

THE EXPERT'S VOICE® IN WEB DEVELOPMENT

THIRD EDITION

PHP Solutions

Dynamic Web Design Made Easy

David Powers

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.



Contents at a Glance

| | |
|--|------|
| About the Author | xv |
| About the Technical Reviewer | xvii |
| Acknowledgments | xix |
| Introduction | xxi |
| ■ Chapter 1: What Is PHP—And Why Should I Care? | 1 |
| ■ Chapter 2: Getting Ready to Work with PHP | 7 |
| ■ Chapter 3: How to Write PHP Scripts..... | 19 |
| ■ Chapter 4: Lightening Your Workload with Includes | 63 |
| ■ Chapter 5: Bringing Forms to Life | 97 |
| ■ Chapter 6: Uploading Files..... | 133 |
| ■ Chapter 7: Using PHP to Manage Files | 173 |
| ■ Chapter 8: Generating Thumbnail Images | 207 |
| ■ Chapter 9: Pages That Remember: Simple Login and Multipage Forms | 235 |
| ■ Chapter 10: Getting Started with a Database | 273 |
| ■ Chapter 11: Connecting to a Database with PHP and SQL..... | 299 |
| ■ Chapter 12: Creating a Dynamic Photo Gallery | 337 |
| ■ Chapter 13: Managing Content | 357 |
| ■ Chapter 14: Formatting Text and Dates | 383 |

■ **Chapter 15: Pulling Data from Multiple Tables** **417**

■ **Chapter 16: Managing Multiple Database Tables** **435**

■ **Chapter 17: Authenticating Users with a Database**..... **465**

Index..... **479**

Introduction

When the first edition of *PHP Solutions* was published, I was concerned that the subtitle, *Dynamic Web Design Made Easy*, sounded overambitious. Even with this third edition, it still makes me a little apprehensive about unduly raising readers' expectations. PHP is not difficult, but nor is it like an instant cake mix: just add water and stir. Every website is different, so it's impossible to grab a script, paste it into a webpage, and expect it to work. My aim was to help web designers with little or no knowledge of programming gain the confidence to dive into the code and adjust it to their own requirements.

The fact that the book has remained so popular since it was first published in 2006 suggests that many readers took up the challenge. Members of Boston PHP did so in large numbers when they adopted the second edition as the text for three series of PHP Percolate, a virtual self-study group for beginners. Hundreds signed up to study the book one chapter a week. It worked for them, so I hope it will work just as well for you.

What's New in this Edition

One useful piece of feedback from PHP Percolate participants and other readers was disappointment when I glossed over a section of advanced code, explaining only what it did rather than how it worked. That omission has been corrected in this edition. Occasionally, I point out that you might want to skip the detailed explanation, but it's there if you're intrigued by how a technique works. As a result, the reference section of Chapter 3 has been expanded to include such esoteric delights as variable variables. No, it's not a typo; "variable variable" is a genuine concept in PHP. It's also quite useful.

This edition brings the content up to date with PHP 5.6, which was released in August 2014. Because hosting companies are often slow to upgrade the version of PHP that they offer, I've made PHP 5.4 the minimum version for the code used in this book. PHP 5.4 made some important changes, introducing a simplified array syntax and dropping support for safe mode and "magic quotes." As well as bringing the code up to date, I've revised every chapter, going through it line by line, clarifying explanations. I've also eliminated a number of errors—without, I hope, introducing new ones.

The biggest changes are to the custom classes for uploading files and creating image thumbnails in Chapters 6 and 8. They now use namespaces to avoid naming clashes with other third-party code. More important, the class definitions have been extensively rewritten to make them more efficient. Another significant change is the use of the new password hashing functions in Chapters 9 and 17. These functions weren't introduced until PHP 5.5, but you can emulate them in PHP 5.4 by including the `password_compat` library in your scripts. Details of how to obtain the library, which consists of a single file, can be found in Chapter 9.

The chapters on working with a database have been reorganized to make them easier to follow. I've also strengthened the explanation of prepared statements, using both MySQL Improved (MySQLi) and the database-neutral PHP Data Objects (PDO). Some Linux distributions now install MariaDB as a drop-in replacement for MySQL. To avoid unnecessary repetition, I normally refer only to MySQL, but all the PHP solutions in this book work equally well with MariaDB.

How This Book Is Organized

Each chapter takes you through a series of stages in a single project, with each stage building on the previous one. By working through each chapter, you get the full picture of how everything fits together. You can later refer to the individual stages to refresh your memory about a particular technique. Although this isn't a reference book, Chapter 3 is a primer on PHP syntax, and some chapters contain short reference sections—notably Chapter 7 (reading from and writing to files), Chapter 9 (sessions), Chapter 10 (data types in MySQL/MariaDB), Chapter 11 (PHP prepared statements), Chapter 13 (the four essential SQL commands), and Chapter 14 (working with dates and times).

So, how easy is easy? I have done my best to ease your path, but there is no magic potion. It requires some effort on your part. Don't attempt to do everything at once. Add dynamic features to your site a few at a time. Get to understand how they work, and your efforts will be amply rewarded. Adding PHP and MySQL/MariaDB to your skills will enable you to build websites that offer much richer content and an interactive user experience.

Using the Example Files

All the files necessary for working through this book can be downloaded from the Apress website at www.apress.com/9781484206362. Make sure you select the download link for *PHP Solutions: Dynamic Web Design Made Easy, Third Edition*. The code is different from the first two editions.

Set up a PHP development environment, as described in Chapter 2. Unzip the files and copy the `phpsols` folder and all its contents into your web server's document root. The code for each chapter is in a folder named after the chapter: `ch01`, `ch02`, and so on. Follow the instructions in each PHP solution, and copy the relevant files to the site root or the work folder indicated.

Where a page undergoes several changes during a chapter, I have numbered the different versions like this: `index_01.php`, `index_02.php`, and so on. When copying a file that has a number, remove the underscore and number from the filename, so `index_01.php` becomes `index.php`. If you are using a program like Dreamweaver that prompts you to update links when moving files from one folder to another, do not update them. The links in the files are designed to pick up the right images and style sheets when located in the target folder. I have done this so you can use a file comparison utility to check your files against mine.

If you don't have a file comparison utility, I strongly urge you to install one. It will save you hours of head scratching when trying to spot the difference between your version and mine. A missing semicolon or mistyped variable can be hard to spot in dozens of lines of code. Windows users can download WinMerge for free from <http://winmerge.org/>. I use Beyond Compare (www.scootersoftware.com), which is now available for Windows, Mac OS X, and Linux. It's not free but is excellent and reasonably priced. BBEdit on a Mac includes a file comparison utility. Alternatively, use the file comparison feature in TextWrangler, which can be downloaded free from www.barebones.com/products/textwrangler/.

Layout Conventions

To keep this book as clear and easy to follow as possible, the following text conventions are used throughout:

Important words or concepts are normally highlighted on the first appearance in **bold type**.

Code is presented in fixed-width font.

New or changed code is normally presented in **bold fixed-width font**.

Pseudo-code and variable input are written in *italic fixed-width font*.

Menu commands are written in the form Menu ► Submenu ► Submenu.

Where I want to draw your attention to something, I've highlighted it, like this:

■ Ahem, don't say I didn't warn you.



What Is PHP—And Why Should I Care?

