PHP and MySQL Web Development

Fifth Edition

New PHP 7 Coverage
The *Aqueduct of Segovia* is one of the greatest surviving monuments of Roman engineering. Built during the time of the emperor Trajan in the first century A.D., it was designed to bring water from the foothills of the Sierra de Guadarrama to the city of Segovia, Spain, some 18 kilometers away.

The structure is built with over 20,000 roughhewn granite blocks, held together entirely without cement or clamps. The 278 meter-long section of the aqueduct that winds through the center of the city has two levels of arches, forming an elegant latticework of stone 34 meters above the city’s streets.

After 2,000 years of weathering both natural and man-made calamities, this timeless example of human ingenuity is still in use today providing a supplemental water supply for the city.
“I’ve never purchased a better programming book... This book proved to be the most informative, easiest to follow, and had the best examples of any other computer-related book I have ever purchased. The text is very easy to follow!”

—Nick Landman

“This book by Welling & Thomson is the only one which I have found to be indispensable. The writing is clear and straightforward but never wastes my time. The book is extremely well laid out. The chapters are the right length and chapter titles quickly take you where you want to go.”

—Wright Sullivan, President, A&E Engineering, Inc., Greer South Carolina

“I just wanted to tell you that I think the book PHP and MySQL Web Development rocks! It’s logically structured, just the right difficulty level for me (intermediate), interesting and easy to read, and, of course, full of valuable information!”

—CodE-E, Austria

“There are several good introductory books on PHP, but Welling & Thomson is an excellent handbook for those who wish to build up complex and reliable systems. It’s obvious that the authors have a strong background in the development of professional applications and they teach not only the language itself, but also how to use it with good software engineering practices.”

—Javier Garcia, senior telecom engineer, Telefonica R&D Labs, Madrid

“I picked up this book two days ago and I am half way finished. I just can’t put it down. The layout and flow is perfect. Everything is presented in such a way so that the information is very palatable. I am able to immediately grasp all the concepts. The examples have also been wonderful. I just had to take some time out to express to you how pleased I have been with this book.”

—Jason B. Lancaster

“This book has proven a trusty companion, with an excellent crash course in PHP and superb coverage of MySQL as used for Web applications. It also features several complete applications that are great examples of how to construct modular, scalable applications with PHP. Whether you are a PHP newbie or a veteran in search of a better desk-side reference, this one is sure to please!”

—WebDynamic

“The true PHP/MySQL bible, PHP and MySQL Web Development by Luke Welling and Laura Thomson, made me realize that programming and databases are now available to the commoners. Again, I know 1/10000th of what there is to know, and already I’m enthralled.”

—Tim Luoma, TnTLuoma.com

“Welling and Thomson’s book is a good reference for those who want to get to grips with practical projects straight off the bat. It includes webmail, shopping cart, session control, and web-forum/weblog applications as a matter of course, and begins with a sturdy look at PHP first, moving to MySQL once the basics are covered.”

—twilight30 on Slashdot
“This book is absolutely excellent, to say the least.... Luke Welling and Laura Thomson give the best in-depth explanations I’ve come across on such things as regular expressions, classes and objects, sessions etc. I really feel this book filled in a lot of gaps for me with things I didn’t quite understand.... This book jumps right into the functions and features most commonly used with PHP, and from there it continues in describing real-world projects, MySQL integration, and security issues from a project manager’s point of view. I found every bit of this book to be well organized and easy to understand.”

—notepad on codewalkers.com

“A top-notch reference for programmers using PHP and MySQL. Highly recommended.”

—The Internet Writing Journal

“This book rocks! I am an experienced programmer, so I didn’t need a lot of help with PHP syntax; after all, it’s very close to C/C++. I don’t know a thing about databases, though, so when I wanted to develop a book review engine (among other projects) I wanted a solid reference to using MySQL with PHP. I have O’Reilly’s mSQL and MySQL book, and it’s probably a better pure-SQL reference, but this book has earned a place on my reference shelf...Highly recommended.”

—Paul Robichaux

“One of the best programming guides I’ve ever read.”

—jackofsometrades from Lahti, Finland

“This is a well-written book for learning how to build Internet applications with two of the most popular open-source Web development technologies.... The projects are the real jewel of the book. Not only are the projects described and constructed in a logical, component-based manner, but the selection of projects represents an excellent cross-section of common components that are built into many web sites.”

—Craig Cecil

“The book takes an easy, step-by-step approach to introduce even the clueless programmer to the language of PHP. On top of that, I often find myself referring back to it in my Web design efforts. I’m still learning new things about PHP, but this book gave me a solid foundation from which to start and continues to help me to this day.”

—Stephen Ward

“This book is one of few that really touched me and made me ‘love’ it. I can’t put it in my bookshelf; I must put it in a touchable place on my working bench as I always like to refer from it. Its structure is good, wordings are simple and straight forward, and examples are clear and step by step. Before I read it, I knew nothing of PHP and MySQL. After reading it, I have the confidence and skill to develop any complicated Web application.”

—Power Wong

“This book is God.... I highly recommend this book to anyone who wants to jump in the deep end with database driven Web application programming. I wish more computer books were organized this way.”

—Sean C Schertell
Contents at a Glance

Introduction 1

I: Using PHP
1 PHP Crash Course 11
2 Storing and Retrieving Data 53
3 Using Arrays 75
4 String Manipulation and Regular Expressions 101
5 Reusing Code and Writing Functions 131
6 Object-Oriented PHP 159
7 Error and Exception Handling 199

II: Using MySQL
8 Designing Your Web Database 209
9 Creating Your Web Database 221
10 Working with Your MySQL Database 247
11 Accessing Your MySQL Database from the Web with PHP 271
12 Advanced MySQL Administration 291
13 Advanced MySQL Programming 315

III: Web Application Security
14 Web Application Security Risks 331
15 Building a Secure Web Application 341
16 Implementing Authentication Methods with PHP 365

IV: Advanced PHP Techniques
17 Interacting with the File System and the Server 379
18 Using Network and Protocol Functions 403
19 Managing the Date and Time 423
20 Internationalization and Localization  437
21 Generating Images  449
22 Using Session Control in PHP  475
23 Integrating JavaScript and PHP  493
24 Other Useful Features  519

V: Building Practical PHP and MySQL Projects
25 Using PHP and MySQL for Large Projects  529
26 Debugging and Logging  543
27 Building User Authentication and Personalization  561
28 Building a Web-Based Email Service with Laravel Part I  Web Edition
29 Building a Web-Based Email Service with Laravel Part II  Web Edition
30 Social Media Integration Sharing and Authentication  Web Edition
31 Building a Shopping Cart  Web Edition

VI: Appendix
A Installing Apache, PHP, and MySQL  599
   Index  615
# Table of Contents

**Introduction**  
1

## I: Using PHP

1 **PHP Crash Course** 11  
   - Before You Begin: Accessing PHP 12  
   - Creating a Sample Application: Bob’s Auto Parts 12  
     - Creating the Order Form 12  
     - Processing the Form 14  
   - Embedding PHP in HTML 14  
     - PHP Tags 16  
     - PHP Statements 16  
     - Whitespace 17  
     - Comments 17  
   - Adding Dynamic Content 18  
     - Calling Functions 19  
     - Using the `date()` Function 19  
   - Accessing Form Variables 20  
     - Form Variables 20  
     - String Concatenation 22  
     - Variables and Literals 23  
   - Understanding Identifiers 23  
   - Examining Variable Types 24  
     - PHP’s Data Types 24  
     - Type Strength 25  
     - Type Casting 25  
     - Variable Variables 25  
   - Declaring and Using Constants 26  
   - Understanding Variable Scope 27  
   - Using Operators 28  
     - Arithmetic Operators 28  
     - String Operators 29  
     - Assignment Operators 29  
     - Comparison Operators 31  
     - Logical Operators 32
3 Using Arrays 75

What Is an Array? 75

Numerically Indexed Arrays 76
  Initializing Numerically Indexed Arrays 76
  Accessing Array Contents 77
  Using Loops to Access the Array 78

Arrays with Different Indices 79
  Initializing an Array 79
  Accessing the Array Elements 79
  Using Loops 79

Array Operators 81

Multidimensional Arrays 82

Sorting Arrays 85
  Using sort() 85
  Using asort() and ksort() to Sort Arrays 86
  Sorting in Reverse 87
Introducing Regular Expressions 119
- The Basics 120
- Delimiters 120
- Character Classes and Types 120
- Repetition 122
- Subexpressions 122
- Counted Subexpressions 123
- Anchoring to the Beginning or End of a String 123
- Branching 123
- Matching Literal Special Characters 123
- Reviewing Meta Characters 124
- Escape Sequences 125
- Backreferences 126
- Assertions 126
- Putting It All Together for the Smart Form 127
Finding Substrings with Regular Expressions 128
Replacing Substrings with Regular Expressions 129
Splitting Strings with Regular Expressions 129
Further Reading 130
Next 130

5 Reusing Code and Writing Functions 131
- The Advantages of Reusing Code 131
- Cost 132
- Reliability 132
- Consistency 132
- Using require() and include() 132
  - Using require() to Include Code 133
  - Using require() for Website Templates 134
  - Using auto_prepend_file and auto_append_file 139
- Using Functions in PHP 140
  - Calling Functions 141
  - Calling an Undefined Function 142
  - Understanding Case and Function Names 143
- Defining Your Own Functions 144
- Examining Basic Function Structure 144
  - Naming Your Function 145
- Using Parameters 146
Retrieving Data from the Database 250
  Retrieving Data with Specific Criteria 251
  Retrieving Data from Multiple Tables 253
  Retrieving Data in a Particular Order 259
  Grouping and Aggregating Data 259
  Choosing Which Rows to Return 261
  Using Subqueries 262
Updating Records in the Database 265
Altering Tables After Creation 265
Deleting Records from the Database 268
Dropping Tables 268
Dropping a Whole Database 268
Further Reading 269
Next 269

11 Accessing Your MySQL Database from the Web with PHP 271
How Web Database Architectures Work 272
Querying a Database from the Web 275
  Checking and Filtering Input Data 276
  Setting Up a Connection 277
  Choosing a Database to Use 278
  Querying the Database 278
  Using Prepared Statements 279
  Retrieving the Query Results 280
  Disconnecting from the Database 281
Putting New Information in the Database 282
Using Other PHP-Database Interfaces 286
  Using a Generic Database Interface: PDO 286
Further Reading 289
Next 289

12 Advanced MySQL Administration 291
Understanding the Privilege System in Detail 291
  The user Table 293
  The db Table 295
  The tables_priv, columns_priv, and procs_priv Tables 296
Access Control: How MySQL Uses the Grant Tables 298
Updating Privileges: When Do Changes Take Effect? 299
III: Web Application Security

14 Web Application Security Risks 331
   Identifying the Threats We Face 331
      Access to Sensitive Data 331
      Modification of Data 334
      Loss or Destruction of Data 334
      Denial of Service 335
      Malicious Code Injection 337
      Compromised Server 338
      Repudiation 338
   Understanding Who We're Dealing With 339
      Attackers and Crackers 339
      Unwitting Users of Infected Machines 339
      Disgruntled Employees 339
      Hardware Thieves 340
      Ourselves 340
Next 340

15 Building a Secure Web Application 341
   Strategies for Dealing with Security 341
      Start with the Right Mindset 342
      Balancing Security and Usability 342
      Monitoring Security 342
      Our Basic Approach 343
   Securing Your Code 343
      Filtering User Input 343
      Escaping Output 348
      Code Organization 350
      What Goes in Your Code 351
      File System Considerations 352
      Code Stability and Bugs 352
      Executing Commands 353
Contents

Securing Your Web Server and PHP 354
  Keep Software Up-to-Date 354
  Browse the php.ini file 355
  Web Server Configuration 356
  Shared Hosting of Web Applications 356
Database Server Security 357
  Users and the Permissions System 358
  Sending Data to the Server 358
  Connecting to the Server 359
  Running the Server 359
Protecting the Network 360
  Firewalls 360
  Use a DMZ 360
  Prepare for DoS and DDoS Attacks 361
Computer and Operating System Security 361
  Keep the Operating System Up to Date 361
  Run Only What Is Necessary 362
  Physically Secure the Server 362
Disaster Planning 362

16 Implementing Authentication Methods with PHP 365
  Identifying Visitors 365
  Implementing Access Control 366
    Storing Passwords 369
    Securing Passwords 369
    Protecting Multiple Pages 371
  Using Basic Authentication 372
  Using Basic Authentication in PHP 372
  Using Basic Authentication with Apache's .htaccess Files 374
Creating Your Own Custom Authentication 377
Further Reading 377

IV: Advanced PHP Techniques

17 Interacting with the File System and the Server 379
  Uploading Files 379
    HTML for File Upload 381
Calculating Dates in PHP 433
Calculating Dates in MySQL 434
Using Microseconds 435
Using the Calendar Functions 436
Further Reading 436

20 Internationalization and Localization 437
Localization Is More than Translation 437
Understanding Character Sets 438
Security Implications of Character Sets 439
Using Multibyte String Functions in PHP 440
Creating a Basic Localizable Page Structure 440
Using gettext() in an Internationalized Application 444
Configuring Your System to Use gettext() 444
Creating Translation Files 445
Implementing Localized Content in PHP Using gettext() 447
Further Reading 448

21 Generating Images 449
Setting Up Image Support in PHP 449
Understanding Image Formats 450
JPEG 450
PNG 450
GIF 451
Creating Images 451
Creating a Canvas Image 452
Drawing or Printing Text on the Image 453
Outputting the Final Graphic 455
Cleaning Up 455
Using Automatically Generated Images in Other Pages 456
Using Text and Fonts to Create Images 457
Setting Up the Base Canvas 460
Fitting the Text onto the Button 461
Positioning the Text 464
Writing the Text onto the Button 464
Finishing Up 465
22 Using Session Control in PHP 475
What Is Session Control? 475
Understanding Basic Session Functionality 476
  What Is a Cookie? 476
  Setting Cookies from PHP 476
  Using Cookies with Sessions 477
  Storing the Session ID 477
Implementing Simple Sessions 478
  Starting a Session 478
  Registering Session Variables 478
  Using Session Variables 479
  Unsetting Variables and Destroying the Session 479
Creating a Simple Session Example 480
Configuring Session Control 482
Implementing Authentication with Session Control 483
Next 491

23 Integrating JavaScript and PHP 493
Understanding AJAX 493
A Brief Introduction to jQuery 494
Using jQuery in Web Applications 494
Using jQuery and AJAX with PHP 504
  The AJAX-Enabled Chat Script/Server 504
  The jQuery AJAX Methods 507
  The Chat Client/jQuery Application 510
Further Reading 517
Next 517

24 Other Useful Features 519
Evaluating Strings: eval() 519
Terminating Execution: die() and exit() 520
Serializing Variables and Objects 521
Getting Information About the PHP Environment 522
  Finding Out What Extensions Are Loaded 522
Identifying the Script Owner 523
Finding Out When the Script Was Modified 523
Temporarily Altering the Runtime Environment 524
Highlighting Source Code 525
Using PHP on the Command Line 526
Next 527

V: Building Practical PHP and MySQL Projects

25 Using PHP and MySQL for Large Projects 529
Applying Software Engineering to Web Development 530
Planning and Running a Web Application Project 530
Reusing Code 531
Writing Maintainable Code 532
  Coding Standards 532
  Breaking Up Code 535
  Using a Standard Directory Structure 536
  Documenting and Sharing In-House Functions 536
Implementing Version Control 536
Choosing a Development Environment 537
Documenting Your Projects 538
Prototyping 538
Separating Logic and Content 539
Optimizing Code 540
  Using Simple Optimizations 540
Testing 541
Further Reading 542
Next 542

26 Debugging and Logging 543
Programming Errors 543
  Syntax Errors 543
  Runtime Errors 544
  Logic Errors 549
Variable Debugging Aid 551
Error Reporting Levels 553
Altering the Error Reporting Settings 554
Triggering Your Own Errors 556
Contents

Logging Errors Gracefully 557
Logging Errors to a Log File 560
Next 560

27 Building User Authentication and Personalization 561
Solution Components 561
  User Identification and Personalization 562
  Storing Bookmarks 563
  Recommending Bookmarks 563
Solution Overview 563
Implementing the Database 565
Implementing the Basic Site 566
Implementing User Authentication 569
  Registering Users 569
  Logging In 575
  Logging Out 579
  Changing Passwords 580
  Resetting Forgotten Passwords 582
Implementing Bookmark Storage and Retrieval 587
  Adding Bookmarks 588
  Displaying Bookmarks 590
  Deleting Bookmarks 591
Implementing Recommendations 594
Considering Possible Extensions 598

28 Building a Web-Based Email Service with Laravel Part I Web Edition

29 Building a Web-Based Email Service with Laravel Part II Web Edition

30 Social Media Integration Sharing and Authentication Web Edition

31 Building a Shopping Cart Web Edition

VI: Appendix

A Installing Apache, PHP, and MySQL 599
  Installing Apache, PHP, and MySQL Under UNIX 600
    Binary Installation 600
    Source Installation 601
    Basic Apache Configuration Modifications 608
Lead Authors

**Laura Thomson** is Director of Engineering at Mozilla Corporation. She was formerly a principal at both OmniTI and Tangled Web Design, and she has worked for RMIT University and the Boston Consulting Group. She holds a Bachelor of Applied Science (Computer Science) degree and a Bachelor of Engineering (Computer Systems Engineering) degree with honors. In her spare time she enjoys riding horses, arguing about free and open source software, and sleeping.

**Luke Welling** is a software engineer and regularly speaks on open source and web development topics at conferences such as OSCON, ZendCon, MySQLUC, PHPCon, OSDC, and LinuxTag. He has worked for OmniTI, for the web analytics company Hitwise.com, at the database vendor MySQL AB, and as an independent consultant at Tangled Web Design. He has taught computer science at RMIT University in Melbourne, Australia, and holds a Bachelor of Applied Science (Computer Science) degree. In his spare time, he attempts to perfect his insomnia.

Contributing Authors

**Julie C. Meloni** is a software development manager and technical consultant living in Washington, D.C. She has written several books and articles on web-based programming languages and database topics, including the bestselling *Sams Teach Yourself PHP, MySQL and Apache All in One*.

**John Coggeshall** is the owner of Internet Technology Solutions, LLC—an Internet and PHP consultancy serving customers worldwide, as well as the owner of CoogleNet, a subscription based WiFi network. As former senior member of Zend Technologies’ Global Services team, he got started with PHP in 1997 and is the author of four published books and over 100 articles on PHP technologies.

**Jennifer Kyrnin** is an author and web designer who has been working on the Internet since 1995. Her other books include *Sams Teach Yourself Bootstrap in 24 Hours, Sams Teach Yourself Responsive Web Design in 24 Hours*, and *Sams Teach Yourself HTML5 Mobile Application Development in 24 Hours*. 
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

You can email or write directly to let us know what you did or didn’t like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book’s title and author, as well as your name and phone or email address.

Email: feedback@developers-library.info

Mail: Reader Feedback
Addison-Wesley Developer's Library
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
Accessing the Free Web Edition

Your purchase of this book in any format, print or electronic, includes access to the corresponding Web Edition, which provides several special features to help you learn:

- The complete text of the book online
- Interactive quizzes and exercises to test your understanding of the material
- Bonus chapters not included in the print or e-book editions
- Updates and corrections as they become available

The Web Edition can be viewed on all types of computers and mobile devices with any modern web browser that supports HTML5.

To get access to the Web Edition of *PHP and MySQL Web Development, Fifth Edition*, all you need to do is register this book:

1. Go to www.informit.com/register
2. Sign in or create a new account
3. Enter ISBN: 9780321833891
4. Answer the questions as proof of purchase

The Web Edition will appear under the Digital Purchases tab on your Account page. Click the Launch link to access the product.
This page intentionally left blank
Welcome to *PHP and MySQL Web Development*. Within its pages, you will find distilled knowledge from our experiences using PHP and MySQL, two of the most important and widely used web development tools around.

Key topics covered in this introduction include

- Why you should read this book
- What you will be able to achieve using this book
- What PHP and MySQL are and why they’re great
- What’s changed in the latest versions of PHP and MySQL
- How this book is organized

Let’s get started.

