VISUAL QUICKPRO GUIDE

COVERS PHP 5 & 7

PHP and MySQL

for Dynamic Web Sites

Fifth Edition

LARRY ULLMAN

LEARN THE QUICK AND EASY WAY!

VISUAL QUICKPRO GUIDE

PHP and MySQL for Dynamic Web Sites

Fifth Edition

LARRY ULLMAN

Peachpit Press

Visual QuickPro Guide
PHP and MySQL for Dynamic Web Sites, Fifth Edition
Larry Ullman

Peachpit Press
www.peachpit.com

To report errors, please send a note to: errata@peachpit.com
Peachpit Press is a division of Pearson Education.

Editor: Mark Taber
Copy Editor: Elizabeth Welch
Technical Reviewer: Timothy Boronczyk
Production Coordinator: David Van Ness
Compositor: Danielle Foster
Proofreader: Scout Festa
Indexer: Valerie Haynes Perry
Cover Design: RHDG / Riezebos Holzbaur Design Group, Peachpit Press
Interior Design: Peachpit Press
Logo Design: MINE™ www.minesf.com

ISBN-13: 978-0-13-430184-6
ISBN-10: 0-13-430184-6

1 17

Printed and bound in the United States of America

Dedication

Dedicated to the fine faculty at my alma mater, Northeast Missouri State University. In particular, I would like to thank Dr. Monica Barron, Dr. Dennis Leavens, Dr. Ed Tyler, and Dr. Cole Woodcox, whom I also have the pleasure of calling my friend. I would not be who I am as a writer, as a student, as a teacher, or as a person if it were not for the magnanimous, affecting, and brilliant instruction I received from these educators.

Special Thanks to:

My heartfelt thanks to everyone at Peachpit Press, as always.

My gratitude to the fine editor on this project, Mark Taber, for leading the way and putting up with too many delayed emails and chapters!

Thanks to David Van Ness and Elizabeth Welch for their hard work, helpful suggestions, and impressive attention to detail. Thanks to Scout Festa for ensuring the writing is “pixel perfect.” Thanks also to Valerie Perry for indexing and Danielle Foster for laying out the book, and thanks to Timothy Boronczyk for his technical review.

Kudos to the good people working on PHP, MySQL, Apache, phpMyAdmin, MAMP, and XAMPP, among other great projects. And a hearty “cheers” to the denizens of the various newsgroups, mailing lists, support forums, etc., who offer assistance and advice to those in need.

Thanks, as always, to the readers, whose support gives my job relevance. An extra helping of thanks to those who provided the translations in Chapter 17, “Example—Message Board,” and who offered up recommendations as to what they’d like to see in this edition.

Finally, I would not be able to get through a single book if it weren’t for the love and support of my wife, Jessica. And a special shout-out to Zoe and Sam, who give me reasons to, and not to, write books!

Table of Contents

Introduction

Today's web users expect exciting pages that are updated frequently and provide a customized experience. For them, web sites are more like communities, to which they'll return time and again. At the same time, site administrators want pages that are easier to update and maintain, understanding that's the only reasonable way to keep up with visitors' expectations. For these reasons and more, PHP and MySQL have become the de facto standards for creating dynamic, database-driven web sites.

This book represents the culmination of my many years of web development experience coupled with the value of having written several previous books on the technologies discussed herein. The focus of this book is on covering the most important knowledge in the most efficient manner. It will teach you how to begin developing dynamic web sites and give you plenty of example code to get you started. All you need to provide is an eagerness to learn.

Well, that and a computer.

What Are Dynamic Web Sites?

Dynamic web sites are flexible and potent creatures, more accurately described as *applications* than merely sites. Dynamic web sites

- Respond to different parameters (for example, the time of day or the version of the visitor's browser)
- Have a "memory," allowing for user registration and login, e-commerce, and similar processes
- Almost always integrate HTML forms, allowing visitors to perform searches, provide feedback, and so forth
- Often have interfaces where administrators can manage the content
- Are easier to maintain, upgrade, and build upon than statically made sites

Many technologies are available for creating dynamic web sites. The most common are ASP.NET (Active Server Pages, a Microsoft construct), JSP (JavaServer Pages), ColdFusion, Ruby on Rails (a web development framework for the Ruby programming language), and PHP. Dynamic sites don't always rely on a database, but more and more of them do, particularly as excellent database applications like MySQL and MongoDB are available at little to no cost.

What Happened to PHP 6?

When I wrote a previous edition of this book, *PHP 6 and MySQL 5 for Dynamic Web Sites: Visual QuickPro Guide*, the next major release of PHP—PHP 6—was approximately 50 percent complete. Thinking that PHP 6 would therefore be released sometime after the book was published, I relied on a beta version of PHP 6 for a bit of that edition's material. And then... PHP 6 died.

One of the key features planned for PHP 6 was support for Unicode, meaning that PHP 6 would be able to work natively with any language. This would be a great addition to an already popular programming tool. Unfortunately, implementing Unicode support went from being complicated to quite difficult, and the developers behind the language tabled development of PHP 6. Not all was lost, however; some of the other features planned for PHP 6, such as support for *namespaces* (an object-oriented programming concept), were added to PHP 5.3.

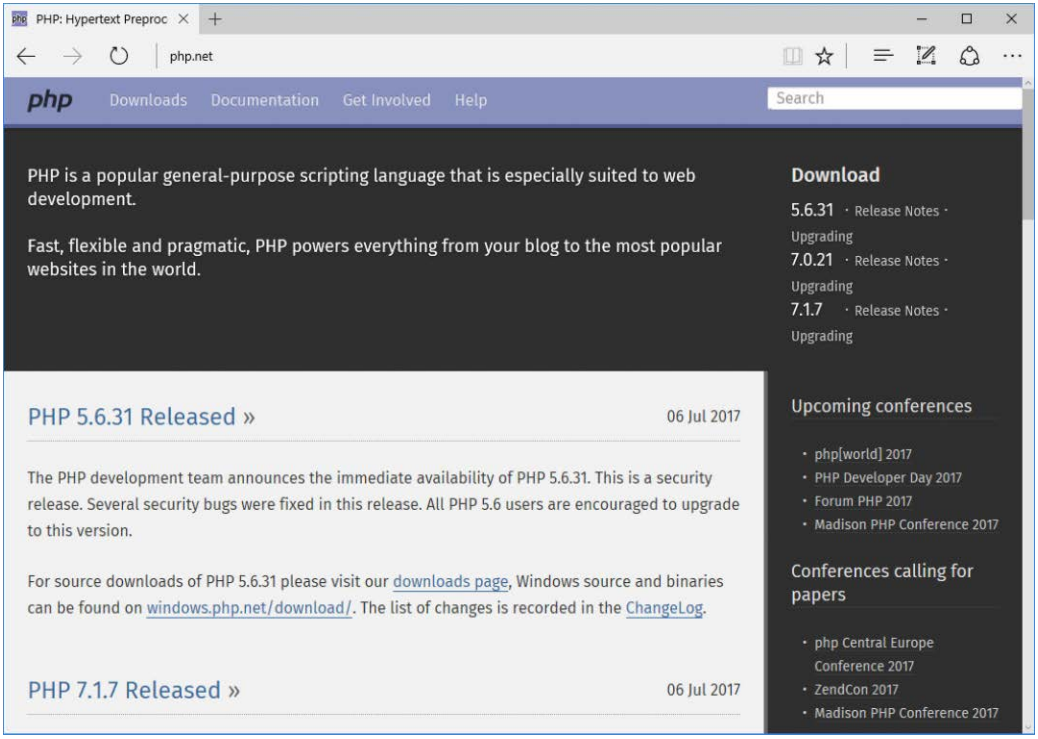
When it was time to release the next major version of PHP, it was decided to name it PHP 7 to avoid confusion with the PHP 6 version that was started but never completed.

What is PHP?

PHP originally stood for “Personal Home Page” when it was created in 1994 by Rasmus Lerdorf to track the visitors to his online résumé. As its usefulness and capabilities grew (and as it started being used in more professional situations), it came to mean “PHP: Hypertext Preprocessor.”

According to the official PHP web site, found at **`www.php.net`** Ⓐ, PHP is a “popular general-purpose scripting language that is especially suited to web development.” It’s a long but descriptive definition, whose meaning I’ll explain.

continues on next page



Ⓐ The home page for PHP.

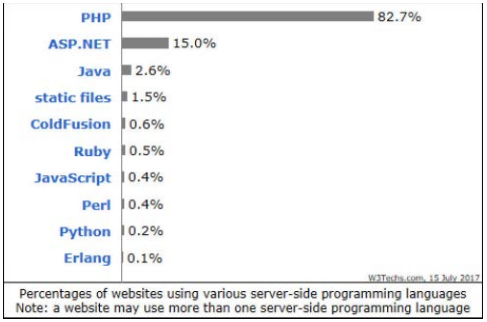
Starting at the end of that statement, to say that PHP *is especially suited to web development* means that although you can use PHP for non-web development purposes, it's best suited for that. The corollary is that although many other technologies can be used for web development, that may not be what they're best suited for. Simply put, if you're hoping to do web development, PHP is an excellent choice.

Also, PHP is a *scripting* language, as opposed to a *compiled* language: PHP was designed to write web scripts, not stand-alone applications (although, with some extra effort, you can create applications in PHP). PHP scripts run only after an event occurs—for example, when a user submits a form or goes to a URL (uniform resource locator, the technical term for a web site address).

I should add to this definition that PHP is a server-side, cross-platform technology, both descriptions being important. *Server-side* refers to the fact that everything PHP does occurs on the server. A web server application, like Apache or Microsoft's IIS (Internet Information Services), is required and all PHP scripts must be accessed through a URL (**`http://something`**). Its cross-platform nature means that PHP runs on most operating systems, including Windows, Unix (and its many variants), and Macintosh. More important, the PHP scripts written on one server will normally work on another with little or no modification.

At the time this book was written, PHP was at version 7.1.7. Although PHP 7 is a major release, the most important changes are in its core, with PHP 7 being significantly more performant than PHP 5.

For the most part, the examples in this book will work fine so long as you're using at least version 5.4. Some functions and



B The Web Technology Surveys site provides this graphic regarding server-side technologies (**`www.w3techs.com/technologies/overview/programming_language/all`**).

features covered will require more specific or current versions, like PHP 5.6 or greater. In those cases, I will make it clear when the functionality was added to PHP, and provide alternative solutions if you have a slightly older version of the language.

More information about PHP can always be found at PHP.net.

Why use PHP?

Put simply, when it comes to developing dynamic web sites, PHP is better, faster, and easier to learn than the alternatives. What you get with PHP is excellent performance, a tight integration with nearly every database available, stability, portability, and a nearly limitless feature set due to its extendibility. All of this comes at no cost (PHP is open source) and with a very manageable learning curve. PHP is one of the best marriages I've ever seen between the ease with which beginning programmers can start using it and the ability for more advanced programmers to do everything they require.

Finally, the proof is in the pudding: PHP has seen an exponential growth in use since its inception, and is the server-side technology of choice on over 82 percent of all web sites **B**. In terms of all programming languages, PHP is the sixth most popular **C**.

continues on next page

Jul 2017	Jul 2016	Change	Programming Language	Ratings	Change
1	1		Java	13.774%	-6.03%
2	2		C	7.321%	-4.92%
3	3		C++	5.576%	-0.73%
4	4		Python	3.543%	-0.62%
5	5		C#	3.518%	-0.40%
6	6		PHP	3.093%	-0.18%

C The Tiobe Index (**`https://www.tiobe.com/tiobe-index/`**) uses a combination of factors to rank the popularity of programming languages.

Of course, you might assume that I, as the author of a book on PHP (several, actually), have a biased opinion. Although not nearly to the same extent as I have with PHP, I've also developed sites using JavaServer Pages (JSP), Ruby on Rails (RoR), Sinatra (another Ruby web framework), and ASP.NET. Each has its pluses and minuses, but PHP is the technology I always return to. You might hear that it doesn't perform or scale as well as other technologies, but Yahoo, Wikipedia, and Facebook all use PHP, and you can't find many sites more visited or demanding than those.

You might have heard that PHP is less secure. But *security isn't in the language*; it's in how that language is used. Rest assured that a complete and up-to-date discussion of all the relevant security concerns is provided by this book.