Officially, PHP stands for PHP: Hypertext Preprocessor. It's an ugly name that gives the impression that it's strictly for nerds or propellerheads. Nothing could be further from the truth. A lighthearted debate on the PHP general mailing list (<http://news.php.net/php.general>) several years ago suggested changing what PHP stands for to Positively Happy People or Pretty Happy Programmers. This book aims to help you put PHP to practical use—and in the process help you understand what makes PHP programmers so happy.

PHP is a scripting language that brings websites to life in the following ways:

- Sends feedback from your website directly to your mailbox
- Uploads files through a webpage
- Generates thumbnails from larger images
- Reads and writes to files
- Displays and updates information dynamically
- Uses a database to display and store information
- Makes websites searchable
- And much more . . .

By reading this book, you'll be able to do all that. PHP is easy to learn; it's platform-neutral, so the same code runs on Windows, Mac OS X, and Linux, and all the software you need to develop with PHP is open source and therefore free. In this chapter, you'll learn about the following:

- How PHP has grown into the most widely used technology for dynamic websites
- How PHP makes webpages dynamic
- How difficult—or easy—PHP is to learn
- Whether PHP is safe
- What software you need in order to write PHP

How PHP Has Grown

PHP is now the most widely used technology for creating dynamic websites, but it started out in 1995 with rather modest ambitions—and a different name. It was originally called Personal Home Page Tools (PHP Tools). One of its main goals was to create a guestbook by gathering information from an online form and displaying it on a webpage. Within three years, it was decided to drop Personal Home Page from the name, because it sounded like something for hobbyists and didn't do justice to the range of sophisticated features that had since been added.

PHP has continued to develop over the years, adding new features all the time. According to W3Techs (<http://w3techs.com/technologies/details/pl-php/all/all>), PHP is used to create dynamic content by more than 80 percent of the 10 million websites it regularly surveys. It's the language that drives highly popular content management systems (CMSs) such as Drupal (<http://drupal.org/>), Joomla! (www.joomla.org), and WordPress (<http://wordpress.org/>). It also runs some of the most heavily used websites, including Facebook (www.facebook.com) and Wikipedia (www.wikipedia.org).

One of the language's great attractions, though, is that it remains true to its roots. PHP's original creator, Rasmus Lerdorf, once described it as "a very programmer-friendly scripting language suitable for people with little or no programming experience as well as the seasoned web developer who needs to get things done quickly." You can start writing useful scripts without needing to learn lots of theory, yet be confident in knowing that you're using a technology with the capability to develop industrial-strength applications.

■ **Note** At the time of this writing, the current version is PHP 5.6. The code assumes you're using a minimum of PHP 5.4, which removed several outdated features, such as "magic quotes." If you have a hosting plan, make sure the server is running at least PHP 5.4.

The next major version of PHP will be called PHP 7. It's been decided to skip PHP 6 to avoid confusion with a version that was abandoned in 2010 for being too ambitious. The emphasis in this book is on code that works *now*, not on what might work at some unspecified time in the future. However, I fully expect that most if not all of the code and techniques will continue to work in PHP 7.

How PHP Makes Pages Dynamic

PHP was originally designed to be embedded in the HTML of a webpage, and that's the way it's often still used. For example, if you want to display the current year in a copyright notice, you could put this in your footer:

```
<p>&copy; <?php echo date('Y'); ?> PHP Solutions</p>
```

On a PHP-enabled web server, the code between the `<?php` and `?>` tags is automatically processed and displays the year like this:

© 2014 PHP Solutions

This is only a trivial example, but it illustrates some of the advantages of using PHP:

- Anyone accessing your site after the stroke of midnight on New Year's Day sees the correct year.
- The date is calculated by the web server so it's not affected if the clock in the user's computer is set incorrectly.

Although it's convenient to embed PHP code in HTML like this, it's repetitive and can lead to mistakes. It can also make your webpages difficult to maintain, particularly once you start using more complex PHP code. Consequently, it's common practice to store a lot of dynamic code in separate files and then use PHP to build your pages from the different components. The separate files—or *include files*, as they're usually called—can contain only PHP, only HTML, or a mixture of both.

As a simple example, you can put your website’s navigation menu in an include file and use PHP to include it in each page. Whenever you need to make any changes to the menu, you edit just one file, the include file, and the changes are automatically reflected in every page that includes the menu. Just imagine how much time that saves on a website with dozens of pages!

With an ordinary HTML page, the content is fixed by the web developer at design time and uploaded to the web server. When somebody visits the page, the web server simply sends the HTML and other assets, such as images and the style sheet. It’s a simple transaction—the request comes from the browser, and the fixed content is sent back by the server. When you build webpages with PHP, much more goes on. Figure 1-1 shows what happens.

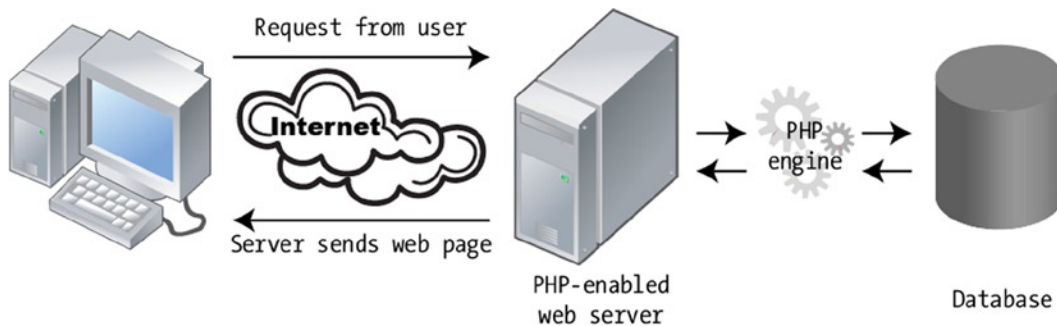


Figure 1-1. The web server builds each PHP page dynamically in response to a request

When a PHP-driven website is visited, it sets in motion the following sequence of events:

1. The browser sends a request to the web server.
2. The web server hands the request to the PHP engine, which is embedded in the server.
3. The PHP engine processes the code. In many cases, it might also query a database before building the page.
4. The server sends the completed page back to the browser.

This process usually takes only a fraction of a second, so the visitor to a PHP website is unlikely to notice any delay. Because each page is built individually, PHP sites can respond to user input, displaying different content when a user logs in or showing the results of a database search.

Creating Pages That Think for Themselves

PHP is a server-side language. The PHP code remains on the web server. After it has been processed, the server sends only the output of the script. Normally, this is HTML, but PHP can also be used to generate other web languages, such as JSON (JavaScript Object Notation).

PHP enables you to introduce logic into your webpages that is based on alternatives. Some decisions are made using information that PHP gleans from the server: the date, the time, the day of the week, information in the page’s URL, and so on. If it’s Wednesday, it will show Wednesday’s TV schedules. At other times, decisions are based on user input, which PHP extracts from online forms. If you have registered with a site, it will display personalized information—that sort of thing.

How Hard Is PHP to Use and Learn?

PHP isn't rocket science, but don't expect to become an expert in five minutes. Perhaps the biggest shock to newcomers is that PHP is far less tolerant of mistakes than browsers are with HTML. If you omit a closing tag in HTML, most browsers will still render the page. If you omit a closing quote, semicolon, or brace in PHP, you'll get an uncompromising error message like the one shown in Figure 1-2. This affects all programming languages, such as JavaScript and C#, not just PHP.