**Note**

Visit our website and register this book at [informit.com/register](http://informit.com/register) for convenient access to any updates, downloads, or errata that might be available for this book.

## Why You Should Read This Book

This book will teach you how to create interactive web applications from the simplest order form through to complex, secure web applications. What’s more, you’ll learn how to do it using open-source technologies.

This book is aimed at readers who already know at least the basics of HTML and have done some programming in a modern programming language before but have not necessarily programmed for the web or used a relational database. If you are a beginning programmer, you should still find this book useful, but digesting it might take a little longer. We’ve tried not to leave out any basic concepts, but we do cover them at speed. The typical readers of this book want to master PHP and MySQL for the purpose of building a large or commercial website. You might already be working in another web development language; if so, this book should get you up to speed quickly.
We wrote the first edition of this book because we were tired of finding PHP books that were basically function references. These books are useful, but they don’t help when your boss or client has said, “Go build me a shopping cart.” In this book, we have done our best to make every example useful. You can use many of the code samples directly in your website, and you can use many others with only minor modifications.

What You Will Learn from This Book

Reading this book will enable you to build real-world, dynamic web applications. If you've built websites using plain HTML, you realize the limitations of this approach. Static content from a pure HTML website is just that—static. It stays the same unless you physically update it. Your users can’t interact with the site in any meaningful fashion.

Using a language such as PHP and a database such as MySQL allows you to make your sites dynamic: to have them be customizable and contain real-time information.

We have deliberately focused this book on real-world applications, even in the introductory chapters. We begin by looking at simple systems and work our way through the various parts of PHP and MySQL.

We then discuss aspects of security and authentication as they relate to building a real-world website and show you how to implement these aspects in PHP and MySQL. We also introduce you to integrating front-end and back-end technologies by discussing JavaScript and the role it can play in your application development.

In the final part of this book, we describe how to approach real-world projects and take you through the design, planning, and building of the following projects:

- User authentication and personalization
- Web-based email
- Social media integration

You should be able to use any of these projects as is, or you can modify them to suit your needs. We chose them because we believe they represent some the most common web applications built by programmers. If your needs are different, this book should help you along the way to achieving your goals.

What Is PHP?

PHP is a server-side scripting language designed specifically for the web. Within an HTML page, you can embed PHP code that will be executed each time the page is visited. Your PHP code is interpreted at the web server and generates HTML or other output that the visitor will see.

PHP was conceived in 1994 and was originally the work of one man, Rasmus Lerdorf. It was adopted by other talented people and has gone through several major rewrites to bring us the
Introduction

broad, mature product we see today. According to Google’s Greg Michillie in May 2013, PHP ran more than three quarters of the world’s websites, and that number had grown to over 82% by July 2016.

PHP is an open-source project, which means you have access to the source code and have the freedom to use, alter, and redistribute it.

PHP originally stood for Personal Home Page but was changed in line with the GNU recursive naming convention (GNU = Gnu’s Not Unix) and now stands for PHP Hypertext Preprocessor.

The current major version of PHP is 7. This version saw a complete rewrite of the underlying Zend engine and some major improvements to the language. All of the code in this book has been tested and validated against the most recent release of PHP 7 at the time of writing, as well as the latest version in the PHP 5.6 family of releases, which is still officially supported.

The home page for PHP is available at http://www.php.net.

The home page for Zend Technologies is http://www.zend.com.

What Is MySQL?

MySQL (pronounced My-Ess-Que-Ell) is a very fast, robust, relational database management system (RDBMS). A database enables you to efficiently store, search, sort, and retrieve data. The MySQL server controls access to your data to ensure that multiple users can work with it concurrently, to provide fast access to it, and to ensure that only authorized users can obtain access. Hence, MySQL is a multiuser, multithreaded server. It uses Structured Query Language (SQL), the standard database query language. MySQL has been publicly available since 1996 but has a development history going back to 1979. It is the world’s most popular open-source database and has won the Linux Journal Readers’ Choice Award on a number of occasions.

MySQL is available under a dual licensing scheme. You can use it under an open-source license (the GPL) free as long as you are willing to meet the terms of that license. If you want to distribute a non-GPL application including MySQL, you can buy a commercial license instead.

Why Use PHP and MySQL?

When setting out to build a website, you could use many different products.

You need to choose the following:

- Where to run your web servers: the cloud, virtual private servers, or actual hardware
- An operating system
- Web server software
- A database management system or other datastore
- A programming or scripting language
You may end up with a hybrid architecture with multiple datastores. Some of these choices are dependent on the others. For example, not all operating systems run on all hardware, not all web servers support all programming languages, and so on.

In this book, we do not pay much attention to hardware, operating systems, or web server software. We don’t need to. One of the best features of both PHP and MySQL is that they work with any major operating system and many of the minor ones.

The majority of PHP code can be written to be portable between operating systems and web servers. There are some PHP functions that specifically relate to the filesystem that are operating system dependent, but these are clearly marked as such in the manual and in this book.

Whatever hardware, operating system, and web server you choose, we believe you should seriously consider using PHP and MySQL.

Some of PHP’s Strengths

Some of PHP’s main competitors are Python, Ruby (on Rails or otherwise), Node.js, Perl, Microsoft .NET, and Java.

In comparison to these products, PHP has many strengths, including the following:

- Performance
- Scalability
- Interfaces to many different database systems
- Built-in libraries for many common web tasks
- Low cost
- Ease of learning and use
- Strong object-oriented support
- Portability
- Flexibility of development approach
- Availability of source code
- Availability of support and documentation

A more detailed discussion of these strengths follows.

Performance

PHP is very fast. Using a single inexpensive server, you can serve millions of hits per day. It scales down to the smallest email form and up to sites such as Facebook and Etsy.
Scalability
PHP has what Rasmus Lerdorf frequently refers to as a “shared-nothing” architecture. This means that you can effectively and cheaply implement horizontal scaling with large numbers of commodity servers.

Database Integration
PHP has native connections available to many database systems. In addition to MySQL, you can directly connect to PostgreSQL, Oracle, MongoDB, and MSSQL, among others. PHP 5 and PHP 7 also have a built-in SQL interface to flat files, called SQLite.

Using the Open Database Connectivity (ODBC) standard, you can connect to any database that provides an ODBC driver. This includes Microsoft products and many others.

In addition to native libraries, PHP comes with a database access abstraction layer called PHP Database Objects (PDOs), which allows consistent access and promotes secure coding practices.

Built-in Libraries
Because PHP was designed for use on the Web, it has many built-in functions for performing many useful web-related tasks. You can generate images on the fly, connect to web services and other network services, parse XML, send email, work with cookies, and generate PDF documents, all with just a few lines of code.

Cost
PHP is free. You can download the latest version at any time from http://www.php.net for no charge.

Ease of Learning PHP
The syntax of PHP is based on other programming languages, primarily C and Perl. If you already know C or Perl, or a C-like language such as C++ or Java, you will be productive using PHP almost immediately.

Object-Oriented Support
PHP version 5 had well-designed object-oriented features, which continued to be refined and improved in PHP version 7. If you learned to program in Java or C++, you will find the features (and generally the syntax) that you expect, such as inheritance, private and protected attributes and methods, abstract classes and methods, interfaces, constructors, and destructors. You will even find some less common features such as iterators and traits.
Portability

PHP is available for many different operating systems. You can write PHP code on free UNIX-like operating systems such as Linux and FreeBSD, commercial UNIX versions, OS X, or on different versions of Microsoft Windows.

Well-written code will usually work without modification on a different system running PHP.

Flexibility of Development Approach

PHP allows you to implement simple tasks simply, and equally easily adapts to implementing large applications using a framework based on design patterns such as Model-View-Controller (MVC).

Source Code

You have access to PHP’s source code. With PHP, unlike commercial, closed-source products, if you want to modify something or add to the language, you are free to do so.

You do not need to wait for the manufacturer to release patches. You also don’t need to worry about the manufacturer going out of business or deciding to stop supporting a product.

Availability of Support and Documentation

Zend Technologies (http://www.zend.com), the company behind the engine that powers PHP, funds its PHP development by offering support and related software on a commercial basis.

The PHP documentation and community are mature and rich resources with a wealth of information to share.

Key Features of PHP 7

In December 2015, the long-awaited PHP 7 release was made available to the public. As mentioned in this introduction, the book covers both PHP 5.6 and PHP 7, which might lead you to ask “what happened to PHP 6?” The short answer is: there is no PHP 6 and never was for the general public. There was a development effort around a codebase that was referred to as “PHP 6” but it never came to fruition; there were many ambitious plans and subsequent complications that made it difficult for the team to continue to pursue. PHP 7 is not PHP 6 and doesn’t include the features and code from that development effort; PHP 7 is its own release with its own focus—specifically a focus on performance.

Under the hood, PHP 7 includes a refactor of the Zend Engine that powers it, which resulted in a significant performance boost to many web applications—sometimes upwards of 100%! While increased performance and decreased memory use were key to the release of PHP 7, so was backward-compatibility. In fact, relatively few backward-incompatible language changes were introduced. These are discussed contextually throughout this book so that the chapters
remain usable with PHP 5.6 or PHP 7, as widespread adoption of PHP 7 has not yet occurred by commercial web-hosting providers.

**Some of MySQL’s Strengths**

MySQL’s main competitors in the relational database space are PostgreSQL, Microsoft SQL Server, and Oracle. There is also a growing trend in the web application world toward use of NoSQL/non-relational databases such as MongoDB. Let’s take a look at why MySQL is still a good choice in many cases.

MySQL has many strengths, including the following:

- High performance
- Low cost
- Ease of configuration and learning
- Portability
- Availability of source code
- Availability of support

A more detailed discussion of these strengths follows.

**Performance**

MySQL is undeniably fast. You can see the developers’ benchmark page at http://www.mysql.com/why-mysql/benchmarks/.

**Low Cost**

MySQL is available at no cost under an open-source license or at low cost under a commercial license. You need a license if you want to redistribute MySQL as part of an application and do not want to license your application under an open-source license. If you do not intend to distribute your application—typical for most web applications—or are working on free or open-source software, you do not need to buy a license.

**Ease of Use**

Most modern databases use SQL. If you have used another RDBMS, you should have no trouble adapting to this one. MySQL is also easier to set up and tune than many similar products.

**Portability**

MySQL can be used on many different UNIX systems as well as under Microsoft Windows.
Source Code

As with PHP, you can obtain and modify the source code for MySQL. This point is not important to most users most of the time, but it provides you with excellent peace of mind, ensuring future continuity and giving you options in an emergency.

In fact, there are now several forks and drop-in replacements for MySQL that you may consider using, including MariaDB, written by the original authors of MySQL, including Michael ‘Monty’ Widenius (https://mariadb.org).

Availability of Support

Not all open-source products have a parent company offering support, training, consulting, and certification, but you can get all of these benefits from Oracle (who acquired MySQL with their acquisition of Sun Microsystems, who had previously acquired the founding company, MySQL AB).

What Is New in MySQL (5.x)?

At the time of writing, the current version of MySQL was 5.7.

Features added to MySQL in the last few releases include

- A wide range of security improvements
- FULLTEXT support for InnoDB tables
- A NoSQL-style API for InnoDB
- Partitioning support
- Improvements to replication, including row-based replication and GTIDs
- Thread pooling
- Pluggable authentication
- Multicore scalability
- Better diagnostic tools
- InnoDB as the default engine
- IPv6 support
- Plugin API
- Event scheduling
- Automated upgrades

Other changes include more ANSI standard compliance and performance improvements.
If you are still using an early 4.x version or a 3.x version of the MySQL server, you should know that the following features were added to various versions from 4.0:

- Views
- Stored procedures
- Triggers and cursors
- Subquery support
- GIS types for storing geographical data
- Improved support for internationalization
- The transaction-safe storage engine InnoDB included as standard
- The MySQL query cache, which greatly improves the speed of repetitive queries as often run by web applications

How Is This Book Organized?

This book is divided into five main parts:

Part I, “Using PHP,” provides an overview of the main parts of the PHP language with examples. Each example is a real-world example used in building an e-commerce site rather than “toy” code. We kick off this section with Chapter 1, “PHP Crash Course.” If you’ve already used PHP, you can whiz through this chapter. If you are new to PHP or new to programming, you might want to spend a little more time on it.

Part II, “Using MySQL,” discusses the concepts and design involved in using relational database systems such as MySQL, using SQL, connecting your MySQL database to the world with PHP, and advanced MySQL topics, such as security and optimization.

Part III, “Web Application Security,” covers some of the general issues involved in developing a web application using any language. We then discuss how you can use PHP and MySQL to authenticate your users and securely gather, transmit, and store data.

Part IV, “Advanced PHP Techniques,” offers detailed coverage of some of the major built-in functions in PHP. We have selected groups of functions that are likely to be useful when building a web application. You will learn about interaction with the server, interaction with the network, image generation, date and time manipulation, and session handling.

Part V, “Building Practical PHP and MySQL Projects,” is our favorite section. It deals with practical real-world issues such as managing large projects and debugging, and provides sample projects that demonstrate the power and versatility of PHP and MySQL.
Accessing the Free Web Edition

Your purchase of this book in any format includes access to the corresponding Web Edition, which provides several special features to help you learn:

- The complete text of the book online
- Interactive quizzes and exercises to test your understanding of the material
- Bonus chapters not included in the print or e-book editions
- Updates and corrections as they become available

The Web Edition can be viewed on all types of computers and mobile devices with any modern web browser that supports HTML5.

To get access to the Web Edition of *PHP and MySQL Web Development, Fifth Edition* all you need to do is register this book:

1. Go to www.informit.com/register
2. Sign in or create a new account
3. Enter ISBN: 9780321833891
4. Answer the questions as proof of purchase

The Web Edition will appear under the Digital Purchases tab on your Account page. Click the Launch link to access the product.

Finally

We hope you enjoy this book and enjoy learning about PHP and MySQL as much as we did when we first began using these products. They are really a pleasure to use. Soon, you'll be able to join the many thousands of web developers who use these robust, powerful tools to easily build dynamic, real-time web applications.
This chapter gives you a quick overview of PHP syntax and language constructs. If you are already a PHP programmer, it might fill some gaps in your knowledge. If you have a background using C, Perl, Python, or another programming language, it will help you get up to speed quickly.

In this book, you’ll learn how to use PHP by working through lots of real-world examples taken from our experiences building real websites. Often, programming textbooks teach basic syntax with very simple examples. We have chosen not to do that. We recognize that what you do is get something up and running, and understand how the language is used, instead of plowing through yet another syntax and function reference that’s no better than the online manual.

Try the examples. Type them in or download them from the website, change them, break them, and learn how to fix them again.

This chapter begins with the example of an online product order form to show how variables, operators, and expressions are used in PHP. It also covers variable types and operator precedence. You will learn how to access form variables and manipulate them by working out the total and tax on a customer order.

You will then develop the online order form example by using a PHP script to validate the input data. You’ll examine the concept of Boolean values and look at examples using if, else, the ?: operator, and the switch statement. Finally, you’ll explore looping by writing some PHP to generate repetitive HTML tables.

Key topics you learn in this chapter include

- Embedding PHP in HTML
- Adding dynamic content
- Accessing form variables
- Understanding identifiers
• Creating user-declared variables
• Examining variable types
• Assigning values to variables
• Declaring and using constants
• Understanding variable scope
• Understanding operators and precedence
• Evaluating expressions
• Using variable functions
• Making decisions with if, else, and switch
• Taking advantage of iteration using while, do, and for loops

Before You Begin: Accessing PHP

To work through the examples in this chapter and the rest of the book, you need access to a web server with PHP installed. To gain the most from the examples and case studies, you should run them and try changing them. To do this, you need a testbed where you can experiment.

If PHP is not installed on your machine, you need to begin by installing it or having your system administrator install it for you. You can find instructions for doing so in Appendix A, “Installing Apache, PHP, and MySQL.”

Creating a Sample Application: Bob’s Auto Parts

One of the most common applications of any server-side scripting language is processing HTML forms. You’ll start learning PHP by implementing an order form for Bob’s Auto Parts, a fictional spare parts company. You can find all the code for the examples used in this chapter in the directory called chapter01 on the CD-ROM.

Creating the Order Form

Bob’s HTML programmer has set up an order form for the parts that Bob sells. This relatively simple order form, shown in Figure 1.1, is similar to many you have probably seen while surfing. Bob would like to be able to know what his customers ordered, work out the total prices of their orders, and determine how much sales tax is payable on the orders.
Part of the HTML for this form is shown in Listing 1.1.

Listing 1.1  orderform.html—HTML for Bob's Basic Order Form

```html
<form action="processorder.php" method="post">
<table style="border: 0px;">
<tr style="background: #cccccc;">
<td style="width: 150px; text-align: center;">Item</td>
<td style="width: 15px; text-align: center;">Quantity</td>
</tr>
<tr>
<td>Tires</td>
<td><input type="text" name="tireqty" size="3" maxlength="3" /></td>
</tr>
<tr>
<td>Oil</td>
<td><input type="text" name="oilqty" size="3" maxlength="3" /></td>
</tr>
<tr>
<td>Spark Plugs</td>
<td><input type="text" name="sparkqty" size="3" maxlength="3" /></td>
</tr>
</table>
</form>
```
Notice that the form’s action is set to the name of the PHP script that will process the customer’s order. (You’ll write this script next.) In general, the value of the action attribute is the URL that will be loaded when the user clicks the Submit button. The data the user has typed in the form will be sent to this URL via the HTTP method specified in the method attribute, either get (appended to the end of the URL) or post (sent as a separate message).

Also note the names of the form fields: tireqty, oilqty, and sparkqty. You’ll use these names again in the PHP script. Because the names will be reused, it’s important to give your form fields meaningful names that you can easily remember when you begin writing the PHP script. Some HTML editors generate field names like field23 by default. They are difficult to remember. Your life as a PHP programmer will be easier if the names you use reflect the data typed into the field.

You should consider adopting a coding standard for field names so that all field names throughout your site use the same format. This way, you can more easily remember whether, for example, you abbreviated a word in a field name or put in underscores as spaces.