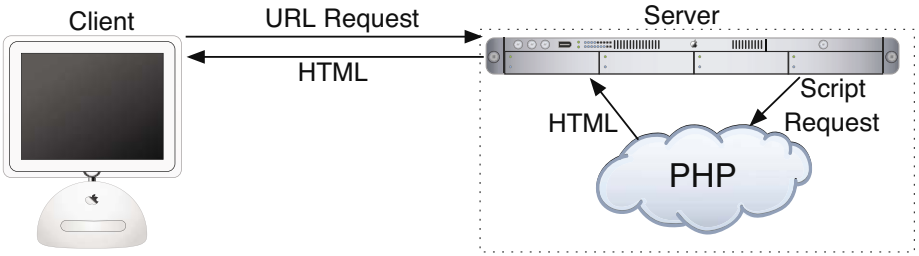
How PHP works

As previously stated, PHP is a server-side language. This means that the code you write in PHP sits on a host computer called a *server*. The server sends web pages to the requesting visitors (you, the client, with your browser).

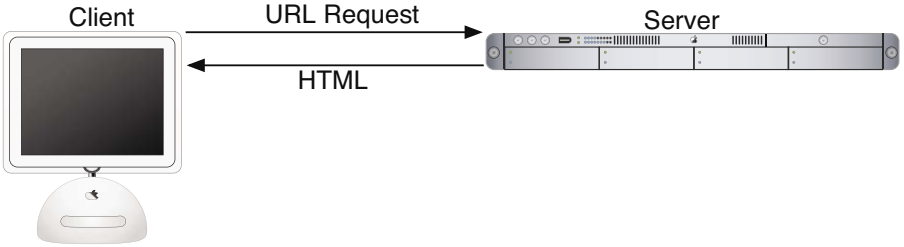
When a visitor goes to a site written in PHP, the server reads the PHP code and then processes it according to its scripted directions. In the example shown in D, the PHP code tells the server to send the appropriate data—HTML code—to the browser, which treats the received code as it would a standard HTML page.

This differs from a static HTML site where, when a request is made, the server merely sends the HTML data to the browser and there is no server-side interpretation occurring E. Because no server-side action is required, you can run HTML pages in your browser without using a server at all.

continues on next page



D How PHP fits into the client/server model when a user requests a page.



E The client/server process when a request for a static HTML page is made.

To the end user and the browser there is no perceptible difference between what **`home.html`** and **`home.php`** may look like, but how that page's content was created will be significantly different.

What is MySQL?

MySQL (**`www.mysql.com`**) Ⓕ is the world's most popular open source database. In fact, today MySQL is a viable competitor to pricey goliaths such as Oracle and Microsoft's SQL Server (and, ironically, MySQL is owned by Oracle). Like PHP, MySQL offers excellent performance, portability, and reliability, with a moderate learning curve and little to no cost.

MySQL is a database management system (DBMS) for relational databases (therefore, MySQL is an RDBMS). A database, in the simplest terms, is a collection of data, be it text, numbers, or binary files, stored and kept organized by the DBMS.

There are many types of databases, from the simple flat-file to relational to object-oriented to NoSQL. A relational database uses multiple tables to store information in its most discernible parts. Although relational databases may involve more thought in the design and programming stages, they offer improved reliability and data integrity that more than make up for the extra effort required. Further, relational databases are more searchable and allow for concurrent users.

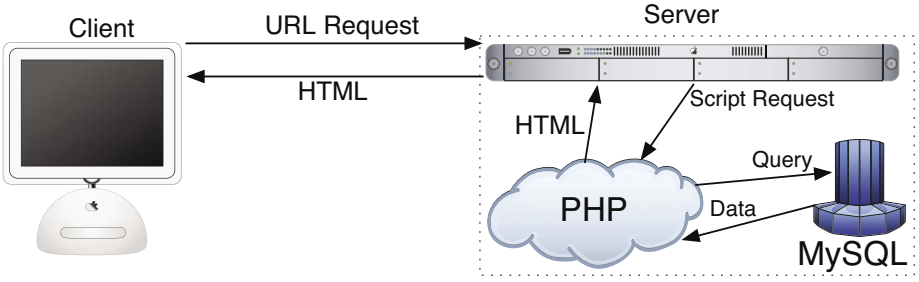
Ⓕ The home page for the MySQL database application.

By incorporating a database into a web application, some of the data generated by PHP can be retrieved from MySQL G. This further moves the site's content from a static (hard-coded) basis to a flexible one, flexibility being the key to a dynamic web site.

MySQL is an open source application, like PHP, meaning that it is free to use or even modify (the source code itself is downloadable). There are occasions when you should pay for a MySQL license, especially if you are making money from the sales or incorporation of the MySQL product. Check MySQL's licensing policy for more information on this.

The MySQL software consists of several pieces, including the MySQL server (*mysqld*, which runs and manages the databases), the MySQL client (*mysql*, which gives you an interface to the server), and numerous utilities for maintenance and other purposes. PHP has always had good support for MySQL, and that is even truer in the most recent versions of the language.

continues on next page



G How most of the dynamic applications in this book will work, using both PHP and MySQL.

MySQL has been known to handle databases as large as 60,000 tables with more than several billion rows. MySQL can work with tables as large as thousands of terabytes on some operating systems, generally a healthy 4 GB otherwise. MySQL is used by NASA and the U.S. Census Bureau, among many others.

As of this writing, MySQL is on version 5.7.18. The version of MySQL you have affects what features you can use, so it's important that you know what you're working with. For this book, MySQL 5.7.14 was used, although you should be able to do everything in this book as long as you're using a version of MySQL greater than 5.0.

Pronunciation Guide

Trivial as it may be, I should clarify up front that MySQL is technically pronounced "My Ess Cue Ell," just as SQL should be said "Ess Cue Ell." This is a question many people have when first working with these technologies. Though not a critical issue, it's always best to pronounce acronyms correctly.

What You'll Need

To follow the examples in this book, you'll need the following tools:

- A web server application (for example, Apache, Nginx, or IIS)
- PHP
- MySQL
- A browser (Microsoft's Internet Explorer or Edge, Mozilla's Firefox, Apple's Safari, Google's Chrome, etc.)
- A text editor, PHP-capable WYSIWYG application (Adobe's Dreamweaver qualifies), or IDE (integrated development environment)
- An FTP application, if using a remote server

One of the great things about developing dynamic web sites with PHP and MySQL is that all of the requirements can be met at no cost whatsoever, regardless of your operating system! Apache, PHP, and MySQL are each free, browsers can be had without cost, and many good text editors are available for nothing.

The appendix discusses the installation process on the Windows and macOS operating systems. If you have a computer, you are only a couple of downloads away from being able to create dynamic web sites (in that case, your computer would represent both the client and the server in D and E). Conversely, you could purchase web hosting for only dollars per month that will provide you with a PHP- and MySQL-enabled environment already online.

About This Book

This book teaches you how to develop dynamic web sites with PHP and MySQL, covering the knowledge that most developers might require. In keeping with the format of the Visual QuickPro series, the information is discussed using a step-by-step approach with corresponding images. The focus has been kept on real-world, practical examples, avoiding "here's something you could do but never would" scenarios. As a practicing web developer myself, I wrote about the information that I use and avoided those topics immaterial to the task at hand. As a practicing writer, I made certain to include topics and techniques that I know readers are asking about.

The structure of the book is linear, and the intention is that you'll read it in order. It begins with three chapters covering the fundamentals of PHP (by the second chapter, you will have already developed your first dynamic web page). After that, there are four chapters on SQL (Structured Query Language, which is used to interact with all databases) and MySQL. Those chapters teach the basics of SQL, database design, and the MySQL application in particular. Then there's one chapter on debugging and error management, information everyone needs. This is followed by a chapter introducing how to use PHP and MySQL together, a remarkably easy thing to do.

The following five chapters teach more application techniques to round out your knowledge. Security, in particular, is repeatedly addressed in those pages. The next two chapters expand your newfound knowledge into subjects that, though not critical, are ones you'll want to pick up in time regardless. Finally, I've included two example chapters, in which the heart of different web applications are developed, with instructions.

Is this book for you?

This book was written for a wide range of people within the beginner-to-intermediate range. The book makes use of HTML5, so solid experience with HTML is a must. Although this book covers many things, it does not formally teach HTML or web design. Some CSS is sprinkled about these pages but also not taught.

Second, this book expects that you have one of the following:

- The drive and ability to learn without much hand holding, or...
- Familiarity with another programming language (even solid JavaScript skills would qualify), or...
- A cursory knowledge of PHP

Make no mistake: This book covers PHP and MySQL from A to Z, teaching everything you'll need to know to develop real-world web sites, but the early chapters in particular cover PHP at a quick pace. For this reason I recommend either some programming experience or a curious and independent spirit when it comes to learning new things. If you find that the material goes too quickly, you should probably start off with the latest edition of my book *PHP for the World Wide Web: Visual Quick-Start Guide*, which goes at a much more tempered pace.

No database experience is required, since SQL and MySQL are discussed starting at a more basic level.

What's new in this edition

The first four editions of this book have been very popular, and I've received a lot of positive feedback on them (thanks!). In writing this new edition, I focused on ensuring the material is accurate, up to date, and in keeping with today's standards and best practices. The changes in this edition include

- Updating all the code to use HTML5
- Use of more modern HTML design techniques, including multiple examples of the Twitter Bootstrap framework
- Updating everything for the latest versions of PHP and MySQL
- Additional PHP and MySQL examples, such as performing transactions from a PHP script
- Even more information and examples for improving the security of your scripts and sites
- Removal of outdated content (e.g., things used in older versions of PHP or no longer applicable)
- Return of the installation appendix to the printed book (in the fourth edition, the appendix was freely available online instead)

For those of you that also own a previous edition (thanks, thanks, thanks!), I hope you find this to be a fresh and sharp update to an already excellent resource.

How this book compares to my other books

This is my fourth PHP and/or MySQL title, after (in order)

- *PHP for the World Wide Web: Visual QuickStart Guide*
- *PHP Advanced and Object-Oriented Programming: Visual QuickPro Guide*
- *MySQL: Visual QuickStart Guide*

I hope this résumé implies a certain level of qualification to write this book, but how do you, as a reader standing in a bookstore, decide which title is for you? Of course, you are more than welcome to splurge and buy the whole set, earning my eternal gratitude, but...

The *PHP for the World Wide Web: Visual QuickStart Guide* book is very much a beginner's guide to PHP. This title overlaps it some, mostly in the first three chapters, but uses new examples so as not to be redundant. For novices, this book acts as a follow-up to that one. The advanced book is really a sequel to this one, as it assumes a fair amount of knowledge and builds on many things taught here. The MySQL book focuses almost exclusively on MySQL (there are but two chapters that use PHP).

continues on next page

With that in mind, read the section "Is this book for you?" and see if the requirements apply. If you have no programming experience at all and would prefer to be taught PHP more gingerly, my first book would be better. If you are already very comfortable with PHP and want to learn more of its advanced capabilities, pick up *PHP Advanced and Object-Oriented Programming: Visual QuickPro Guide*. If you are most interested in MySQL and are not concerned with learning much about PHP, check out *MySQL: Visual QuickStart Guide*.

That being said, if you want to learn everything you need to know to begin developing dynamic web sites with PHP and MySQL today, then this is the book for you! It references the most current versions of both technologies, uses techniques not previously discussed in other books, and contains its own unique examples.

And whatever book you do choose, make sure you're getting the most recent edition or, barring that, the edition that best matches the versions of the technologies you'll be using.

Companion Web Site

I have developed a companion web site specifically for this book, which you may reach at **`LarryUllman.com`**. There you will find every script from this book, a text file containing lengthy SQL commands, and a list of errata that occurred during publication. (If you have problems with a command or script, and you are following the book exactly, check the errata to ensure there is not a printing error before driving yourself absolutely mad.) At this web site you will also find a popular forum where readers can ask and answer each other's questions (I answer many of them myself), and more!

Questions, comments, or suggestions?