Figure 1-2. Server-side languages like PHP are intolerant of most coding errors

If you're the sort of web designer or developer who uses a visual design tool like Adobe Dreamweaver and never looks at the underlying code, it's time to rethink your approach. Mixing PHP with poorly structured HTML is likely to lead to problems. PHP uses loops to perform repetitive tasks, such as displaying the results of a database search. A loop repeats the same section of code—usually a mixture of PHP and HTML—until all results have been displayed. If you put the loop in the wrong place or if your HTML is badly structured, your page is likely to collapse like a house of cards.

If you're not already in the habit of doing so, it's a good idea to check your pages using the World Wide Web Consortium's (W3C) Markup Validation Service (<http://validator.w3.org/unicorn>).

■ **Note** The W3C is the international body that develops standards such as HTML and CSS to ensure the long-term growth of the web. It's led by the inventor of the World Wide Web, Tim Berners-Lee. To learn about the W3C's mission, see www.w3.org/Consortium/mission.

Can I Just Copy and Paste the Code?

There's nothing wrong with copying the code in this book. That's what it's there for. I've structured this book as a series of practical projects. I explain what the code is for and why it's there. Even if you don't understand exactly how it all works, this should give you sufficient confidence to know which parts of the code to adapt to your own needs and which parts are best left alone. But to get the most out of this book, you need to start experimenting with the tools found in these pages and then come up with your own solutions.

PHP is a toolbox full of powerful features. It has thousands of built-in functions that perform all sorts of tasks, such as converting text to uppercase, generating thumbnail images from full-sized ones, or connecting to a database. The real power comes from combining these functions in different ways and adding your own conditional logic.

How Safe Is PHP?

PHP is like the electricity or kitchen knives in your home: handled properly, it's very safe; handled irresponsibly, it can do a lot of damage. One of the inspirations for the first edition of this book was a spate of attacks that exploited a vulnerability in email scripts, turning websites into spam relays. The solution is quite simple, as you'll learn in Chapter 5, but even a decade later, I still see people using the same insecure techniques, exposing their sites to attack.

PHP is not unsafe, nor does everyone need to become a security expert to use it. What is important is to understand the basic principle of PHP safety: *always check user input before processing it*. You'll find that to be a constant theme throughout this book. Most security risks can be eliminated with very little effort.

The best way to protect yourself is to understand the code you're using.

What Software Do I Need to Write PHP?

Strictly speaking, you don't need any special software to write PHP scripts. PHP code is plain text and can be created in any text editor, such as Notepad on Windows or TextEdit on Mac OS X. Having said that, your life will be a lot easier if you use a program that has features designed to speed up the development process. There are many available—both free and on a paid-for basis.

What to Look for When Choosing a PHP Editor

If there's a mistake in your code, your page will probably never make it as far as the browser, and all you'll see is an error message. You should choose a script editor that has the following features:

- **PHP syntax checking:** This used to be found only in expensive, dedicated programs, but it's now a feature in several free programs. Syntax checkers monitor the code as you type and highlight errors, saving a great deal of time and frustration.
- **PHP syntax coloring:** Code is highlighted in different colors according to the role it plays. If your code is in an unexpected color, it's a sure sign you've made a mistake.
- **PHP code hints:** PHP has so many built-in functions that it can be difficult to remember how to use them, even for an experienced user. Many script editors automatically display tooltips with reminders of how a particular piece of code works.
- **Line numbering:** Finding a specific line quickly makes troubleshooting a lot simpler.
- **A “balance braces” feature:** Parentheses (`()`), square brackets (`[]`), and curly braces (`{}`) must always be in matching pairs. It's easy to forget to close a pair. All good script editors help find the matching parenthesis, bracket, or brace.

The program you're already using to build webpages might already have these features. For example, Adobe Dreamweaver CS5 and later does (www.adobe.com/products/dreamweaver/). It also has embedded PHP documentation.

Even if you don't plan to do a lot of PHP development, you should consider using a dedicated script editor if your web development program doesn't support syntax checking. The following dedicated script editors have all the essential features, such as syntax checking and code hints. It's not an exhaustive list, but rather one based on personal experience.

- **PhpStorm** (www.jetbrains.com/phpstorm/): Although this is a dedicated PHP editing program, it also has excellent support for HTML, CSS, and JavaScript. It's currently my favorite program for developing with PHP.
- **Sublime Text** (www.sublimetext.com/): If you're a Sublime Text fan, there are plug-ins for PHP syntax coloring, syntax checking, and documentation.
- **Zend Studio** (www.zend.com/en/products/studio/): If you're really serious about PHP development, Zend Studio is the most fully featured integrated development environment (IDE) for PHP. It's created by Zend, the company run by leading contributors to the development of PHP. Zend Studio runs on Windows, Mac OS X, and Linux. It used to be expensive, but the price for individual developers is now more affordable, and it includes 12 months of free upgrades and support.
- **PHP Development Tools** (www.eclipse.org/pdt/): PDT is actually a cut-down version of Zend Studio and has the advantage of being free. It runs on Eclipse, the open-source IDE that supports multiple computer languages. If you have used Eclipse for other languages, you should find it relatively easy to use. PDT runs on Windows, Mac OS X, and Linux and is available either as an Eclipse plug-in or as an all-in-one package that automatically installs Eclipse and the PDT plug-in.
- **Komodo Edit** (<http://komodoide.com/komodo-edit/>): This is a free, open-source IDE for PHP and a number of other popular computer languages. It's available for Windows, Mac OS X, and Linux. It's a cut-down version of Komodo IDE, which is a paid-for program with more advanced features.

So, Let's Get on with It . . .

This chapter has provided only a brief overview of what PHP can do to add dynamic features to your websites and what software you need to do so. The first stage in working with PHP is to set up a testing environment. The next chapter covers what you need for both Windows and Mac OS X.



Getting Ready to Work with PHP

Now that you've decided to use PHP to enrich your webpages, you need to make sure that you have everything you need to get on with the rest of this book. Although you can test everything on your remote server, it's usually more convenient to test PHP pages on your local computer. Everything you need to install is free. In this chapter, I'll explain the various options for Windows and Mac OS X. The necessary components are normally installed by default on Linux.

What this chapter covers:

- Checking if your website supports PHP
- Why PHP 5.4 should be the minimum version
- Deciding whether to create a local testing setup
- Using a ready-made package in Windows and Mac OS X
- Where to store your PHP files
- Checking the PHP configuration on your local and remote servers

Checking Whether Your Website Supports PHP

The easiest way to find out whether your website supports PHP is to ask your hosting company. The other way to find out is to upload a PHP page to your website and see if it works. Even if you know that your site supports PHP, do the following test to confirm which version is running:

1. Open a text editor, such as Notepad or TextEdit, and type the following code into a blank page:

```
<?php echo phpversion(); ?>
```
2. Save the file as `phpversion.php`. It's important to make sure that your operating system doesn't add a `.txt` filename extension after the `.php`. Mac users should also make sure that TextEdit doesn't save the file in Rich Text Format (RTF). If you're at all unsure, use `phpversion.php` from the `ch02` folder in the files accompanying this book.
3. Upload `phpversion.php` to your website in the same way you would an HTML page and then type the URL into a browser. Assuming you upload the file to the top level of your site, the URL will be something like <http://www.example.com/phpversion.php>.

