “Variables,” briefly discussed the various types of variables, how to assign values to them, and how they’re generally used. In this chapter, you’ll work specifically with number variables—both integers (whole numbers) and floating-point numbers (aka floats or decimals).

You’ll begin by creating an HTML form that will be used to generate number variables. Then you’ll learn how to perform basic arithmetic, how to format numbers, and how to cope with operator precedence. The last two sections of this chapter cover incrementing and decrementing numbers, plus generating random numbers. Throughout the chapter, you’ll also learn about other useful number-related PHP functions.
Creating the Form

Most of the PHP examples in this chapter will perform various calculations based on an e-commerce premise. A form will take price, quantity, discount amount, tax rate, and shipping cost, and the PHP script that handles the form will return a total cost. That cost will also be broken down by the number of payments the user wants to make in order to generate a monthly cost value.

To start, let’s create an HTML page that allows the user to enter the values.

To create the HTML form:

1. Begin a new HTML document in your text editor or IDE, to be named calculator.html (Script 4.1):

   ```html
   <doctype html>
   <html lang="en">
   <head>
     <meta charset="utf-8">
     <title>Product Cost Calculator</title>
   </head>
   <body><!-- Script 4.1 - calculator.html -->
   <div><p>Fill out this form to calculate the total cost:</p>

   2. Create the initial form tag:

   ```html
   <form action="handle_calc.php" method="post">
   This form tag begins the HTML form. Its action attribute indicates that the form data will be submitted to a page named handle_calc.php. The tag's method attribute tells the page to use POST to send the data. See Chapter 3, “HTML Forms and PHP,” for more details on choosing a method.
3. Create the inputs for the price, quantity, discount, and tax:

```html
<p>Price: <input type="text" name="price" size="5"></p>
<p>Quantity: <input type="number" name="quantity" size="5" min="1" value="1"></p>
<p>Discount: <input type="text" name="discount" size="5"></p>
<p>Tax: <input type="text" name="tax" size="5"> (%)</p>
```

Although HTML5 does have a number input type, it's not always the right solution because it's more naturally suited to taking integer values. For that reason, the quantity input will be a number type, whereas the others will be text.

To guide the user, a parenthetical indicates that the tax should be entered as a percent.

Remember that the names used for the inputs should correspond to valid PHP variable names: Use letters, numbers, and the underscore only; don't start with a number; and so forth.

continues on next page
4. Add a field in which the user can select a shipping method:

```html
<p>Shipping method: <select name="shipping">
  <option value="5.00">Slow and steady</option>
  <option value="8.95">Put a move on it.</option>
  <option value="19.36">I need it yesterday!</option>
</select></p>
```

The shipping selection is made using a drop-down menu. The value of the selected option is the cost for that option. If the user selects, for example, the "Put a move on it." option, the value of `$_POST['shipping']` in `handle_calc.php` will be 8.95.

5. Complete the HTML form:

```html
<p>Number of payments to make: <input type="number" name="payments" size="5" min="1" value="1"></p>
<input type="submit" name="submit" value="Calculate!">
</form>
```

The final two input types take a number for how many payments are required and then create a submit button (labeled "Calculate!"). The closing form tag marks the end of the form section of the page.

6. Complete the HTML page:

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

7. Save the script as `calculator.html`, and view it in your browser.

Because this is an HTML page, you can view it directly in a browser.
Performing Arithmetic

Just as you learned in grade school, basic mathematics involves the principles of addition, subtraction, multiplication, and division. These are performed in PHP using the most obvious operators:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)

To use these operators, you'll create a PHP script that calculates the total cost for the sale of some widgets. This handling script could be the basis of a shopping cart application—a very practical web page feature (although in this case the relevant number values will come from calculator.html).

When you're writing this script, be sure to note the comments (Script 4.2) used to illuminate the different lines of code and the reasoning behind them.

To create your sales cost calculator:

1. Create a new document in your text editor or IDE, to be named handle_calc.php (Script 4.2):

```php
// Script 4.2 - handle_calc.php
/* This script takes values from calculator.html and performs total cost and monthly payment calculations. */

// Address error handling, if you want.
// Get the values from the $_POST array:
$price = $_POST['price'];
$quantity = $_POST['quantity'];
$discount = $_POST['discount'];
$tax = $_POST['tax'];
$shipping = $_POST['shipping'];
$payments = $_POST['payments'];

// Calculate the total:
$total = $price * $quantity;
$total = $total + $shipping;
$total = $total - $discount;

// Determine the tax rate:
$taxrate = $tax / 100;
$taxrate = $taxrate + 1;

// Factor in the tax rate:
$total = $total * $taxrate;

// Calculate the monthly payments:
$monthly = $total / $payments;
```

code continues on next page
The head of the document defines one CSS class, named *number*. Any element within the page that has that class value will be given extra font weight. In other words, when the numbers from the form, and the results of the various calculations, are printed in the script's output, they'll be made more obvious.

2. Insert the PHP tag and address error handling, if desired:

```php
<?php // Script 4.2 - handle_calc.php

Depending on your PHP configuration, you may or may not want to add a couple of lines that turn on display_errors and adjust the level of error reporting. See Chapter 3 for specifics.

(However, as also mentioned in that chapter, it's best to make these adjustments in PHP's primary configuration file.)

3. Assign the $_POST elements to local variables:

```php
$price = $_POST['price'];
$quantity = $_POST['quantity'];
$discount = $_POST['discount'];
$tax = $_POST['tax'];
$shipping = $_POST['shipping'];
$payments = $_POST['payments'];
```

The script will receive all the form data in the predefined $_POST variable. To access individual form values, refer to $_POST['index'], replacing *index* with the corresponding form element's name value. These values are assigned to individual local variables here, to make it easier to use them throughout the rest of the script.

Note that each variable is given a descriptive name and is written entirely in lowercase letters.

### Script 4.2 continued

40 // Print out the results:
41 print "You have selected to purchase:<br>
42 <span class="number">$quantity</span> widget(s) at <br>$<span class="number">$price</span> price each plus a <br>$<span class="number">$shipping</span> shipping cost and a <br>$<span class="number">$tax</span> percent tax rate.<br>
43 After your $<span class="number">$discount</span> discount, the total cost is <br>$<span class="number">$total</span>.<br>
44 Divided over $<span class="number">$payments</span> monthly payments, that would be $<span class="number">$monthly</span> each.<p>";
4. Begin calculating the total cost:

\[
\text{$total = $price \times $quantity;}
\text{$total = $total + $shipping;}
\text{$total = $total - $discount;}
\]

The asterisk (*) indicates multiplication in PHP, so the total is first calculated as the number of items purchased ($quantity) multiplied by the price. Then the shipping cost is added to the total value (remember that the shipping cost correlates to the `value` attribute of each shipping drop-down menu's `option` tags), and the discount is subtracted.

Note that it's perfectly acceptable to determine a variable's value in part by using that variable's existing value (as is done in the last two lines).

5. Calculate the tax rate and the new total:

\[
\text{$taxrate = $tax / 100;}
\text{$taxrate = $taxrate + 1;}
\text{$total = $total \times $taxrate;}
\]

The tax rate should be entered as a percent—for example, 8 or 5.75. This number is then divided by 100 to get the decimal equivalent of the percent (0.08 or 0.0575). Finally, you calculate how much something costs with tax by adding 1 to the percent and then multiplying that new rate by the total. This is the mathematical equivalent of multiplying the decimal tax rate times the total and then adding this result to the total (for example, a 5 percent tax on $100 is $5, making the total $105, which is the same as multiplying $100 times 1.05).

6. Calculate the monthly payment:

\[
\text{$monthly = $total / $payments;}
\]

As an example of division, assume that the widgets can be paid for over the course of many months. Hence, you divide the total by the number of payments to find the monthly payment.

7. Print the results:

```php
print "<p>You have selected to<br>→ purchase:<br><span class="number">$quantity</span> widget(s) at<br><span class="number">$price</span> price each plus a<br><span class="number">$shipping</span> shipping cost and a<br><span class="number">$tax</span> percent tax rate.<br>After your $<span class="number">$discount</span> discount, the total cost is<br><span class="number">$total</span>.<br>Divided over $<span class="number">$payments</span> monthly payments, that would be<br><span class="number">$monthly</span> each.</p>";
```

The `print` statement sends every value to the browser along with some text. To make it easier to read, `<br>` tags are added to format the browser result; in addition, the `print` function operates over multiple lines to make the PHP code cleaner. Each variable's value will be highlighted in the browser by wrapping it within span tags that have a `class` attribute of `number` (see Step 1).

8. Close the PHP section, and complete the HTML page:

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

9. Save the script as `handle_calc.php` and place it in the proper directory for your PHP-enabled server.

Make sure that `calculator.html` is in the same directory.

continues on next page
10. Test the script in your browser by filling out A and submitting B the form.

Not to belabor the point, but make sure you start by loading the HTML form through an URL (http://something) so that when it’s submitted, the PHP script is also run through a URL.

You can experiment with these values to see how effectively your calculator works. If you omit any values, the resulting message will just be a little odd but the calculations should still work C.

**Tip** As you’ll certainly notice, the calculator comes up with numbers that don’t correspond well to real dollar values (see B and C). In the next section, “Formatting Numbers,” you’ll learn how to address this issue.

**Tip** If you want to print the value of the total before tax or before the discount (or both), you can do so in two ways. You can insert the appropriate print statements immediately after the proper value has been determined but before the $total variable has been changed again. Or you can use new variables to represent the values of the subsequent calculations (for example, $total_with_tax and $total_less_discount).

**Tip** Attempting to print a dollar sign followed by the value of a variable, such as $10 (where 10 comes from a variable), has to be handled carefully. You can’t use the syntax $$variable, because the combination of two dollar signs creates a type of variable that’s too complex to discuss in this book. One solution is to put something—a space or an HTML tag, as in this example—between the dollar sign and the variable name. Another option is to escape the first dollar sign:

```php
print "The total is \$total";
```

A third option is to use concatenation, which is introduced in the next chapter.

**Tip** This script performs differently, depending on whether the various fields are submitted. The only truly problematic field is the number of monthly payments: If this is omitted, you’ll see a division-by-zero warning. Chapter 6, “Control Structures,” will cover validating form data before it’s used.
Formatting Numbers

Although the calculator is on its way to being practical, it still has one legitimate problem: You can’t ask someone to make a monthly payment of $10.13183333! To create more usable numbers, you need to format them.

Two functions are appropriate for this purpose. The first, round(), rounds a value to a specified number of decimal places. The function’s first argument is the number to be rounded. This can be either a number or a variable that has a numeric value. The second argument is optional; it represents the number of decimal places to which to round. If omitted, the number will be rounded to the nearest integer. For example:

```
round(4.30); // 4
round(4.289, 2); // 4.29
$num = 236.26985;
round($num); // 236
```

The other function you can use in this situation is number_format(). It works like round() in that it takes a number (or a variable with a numeric value) and an optional decimal specifier. This function has the added benefit of formatting the number with commas, the way it would commonly be written:

```
number_format(428.4959, 2); // 428.50
number_format(428, 2); // 428.00
number_format(1234567); // 1,234,567
```

Let’s rewrite the PHP script to format the numbers appropriately.
To format numbers:

1. Open handle_calc.php in your text editor or IDE, if it is not already open (Script 4.2).
2. After all the calculations but before the print statement, add the following (Script 4.3):

   ```php
   $total = number_format($total, 2);
   $monthly = number_format($monthly, 2);
   
   To format these two numbers, apply this function after every calculation has been made but before they're sent to the browser. The second argument (the 2) indicates that the resulting number should have exactly two decimal places; this setting rounds the numbers and adds zeros at the end, as necessary.
   ```

---

Script 4.3 The `number_format()` function is applied to the values of two number variables, so they are more appropriate to the example.

```php
<?php // Script 4.3 - handle_calc.php #2

    // This script takes values from calculator.html and performs total cost and monthly payment calculations. */

    // Address error handling, if you want.

    // Get the values from the $_POST array:
    $price = $_POST['price'];
    $quantity = $_POST['quantity'];
    $discount = $_POST['discount'];
    $tax = $_POST['tax'];
    $shipping = $_POST['shipping'];
    $payments = $_POST['payments'];

    // Calculate the total:
    $total = $price * $quantity;
    $total = $total + $shipping;
    $total = $total - $discount;

    // Determine the tax rate:
    $taxrate = $tax/100;
    $taxrate = $taxrate + 1;

    // Factor in the tax rate:
    $total = $total * $taxrate;

    // Calculate the monthly payments:
    $monthly = $total / $payments;

    // Apply the proper formatting:
    $total = number_format($total, 2);
    $monthly = number_format($monthly, 2);

    code continues on next page
```
3. Save the file, place it in the same directory as calculator.html, and test it in your browser A and B.

**tip** Another, much more complex way to format numbers is to use the printf() and sprintf() functions. Because of their tricky syntax, they’re not discussed in this book; see the PHP manual for more information.

**tip** Non-Windows versions of PHP also have a money_format() function, which can be used in lieu of number_format().

**tip** The round() function rounds exact halves (.5, .05, .005, and so on) up, although this behavior can be configured. See the PHP manual for details.

**tip** In PHP, function calls can have spaces between the function name and its parentheses or not. Both of these are fine:

    round ($num);
    round($num);

**tip** The number_format() function takes two other optional arguments that let you specify what characters to use to indicate a decimal point and break up thousands. This is useful, for example, for cultures that write 1,000.89 as 1.000,89. See the PHP manual for the correct syntax, if you want to use this option.

---

Script 4.3 continued

```php
44 // Print out the results:
45 print "<p>You have selected to purchase:<br>46 <span class="number">$quantity</span> widget(s) at<br>47 $<span class="number">$price</span> price each plus a<br>48 $<span class="number">$shipping</span> shipping cost and a <br>49 <span class="number">$tax</span> percent tax rate.<br>50 After your $<span class="number">$discount</span> discount, the total cost is<br>51 $<span class="number">$total</span>.<br>52 Divided over $<span class="number">$payments</span> monthly payments, that would be $<span class="number">$monthly</span> each.<br>53 ?>
54 </body>
56 </html>
```

A Another form entry.

B The updated version of the script returns more appropriate number values thanks to the number_format() function.
Understanding Precedence

Inevitably, after a discussion of the various sorts of mathematical operators comes the discussion of precedence. *Precedence* refers to the order in which a series of calculations are executed. For example, what is the value of the following variable?

```php
$number = 10 - 4 / 2;
```

Is `$number` worth 3 (10 minus 4 equals 6, divided by 2 equals 3) or 8 (4 divided by 2 equals 2, subtracted from 10 equals 8)? The answer here is 8, because division takes precedence over subtraction.

Appendix B, "Resources and Next Steps," shows the complete list of operator precedence for PHP (including operators that haven’t been covered yet). However, instead of attempting to memorize a large table of peculiar characters, you forgo any deliberation by using parentheses. Parentheses always take precedence over any other operator. Thus:

```php
$number = (10 - 4) / 2; // 3
$number = 10 - (4 / 2); // 8
```

Using parentheses in your calculations ensures that you never see peculiar results due to precedence issues. Parentheses can also be used to rewrite complex calculations in fewer lines of code. Let’s rewrite the `handle_calc.php` script, combining multiple lines into one by using parentheses, while maintaining accuracy.

To manage precedence:

1. Open `handle_calc.php` in your text editor or IDE, if it is not already open (Script 4.3).

---

Script 4.4 By using parentheses, calculations made over multiple lines (compare with Script 4.3) can be condensed without affecting the script’s mathematical accuracy.