If you have any questions on PHP or MySQL, you can turn to one of the many web sites, mailing lists, newsgroups, and FAQ repositories already in existence. A quick search online will turn up virtually unlimited resources. For that matter, if you need an immediate answer, those sources or a quick online search will most assuredly serve your needs (in all likelihood, someone else has already seen and solved your exact problem).

You can also direct your questions, comments, and suggestions to me. You'll get the fastest reply using the book's corresponding forum (I always answer those questions first). If you'd rather email me, my contact information is available on my site. I do try to answer every email I receive, although I cannot guarantee a quick reply.

Accessing the free Web Edition

Your purchase of this book in any format includes access to the corresponding Web Edition, which provides several special online-only features:

- The complete text of the book, with all the figures and in full color
- 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 for Dynamic Web Sites: Visual QuickPro Guide* all you need to do is register this book:

1. Go to **`www.peachpit.com/register`**.
2. Sign in or create a new account.
3. Enter ISBN: 9780134301846.
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.

Introduction to PHP

Although this book focuses on using MySQL and PHP together, you'll do the majority of your legwork using PHP alone. In this and the following chapter, you'll learn PHP's basics, from syntax to variables, operators, and language constructs (conditionals, loops, and whatnot). As you are picking up these fundamentals, you'll also develop usable code that you'll integrate into larger applications later in the book.

This introductory chapter will cruise through most of the basics of the PHP language. You'll learn the syntax for coding PHP, how to send data to the browser, and how to use two kinds of variables—strings and numbers—plus constants. Some examples may seem inconsequential, but they'll demonstrate ideas you'll need to master in order to write more advanced scripts further down the line. The chapter concludes with some quick debugging tips...you know...just in case!

In This Chapter

Basic Syntax

As stated in the book's introduction, PHP is an *HTML-embedded* scripting language, meaning that you can intermingle PHP and HTML code within the same file. So to begin programming with PHP, start with a simple web page. **Script 1.1** is an example of a no-frills, no-content HTML5 document, which will be used as the foundation for most web pages in the book (this book does not formally discuss HTML5; see a resource dedicated to the topic for more information). Please also note that the template uses UTF-8 encoding, a topic discussed in the following sidebar.

To add PHP code to a page, place it within PHP tags:

```
<?php
?>
```

Script 1.1 A basic HTML5 page.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Page Title</title>
6   </head>
7   <body>
8      <!-- Script 1.1 - template.html -->
9   </body>
10  </html>
```

Understanding Encoding

Encoding is a big subject, but what you most need to understand is this: *the encoding you use in a file dictates what characters can be represented* (and therefore, what languages can be used). To select an encoding, you must first confirm that your text editor or integrated development environment (IDE)—whatever application you're using to create the HTML and PHP scripts—can save documents using that encoding. Some applications let you set the encoding in the preferences or options area; others set the encoding when you save the file.

To indicate the encoding to the browser, there's the corresponding **meta** tag:

```
<meta charset="utf-8">
```

The *charset=utf-8* part says that UTF-8 encoding is being used, short for *8-bit Unicode Transformation Format*. Unicode is a way of reliably representing every symbol in every alphabet. Version 9.0.0 of Unicode—the current version as of this writing—supports over 128,000 characters!

If you want to create a multilingual web page, UTF-8 is the way to go, and I'll be using it in this book's examples. You don't have to, of course. But whatever encoding you do use, make sure that the encoding indicated by the HTML page matches the actual encoding set in your text editor or IDE. If you don't, you'll likely see odd characters when you view the page in a browser.

Script 1.2 This first PHP script doesn't do anything, but it does demonstrate how a PHP script is written. It'll also be used as a test script, prior to getting into elaborate PHP code.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Basic PHP Page</title>
6   </head>
7   <body>
8      <!-- Script 1.2 - first.php -->
9      <p>This is standard HTML.</p>
10  <?php
11  ?>
12  </body>
13  </html>
```

Anything written within these tags will be treated by the web server as PHP, meaning the PHP interpreter will process the code. Any text outside of the PHP tags is immediately sent to the browser as regular HTML. Because PHP is most often used to create content displayed in the browser, the PHP tags are normally put somewhere within the page's body.

Along with placing PHP code within PHP tags, your PHP files must have a proper *extension*. The extension tells the server to treat the script in a special way—namely, as a PHP page. Most web servers use **`.html`** for standard HTML pages and **`.php`** for PHP files.

Before getting into the steps, understand that *you must already have a working PHP installation*! This could be on a hosted site or your own computer, after following the instructions in Appendix A, "Installation."

To make a basic PHP script:

1. Create a new document in your text editor or IDE, to be named **`first.php`** (**Script 1.2**).

   It generally does not matter what application you use, be it Adobe Dreamweaver (a fancy IDE), Sublime Text (a great and popular plain-text editor), or vi (a plain-text Unix editor, lacking a graphical interface). Still, some text editors and IDEs make typing and debugging HTML and PHP easier (conversely, Notepad on Windows does some things that make coding harder: *don't use Notepad*!). If you don't already have an application you're attached to, search online or use the book's corresponding forum (**`LarryUllman.com/forums/`**) to find one.

continues on next page

2. Create a basic HTML document:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Basic PHP Page</title>
</head>
<body>
  <!-- Script 1.2 - first.php -->
  <p>This is standard HTML.</p>
</body>
</html>
```

This is a basic HTML5 page. One of the niceties of HTML5 is its minimal doctype and syntax.

3. Before the closing **body** tag, insert the PHP tags:

```
<?php
?>
```

These are the *formal* PHP tags, also known as *XML-style* tags. Although PHP supports other tag types, I recommend that you use the formal type, and I will do so throughout this book.

4. Save the file as **first.php**.

Remember that if you don't save the file using an appropriate PHP extension, the script will not execute properly. (Just one of the reasons not to use Notepad is that it will secretly add the **.txt** extension to PHP files, thereby causing many headaches.)

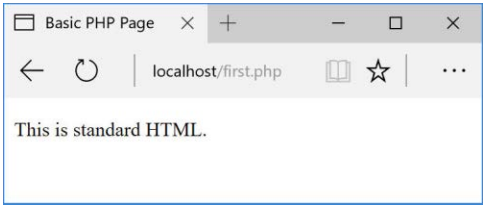
5. Place the file in the proper directory of your web server.

If you are running PHP on your own computer (presumably after following the installation directions in Appendix A), you just need to move, copy, or save the file to a specific folder on your computer. Check Appendix A or the documentation for your particular web server to identify the correct directory, if you don't already know what it is.

If you are running PHP on a hosted server (i.e., on a remote computer), you'll need to use a Secure File Transfer Protocol (SFTP) application to upload the file to the proper directory. Your hosting company will provide you with access and the other necessary information.

6. Run **first.php** in your browser Ⓐ.

Because PHP scripts need to be parsed by the server, you *absolutely must* access them via a URL (i.e., the address in the browser must begin with **http://** or **https://**). You cannot simply open them in your browser as you would a file in other applications (in which case the address would start with **file://** or **C:** or the like).



Ⓐ While it seems like any other (simple) HTML page, this is in fact a PHP script and the basis for the rest of the examples in the book.

If you are running PHP on your own computer, you'll need to use a URL like **`http://localhost/first.php`**, **`http://127.0.0.1/first.php`**, or **`http://localhost/~<user>/first.php`** (on macOS, using your actual username for **`<user>`**). If you are using a hosted site, you'll need to use **`http://your-domain-name/first.php`** (e. g., **`http://www.example.com/first.php`**).

7. If you don't see results like those in A, start debugging!

   Part of learning any programming language is mastering debugging. It's a sometimes painful but absolutely necessary process. With this first example, if you don't see a simple, but perfectly valid, web page, follow these steps:

   A. Confirm that you have a working PHP installation (see Appendix A for testing instructions).

   B. Make sure that you are running the script through a URL. The address in the browser must begin with **`http`**. If it starts with **`file://`**, that's a problem B.

   C. If you get a file not found (or similar) error, you've likely put the file in the wrong directory or mistyped the file's name (either when saving it or in your browser).

   If you've gone through all this and you are still having problems, turn to the book's corresponding forum (**`LarryUllman.com/forums/`**).

TIP To find more information about HTML, check out Elizabeth Castro's excellent *HTML and CSS: Visual QuickStart Guide* (Peachpit, 2013), or search online.

TIP You can embed multiple sections of PHP code within a single HTML document (i.e., you can go in and out of the two languages). You'll see examples of this throughout the book.

TIP You can declare the encoding of an external CSS file by adding `@charset "utf-8";` as the first line in the file. If you're not using UTF-8, change the line accordingly.

B PHP code will only be executed when run through http://.

Sending Data to the Browser

To create dynamic web sites with PHP, you must know how to send data to the browser. PHP has a number of built-in functions for this purpose; the most common are **echo** and **print**. I tend to favor **echo**:

```
echo 'Hello, world!';
echo "What's new?";
```

You could use **print** instead if you prefer (the name more obviously indicates what it does):

```
print 'Hello, world!';
print "What's new?";
```

As you can see from these examples, you can use either single or double quotation marks (but there is a distinction between the two types of quotation marks, which I'll make clear by this chapter's end). The first quotation mark after the function name indicates the start of the message to be printed. The next matching quotation mark (i.e., the next quotation mark of the same kind as the opening mark) indicates the end of the message to be printed.

Along with learning how to send data to the browser, you should also notice that in PHP all statements—a line of executed code, in layman's terms—must end with a semicolon. Also, PHP is *case-insensitive* when it comes to function names, so **ECHO**, **echo**, **eCHo**, and so forth will all work. The all-lowercase version is easiest to type, of course.

Needing an Escape

As you might discover, one of the complications with sending data to the browser involves printing single and double quotation marks. Either of the following will cause errors:

```
echo "She said, "How are you?"";
echo 'I'm just ducky.';
```

There are two solutions to this problem. First, use single quotation marks when printing a double quotation mark, and vice versa:

```
echo 'She said, "How are you?"';
echo "I'm just ducky.";
```

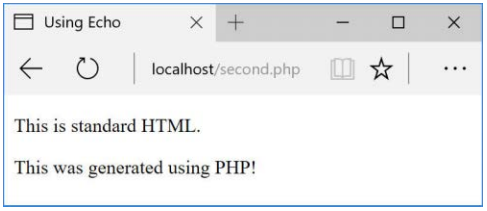
Or, you can *escape* the problematic character by preceding it with a backslash:

```
echo "She said, \"How are you?\"";
echo 'I\'m just ducky.';
```

An escaped quotation mark will merely be printed like any other character. Understanding how to use the backslash to escape a character is an important concept, and one that will be covered in more depth at the end of this chapter.

Script 1.3 Using **print** or **echo**, PHP can send data to the browser.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Using Echo</title>
6   </head>
7   <body>
8      <!-- Script 1.3 - second.php -->
9      <p>This is standard HTML.</p>
10  <?php
11  echo 'This was generated using PHP!';
12  ?>
13  </body>
14  </html>
```



Ⓐ The results still aren't glamorous, but this page was in part dynamically generated by PHP.

To send data to the browser:

1. Open **first.php** (refer to Script 1.2) in your text editor or IDE.
2. Between the PHP tags (lines 10 and 11), add a simple message (**Script 1.3**):

   ```
   echo 'This was generated using
   → PHP!';
   ```

   It truly doesn't matter what message you type here, which function you use (**echo** or **print**), or which quotation marks, for that matter—just be careful if you are printing a single or double quotation mark as part of your message (see the sidebar "Needing an Escape").
3. If you want, change the page title to better describe this script (line 5):

   ```
   <title>Using Echo</title>
   ```

   This change affects only the browser window's title bar.
4. Save the file as **second.php**, place it in your web directory, and test it in your browser Ⓐ.

   Remember that all PHP scripts must be run through a URL (**http://something**)!

continues on next page

5. If necessary, debug the script.

   If you see a parse error instead of your message ❶, check that you have both opened and closed your quotation marks and escaped any problematic characters (see the sidebar). Also be certain to conclude each statement with a semicolon.

