

# 1

# Getting Started with PHP

When learning any new programming language, you should always begin with an understanding of the basic syntax and functionality, which is what you'll learn in this chapter. The focus here is on the fundamentals of both HTML and PHP, and how the two languages work together. The chapter also covers some recommended programming and debugging techniques, the mastery of which will improve your work in the long run.

If you've never programmed before, a focused reading of this chapter will start you on the right track. If you have some programming experience, you'll be able to breeze through these pages, gaining a perspective for the book's remaining material in the meantime. By the end of this chapter you will have successfully written and executed your first PHP scripts and be on your way to developing dynamic Web applications.

---

## In This Chapter

Basic HTML Syntax	2
Basic PHP Syntax	7
Using FTP	10
Testing Your Script	12
Sending Text to the Browser	15
Using the PHP Manual	18
Sending HTML to the Browser	21
Adding Comments to Scripts	24
Basic Debugging Steps	27
Review and Pursue	30

---

# Basic HTML Syntax

All Web pages are made using HTML (Hypertext Markup Language). Every Web browser, be it Google's Chrome, Mozilla's Firefox, Microsoft's Internet Explorer and Edge, Apple's Safari, turns HTML code—

```
<h1>Hello, World!</h1>  
I just wanted to say <em>Hello</em>.
```

—into the stylized Web page seen by the user <sup>A</sup>.

As of this writing, the current version of HTML is 5, which should remain the norm for some time to come (it was officially standardized in 2014). HTML5 is a solid and practical version of the language, well suited for today's web.

Before getting into the syntax of PHP, let's create one simple but valid HTML document that will act as a template for almost all of this book's examples

## Hello, World!

I just wanted to say *Hello*.

<sup>A</sup> How one Web browser renders the HTML code.

## Basic CSS

The HTML elements define a page's content, but formatting the look and behavior of such content is best left to CSS (Cascading Style Sheets). As with HTML, this book does not teach CSS in any detail, but as some of the book's code will use CSS, you should be familiar with its basic syntax, too.

You can add CSS to a Web page in a couple of ways. The first, and easiest, method is to use HTML **style** tags:

```
<style type="text/css">  
rules  
</style>
```

The CSS rules are defined between the opening and closing tags.

You can also use the **link** HTML tag to incorporate CSS rules defined in an external file:

```
<link href="styles.css" rel="stylesheet" type="text/css" />
```

CSS rules are applied to combinations of general page elements, CSS classes, and specific items:

```
img { border: 0px; }  
.error { color: red; }  
#about { background-color: #ccc; }
```

The first rule applies to every image tag. The second applies to any element that has a class of *error*:

```
<p class="error">Error!</p>
```

The third rule applies to just the specific element that has an ID value of *about*:

```
<p id="about">About...</p>
```

(Not all elements need to have an **id** attribute, but no two elements can have the same **id** value.)

For the most part, this book will just use CSS to do simple things, such as changing the color or background color of an element or some text.

For more on CSS, search the Web or see a dedicated book on the subject.

## To create an HTML page:

1. Open your text editor or Integrated Development Environment (IDE).  
You can use pretty much any application to create HTML and PHP pages. Popular choices include

Adobe's Dreamweaver  
([www.adobe.com](http://www.adobe.com))

Aptana Studio ([www.aptana.com](http://www.aptana.com))

PhpStorm ([www.jetbrains.com](http://www.jetbrains.com))

Sublime Text ([www.sublimetext.com](http://www.sublimetext.com))

Atom (<https://atom.io>)

The first three are IDEs, making them more complicated to use but also more powerful. The last two are text editors. All these programs run on most common operating systems.

2. Choose File > New to create a new, blank document.  
Some text editors allow you to start by creating a new document of a certain type—for example, a new XHTML file **B**. If your application has this option, use it.

3. Start with the XHTML header lines (Script 1.1):

```
<!doctype html>  
<html lang="en">
```

A valid HTML document begins with these lines. They tell the Web browser what type of document to expect. For this template, and in the entire book, *HTML5* pages will be created. One of the niceties of HTML5 is its minimal doctype and syntax.