```php
// This script takes values from calculator.html and performs total cost and monthly payment calculations.*/

// Address error handling, if you want.

// Get the values from the $_POST array:

$price = $_POST['price'];
$quantity = $_POST['quantity'];
$discount = $_POST['discount'];
$tax = $_POST['tax'];
$shipping = $_POST['shipping'];
$payments = $_POST['payments'];

// Calculate the total:

$total = (($price * $quantity) + $shipping) - $discount;

// Determine the tax rate:

$taxrate = ($tax / 100) + 1;

// Factor in the tax rate:

$total = $total * $taxrate;

// Calculate the monthly payments:

$monthly = $total / $payments;

// Apply the proper formatting:

$total = number_format($total, 2);
$monthly = number_format($monthly, 2);
```

code continues on next page
Replace the three lines that initially calculate the order total with the following (Script 4.4):

```
$total = (($price * $quantity) + $shipping) - $discount;
```

In this script, it’s fine to make all the calculations in one step, as long as you use parentheses to ensure that the math works properly. The other option is to memorize PHP’s rules of precedence for multiple operators, but using parentheses is a lot easier.

Change the two lines that calculate and add in the tax to this:

```
$taxrate = ($tax / 100) + 1;
```

Again, the tax calculations can be made in one line instead of two separate ones.

Save the script, place it in the same directory as `calculator.html`, and test it in your browser.

Be sure that you match your parentheses consistently as you create your formulas (every opening parenthesis requires a closing parenthesis). Failure to do so will cause parse errors.

Granted, using the methods applied here, you could combine all the total calculations into just one line of code (instead of three)—but there is such a thing as oversimplifying.

Testing the form one more time.

Even though the calculations have been condensed, the math works out the same. If you see different results or get an error message, double-check your parentheses for balance (an equal number of opening and closing parentheses).
Incrementing and Decrementing a Number

PHP, like most programming languages, includes shortcuts that let you avoid ugly constructs such as

```php
$tax = $tax + 1;
```

When you need to increase the value of a variable by 1 (known as an *incremental* adjustment) or decrease the value of a variable by 1 (a *decremental* adjustment), you can use `++` and `--`, respectively:

```php
$var = 20; // 20
$var++; // 21
$var++; // 22
$var--; // 21
```

Solely for the sake of testing this concept, you'll rewrite the `handle_calc.php` script one last time.

**To increment the value of a variable:**

1. Open `handle_calc.php` in your text editor or IDE, if it is not already open (Script 4.4).
2. Change the tax rate calculation from Script 4.3 to read as follows (Script 4.5):

   ```php
   $taxrate = $tax / 100;
   $taxrate++;;
   
   The first line calculates the tax rate as the `$tax` value divided by 100. The second line increments this value by 1 so that it can be multiplied by the total to determine the total with tax.
   
3. Save the script, place it in the same directory as `calculator.html`, and test it in your browser.

---

**Script 4.5** Incrementing or decrementing a number is a common operation using `++` or `--`, respectively.

```php
<?php // Script 4.3 - handle_calc.php #4 /* This script takes values from calculator.html and performs total cost and monthly payment calculations. */

// Address error handling, if you want.

// Get the values from the $_POST array:
$price = $_POST['price'];
$quantity = $_POST['quantity'];
$discount = $_POST['discount'];
$tax = $_POST['tax'];
$shipping = $_POST['shipping'];
$payments = $_POST['payments'];

// Calculate the total:
$total = ($price * $quantity) + $shipping - $discount;

// Determine the tax rate:
$taxrate = $tax / 100;
$taxrate++;

// Factor in the tax rate:
$total = $total * $taxrate;

// Calculate the monthly payments:
$monthly = $total / $payments;

// Apply the proper formatting:
$total = number_format ($total, 2);
$monthly = number_format ($monthly, 2);
```

(code continues on next page)
Although functionally it doesn't matter whether you code $taxrate = $taxrate + 1; or the abbreviated $taxrate++, the latter method (using the increment operator) is more professional and common.

In Chapter 6, you'll see how the increment operator is commonly used in conjunction with loops.

**Arithmetic Assignment Operators**

PHP also supports a combination of mathematical and assignment operators. These are +=, -=, *=, and /=. Each will assign a value to a variable by performing a calculation on it. For example, these next two lines both add 5 to a variable:

```php
$num = $num + 5;
$num += 5;
```

This means the handle_calc.php script could determine the tax rate using this:

```php
$tax = $_POST['tax']; // Say, 5
$tax /= 100; // Now $tax is .05
$tax += 1; // 1.05
```

You'll frequently see these shorthand ways of performing arithmetic.

The last execution of the form.

It won’t affect your calculations if you use the long or short version of incrementing a variable (compare Scripts 4.4 and 4.5).
Creating Random Numbers

The last function you’ll learn about in this chapter is `mt_rand()`, a random-number generator. All it does is output a random number:

```php
$n = mt_rand(); // 31
$n = mt_rand(); // 87
```

The `mt_rand()` function can also take minimum and maximum parameters, if you prefer to limit the generated number to a specific range:

```php
$n = mt_rand(0, 10);
```

These values are inclusive, so in this case 0 and 10 are feasible returned values.

As an example of generating random numbers, let’s create a simple “Lucky Numbers” script.

### To generate random numbers:

1. Begin a new document in your text editor or IDE, to be named `random.php` (Script 4.6):

```html
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Lucky Numbers</title>
</head>
<body>
<p>Your lucky numbers are:<br>$n1<br>$n2<br>$n3</p>
</body>
</html>
```

2. Include the PHP tag and address error management, if you need to:

```php
<?php // Script 4.6 - random.php
/* This script generates 3 random numbers. */

// Create three random numbers:
$n1 = mt_rand(1, 99);
$n2 = mt_rand(1, 99);
$n3 = mt_rand(1, 99);

// Print out the numbers:
print "Your lucky numbers are:<br>
$n1<br>$n2<br>$n3";
?>
</body>
</html>
```
3. Create three random numbers:

$\text{n1} = \text{mt_rand}(1, 99);
$\text{n2} = \text{mt_rand}(1, 99);
$\text{n3} = \text{mt_rand}(1, 99);

This script prints out a person's lucky numbers, like those found on the back of a fortune cookie. These numbers are generated by calling the \text{mt_rand()} function three separate times and assigning each result to a different variable.

4. Print out the numbers:

\text{print "<p>Your lucky numbers are:<br>\n$\text{n1}<br>$\text{n2}<br>$\text{n3}</p>";}

The \text{print} statement is fairly simple. The numbers are printed, each on its own line, by using the HTML break tag.

5. Close the PHP code and the HTML page:

?>
</body>
</html>

6. Save the file as \text{random.php}, place it in the proper directory for your PHP-enabled server, and test it in your browser. Refresh the page to see different numbers.

- \text{The getrandmax()} function returns the largest possible random number that can be created using \text{mt_rand()}. This value differs by operating system.

- PHP has other functions for generating random numbers, such as \text{random_int()}. Unlike \text{mt_rand()}, \text{random_init()} creates cryptographically secure random numbers.
Review and Pursue

If you have any problems with the review questions or the pursue prompts, turn to the book’s supporting forum (www.LarryUllman.com/forums/).

Review

- What are the four primary arithmetic operators?
- Why will the following code not work:
  ```
  print "The total is $$total";
  ```
  What must be done instead?
- Why must an HTML page that contains a form that’s being submitted to a PHP script be loaded through a URL?
- What functions can be used to format numerical values? How do you format numbers to a specific number of decimals?
- What is the importance of operator precedence?
- What are the incremental and decremental operators?
- What are the arithmetic assignment operators?