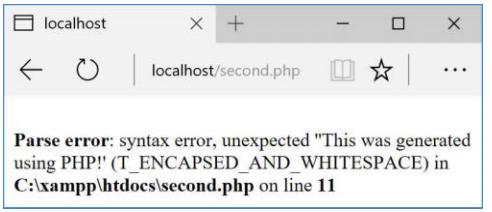
   If you see an entirely blank page, this is probably for one of two reasons:

   - There is a problem with your HTML. Test this by viewing the source of your page and looking for HTML problems there ❷.
   - An error occurred, but *display_errors* is turned off in your PHP configuration, so nothing is shown. In this case, see the section in Appendix A on how to configure PHP so that you can turn *display_errors* back on.

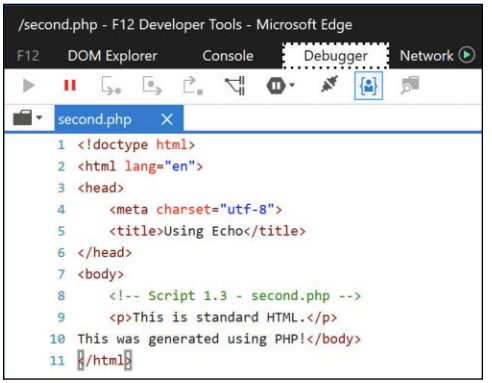
TIP Technically, `echo` and `print` are language constructs, not functions. That being said, don't be bothered as I continue to call them "functions" for convenience. Also, as you'll see later in the book, I include the parentheses when referring to functions—say, `number_format()`, not just `number_format`—to help distinguish them from variables and other parts of PHP. This is just my own little convention.

TIP You can, and often will, use `echo` and `print` to send HTML code to the browser, like so ❸:

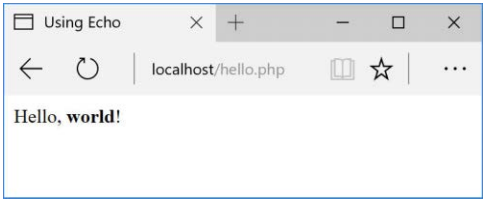
```
echo '<p>Hello,
→ <strong>world</strong>!</p>';
```



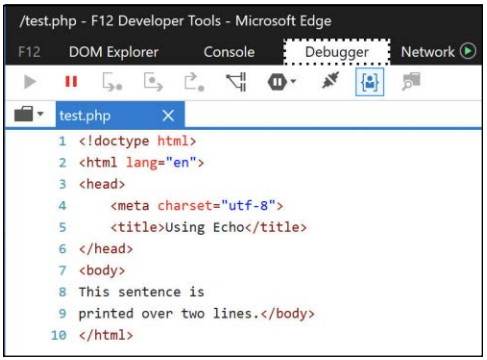
❶ This may be the first of many parse errors you see as a PHP programmer (this one is caused by the omission of the terminating quotation mark).



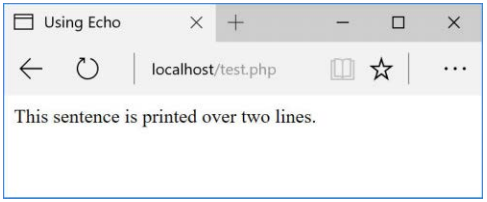
❷ One possible cause of a blank PHP page is a simple HTML error, like the closing `title` tag here (it's missing the slash).



❸ PHP can send HTML code (like the formatting here) as well as simple text Ⓐ to the browser.



E Printing text and HTML over multiple PHP lines will generate HTML source code that also extends over multiple lines. Note that extraneous white spacing in the HTML source will not affect the look of a page F but can make the source easier to review.



F The return in the HTML source E has no effect on the rendered result. The only way to alter the spacing of a displayed web page is to use HTML tags (like **
** and ** **).

TIP **`echo` and `print` can both be used over multiple lines:**

```
echo 'This sentence is
printed over two lines.';
```

What happens in this case is that the return (created by pressing Enter or Return) becomes part of the printed message and isn't terminated until the closing quotation mark. The net result will be the "printing" of the return in the HTML source code E. This will not have an effect on the generated page F. For more on this, see the sidebar "Understanding White Space."

Writing Comments

Creating executable PHP code is only a part of the programming process (admittedly, it's the most important part). A secondary but still crucial aspect to any programming endeavor is documenting your code.

In HTML you can add comments using special tags:

```
<!-- Comment goes here. -->
```

HTML comments are viewable in the source but do not appear in the rendered page (see E and F in the previous section).

PHP comments are different in that they aren't sent to the browser at all, meaning they won't be viewable to the end user, even when looking at the HTML source.

PHP supports three comment syntaxes. The first uses what's called the pound, hash, or number symbol (**#**):

```
# This is a comment.
```

The second uses two slashes:

```
// This is also a comment.
```

Both of these cause PHP to ignore everything that follows until the end of the line (when you press Return or Enter). Thus, these two comments are for single lines only. They are also often used to place a comment on the same line as some PHP code:

```
print 'Hello!'; // Say hello.
```

A third style allows comments to run over multiple lines:

```
/* This is a longer comment
that spans two lines. */
```

Understanding White Space

With PHP you send data—like HTML tags and text—to the browser, which will, in turn, render that data as the web page the end user sees. Thus, what you are often doing with PHP is creating the *HTML source* of a web page. With this in mind, there are three areas of notable *white space* (extra spaces, tabs, and blank lines): in your PHP scripts, in your HTML source, and in the rendered web page.

PHP is generally white space insensitive, meaning that you can space out your code however you want to make your scripts more legible. HTML is also generally white space insensitive. Specifically, the only white space in HTML that affects the rendered page is a single space (multiple spaces still get rendered as one). If your HTML source has text on multiple lines, that doesn't mean it'll appear on multiple lines in the rendered page (E and F).

To alter the spacing in a rendered web page, use the HTML tags **
** (line break) and ** ** (paragraph). To alter the spacing of the HTML *source* created with PHP, you can

- Use **echo** or **print** over the course of several lines.

  or

- Print the newline character (**\n**) within double quotation marks, which is equivalent to Enter or Return.

Script 1.4 These basic comments demonstrate the three comment syntaxes you can use in PHP.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Comments</title>
6   </head>
7   <body>
8   <?php
9
10  # Script 1.4 - comments.php
11  # Created March 16, 2011
12  # Created by Larry E. Ullman
13  # This script does nothing much.
14
15  echo '<p>This is a line of text.<br>This
    is another line of text.</p>';
16
17  /*
18  echo 'This line will not be
    executed.';
19  */
20
21  echo "<p>Now I'm done.</p>";
    // End of PHP code.
22
23  ?>
24  </body>
25  </html>
```

To comment your scripts:

1. Begin a new PHP document in your text editor or IDE, to be named **comments.php**, starting with the initial HTML (**Script 1.4**):

   ```
   <!doctype html>
   <html lang="en">
   <head>
     <meta charset="utf-8">
     <title>Comments</title>
   </head>
   <body>
   ```

2. Add the initial PHP tag and write your first comments:

   ```
   <?php
   # Script 1.4 - comments.php
   # Created April 23, 2017
   # Created by Larry E. Ullman
   # This script does nothing much.
   ```

   One of the first comments each script should contain is an introductory block that lists creation date, modification date, creator, creator's contact information, purpose of the script, and so on. Some people suggest that the shell-style comments (**#**) stand out more in a script and are therefore best for this kind of notation.

3. Send some HTML to the browser:

   ```
   echo '<p>This is a line of text.
   →<br>This is another line of
   →text.</p>';
   ```

   It doesn't matter what you do here—just make something for the browser to display. For the sake of variety, the **echo** statement will print some HTML tags, including a line break (**
**) to add some spacing to the generated HTML page.

continues on next page

4. Use the multiline comments to comment out a second **echo** statement:

```
/*
echo 'This line will not be
→ executed.';
*/
```

By surrounding any block of PHP code with **/*** and ***/**, you can render that code inert without having to delete it from your script. By later removing the comment tags, you can reactivate that section of PHP code.

5. Add a final comment after a third **echo** statement:

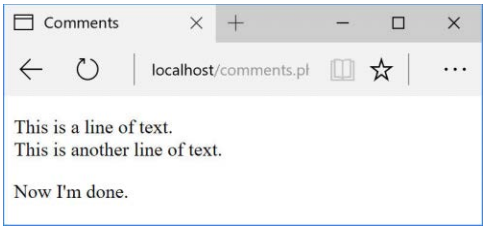
```
echo "<p>Now I'm done.</p>";
→ // End of PHP code.
```

This last (superfluous) comment shows how to place a comment at the end of a line, a common practice. Note that double quotation marks surround this message, since single quotation marks would conflict with the apostrophe (see the "Needing an Escape" sidebar, earlier in the chapter).

6. Close the PHP section and complete the HTML page:

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

7. Save the file as **comments.php**, place it in your web directory, and test it in your browser A.



A The PHP comments in Script 1.4 don't appear in the web page or the HTML source B.

8. If you're the curious type, check the source code in your browser to confirm that the PHP comments do not appear there B.

TIP You shouldn't nest—place one inside another—multiline comments (`/* */`). Doing so will cause problems.

TIP Any of the PHP comments can be used at the end of a line (say, after a function call):

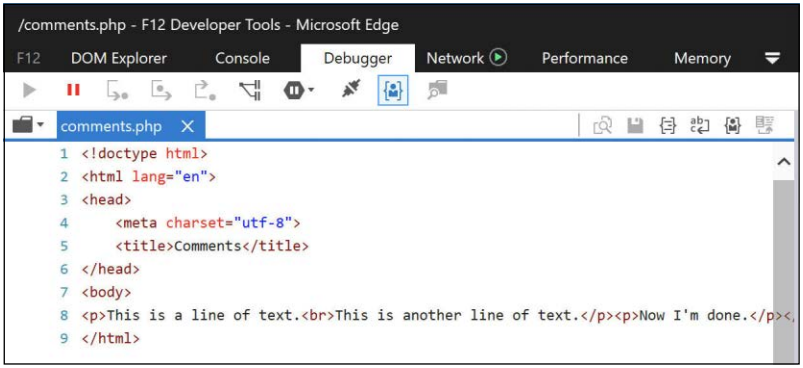
```
echo 'Howdy'; /* Say 'Howdy' */
```

Although this is allowed, it's far less common.

TIP In the interest of saving space, the scripts in this book will not be as well documented as I would suggest they should be.

TIP It's also important that you keep the comments up to date and accurate when you change a script. There's nothing more confusing than a comment that says one thing when the code really does something else.

TIP Some developers argue that it's unnecessary to comment individual bits of code because the code itself should make its purpose clear. In my experience, adding comments helps.



B The PHP comments from Script 1.4 are nowhere to be seen in the client's browser.

What Are Variables?

Variables are containers used to temporarily store values. These values can be numbers, text, or much more complex data. PHP supports eight types of variables. These include four scalar (single-valued) types—*Boolean* (TRUE or FALSE), *integer*, *floating point* (decimals), and *strings* (one or more characters); two nonscalar (multivalued)—*arrays* and *objects*; plus *resources* (which you'll see when interacting with databases) and *NULL* (which is a special type that has no value).

Regardless of what type you are creating, all variable names in PHP follow certain syntactical rules:

- A variable's name must start with a dollar sign (**$**)—for example, **$name**.
- The variable's name can contain a combination of letters, numbers, and the underscore—for example, **$my_report1**.
- The first character after the dollar sign must be either a letter or an underscore (it cannot be a number).
- *Variable names in PHP are case-sensitive!* This is a very important rule. It means that **$name** and **$Name** are different variables.

To begin working with variables, this next script will print out the value of three *predefined variables*. Whereas a standard variable is assigned a value during the execution of a script, a predefined variable will already have a value when the script begins its execution. Most of these predefined variables reflect properties of the server as a whole, such as the operating system in use.

Before getting into this script, there are two more things you should know. First, variables can be assigned values using the equals sign (**=**), also called the *assignment operator*. Second, to display the value of a variable, you can print the variable without quotation marks:

```
print $some_var;
```

Or variables can be printed within *double* quotation marks:

```
print "Hello, $name";
```

You cannot print variables within single quotation marks:

```
print 'Hello, $name';
→// This won't work!
```

To use variables:

1. Begin a new PHP document in your text editor or IDE, to be named **predefined.php**, starting with the initial HTML (**Script 1.5**):

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Predefined Variables</
title>
</head>
<body>
```