**Processing the Form**

To process the form, you need to create the script mentioned in the action attribute of the form tag called processorder.php. Open your text editor and create this file. Then type in the following code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Bob's Auto Parts - Order Results</title>
</head>
<body>
    <h1>Bob's Auto Parts</h1>
    <h2>Order Results</h2>
</body>
</html>
```

Notice how everything you've typed so far is just plain HTML. It’s now time to add some simple PHP code to the script.

**Embedding PHP in HTML**

Under the <h2> heading in your file, add the following lines:

```php
<?php
    echo '<p>Order processed.</p>';?
```

Save the file and load it in your browser by filling out Bob’s form and clicking the Submit Order button. You should see something similar to the output shown in Figure 1.2.

![Figure 1.2](image-url)

Notice how the PHP code you wrote was embedded inside a normal-looking HTML file. Try viewing the source from your browser. You should see this code:

```html
<html>
<head>
  <title>Bob's Auto Parts - Order Results</title>
</head>
<body>
  <h1>Bob's Auto Parts</h1>
  <h2>Order Results</h2>
  <p>Order processed.</p>
</body>
</html>
```

None of the raw PHP is visible because the PHP interpreter has run through the script and replaced it with the output from the script. This means that from PHP you can produce clean HTML viewable with any browser; in other words, the user’s browser does not need to understand PHP.

This example illustrates the concept of server-side scripting in a nutshell. The PHP has been interpreted and executed on the web server, as distinct from JavaScript and other client-side technologies interpreted and executed within a web browser on a user’s machine.

The code that you now have in this file consists of four types of text:

- HTML
- PHP tags
- PHP statements
- Whitespace
You can also add comments.

Most of the lines in the example are just plain HTML.

**PHP Tags**

The PHP code in the preceding example began with `<?php` and ended with `?>`. This is similar to all HTML tags because they all begin with a less than (`<`) symbol and end with a greater than (`>`) symbol. These symbols (`<?php` and `?>`) are called *PHP tags*. They tell the web server where the PHP code starts and finishes. Any text between the tags is interpreted as PHP. Any text outside these tags is treated as normal HTML. The PHP tags allow you to *escape* from HTML.

There are actually two styles of PHP tags; each of the following fragments of code is equivalent:

- **XML style**

  ```php
  <?php echo '<p>Order processed.</p>'; ?>
  ```

  This is the tag style that we use in this book; it is the preferred PHP tag style. The server administrator cannot turn it off, so you can guarantee it will be available on all servers, which is especially important if you are writing applications that may be used on different installations. This tag style can be used with Extensible Markup Language (XML) documents. In general, we recommend you use this tag style.

- **Short style**

  ```php
  <? echo '<p>Order processed.</p>'; ?>
  ```

  This tag style is the simplest and follows the style of a Standard Generalized Markup Language (SGML) processing instruction. To use this type of tag—which is the shortest to type—you either need to enable the `short_open_tag` setting in your configuration file or compile PHP with short tags enabled. You can find more information on how to use this tag style in Appendix A. The use of this style is not recommended for use in code you plan to distribute. It will not work in many environments as it is no longer enabled by default.

**PHP Statements**

You tell the PHP interpreter what to do by including PHP statements between your opening and closing tags. The preceding example used only one type of statement:

```php
echo '<p>Order processed.</p>';;
```

As you have probably guessed, using the `echo` construct has a very simple result: It prints (or echoes) the string passed to it to the browser. In Figure 1.2, you can see the result is that the text *Order processed.* appears in the browser window.

Notice that there is a semicolon at the end of the `echo` statement. Semicolons separate statements in PHP much like periods separate sentences in English. If you have programmed in C or Java before, you will be familiar with using the semicolon in this way.

Leaving off the semicolon is a common syntax error that is easily made. However, it's equally easy to find and to correct.
Whitespace

Spacing characters such as newlines (carriage returns), spaces, and tabs are known as whitespace. As you probably already know, browsers ignore whitespace in HTML, and so does the PHP engine. Consider these two HTML fragments:

```html
<h1>Welcome to Bob's Auto Parts!</h1><p>What would you like to order today?</p>
and
<h1>Welcome to Bob's Auto Parts!</h1><p>What would you like to order today?</p>
```

These two snippets of HTML code produce identical output because they appear the same to the browser. However, you can and are encouraged to use whitespace sensibly in your HTML as an aid to humans—to enhance the readability of your HTML code. The same is true for PHP. You don't need to have any whitespace between PHP statements, but it makes the code much easier to read if you put each statement on a separate line. For example,

```php
echo 'hello ';  
echo 'world';
```

```php
echo 'hello '; echo 'world';
```

are equivalent, but the first version is easier to read.

Comments

Comments are exactly that: Comments in code act as notes to people reading the code. Comments can be used to explain the purpose of the script, who wrote it, why they wrote it the way they did, when it was last modified, and so on. You generally find comments in all but the simplest PHP scripts.

The PHP interpreter ignores any text in comments. Essentially, the PHP parser skips over the comments, making them equivalent to whitespace.

PHP supports C, C++, and shell script-style comments.

The following is a C-style, multiline comment that might appear at the start of a PHP script:

```php
/* Author: Bob Smith  
   Last modified: April 10  
   This script processes the customer orders. */
```

Multiline comments should begin with a /* and end with */. As in C, multiline comments cannot be nested.
You can also use single-line comments, either in the C++ style:
```php
// echo '<p>Order processed.</p>'; // Start printing order
```
or in the shell script style:
```php
# echo '<p>Order processed.</p>'; # Start printing order
```
With both of these styles, everything after the comment symbol (# or //) is a comment until you reach the end of the line or the ending PHP tag, whichever comes first.

In the following line of code, the text before the closing tag, here is a comment, is part of a comment. The text after the closing tag, here is not, will be treated as HTML because it is outside the closing tag:
```php
// here is a comment ?> here is not
```

### Adding Dynamic Content

So far, you haven’t used PHP to do anything you couldn’t have done with plain HTML.

The main reason for using a server-side scripting language is to be able to provide dynamic content to a site’s users. This is an important application because content that changes according to users’ needs or over time will keep visitors coming back to a site. PHP allows you to do this easily.

Let’s start with a simple example. Replace the PHP in `processorder.php` with the following code:
```php
<?php
    echo '<p>Order processed at 
        echo date('H:i, jS F Y');
        echo '</p>';
?>
```
You could also write this on one line, using the concatenation operator (.), as
```php
<?php
    echo '<p>Order processed at '.date('H:i, jS F Y')."</p";}
?>
```
In this code, PHP’s built-in `date()` function tells the customer the date and time when his order was processed. This information will be different each time the script is run. The output of running the script on one occasion is shown in Figure 1.3.
Adding Dynamic Content

Figure 1.3  PHP's `date()` function returns a formatted date string

Calling Functions
Look at the call to `date()`. This is the general form that function calls take. PHP has an extensive library of functions you can use when developing web applications. Most of these functions need to have some data passed to them and return some data.

Now look at the function call again:

```
date('H:i, jS F')
```

Notice that it passes a string (text data) to the function inside a pair of parentheses. The element within the parentheses is called the function's argument or parameter. Such arguments are the input the function uses to output some specific results.

Using the `date()` Function
The `date()` function expects the argument you pass it to be a format string, representing the style of output you would like. Each letter in the string represents one part of the date and time. `H` is the hour in a 24-hour format with leading zeros where required, `i` is the minutes with a leading zero where required, `j` is the day of the month without a leading zero, `S` represents the ordinal suffix (in this case th), and `F` is the full name of the month.

Note
If `date()` gives you a warning about not having set the timezone, you should add the `date.timezone` setting to your `php.ini` file. More information on this can be found in the sample `php.ini` file in Appendix A.
For a full list of formats supported by `date()`, see Chapter 19, “Managing the Date and Time.”

### Accessing Form Variables

The whole point of using the order form is to collect customers’ orders. Getting the details of what the customers typed is easy in PHP, but the exact method depends on the version of PHP you are using and a setting in your `php.ini` file.

#### Form Variables

Within your PHP script, you can access each form field as a PHP variable whose name relates to the name of the form field. You can recognize variable names in PHP because they all start with a dollar sign ($). (Forgetting the dollar sign is a common programming error.)

Depending on your PHP version and setup, you can access the form data via variables in different ways. In recent versions of PHP, all but one of these ways have been deprecated, so beware if you have used PHP in the past that this has changed.

You may access the contents of the field `tireqty` in the following way:

```
$_POST['tireqty']
```

$$_POST is an array containing data submitted via an HTTP POST request—that is, the form method was set to POST. There are three of these arrays that may contain form data: $$_POST, $$_GET, and $$_REQUEST. One of the $$_GET or $$_POST arrays holds the details of all the form variables. Which array is used depends on whether the method used to submit the form was GET or POST, respectively. In addition, a combination of all data submitted via GET or POST is also available through $$_REQUEST.

If the form was submitted via the POST method, the data entered in the `tireqty` box will be stored in $$_POST['tireqty']. If the form was submitted via GET, the data will be in $$_GET['tireqty']. In either case, the data will also be available in $$_REQUEST['tireqty'].

These arrays are some of the superglobal arrays. We will revisit the superglobals when we discuss variable scope later in this chapter.

Let’s look at an example that creates easier-to-use copies of variables.

To copy the value of one variable into another, you use the assignment operator, which in PHP is an equal sign (=). The following statement creates a new variable named `tireqty` and copies the contents of $$_POST['tireqty'] into the new variable:

```
$tireqty = $_POST['tireqty'];
```

Place the following block of code at the start of the processing script. All other scripts in this book that handle data from a form contain a similar block at the start. Because this code
Accessing Form Variables

will not produce any output, placing it above or below the `<html>` and other HTML tags that start your page makes no difference. We generally place such blocks at the start of the script to make them easy to find.

```php
<?php

// create short variable names
$tireqty = $_POST['tireqty'];
$oilqty = $_POST['oilqty'];
$sparkqty = $_POST['sparkqty'];
?
```

This code creates three new variables—$tireqty, $oilqty, and $sparkqty—and sets them to contain the data sent via the POST method from the form.

You can output the values of these variables to the browser by doing, for example:

```php
echo $tireqty.' tires<br />
```

However, this approach is not recommended.

At this stage, you have not checked the variable contents to make sure sensible data has been entered in each form field. Try entering deliberately wrong data and observe what happens.

After you have read the rest of the chapter, you might want to try adding some data validation to this script.

Taking data directly from the user and outputting it to the browser like this is an extremely risky practice from a security perspective. We do not recommend this approach. You should filter input data. We will start to cover input filtering in Chapter 4, “String Manipulation and Regular Expressions,” and discuss security in depth in Chapter 14, “Web Application Security Risks.”

For now, it’s enough to know that you should echo out user data to the browser after passing it through a function called htmlspecialchars(). For example, in this case, we would do the following:

```php
echo htmlspecialchars($tireqty).' tires<br />
```

To make the script start doing something visible, add the following lines to the bottom of your PHP script:

```php
echo '<p>Your order is as follows: </p>';
echo htmlspecialchars($tireqty).' tires<br />';
echo htmlspecialchars($oilqty).' bottles of oil<br />';
echo htmlspecialchars($sparkqty).' spark plugs<br />';
```

If you now load this file in your browser, the script output should resemble what is shown in Figure 1.4. The actual values shown, of course, depend on what you typed into the form.
Figure 1.4  The form variables the user typed in are easily accessible in processorder.php

The following sections describe a couple of interesting elements of this example.

String Concatenation

In the sample script, echo prints the value the user typed in each form field, followed by some explanatory text. If you look closely at the echo statements, you can see that the variable name and following text have a period (.) between them, such as this:

```
echo htmlspecialchars($tireqty).' tires<br />';
```

This period is the string concatenation operator, which adds strings (pieces of text) together. You will often use it when sending output to the browser with echo. This way, you can avoid writing multiple echo commands.

You can also place simple variables inside a double-quoted string to be echoed. (Arrays are somewhat more complicated, so we look at combining arrays and strings in Chapter 4.) Consider this example:

```
$tireqty = htmlspecialchars($tireqty);
echo "$tireqty tires<br />
```

This is equivalent to the first statement shown in this section. Either format is valid, and which one you use is a matter of personal taste. This process, replacing a variable with its contents within a string, is known as interpolation.

Note that interpolation is a feature of double-quoted strings only. You cannot place variable names inside a single-quoted string in this way. Running the following line of code

```
echo 'Stireqty tires<br />'; 
```

simply sends $tireqty tires<br /> to the browser. Within double quotation marks, the variable name is replaced with its value. Within single quotation marks, the variable name or any other text is sent unaltered.
Understanding Identifiers

Variables and Literals

The variables and strings concatenated together in each of the `echo` statements in the sample script are different types of things. Variables are symbols for data. The strings are data themselves. When we use a piece of raw data in a program like this, we call it a literal to distinguish it from a variable. `$tireqty` is a variable, a symbol that represents the data the customer typed in. On the other hand, `' tires<br />'` is a literal. You can take it at face value. Well, almost. Remember the second example in the preceding section? PHP replaced the variable name `$tireqty` in the string with the value stored in the variable.

Remember the two kinds of strings mentioned already: ones with double quotation marks and ones with single quotation marks. PHP tries to evaluate strings in double quotation marks, resulting in the behavior shown earlier. Single-quoted strings are treated as true literals.

There is also a third way of specifying strings using the heredoc syntax (`<<<`), which will be familiar to Perl users. Heredoc syntax allows you to specify long strings tidily, by specifying an end marker that will be used to terminate the string. The following example creates a three-line string and echoes it:

```
echo <<<theEnd
    line 1
    line 2
    line 3
theEnd
```

The token `theEnd` is entirely arbitrary. It just needs to be guaranteed not to appear in the text. To close a heredoc string, place a closing token at the start of a line.

Heredoc strings are interpolated, like double-quoted strings.

Understanding Identifiers

Identifiers are the names of variables. (The names of functions and classes are also identifiers; we look at functions and classes in Chapter 5, “Reusing Code and Writing Functions,” and Chapter 6, “Object-Oriented PHP.”) You need to be aware of the simple rules defining valid identifiers:

- Identifiers can be of any length and can consist of letters, numbers, and underscores.
- Identifiers cannot begin with a digit.
- In PHP, identifiers are case sensitive. `$tireqty` is not the same as `$tireQty`. Trying to use them interchangeably is a common programming error. Function names are an exception to this rule: Their names can be used in any case.
- A variable can have the same name as a function. This usage is confusing, however, and should be avoided. Also, you cannot create a function with the same name as another function.
You can declare and use your own variables in addition to the variables you are passed from the HTML form.

One of the features of PHP is that it does not require you to declare variables before using them. A variable is created when you first assign a value to it. See the next section for details.

You assign values to variables using the assignment operator (=) as you did when copying one variable's value to another. On Bob's site, you want to work out the total number of items ordered and the total amount payable. You can create two variables to store these numbers. To begin with, you need to initialize each of these variables to zero by adding these lines to the bottom of your PHP script.

```
$totalqty = 0;
$totalamount = 0.00;
```

Each of these two lines creates a variable and assigns a literal value to it. You can also assign variable values to variables, as shown in this example:

```
$totalqty = 0;
$totalamount = $totalqty;
```

### Examining Variable Types

A variable's type refers to the kind of data stored in it. PHP provides a set of data types. Different data can be stored in different data types.

#### PHP's Data Types

PHP supports the following basic data types:

- **Integer**—Used for whole numbers
- **Float** (also called **double**)—Used for real numbers
- **String**—Used for strings of characters
- **Boolean**—Used for **true** or **false** values
- **Array**—Used to store multiple data items (see Chapter 3, “Using Arrays”)
- **Object**—Used for storing instances of classes (see Chapter 6)

Three special types are also available: NULL, resource, and callable.

Variables that have not been given a value, have been unset, or have been given the specific value **NULL** are of type **NULL**.

Certain built-in functions (such as database functions) return variables that have the type resource. They represent external resources (such as database connections). You will almost certainly not directly manipulate a resource variable, but frequently they are returned by functions and must be passed as parameters to other functions.

Callables are essentially functions that are passed to other functions.
Type Strength

PHP is called a weakly typed or dynamically typed language. In most programming languages, variables can hold only one type of data, and that type must be declared before the variable can be used, as in C. In PHP, the type of a variable is determined by the value assigned to it.

For example, when you created $totalqty and $totalamount, their initial types were determined as follows:

```php
$totalqty = 0;
$totalamount = 0.00;
```

Because you assigned 0, an integer, to $totalqty, this is now an integer type variable. Similarly, $totalamount is now of type float.

Strangely enough, you could now add a line to your script as follows:

```php
$totalamount = 'Hello';
```

The variable $totalamount would then be of type string. PHP changes the variable type according to what is stored in it at any given time.

This ability to change types transparently on the fly can be extremely useful. Remember PHP “automagically” knows what data type you put into your variable. It returns the data with the same data type when you retrieve it from the variable.

Type Casting

You can pretend that a variable or value is of a different type by using a type cast. This feature works identically to the way it works in C. You simply put the temporary type in parentheses in front of the variable you want to cast.

For example, you could have declared the two variables from the preceding section using a cast:

```php
$totalqty = 0;
$totalamount = (float)$totalqty;
```

The second line means “Take the value stored in $totalqty, interpret it as a float, and store it in $totalamount.” The $totalamount variable will be of type float. The cast variable does not change types, so $totalqty remains of type integer.

You can also use built-in functions to test and set type, which you will learn about later in this chapter.

Variable Variables

PHP provides one other type of variable: the variable variable. Variable variables enable you to change the name of a variable dynamically.

As you can see, PHP allows a lot of freedom in this area. All languages enable you to change the value of a variable, but not many allow you to change the variable's type, and even fewer allow you to change the variable's name.
A variable variable works by using the value of one variable as the name of another. For example, you could set

```php
$varname = 'tireqty';
```

You can then use `$$varname` in place of `$tireqty`. For example, you can set the value of `$tireqty` as follows:

```php
$$varname = 5;
```

This is equivalent to

```php
$tireqty = 5;
```

This approach might seem somewhat obscure, but we’ll revisit its use later. Instead of having to list and use each form variable separately, you can use a loop and variable variable to process them all automatically. You can find an example illustrating this in the section on for loops later in this chapter.

### Declaring and Using Constants

As you saw previously, you can readily change the value stored in a variable. You can also declare constants. A constant stores a value just like a variable, but its value is set once and then cannot be changed elsewhere in the script.

In the sample application, you might store the prices for each item on sale as a constant. You can define these constants using the `define` function:

```php
define('TIREPRICE', 100);
define('OILPRICE', 10);
define('SPARKPRICE', 4);
```

Now add these lines of code to your script. You now have three constants that can be used to calculate the total of the customer’s order.

Notice that the names of the constants appear in uppercase. This convention, borrowed from C, makes it easy to distinguish between variables and constants at a glance. Following this convention is not required but will make your code easier to read and maintain.

One important difference between constants and variables is that when you refer to a constant, it does not have a dollar sign in front of it. If you want to use the value of a constant, use its name only. For example, to use one of the constants just created, you could type

```php
echo TIREPRICE;
```

As well as the constants you define, PHP sets a large number of its own. An easy way to obtain an overview of them is to run the `phpinfo()` function:

```php
phpinfo();
```

This function provides a list of PHP’s predefined variables and constants, among other useful information. We will discuss some of them as we go along.

One other difference between variables and constants is that constants can store only boolean, integer, float, or string data. These types are collectively known as scalar values.
Understanding Variable Scope

The term **scope** refers to the places within a script where a particular variable is visible. The six basic scope rules in PHP are as follows:

- Built-in superglobal variables are visible everywhere within a script.
- Constants, once declared, are always visible globally; that is, they can be used inside and outside functions.
- Global variables declared in a script are visible throughout that script, but not inside functions.
- Variables inside functions that are declared as global refer to the global variables of the same name.
- Variables created inside functions and declared as static are invisible from outside the function but keep their value between one execution of the function and the next. (We explain this idea fully in Chapter 5.)
- Variables created inside functions are local to the function and cease to exist when the function terminates.

The arrays `$_GET` and `$_POST` and some other special variables have their own scope rules. They are known as **superglobals** and can be seen everywhere, both inside and outside functions.

The complete list of superglobals is as follows:

- `$GLOBALS`—An array of all global variables (Like the `global` keyword, this allows you to access global variables inside a function—for example, as `$GLOBALS['myvariable']`.)
- `$SERVER`—An array of server environment variables
- `$GET`—An array of variables passed to the script via the `GET` method
- `$POST`—An array of variables passed to the script via the `POST` method
- `$COOKIE`—An array of cookie variables
- `$FILES`—An array of variables related to file uploads
- `$ENV`—An array of environment variables
- `$REQUEST`—An array of all user input including the contents of input including `$GET`, `$POST`, and `$COOKIE` (but not including `$FILES`)
- `$SESSION`—An array of session variables

We come back to each of these superglobals throughout the book as they become relevant.

We cover scope in more detail when we discuss functions and classes later in this chapter. For the time being, all the variables we use are global by default.
Using Operators

Operators are symbols that you can use to manipulate values and variables by performing an operation on them. You need to use some of these operators to work out the totals and tax on the customer's order.

We've already mentioned two operators: the assignment operator (=) and the string concatenation operator (.). In the following sections, we describe the complete list.

In general, operators can take one, two, or three arguments, with the majority taking two. For example, the assignment operator takes two: the storage location on the left side of the = symbol and an expression on the right side. These arguments are called operands—that is, the things that are being operated upon.

Arithmetic Operators

Arithmetic operators are straightforward; they are just the normal mathematical operators. PHP's arithmetic operators are shown in Table 1.1.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>$a + $b</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>$a - $b</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>$a * $b</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>$a / $b</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>$a % $b</td>
</tr>
</tbody>
</table>

With each of these operators, you can store the result of the operation, as in this example: $result = $a + $b;

Addition and subtraction work as you would expect. The result of these operators is to add or subtract, respectively, the values stored in the $a and $b variables.

You can also use the subtraction symbol (-) as a unary operator—that is, an operator that takes one argument or operand—to indicate negative numbers, as in this example: $a = -1;

Multiplication and division also work much as you would expect. Note the use of the asterisk as the multiplication operator rather than the regular multiplication symbol, and the forward slash as the division operator rather than the regular division symbol.

The modulus operator returns the remainder calculated by dividing the $a variable by the $b variable. Consider this code fragment:

```
$a = 27;
$b = 10;
$result = $a % $b;
```
The value stored in the $result variable is the remainder when you divide 27 by 10—that is, 7.
You should note that arithmetic operators are usually applied to integers or doubles. If you apply them to strings, PHP will try to convert the string to a number. If it contains an e or an E, it will be read as being in scientific notation and converted to a float; otherwise, it will be converted to an integer. PHP will look for digits at the start of the string and use them as the value; if there are none, the value of the string will be zero.

String Operators
You've already seen and used the only string operator. You can use the string concatenation operator to add two strings and to generate and store a result much as you would use the addition operator to add two numbers:

```php
$a = "Bob's ";
$b = "Auto Parts";
$result = $a.$b;
```

The $result variable now contains the string "Bob's Auto Parts".

Assignment Operators
You've already seen the basic assignment operator (=). Always refer to this as the assignment operator and read it as “is set to.” For example,

```php
$totalqty = 0;
```

This line should be read as "$totalqty is set to zero." We explain why when we discuss the comparison operators later in this chapter, but if you call it equals, you will get confused.

Values Returned from Assignment
Using the assignment operator returns an overall value similar to other operators. If you write

```php
$a + $b
```

the value of this expression is the result of adding the $a and $b variables together. Similarly, you can write

```php
$a = 0;
```

The value of this whole expression is zero.

This technique enables you to form expressions such as

```php
$b = 6 + ($a = 5);
```

This line sets the value of the $b variable to 11. This behavior is generally true of assignments: The value of the whole assignment statement is the value that is assigned to the left operand.

When working out the value of an expression, you can use parentheses to increase the precedence of a subexpression, as shown here. This technique works exactly the same way as in mathematics.
Combined Assignment Operators

In addition to the simple assignment, there is a set of combined assignment operators. Each of them is a shorthand way of performing another operation on a variable and assigning the result back to that variable. For example,

```php
$a += 5;
```

This is equivalent to writing

```php
$a = $a + 5;
```

Combined assignment operators exist for each of the arithmetic operators and for the string concatenation operator. A summary of all the combined assignment operators and their effects is shown in Table 1.2.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>$a += $b</td>
<td>$a = $a + $b</td>
</tr>
<tr>
<td>-=</td>
<td>$a -= $b</td>
<td>$a = $a - $b</td>
</tr>
<tr>
<td>*=</td>
<td>$a *= $b</td>
<td>$a = $a * $b</td>
</tr>
<tr>
<td>/=</td>
<td>$a /= $b</td>
<td>$a = $a / $b</td>
</tr>
<tr>
<td>%=</td>
<td>$a %= $b</td>
<td>$a = $a % $b</td>
</tr>
<tr>
<td>.=</td>
<td>$a .= $b</td>
<td>$a = $a . $b</td>
</tr>
</tbody>
</table>

Pre- and Post-Increment and Decrement

The pre- and post-increment (++) and decrement (--) operators are similar to the += and -= operators, but with a couple of twists.

All the increment operators have two effects: They increment and assign a value. Consider the following:

```php
$s = 4;
echo ++$s;
```

The second line uses the pre-increment operator, so called because the ++ appears before the $s. This has the effect of first incrementing $s by 1 and second, returning the incremented value. In this case, $s is incremented to 5, and then the value 5 is returned and printed. The value of this whole expression is 5. (Notice that the actual value stored in $s is changed: It is not just returning $s + 1.)

If the ++ is after the $s, however, you are using the post-increment operator. It has a different effect. Consider the following:

```php
$s = 4;
echo $s++;
In this case, the effects are reversed. That is, first, the value of \(a\) is returned and printed, and second, it is incremented. The value of this whole expression is 4. This is the value that will be printed. However, the value of \(a\) after this statement is executed is 5.

As you can probably guess, the behavior is similar for the \(--\) (decrement) operator. However, the value of \(a\) is decremented instead of being incremented.

**Reference Operator**

The reference operator (\&, an ampersand) can be used in conjunction with assignment. Normally, when one variable is assigned to another, a copy is made of the first variable and stored elsewhere in memory. For example,

\[
\begin{align*}
\text{\$a} &= 5; \\
\text{\$b} &= \$a;
\end{align*}
\]

These code lines make a second copy of the value in \(a\) and store it in \(b\). If you subsequently change the value of \(a\), \(b\) will not change:

\[
\text{\$a} = 7; // \$b\text{ will still be } 5
\]

You can avoid making a copy by using the reference operator. For example,

\[
\begin{align*}
\text{\$a} &= 5; \\
\text{\$b} &= &\&\text{\$a}; \\
\text{\$a} &= 7; // \$a\text{ and } \$b\text{ are now both } 7
\end{align*}
\]

References can be a bit tricky. Remember that a reference is like an alias rather than like a pointer. Both \(a\) and \(b\) point to the same piece of memory. You can change this by unsetting one of them as follows:

\[
\text{unset}(\$a);
\]

Unsetting does not change the value of \(b\) (7) but does break the link between \(a\) and the value 7 stored in memory.

**Comparison Operators**

The comparison operators compare two values. Expressions using these operators return either of the logical values `true` or `false` depending on the result of the comparison.

**The Equal Operator**

The equal comparison operator (\(==\), two equal signs) enables you to test whether two values are equal. For example, you might use the expression

\[
\text{\$a} == \$b
\]

to test whether the values stored in \(a\) and \(b\) are the same. The result returned by this expression is `true` if they are equal or `false` if they are not.

You might easily confuse `==` with `=` the assignment operator. Using the wrong operator will work without giving an error but generally will not give you the result you wanted. In general,
nonzero values evaluate to true and zero values to false. Say that you have initialized two variables as follows:

```php
$a = 5;
$b = 7;
```

If you then test `$_a = $b`, the result will be true. Why? The value of `$_a = $b` is the value assigned to the left side, which in this case is 7. Because 7 is a nonzero value, the expression evaluates to true. If you intended to test `$_a == $b`, which evaluates to false, you have introduced a logic error in your code that can be extremely difficult to find. Always check your use of these two operators and check that you have used the one you intended to use.

Using the assignment operator rather than the equals comparison operator is an easy mistake to make, and you will probably make it many times in your programming career.

**Other Comparison Operators**

PHP also supports a number of other comparison operators. A summary of all the comparison operators is shown in Table 1.3. One to note is the identical operator (===), which returns true only if the two operands are both equal and of the same type. For example, `0 =='0'` will be true, but `0==='0'` will not because one zero is an integer and the other zero is a string.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equals</td>
<td>$a == $b</td>
</tr>
<tr>
<td>===</td>
<td>Identical</td>
<td>$a === $b</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal</td>
<td>$a != $b</td>
</tr>
<tr>
<td>!==</td>
<td>Not identical</td>
<td>$a !== $b</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal (comparison operator)</td>
<td>$a &lt;&gt; $b</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>$a &lt; $b</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than (comparison operator)</td>
<td>$a &gt; $b</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>$a &lt;= $b</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>$a &gt;= $b</td>
</tr>
</tbody>
</table>

**Logical Operators**

The logical operators combine the results of logical conditions. For example, you might be interested in a case in which the value of a variable, $a, is between 0 and 100. You would need to test both the conditions $a >= 0 and $a <= 100, using the AND operator, as follows:

```php
$a >= 0 && $a <=100
```

PHP supports logical AND, OR, XOR (exclusive or), and NOT.

The set of logical operators and their use is summarized in Table 1.4.
Table 1.4  PHP's Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Use</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>NOT</td>
<td>!$b</td>
<td>Returns true if $b is false and vice versa</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
<td>$a &amp;&amp; $b</td>
<td>Returns true if both $a and $b are true; otherwise false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OR</td>
</tr>
<tr>
<td>and</td>
<td>AND</td>
<td>$a and $b</td>
<td>Same as &amp;&amp;, but with lower precedence</td>
</tr>
<tr>
<td>or</td>
<td>OR</td>
<td>$a or $b</td>
<td>Same as</td>
</tr>
<tr>
<td>xor</td>
<td>XOR</td>
<td>$a x or $b</td>
<td>Returns true if either $a or $b is true, and false if they are both true or both false.</td>
</tr>
</tbody>
</table>

The and and or operators have lower precedence than the && and || operators. We cover precedence in more detail later in this chapter.

Bitwise Operators

The bitwise operators enable you to treat an integer as the series of bits used to represent it. You probably will not find a lot of use for the bitwise operators in PHP, but a summary is shown in Table 1.5.

Table 1.5  PHP's Bitwise Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Use</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
<td>$a &amp; $b</td>
<td>Bits set in $a and $b are set in the result.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a</td>
<td>$b</td>
</tr>
<tr>
<td>~</td>
<td>Bitwise NOT</td>
<td>~$a</td>
<td>Bits set in $a are not set in the result and vice versa.</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise XOR</td>
<td>$a ^ $b</td>
<td>Bits set in $a or $b but not in both are set in the result.</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Left shift</td>
<td>$a &lt;&lt; $b</td>
<td>Shifts $a left $b bits.</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Right shift</td>
<td>$a &gt;&gt; $b</td>
<td>Shifts $a right $b bits.</td>
</tr>
</tbody>
</table>

Other Operators

In addition to the operators we have covered so far, you can use several others.

The comma operator (,) separates function arguments and other lists of items. It is normally used incidentally.
Two special operators, `new` and `->`, are used to instantiate a class and access class members, respectively. They are covered in detail in Chapter 6.

There are a few others that we discuss briefly here.

**The Ternary Operator**

The ternary operator (`?:`) takes the following form:

```
condition ? value if true : value if false
```

This operator is similar to the expression version of an `if-else` statement, which is covered later in this chapter.

A simple example is

```
($grade >= 50 ? 'Passed' : 'Failed')
```

This expression evaluates student grades to 'Passed' or 'Failed'.

**The Error Suppression Operator**

The error suppression operator (`@`) can be used in front of any expression—that is, anything that generates or has a value. For example,

```
$a = @(57/0);
```

Without the `@` operator, this line generates a divide-by-zero warning. With the operator included, the error is suppressed.

If you are suppressing warnings in this way, you need to write some error handling code to check when a warning has occurred. If you have PHP set up with the `track_errors` feature enabled in `php.ini`, the error message will be stored in the global variable `$php_errormsg`.

**The Execution Operator**

The execution operator is really a pair of operators—a pair of backticks (``) in fact. The backtick is not a single quotation mark; it is usually located on the same key as the ~ (tilde) symbol on your keyboard.

PHP attempts to execute whatever is contained between the backticks as a command at the server’s command line. The value of the expression is the output of the command.

For example, under Unix-like operating systems, you can use

```
$out = `ls -la`;
echo '<pre>'.$out.'</pre';
```

Or, equivalently on a Windows server, you can use

```
$out = `dir c:`;
echo '<pre>'.$out.'</pre';
```
Either version obtains a directory listing and stores it in \$out. It can then be echoed to the browser or dealt with in any other way.

There are other ways of executing commands on the server. We cover them in Chapter 17, “Interacting with the File System and the Server.”

**Array Operators**

There are a number of array operators. The array element operators (\[\]) enable you to access array elements. You can also use the \=> operator in some array contexts. These operators are covered in Chapter 3.

You also have access to a number of other array operators. We cover them in detail in Chapter 3 as well, but we included them here in Table 1.6 for completeness.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Use</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Union</td>
<td>$a + $b</td>
<td>Returns an array containing everything in $a and $b</td>
</tr>
<tr>
<td>==</td>
<td>Equality</td>
<td>$a == $b</td>
<td>Returns \text{true} if $a and $b have the same key and value pairs</td>
</tr>
<tr>
<td>===</td>
<td>Identity</td>
<td>$a === $b</td>
<td>Returns \text{true} if $a and $b have the same key and value pairs in the same order and of the same type.</td>
</tr>
<tr>
<td>!=</td>
<td>Inequality</td>
<td>$a != $b</td>
<td>Returns \text{true} if $a and $b are not equal</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Inequality</td>
<td>$a &lt;&gt; $b</td>
<td>Returns \text{true} if $a and $b are not equal</td>
</tr>
<tr>
<td>!==</td>
<td>Non-identity</td>
<td>$a !== $b</td>
<td>Returns \text{true} if $a and $b are not identical</td>
</tr>
</tbody>
</table>

You will notice that the array operators in Table 1.6 all have equivalent operators that work on scalar variables. As long as you remember that + performs addition on scalar types and union on arrays—even if you have no interest in the set arithmetic behind that behavior—the behaviors should make sense. You cannot usefully compare arrays to scalar types.

**The Type Operator**

There is one type operator: \text{instanceof}. This operator is used in object-oriented programming, but we mention it here for completeness. (Object-oriented programming is covered in Chapter 6.)

The \text{instanceof} operator allows you to check whether an object is an instance of a particular class, as in this example:

```php
class sampleClass{};
$myObject = new sampleClass();
if ($myObject instanceof sampleClass)
    echo "myObject is an instance of sampleClass";
```
Working Out the Form Totals

Now that you know how to use PHP's operators, you are ready to work out the totals and tax on Bob's order form. To do this, add the following code to the bottom of your PHP script:

```php
$totalqty = 0;
$totalqty = $tireqty + $oilqty + $sparkqty;
echo "Items ordered: $totalqty.<br/>
$totalamount = 0.00;

define('TIREPRICE', 100);
define('OILPRICE', 10);
define('SPARKPRICE', 4);

$totalamount = $tireqty * TIREPRICE
+ $oilqty * OILPRICE
+ $sparkqty * SPARKPRICE;

echo "Subtotal: $".number_format($totalamount,2)."<br/>"

taxrate = 0.10; // local sales tax is 10%
$totalamount = $totalamount * (1 + $taxrate);
echo "Total including tax: $".number_format($totalamount,2).""
```

If you refresh the page in your browser window, you should see output similar to Figure 1.5.

As you can see, this piece of code uses several operators. It uses the addition (+) and multiplication (*) operators to work out the amounts and the string concatenation operator (.) to set up the output to the browser.

Figure 1.5  The totals of the customer's order have been calculated, formatted, and displayed
It also uses the `number_format()` function to format the totals as strings with two decimal places. This is a function from PHP’s Math library.

If you look closely at the calculations, you might ask why the calculations were performed in the order they were. For example, consider this statement:

```php
$totalamount = $tireqty * TIREPRICE + $oilqty * OILPRICE + $sparkqty * SPARKPRICE;
```

The total amount seems to be correct, but why were the multiplications performed before the additions? The answer lies in the precedence of the operators—that is, the order in which they are evaluated.

**Understanding Precedence and Associativity**

In general, operators have a set precedence, or order, in which they are evaluated. Operators also have associativity, which is the order in which operators of the same precedence are evaluated. This order is generally left to right (called *left* for short), right to left (called *right* for short), or *not relevant*.

Table 1.7 shows operator precedence and associativity in PHP. In this table, operators with the lowest precedence are at the top, and precedence increases as you go down the table.

<table>
<thead>
<tr>
<th>Associativity</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>left</em> ,</td>
<td></td>
</tr>
<tr>
<td><em>left</em> Or</td>
<td></td>
</tr>
<tr>
<td><em>left</em> Xor</td>
<td></td>
</tr>
<tr>
<td><em>left</em> And</td>
<td></td>
</tr>
<tr>
<td><em>right</em> Print</td>
<td></td>
</tr>
<tr>
<td><em>left</em> = += -= *= /= .= %= &amp;=</td>
<td>= ^= ~== &lt;&lt;= &gt;&gt;=</td>
</tr>
<tr>
<td><em>left</em> ? :</td>
<td></td>
</tr>
<tr>
<td><em>left</em></td>
<td></td>
</tr>
<tr>
<td><em>left</em> &amp;&amp;</td>
<td></td>
</tr>
<tr>
<td><em>left</em></td>
<td></td>
</tr>
<tr>
<td><em>left</em> ^</td>
<td></td>
</tr>
<tr>
<td><em>left</em> &amp;</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>== != === !==</td>
</tr>
</tbody>
</table>
### Associativity

<table>
<thead>
<tr>
<th>Associativity</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>&lt; &lt;= &gt; &gt;=</td>
</tr>
<tr>
<td>left</td>
<td>&lt;&lt; &gt;&gt;</td>
</tr>
<tr>
<td>left</td>
<td>+ - .</td>
</tr>
<tr>
<td>left</td>
<td>* / %</td>
</tr>
<tr>
<td>right</td>
<td>!</td>
</tr>
<tr>
<td>n/a</td>
<td>Instanceof</td>
</tr>
<tr>
<td>right</td>
<td>~ (int) (float) (string) (array) (object) (bool) @</td>
</tr>
<tr>
<td>n/a</td>
<td>++ --</td>
</tr>
<tr>
<td>n/a</td>
<td>[]</td>
</tr>
<tr>
<td>n/a</td>
<td>clone new</td>
</tr>
<tr>
<td>n/a</td>
<td>()</td>
</tr>
</tbody>
</table>

Notice that we haven't yet covered the operator with the highest precedence: plain old parentheses. The effect of using parentheses is to raise the precedence of whatever is contained within them. This is how you can deliberately manipulate or work around the precedence rules when you need to.

Remember this part of the preceding example:

```php
$totalamount = $totalamount * (1 + $taxrate);
```

If you had written

```php
$totalamount = $totalamount * 1 + $taxrate;
```

the multiplication operation, having higher precedence than the addition operation, would be performed first, giving an incorrect result. By using the parentheses, you can force the subexpression `1 + $taxrate` to be evaluated first.

You can use as many sets of parentheses as you like in an expression. The innermost set of parentheses is evaluated first.

Also note one other operator in this table we have not yet covered: the `print` language construct, which is equivalent to `echo`. Both constructs generate output.

We generally use `echo` in this book, but you can use `print` if you find it more readable. Neither `print` nor `echo` is really a function, but both can be called as a function with parameters in parentheses. Both can also be treated as an operator: You simply place the string to work with after the keyword `echo` or `print`.

Calling `print` as a function causes it to return a value (1). This capability might be useful if you want to generate output inside a more complex expression but does mean that `print` is marginally slower than `echo`. 

Using Variable Handling Functions

Before we leave the world of variables and operators, let’s look at PHP’s variable handling functions. PHP provides a library of functions that enable you to manipulate and test variables in different ways.

Testing and Setting Variable Types

Most of the variable functions are related to testing the type of function. The two most general are `gettype()` and `settype()`. They have the following function prototypes; that is, this is what arguments expect and what they return:

```php
string gettype(mixed var);
bool settype(mixed var, string type);
```

To use `gettype()`, you pass it a variable. It determines the type and returns a string containing the type name: `bool`, `int`, `double` (for floats, confusingly, for historical reasons), `string`, `array`, `object`, `resource`, or `NULL`. It returns `unknown` type if it is not one of the standard types.

To use `settype()`, you pass it a variable for which you want to change the type and a string containing the new type for that variable from the previous list.

**Note**

This book and the php.net documentation refer to the data type “mixed.” There is no such data type, but because PHP is so flexible with type handling, many functions can take many (or any) data types as an argument. Arguments for which many types are permitted are shown with the pseudo-type “mixed.”

You can use these functions as follows:

```php
$a = 56;
echo gettype($a).'<br '/>
settype($a, 'float');
echo gettype($a).'<br '/>
```

When `gettype()` is called the first time, the type of `$a` is integer. After the call to `settype()`, the type is changed to `float`, which is reported as `double`. (Be aware of this difference.)

PHP also provides some specific type-testing functions. Each takes a variable as an argument and returns either `true` or `false`. The functions are

- `is_array()`—Checks whether the variable is an array
- `is_double()`, `is_float()`, `is_real()` (All the same function)—Checks whether the variable is a float
- `is_long()`, `is_int()`, `is_integer()` (All the same function)—Checks whether the variable is an integer
- `is_string()` — Checks whether the variable is a string
- `is_bool()` — Checks whether the variable is a boolean
- `is_object()` — Checks whether the variable is an object
- `is_resource()` — Checks whether the variable is a resource
- `is_null()` — Checks whether the variable is null
- `is_scalar()` — Checks whether the variable is a scalar—that is, an integer, boolean, string, or float
- `is_numeric()` — Checks whether the variable is any kind of number or a numeric string
- `is_callable()` — Checks whether the variable is the name of a valid function

### Testing Variable Status

PHP has several functions for testing the status of a variable. The first is `isset()`, which has the following prototype:

```php
bool isset(mixed var[, mixed var[, ...]])
```

This function takes a variable name as an argument and returns `true` if it exists and `false` otherwise. You can also pass in a comma-separated list of variables, and `isset()` will return `true` if all the variables are set.