If you see a three-part number like **5.6.1** displayed onscreen, you're in business: PHP is enabled. The number tells you which version of PHP is running on your server. *You need a minimum of 5.4.0 to use all the code in this book.*

4. If you get a message that says something like **"Parse error"** it means PHP is supported but that you have made a mistake in typing the code in the file. Use the version in the `ch02` folder instead.
5. If you just see the original code, it means PHP is not supported.

Official support for PHP 5.3 ended in August 2014. If your server is running PHP 5.3 or earlier, contact your host and tell them you want the most recent stable version of PHP. If your host refuses, it's time to change your hosting company.

WHY PHP 5.4 SHOULD BE THE MINIMUM VERSION

As a general principle, PHP tries to preserve backward compatibility between point releases (where only the numbers after the first dot in the version number change). However, a number of outdated features were removed from PHP 5.4. New syntax was also introduced for arrays.

Although most of the code in this book will run correctly on older versions of PHP, you may get unexpected results if you use a server that still relies on those features. The most important changes that affect the code in this book are the removal of safe mode and magic quotes.

Safe mode is often used in shared hosting environments. Among its effects, safe mode restricts where include files can be located and which files can be read from and written to. With the removal of safe mode in PHP 5.4, these restrictions no longer apply.

Magic quotes were a misguided attempt to make PHP safer for inexperienced developers by inserting backslashes before quotes in user-submitted data. The idea was to prevent a malicious attack known as *SQL injection*. Unfortunately, magic quotes caused more problems than they solved, often leaving text peppered with unwanted backslashes. If you run the code in this book on PHP 5.3 or earlier, you'll get unwanted backslashes if magic quotes haven't been disabled.

The code in this book also uses simplified syntax for arrays, which won't work in older versions of PHP.

The most important reason for not using an old version of PHP is security. When vulnerabilities are discovered, security updates are made only to the current and two previous versions. At the time of this writing, the current version is PHP 5.6. That means PHP 5.4 and 5.5 will benefit from any security updates. But as soon as the next version comes out, PHP 5.4 will cease being patched for security threats. Using an up-to-date version of PHP isn't simply a matter of gaining access to the latest features; it helps protect your website and valuable data from malicious attacks.

Deciding Where to Test Your Pages

Unlike ordinary webpages, you can't just double-click PHP pages in Windows File Explorer or Finder on a Mac and view them in your browser. They need to be **parsed**, or processed, through a web server that supports PHP. If your hosting company supports PHP, you can upload your files to your website and test them there. However, you need to upload the file every time you make a change. In the early days, you'll probably find you have to do this often because of some minor mistake in your code. As you become more experienced, you'll still need to upload files frequently because you'll want to experiment with different ideas.

If you want to get working with PHP straight away, by all means use your own website as a test bed. However, you'll soon discover the need for a local PHP test environment. The rest of this chapter is devoted to showing you how to do this, with instructions for both Windows and Mac OS X.

What You Need for a Local Test Environment

To test PHP pages on your local computer, you need to install the following:

- A web server: this is a piece of software that displays webpages, not a separate computer
- PHP
- MySQL and a web-based front end for MySQL called phpMyAdmin, which are required in order to work with a database

■ **Tip** Some Linux distributions install MariaDB (<https://mariadb.org/>) as a drop-in replacement for MySQL. The code in this book is fully compatible with MariaDB.

All the software you need is free. The only cost to you is the time it takes to download the necessary files, plus, of course, the time to make sure everything is set up correctly. In most cases, you should be up and running in less than an hour, probably considerably less. As long as you have at least 1GB of free disk space, you should be able to install all the software on your computer—even one with modest specifications.

■ **Tip** If you already have a PHP test environment on your local computer, there's no need to reinstall. Just check the section at the end of this chapter titled "Checking Your PHP Settings".

Individual Programs or an All-in-one Package?

For many years, I advocated installing each component of a PHP testing environment separately, rather than using a package that installs Apache, PHP, MySQL, and phpMyAdmin in a single operation. My advice was based on the dubious quality of some early all-in-one packages, which installed easily but were next to impossible to uninstall or upgrade. However, the all-in-one packages currently available are excellent, and I have no hesitation in now recommending them.

On my computers, I use XAMPP for Windows (www.apachefriends.org/index.html) and MAMP for Mac OS X (www.mamp.info/en/). Other packages are available; it doesn't matter which you choose.

■ **Tip** Setting up a PHP testing environment with an all-in-one package is normally trouble free. The main cause of difficulty is a conflict with another program using port 80, which the web server uses to listen for page requests. If Skype is installed, go to Tools ► Options ► Advanced ► Connection and make sure that port 80 is not being used for incoming connections. Try port 33087 instead.

Setting Up on Windows

Make sure that you're logged on as an administrator before proceeding.

Getting Windows to Display Filename Extensions

By default, most Windows computers hide the three- or four-letter filename extension, such as `.doc` or `.html`, so all you see in dialog boxes and Windows File Explorer is `thisfile` instead of `thisfile.doc` or `thisfile.html`. Windows 8 does display the filename extension for PHP files, but it's useful to turn on the display of filename extension for all files. In Windows 7, it's essential for working with PHP.

Use these instructions to enable the display of filename extensions in Windows 8:

1. Open **File Explorer**.
2. Select **View** to expand the ribbon at the top of the **File Explorer** window.
3. Select the “**Filename extensions**” check box.

Use these instructions in Windows 7:

4. Open **Start ► Computer**.
5. Select **Organize ► Folder** and then **Search Options**.
6. In the dialog box that opens, select the **View** tab.
7. In the **Advanced Settings** section, uncheck the box marked “**Hide extensions for known file types.**”
8. Click **OK**.

Displaying filename extensions is more secure—you can tell if a virus writer has attached an `.exe` or `.scr` executable file to an innocent-looking document.

Choosing a Web Server

Most PHP installations run on the Apache web server. Both are open source and work well together. However, Windows has its own web server, Internet Information Services (IIS), which also supports PHP. Microsoft has worked closely with the PHP development team to improve the performance of PHP on IIS to roughly the same level as Apache. So, which should you choose?

The answer depends on whether you develop webpages using ASP or ASP.NET, or intend to do so. ASP and ASP.NET require IIS. You can install Apache on the same computer as IIS, but they both listen for requests on port 80. You can't run both servers simultaneously on the same port.

Unless you need IIS for ASP or ASP.NET, I recommend that you install Apache, using XAMPP or one of the other all-in-one packages, as described in the next section. If you need to use IIS, the most convenient way to install PHP is to use the Microsoft Web Platform Installer (Web PI), which you can download from www.microsoft.com/web/downloads/platform.aspx.

Installing an All-in-one Package on Windows

There are three popular packages for Windows that install Apache, PHP, MySQL, phpMyAdmin, and several other tools on your computer in a single operation: XAMPP (www.apachefriends.org/index.html), WampServer (www.wampserver.com/en/), and EasyPHP (www.easyphp.org). The installation process normally takes only a few minutes. Once the package has been installed, you might need to change a few settings, as explained later in this chapter.

Versions are liable to change over the lifetime of a printed book, so I won't describe the installation process. Each package has instructions on its website. There are also helpful videos for setting up WampServer and XAMPP in David Gassner's *Installing Apache, MySQL, and PHP* course on lynda.com. Although lynda.com is a subscription service, at the time of this writing all the videos in that course can be viewed free of charge even if you're not a subscriber (www.lynda.com/Apache-HTTP-Server-tutorials/Installing-Apache-MySQL-PHP/77958-2.html).