2. Add the opening PHP tag and the first comment:

```
<?php # Script 1.5 - predefined.php
```

From here on out, scripts will no longer comment on the creator, creation date, and so forth, although you should continue to document your scripts thoroughly. Scripts will, however, make a comment indicating the script's number and filename for ease of cross-referencing (both in the book and when you download them from the book's supporting web site, **LarryUllman.com**).

continues on next page

Script 1.5 This script prints three of PHP's many predefined variables.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Predefined Variables</title>
6   </head>
7   <body>
8   <?php # Script 1.5 - predefined.php
9
10  // Create a shorthand version of the variable names:
11  $file = $_SERVER['SCRIPT_FILENAME'];
12  $user = $_SERVER['HTTP_USER_AGENT'];
13  $server = $_SERVER['SERVER_SOFTWARE'];
14
15  // Print the name of this script:
16  echo "<p>You are running the file:<br><strong>$file</strong>.</p>\n";
17
18  // Print the user's information:
19  echo "<p>You are viewing this page using:<br><strong>$user</strong></p>\n";
20
21  // Print the server's information:
22  echo "<p>This server is running:<br><strong>$server</strong>.</p>\n";
23
24  ?>
25  </body>
26  </html>
```

3. Create a shorthand version of the first variable to be used in this script:

```
$file = $_SERVER['SCRIPT_FILENAME'];
```

This script will use three variables, each of which comes from the larger predefined **$_SERVER** variable. **$_SERVER** refers to a mass of server-related information. The first variable the script uses is **$_SERVER['SCRIPT_FILENAME']**. This variable stores the full path and name of the script being run (for example, **C:\Program Files\Apache\htdocs\predefined.php**).

The value stored in **$_SERVER['SCRIPT_FILENAME']** will be assigned to the new variable **$file**. Creating new variables with shorter names and then assigning them values from **$_SERVER** will make it easier to refer to the variables when printing them. (It also gets around another issue you'll learn about in due time.)

4. Create a shorthand version of two more variables:

```
$user = $_SERVER
→ ['HTTP_USER_AGENT'];
$server = $_SERVER
→ ['SERVER_SOFTWARE'];
```

$_SERVER['HTTP_USER_AGENT'] represents the browser and operating system of the user accessing the script. This value is assigned to **$user**.

$_SERVER['SERVER_SOFTWARE'] represents the web application on the server that's running PHP (e.g., Apache, Abyss, Xitami, or IIS). This is the program that must be installed (see Appendix A) in order to run PHP scripts on that computer.

5. Print out the name of the script being run:

```
echo "<p>You are running the
→ file:<br /><strong>$file
→ </strong>.</p>\n";
```

The first variable to be printed is **$file**. Notice that this variable must be used within double quotation marks and that the statement also makes use of the PHP newline character (**\n**), which will add a line break in the generated HTML source. Some basic HTML tags—paragraph and strong—are added to give the generated page a bit of flair.

6. Print out the information of the user accessing the script:

```
echo "<p>You are viewing this page
→ using:<br><strong>$user</strong>
→ </p>\n";
```

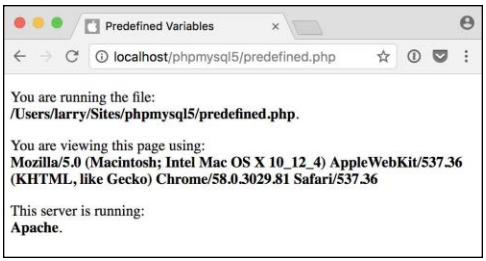
This line prints the second variable, **$user**. To repeat what's said in the fourth step, **$user** correlates to **$_SERVER['HTTP_USER_AGENT']** and refers to the operating system, browser type, and browser version being used to access the web page.

7. Print out the server information:

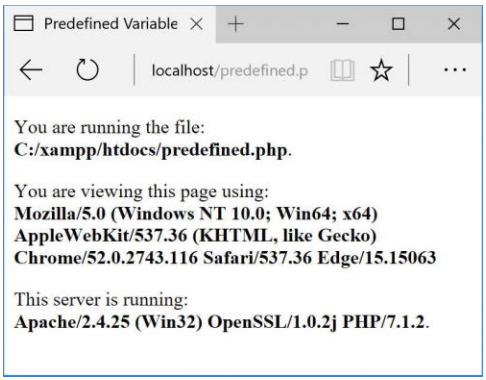
```
echo "<p>This server is running:
→ <br><strong>$server</strong>.
→ </p>\n";
```

8. Complete the PHP block and the HTML page:

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



A The `predefined.php` script reports back to the viewer information about the script, the browser being used to view it, and the server itself.



B This is the book's first truly dynamic script, in that the web page changes depending on the server running it and the browser viewing it (compare with **A**).

9. Save the file as `predefined.php`, place it in your web directory, and test it in your browser **A**.

TIP **If you have problems with this, or any other script, turn to the book's corresponding forum (`LarryUllman.com/forums/`) for assistance.**

TIP **If possible, run this script using a different browser and/or on another server B.**

TIP **Variable names cannot contain spaces. The underscore is commonly used in lieu of a space.**

TIP **The most important consideration when creating variables is to use a consistent naming scheme. In this book you'll see that I use all-lowercase letters for my variable names, with underscores separating words (`$first_name`). Some programmers prefer to use capitalization instead: `$FirstName` (known as "camel-case" style).**

TIP **PHP is very casual in how it treats variables, meaning that you don't need to initialize them (set an immediate value) or declare them (set a specific type), and you can convert a variable among the many types without problem.**

Introducing Strings

Now that you've been introduced to the general concept of variables, let's look at variables in detail. The first variable type to delve into is the *string*. A string is merely a quoted chunk of characters: letters, numbers, spaces, punctuation, and so forth. These are all strings:

- 'Tobias'
- "In watermelon sugar"
- '100'
- 'August 2, 2017'

To make a string variable, assign a string value to a valid variable name:

```
$first_name = 'Tobias';
$today = 'August 2, 2011';
```

When creating strings, you can use either single or double quotation marks to encapsulate the characters, just as you would when printing text. Likewise, you must use the same type of quotation mark for the beginning and the end of the string. If that same mark appears within the string, it must be escaped:

```
$var = "Define \"platitude\", please.";
```

Or you can instead use the other quotation mark type:

```
$var = 'Define "platitude", please.';
```

To print out the value of a string, use either **`echo`** or **`print`**:

```
echo $first_name;
```

To print the value of string within a context, you must use double quotation marks:

```
echo "Hello, $first_name";
```

You've already worked with strings once—when using the predefined variables in the preceding section, as the values of those variables happened to be strings. In this next example, you'll create and use your own strings.

Script 1.6 String variables are created and their values are sent to the browser in this script.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Strings</title>
6   </head>
7   <body>
8   <?php # Script 1.6 - strings.php
9
10  // Create the variables:
11  $first_name = 'Haruki';
12  $last_name = 'Murakami';
13  $book = 'Kafka on the Shore';
14
15  // Print the values:
16  echo "<p>The book <em>$book</em> was
    written by $first_name
    $last_name.</p>";
17
18  ?>
19  </body>
20  </html>
```

To use strings:

1. Begin a new PHP document in your text editor or IDE, to be named **strings.php**, starting with the initial HTML and including the opening PHP tag (**Script 1.6**):

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Strings</title>
</head>
<body>
<?php # Script 1.6 - strings.php
```

2. Within the PHP tags, create three variables:

```
$first_name = 'Haruki';
$last_name = 'Murakami';
$book = 'Kafka on the Shore';
```

   This rudimentary example creates **$first_name**, **$last_name**, and **$book** variables that will then be printed out in a message.

continues on next page

3. Add an **echo** statement:

```
echo "<p>The book <em>$book
→</em> was written by
→$first_name $last_name.</p>";
```

All this script does is print a statement of authorship based on three established variables. A little HTML formatting—the emphasis on the book's title—is thrown in to make it more attractive. Remember to use double quotation marks here for the variable values to be printed out appropriately (more on the importance of double quotation marks at this chapter's end).

4. Complete the PHP block and the HTML page:

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

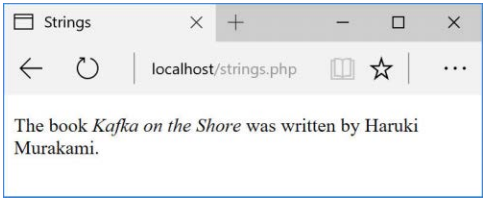
5. Save the file as **strings.php**, place it in your web directory, and test it in your browser Ⓐ.

6. If desired, change the values of the three variables, save the file, and run the script again Ⓑ.

TIP If you assign another value to an existing variable (e.g., $book), the new value will overwrite the old one. For example:

```
$book = 'High Fidelity';
$book = 'The Corrections';
/* $book now has a value of
'The Corrections'. */
```

TIP PHP has no set limits on how big a string can be. It's theoretically possible that you'll be limited by the resources of the server, but it's doubtful that you'll ever encounter such a problem.



Ⓐ The resulting web page is based on printing out the values of three variables.

Ⓑ The output of the script is changed by altering the variables in it.

Script 1.7 Concatenation gives you the ability to append more characters onto a string.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Concatenation</title>
6   </head>
7   <body>
8   <?php # Script 1.7 - concat.php
9
10  // Create the variables:
11  $first_name = 'Melissa';
12  $last_name = 'Bank';
13  $author = $first_name . ' ' .
    $last_name;
14
15  $book = 'The Girls\' Guide to Hunting
    and Fishing';
16
17  //Print the values:
18  echo "<p>The book <em>$book</em> was
    written by $author.</p>";
19
20  ?>
21  </body>
22  </html>
```

Concatenating Strings

Concatenation is like addition for strings, whereby characters are added to the end of the string. It is performed using the *concatenation operator*, which is the period (**.**):

```
$city= 'Seattle';
$state = 'Washington';
$address = $city . $state;
```

The **$address** variable now has the value *SeattleWashington*, which almost achieves the desired result (*Seattle, Washington*). To improve upon this, you could write

```
$address = $city . ', ' . $state;
```

so that a comma and a space are concatenated to the variables as well.

Because of how liberally PHP treats variables, concatenation is possible with strings and numbers. Either of these statements will produce the same result (*Seattle, Washington 98101*):

```
$address = $city . ', ' . $state .
 ' 98101';
$address = $city . ', ' . $state .
 ' ' . 98101;
```

Let's modify **strings.php** to use this new operator.

To use concatenation:

1. Open **strings.php** (refer to Script 1.6) in your text editor or IDE.
2. After you've established the **$first_name** and **$last_name** variables (lines 11 and 12), add this line (**Script 1.7**):

   ```
   $author = $first_name . ' ' .
   → $last_name;
   ```

   As a demonstration of concatenation, a new variable—**$author**—will be created as the concatenation of two existing strings and a space in between.

continues on next page

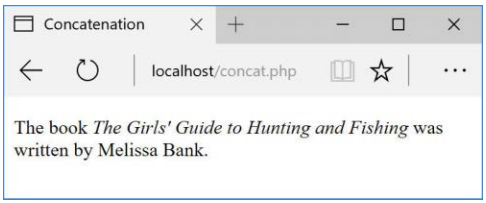
3. Change the **echo** statement to use this new variable:

```
echo "<p>The book <em>$book</em>
→ was written by $author.</p>";
```

   Since the two variables have been turned into one, the **echo** statement should be altered accordingly.

4. If desired, change the HTML page title and the values of the first name, last name, and book variables.

5. Save the file as **concat.php**, place it in your web directory, and test it in your browser Ⓐ.



Ⓐ In this revised script, the end result of concatenation is not apparent to the user.

TIP **PHP has a slew of useful string-specific functions, which you'll see over the course of this book. For example, to calculate how long a string is (how many characters it contains), use `strlen()`:**

```
$num = strlen('some string'); // 11
```

TIP **You can have PHP convert the case of strings with `strtolower()`, which makes it entirely lowercase; `strtoupper()`, which makes it entirely uppercase; `ucfirst()`, which capitalizes the first character; and `ucwords()`, which capitalizes the first character of every word.**

TIP **If you are merely concatenating one value to another, you can use the concatenation assignment operator (`.=`). The following are equivalent:**

```
$title = $title . $subtitle;
$title .= $subtitle;
```

TIP **The initial example in this section could be rewritten using either**

```
$address = "$city, $state";
```

or

```
$address = $city;
$address .= ',';
$address .= $state;
```

Using the PHP Manual

The PHP manual—accessible online at **www.php.net/manual**—lists every function and feature of the language. The manual is organized with general concepts (installation, syntax, variables) discussed first and ends with the functions by topic (MySQL, string functions, and so on).