Pursue

- Look up the PHP manual page for one of the new functions mentioned in this chapter. Use the links on that page to investigate a couple of other number-related functions that PHP has.
- Create another HTML form for taking numeric values. Then create the PHP script that receives the form data, performs some calculations, formats the values, and prints the results.

Other Mathematical Functions

PHP has a number of built-in functions for manipulating mathematical data. This chapter introduced `round()`, `number_format()`, and `mt_rand()`.

PHP has broken `round()` into two other functions. The first, `ceil()`, rounds every number to the next highest integer. The second, `floor()`, rounds every number to the next lowest integer.

Another function the calculator page could make good use of is `abs()`, which returns the absolute value of a number. In case you don’t remember your absolute values, the function works like this:

```
$number = abs(-23); // 23
$number = abs(23); // 23
```

In layman’s terms, the absolute value of a number is always a positive number.

Beyond these functions, PHP supports all the trigonometry, exponent, base conversion, and logarithm functions you’ll ever need. See the PHP manual for more information.
## Numbers
- **0666**, explained, 302
- **0777** permissions, 330

## Symbols
- `//`, using with comments, 24
- `/*` and `*/`, using with comments, 24, 26
- `=` assignment operator, 89
- `+=` assignment operator, 89, 457
- `-=` assignment operator, 89, 457
- `*=` assignment operator, 89
- `#`, using with comments, 24
- `?>` tag, 9
- `<? and `?>` short tags, 9
- `<!-- and -->`, using with comments, 25
- `+` (addition) operator, 79, 135, 457
- `&` (ampersand), using with forms, 68
- `&&` (and) logical operator, 135, 139, 457
- `*` (assignment) operator, 89, 135, 457
- `\` (backslash), using with strings, 39
- `&` (bitwise) operator, 310
- `[]` (brackets), using with keys in arrays, 161
- `{}` (braces)
  - versus parentheses `()`, 172
  - using with conditionals, 143
  - using with `if` conditional, 125
- `. ` (concatenation) operator, 97, 135, 457
- `--` (decrement) operator, 88–89, 135, 457
- `/` (division) operator, 79, 135, 457
- `$` (dollar sign)
  - preceding variables with, 36, 58
  - printing, 82
- `/\` (double backslashes), using with absolute paths, 329

- `"` (double quotation marks)
  - effect of, 44–47
  - parse error generated by, 170
  - versus single quotation marks `'`, 169
  - using with constants, 207
  - using with `print`, 17, 21
  - using with strings, 39
  - `'` (empty string), using with functions, 284
- `/` (equality) operator, 135
- `==` (equality) operator, 135, 457
- `=` (equals sign), using with variables, 41
- `>` (greater than) operator, 135, 457
- `>=` (greater than or equal to) operator, 135, 457
- `+=` (increment) operator, 88–89, 135, 457
- `<=` (less than or equal to) operator, 135, 457
- `<!` (negation) logical operator, 135, 457
- `??` (null coalescing) logical operator, 135, 143
- `=>` (null coalescing) logical operator, 135, 143
- `|` (or) logical operator, 135, 139, 457
- `..` (parent folder), 303
- `()` (parentheses)
  - versus braces `{}`, 172
  - using in calculations, 86–87
  - using with conditionals, 143
- `(pipe), explained, 67
- `;` (semicolon)
  - error related to, 65
  - using in MySQL client, 438
  - using with `print` command, 15
(single quotation marks) using, 44
to versus double quotation marks ("), 169
<> (spaceship) operator, 135, 138, 457
- (subtraction) operator, 79, 135, 457
_ (underscore) using with forms, 51
using with functions, 270
using with variables, 37

A
absolute paths, 203, 303, 329
access to pages, denying and troubleshooting, 405, 453
action attribute, including in forms, 50, 53, 57
add_entry.php document creating, 361–365
opening, 368
add_quote.php document creating, 306–309, 405–408
opening, 311–312
addition (+) operator, 79, 135, 457
addslashes() function, 370
administrator. See is_administrator() function
Adobe Dreamweaver, 4
alphabetical sort, performing on arrays, 184
ALTER privileges, 446
ALTER SQL command, 346
AM or PM, formatting with date() function, 211, 458
am or pm, formatting with date() function, 211, 458
ampersand (&), using with forms, 68
AMPPS website, 428, 433
And (&&) logical operator, 135, 139, 457
AND logical operator, 135, 139, 457
Apache, 10
Aptana Studio, 4
arguments
passing, 277
setting default values, 282–284
using with functions, 276–281
arithmetic, performing, 79–82
arithmetic operators, 89, 135, 457
array elements accessing, 161, 163, 170–172, 177
adding, 167–168
deleting, 166
entering, 165
pointing to, 173
array() function, 162–163
array values, printing, 171–172
arrays. See also multidimensional arrays
adding items to, 166–169
creating, 162–165
creating from HTML forms, 186–190
deleting, 166
explained, 160
indexes and keys in, 161
merging, 169
parse errors, 170
printing, 164
versus scalar variable types, 160
sorting, 178–181
syntactical rules, 161
transforming between strings, 182–185
using, 40
asort() functions, using with arrays, 178–180
.aspx extension, 9
assignment operator, 89, 135
associative arrays, 40
Atom, 4

B
backslash (\), using with strings, 39
basename() function, 329
binary digits, 310
birth year, creating input for, 123
Bitnami website, 428, 433
bitwise (&), 310
blank pages, troubleshooting, 452
<body> section, creating, 5
$books multidimensional array, 174–176
bool type, 281
Boolean TRUE and FALSE, 121, 125, 131, 139, 395. See also false value
braces ({}), 172
  versus parentheses (()), 172
  using with conditionals, 143
  using with if conditional, 125
brackets ([[]), using with keys in arrays, 161
break language construct, 148
buffer size, setting, 236

C
  calculations, performing, 76–78
  calculator1.php document
    creating, 286–289
    opening, 293
  calculator.html script, creating, 76–78
camel-hump and camel-case
consventions, 37
case-sensitive searches, performing, 117
class character set, setting for database, 392
class characters, escaping, 62
checkboxes
  confirming, 142
  creating for HTML form, 124
  presetting status of, 227
closing tag, adding, 5
combined operators, 457
comments, adding to scripts, 24–26
concatenating strings, 97–100
congenation (,) operator, 97, 135, 457
conditionals. See also nesting conditionals
  best practices, 143
  explained, 121
  nesting, 139
  troubleshooting, 454
  using functions in, 131
configuration changes, confirming, 437
configuring PHP, 436–437
constants. See also predefined constants
  benefits, 210
header.html file, 209
  naming, 210
  printing, 209–210
  and superglobal arrays, 294
  using, 207–210
control panel
  creating for directory, 326–329
  viewing file permissions in, 301
control structures
  comparison operators, 135–138
  default action, 132
  die language construct, 150
  else statement, 132–134
  elseif statement, 144–147
  HTML form for, 122–124
  if conditional, 125–127
  logical operators, 138–143
  for loop, 152–156
  switch conditional, 148–151
  validation functions, 128–131
  while loop, 156
$_COOKIE array, 251
cookie data, retrieving with PHP, 251–253
cookies
  adding parameters to, 254–256
  checking for presence of, 395
  comparing to sessions, 260–261
  creating, 246–250
  data limitation, 250
  debugging, 244
  deleting, 257–259
  encoding values of, 253
  expiration value, 254–255
  explained, 244–245
  httponly argument, 255
  path and domain arguments, 254–256
  reading from, 251–253
  security issues, 245, 252, 255
  sending, 247–250
  setting expiration date, 255–256
  testing safety of, 250
  transmitting and receiving, 245
  using tabs and newlines with, 252
  using to identify administrators, 393
copying files on servers, 324
count() function, using with arrays, 167
CREATE DATABASE command, 445, 447
CREATE privileges, 446
CREATE SQL command, 346
CREATE TABLE SQL command, 356–357
create_table.php document, creating, 357–359
creating documents, 4
CSS (Cascading Style Sheets)
  basics, 3
  font size and color, 251
css folder, creating, 204
CSS templates, 200. See also templates
CSV (comma-separated values) format, 338
customize.php document
  creating, 247–250
  opening, 255
Cut and Paste, using with templates, 199