Setting Up on Mac OS X

The Apache web server and PHP are preinstalled on Mac OS X, but they're not enabled by default. Rather than using the preinstalled versions, I recommend that you use MAMP, which installs Apache, PHP, MySQL, phpMyAdmin, and several other tools in a single operation.

To avoid conflicts with the preinstalled versions of Apache and PHP, MAMP locates all the applications in a dedicated folder on your hard disk. This makes it easier to uninstall everything by simply dragging the MAMP folder to the Trash if you decide you no longer want MAMP on your computer.

Installing MAMP

Before you begin, make sure you're logged in to your computer with administrative privileges.

1. Go to www.mamp.info/en/downloads/ and select the link for **MAMP & MAMP PRO**. This downloads a disk image that contains both the free and paid-for versions of MAMP.
2. When the download completes, launch the disk image. You'll be presented with a license agreement. You must click **Agree** to continue with mounting the disk image.
3. Follow the onscreen instructions.
4. Verify that **MAMP** has been installed in your **Applications** folder.

■ **Note** MAMP automatically installs both the free and paid-for versions in separate folders called MAMP and MAMP PRO. The paid-for version makes it easier to configure PHP and to work with virtual hosts, but the free version is perfectly adequate, especially for beginners. If you want to remove the MAMP PRO folder, don't drag it to the Trash. Open the folder and double-click the MAMP PRO **uninstall** icon. The paid-for version requires both folders.

Testing and configuring MAMP

By default, MAMP uses nonstandard ports for Apache and MySQL. Unless you're using multiple installations of Apache and MySQL, you should change the port settings.

1. Double-click the **MAMP** icon in **Applications/MAMP**. Your default browser should launch and present you with the MAMP welcome page. Note that the URL in the browser address bar begins with `localhost:8888`. The `:8888` indicates that Apache is listening for requests on the nonstandard port 8888.
2. Minimize the browser and locate the MAMP control panel (see Figure 2-1), which should be running on your desktop. The tiny green lights to the right of **Apache Server** and **MySQL Server** indicate that both servers are running.

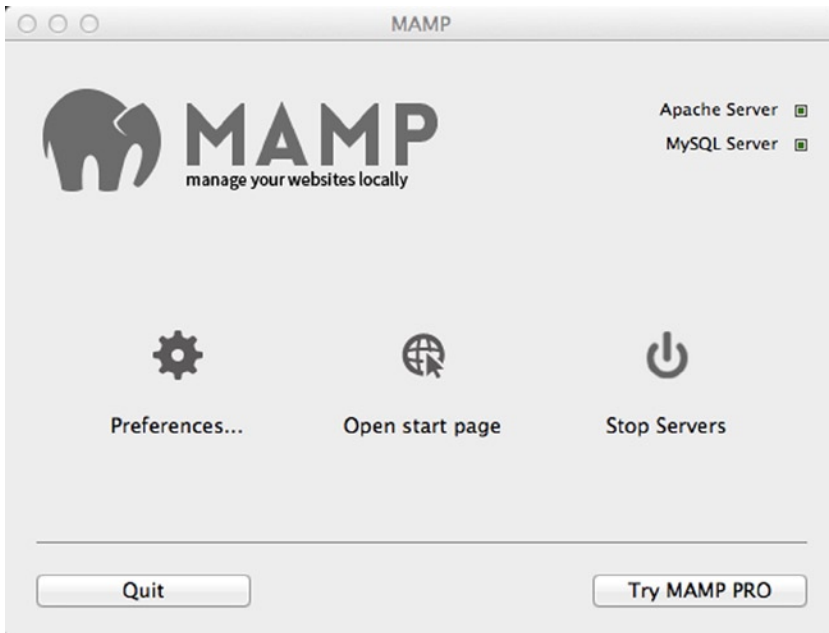


Figure 2-1. The MAMP control panel

3. Click the **Preferences** icon and select **Ports** at the top of the panel that opens. It shows that Apache and MySQL are running on ports 8888 and 8889 (see Figure 2-2).

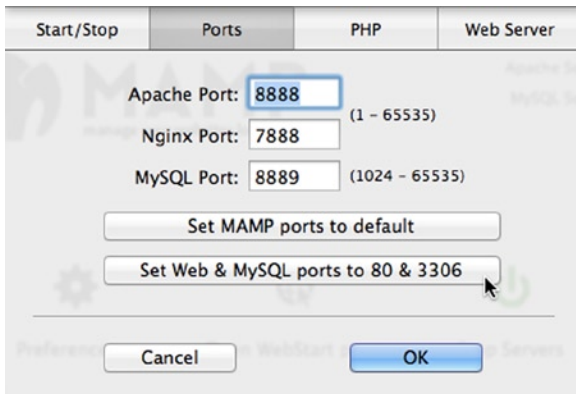


Figure 2-2. Changing the Apache and MySQL ports

4. Click “**Set Web & MySQL ports to 80 & 3306**” as shown in Figure 2-2. The numbers change to the standard ports: 80 for Apache and 3306 for MySQL.

■ **Note** MAMP now supports Nginx as an alternative web server. When I clicked “Set Web & MySQL ports to 80 & 3306,” both Apache Port and Nginx Port changed to 80, which prevented the settings from being accepted. If this happens, manually reset Nginx Port to 7888.

5. Click **OK** and enter your Mac password when prompted. MAMP restarts both servers.

■ **Tip** If any other program is using port 80, Apache won't restart. If you can't find what's preventing Apache from using port 80, open the MAMP preferences panel and click “**Set MAMP ports to default.**”

6. When both lights are green again, click “**Open start page**” in the MAMP Control Panel. This reloads the MAMP welcome page into your browser. This time, the URL shouldn't have a colon followed by a number appearing after localhost because Apache is now listening on the default port.

Where to Locate Your PHP Files (Windows & Mac)

You need to create your files in a location where the web server can process them. Normally, this means that the files should be in the server's document root or in a subfolder of the document root. The default location of the document root for the most common setups is as follows:

- **XAMPP:** C:\xampp\htdocs
- **WampServer:** C:\wamp\www
- **EasyPHP:** C:\EasyPHP\www
- **IIS:** C:\inetpub\wwwroot
- **MAMP:** Macintosh HD:Applications:MAMP:htdocs

To view a PHP page, you need to load it in a browser using a URL. The URL for the web server's document root in your local testing environment is `http://localhost/`.

■ **Caution** If you needed to reset MAMP back to its default ports, you will need to use `http://localhost:8888` instead of `http://localhost`.

If you store the files for this book in a subfolder of the document root called `phpsols`, the URL is `http://localhost/phpsols/` followed by the name of the folder (if any) and file.

■ **Tip** Use `http://127.0.0.1/` if you have problems with `http://localhost/`. 127.0.0.1 is the loopback IP address all computers use to refer to the local machine.

Using Virtual Hosts

The alternative to storing your PHP files in the web server's document root is to use a virtual host. A **virtual host** creates a unique address for each site and is how hosting companies manage shared hosting. MAMP PRO simplifies setting up virtual hosts through its control panel. EasyPHP also has a plug-in module for administering virtual hosts.