To quickly look up any function in the PHP manual, go to **php.net/functionname** in your browser (for example, **php.net/print**). For each function, the manual indicates the following:

- The versions of PHP the function is available in
- How many and what types of arguments the function takes (optional arguments are wrapped in square brackets)
- What type of value the function returns

The manual also contains a description of the function.

You should be in the habit of checking out the PHP manual whenever you're confused by a function or how it's properly used, or need to learn more about any feature of the language. It's also critically important that you know what version of PHP you're running, since functions and other particulars of PHP do change over time.

Introducing Numbers

In introducing variables, I stated that PHP has both integer and floating-point (decimal) number types. In my experience, though, these two types can be classified under the generic title *numbers* without losing much valuable distinction. Valid numbers in PHP can be anything like

- 8
- 3.14
- 10980843985
- –4.2398508
- 4.4e2

Notice that these values are never quoted—quoted numbers are strings with numeric values—nor do they include commas to indicate thousands. Also, a number is assumed to be positive unless it is preceded by the minus sign (**–**).

Along with the standard arithmetic operators you can use on numbers (Table 1.1), dozens of functions are built into PHP. Two common ones are **`round()`** and **`number_format()`**. The former rounds a decimal to the nearest integer:

```
$n = 3.14;
$n = round($n); // 3
```

It can also round to a specified number of decimal places:

```
$n = 3.141592;
$n = round($n, 3); // 3.142
```

The **`number_format()`** function turns a number into the more commonly written version, grouped into thousands using commas:

```
$n = 20943;
$n = number_format($n); // 20,943
```

This function can also set a specified number of decimal points:

```
$n = 20943;
$n = number_format($n, 2); //
20,943.00
```

To practice with numbers, let's write a mock-up script that performs the calculations you might use in an e-commerce shopping cart.

TABLE 1.1 Arithmetic Operators

Operator	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus
+ +	Increment
––	Decrement

To use numbers:

1. Begin a new PHP document in your text editor or IDE, to be named **numbers.php** (Script 1.8):

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Numbers</title>
</head>
<body>
<?php # Script 1.8 - numbers.php
```

2. Establish the requisite variables:

```
$quantity = 30;
$price = 119.95;
$taxrate = .05;
```

   This script will use three hard-coded variables on which calculations will be made. Later in the book, you'll see how these values can be dynamically determined (i.e., by user interaction with an HTML form).

3. Perform the calculations:

```
$total = $quantity * $price;
$total = $total + ($total *
→$taxrate);
```

   The first line establishes the order total as the number of widgets purchased multiplied by the price of each widget. The second line then adds the amount of tax to the total (calculated by multiplying the tax rate by the total).

4. Format the total:

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

   The **number_format()** function will group the total into thousands and round it to two decimal places. Applying this function will properly format the calculated value.

Script 1.8 The **numbers.php** script performs basic mathematical calculations, like those used in an e-commerce application.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Numbers</title>
6   </head>
7   <body>
8   <?php # Script 1.8 - numbers.php
9
10  // Set the variables:
11  $quantity = 30; // Buying 30 widgets.
12  $price = 119.95;
13  $taxrate = .05; // 5% sales tax.
14
15  // Calculate the total:
16  $total = $quantity * $price;
17  $total = $total + ($total * $taxrate);
    // Calculate and add the tax.
18
19  // Format the total:
20  $total = number_format ($total, 2);
21
22  // Print the results:
23  echo '<p>You are purchasing <strong>' .
    $quantity . '</strong> widget(s) at a
    cost of <strong>$' . $price . '</strong>
    each. With tax, the total comes to
    <strong>$' . $total . '</strong>.</p>';
24
25  ?>
26  </body>
27  </html>
```

5. Print the results:

```
echo '<p>You are purchasing
→<strong>' . $quantity .
→'</strong> widget(s) at a cost
→of <strong>$' . $price .
→'</strong> each. With tax, the
→total comes to <strong>$' .
→$total . '</strong>.</p>';
```

The last step in the script is to print out the results. The `echo` statement uses both single-quoted text and concatenated variables in order to print out the full combination of HTML, dollar signs, and variable values. You'll see an alternative approach in the last example of this chapter.

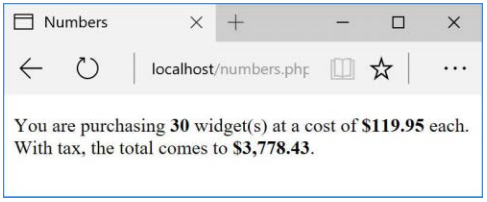
6. Complete the PHP code and the HTML page:

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

7. Save the file as `numbers.php`, place it in your web directory, and test it in your browser Ⓐ.

8. If desired, change the initial three variables and rerun the script Ⓑ.

TIP PHP supports a maximum integer of around two billion on most platforms. With numbers larger than that, PHP will automatically use a floating-point type.



Ⓐ The numbers PHP page (Script 1.8) performs calculations based on set values.

TIP When dealing with arithmetic, the issue of precedence arises—the order in which complex calculations are made. While the PHP manual and other sources tend to list the hierarchy of precedence, I find programming to be safer and more legible when I group clauses in parentheses to force the execution order (see line 17 of Script 1.8).

TIP Computers are notoriously poor at dealing with decimals. For example, the number 2.0 may actually be stored as 1.99999. Most of the time this won't be a problem, but in cases where mathematical precision is paramount, rely on integers, not decimals. The PHP manual has information on this subject, as well as alternative functions for improving computational accuracy.

TIP Many of the mathematical operators also have a corresponding assignment operator, letting you create a shorthand for assigning values. The line

```
$total = $total + ($total * $taxrate);
```

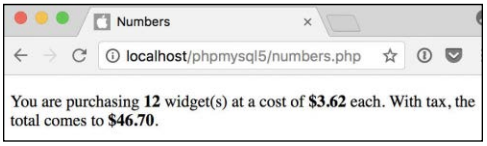
could be rewritten as

```
$total += ($total * $taxrate);
```

TIP If you set a `$price` value without using two decimals (e.g., 119.9 or 34), you would want to apply `number_format()` to `$price` before printing it.

TIP New in PHP 7 is the `intdiv()` function, which returns the integer quotient of a division:

```
echo intdiv(7, 3); // 2
```



Ⓑ To change the generated web page, alter any or all of the three variables (compare with Ⓐ).

Introducing Constants

Constants, like variables, are used to temporarily store a value, but otherwise, constants and variables differ in many ways. For starters, to create a constant, you use the **`define()`** function instead of the assignment operator **(=)**:

```
define('NAME', value);
```

Notice that, as a rule of thumb, constants are named using all capitals, although this is not required. Most importantly, constants do not use the initial dollar sign as variables do (because constants are not variables).

A constant is normally assigned a *scalar* value, like a string or a number:

```
define('USERNAME', 'troutocity');
define('PI', 3.14);
```

And unlike variables, a constant's value cannot be changed.

To access a constant's value, like when you want to print it, you cannot put the constant within quotation marks:

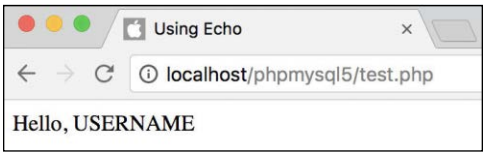
```
echo "Hello, USERNAME"; // Won't work!
```

With that code, PHP literally prints *Hello, USERNAME* Ⓐ and not the value of the **`USERNAME`** constant because there's no indication that **`USERNAME`** is anything other than literal text. Instead, either print the constant by itself:

```
echo 'Hello, ';
echo USERNAME;
```

or use the concatenation operator:

```
echo 'Hello, ' . USERNAME;
```



Ⓐ Constants cannot be placed within quoted strings.

PHP runs with several predefined constants, much like the predefined variables used earlier in the chapter. These include **PHP_VERSION** (the version of PHP running) and **PHP_OS** (the operating system of the server). This next script will print those two values, along with the value of a user-defined constant.

To use constants:

1. Begin a new PHP document in your text editor or IDE, to be named **constants.php** (Script 1.9).

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Constants</title>
</head>
<body>
<?php # Script 1.9 - constants.php
```

2. Create a new date constant:

```
define('TODAY', 'April 23, 2017');
```

An admittedly trivial use of constants, but this example will illustrate the point. In Chapter 9, "Using PHP with MySQL," you'll see how to use constants to store your database access information.

3. Print out the date, the PHP version, and operating system information:

```
echo '<p>Today is ' . TODAY .
→'.<br>This server is running
→version <strong>' . PHP_VERSION .
→'</strong> of PHP on the
→<strong>' . PHP_OS . '</strong>
→operating system.</p>';
```

Since constants cannot be printed within quotation marks, use the concatenation operator in the **echo** statement.

continues on next page

Script 1.9 Constants are another temporary storage tool you can use in PHP, distinct from variables.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Constants</title>
6   </head>
7   <body>
8   <?php # Script 1.9 - constants.php
9
10  // Set today's date as a constant:
11  define('TODAY', 'April 23, 2017');
12
13  // Print a message, using predefined constants and the TODAY constant:
14  echo '<p>Today is ' . TODAY . '.<br>This server is running version <strong>' .
    PHP_VERSION . '</strong> of PHP on the <strong>' . PHP_OS . '</strong> operating
    system.</p>';
15
16  ?>
17  </body>
18  </html>
```

4. Complete the PHP code and the HTML page:

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

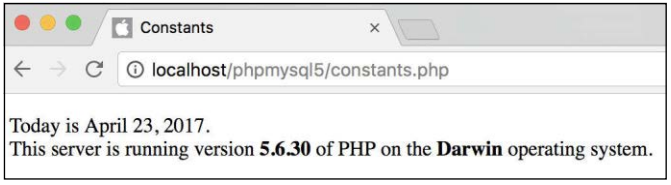
5. Save the file as **constants.php**, place it in your web directory, and test it in your browser B.

TIP If possible, run this script on another PHP-enabled server C.

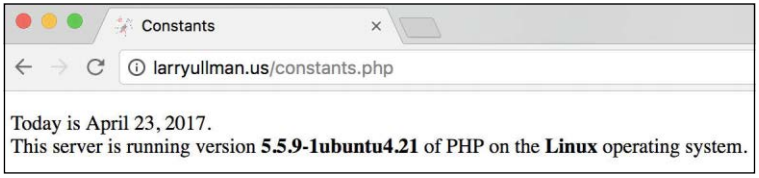
TIP The operating system called Darwin B is the technical name for macOS.

TIP In Chapter 12, "Cookies and Sessions," you'll learn about another constant, SID (which stands for session ID).

TIP As of PHP 7, you can now create an array constant. You'll learn more about arrays in Chapter 2, "Programming with PHP."



B By making use of PHP's constants, you can learn more about your PHP setup.



C Running the same script (refer to Script 1.9) on different servers garners different results.

Single vs. Double Quotation Marks

In PHP, it's important to understand how single quotation marks differ from double quotation marks. With **echo** and **print**, or when assigning values to strings, you can use either, as in the examples used so far. But there is a key difference between the two types of quotation marks and when you should use which. You've seen this difference already, but it's an important enough concept to merit more discussion.

In PHP, *values enclosed within single quotation marks will be treated literally*, whereas *those within double quotation marks will be interpreted*. In other words, placing variables and special characters (**Table 1.2**) within double quotes will result in their represented values printed, not their literal values. For example, assume that you have

```
$var = 'test';
```

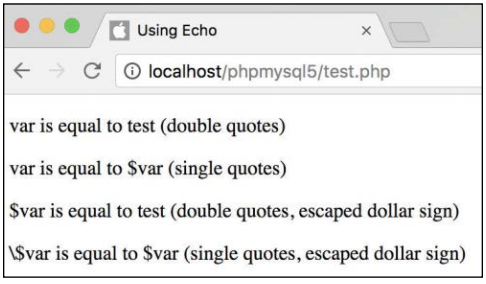
TABLE 1.2 Escape Sequences

Code	Meaning
\"	Double quotation mark
\'	Single quotation mark
\\	Backslash
\n	Newline
\r	Carriage return
\t	Tab
\$	Dollar sign

The code **echo "var is equal to $var";** will print out *var is equal to test*, but the code **echo 'var is equal to $var';** will print out *var is equal to $var*. Using an escaped dollar sign, the **code echo "\$var is equal to $var";** will print out *$var is equal to test*, whereas the code **echo '\$var is equal to $var';** will print out *\$var is equal to $var* Ⓐ.