D
database connections, making, 348–351
database information, best practices, 351
databases. See also MySQL databases;
  query data
    connection code, 359, 392
definition, 346
deleting data in, 376–381
inserting data into, 360–365
permissions, 352
resources, 450
retrieving data from, 371–375
updating data in, 382–387
date and time functions
  table, 458
  working with, 211–213
date() and time() functions
  table, 458
  using, 211–213, 254
  using with sessions, 265
DateTime class, 213
day pull-down menu, creating, 272
daylight savings, formatting with date() function, 211, 458
days, formatting with date() function, 211, 458
$dbc conditional, 348, 359
DBMS (database management system), 345, 347
debugging
  PHP scripts, 440
  steps, 27–28
decrement (-- operator, 88–89, 135, 457
decrementing numbers, 88–89
decrypting data, 112
default argument values, 282–284
default case, using with switch conditional, 151
DELETE FROM tablename query, 381
DELETE privileges, 446
DELETE query, running on databases, 376–381
DELETE SQL command, 346
delete_entry.php script, writing, 376–381
delete_quote.php document, creating, 418–421
deleting
  arrays and array elements, 166
cookies, 257–259
data in databases, 376–381
files, 324
quotes, 418–421
sessions, 266–267
delineated format, explained, 338
denying access to pages, 405
deprecated function, explained, 20
die() and exit() functions, 354
directories. See also web root directory
  creating, 330–337
  displaying contents of, 326–327
  navigating, 325–329
  permissions, 302
directory control panel, creating, 326–329
dirname() function, 329
display_errors setting
  using, 63–64
  using with cost calculator, 80
  using in debugging, 28
division (/) operator, 79, 135, 457
documents, creating, 4
dollar sign ($)
  preceding variables with, 36, 58
  printing, 82
double backslashes (\), using with absolute
  paths, 329
double quotation marks ("")
  effect of, 44–47
  parse error generated by, 170
  versus single quotation marks ('), 169
  using with print, 17, 21
  using with strings, 39
double-precision floating-point numbers, 38
doubles, 38
DROP SQL command, 346, 446
drop-down menu, creating for HTML
  form, 124
E
Edit menu, accessing for templates, 199
edit_entry.php document, creating, 383–387
edit_quote.php document, creating, 412–417
else statement, 132–134
elseif statement, 144–147
email, sending, 228–232
e-mail address
creating inputs for, 123
validating, 129
empty() function, 128, 131
empty string (" "), using with functions, 284.
See also strings
encoding
explained, 5
external files, 206
encrypting
data, 112
passwords, 337
ENTRIES table, columns in, 356
equality (/ and ==) operator, 135, 457
equals sign (=), using with variables, 41
error codes for files, 317
Error level, 65
error messages. See also parse errors;
troubleshooting
Add a Blog Entry, 364
arguments, 277
connection attempt refused, 10
Could not connect to the database, 350
Could not create the table, 358
Delete an Entry, 381
displaying in scripts, 63–64
double quotation marks ("), 21
e-mail address and password, 402
foreach loop, 176
functions, 275
header() call, 233
include() function, 201
nonexisting variables, 61
Not Found, 14
output buffering, 233
permission denied, 299
for registration results, 142
related to color selection, 147
related to external files, 201, 206
related to header() call, 233
require() function, 201
setcookie() function, 246
trusting, 28, 451
unassigned value, 72
undefined function call, 275
Undefined variable, 43
error reporting, 65–67
error suppression operator, 354
error_reporting levels and constants, 65–67
event.html document, creating, 186–187
event.php document, creating, 188–190
everyone permission, 298, 301
exclusive or (XOR) logical operator, 139
eexecute permission, 298
exit() and die() functions, 237, 354
explode() function
and fgets(), 338
using with arrays, 182, 184
external files. See also file extensions
benefits, 206
closing PHP tag, 206
using, 201–206
writing to, 306–309
FALSE and TRUE, 121
false value, 19. See also Boolean TRUE and FALSE
fclose() function, 305
feedback.html document
creating, 51
opening, 56
feof() function, 338
fgetcsv() function, 338, 342
fgets() function, 338, 348
file error codes, 317
file extensions. See also external files
being aware of, 9
and included files, 206
file() function, 313, 338
file navigation, 203
file paths, 303
file permissions, 298–302, 352
FILE privileges, 446
file uploads, handling, 316–324
FILE_APPEND constant, 303–304
file_exists() function, 300
file_get_contents() function, 313
fileatime() function, 329
filemtime() function, 328
filename() function, 325
fileperms() function, 329
files. See also saving documents and scripts
copying on servers, 324
deleting, 324
locking, 310–312
organizing, 204
reading from, 313–315
reading incrementally, 338–342
writing to, 303–309
$_FILES array, elements of, 317
filesize() function, 328
filter() function, 131
finfo_file() function, 329
firewalls, 429
first name, checking entry of, 223
flag variable, creating for sticky form, 222
float type, 281
floating-point numbers, 38
flock() lock types, 310
folders and files, organizing, 204
font size and color, setting in CSS, 251
footer, adding to template, 197
footer file, creating for template, 200
footer.html document
  creating, 398–399
  opening, 212
fopen() function, 305, 348
for loop, 152–156
  using with functions, 272
  using with numerically indexed arrays, 172
foreach loop
  error generated by, 176, 189
  using with array elements, 170–172
  using with directory control panel, 328
  using with functions, 272
  using with multidimensional arrays, 177
form data. See also HTML forms; sticky forms
  accessing, 62
  displaying, 62
  processing, 217
  receiving in PHP, 58–62
  sending to pages manually, 68–72
  validating, 128–131
form methods, choosing, 54–57
form submission, determining, 214–215
form tags, 50
  creating, 122
  using with functions, 274
formatting numbers, 83–85
forms. See HTML forms
forums, 96
frameworks, 455–456
function keyword, 271
function_exists() function, 275
functions. See also PHP functions; undefined functions; user-defined functions
  accessing, 281
  arguments, 276–281
  with arguments and value, 287
  best practice, 275
  calling without arguments, 282
  creating and calling, 272–275
  default argument values, 282–285
  defining with parameters, 276–277
  design theory, 295
  error related to, 65
  invoking, 271
  looking up definitions of, 18–20
  naming conventions, 270
return statement, 285
using spaces with, 85
using within conditionals, 131
functions.php script
code, 397
creating, 394–395
fwrite() function, 305, 348
G
garbage collection, 267
A Gentle Introduction to SQL website, 450
$_GET and $_POST, 55–62, 68
$_GET array, 161
GET method, using with HTTP headers, 240
getrandmax() function, explained, 91
Git version control software, 11
\texttt{glob()} function, 329
\texttt{global} statement, 290–294
GMT difference, formatting with \texttt{date()} function, 211–212, 458
\texttt{GRANT} privileges, 446–448
greater than (>) operator, 135, 457
greater than or equal to (\geq) operator, 135, 457
\$greeting variable, 97
grocery list array, 160