**Script 1.1** This sample document shows the basics of HTML code.

```
1  <!doctype html>  
2  <html lang="en">  
3  <head>  
4      <meta charset="utf-8">  
5      <title>Welcome to this Page!</title>  
6  </head>  
7  <body>  
8  <h1>This is a basic HTML page!</h1>  
9  <br />  
10 <p>Even with <em>some</em> decoration,  
    it's still not very exciting.</p>  
11 </body>  
12 </html>
```

## Understanding Encoding

Encoding is a huge 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 IDE 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 to the Web browser the encoding being used, there's the corresponding **meta** tag:

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

The **charset=utf-8** part says that UTF-8 (short for *8-bit Unicode Transformation Format*) encoding is being used. Unicode is a way of reliably representing every symbol in every alphabet. Version 6 of Unicode—the current version as of this writing—supports over 99,000 characters! The most commonly used Unicode encoding is UTF-8.

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 used by the text editor or IDE. If you don't, you'll likely see odd characters when you view the page in a Web browser.

4. Create the head section of the page:

```
<head>
  <meta charset=utf-8"/>
  → <title>Welcome to this
    Page!
  → </title>
```

```
</head>
```

The head of an HTML page include the **charset meta** tag. The “Understanding Encoding” sidebar discusses what this means in more detail.

The head also contains the page's title, which will appear at the top of the browser window, as well as in the browser's bookmarks and history. You can also place JavaScript and CSS references in the head.

5. Create the body section:

```
<body>
<h1>This is a basic HTML page!
→ </h1>
<br>
<p>Even with <em>some</em>
→ decoration, it's still not
→ very exciting.</p>
</body>
```

The page's content—what is seen in the Web browser—goes between opening and closing **body** tags.

6. Complete the page with a closing HTML tag:
- ```
</html>
```

*continues on next page*

Character encoding is the way characters -- letters, numbers, symbols -- are represented in numerical values that a computer can understand. Web pages these days use UTF-8 Unicode which supports most written languages including accents commonly used in European languages, as well as non-alphabetic scripts, such as Chinese and Japanese. Encoding dictates what characters can be represented, i.e. what language can be used. Properly encoded Web pages declare the encoding to a browser through a meta tag in the header. Without this tag, a browser may not know to switch to the proper encoding and characters may be displayed as gibberish. UTF-8 (8-bit Unicode Transformation Format) enables every symbol in every alphabet.

6.

7. Choose File > Save As. In the dialog box that appears, choose Text Only (or ASCII) for the format, if you're given the option.


HTML and PHP documents are just plain text files (unlike, for example, a Microsoft Word document, which is stored in a proprietary, binary format). You may also need to indicate the encoding (utf-8) when you save the file (again, see the sidebar).

8. Navigate to the location where you wish to save the script.

You can place this script anywhere you'd like on your computer, although using one dedicated folder for every script in this book, perhaps with subfolders for each chapter, makes sense.

9. Save the file as **welcome.html**.

HTML5 pages use the standard **.html** extension.


10. Test the page by viewing it in your Web browser .

Unlike with PHP scripts (as you'll soon discover), you can test HTML pages by opening them directly in a browser.

**TIP** Use the book's support forum ([www.LarryUllman.com/forum/](http://www.LarryUllman.com/forum/)) or search the Web to find a good HTML and PHP editor or IDE.

**TIP** For more information on HTML, check out Elizabeth Castro's excellent book, *HTML, and CSS, Eighth Edition: Visual QuickStart Guide* (Peachpit Press, 2014).



-  The HTML page, as interpreted by the browser.

# Basic PHP Syntax

Now that you've seen how HTML will be handled in this book, it's time to begin PHP scripting. To create your first PHP page, you'll start exactly as you would if you were creating an HTML document from scratch. Understanding the reason for this is vitally important: Web browsers are client applications that understand HTML; *PHP is a server-side technology*, which cannot be run in the client. To bridge this gap, PHP will be used on the server to generate HTML that's run in a Web browser (refer to the section "How PHP Works" in this book's "Introduction" for a visual representation of this relationship).

There are three main differences between a standard HTML document and a PHP script. First, PHP scripts should be saved with the **.php** file extension (for example, **index.php**). Second, you place PHP code within **<?php** and **?>** tags, normally within the context of some HTML:

```
...
<body><h1>This is HTML.</h1>
<?php PHP code! ?>
<p>More HTML</p>
...
```

The PHP tags indicate the parts of the page to be run through the PHP processor on the server. This leads to the third major difference: *PHP scripts must be run on a PHP-enabled Web server* (whereas HTML pages can be viewed on any computer, directly in a browser). This means that *PHP scripts must always be run through a URL* (i.e., <http://example.com/page.php>). If you're viewing a PHP script in a Web browser and the address does not begin with *http*, the PHP script will not work.