Manually setting up virtual hosts involves editing one of your computer's system files to register the host name on your local machine. You also need to tell the web server in your local testing environment where the files are located. The process isn't difficult, but it needs to be done each time you set up a new virtual host.

The advantage of setting up each site in a virtual host is that it matches more accurately the structure of a live website. However, when learning PHP, it's probably more convenient to use a subfolder of your testing server's document root. Once you have gained experience with PHP, you can advance to using virtual hosts. Instructions for manually setting up virtual hosts in Apache are on my website at the following addresses:

- **Windows:** http://foundationphp.com/tutorials/apache_vhosts.php
- **MAMP:** http://foundationphp.com/tutorials/vhosts_mamp.php

■ **Tip** Remember to start the web server in your testing environment to view PHP pages.

Checking Your PHP Settings

After installing PHP, it's a good idea to check its configuration settings. In addition to the core features, PHP has a large number of optional extensions. Both the all-in-one packages and the Microsoft Web PI install all the extensions that you need for this book. However, some of the basic configuration settings might be slightly different. To avoid unexpected problems, adjust your PHP configuration to match the settings recommended in the following pages.

Displaying the Server Configuration with `phpinfo()`

PHP has a built-in command, `phpinfo()`, that displays details of how PHP is configured on the server. The amount of detail produced by `phpinfo()` can feel like massive information overload, but it's invaluable for determining why something works perfectly on your local computer yet not on your live website. The problem usually lies in the remote server having disabled a feature or not having installed an optional extension.

The all-in-one packages make it easy to run `phpinfo()`:

- **XAMPP:** Click the **phpinfo** link in the menu on the left of the XAMPP welcome screen.
- **MAMP:** Click **phpinfo** in the main menu at the top of the MAMP start page.
- **WampServer:** Open the WampServer menu and click **Localhost**. The link for `phpinfo()` is under **Tools**.

Alternatively, create a simple test file and load it in your browser using the following instructions:

1. Make sure that Apache or IIS is running on your local computer.
2. Type the following in a script editor:

```
<?php phpinfo(); ?>
```

There should be nothing else in the file.

3. Save the file as `phpinfo.php` in the server's document root (see "Where to Locate Your PHP Files (Windows and Mac)" earlier in this chapter).

Caution Make sure your editor doesn't add a `.txt` or `.rtf` extension after `.php`.

4. Type `http://localhost/phpinfo.php` in your browser address bar and press Enter.
5. You should see a page similar to that in Figure 2-3 displaying the version of PHP followed by extensive details of your PHP configuration.


| PHP Version 5.6.1  | |
|---|---|
| System | Darwin davids-air.home 13.4.0 Darwin Kernel Version 13.4.0: Sun Aug 17 19:50:11 PDT 2014; root:xnu-2422.115.4~1/RELEASE_X86_64 x86_64 |
| Build Date | Oct 13 2014 18:05:43 |
| Configure Command | './configure' '--with-mysql=/Applications/MAMP/Library' '--with-apxs2=/Applications/MAMP/Library/bin/apxs' '--with-gd' '--with-jpeg-dir=/Applications/MAMP/Library' '--with-png-dir=/Applications/MAMP/Library' '--with-zlib' '--with-zlib-dir=/Applications/MAMP/Library' '--with-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php/php5.6.1' '--exec-prefix=/Applications/MAMP/bin/php/php5.6.1' '--sysconfdir=/Applications/MAMP/bin/php/php5.6.1/conf' '--with-config-file-path=/Applications/MAMP/bin/php/php5.6.1/conf' '--enable-fip' '--enable-gd-native-ttf' '--with-bz2=/usr' '--with-ldap' '--with-mysql=/Applications/MAMP/Library/bin/mysql_config' '--with-tlib=/Applications/MAMP/Library' '--enable-mbstring=all' '--with-curl=/Applications/MAMP/Library' '--enable-sockets' '--enable-bcmath' '--with-imap=shared,/Applications/MAMP/Library/lib/imap-2007f' '--enable-soap' '--with-kerberos' '--enable-calendar' '--with-pgsql=shared,/Applications/MAMP/Library/pg' '--enable-exif' '--with-libxml-dir=/Applications/MAMP/Library' '--with-gettext=shared,/Applications/MAMP/Library' '--with-xsl=/Applications/MAMP/Library' '--with-pdo-mysql=shared,/Applications/MAMP/Library' '--with-pdo-pgsql=shared,/Applications/MAMP/Library/pg' '--with-mcrypt=shared,/Applications/MAMP/Library' '--with-openssl' '--enable-zip' '--with-iconv=/Applications/MAMP/Library' '--enable-openssl' '--enable-intl' '--with-tidy=shared' '--with-icu-dir=/Applications/MAMP/Library' '--enable-wddx' '--with-libexpat-dir=/Applications/MAMP/Library' '--with-readline' 'CFLAGS=-arch 'LDLAGS=-arch 'LIBS=-lresolv 'CXXFLAGS=-arch |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | disabled |
| Configuration File (php.ini) Path | /Applications/MAMP/bin/php/php5.6.1/conf |
| Loaded Configuration File | /Applications/MAMP/bin/php/php5.6.1/conf/php.ini |
| Scan for additional .ini files | (none) |
| Additional .ini files parsed | |

Figure 2-3. Running the `phpinfo()` command displays full details of your PHP configuration

6. Make a note of the value for the **Loaded Configuration File** item. This tells you where to find `php.ini`, the text file that you need to edit in order to change most settings in PHP.
7. Scroll down to the section labeled **Core** and compare the settings with those recommended in Table 2-1. Make a note of any differences so you can change them as described later in this chapter.

Table 2-1. Recommended PHP configuration settings

| Directive | Local value | Remarks |
|-----------------|--------------|---|
| display_errors | On | Essential for debugging mistakes in your scripts. If set to Off , errors result in a completely blank screen, leaving you clueless as to the possible cause. |
| error_reporting | 32767 | This sets error reporting to the highest level. |
| file_uploads | On | Allows you to use PHP to upload files to a website. |
| log_errors | Off | With display_errors set on, you don't need to fill your hard disk with an error log. |

8. The rest of the configuration page shows you which PHP extensions are enabled. Although the page seems to go on forever, the extensions are all listed in alphabetical order after **Core**. To work with this book, make sure the following extensions are enabled:
 - **gd**: Enables PHP to generate and modify images and fonts.
 - **mysqli**: Connects to MySQL (note the “i,” which stands for “improved” and distinguishes this extension from the older `mysql` one, which should no longer be used).
 - **PDO**: Provides software-neutral support for databases (optional).
 - **pdo_mysql**: Alternative method of connecting to MySQL (optional).
 - **session**: Sessions maintain information associated with a user and are used, among other things, for user authentication.

You should also run `phpinfo()` on your remote server to check which features are enabled. If the listed extensions aren't supported, some of the code in this book won't work when you upload your files to your website. PDO and `pdo_mysql` aren't always enabled on shared hosting, but you can use `mysqli` instead. The advantage of PDO is that it's software-neutral, so you can adapt scripts to work with a database other than MySQL by changing only one or two lines of code. Using `mysqli` ties you to MySQL.

If any of the Core settings in your setup are different from the recommendations in Table 2-1, you will need to edit the PHP configuration file, `php.ini`, as described in the next section.