You can wipe a variable out of existence by using its companion function, `unset()`, which has the following prototype:

```php
void unset(mixed var[, mixed var[, ...]])
```

This function gets rid of the variable it is passed.

The `empty()` function checks to see whether a variable exists and has a nonempty, nonzero value; it returns `true` or `false` accordingly. It has the following prototype:

```php
bool empty(mixed var)
```

Let's look at an example using these three functions.

Try adding the following code to your script temporarily:

```php
echo 'isset($tireqty): '.isset($tireqty).'<br />
echo 'isset($nothere): '.isset($nothere).'<br />
echo 'empty($tireqty): '.empty($tireqty).'<br />
echo 'empty($nothere): '.empty($nothere).'<br '/
```

Refresh the page to see the results.

The variable `$tireqty` should return `1 (true)` from `isset()` regardless of what value you entered in that form field and regardless of whether you entered a value at all. Whether it is `empty()` depends on what you entered in it.
The variable $nothere does not exist, so it generates a blank (false) result from isset() and a 1 (true) result from empty().

These functions are handy when you need to make sure that the user filled out the appropriate fields in the form.

Reinterpreting Variables
You can achieve the equivalent of casting a variable by calling a function. The following three functions can be useful for this task:

- `intval(mixed var[, int base=10])`
- `floatval(mixed var)`
- `strval(mixed var)`

Each accepts a variable as input and returns the variable's value converted to the appropriate type. The `intval()` function also allows you to specify the base for conversion when the variable to be converted is a string. (This way, you can convert, for example, hexadecimal strings to integers.)

Making Decisions with Conditionals
Control structures are the structures within a language that allow you to control the flow of execution through a program or script. You can group them into conditional (or branching) structures and repetition structures (or loops).

If you want to sensibly respond to your users’ input, your code needs to be able to make decisions. The constructs that tell your program to make decisions are called conditionals.

if Statements
You can use an if statement to make a decision. You should give the if statement a condition to use. If the condition is true, the following block of code will be executed. Conditions in if statements must be surrounded by parentheses ()

For example, if a visitor orders no tires, no bottles of oil, and no spark plugs from Bob, it is probably because she accidentally clicked the Submit Order button before she had finished filling out the form. Rather than telling the visitor “Order processed,” the page could give her a more useful message.

When the visitor orders no items, you might like to say, “You did not order anything on the previous page!” You can do this easily by using the following if statement:

```php
if ($totalqty == 0)
    echo 'You did not order anything on the previous page!<br />';
```

The condition you are using here is `$totalqty == 0`. Remember that the equals operator (==) behaves differently from the assignment operator (=).
Chapter 1  PHP Crash Course

The condition $totalqty == 0 will be true if $totalqty is equal to zero. If $totalqty is not equal to zero, the condition will be false. When the condition is true, the echo statement will be executed.

**Code Blocks**

Often you may have more than one statement you want executed according to the actions of a conditional statement such as if. You can group a number of statements together as a block. To declare a block, you enclose it in curly braces:

```php
if ($totalqty == 0) {
    echo '<p style="color:red">';
    echo 'You did not order anything on the previous page!';
    echo '</p>);
}
```

The three lines enclosed in curly braces are now a block of code. When the condition is true, all three lines are executed. When the condition is false, all three lines are ignored.

**Note**

As already mentioned, PHP does not care how you lay out your code. However, you should indent your code for readability purposes. Indenting is used to enable you to see at a glance which lines will be executed only if conditions are met, which statements are grouped into blocks, and which statements are parts of loops or functions. In the previous examples, you can see that the statement depending on the if statement and the statements making up the block are indented.

**else Statements**

You may often need to decide not only whether you want an action performed, but also which of a set of possible actions you want performed.

An else statement allows you to define an alternative action to be taken when the condition in an if statement is false. Say you want to warn Bob's customers when they do not order anything. On the other hand, if they do make an order, instead of a warning, you want to show them what they ordered.

If you rearrange the code and add an else statement, you can display either a warning or a summary:

```php
if ($totalqty == 0) {
    echo "You did not order anything on the previous page!\n";
} else {
    echo htmlspecialchars($tireqty).' tires<br />
    echo htmlspecialchars($oilqty).' bottles of oil<br />
    echo htmlspecialchars($sparkqty).' spark plugs<br />
}
```
You can build more complicated logical processes by nesting if statements within each other. In the following code, the summary will be displayed only if the condition $totalqty == 0 is true, and each line in the summary will be displayed only if its own condition is met:

```php
if ($totalqty == 0) {
    echo "You did not order anything on the previous page!\n";
} else {
    if ($tireqty > 0)
        echo htmlspecialchars($tireqty) . ' tires\n';
    if ($oilqty > 0)
        echo htmlspecialchars($oilqty) . ' bottles of oil\n';
    if ($sparkqty > 0)
        echo htmlspecialchars($sparkqty) . ' spark plugs\n';
}
```

### elseif Statements

For many of the decisions you make, you have more than two options. You can create a sequence of many options using the elseif statement, which is a combination of an else and an if statement. When you provide a sequence of conditions, the program can check each until it finds one that is true.

Bob provides a discount for large orders of tires. The discount scheme works like this:

- Fewer than 10 tires purchased—No discount
- 10–49 tires purchased—5% discount
- 50–99 tires purchased—10% discount
- 100 or more tires purchased—15% discount

You can create code to calculate the discount using conditions and if and elseif statements. In this case, you need to use the AND operator (&&) to combine two conditions into one:

```php
if ($tireqty < 10) {
    $discount = 0;
} elseif (($tireqty >= 10) && ($tireqty <= 49)) {
    $discount = 5;
} elseif (($tireqty >= 50) && ($tireqty <= 99)) {
    $discount = 10;
} elseif ($tireqty >= 100) {
    $discount = 15;
}
```

Note that you are free to type elseif or else if—versions with or without a space are both correct.

If you are going to write a cascading set of elseif statements, you should be aware that only one of the blocks or statements will be executed. It did not matter in this example because
all the conditions were mutually exclusive; only one can be true at a time. If you write conditions in a way that more than one could be true at the same time, only the block or statement following the first true condition will be executed.

**switch Statements**

The `switch` statement works in a similar way to the `if` statement, but it allows the condition to take more than two values. In an `if` statement, the condition can be either `true` or `false`. In a `switch` statement, the condition can take any number of different values, as long as it evaluates to a simple type (integer, string, or float). You need to provide a `case` statement to handle each value you want to react to and, optionally, a default case to handle any that you do not provide a specific `case` statement for.

Bob wants to know what forms of advertising are working for him, so you can add a question to the order form. Insert this HTML into the order form, and the form will resemble Figure 1.6:

```html
<tr>
    <td>How did you find Bob's?</td>
    <td><select name="find">
        <option value = "a">I'm a regular customer</option>
        <option value = "b">TV advertising</option>
        <option value = "c">Phone directory</option>
        <option value = "d">Word of mouth</option>
    </select>
</td>
</tr>
```

![Figure 1.6 The order form now asks visitors how they found Bob's Auto Parts](image_url)
This HTML code adds a new form variable (called find) whose value will be 'a', 'b', 'c', or 'd'. You could handle this new variable with a series of if and elseif statements like this:

```php
if ($find == "a") {
    echo "<p>Regular customer.</p>";
} elseif ($find == "b") {
    echo "<p>Customer referred by TV advert.</p>";
} elseif ($find == "c") {
    echo "<p>Customer referred by phone directory.</p>";
} elseif ($find == "d") {
    echo "<p>Customer referred by word of mouth.</p>";
} else {
    echo "<p>We do not know how this customer found us.</p>";
}
```

Alternatively, you could write a switch statement:

```php
switch($find) {
    case "a":
        echo "<p>Regular customer.</p>";
        break;
    case "b":
        echo "<p>Customer referred by TV advert.</p>";
        break;
    case "c":
        echo "<p>Customer referred by phone directory.</p>";
        break;
    case "d":
        echo "<p>Customer referred by word of mouth.</p>";
        break;
    default:
        echo "<p>We do not know how this customer found us.</p>";
        break;
}
```

(Note that both of these examples assume you have extracted $find from the $_POST array.)

The switch statement behaves somewhat differently from an if or elseif statement. An if statement affects only one statement unless you deliberately use curly braces to create a block of statements. A switch statement behaves in the opposite way. When a case statement in a switch is activated, PHP executes statements until it reaches a break statement. Without break statements, a switch would execute all the code following the case that was true. When a break statement is reached, the next line of code after the switch statement is executed.

**Comparing the Different Conditionals**

If you are not familiar with the statements described in the preceding sections, you might be asking, “Which one is the best?”
That is not really a question we can answer. There is nothing that you can do with one or more \texttt{else}, \texttt{elseif}, or \texttt{switch} statements that you cannot do with a set of \texttt{if} statements. You should try to use whichever conditional will be most readable in your situation. You will acquire a feel for which suits different situations as you gain experience.

**Repeating Actions Through Iteration**

One thing that computers have always been very good at is automating repetitive tasks. If you need something done the same way a number of times, you can use a loop to repeat some parts of your program.

Bob wants a table displaying the freight cost that will be added to a customer’s order. With the courier Bob uses, the cost of freight depends on the distance the parcel is being shipped. This cost can be worked out with a simple formula.

You want the freight table to resemble the table in Figure 1.7.

![Figure 1.7](http://www.yourdomain.com/chapter01/freight.html)

This table shows the cost of freight as distance increases

Listing 1.2 shows the HTML that displays this table. You can see that it is long and repetitive.

**Listing 1.2  freight.html—HTML for Bob’s Freight Table**

```html
<!DOCTYPE html>
<html>
<head>
  <title>Bob’s Auto Parts - Freight Costs</title>
</head>
<body>
```

![Listing 1.2](http://www.yourdomain.com/chapter01/freight.html)
Rather than requiring an easily bored human—who must be paid for his time—to type the HTML, having a cheap and tireless computer do it would be helpful.

Loop statements tell PHP to execute a statement or block repeatedly.

### while Loops

The simplest kind of loop in PHP is the **while** loop. Like an if statement, it relies on a condition. The difference between a while loop and an if statement is that an if statement executes the code that follows it only once if the condition is **true**. A while loop executes the block repeatedly for as long as the condition is **true**.

You generally use a while loop when you don't know how many iterations will be required to make the condition true. If you require a fixed number of iterations, consider using a **for** loop.

The basic structure of a while loop is:

```
while( condition ) expression;
```
The following `while` loop will display the numbers from 1 to 5:

```php
$num = 1;
while ($num <= 5 ){
    echo $num."<br />
";
    $num++;
}
```

At the beginning of each iteration, the condition is tested. If the condition is `false`, the block will not be executed and the loop will end. The next statement after the loop will then be executed.

You can use a `while` loop to do something more useful, such as display the repetitive freight table in Figure 1.7. Listing 1.3 uses a `while` loop to generate the freight table.

**Listing 1.3 freight.php—Generating Bob’s Freight Table with PHP**

```html
<!DOCTYPE html>
<html>
<head>
<title>Bob’s Auto Parts - Freight Costs</title>
</head>
<body>
<table style="border: 0px; padding: 3px">
<tr>
<td style="background: #cccccc; text-align: center;">Distance</td>
<td style="background: #cccccc; text-align: center;">Cost</td>
</tr>
<?php
$distance = 50;
while ($distance <= 250) {
    echo "<tr>
        <td style="text-align: right;">".$distance."</td>
        <td style="text-align: right;">".($distance / 10)."</td>
    </tr>";
    $distance += 50;
}
?>
</table>
</body>
</html>
```

To make the HTML generated by the script readable, you need to include newlines and spaces. As already mentioned, browsers ignore this whitespace, but it is important for human readers. You often need to look at the HTML if your output is not what you were seeking.

In Listing 1.3, you can see `\n` inside some of the strings. When inside a double-quoted string, this character sequence represents a newline character.
for and foreach Loops

The way that you used the while loops in the preceding section is very common. You set a counter to begin with. Before each iteration, you test the counter in a condition. And at the end of each iteration, you modify the counter.

You can write this style of loop in a more compact form by using a for loop. The basic structure of a for loop is

```plaintext
for( expression1; condition; expression2)
    expression3;
```

- `expression1` is executed once at the start. Here, you usually set the initial value of a counter.
- The `condition` expression is tested before each iteration. If the expression returns false, iteration stops. Here, you usually test the counter against a limit.
- `expression2` is executed at the end of each iteration. Here, you usually adjust the value of the counter.
- `expression3` is executed once per iteration. This expression is usually a block of code and contains the bulk of the loop code.

You can rewrite the while loop example in Listing 1.3 as a for loop. In this case, the PHP code becomes

```php
<?php
for ($distance = 50; $distance <= 250; $distance += 50) {
    echo "<tr>
    <td style="text-align: right;">".$distance."</td>
    <td style="text-align: right;">".(\$distance / 10)."</td>
    </tr>
};
?>
```

Both the while and for versions are functionally identical. The for loop is somewhat more compact, saving two lines.

Both these loop types are equivalent; neither is better or worse than the other. In a given situation, you can use whichever you find more intuitive.

As a side note, you can combine variable variables with a for loop to iterate through a series of repetitive form fields. If, for example, you have form fields with names such as `name1`, `name2`, `name3`, and so on, you can process them like this:

```php
for ($i=1; $i <= $numnames; $i++){
    $temp= "name$i";
    echo htmlspecialchars($temp).'<br />'; // or whatever processing you want to do
}
```

By dynamically creating the names of the variables, you can access each of the fields in turn.
As well as the for loop, there is a foreach loop, designed specifically for use with arrays. We discuss how to use it in Chapter 3.

**do...while Loops**

The final loop type we describe behaves slightly differently. The general structure of a do...while statement is

```php
do
    expression;
while(condition);
```

A do...while loop differs from a while loop because the condition is tested at the end. This means that in a do...while loop, the statement or block within the loop is always executed at least once.

Even if you consider this example in which the condition will be false at the start and can never become true, the loop will be executed once before checking the condition and ending:

```php
$num = 100;
do{
    echo $num."<br />
}while ($num < 1);
```

**Breaking Out of a Control Structure or Script**

If you want to stop executing a piece of code, you can choose from three approaches, depending on the effect you are trying to achieve.

If you want to stop executing a loop, you can use the break statement as previously discussed in the section on switch. If you use the break statement in a loop, execution of the script will continue at the next line of the script after the loop.

If you want to jump to the next loop iteration, you can instead use the continue statement.

If you want to finish executing the entire PHP script, you can use `exit`. This approach is typically useful when you are performing error checking. For example, you could modify the earlier example as follows:

```php
if($totalQty == 0){
    echo "You did not order anything on the previous page!<br />
    exit;
}
```

The call to `exit` stops PHP from executing the remainder of the script.
Employing Alternative Control Structure Syntax

For all the control structures we have looked at, there is an alternative form of syntax. It consists of replacing the opening brace ({}) with a colon (:) and the closing brace with a new keyword, which will be `endif`, `endswitch`, `endwhile`, `endfor`, or `endforeach`, depending on which control structure is being used. No alternative syntax is available for `do...while` loops.

For example, the code

```php
if ($totalqty == 0) {
    echo "You did not order anything on the previous page!<br />
    exit;
}
```

could be converted to this alternative syntax using the keywords `if` and `endif`:

```php
if ($totalqty == 0) :
    echo "You did not order anything on the previous page!<br />
    exit;
endif;
```

Using `declare`

One other control structure in PHP, the `declare` structure, is not used as frequently in day-to-day coding as the other constructs. The general form of this control structure is as follows:

```php
declare (directive)
{
    // block
}
```

This structure is used to set `execution directives` for the block of code—that is, rules about how the following code is to be run. Currently, only two execution directives, `ticks` and `encoding`, have been implemented.

You use ticks by inserting the directive `ticks=n`. It allows you to run a specific function every `n` lines of code inside the code block, which is principally useful for profiling and debugging.

The encoding directive is used to set encoding for a particular script, as follows:

```php
declare(encoding='UTF-8');
```

In this case, the `declare` statement may not be followed by a code block if you are using namespaces. We’ll talk about namespaces more later.
The `declare` control structure is mentioned here only for completeness. We consider some examples showing how to use `tick` functions in Chapters 25, “Using PHP and MySQL for Large Projects,” and 26, “Debugging and Logging.”

**Next**

Now you know how to receive and manipulate the customer's order. In the next chapter, you'll learn how to store the order so that it can be retrieved and fulfilled later.
Symbols

[] (array element operator), 35
-- (decrement operator), 30–31
== (equal operator), 31–32
$_POST$ array, 20
$.ajax() method, 508–509
$.get() method, 510
$.getJSON() method, 510
$.getScript() method, 510
$.post() method, 510
$this$ pointer, 164
\ (backslash), escape sequences, 125–126
^ (caret symbol), 121
, (comma operator), 33
@ (error suppression operator), 34
` ` (execution operator), 34–35
/ (forward slash), 56, 120
% (percent) symbol, printing, 110
& (reference operator), 31
; (semicolon), 16, 222–223
() (parentheses), order of precedence, 37–38
?: (ternary operator), 34
| (vertical pipe), 123
A

absolute path, 56
abstract classes, 188
access control implementing, 366–369
access modifiers, 165, 166
  visibility, controlling, 169–170
accessing
  array contents, 77–79
  array elements, 79
    with each() construct, 80–81
    with foreach loop, 80
  form variables, 20–22
    assignment operators, 20
    htmlspecialchars() function, 21–22
PHP, 12
accessor functions, 166–168, 178
ACID (atomicity, consistency, isolation, and durability), 317–318
add_bms.php, 588–589
addClass() method, 498
adding
  dynamic content, 18–19
  locks to files, 71–73
addition operator, 28
address field (Bob’s Auto Parts order form), 54
administrator privileges (MySQL), 229
advantages of reusing code
  consistency, 132
  cost, 132
  reliability, 132
aggregating SQL data, 259–261
AJAX (Asynchronous JavaScript and XML), 493–494
  $.ajax() method, 508–509
  asynchronous requests, 493
  helper methods, 509–510
  $.get(), 510
  $.getscript(), 510
  $.post(), 510
real-time chat application, building
  chat server, 504–507
aliases
  for namespaces, 198
  for tables, 257–258
ALTER TABLE command (SQL), 265–268
altering
  error reporting settings, 554–556
  tables after creation, 265–268
alternative control structure syntax, 51
anchoring regular expressions to beginning or end of string, 123
anonymous functions, 155–157
Apache
  HTTP Server
    .htaccess files, 374–377
    configuring, 356
  installing
    on UNIX, 600–602
    on Windows and Mac, 612–613
applying
  functions to array elements, 97–98
  localization to web pages, 440–445
    language selector page, 442–444
  software engineering to web development, 530
  templates to web pages, 134–139
  text to buttons, 461–464
arbitrary lengths, reading, 69
ARCHIVE table type, 316
arguments, 39
arithmetic operators, 28–29
array elements, 76
  accessing, 79
with each() construct, 80–81
with foreach loop, 80
applying functions to, 97–98
counting, 98–99
indices, 76
array key-value pairs for getdate() function, 427–428
array operators, 35, 81–82
array_count() function, 98–99
array_multisort() function, 87–88
array_pop() function, 92
array_push() function, 92
array_reverse() function, 92
array_walk() function, 97–98
arrays, 24, 75–76
$_POST, 20
accessing contents, 77–78, 78–79
bounding box contents, 463
converting to scalar variables, 99–100
initializing, 79
loading from files, 92–96
multidimensional arrays, 75, 82–85
sorting, 87–90
three-dimensional arrays, 84–85
two-dimensional arrays, 82–84
navigating, 96–97
numerically indexed arrays, 76–77
reordering, 90–91
with shuffle() function, 90–91
reversing, 92
sorting, 85–87
with asort() function, 86–87
with ksort() function, 86–87
reverse sorting, 83
with sort() function, 85–86
superglobal, 20, 27
asort() function, 86–87
assertions, 126–127
assigning values to variables, 24
assignment operators, 20
combined assignment operators, 30
values returned from, 29
associativity, 37–38
asynchronous requests, 493
atomic column values, 216–217
attackers, 339
attributes, 160, 162, 164–165, 177
access modifiers, 165, 166
accessor functions, 166–168
overriding, 170–172
preventing, 172
authentication, 333
access control, 366–369
basic authentication, 372–377
in PHP, 372–373
custom authentication, creating, 377
identifying visitors, 365–366
passwords
hash functions, 370–371
storing, 369
PHPbookmark project, 569–587
changing passwords, 580–582
logging in, 576–579
logging out, 580
registering users, 569–575
resetting forgotten passwords, 582–587
in session control, 483–491
authmain.php, 483–489
logout.php, 490–491
members_only.php, 489
authmain.php, 483–489
auto_append_file directive, 139–140
_autoload() function, 189
AUTO_INCREMENT keyword (MySQL), 234
auto_prepend_file directive, 139–140
autocommit mode (MySQL), 318
automatically generated images, 456
available extensions, identifying, 522–523
avoiding FTP timeouts, 420