\textbf{H}
\begin{itemize}
\item \texttt{handle\_form.php} document
  \begin{itemize}
  \item creating, 59
  \item opening, 66
  \end{itemize}
\item \texttt{handle\_post.php} document
  \begin{itemize}
  \item creating, 79–82, 98–99
  \item opening, 84, 86, 88, 101, 106, 109, 115, 118
  \end{itemize}
\item \texttt{handle\_reg.php} document
  \begin{itemize}
  \item creating, 126–127
  \item opening, 129, 132, 136, 140, 145, 149
  \end{itemize}
\item hash, 40
\item \texttt{\textless\head\textgreater} tag, creating, 5
\item header file, creating for template, 198–199, 203
\item \texttt{header()} function
  \begin{itemize}
  \item and HTTP headers, 237–240
  \item and output buffering, 233
  \item using \texttt{exit} with, 150
  \end{itemize}
\item header lines, creating, 4
\item \texttt{header.html} document
  \begin{itemize}
  \item creating, 396–397
  \item opening, 209, 234
  \end{itemize}
\item headers already sent error, troubleshooting, 453
\item \texttt{headers\_sent()} function, 240
\item \texttt{Hello, World!} greeting, sending to browser, 2, 16–17
\item \texttt{hello1.php} document
  \begin{itemize}
  \item creating, 16–17
  \item opening, 21
  \end{itemize}
\item \texttt{hello2.php} document
  \begin{itemize}
  \item creating, 21–22
  \item opening, 25
  \end{itemize}
\item \texttt{hello.html} script, creating, 69–70
\end{itemize}
hidden input, checking for, 219
home page, creating, 422–425
hours, formatting with \texttt{date()} function, 211, 458
HTML (Hypertext Markup Language)
  \begin{itemize}
  \item current version, 2
  \item resources, 6
  \item sending to browsers, 21–23
  \item syntax, 2
  \end{itemize}
HTML comments, accessing, 26
\texttt{.html} extension, 9
HTML forms. See also \texttt{form data}; sticky forms
  \begin{itemize}
  \item control structures, 122–124
  \item for cookies, 249
  \item creating, 50–53
  \item creating arrays from, 186–190
  \item displaying and handling, 214, 216–219
  \item \texttt{event.php} page, 187–190
  \item handling, 59–61
  \item handling with PHP, 214–219
  \item hidden type of input in, 62
  \item making sticky, 220–227
  \item for numbers, 76–78
  \item radio-button value, 62
  \item re-displaying, 219
  \item for strings, 94–96
  \item for strings and arrays, 183–185
  \end{itemize}
HTML pages
  \begin{itemize}
  \item creating, 4–6
  \item example, 6
  \item versus PHP scripts, 7
  \item viewing source, 23
  \end{itemize}
HTML source code, checking, 28
HTML tags
  \begin{itemize}
  \item addressing in PHP, 106–107
  \item using PHP functions with, 104–107
  \end{itemize}
\texttt{\textless/\html\textgreater} tag, adding, 5
HTML5, 2
\texttt{htmlentities()} function, 384–385
\texttt{htmlspecialchars()} function, 328
HTTP (Hypertext Transfer Protocol), 237
HTTP headers, manipulating, 237–240

\textbf{I}
\begin{itemize}
\item \texttt{id} primary key, 387, 391
\item \texttt{if} conditional, 121, 125–127, 140
\end{itemize}
if-else conditional, 132–134, 143
if-elseif conditionals, simplifying, 148–150
if-elseif-else conditional, 144–147
IIS (Internet Information Server), 10
implode() function, using with arrays, 182, 184
include() function
  failure of, 201
  and parentheses (()), 206
  using with constants, 207
  using with external files, 202
increment (++) operator, 88–89, 135, 457
index errors, troubleshooting, 452
INDEX privileges, 446
indexed arrays, 40, 165
index.php document, creating,
  202–205, 423–425
inequality (%) operator, 135
inequality (!=) operator, 457
ini_set() function, 263
INSERT INTO tablename SQL command,
  360, 363
INSERT privileges, 446
INSERT SQL command, 346
installation
  on Mac OS X, 433–435
  on Windows, 428–432
int type, 281
integers, 38
invalid MySQL argument error,
  troubleshooting, 453
is_administrator() function, 394, 406
is_array conditional, 189
is_dir() function, 325
is_file() function, 325
is_numeric() function, 128, 131
is_readable() function, 315
isset() function, 128, 131

K
keyboard shortcuts
  Cut and Paste, 199
  Edit menu, 199
ksort() functions, using with arrays, 178–180

L
language constructs, 150
languages. See multilingual web pages
Laravel PHP framework, 455
leap year, formatting, 458
legacy file writing, 305
less than (<) operator, 135, 457
less than or equal to (<=) operator, 135, 457
linking strings, 100
links
  using to pass values, 68–69
  using with multiple values, 72
list() function
  using with array elements, 189
  using with functions, 288
list_dir.php document, creating,
  326–329
list_dir.php script, 325
list.html document, creating, 183
list.php document, creating,
  184–185
local variables, 97, 290
locking files, 310–312
$logged in variable, 339, 341
logical operators, 135, 138–143, 457
login form, displaying, 218–219
login page
  HTTP headers added to, 240
  purpose of, 216–217
login.php document
  creating, 216–219, 266–267,
    338–342, 400–404
  opening, 238, 262
loops
  nesting, 156
  troubleshooting, 454
ltrim() function, 119

J
JavaScript, 105, 456
join() function, 185
jQuery website, 456
Mac OS X
- Get Info panel, 302
- installation on, 433–435
- installing XAMPP on, 434–435

Magic Quotes, 62

mail() function, 228–230, 232, 437
make_date_menus() function, 274
make_text_input() function, 279, 295
MAMP website, 433
MariaDB, installation by XAMPP, 429
math. See arithmetic
memory allocation, error related to, 65
menus.php document, creating, 272–274
merging arrays, 169
messages, printing, 16
meta tags, using for encoding, 5
method attribute, using with forms, 54–57
microseconds, formatting with date() function, 211
microseconds parameters, formatting with date() function, 458
minutes, formatting with date() function, 211
modulus (%) operator, 135, 457
money_format() function, using with numbers, 85
month pull-down menu, creating, 272
month values, formatting, 458
monthly payment, calculating, 81
months, formatting with date() function, 211
move_uploaded_file() function, 317, 319–320
Mozilla Developer Network website, 456
mtrand() function, using, 90–91
multidimensional arrays, creating, 40, 173–177. See also arrays
multilingual forums, 96
multilingual web pages, creating, 5
multiplication (*) operator, 79, 135, 457
myblog database, 349
myquotes database, 390
MySQL client
- debugging PHP scripts, 440
- using, 438–440
- using semicolon (;) in, 438
  on Windows, 440
MySQL database management system (DBMS), 345
MySQL databases. See also databases; tables
- apostrophes (‘) in form data, 370
- connecting to, 348–351
- creating, 445, 447
- creating tables, 355–359
- error handling, 352–354
- inserting records into, 365
  localhost value, 351
- myblog, 349
- queries and query results, 347
- sending SQL statements to, 346
- support in PHP, 346
- username and password values, 349
MySQL users
- creating, 445–448
- privileges, 445–448
- root user password, 443–445
mysqli_affected_rows() function, 380, 387
mysqli_connect.php document
  creating, 348–350, 392
  opening, 353
mysqli_errno() function, 352–354
mysqli_fetch_array() function, 371–372, 375
mysqli_num_rows() function, 375, 387
mysqli_query() function, 346, 357, 371, 379
mysqli_real_escape_string() function, 367–370, 383, 385, 387

N
name value, using to print greetings, 70–72
$name variable, creating via concatenation, 99
names, concatenating, 100
natsort() functions, using with strings, 181
navigating
  directories, 325–329
  files, 203
negation (!) logical operator, 135, 457
nesting conditionals, 139, 217–218. See also conditionals
nesting loops, 156
newlines (\n)
  converting to breaks, 101–103, 107
  using, 22
  using with cookies, 22, 252
Nginx, 10
nl2br() function
looking up, 19
using concatenation with, 100
using with newlines, 102
nobody permission, 302
Not Found response, receiving, 14
Notice error level, 65
NULL, using with functions, 284
null coalescing (?=) logical operator, 135, 143
null coalescing (//=) logical operator, 457
number_format() function, using, 83–85
numbers. See also random numbers
creating HTML form for, 76–78
formatting, 83–85
incrementing and decrementing, 88–89
types of, 38
valid and invalid, 38
numeric indexes
setting, 165
using for loop with, 172