Editing `php.ini`

The PHP configuration file, `php.ini`, is a very long file, which tends to unnerve newcomers to programming, but there's nothing to worry about. It's written in plain text, and one reason for its length is that it contains copious comments explaining the various options. That said, it's a good idea to make a backup copy before editing `php.ini` in case you make a mistake.

How you open `php.ini` depends on your operating system and how you installed PHP:

- If you used an all-in-one package, such as XAMPP, on Windows, double-click `php.ini` in Windows Explorer. The file opens automatically in Notepad.
- If you installed PHP using the Microsoft Web PI, `php.ini` is normally located in a subfolder of Program Files. Although you can open `php.ini` by double-clicking it, you won't be able to save any changes you make. Instead, right-click **Notepad** and select **Run as Administrator**. (In Windows 7, you need to access Notepad from the **Start** menu. It's in the **Accessories** folder.) Inside Notepad, select **File ► Open** and set the option to display **All Files (*.*)**. Navigate to the folder where `php.ini` is located, select the file, and click **Open**.
- On Mac OS X, `php.ini` is displayed in Finder as an executable file. Use a text editor, such as BBEdit or TextWrangler (both available from www.barebones.com), to open `php.ini`.

Lines that begin with a semicolon (;) are comments. The lines you need to edit do not begin with a semicolon.

Use your text editor's Find functionality to locate the directives you need in order to change your settings to match the recommendations in Table 2-1. Most directives are preceded by one or more examples of how they should be set. Make sure you don't edit one of the commented examples by mistake.

For directives that use On or Off, just change the value to the recommended one. For example, if you need to turn on the display of error messages, edit this line:

```
display_errors = Off
```

by changing it to this:

```
display_errors = On
```

To set the level of error reporting, you need to use PHP constants, which are written in uppercase and are case-sensitive. The directive should look like this:

```
error_reporting = E_ALL
```

After editing `php.ini`, save the file and then restart Apache or IIS so that the changes take effect. If the web server won't start, check the server's error log file. It can be found in the following locations:

- **XAMPP:** In the XAMPP Control Panel, click the **Logs** button alongside **Apache** and then select **Apache (error.log)**.
- **MAMP:** In Applications:MAMP:logs, double-click **apache_error.log** to open it in Console.
- **WampServer:** In the WampServer menu, select **Apache ► Apache error log**.
- **EasyPHP:** Right-click the EasyPHP icon in the system tray and select **Log Files ► Apache**.
- **IIS:** The default location of log files is C:\inetpub\logs.

The most recent entry in the error log should give you an indication of what prevented the server from restarting. Use that information to correct the changes you made to `php.ini`. If that doesn't work, be thankful you made a backup of `php.ini` before editing it. Start again with a fresh copy and check your edits carefully.

What's Next?

Now that you've got a working test bed for PHP, you're no doubt raring to go. The last thing I want to do is dampen any enthusiasm, but before using PHP in a live website, you should have a basic understanding of the rules of the language. So, before jumping into the cool stuff, read the next chapter, which explains how to write PHP scripts. Don't skip it—it's really important.



How to Write PHP Scripts

If you run screaming at the sight of code, this is the chapter you'll enjoy the least, but it's an important one that I've tried to make as user friendly as possible. The chapter is in two parts: the first section offers a quick overview of how PHP works and gives you the basic rules; the second section goes into more detail.

You can read just the first section and come back to the more detailed parts later, or you can read the chapter straight through. However, don't attempt to memorize everything at one sitting. The best way to learn is by doing. Coming back to the second part of the chapter for a little information at a time is likely to be more effective.

If you're already familiar with PHP, you may want to skim through the main headings to see what this chapter contains and brush up your knowledge on any aspects that you're a bit hazy about.

This chapter covers:

- Understanding how PHP is structured
- Embedding PHP in a webpage
- Storing data in variables and arrays
- Getting PHP to make decisions
- Looping through repetitive tasks
- Using functions for preset tasks
- Understanding PHP objects and classes
- Displaying PHP output
- Understanding PHP error messages

PHP: The Big Picture

At first glance, PHP code can look quite intimidating, but once you understand the basics, you'll discover that the structure is remarkably simple. If you have worked with any other computer language, such as JavaScript or jQuery, you'll find they have a lot in common.

Every PHP page *must* have the following:

- The correct filename extension, usually `.php`
- Opening and closing PHP tags surrounding each block of PHP code (although the closing PHP tag can be omitted if the file contains only PHP code)

A typical PHP page will use some or all of the following elements:

- Variables to act as placeholders for unknown or changing values
- Arrays to hold multiple values
- Conditional statements to make decisions
- Loops to perform repetitive tasks
- Functions or objects to perform preset tasks

Let's take a quick look at each of these in turn, starting with the filename and the opening and closing tags.

Telling the Server to Process PHP

PHP is a **server-side language**. This means that the web server processes your PHP code and sends only the results—usually as HTML—to the browser. Because all the action is on the server, you need to tell it that your pages contain PHP code. This involves two simple steps, namely:

- Give every page a PHP filename extension; the default is `.php`. Do not use anything other than `.php` unless you are specifically told to do so by your hosting company.
- Enclose all PHP code within PHP tags.

The opening tag is `<?php` and the closing tag is `?>`. If you put the tags on the same line as surrounding code, there doesn't need to be a space before the opening tag or after the closing one, but there must be a space after the `php` in the opening tag like this:

```
<p>This is HTML with embedded PHP<?php //some PHP code ?>.</p>
```

When inserting more than one line of PHP, it's a good idea to put the opening and closing tags on separate lines for the sake of clarity.

```
<?php
// some PHP code
// more PHP code
?>
```

You may come across `<?` as an alternative short version of the opening tag. However, `<?` doesn't work on all servers. Stick with `<?php`, which is guaranteed to work.

■ **Note** To save space, most examples in this book omit the PHP tags. You must always use them when writing your own scripts or embedding PHP into a webpage.

Embedding PHP in a Webpage

PHP is an **embedded** language. This means that you can insert blocks of PHP code inside ordinary webpages. When somebody visits your site and requests a PHP page, the server sends it to the PHP engine, which reads the page from top to bottom looking for PHP tags. HTML passes through untouched, but whenever the PHP engine encounters a `<?php` tag, it starts processing your code and continues until it reaches the closing `?>` tag. If the PHP code produces any output, it's inserted at that point.

Tip A page can have multiple PHP code blocks, but they cannot be nested inside each other.

Figure 3-1 shows a block of PHP code embedded in an ordinary webpage and what it looks like in a browser and in a page-source view after it has been passed through the PHP engine. The code calculates the current year, checks whether it's different from a fixed year (represented by `$startYear` in line 26 of the code on the left of the figure), and displays the appropriate year range in a copyright statement. As you can see from the page-source view at the bottom right of the figure, there's no trace of PHP in what's sent to the browser.