backing up
files, 412–420
MySQL databases, 310–311
backreferences, 126
backtraces, 202
balancing security with usability, 342
bar chart, drawing, 465–474
basename() function, 397
basic authentication, 372–377
.inaccess files, 374–377
in PHP, 372–373
basic values, filtering, 346–347
basic_auth.php, 372–373
Bill Gates Wealth Clock, 407
bitwise operators, 33
blank canvas, creating, 452–453
BLOBs (binary large objects), 244
blocks, declaring, 42
Bob's Auto Parts site
exception handling, 204–208
order form
address field, 54
creating, 12–14
fields, naming, 14
processing, 14
totals, calculating, 36–37
Smart Form Mail application, creating, 101–104
bookmarks (PHPbookmark project), 561
adding, 588–590
deleting, 591–594
displaying, 590–591
Book-O-Rama bookstore application, 213–214
inserting information into database, 282–285
results.php, 273–275
schema, 221
search form, 272–273
Boolean values, 24
bottom-up approach to security, 343
bounding box, 462–463
branching, 123
breaking up code, 535–536
browsedir2.php, 392
browsedir.php, 390
browsers
cookies, 476, 477
session ID, storing, 477–478
setting from PHP, 476–477
outputting images to, 455
session control, 475
authentication, 483–491
configuring, 482–483
sessions
creating, 480–482
registering variables, 478–479
starting, 478
browsing php.ini file, 355–356
Bubbler, 510
built-in functions, 144
buttons
applying text, 461–464
classes, 619

creating, 457–465
  base canvas, setting up, 460–461
outputting to browser, 465
positioning text on, 464
text, writing on, 464–465

calculating
dates
  in MySQL, 434–435
  in PHP, 433–434
totals on order forms, 36–37
calendar functions, 436
  _call method, 188–189
callable type, 24
calling
  class operations, 165
  functions, 19, 141–142
    recursive functions, 154–155
    undefined functions, 142–143
canvas images
  creating, 452–453
  printing text on, 453–454
Cartesian product, 254–255
case of strings, changing, 111–112
case sensitivity, of identifiers, 239
catch blocks, 200
CHAR type columns, 235
character class, 121–122
character sets, 120–121, 438–440
  multi-byte, 438
  security implications, 439–440
  single-byte, 438
characters. See also special characters, 123–124
  reading, 69
charts, drawing from stored MySQL data, 465–474

chat application
  chat server, building, 504–507
  user interface, building, 510–517
chat.php, 504–507
checkdate() function, 428–429
checking
  for existence of files, 70
  length of strings, 115–116
choosing
  development environment, 537–538
  file mode, 55
  keys, 217
chop() function, 104
classes, 161
  $this pointer, 164
  abstract classes, 188
  attributes, 162, 164–165, 177
  converting to strings, 194
  designing, 176–177
  Exception class, 201–202
  inheritance, 161–162, 168–169
    late static bindings, 186–187
    multiple inheritance, 172–173
    preventing, 172
  instantiating, 163–164
  namespaces, 195–197
    global namespaces, 197–198
    importing, 198
    subnamespaces, 197
  naming, 177
  ObjectIterator, 192
  operations, 162–163
    calling, 165
  polymorphism, 161
  structure of, 162–163
  traits, 174–176
  writing code for, 177–184
accessor functions, 178
metatags, 177
click event, 500
Clifford, John, 510
cloning objects, 187–188
closedir() function, 391
closing files, 63–65
closures, 155–157
code
breaking up, 535–536
checking out, 537
for classes, writing, 177–184
operations, 181
commenting, 534
debugging, 352–353
indenting, 42, 534–535
maintainability, 532
optimizing, 540–541
organizing, 350–351
reusing, 133–134
advantages of, 131–132
functions, 140–157
in large web projects, 531–532
require() statement, 134–139
traits, 174–176
securing, 343
command execution, 353–354
escaping output, 348–350
filtering input data, 343–348
source code, highlighting, 525–526
standards, 532
defining naming conventions, 532–534
testing, 541–542
code blocks, 42
columns, 211, 235–237
atomic column values, 216–217
data types, 240–246
date and time types, 243–244
numeric types, 241–242
string types, 244–246
displaying, 302
indexes, creating, 238
MySQL
CHAR type, 235
VARCHAR type, 235–236
primary key, 211
columns_priv table, 296–298
combined assignment operators, 30
command line
executing scripts on, 526–527
running PHP on, 526–527
commands
executing, 353–354
MySQL
CREATE INDEX, 238
CREATE TABLE, 232–233
CREATE USER, 226
DESCRIBE, 304
EXPLAIN, 304–309
GRANT, 226–227, 230–231
REVOKE, 230, 230–231
SHOW, 301–304
show tables, 237
use, 232
mysql, 223
SQL
ALTER TABLE, 265–268
DELETE, 268
INSERT, 248–249
ORDER BY clause, 259
SELECT, 250–251, 252–253
UPDATE, 265
CREATE TABLE command

comments, 17–18
comparing
  conditionals, 45–46
  constants and variables, 26
  SQL and MySQL, 248
strings, 115
comparison operators, 31–32
  equal operator, 31–32
  for WHERE clause, 252–253
concatenating strings, 22
conditionals, 41
  code blocks, 42
  comparing, 45–46
  else statements, 42–43
  elseif statements, 43–44
  if statements, 41–42
  switch statement, 44–45
configuring
  Apache HTTP Server, 356
  MySQL users, 225–232
  PHP image support, 449–450
  session control, 482–483
    authentication, 483–491
connecting
  to MySQL, 277–278
  to network services, interaction failures, 548–549
  ODBC, 286
constants, 26
  error reporting levels, 553–554
  per-class constants, 185
  and variables, 26
constructors, 163
consuming data from other websites, 404–408
control structures
  alternative syntax, 51
  conditionals, 41
    code blocks, 42
    comparing, 45–46
    else statements, 42–43
    elseif statements, 43–44
    if statements, 41–42
    switch statement, 44–45
    declare structure, 51–52
    repetition structures, 46–50
      do.while loops, 50
      foreach loops, 49–50
      for loops, 49–50
      while loops, 47–48
      stopping, 50
      for stored procedures, 323–327
        declare handlers, 325
controlling visibility, 169–170
conversion specification, 109
  type codes, 110–111
converting
  arrays to scalar variables, 99–100
  classes to strings, 194
  dates and times to Unix timestamp, 426
  Gregorian to Julian calendar, 436
  between PHP and MySQL date formats, 431–433
cookies, 476, 477
  session ID, 476
  setting from PHP, 476–477
correlated subqueries, 264
count() function, 93, 98–99
counted subexpressions, 123
counting array elements, 98–99
crackers, 339
CREATE INDEX command, 238
CREATE TABLE command, 232–233
CREATE USER command, 226
creating
Bob’s Auto Parts order form, 12–14
buttons
base canvas, setting up, 460–461
outputting to browser, 465
text, applying, 461–464
text, positioning, 464
column indexes, 238
directories, 394
files, 398
HTML elements, 497–498
images, 451–455
make_button.php, 458–460
MySQL tables, 232–234
MySQL users, 224
sessions, 480–482
cross joins, 258
crypt() function, 370
CSV table type, 316
current() function, 96–97
cursors, 323, 325
custom authentication, creating, 377
customer feedback form (Bob’s Auto
Parts site), creating, 101–104
customer order form
address field, 54
creating, 12–14
fields, naming, 14
processing, 14
totals, calculating, 36–37

data hiding, 160
data storage, RDBMSs, 74
data types, 24–25
for MySQL columns, 240–246
date and time types, 243–244
numeric types, 241–242
string types, 244–246
scalar values, 26
type casting, 25
type strength, 25
databases. See also RDBMSs (relational
database management systems)
advantages of, 209
designing, 213–220
dropping, 268–269
MySQL, 209
backing up, 310–311
chat server, building, 504–507
DATE_FORMAT() function, 431–432
dates, calculating, 434–435
displaying, 302
inserting data, 282–285
interaction failures, 547–548
restoring, 311
security, 299–301
UNIX_TIMESTAMP() function, 432–433
users, setting up, 225–232
null values, 217–218
ODBC, 286
optimizing, 309–310
design optimization, 309
table optimization, 310
PHPbookmark project, implementing,
565–566
querying, 278
RDBMSs, 74
replication, 311–313
initial data transfer, performing,
313
master, setting up, 312–313
slaves, setting up, 313
schemas, 212
security, 357–359
transactions, 317–319
update anomalies, 215
web database architecture, 218–220, 272
Date, C.J., 220
date and time type columns, 243–244
date() function, 18, 19–20, 424–427
format codes, 424–425
Unix timestamps, 426–427
DATE_FORMAT() function, 431–432
dates
calculating
  in MySQL, 434–435
  in PHP, 433–434
calendar functions, 436
converting between PHP and MySQL formats, 431–433
Gregorian dates, 436
Julian dates, 436
validating with checkdate() function, 428–429
db table, 295–296
DDL (Data Definition Language), 248
debugging, 352–353
  variables, 551–553
declare handlers, 325
declare structure, 51–52
declaring
  blocks, 42
  constants, 26
  functions, 144
decrement operators, 30–31
define() function, 26
defining naming conventions for large projects, 532–534
DELETE command (SQL), 268
delete_bms.php, 592–593
deleting
  bookmarks, 591–594
  files, 70, 398
  records from database, 268
deletion anomalies, 215
delimiters, 120
denial of service, 335–337, 361
descenders, 463
DESCRIBE command, 304
designing
  classes, 176–177
  RDBMSs, 213–220
destroying
  image identifiers, 455
  sessions, 479
destructors, 163
die() function, 520–522
directories
  creating, 394
  reading from, 390–393
  retrieving information, 394
  submission form, 408
directory structure for large projects, 536
directory_submit.php, 409–412
disaster planning, 362–364
disconnecting from MySQL database, 281
disgruntled employees, threats posed by, 339
displaying
  bookmarks, 590–591
  columns, 302
  databases, 302
  MySQL privileges, 302
  tables, 237
division operator, 28
DML (Data Manipulation Language), 248
DMZs (demilitarized zones), 360–361
documentation
function libraries, 536
PHP manual, 531
project documentation, 538
dot notation, 255
double-quoted strings, interpolation, 22
do.while loops, 50
drawing bar charts, 465–474
dropping
databases, 268–269
tables, 268
DSN (data source name), 288
dump_array() function, 552–553
dump_variables.php, 551–553
dynamic content, adding, 18–19
each() construct, accessing array contents, 80–81
each() function, 80
echo statement, 22
else statements, 42–43
elseif statements, 43–44
email, sending and reading, 404
embedding PHP in HTML, 14–19
comments, 17–18
statements, 16
tags, 16
whitespace, 17
empty() function, 40
encapsulation, 160
end() function, 96–97
environment variables, 401–402
equal operator, 31–32
equi-joins, 258
error handling, 208
error reporting levels, 553–554
logging errors, 560
graceful error logging, 557–559
logic errors, 549–551
opening files, 58–61
programming errors, 543–551
runtime errors, 544–549
causes of, 545–549
syntax errors, 543–544
triggering your own errors, 556
error messages for undefined functions, 142–143
error reporting levels, 553–554
error reporting settings, altering, 554–556
error suppression operator, 34, 60
escape sequences, 125–126
escapeshellcmd() function, 354
escaping
from HTML, 16
output, 348–350
eval() function, 519–520
evaluating
SELECT queries, 304–309
strings, 519–520
event handling
jQuery, 499–504
click event, 500
focusout event, 503
on() method, 499–500
ready event, 499
submit event, 504
triggers, 327–329
Exception class, 201–202
exception handling, 199–201, 557
in Bob’s Auto Parts site, 204–208
catch blocks, 200
Exception class, 201–202
finally blocks, 200
throw keyword, 200
try blocks, 199
user-defined exceptions, 202–204
executing commands, 353–354
execution directives, 51–52
execution operator, 34–35
existence of files, checking for, 70
exit() function, 520–522
EXPLAIN command, 304–309
explode() function, 95–96
splitting strings with, 112–113
extensions
loaded extensions, identifying, 522–523
PDO data access abstraction extension, 286–289
php_gd2.dll extension, registering, 450
extract() function, 99–100
fclose() function, 63–65
feedback form (Bob's Auto Parts site), creating, 101–104
feof() function, 66–67
fgetc() function, 69
fgetcsv() function, 67–68
fgets() function, 67–68
fgetts() function, 67–68
fields, naming, 14
file formats, 62–63
file() function, 68–69, 93
file mode, 55
choosing, 55
fopen() function, 57
file systems
absolute path, 56
file information, retrieving, 395–397
relative path, 56
security, 352
file_exists() function, 70
file_get_contents() function, 68–69
file_put_contents() function, 61
filetime() function, 397
filedetails.php, 395–396
fileowner() function, 397
fileperms() function, 397
files
.htaccess files, 374–377
backing up, 412–420
characters, reading, 69
closing, 63–65
creating, 398
deleting, 70, 398
existence of, checking for, 70
flat files, 53–54
problems with, 73
image files
creating, 451–455
GIFs, 451
JPEGs, 450
PNGs, 450–451
loading arrays from, 92–96
locking, 71–73
logging errors to, 560
moving, 398
navigating inside, 70–71
opening, 55
error handling, 58–61
with fopen() function, 56–58
through FTP or HTTP, 58
in PHPBookmark application, 564–565
processing, 55
properties, changing, 397–398
reading from, 55, 65–66, 67–68, 68–69
files

as cause for runtime errors, 546–547
line-by-line, 67–68
require() statement, 132–134
size of, determining, 70
uploading, 379–389, 420
   HTML form, 381–382
   php.ini settings, 380–381
   tracking upload progress, 387–388
   troubleshooting, 389
   writing the file handling script, 382–387
writing to, 55, 61
filesize() function, 70, 397
filtering
   input data, 276, 343–348
      basic values, 346–347
      double-checking expected values, 344–346
      strings, 347–348
   strings, 105–107
   for output to browser, 105–106
   for output to email, 106–107
final keyword, 172
finally blocks, 200
finding
   non-matching rows, 256–257
   strings within strings, 116–117
   substrings with regular expressions, 128–129
firewalls, 360
flat files, 53–54
   problems with, 73
float data type, 25
floating-point types, 242
floatval() function, 41
flock() function, 71–73
focusout event, 503
fonts, TrueType, 457
fopen() function, 55, 66
   file mode, 57
   opening files with, 56–58
   parameters, 56
foreach loops, 49–50, 190
   accessing array elements, 80
FOREIGN KEY keyword (MySQL), 235
foreign keys, 212, 319
   Book-O-Rama bookstore application, 221
forgot_passwd.php, 583–584
format codes, date() function, 424–425
formatting
   strings
      changing case of, 111–112
      conversion specification, 109
      for printing, 109–111
      timestamps, 429–431
forms
   Book-O-Rama bookstore application
      HTML form, 282–285
      search form, 272–273
   customer order form
      creating, 12–14
      fields, naming, 14
      processing, 14
   Smart Form Mail application
      creating, 101–104
      regular expressions, 127–128
      submission form, 408
      variables, accessing, 20–22
fpassthru() function, 68–69
fputs() function, 61
fread() function, 69
front end interface, building for chat application, 504–507
fseek() function, 70–71
ftell() function, 70–71
FTP
  avoiding timeouts, 420
  backing up files with, 412–420
  files, opening, 58
ftp_mirror.php, 413–416
ftp_nlist() function, 421
ftp_size() function, 420
full joins, 254–255
func_num_args() function, 148
functions, 140
  _autoload(), 189
  _get(), 166–168
  _set(), 166–168
  accessor functions, 166–168, 178
  aggregate functions (MySQL), 259–261
  applying to array elements, 97–98
  arguments, 39
  array_count(), 98–99
  array_multisort(), 87–88
  array_pop(), 92
  array_push(), 92
  array_reverse(), 92
  array_walk(), 97–98
  asort(), 86–87
  backtraces, 202
  basename(), 397
  built-in, 144
  calling, 19, 141–142
  case functions, 112
  case sensitivity, 143
  checkdate(), 428–429
  chop(), 104
  closedir(), 391
  closures, 155–157
  count(), 93, 98–99
  crypt(), 370
  current(), 96–97
  date(), 18, 19–20, 424–427
    format codes, 424–425
  DATE_FORMAT(), 431–432
  define(), 26
  die(), 520–522
  dump_array(), 552–553
  each(), 80
  empty(), 40
  end(), 96–97
  escapehellcmd(), 354
  eval(), 519–520
  exit(), 520–522
  explode(), 95–96
    splitting strings with, 112–113
  extract(), 99–100
  fclose(), 63–65
  feof(), 66–67
  fgetc(), 69
  fgets(), 67–68
  fgetsv(), 67–68
  file(), 68–69, 93
  file_exists(), 70
  file_get_contents(), 68–69
  file_put_contents(), 61
  filemtime(), 397
  fileowner(), 397
  fileperms(), 397
  filesize(), 70, 397
  floatval(), 41
  flock(), 71–73
  fopen(), 55, 66
    file mode, 57
    opening files with, 56–58
    parameters, 56
lookup functions, 408–412
ltrim(), 104
mail(), 104, 404
microtime(), 435
mkdir(), 394
mktime(), 426–427
multibyte string functions, 440
mysqli(), 547
namespaces, 195–197
global namespaces, 197–198
importing, 198
subnamespaces, 197
naming, 145–146
next(), 96–97
nl2br(), 70, 107–109
nonexistent, as cause for runtime errors, 545–546
number_format(), 37
in ObjectIterator class, 192
opendir(), 391
overloading, 145
parameters, 146–148
passing, 141
passing by reference, 150–151
passthru(), 399
phpinfo(), 26, 141
pollServer(), 515–516
pos(), 96–97
preg_match(), 128–129
preg_split(), 129–130
prev(), 96–97
printf(), 109–111
program execution, 398–401
prototype, 141–142
putenv(), 401–402
range(), 77
readdir(), 391
libraries, 536

readfile(), 68–69
recursive, 154–155
reset(), 96–97
return keyword, 152–153
returning values from, 153
rewind(), 70–71
rmdir(), 394
rsort(), 83
scope, 148–150
serialize(), 521
session_start(), 478
set_error_handler(), 557–558
setcookie(), 476
settype(), 39
show_source(), 525
shuffle(), 90–91
sizeof(), 98–99
sort(), 76, 85–86
sprintf(), 109
str_replace(), 107, 118–119
strcasecmp(), 115
strchr(), 117
strcmp(), 115
strftime(), 429–431
stristr(), 117
strnatcmp(), 115
strtok(), 113–114
strpos(), 117–118
strstr(), 116–117
strtolower(), 112
strtoupper(), 112
structure of, 144–145
strval(), 41
substr(), 114
system(), 399
trigger_error(), 556
trim(), 104
uasort(), 89
ucfirst(), 112
ucwords(), 112
uksort(), 89
umask(), 394
undefined functions, calling, 142–143
UNIX_TIMESTAMP(), 432–433
unlink(), 70
unserialize(), 521
urlencode(), 407
user-defined, 144
usort(), 88–89
variable functions, 146
variable handling functions, 39–40
vprintf(), 111
vsprintf(), 111
fwrite() function, 61
parameters, 62