As these examples should illustrate, double quotation marks will replace a variable's name (**$var**) with its value (*test*) and a special character's code (**\$**) with its represented value (*$*). Single quotes will always display exactly what you type, except for the escaped single quote (**\'**) and the escaped backslash (****), which are printed as a single quotation mark and a single backslash, respectively.

As another example of how the two quotation marks differ, let's modify the **numbers.php** script as an experiment.



Ⓐ How single and double quotation marks affect what gets printed by PHP.

To use single and double quotation marks:

1. Open **numbers.php** (refer to Script 1.8) in your text editor or IDE.
2. Delete the existing **echo** statement (**Script 1.10**).
3. Print a caption and then rewrite the original **echo** statement using double quotation marks:

```
echo "<h3>Using double quotation
→ marks:</h3>";
echo "<p>You are purchasing
→ <strong>$quantity</strong>
→  widget(s) at a cost of
→ <strong>\$$price</strong> each.
→ With tax, the total comes to
→ <strong>\$$total</strong>.</p>\n";
```

In the original script, the results were printed using single quotation marks and concatenation. The same result can be achieved using double quotation marks. When using double quotation marks, the variables can be placed within the string.

There is one catch, though: trying to print a dollar amount as $12.34 (where 12.34 comes from a variable) would suggest that you would code **$$var**. That will not work (for complicated reasons). Instead, escape the initial dollar sign, resulting in **\$$var**, as you see twice in this code. The first dollar sign will be printed, and the second becomes the start of the variable name.

Script 1.10 This, the final script in the chapter, demonstrates the differences between using single and double quotation marks.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Quotation Marks</title>
6   </head>
7   <body>
8   <?php # Script 1.10 - quotes.php
9
10  // Set the variables:
11  $quantity = 30; // Buying 30 widgets.
12  $price = 119.95;
13  $taxrate = .05; // 5% sales tax.
14
15  // Calculate the total.
16  $total = $quantity * $price;
17  $total = $total + ($total * $taxrate);
    // Calculate and add the tax.
18
19  // Format the total:
20  $total = number_format ($total, 2);
21
22  // Print the results using double
    quotation marks:
23  echo "<h3>Using double quotation
    marks:</h3>";
24  echo "<p>You are purchasing
    <strong>$quantity</strong> widget(s)
    at a cost of <strong>\$$price
    </strong> each. With tax, the total
    comes to <strong>\$$total</strong>.
    </p>\n";
25
26  // Print the results using single
    quotation marks:
27  echo '<h3>Using single quotation
    marks:</h3>';
28  echo '<p>You are purchasing
    <strong>$quantity</strong> widget(s)
    at a cost of <strong>\$$price
    </strong> each. With tax, the total
    comes to <strong>\$$total</strong>.
    </p>\n';
29
30  ?>
31  </body>
32  </html>
```

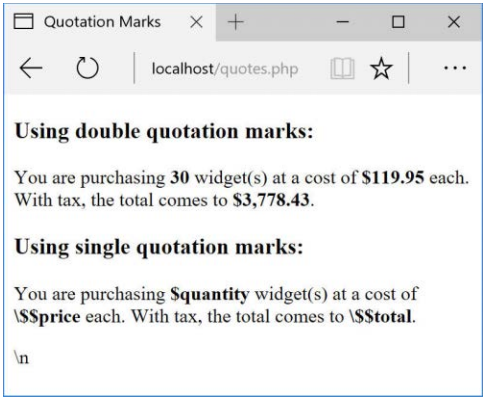
4. Repeat the `echo` statements, this time using single quotation marks:

```
echo '<h3>Using single quotation
→ marks:</h3>';
echo '<p>You are purchasing
→ <strong>$quantity</strong>
→ widget(s) at a cost of
→ <strong>\$$price</strong> each.
→ With tax, the total comes to
→ <strong>\$$total</strong>.</p>\n';
```

   This `echo` statement is used to highlight the difference between using single or double quotation marks. It will not work as desired, and the resulting page will show you exactly what does happen instead.

5. If you want, change the page's title.

6. Save the file as `quotes.php`, place it in your web directory, and test it in your browser B.



B These results demonstrate when and how you'd use one type of quotation mark as opposed to the other.

7. View the source of the web page to see how using the newline character (`\n`) within each quotation mark type also differs.

   You should see that when you place the newline character within double quotation marks it creates a newline in the HTML source. When placed within single quotation marks, the literal characters `\` and `n` are printed instead.

TIP Because PHP will attempt to find variables within double quotation marks, using single quotation marks is theoretically faster. If you need to print the value of a variable, though, you must use double quotation marks.

Because valid HTML often includes a lot of double-quoted attributes, it's often easiest to use single quotation marks when printing HTML with PHP:

```
echo '<table class="data">';
```

If you were to print out this HTML using double quotation marks, you would have to escape all of the double quotation marks in the string:

```
echo "<table class=\"data\">";
```

TIP In newer versions of PHP, you can actually use `$$price` and `$$total` without preceding them with a backslash (thanks to some internal magic). In older versions of PHP, you cannot. To guarantee reliable results, regardless of PHP version, I recommend using the `\$$var` syntax when you need to print a dollar sign immediately followed by the value of a variable.

TIP If you're still unclear as to the difference between the types, use double quotation marks and you're less likely to have problems.

Basic Debugging Steps

Debugging is by no means a simple concept to grasp, and unfortunately, it's one that is only truly mastered by doing. The next 50 pages could be dedicated to the subject and you'd still only be able to pick up a fraction of the debugging skills that you'll eventually acquire and need.

The reason I introduce debugging in this somewhat harrowing way is that it's important not to enter into programming with delusions. Sometimes code won't work as expected, you'll inevitably create careless errors, and some days you'll want to pull your hair out, even when using a comparatively user-friendly language such as PHP. In short, prepare to be perplexed and frustrated at times. I've been coding in PHP since 1999, and occasionally I still get stuck in the programming muck. But debugging is a very important skill to have, and one that you will eventually pick up out of necessity and experience. As you begin your PHP programming adventure, I can offer the following basic but concrete debugging tips.

Note that these are just some general debugging techniques, specifically tailored to the beginning PHP programmer. Chapter 8, "Error Handling and Debugging," goes into other techniques in more detail.

To debug a PHP script:

- Make sure you're always running PHP scripts through a URL!

  This is perhaps the most common beginner's mistake. PHP code must be run through the web server application, which means it must be requested via **`http://something`**. When you see actual PHP code instead of the result of that code's execution, most likely you're not running the PHP script through a URL.

- Know what version of PHP you're running.

  Some problems will arise from the version of PHP in use. Before you ever use any PHP-enabled server, run a **`phpinfo.php`** script (see Appendix A) or reference the **`PHP_VERSION`** constant to confirm the version of PHP in use.

- Make sure **`display_errors`** is on.

  This is a basic PHP configuration setting (also discussed in Appendix A). You can confirm this setting by executing the **`phpinfo()`** function (just use your browser to search for *display_errors* in the resulting page). For security reasons, PHP may not be set to display the errors that occur. If that's the case, you'll end up seeing blank pages when problems occur. To debug most problems, you'll need to see the errors, so turn this setting on while you're learning. You'll find instructions for doing so in Appendix A.

- Check the HTML source code.

  Sometimes the problem is hidden in the HTML source of the page. In fact, sometimes the PHP error message can be hidden there!

- Trust the error message.

  Another very common beginner's mistake is to not fully read or trust the error that PHP reports. Although an error message can often be cryptic and may seem meaningless, it can't be ignored. At the very least, PHP is normally correct as to the line on which the problem can be found. And if you need to relay that error message to someone else (like when you're asking me for help), do include the entire error message!

- Take a break!

  So many of the programming problems I've encountered over the years, and the vast majority of the toughest ones, have been solved by stepping away from the computer for a while. It's easy to get frustrated and confused, and in such situations, any further steps you take are likely to only make matters worse.

Review and Pursue

Each chapter ends with a "Review and Pursue" section where you'll find questions regarding the material just covered and prompts for ways to expand your knowledge and experience on your own. If you have any problems with these sections, either in answering the questions or pursuing your own endeavors, turn to the book's supporting forum (**`LarryUllman.com/forums/`**).

Review

- What tags are used to surround PHP code?
- What extension should a PHP file have?
- What does a page's *encoding* refer to? What impact does the encoding have on the page?
- What PHP functions, or language constructs, can you use to send data to the browser?
- How does using single versus double quotation marks differ in creating or printing strings?
- What does it mean to *escape* a character in a string?
- What are the three comment syntaxes in PHP? Which one can be used over multiple lines?
- What character do all variable names begin with? What characters can come next? What other characters can be used in a variable's name?
- Are variable names case-sensitive or case-insensitive?
- What is the assignment operator?
- How do you create a string variable?
- What is the concatenation operator? What is the concatenation assignment operator?
- How are constants defined and used?

Pursue

- If you don't already know—*for certain*—what version of PHP you're running, check now.
- Look up one of the mentioned string functions in the PHP manual. Then check out some of the other available string functions listed therein.
- Look up one of the mentioned number functions in the PHP manual. Then check out some of the other available number functions listed therein.
- Search the PHP manual for the **`$_SERVER`** variable to see what other information it contains.
- Create a new script, from scratch, that defines and displays the values of some string variables. Use double quotation marks in the **`echo`** or **`print`** statement that outputs the values. For added complexity, include some HTML in the output. Then rewrite the script so that it uses single quotation marks and concatenation instead of double quotation marks.
- Create a new script, from scratch, that defines, manipulates, and displays the values of some numeric variables.

Programming with PHP

Now that you have the fundamentals of the PHP scripting language down, it's time to build on those basics and start truly programming. In this chapter you'll begin creating more elaborate scripts while still learning some of the standard constructs, functions, and syntax of the language.

You'll start by creating an HTML form and then learn how you can use PHP to handle the submitted values. From there, the chapter covers conditionals and the remaining operators (Chapter 1, "Introduction to PHP," presented the assignment, concatenation, and mathematical operators), arrays (another variable type), and one last language construct, loops.

In This Chapter

Creating an HTML Form

Handling an HTML form with PHP is an important process in any dynamic web site. Two steps are involved: first you create the HTML form itself, and then you create the corresponding PHP script that will receive and process the form data.

It is outside the realm of this book to go into HTML forms in any detail, but I will lead you through one quick example so that it may be used throughout the chapter. If you're unfamiliar with the basics of an HTML form, including the various types of elements, see an HTML resource for more information.

An HTML form is created using the **form** tags and various elements for taking input. The **form** tags look like

```
<form action="script.php"
→ method="post">
</form>
```

In terms of PHP, the most important attribute of your **form** tag is **action**, which dictates to which page the form data will be sent. The second attribute—**method**—has its own issues (see the "Choosing a Method" sidebar), but *post* is the value you'll use most frequently.

The different inputs—be they text boxes, radio buttons, select menus, check boxes, etc.—are placed within the opening and closing **form** tags. As you'll see in the next section, what kinds of inputs your form has makes little difference to the PHP script handling it. You should, however, pay attention to the names you give your form inputs—they'll be of critical importance when it comes to your PHP code.

Choosing a Method

The **method** attribute of a form dictates how the data is sent to the handling page. The two options—*get* and *post*—refer to the HTTP (Hypertext Transfer Protocol) method to be used. The **GET** method sends the submitted data to the receiving page as a series of name-value pairs appended to the URL—for example,

```
http://www.example.com/script.php
→ ?name=Homer&gender=M&age=35
```

The benefit of using the **GET** method is that the resulting page can be bookmarked in the user's browser since it's a complete URL. For that matter, you can also click Back in your browser to return to a **GET** page, or reload it without problems, none of which is true for **POST**. But there is a limit in how much data can be transmitted via **GET**, and this method is less secure since the data is visible.

Generally speaking, *GET is used for requesting information*, like a particular record from a database or the results of a search (searches almost always use **GET**). *The POST method is used when an action is expected*: the updating of a database record or the sending of an email. For these reasons I will primarily use **POST** throughout this book, with noted exceptions.

To create an HTML form:

1. Begin a new HTML document in your text editor or IDE, to be named **form.html** (**Script 2.1**):

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Simple HTML Form</title>
  <style type="text/css">
  label {
     font-weight: bold;
     color: #300ACC;
  }
  </style>
</head>
<body>
<!-- Script 2.1 - form.html -->
<form action="handle_form.php"
→ method="post">
```

The document uses the same basic syntax for an HTML page as in the previous chapter. I have added some inline CSS (Cascading Style Sheets) in order to style the form slightly (specifically, making **label** elements bold and blue).