O
ob_clean() function, 234, 236
ob_end_flush() function, 234–237
ob_flush() function, 236
ob_get_contents() function, 236
ob_get_length() function, 236
ob_start() function, invoking, 233–234
octal format, 302
$okay variable, using with control structures, 126–127, 129–130
OOP (object-oriented programming), 455
The Open Web Application Security Project
website, 455
operator precedence table, 457
operators
for arithmetic, 79
table, 457
or (| |) logical operator, 135, 139, 457
OR logical operator, 135, 139, 457
ORDER BY RAND() clause, 424
ordinal suffix, 458
organizing files and folders, 204
others permission, 301
output buffering, 233–236, 250
owner of file, explained, 298

P
pages. See HTML pages
parameters, defining functions with, 276–277
parent folder (..), 303
parentheses (()), 172
using in calculations, 86–87
using with conditionals, 143
Parse error level, 65
parse errors. See also error messages;
troubleshooting
avoiding, 170
double quotation marks ("), 58
receiving, 43
troubleshooting, 454
password values, validating, 136–137
password_hash() function, 112, 337
password_verify() function, 337
passwords
encrypting, 337
entering in HTML form, 123
managing, 124
validating, 130, 224
permissions, 298–302, 309, 352
PHP
configuring, 436–437
configuring for file uploads, 318–319
PHP code, storing, 236
.php extension, 9
PHP functions, using with HTML tags, 104–107. See also functions
PHP manual, using, 18–20, 449
PHP scripts
accessing, 14
adding comments to, 24–26
creating, 8, 70–71
debugging, 28, 440
executing, 9
versus HTML pages, 7
requesting, 215
running through URLs, 451
testing, 12–14
testing in browsers, 12–14
<?php tag, 8
PHP version, verifying, 451
phpinfo() function, 8–9, 436
$\textit{phpinfo.php} \text{ document, creating, 8–9}
$\textit{php.ini} \text{ file}
  editing, 437
  saving, 436
  session settings, 263
$\textit{phpMyAdmin, using}, 347, 441–442
$\textit{PhpStorm}, 4
pipe (|), explained, 67
$\textit{$_POST} \text{ and } $\textit{$_GET}$, 58–62, 68
$\textit{POST} \text{ and } $\textit{GET}$, using with \textit{method} attribute, 54–57
$\textit{$_POST} \text{ array}, 161
$\textit{$_POST} \text{ elements, using with cost}
  calculator, 80
$\textit{postfix mail server}, 437
$\textit{posting.html} \text{ document, creating, 94–96}
preference
  managing, 86–87
  table, 457
predefined constants, 210. \textit{See also}
  constants
predefined variables, printing, 33–35. \textit{See also}
  variables
$\textit{predefined.php} \text{ document, creating, 33}
preset HTML form values cut off error,
  troubleshooting, 454
primary keys, 365, 387
$\textit{print} \text{ language construct, using, 15–16, 21,}
  32–33
$\textit{print} \text{ statement}
  control variables, 129–130
  forms, 61
  HTML form tags, 274
  \textit{str_ireplace()} \text{ and } \textit{trim()}, 118–119
  substrings, 115–116
  \textit{urlencode()} \text{ function, 109–111
  variables, 41
$\textit{printf()} \text{ function, using with numbers, 85
printing}
  $ (dollar sign), 82
  arrays, 164
  constants, 209–210
  greetings, 70–71
  multidimensional arrays, 176
  predefined variables, 33–35
  results from cost calculator, 81–82
  values of arrays, 171–172
  values of constants, 207
printing messages, 16
$\textit{problem} \text{ variable}
  creating, 222
  using, 224
  using with databases, 362, 364
$\textit{PROCESS} \text{ privileges, 446
Project Euler, 455
pull-down menus}
  creating, 272–274
  preselecting, 227
$\textit{Q}$
query data, securing, 366–770. \textit{See also}
  databases
quotation marks ("). \textit{See double quotation}
  marks ("), single quotation marks (’)
quotes
  adding, 304, 405–408
  deleting, 418–421
  editing, 412–417
  listing, 409–411
  storing in text file, 306–307
$\textit{quotes.php} \text{ script, creating, 45
quotes.txt} \text{ file}
  creating, 300
  opening, 300
\textbf{R}
radio buttons, presetting status of, 227
$\textit{RAND()} \text{ function, 424
$\textit{rand()} \text{ function, using, 90–91
random numbers, 90–91. \textit{See also}
  numbers
$\textit{random.php} \text{ document, creating, 90–91
read permission, 298, 301–302
$\textit{readfile()} \text{ function, 315
reading}
  from files, 313–315
  files incrementally, 338–342
$\textit{register.html} \text{ directory, 122, 130
$\textit{register.html} \text{ document}
  creating, 122–124
  opening, 153
$\textit{register.php} \text{ script, 331–337
creating, 331–337
opening, 229
registration form
  error message in, 231
  making sticky, 220–227
registration page, creating, 122–124
registration script, creating for directory, 331–332
relative paths, 203, 303
RELOAD privileges, 446
require() function
  failure of, 201
  and parentheses (()), 206
  using with constants, 207
required attribute, using with forms, 50, 52
reset.php script, creating, 258–259
resources, books, 456
$result reference, using with databases, 371–372
return statement
  using with functions, 285
  and variable scope, 290
REVOKE privileges, 446, 448
rmdir() function, 337
round() function, using with numbers, 83, 85
rsort() functions, using with arrays, 178, 180
rtrim() function, 119