GD2 image library, 449
generating
bar charts from stored MySQL data, 465–474
charts from stored MySQL data, 465–474
generators, 192–193
_get() function, 166–168
get_loaded_extensions() function, 523
getdate() function, 427–428
array key-value pairs, 427–428
getenv() function, 401–402
getlastmod() function, 524
gettext() function, 444–448, 446
gettype() function, 39
GIF (Graphics Interchange Format) files, 451
Git, 537
   global keyword, 150
   global namespaces, 197–198
GNU gettext
   installing, 444–445
   translation files, 445–447
general error logging, 557–559
GRANT command, 226–227, 230–231
grant tables, 291–299
   columns_priv table, 296–298
   connection verification, 298
   db table, 295–296
   procs_priv table, 296–298
   request verification, 298
   tables_priv table, 296–298
   user table, 293–295
Greenspun, Philip, 407
Gregorian dates, 436
   grouping SQL data, 259–261

handle.php, 558
handles, 161
hash functions, 370–371
header() function, 455
headers, 438–439
   locale-specific, 441–442
helper methods, 509–510
   $.getJSON(), 510
   $.getScript(), 510
   $.post(), 510
heredoc syntax, 23
highlight_string() function, 525
highlighting source code, 525–526
hosting providers, 599–600

HTML
   Book-O-Rama form, 282–285
   elements
   creating, 497–498
   selecting with jQuery selectors, 496–497
   escaping, 16
   file upload form, 381–382
   PHP, embedding, 14–19, 16
      comments, 17–18
      statements, 16
      whitespace, 17
   reusing, applying templates to web pages, 134–139
   submission form, 408
htmlspecialchars() function, 21–22, 105–106
HTTP files, opening, 58

identifiers, 23–24, 239–240
   case sensitivity, 239
   rules, 239
identifying script owner, 523
IETF (Internet Engineering Task Force), 404
if statements, 41–42
image identifiers, destroying, 455
imagecolorallocate() function, 453
imagecreatetruecolor() function, 452–453
imagecreatfrompng() function, 461
imagefill() function, 453–454
imagefilledrectangle() function, 472
imageline() function, 472
ImageMagick image library, 449
images
   automatically generated, 456
bar chart, drawing from stored SQL data, 465–474
buttons
  creating, 457–465
  outputting to browser, 465
  positioning text on, 464
text, applying, 461–464
writing text on, 464–465
canvas images
  creating, 452–453
  printing text on, 453–454
creating, 451–455
  make_button.php, 458–460
GIFs, 451
JPEGs, 450
libraries, 449
outputting to browser, 455
php_gd2.dll extension, registering, 450
PNGs, 450–451
simplegraph.php, 451–452
  support in PHP, configuring, 449–450
imagestring() function, 454
imagettftext() function, 462
IMAP4 (Internet Message Access protocol), 404
implode() function, 113
importing namespaces, 198
increment operators, 30–31
indenting code, 42
indexes, creating, 310
indices, 76
  numerically indexed arrays, 76–77
inheritance, 161–162, 168–169
  late static bindings, 186–187
  multiple inheritance, 172–173
  overriding, 170–172
  preventing, 172
ini_get() function, 524–525
ini_set() function, 524
initializing arrays, 79
  numerically indexed arrays, 76–77
inner joins, 258
InnoDB table type, 316
  transactions, 318–319
input data, filtering, 343–348
  basic values, 346–347
  double-checking expected values, 344–346
  strings, 347–348
INSERT command (SQL), 248–249
inserting data into SQL database, 248–250, 282–285
insertion anomalies, 215
installing
  Apache
    on UNIX, 600–602
    on Windows and Mac, 612–613
  GNU gettext, 444–445
  MySQL on UNIX, 602–605
  PEAR, 613–614
  PHP
    with other web servers, 614
    on UNIX, 605–609
    on Windows and Mac, 612–613
instanceof operator, 35, 185–186
instantiating classes, 163–164
integers, 25
integral data types, 241
interacting with the environment, 401–402
interfaces, 173–174
  Book-O-Rama HTML form, 282–285
  Iterator, 190–191
  PDO data access abstraction extension, 286–289
internationalization, 437–438
  applying to web pages, 440–445
  language selector page, 442–444
  locale-specific headers, 441–442
  gettext() function, 444–448
  GNU gettext, installing, 444–445
  translation files, 445–447
interpolation, 22
intval() function, 41
isset() function, 40, 152
iteration, 46–50, 190–192
  accessing array contents, 78–79
  do.while loops, 50
  foreach loops, 49–50
  for loops, 49–50
  while loops, 47–48
Iterator interface, 190–191

J
JavaScript. See also AJAX; jQuery
  AJAX, 493–494
join() function, 113
joining strings, 113
joins
  cross joins, 258
  equi-joins, 258
  full joins, 254–255
  inner joins, 258
  joining more than two tables, 255–256
  left joins, 256–257
JPEG (Joint Photographic Experts Group) files, 450
jQuery, 494–504
  $.ajax() method, 508–509
  addClass() method, 498
  AJAX helper methods, 509–510
  $.getJSON(), 510
  $.getscript(), 510
  $.post(), 510
  events, 499–504
  click event, 500
  focusout, 503
  on() method, 499–500
  ready event, 499
  submit, 504
  namespace, 495
  pseudo-selectors, 497
  selectors, 495–498
  acting on, 498
  syntax, 496–497
  selectors (jQuery), creating HTML elements, 497–498
  val() method, 498
  in web applications, 494–495
Julian dates, 436

K
keys, 76, 211–212
  Book-O-Rama bookstore application, 221
  choosing, 217
  foreign keys, 212, 319
  success, 507
keywords
  clone, 187–188
  final, 172
  global, 150
MySQL
  AUTO_INCREMENT, 234
  FOREIGN KEY, 235
  NOT NULL, 234
  PRIMARY KEY, 234–235
  return, 152–153
static, 185
throw, 200
trait, 174–176
yield, 192–193
krsort() function, 83
ksort() function, 86–87

languages
headers, 438–439
multi-byte, 438
single-byte, 438

large web application projects, 529
choosing a development environment, 537–538
coding standards, 532
breaking up code, 535–536
commenting your code, 534
defining naming conventions, 532–534
indenting, 534–535
directory structure, 536
documenting, 538
function libraries, 536
optimizing code, 540–541
prototyping, 538–539
reusing code, 531–532
separating logic from content, 539–540
testing code, 541–542
version control, 536–537
writing maintainable code, 532

late static bindings, 186–187
left joins, 256–257

length of strings, checking, 115–116

libraries
function libraries, 536
image libraries, 449
jQuery library, loading, 494–495

LIMIT clause (SELECT command), 261–262
line-by-line reading from files, 67–68
linking tables, 218
list() construct, 81
list_functions.php, 522–523

loading
arrays from files, 92–96
files with require() statement, 132–134
jQuery library, 494–495

local variables, 438

localization, 437–438
applying to web pages, 440–445
language selector page, 442–444
character sets, 438–440
multi-byte, 438
security implications, 439–440
single-byte, 438
gettext() function, 444–448
GNU gettext, installing, 444–445
translation files, 445–447
headers, 438–439
locale-specific, 441–442
locales, 438
multibyte string functions, 440

locking files, 71–73

logging errors
graceful error logging, 557–559
to log file, 560

logging in to MySQL, 223–224
logic, separating from content, 539–540
logic errors, 549–551
logical operators, 32–33

login.php, 566–567
logout.php, 490–491
lookup functions, 408–412
lookup.php, 405
for loops, 49–50
loops
  accessing array contents, 78–79
  do-while loops, 50
  foreach loops, 49–50, 190
  for loops, 49–50
  while loops, 47–48
ltrim() function, 104

M
Mac OS, installation packages, 612–613
mail() function, 104, 404
maintainability of code, 532
make_button.php, 458–460
many-to-many relationships, 213
master, setting up for replication, 312–313
matching
  special characters, 123–124
  substrings with string functions, 116
max_execution_time directive, 524
member.php, 576–577
members_only.php, 489
MEMORY table type, 316
Mercurial, 537
MERGE table type, 316
meta characters, 124–125
metatags, 177
on() method, 499–500
methods
  $.ajax(), 508–509
  AJAX helper methods, 509–510
  $.get(), 510
  $.getJSON(), 510
  $.getScript(), 510
  $.post(), 510
in Exception class, 201–202
jQuery
  on(), 499–500
  addClass(), 498
  val(), 498
overloading, 188–189
static, 185
microseconds, 435
microtime() function, 435
mirroring files, 412–420
mkdir() function, 394
mktime() function, 426–427
modification date of scripts, obtaining, 523–524
modulus operator, 28
monitoring security, 342–343
moving files, 398
multibyte string functions, 440
multidimensional arrays, 75, 82–85
  sorting, 87–90
    with array_multisort() function, 87–88
    reverse sorting, 89–90
    user-defined sorts, 88–89
  three-dimensional arrays, 84–85
  two-dimensional arrays, 82–84
multiline comments, 17
multiple inheritance, 172–173
multiplication operator, 28
MySQL, 209, 221–222. See also MySQL
monitor
  aggregating data, 259–261
  autocommit mode, 318
chat server, building, 504–507

columns
data types, 240–246
date and time types, 243–244
indexes, creating, 238
numeric types, 241–242
string types, 244–246

commands
AUTO_INCREMENT keyword, 234
CREATE USER, 226
DESCRIBE, 304
EXPLAIN, 304–309
FOREIGN KEY keyword, 235
GRANT, 226–227, 230–231
mysql, 223
NOT NULL keyword, 234
PRIMARY KEY keyword, 234–235
REVOKE, 230–231
SHOW, 301–304
SHOW command, 303–304
databases
backing up, 310–311
creating, 224
restoring, 311
selecting, 232
date format, converting to PHP, 431–433
DATE_FORMAT() function, 431–432
dates, calculating, 434–435
drawing charts from stored data, 465–474
identifiers, 239–240
case sensitivity, 239
rules, 239
installing
on UNIX, 602–605
on Windows and Mac, 612–613
joins
cross joins, 258
equi-joins, 258
full joins, 254–255
inner joins, 258
joining more than two tables, 255–256
left joins, 256–257
logging in, 223–224
optimizing databases, 309–310
design optimization, 309
table optimization, 310
privileges, 291–299
columns_priv table, 296–298
db table, 295–296
displaying, 302
procs_priv table, 296–298
tables_priv table, 296–298
updating, 299
user table, 293–295
querying from the Web, 275–281
disconnecting from database, 281
filtering input data, 276
prepared statements, 279–280
retrieving the results, 280–281
selecting the database, 278
setting up connection, 277–278
runtime errors, 547–548
security, 299–301
passwords, 300
web issues, 301
stored procedures, 320–327
control structures, 323–327
cursors, 323, 325
declare handlers, 325
descriptor of, 320–323
local variables, 323
tables
  aliases, 257–258
  creating, 232–234
  dropping, 268
  viewing, 237–238
UNIX_TIMESTAMP() function, 432–433
user privileges, 300–301
users, 225–232
  creating, 224
  principle of least privilege, 225
  privileges, 225–231, 227–230
  web access, 231–232
mysq1 command, 223
MySQL monitor, 222–223
mysqli() function, 547
mysqli library, 277
  prepared statements, 279–280

N
namespaces, 195–197
  aliasing, 198
  global namespaces, 197–198
  importing, 198
  jQuery, 495
  subnamespaces, 197
naming
  classes, 177
  fields, 14
  functions, 145–146
  tables, 257–258
navigating
  within arrays, 96–97
  inside files, 70–71
network security, 360–361
  denial of service attacks, 361
  DMZ, 360–361
  firewalls, 360
  network services, interaction failures, 548–549
next() function, 96–97
Nginx servers, 614
nl2br() function, 70, 107–109
nonexistent functions, as cause for runtime errors, 545–546
non-matching rows, finding, 256–257
NOT NULL keyword (MySQL), 234
NOT operator, 32–33
NULL type, 24
null values, 217–218
number_format() function, 37
numeric type columns, 241–242
  floating-point types, 242
  integral data types, 241
numerically indexed arrays, 76–77

O
ObjectIterator class, 192
objects, 24, 160, 161
  classes, 161
  cloning, 187–188
  instantiating a class, 163–164
  interfaces, 160, 173–174
  serializing, 521
ODBC (Open Database Connectivity), 286
one-to-many relationships, 213
one-to-one relationships, 213
one-way hash functions, 370
OO (object-oriented) development, 159
  _autoload() function, 189
  accessor functions, 166–168
  attributes, 160
    overriding, 170–172
classes, 161
  abstract classes, 188
attributes, 162, 164–165, 177
constructors, 163
converting to strings, 194
designing, 176–177
destructors, 163
Exception class, 201–202
instantiating, 163–164
ObjectIterator, 192
operations, 162–163
structure of, 162–163
writing code for, 177–184
encapsulation, 160
generators, 192–193
inheritance, 161–162, 168–169
  multiple inheritance, 172–173
  preventing, 172
instanceof operator, 185–186
interfaces, 173–174
  Iterator, 190–191
iteration, 190–192
late static bindings, 186–187
namespaces, 195–197
  global namespaces, 197–198
  importing, 198
  subnamespaces, 197
objects, 160, 161
  cloning, 187–188
  serializing, 521
operations, 160
  calling, 165
per-class constants, 185
polymorphism, 161
reflection API, 194–195
static methods, 185
traits, 174–176
type hinting, 185–186
opendir() function, 391
opening files, 55
  error handling, 58–61
  with fopen() function, 56–58
  through FTP or HTTP, 58
operands, 28
operating system, securing, 361–362
operations, 160, 162–163, 181
  calling, 165
  constructors, 163
  destructors, 163
  overriding, 170–172
  preventing, 172
AND operator, 32–33
OR operator, 32–33
operators, 28
  arithmetic operators, 28–29
  array operators, 35, 81–82
  assignment operators, 20, 29–31
    combined assignment operators, 30
    values returned from, 29
  associativity, 37–38
  bitwise operators, 33
  comparison operators, 31–32
    equal operator, 31–32
  decrement operators, 30–31
  error suppression operator, 34, 60
  execution operator, 34–35
  increment operators, 30–31
  instanceof, 185–186
  logical operators, 32–33
  precedence, 37–38
  reference operator, 31
  string concatenation operator, 22
  string operators, 29
  for subqueries, 263
  ternary operator, 34
  type operator, 35
optimizing

code, 540–541
databases, 309–310
design optimization, 309
table optimization, 310

options for session configuration, 482–483

ORDER BY clause, 259

order forms

address field, 54
creating, 12–14
fields, naming, 14
processing, 14
storing and retrieving orders, 54
strings, 115
totals, calculating, 36–37

organizing code, 350–351

outputting

buttons to browser, 465
images, 455

overloading methods, 188–189

overriding, 170–172

preventing, 172

owner of scripts, identifying, 523

parameters, 146–148

extract() function, 100
fopen() function, 56
fwrite() function, 62
htmlspecialchars() function, 105–106

passing, 141

parser errors, 543–544

passing by reference, 150–151

passing by value, 150–151

passing parameters, 141

passthru() function, 399

passwords, 369–371

hash functions, 370–371

MySQL, 300

storing, 369

pattern matching, delimiters, 120

PEAR (PHP Extension and Application Repository), installing, 613–614

per-class constants, 185

performance, optimizing databases
design optimization, 309
table optimization, 310

permissions, 59

PHP

accessing, 12

basic authentication, 372–373
dates, calculating, 433–434

embedding in HTML, 14–19

comments, 17–18
tags, 16

whitespace, 17

English language manual, 531

environment information, obtaining, 522

installing

with other web servers, 614

on UNIX, 605–609

on Windows and Mac, 612–613

statements, 16
tags

short style, 16

XML style, 16

PHP interpreter, 600

php_gd2.dll extension, registering, 450

PHPbookmark project, 561

add_bms.php, 588–589

basic site, implementing, 566–569

bookmark_fns.php, 567–568
bookmarks
  adding, 588–590
  deleting, 591–594
  displaying, 590–591
database, implementing, 565–566
delete_bms.php, 592–593
files, 564–565
forgot_passwd.php, 583–584
implementing recommendations, 594–597
login.php, 566–567
member.php, 576–577
recommend.php, 595–597
register_form.php, 569–570
register_new.php, 570–572
solution components, 561–565
user authentication, 569–587
  changing passwords, 580–582
  logging in, 576–579
  logging out, 580
  registering users, 569–575
  resetting forgotten passwords, 582–587
phpinfo() function, 26, 141
php.ini file
  browsing, 355–356
  date.timezone setting, 424
  file upload settings, 380–381
  session upload progress configuration settings, 387
planning web application projects, 530–531
PNG (Portable Network Graphics) files, 450–451
PO (Portable Object) files, 445–446
Poedit, 446
pollServer() function, 515–516
polymorphism, 161
POP (Post Office Protocol), 404
pos() function, 96–97
position of substrings, identifying, 117–118
positioning text on buttons, 464
POSIX-style regular expressions, 119
precedence, 37–38
preg_match() function, 128–129
preg_split() function, 129–130
prepared statements, 279–280
Pressman, Roger, 542
prev() function, 96–97
preventing inheritance, 172
primary key, 211
PRIMARY KEY keyword (MySQL), 234–235
primary keys, Book-O-Rama bookstore application, 221
principle of least privilege, 225
printf() function, 109–111
printing
  echo statement, 22
  formatting strings for, 109–111
  percent symbol, 110
  text on canvas images, 453–454
private access modifier, 166
  visibility, controlling, 169–170
privileges (MySQL), 225–231, 227–230, 291–299, 300–301
  administrator privileges, 229
  columns_priv table, 296–298
  CREATE USER command, 226
db table, 295–296
  displaying, 302
  GRANT command, 226–227
  principle of least privilege, 225
  procs_priv table, 296–298
  revoking, 230
  special privileges, 230
tables_priv table, 296–298
updating, 299
user privileges, 228
user table, 293–295
processfeed_v2.php, 108–109
processing
customer order form, 14
files, 55
processorder.php, 14–19
creating, 14
dynamic content, adding, 18–19
with exception handling, 205–208
form variables, accessing, 20–22
functions, calling, 19
procs_priv table, 296–298
progex.php, 400–401
program execution functions, 398–401
programming errors, 543–551
logic errors, 549–551
runtime errors, 544–549
causes of, 545–549
syntax errors, 543–544
properties of files, changing, 397–398
protected access modifier, 166
protecting multiple web pages, 371
protocols, 403–404
prototype, 141–142
prototyping web applications, 538–539
pseudo-selectors, 497
public access modifier, 166
visibility, controlling, 169–170
putenv() function, 401–402
querying databases
SELECT queries, evaluating, 304–309
subqueries, 262–263
correlated subqueries, 264
operators, 263
row subqueries, 264
as temporary table, 264
from the Web, 275–281
disconnecting from database, 281
filtering input data, 276
prepared statements, 279–280
retrieving the results, 280–281
selecting the database, 278
setting up connection, 277–278
R
range() function, 77
RDBMSs (relational database management systems), 74
atomic column values, 216–217
columns, 211
design principles, 213–220
keys, 211–212
choosing, 217
MySQL
databases, creating, 224
databases, selecting, 232
logging in, 223–224
mysql command, 223
privileges, 225–231
tables, creating, 232–234
users, creating, 224
null values, 217–218
relationships, 213
rows, 211
schemas, 212
tables, 210, 218
update anomalies, 215
values, 211
readdir() function, 391
readfile() function, 68–69
reading
arbitrary lengths, 69
caracters, 69
e-mail, 404
from files, 55, 65–66, 67–68, 68–69
as cause for runtime errors, 546–547
line-by-line, 67–68
form directories, 390–393
ready event, 499
real-time chat application, chat server, building, 504–507
recommend.php, 595–597
records
deleting, 268
storing, 62
updating, 265
recursive functions, 154–155
reducing web application security risks
access to sensitive data, 332–333
denial of service, 336–337
loss of data, 334–335
malicious code injection, 337
reference operator, 31
reflection API, 194–195
register_form.php, 569–570
register_new.php, 570–572
registering
php gd2.dll extension, 450
session variables, 478–479
regular expressions, 119–130
anchoring to beginning or end of string, 123
assertions, 126–127
backreferences, 126
branching, 123
character class, 121–122
character sets, 120–121
counted subexpressions, 123
delimiters, 120
escape sequences, 125–126
meta characters, 124–125
POSIX, 119
repetition, 122
in Smart Form Mail application, 127–128
special characters, matching, 123–124
strings, splitting, 129–130
substrings, finding, 128–129
substrings, replacing, 129
relationships, 213
relative path, 56
reordering arrays, 90–91
with shuffle() function, 90–91
repetition in regular expressions, 122
repetition structures, 46–50
accessing array contents, 78–79
do.while loops, 50
foreach loops, 49–50
for loops, 49–50
while loops, 47–48
replacing substrings
with regular expressions, 129
with string functions, 116
replication, 311–313
initial data transfer, performing, 313
master, setting up, 312–313
slaves, setting up, 313
repudiation, 338–339
require() statement, 132–134
adding templates to web pages, 134–139
reset() function, 96–97
resource type, 24
restoring MySQL databases, 311
results.php, 273–275
querying from the Web, filtering input data, 276
retrieving data from SQL databases, 250–259
  criteria, specifying, 251–253
  joining more than two tables, 255–256
  from multiple tables, 253–258
    finding rows that don’t match, 256–257
    full joins, 254–255
  ORDER BY clause, 259
  SELECT command, 250–251
return keyword, 152–153
returning values from functions, 153
reusing code
  advantages of, 131–132
    consistency, 132
    cost, 132
    reliability, 132
  functions, 140
    built-in functions, 144
    calling, 141–142
    case sensitivity, 143
    closures, 155–157
    naming, 145–146
    parameters, 146–148
    parameters, passing, 141
    prototype, 141–142
    recursive functions, 154–155
    return keyword, 152–153
    returning values from, 153
    scope, 148–150
    structure of, 144–145
    undefined functions, calling, 142–143
    user-defined, 144
    variable functions, 146
  in large web projects, 531–532
  maintainability, 532
require() statement, 132–134
  applying templates to web pages, 134–139
  traits, 174–176
reverse sorting functions, 83, 89–90
reversing arrays, 92
REVOKE command, 230, 230–231
rewind() function, 70–71
RFCs (Requests for Comments), 404
rmdir() function, 394
row subqueries, 264
rows, 211
  inserting into SQL database, 248–250
  non-matching rows, finding, 256–257
rsort() function, 83
rules
  for identifiers, 239
  of variable scope, 27
running PHP on command line, 526–527
runtime environment, temporarily
  modifying, 524–525
runtime errors, 544–549
  causes of, 545–549
    calls to nonexistent functions, 545–546
    connections to network services, 548–549
    failure to check input data, 549
    interaction with MySQL, 547–548
    reading or writing files, 546–547
S
SaaS version control systems, 537
scalar values, 26
scalar variables, creating from arrays, 99–100
scandir.php, 393
schemas, 212
scope, 27, 148–150