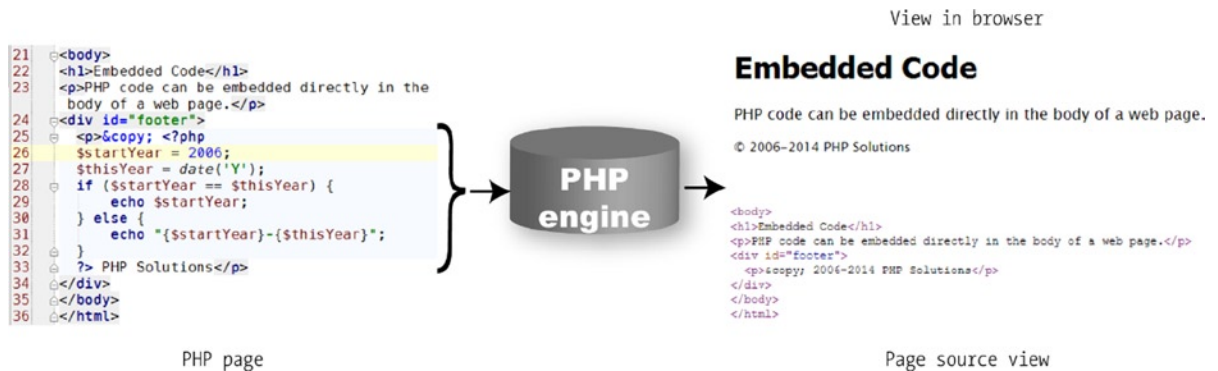


Figure 3-1. The PHP code remains on the server; only the output is sent to the browser

Tip PHP doesn't always produce direct output for the browser. It may, for instance, check the contents of form input before sending an email message or inserting information into a database. Therefore, some code blocks are placed above or below the main HTML code, or in external files. Code that produces direct output, however, always goes where you want the output to be displayed.

Storing PHP in an External File

As well as embedding PHP in HTML, it's common practice to store frequently used code in separate files. When a file contains only PHP code, the opening `<?php` tag is mandatory, but the closing `?>` tag is optional. In fact, the recommended practice is to leave out the closing PHP tag. However, you *must* use the closing `?>` tag if the external file contains HTML after the PHP code.

Using Variables to Represent Changing Values

The code in Figure 3-1 probably looks like an awfully long-winded way to display a range of years. Surely it's much simpler to just type out the actual dates? Yes, it is, but the PHP solution saves you time in the long run. Instead of your needing to update the copyright statement every year, the PHP code does it automatically. You write the code once and forget it. What's more, as you'll see in the next chapter, if you store the code in an external file, any changes to the external file are reflected on every page of your site.

This ability to display the year automatically relies on two key aspects of PHP: **variables** and **functions**. As the name suggests, functions do things; they perform preset tasks, such as getting the current date and converting it into human-readable form. I'll cover functions a little later, so let's work on variables first. The script in Figure 3-1 contains two variables: `$startYear` and `$thisYear`.

■ **Tip** A **variable** is simply a name that you give to something that may change or that you don't know in advance. Variables in PHP always begin with \$ (a dollar sign).

Although the concept of variables sounds abstract, we use variables all the time in everyday life. When you meet somebody for the first time, one of the first things you ask is "What's your name?" It doesn't matter whether the person you've just met is Tom, Dick, or Harry, the word "name" remains constant. Similarly, with your bank account, money goes in and out all of the time (mostly out, it seems), but as Figure 3-2 shows, it doesn't matter whether you're scraping the bottom of the barrel or as rich as Croesus, the amount available is always referred to as the balance.



Figure 3-2. The balance on your bank statement is an everyday example of a variable—the name stays the same, even though the value may change from day to day

So, "name" and "balance" are everyday variables. Just put a dollar sign in front of them and you have two ready-made PHP variables, like this:

```
$name
$balance
```

Simple.

Naming Variables

You can choose just about anything you like as the name for a variable, as long as you keep the following rules in mind:

- Variables always begin with a dollar sign (\$).
- The first character after the dollar sign cannot be a number.
- No spaces or punctuation marks are allowed, except for the underscore (_).
- Variable names are case-sensitive: `$startYear` and `$startyear` are not the same.

When choosing names for variables, it makes sense to choose something that tells you what it's for. The variables you've seen so far—`$startYear`, `$thisYear`, `$name`, and `$balance`—are good examples. Because you can't use spaces in variable names, it's a good idea to capitalize the first letter of the second or subsequent words when combining them (sometimes called **camel case**). Alternatively, you can use an underscore (`$start_year`, `$this_year`, etc.).

Technically speaking, you can use an underscore as the first character after the dollar sign, but it's not recommended. PHP predefined variables (e.g., the superglobal arrays described a little later in this chapter) begin with an underscore, so there's a danger that you may accidentally choose the same name and cause problems for your script.

Don't try to save time by using really short variables. Using `$s`, `$t`, `$n`, and `$b` instead of the more descriptive ones makes code harder to understand—and that makes it hard to write. More important, it makes errors more difficult to spot. As always, there are exceptions to a rule. By convention, `$i`, `$j`, and `$k` are frequently used to keep count of the number of times a loop has run, and `$e` is used in error checking. You'll see examples of these later in this chapter.

■ **Caution** Although you have considerable freedom in the choice of variable names, you can't use `$this`, because it has a special meaning in PHP object-oriented programming. It's also advisable to avoid using any of the keywords listed at <http://php.net/manual/en/reserved.php>.

Assigning Values to Variables

Variables get their values from a variety of sources, including the following:

- User input through online forms
- A database
- An external source, such as a news feed or XML file
- The result of a calculation
- Direct inclusion in the PHP code

Wherever the value comes from, it's always assigned with an equal sign (=), like this:

```
$variable = value;
```

The variable goes on the left of the equal sign, and the value goes on the right. Because it assigns a value, the equal sign is called the **assignment operator**.

■ **Caution** Familiarity with the equal sign from childhood makes it difficult to get out of the habit of thinking that it means “is equal to.” However, PHP uses two equal signs (==) to signify equality. This is a major cause of beginner mistakes, and it often catches more experienced developers, too. The difference between = and == is covered in more detail later in this chapter.

Ending Commands With a Semicolon

PHP is written as a series of commands or statements. Each statement normally tells the PHP engine to perform a particular action, and it must always be followed by a semicolon, like this:

```
<?php
do this;
now do something else;
?>
```

As with all rules, there is an exception: you can omit the semicolon if there's only one statement in the code block. However, *don't do it*. Unlike JavaScript, PHP won't automatically assume there should be a semicolon at the end of a line if you leave it out. This has a nice side effect: you can spread long statements over several lines and lay out your code for ease of reading. PHP, like HTML, ignores whitespace in code. Instead, it relies on semicolons to indicate where one command ends and the next one begins.

■ **Tip** Using a semicolon at the end of a PHP statement (or command) is always right. A missing semicolon will bring your script to a grinding halt.

Commenting Scripts

PHP treats everything between the opening and closing PHP tags as statements to be executed unless you tell it not to do so by marking a section of code as a comment. The following three reasons explain why you may want to do this:

- To insert a reminder of what the script does
- To insert a placeholder for code to be added later
- To disable a section of code temporarily

When a script is fresh in your mind, it may seem unnecessary to insert anything that isn't going to be processed. However, if you need to revise the script several months later, you'll find comments much easier to read than trying to follow the code on its own. Comments are also vital when you're working in a team. They help your colleagues understand what the code is intended to do.

During testing, it's often useful to prevent a line of code, or even a whole section, from running. PHP ignores anything marked as a comment, so this is a useful way of turning on and off code.

There are three ways of adding comments: two for single-line comments and one for comments that stretch over several lines.

Single-line Comments

The most common method of adding a single-line comment is to precede it with two forward slashes, like this:

```
// this is a comment and will be ignored by the PHP engine
```