S
safe mode, running PHP in, 309
sales cost calculator, creating, 79–82
Save As feature, 6
saving documents and scripts, 6, 132, 417.
  See also files
    add_entry.php, 365
    add_quote.php, 309, 408
    books.php, 176
    calculator1.php, 289, 294
    calculator.html, 78, 85, 87–88
    create_table.php, 359
    delete_entry.php, 381
    edit_entry.php, 386
    event.php, 190
    feedback.html, 53
    footer.html, 213, 399
    functions.php, 395
    handle_calc.php, 81
    handle_form.php, 61, 64, 67
    handle_post.php, 99
    handle_reg.php, 127, 130
    header.html, 199, 209, 397
    hello1.php, 17
    hello2.php, 22
    hello3.php, 26
    hello.html, 70
    hello.php, 71
    index.php, 204
    list_dir.php, 329
    list.html, 183
    list.php, 185
    login.php, 262, 342, 403
    logout.php, 267, 404
    menus.php, 274
    mysqli_connect.php, 350, 392
    phpinfo.php, 9
    php.ini, 436
    posting.html, 96, 103, 107
    predefined.php, 34
    quotes.php, 47
    random.php, 91
    register.html, 138
    register.php, 155, 337
    reset.php, 259
    soups1.php, 164
    soups2.php, 168
    sticky2.php, 284
    template.html, 197
    upload_file.php, 324
    users.txt, 330
    variables.php, 43
    view_entries.php, 375
    view_quote.php, 314–315
    view_quotes.php, 417
    welcome.html file, 6
    welcome.php, 240, 265
scalar variable types
  versus arrays, 160
  using print with, 41
scripts. See PHP scripts
seconds, formatting, 211, 458
securing query data, 366–370
security, resources, 455
security issues, related to cookies, 245
SELECT privileges, 446
SELECT query, 371, 373–374
SELECT SQL command, 346
semicolon (;)
  error related to, 65
  using in MySQL client, 438
  using with print command, 15
sending email, 228–232
sendmail server, 437
$_SERVER variable, 32–35, 58
servers. See also web server applications
  configuring to send email, 232
  setting time zones for, 213
  using SFTP with, 10–11
$_SESSION array, 261, 264
session data
  destroying, 267
  storing, 265
Session ID (SID) constant, 263
session variables, accessing, 264–265
session_name() function, 263
session_set_cookie_params() function, 263
session_start() function, 233, 264, 266
sessions
  comparing to cookies, 260–261
  creating, 261–263
  deleting, 266–267
  explained, 260
  security issues, 265
  storing values in, 263
  verifying variables, 265
setcookie() function, 233, 246–250, 257–259
SFTP (Secure File Transfer Protocol), using, 10–11
sha1() function, 335, 337
short array syntax, using, 162
short tags, 9
shuffle() function, using with arrays, 178
SHUTDOWN privileges, 446
single quotation marks (‘)
  versus double quotation marks (“), 169
  using, 44
site structure, 203. See also website project
$size variable, using with default values, 283
sizeof() function, using with arrays, 169
Slim microframework website, 456
SMTP servers, 437
sorting arrays, 178–181
sort.php document, creating, 179–181
$soups array, 163
soups1.php document
  creating, 163–164
  opening, 167
soups2.php document, creating, 167
soups3.php document, creating, 171–172
spaceship (<=>) operator, 135, 138, 457
spacing of HTML code, displaying, 22
sprintf() function, using with numbers, 85
SQL (Structured Query Language), features of, 346–347
SQL commands, 346
SQL Course website, 450
SQL statements, sending to MySQL, 346
SQL.org website, 450
square brackets ([]), using with keys in arrays, 161
sticky forms, 220–221. See also form data;
  HTML forms
sticky text inputs, creating, 278–281
sticky1.php document
  creating, 278–281
  opening, 283
str_ireplace(), using with trim(), 118–119
string case, adjusting, 117
string type, 281
strings. See also empty string (‘ ’);
  substrings
    checking formats of, 116
    comparing, 113
    concatenating, 97–100
    counting backward in, 114
    creating HTML form for, 94–96
    encoding and decoding, 108–112
    encrypting and decrypting, 112
    performing case-sensitive searches, 117
    replacing parts of, 117–119
    transforming between arrays, 182–185
    using, 39
strip_tags() function, 104, 107
stripslashes() function, using with Magic Quotes, 62
strtok() function, 113
strtolower() function, 218
A Study in Scarlett website, 455
trim() function
- using in comparisons, 138
- using with strings, 117–119

troubleshooting. See also error messages; parse errors
- access denied, 453
- advice, 451
- blank pages, 452
- calls to undefined functions, 452
- conditionals and loops, 454
- headers already sent, 453
- invalid MySQL argument, 453
- parse errors, 454
- preset HTML form values cut off, 454
- undefined variable and index errors, 452
- variables without values, 452
- TRUE and FALSE, 121, 395
- true value, 19. See also Boolean TRUE and FALSE

TRUNCATE TABLE tablename query, 381

T
- tab (\t), using with cookies, 252
- tables. See also MySQL databases
  - creating, 355–360
  - primary keys, 355
  - using primary keys in, 387
- tags. See HTML tags
- tax rate, calculating, 81
- $tax variable, 293
- template.html document
  - creating, 195–197
  - opening, 198, 200
- templates. See also CSS templates
  - creating, 194
  - footer file, 200
  - header file, 198–199
  - layout model, 195–197
  - website project, 396–399
- testing
  - PHP scripts, 12–14
  - safety of sending cookies, 250
- text, sending to browsers, 15–17
- text area, presetting value of, 227
- text file, creating for file permissions, 299–300
- text input type, checking, 138
- textarea form element
  - adding to forms, 53
  - using with newlines, 101–102
- time() and date() functions
  - table, 458
  - using, 211–213, 254
  - using with sessions, 265
- time zones
  - formatting with date() function, 211–212, 458
  - setting for servers, 213
- tokens, substrings as, 113
- Transmit FTP application, 301

U
- uasort() functions, using with arrays, 181
- undefined functions, troubleshooting calls to, 452. See also functions
  - Undefined index notice, 170
  - Undefined offset notice, 170
  - Undefined variable error, 43
- underscore (_)
  - forms, 51
  - functions, 270
  - variables, 37
- unlink() function, 324
- UPDATE privileges, 446
- UPDATE SQL command, 346, 382–387
- upload_file.php document, creating, 319–324
- uploaded files, renaming, 324
- uploads folder, creating, 318
- urlencode() function, 108–112
- “user,” defining, 299
- user-defined functions, 270–271, 275, 278, 288, 393–395. See also functions
  - username, using on registration pages, 124
Index

**users** folder, creating, 330
users.txt script, 339–340
UTF-8 encoding, 5–6

**V**
validating, passwords, 224
validation functions, 128–131
value types, declaring, 289
values, assigning to variables, 135
$var variable, 291
var_dump() function, using with
  arrays, 165
variable errors, troubleshooting, 452
variable names, case sensitivity of, 36
variable scope
  explained, 290–292
global statement, 293–294
variables. See also predefined variables
accessing, 42–43
arrays, 40
assigning values to, 135
avoiding referring to, 131
documenting purpose of, 37
equation related to, 65
explained, 32
incrementing values of, 88–89
minimizing bugs, 37
naming conventions, 37
numbers, 38
referring to, 37
scalar and nonscalar, 41
strings, 39
syntax, 36–37
types of, 38–40
valid and invalid, 37
validating, 147
values, 41–43
warnings related to, 37
variables.php document, creating, 42–43
version control software, 11
view_blog.php document, 386–387
view_entries.php document, creating, 372–375
view_quotes.php document, creating, 314–315, 409–411
view_settings.php document, creating, 251–253

**W**
W3Schools
  JavaScript pages, 456
SQL Tutorial website, 450
WAMP website, 428
Warning error level
web pages. See HTML pages
web root directory, 298. See also directories
web server applications, 10. See also servers
website project. See also site structure
adding quotes, 405–408
administrator, 390
creating home page, 422–425
database connection, 392
deleting quotes, 418–421
denying access, 405
editing quotes, 412–417
file organization and structure, 391
footer.html document, 398
identifying goals of, 390
listing quotes, 409–411
logging in, 400–403
logging out, 404
myquotes database, 390
security, 390
template, 396–399
user-defined function, 393–395
websites
Adobe Dreamweaver, 4
AMPPS, 428, 433
Apache, 10
Aptana Studio, 4
Atom, 4
Bitnami, 428, 433
Git version control software, 11
IIS (Internet Information Server), 10
JavaScript resources, 456
MAMP, 433
Nginx, 10
PHP frameworks, 455–456
PHP manual, 18
PhpStorm, 4
Project Euler, 455
security resources, 455
SQL resources, 450
Sublime Text, 4
WAMP, 428
Windows installers, 428
XAMPP, 428, 433
welcome.html file, saving, 6
welcome.php document, creating, 238–240, 264–265
while loop, 152, 156, 338
white space, using, 22
whole numbers, 38
Windows
  installation on, 428–432
  installing XAMPP on, 430–432
wordwrap() function, 107
write permission, 298, 301–302
writeable directory, creating, 318–319
writing to files, 303–309

X
XAMPP
  command prompt, 439
  files installed by, 429, 433
  and firewalls, 429
  installing on Mac OS X, 434–435
  installing on Windows, 430–432
  MariaDB, 429
  website, 428, 433
XOR (exclusive or) logical operator, 135, 139, 457
XSS (cross-site scripting) attacks, 105

Y
year pull-down menu, creating, 272
year values
  formatting, 211, 458
  validating, 136–137, 141
Yii PHP framework, 455