<script> tag, 494–495

scripts

- add_bms.php, 588–589
- adding locks to, 71–73
- authmain.php, 483–489
- basic_auth.php, 372–373
- bookmark_fns.php, 567–568
- browsedir2.php, 392
- browsedir.php, 390
- chat.php, 504–507
- delete_bms.php, 592–593
- directory_submit.php, 409–412
- dump_variables.php, 551–553
- executing on command line, 526–527
- filedetails.php, 395–396
- forgot_passwd.php, 583–584
- ftp_mirror.php, 413–416
- functions, calling, 19
- handle.php, 558
- list_functions.php, 522–523
- login.php, 566–567
- logout.php, 490–491
- lookup.php, 405
- make_button.php, 458–460
- member.php, 576–577
- members_only.php, 489
- modification date, obtaining, 523–524
- owner, identifying, 523
- processfeedback_v2.php, 108–109
- processfeedback.php, 101–104
- processorder.php
  - creating, 14
  - dynamic content, adding, 18–19
  - with exception handling, 205–208
- progex.php, 400–401
- recommend.php, 595–597
- register_form.php, 569–570
- register_new.php, 570–572
- results.php, 273–275
- scandir.php, 393
- secret.php, 369
- show_poll.php, 468–474
- simplegraph.php, 451–452
- stopping, 50
- terminating, 520–522
- upload.php, 382–387
- vieworders.php, 65–66

search form (Book-O-Rama bookstore application), 272–273

secret.php, 367–369

security

application security threats
  - access to sensitive data, 331–333
  - actors, 339–340
  - compromised server, 338
  - denial of service, 335–337
  - loss of data, 334–335
  - malicious code injection, 337
  - modification of data, 334
  - repudiation, 338–339

attackers, 339

authentication
  - access control, 366–369
  - basic authentication, 372–377
  - custom authentication, creating, 377
  - passwords, 369–371
  - PHPBookmark project, 569–587
  - in session control, 483–491
  - visitors, identifying, 365–366

character sets, 439–440

code, securing, 343
  - bugs, 352–353
  - escaping output, 348–350
filtering user input, 343–348
organizing code, 350–351
 crackers, 339
database servers, securing, 357–359
disaster planning, 362–364
 file systems, 352
 MySQL, 299–301
 passwords, 300
 user privileges, 300–301
 web issues, 301
 networks, securing, 360–361
 denial of service attacks, 361
 DMZ, 360–361
 firewalls, 360
 operating system, securing, 361–362
 permissions, 59
 strategies for handling, 341–343
 balancing security and usability, 342
 monitoring, 342–343
 starting with the right mindset, 342
twofold approach to, 343
 web pages, protecting, 371
 web servers, securing, 354–357
 browsing php.ini file, 355–356
 shared hosting of web applications, 356–357
 updating software, 354–355
 **SELECT command (SQL), 250–251**
evaluating, 304–309
 LIMIT clause, 261–262
 ORDER BY clause, 259
 WHERE clause, 252–253
 comparison operators, 252–253
 selecting
 HTML elements with selectors, 496–497
 MySQL database, 232
 SQL databases from the web, 278
 table types, 316
 **selectors (jQuery), 495–498**
 acting on, 498
 HTML elements, creating, 497–498
 pseudo-selectors, 497
 syntax, 496–497
 **sending email, 404**
 **serialization, 521**
 **serialize() function, 521**
 **session control, 475**
 authentication, 483–491
 authmain.php, 483–489
 logout.php, 490–491
 members_only.php, 489
 configuring, 482–483
 cookies, 476, 477
 setting from PHP, 476–477
 session ID, storing, 477–478
 sessions
 creating, 480–482
 destroying, 479
 registering variables, 478–479
 starting, 478
 **session ID, 476**
 storing, 477–478
 **session variables, 476, 479**
 unsetting, 479
 **session_start() function, 478**
 **set_error_handler() function, 557–558**
 _set() function, 166–168
 setcookie() function, 476
 settype() function, 39
 SGML (Standard Generalized Markup Language), 16
 shared hosting of web applications, security issues, 356–357
SSL (Secure Sockets Layer), troubleshooting

short style PHP tags, 16
SHOW command (MySQL), 301–304
syntax, 303–304
show poll.php, 468–474
show tables command, 237
show_source() function, 525
shuffle() function, 90–91
simple tables, 218
simplegraph.php, 451–452
single-byte languages, 438
single-line comments, 18
size of files, determining, 70
sizeof() function, 98–99
slaves, setting up for replication, 313
Smart Form Mail application
creating, 101–104
regular expressions, 127–128
SMTP (Simple Mail Transfer Protocol), 404
software, updating, 354–355
Software Engineering: A Practitioner’s Approach, 542
software engineering, applying to web development, 530
solution components for PHPbookmark project, 561–565
sort() function, 76, 85–86
sorting arrays, 85–87
with asort() function, 86–87
with ksort() function, 86–87
multidimensional arrays, 87–90
reverse sorting, 83
with sort() function, 85–86
source code, highlighting, 525–526
special characters
meta characters, 124–125
pattern matching, 123–124
special privileges (MySQL), 230
splitting strings
explode() function, 112–113
with regular expressions, 129–130
with strtok() function, 113–114
with substr() function, 114
sprintf() function, 109
SQL (Structured Query Language), 247–248. See also MySQL
aggregating data, 259–261
INSERT command, 248–249
inserting data, 248–250
joins
cross joins, 258
equijoins, 258
full joins, 254–255
inner joins, 258
joining more than two tables, 255–256
left joins, 256–257
querying from the Web, 275–281
disconnecting from database, 281
filtering input data, 276
prepared statements, 279–280
retrieving the results, 280–281
selecting the database, 278
setting up connection, 277–278
retrieving data, 250–259
from multiple tables, 251–253
SELECT command, 250–251
with specific criteria, 251–253
subqueries, 262–263
correlated subqueries, 264
operators, 263
row subqueries, 264
as temporary table, 264
SSL (Secure Sockets Layer), troubleshooting, 610–612
stand-alone functions, _autoload(), 189
starting sessions, 478
statements, 16. See also commands
   echo, 22
   else, 42–43
   elseif, 43–44
   if, 41–42
   LOAD DATA INFILE, 315
   prepared statements, 279–280
   require(), 132–134
      applying templates to web pages, 134–139
   semicolons, 16
   switch, 44–45
static keyword, 185
status of variables, testing, 40–41
stopping scripts, 50, 520–522
storage engines, 316–317
   ARCHIVE, 316
   CSV, 316
   InnoDB, 316
      foreign keys, 319
      transactions, 318–319
   MEMORY, 316
   MERGE, 316
   MyISAM, 316
stored procedures, 320–327
   control structures, 323–327
      declare handlers, 325
   cursors, 323, 325
   example of, 320–323
   local variables, 323
storing
dates and times, Unix timestamps, 426–427
orders, 54
passwords, 300, 369
in RDBMSs, 74
records, 62
session ID, 477–478
str_replace() function, 107, 118–119
strategies for handling security, 341–343
   balancing with usability, 342
   monitoring, 342–343
   starting with the right mindset, 342
strcasecmp() function, 115
strchr() function, 117
strcmp() function, 115
strftime() function, 429–431
string operators, 29
string type columns, 244–246
strings. See also regular expressions
   changing case of, 111–112
   checking length of, 115–116
   comparing, 115
   concatenating, 22
   creating from classes, 194
   evaluating, 519–520
   filtering for output, 105–107, 347–348
      to browser, 105–106
      to email, 106–107
   finding within strings, 116–117
   formatting
      conversion specification, 109
      for printing, 109–111
   heredoc syntax, 23
   interpolation, 22
   joining, 113
   multibyte string functions, 440
   ordering, 115
   regular expressions, anchoring to
      beginning or end of, 123
   splitting
      explode() function, 112–113
with regular expressions, 129–130
with strtok() function, 113–114
with substr() function, 114
substrings
  find-and-replace operations, 118–119
  finding position of, 117–118
  replacing with string functions, 116
  trimming, 104
stristr() function, 117
strlen() function, 115–116
strnatcmp() function, 115
strpos() function, 117–118
strstr() function, 116–117
strtok() function, 113–114
strtolower() function, 112
strtoupper() function, 112
structure
  of classes, 162–163
  of functions, 144–145
strval() function, 41
subclasses, 161–162
  inheritance, 168–169
submit event, 504
subnamespaces, 197
subqueries, 262–263
  correlated subqueries, 264
  operators, 263
  row subqueries, 264
  as temporary table, 264
substr() function, 114
substr_replace() function, 118–119
substrings
  find-and-replace operations, 118–119
  finding position of, 117–118
  finding with regular expressions, 128–129
  replacing
    with regular expressions, 129
    with string functions, 116
subtraction operator, 28
Subversion, 537
success key, 507
superclasses, 161–162
superglobal arrays, 20, 27
support for images in PHP, setting up, 449–450
switch statement, 44–45
syntax
  heredoc, 23
  jQuery selectors, 496–497
  semicolons, 16
  SHOW command, 303–304
syntax errors, 543–544
system() function, 399

<table>
<thead>
<tr>
<th>T</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>table types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCHIVE, 316</td>
</tr>
<tr>
<td>CSV, 316</td>
</tr>
<tr>
<td>InnoDB, 316</td>
</tr>
<tr>
<td>foreign keys, 319</td>
</tr>
<tr>
<td>transactions, 318–319</td>
</tr>
<tr>
<td>MEMORY, 316</td>
</tr>
<tr>
<td>MERGE, 316</td>
</tr>
<tr>
<td>MyISAM, 316</td>
</tr>
<tr>
<td>selecting, 316</td>
</tr>
<tr>
<td>tables, 210, 218</td>
</tr>
<tr>
<td>aliases, 257–258</td>
</tr>
<tr>
<td>altering after creation, 265–268</td>
</tr>
<tr>
<td>columns, 235–237</td>
</tr>
<tr>
<td>CHAR type, 235</td>
</tr>
<tr>
<td>VARCHAR type, 235–236</td>
</tr>
<tr>
<td>creating, 232–234</td>
</tr>
</tbody>
</table>
displaying, 302
dropping, 268
grant tables, 292–293
columns_priv table, 296–298
connection verification, 298
db table, 295–296
procs_priv table, 296–298
request verification, 298	
tables_priv table, 296–298
user table, 293–295
joining
full joins, 254–255
left joins, 256–257
linking tables, 218
optimizing, 310
records
deleting, 268
updating, 265
relationships, 213
retrieving data
criteria, specifying, 251–253
from multiple tables, 251–253
rows, inserting into SQL database, 248–250
simple tables, 218
subqueries as temporary table, 264
triggers, 327–329
viewing, 237–238

tables_priv table, 296–298

tags
JavaScript, <script> 494–495
PHP, 16
short style, 16
XML style, 16

templates, applying to web pages, 134–139

terminating scripts, 520–522

ternary operator, 34

testing
code, 541–542
PHP support, 610
variable status, 40–41

text
applying to buttons, 461–464
bounding box, 462–463
descenders, 463
positioning on buttons, 464
regular expressions
anchoring to beginning or end of string, 123
assertions, 126–127
backreferences, 126
branching, 123
character class, 121–122
character sets, 120–121
counted subexpressions, 123
delimiters, 120
escape sequences, 125–126
meta characters, 124–125
repetition, 122
in Smart Form Mail application, 127–128
special characters, matching, 123–124
strings, splitting, 129–130
substrings, finding, 128–129
writing on buttons, 464–465

threats to web application security
access to sensitive data, 331–333
actors, 339–340
compromised server, 338
denial of service, 335–337
malicious code injection, 337
modification of data, 334
repudiation, 338–339
user-defined exceptions

three-dimensional arrays, 84–85
throw keyword, 200
time, microseconds, 435
timestamps, formatting, 429–431
timezones, 423–424
top-down approach to security, 343
totals, calculating on order forms, 36–37
tracking file upload progress, 387–388
traits, 174–176
transactions, 317–319
    using InnoDB, 318–319
translation files, 445–447
trigger_error() function, 556
triggering your own errors, 556
triggers, 327–329
trim() function, 104
trimming strings, 104
troubleshooting. See also error handling;
exception handling
    with EXPLAIN command, 308–309
    file upload, 389
    opening files, 58–61
    SSL, 610–612
TrueType fonts, 457
try blocks, 199
two-dimensional arrays, 82–84
twofold approach to security, 343
two-table joins, 254–255
type casting, 25
type codes for conversion specification, 110–111
type hinting, 185–186
type operator, 35
type strength, 25

ucwords() function, 112
uksort() function, 89
umask() function, 394
unary operator, 28–29
undefined functions, calling, 142–143
UNIX
    Apache, installing, 600–602
    MySQL, installing, 602–605
    PHP, installing, 605–609
Unix Epoch, 426
Unix timestamps, 426–427
    converting date and time to, 426
UNIX_TIMESTAMP() function, 432–433
unlink() function, 70
unserialize() function, 521
unsetting session variables, 479
update anomalies, 215
UPDATE command (SQL), 265
updating
    privileges, 299
    records, 265
    software, 354–355
uploading files, 379–389, 420
    HTML form, 381–382
    php.ini settings, 380–381
    tracking upload progress, 387–388
    troubleshooting, 389
    writing the file handling script, 382–387
upload.php, 382–387
urlencode() function, 407
usability, balancing with security, 342
use command, 232
user interface for chat application, building, 504–507
user personalization, 561
user table, 293–295
user-defined exceptions, 202–204
user-defined functions, 144
  parameters, 147
user-defined sorts, 88–89
users
  authentication, identifying visitors, 365–366
  MySQL, 225–232
    creating, 224, 225–227
    principle of least privilege, 225
    privileges, 227–230, 300–301
    privileges (MySQL), 291–299
    web access, 231–232
usort() function, 88–89
val() method, 498
validating dates with checkdate() function, 428–429
values, 211
  atomic column values, 216–217
  basic values, filtering, 346–347
  null values, 217–218
VARCHAR type columns, 235–236
variable functions, 146
variable handling functions, 39–40
variable variables, 25–26
variables, 23
  accessing, 20–22
  arrays, 75–76
    accessing contents, 77–78
    converting to scalar variables, 99–100
    initializing, 79
    loading from files, 92–96
    multidimensional arrays, 75
    navigating, 96–97
    numerically indexed arrays, 76–77
    reordering, 90–91
    reversing, 92
    sorting, 85–87
    three-dimensional arrays, 84–85
    two-dimensional arrays, 82–84
    assigning values to, 24
    assignment operators, 20
    and constants, 26
    data types, 24–25
      scalar values, 26
      type casting, 25
      type strength, 25
debugging, 551–553
environment variables, 401–402
handles, 161
identifiers, 23–24
interpolation, 22
local variables, 323
scope, 27, 148–150
serializing, 521
session variables, 476, 479
  registering, 478–479
unsetting, 479
status, testing, 40–41
version control, 536–537
viewing tables, 237–238
vieworders.php script, 65–66
visibility, controlling, 169–170
visitors, identifying, 365–366
vprintf() function, 111
vsprintf() function, 111
web access, configuring for MySQL users, 231–232
web application development
  applying to software engineering, 530
chat application
  chat server, building, 504–507
  user interface, building, 510–517
internationalized software, 437–438
jQuery, 494–495
large projects
  breaking up code, 535–536
  choosing a development environment, 537–538
  coding standards, 532
  commenting your code, 534
  defining naming conventions, 532–534
  directory structure, 536
  documenting, 538
  function libraries, 536
  indenting code, 534–535
  optimizing code, 540–541
  planning, 530–531
  prototyping, 538–539
  separating logic from content, 539–540
  testing code, 541–542
  version control, 536–537
  writing maintainable code, 532
localization, 437–438
  character sets, 438–440
  locales, 438
operating system, securing, 361–362
reusing code, 531–532
security
  code, securing, 343–352
  database servers, securing, 357–359
  disaster planning, 362–364
  executing commands, 353–354
  file system considerations, 352
  network security, 360–361
strategies for handling, 341–343
web servers, 354–357
threats
  access to sensitive data, 331–333
  compromised server, 338
  denial of service, 335–337
  loss of data, 334–335
  malicious code injection, 337
  modification of data, 334
  repudiation, 338–339
web database architecture, 218–220, 272
web pages
  internationalization
    language selector page, 442–444
    locale-specific headers, 441–442
  localizing, 440–445
  protecting, 371
  templates, applying with require() statement, 134–139
web servers, 218–219
  Apache HTTP Server
    .htaccess files, 374–377
    configuring, 356
  Nginx, 614
  security, 354–357
    browsing php.ini file, 355–356
  shared hosting of web applications, 356–357
  updating software, 354–355
websites
  Bill Gates Wealth Clock, 407
  consuming date from other sites, 404–408
  cookies, 476, 477
    session ID, storing, 477–478
  setting from PHP, 476–477
  session control, 475
  visitors, identifying, 365–366
WHERE clause (SELECT command), 252–253
  comparison operators, 252–253
while loops, 47–48
whitespace, 17
Windows operating system, installation packages, 612–613
writing
  code for classes, 177–184
    accessor functions, 178
    attributes, 177
    metatags, 177
    operations, 181
  file upload script, 382–387
to files, 55, 61
  as cause for runtime errors, 546–547
text on buttons, 464–465

X

XML, AJAX, 493–494
XML style PHP tags, 16
XOR operator, 32–33

Y-Z

yield keyword, 192–193