CSS is the preferred way to handle many formatting and layout issues in an HTML page. You'll see a little bit of CSS here and there in this book; if you're not familiar with the subject, check out a dedicated CSS reference.

Finally, an HTML comment indicates the file's name and number.

continues on next page

Script 2.1 This simple HTML form will be used for several of the examples in this chapter.

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4   <meta charset="utf-8">
5   <title>Simple HTML Form</title>
6   <style type="text/css">
7   label {
8      font-weight: bold;
9      color: #300ACC;
10  }
11  </style>
12  </head>
13  <body>
14  <!-- Script 2.1 - form.html -->
15
16  <form action="handle_form.php" method="post">
17
18  <fieldset><legend>Enter your information in the form below:</legend>
19
20  <p><label>Name: <input type="text" name="name" size="20" maxlength="40"></label></p>
21
22  <p><label>Email Address: <input type="email" name="email" size="40" maxlength="60"></label></p>
23
```

code continues on next page

2. Add the initial **form** tag:

```
<form action="handle_form.php"
→ method="post">
```

Since the **action** attribute dictates to which script the form data will go, you should give it an appropriate name (*handle_form* to correspond with this page: **form.html**) and the **.php** extension (since a PHP script will handle this form's data).

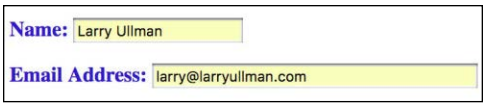
3. Begin the HTML form:

```
<fieldset><legend>Enter your
→ information in the form
→ below:</legend>
```

I'm using the **fieldset** and **legend** HTML tags because they group the form elements nicely (they add a box around the form with a title at the top). This isn't pertinent to the form itself, though.

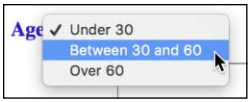
Script 2.1 *continued*

```
24      <p><label for="gender">Gender: </label><input type="radio" name="gender" value="M"> Male
        <input type="radio" name="gender" value="F"> Female</p>
25
26      <p><label>Age:
27      <select name="age">
28         <option value="0-29">Under 30</option>
29         <option value="30-60">Between 30 and 60</option>
30         <option value="60+">Over 60</option>
31      </select></label></p>
32
33      <p><label>Comments: <textarea name="comments" rows="3" cols="40"></textarea></label></p>
34
35      </fieldset>
36
37      <p align="center"><input type="submit" name="submit" value="Submit My Information"></p>
38
39  </form>
40
41  </body>
42  </html>
```



A Two form inputs.

B If multiple radio buttons have the same **name** value, only one can be selected by the user.



C The pull-down menu offers three options, of which only one can be selected (in this example).

4. Add a text and an email input:

```
<p><label>Name: <input type="text"
→ name="name" size="20"
→ maxlength="40"></label></p>
<p><label>Email Address:
→ <input type="email" name="email"
→ size="40" maxlength="60">
→ </label></p>
```

These are just simple text inputs, allowing users to enter their name and email address A. The **label** tags just tie each textual label to the associated element.

5. Add a pair of radio buttons:

```
<p><label for="gender">Gender:
→ </label><input type="radio"
→ name="gender" value="M"> Male
→ <input type="radio"
→ name="gender" value="F">
→ Female</p>
```

The radio buttons B both have the same name, meaning that only one of the two can be selected. They have different values, though.

6. Add a pull-down menu:

```
<p><label>Age:
<select name="age">
  <option value="0-29">Under 30
  → </option>
  <option value="30-60">Between 30
  →  and 60</option>
  <option value="60+">Over 60
  → </option>
</select></label></p>
```

The **select** tag starts the pull-down menu, and then each **option** tag will create another line in the list of choices C.

continues on next page

7. Add a text box for comments:

```
<p><label>Comments: <textarea
→ name="comments" rows="3"
→ cols="40"></textarea></label></p>
```

Textareas are different from text inputs; they are presented as a box D, not as a single line. They allow the user to type much more information and are handy for taking user comments.

8. Complete the form:

```
</fieldset>
<p align="center"><input
→ type="submit" name="submit"
→ value="Submit My Information">
→ </p>
```

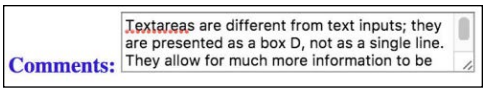
The first tag closes the `fieldset` that was opened in Step 3. Then a submit button is created and centered using a `p` tag. Finally, the form is closed.

9. Complete the HTML page:

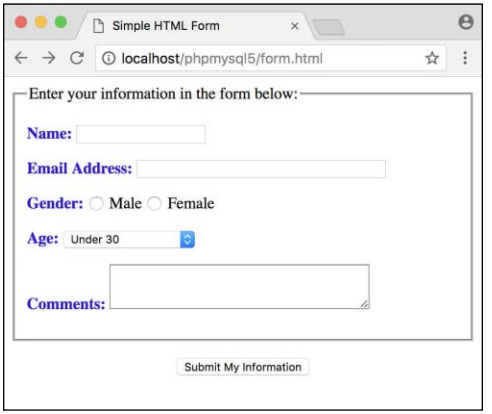
```
</body>
</html>
```

10. Save the file as `form.html`, place it in your web directory, and view it in your browser E.

TIP Since this page contains just HTML, it uses an `.html` extension. It could instead use a `.php` extension without harm (since code outside of the PHP tags is treated as HTML).



D The textarea form element type allows for lots and lots of text.



E The complete form, which requests some basic information from the user.

Handling an HTML Form

Now that the HTML form has been created, it's time to write a bare-bones PHP script to handle it. To say that this script will be *handling* the form means that the PHP page will do something with the data it receives (which is the data the user entered in the form). In this chapter, the scripts will simply print the data back to the browser. In later examples, form data will be stored in a MySQL database, compared against previously stored values, sent in emails, and more.

The beauty of PHP—and what makes it so easy to learn and use—is how well it interacts with HTML forms. PHP scripts store the received information in special variables. For example, say you have a form with an input defined like so:

```
<input type="text" name="city">
```

Whatever the user types into that input will be accessible via a PHP variable named **$_REQUEST['city']**. It is very important that the spelling and capitalization match *exactly*! PHP is case-sensitive when it comes to variable names, so **$_REQUEST['city']** will work, but **$_Request['city']** and **$_REQUEST['City']** will have no value.

This next example will be a PHP script that handles the already-created HTML form (Script 2.1). This script will assign the form data to new variables (to be used as shorthand, just like in Script 1.5, **predefined.php**). The script will then print the received values.

To handle an HTML form:

1. Begin a new PHP document in your text editor or IDE, to be named **handle_form.php**, starting with the HTML (**Script 2.2**):

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Form Feedback</title>
</head>
<body>
```

2. Add the opening PHP tag and create a shorthand version of the form data variables:

```
<?php # Script 2.2 - handle_form.php
$name = $_REQUEST['name'];
$email = $_REQUEST['email'];
$comments = $_REQUEST['comments'];
```

   Following the rules outlined before, the data entered into the first form input, which is called *name*, will be accessible through the variable **$_REQUEST['name']** (**Table 2.1**). The data entered into the email form input, which has a **name** value of *email*, will be accessible through **$_REQUEST['email']**. The same applies to the comments data. Again, the spelling and capitalization of your variables here must exactly match the corresponding **name** values in the HTML form.

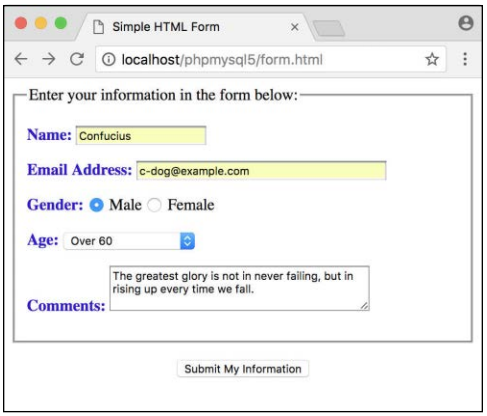
   At this point, you won't make use of the age, gender, and submit form elements.

Script 2.2 This script receives and prints out the information entered into an HTML form (Script 2.1).

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4      <meta charset="utf-8">
5      <title>Form Feedback</title>
6   </head>
7   <body>
8   <?php # Script 2.2 - handle_form.php
9
10  // Create a shorthand for the form data:
11  $name = $_REQUEST['name'];
12  $email = $_REQUEST['email'];
13  $comments = $_REQUEST['comments'];
14  /* Not used:
15  $_REQUEST['age']
16  $_REQUEST['gender']
17  $_REQUEST['submit']
18  */
19
20  // Print the submitted information:
21  echo "<p>Thank you, <strong>
    $name</strong>, for the following
    comments:</p>
22  <pre>$comments</pre>
23  <p>We will reply to you at <em>$email
    </em>.</p>\n";
24
25  ?>
26  </body>
27  </html>
```

TABLE 2.1 Form Elements to PHP Variables

Element Name	Variable Name
name	$_REQUEST['name ']
email	$_REQUEST['email ']
comments	$_REQUEST['comments ']
age	$_REQUEST['age ']
gender	$_REQUEST['gender ']
submit	$_REQUEST['submit']



A To test **handle_form.php**, you must load the form through a URL, then fill it out and submit it.

B The script should display results like this.

C If you see the PHP code after submitting the form, the problem is likely that you did not access the form through a URL.

3. Print out the received name, email, and comments values:

```
echo "<p>Thank you, <strong>
→$name</strong>, for the
→following comments:</p>
<pre>$comments</pre>
<p>We will reply to you at
→<em>$email</em>.</p>\n";
```

The submitted values are simply printed out using the **echo** statement, double quotation marks, and a wee bit of HTML formatting.

4. Complete the page:

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

5. Save the file as **handle_form.php** and place it in the same web directory as **form.html.**

6. Test both documents in your browser by loading **form.html** through a URL (**http://something**) and then filling out A and submitting the form B.

Because the PHP script must be run through a URL (see Chapter 1), the form must also be run through a URL. Otherwise, when you go to submit the form, you'll see PHP code C instead of the proper result B.

TIP **$_REQUEST is a special variable type, known as a superglobal. It stores all of the data sent to a PHP page through either the GET or POST method, as well as data accessible in cookies. Superglobals will be discussed later in the chapter.**

TIP **If you have any problems with this script, apply the debugging techniques suggested in Chapter 1. If you still can't solve the problem, check out the extended debugging techniques listed in Chapter 8, "Error Handling and Debugging." If you're still stymied, turn to the book's supporting forum for assistance (`LarryUllman.com/forums/`).**

TIP **If the PHP script shows blank spaces where a variable's value should have been printed, it means that the variable has no value. The two most likely causes are 1) you failed to enter a value in the form, or 2) you misspelled or mis-capitalized the variable's name.**

TIP **If you see any Undefined variable: variablename errors, this is because the variables you refer to have no value and PHP is set on the highest level of error reporting. The previous tip provides suggestions as to why a variable wouldn't have a value. Chapter 8 discusses error reporting in detail.**

TIP **To see how PHP handles the different form input types, print out the `$_REQUEST['age']` and `$_REQUEST['gender']` values ❶.**

❶ The values of gender and age correspond to those defined in the form's HTML.