To make this first PHP script do something without too much programming fuss, you'll use the **phpinfo()** function. This function, when called, sends a table of information to the Web browser. That table lists the specifics of the PHP installation on that particular server. It's a great way to test your PHP installation, and it has a high "bang for your buck" quality.

However, the `phpinfo()` function not only outputs a table of information, it also creates a complete HTML page for you. So this first PHP script does not require the standard HTML code, although subsequent scripts in this chapter will.

### To create a new PHP script on your computer:

1. Create a new PHP document in your text editor or IDE, to be named **phpinfo.php** (Script 1.2).  
For this specific case, you'll start with a blank file. But if your text editor or IDE has PHP file templates for you, you can certainly start with one of these.
2. Begin the page **<?php** on its own line.  
This opening PHP tag tells the server that the following code is PHP and should be handled as such

If your application has a PHP template for you, it may have created the PHP tags already.

**Script 1.2** This first PHP script invokes a single PHP function.

```
1  <?php
2  phpinfo();
3  ?>
```



3. Add the following on the next line:

**phpinfo();**

The syntax will be explained in detail later, but in short, this is just a call to an existing PHP function named *phpinfo*. You must use the opening and closing parentheses, with nothing between them, and the semicolon.

4. Type **?>** on its own line, just before the closing body tag.

The closing PHP tag tells the server that the PHP section of the script is over.

Again, because the **phpinfo()** function generates a complete HTML page for you, no HTML tags are needed.

5. Save the script as **phpinfo.php**.

Not to overstate the point, but remember that PHP scripts must use a valid file extension. Most likely you'll have no problems if you save your files as *filename.php*.

You also need to be certain that the application or operating system is not adding a hidden extension to the file. Notepad on Windows, for example, will attempt to add **.txt** to uncommon file extensions, which renders the PHP script unusable. (Generally speaking, do not use Notepad.)

**TIP** Just as a file's extension on your computer tells the operating system in what application to open the file, a Web page's extension tells the server how to process the file: *file.php* goes through the PHP module, *file.aspx* is processed as ASP.NET, and *file.html* is a static HTML document (normally). The extension associations are determined by the Web server's settings.

**TIP** If you're developing PHP scripts for a hosted Web site, check with your hosting company to learn which file extensions you can use for PHP documents. In this book you'll see **.php**, the most common extension.

**TIP** You'll occasionally see PHP's *short tags*—simply **<?** and **?>**—used in other people's scripts, although I recommend sticking with the formal tags **<?>** and **?>**. Support for the short tags must be enabled on a server, and using them makes your code less portable.

**TIP** You'll find it handy to have a copy of the **phpinfo.php** file around. As you'll soon see, this script will report upon PHP's capabilities, settings, and other features of your server. In fact, this book will frequently suggest you return to this script for those purposes.

**TIP** PHP scripts can also be executed without a Web browser, using a command-line interface and a stand-alone PHP executable. But that topic is well outside the scope of this book (and it's a much less common use of PHP regardless).

# Using SFTP

Unlike HTML, which can be tested directly in a Web browser, PHP scripts need to be run from a PHP-enabled server in order for you to see the results. Specifically, PHP is run through a *Web server application*, like Apache (<http://httpd.apache.org>), Nginx ([www.nginx.com](http://www.nginx.com)), or Internet Information Server (IIS, [www.iis.net](http://www.iis.net)).

You can obtain a PHP-enabled server in one of two ways:

- Install the software on your computer.
- Acquire Web hosting.

PHP is open source software (meaning, in part, that it's free) and is generally easy to install (with no adverse effect on your computer as a whole). If you want to install PHP and a Web server on your computer, follow the directions in Appendix A, "Installation and Configuration." Once you've done so, you can skip ahead to the next section of the chapter, where you'll learn how to test your first PHP script.

If you're not running PHP on your own computer, you'll need to transfer your

PHP scripts to the PHP-enabled server using SFTP (Secure File Transfer Protocol). The Web hosting company or server's administrator will provide you with SFTP access information, which you'll enter into an FTP client. Many FTP client applications are available; this next sequence of steps, uses the free FileZilla (<http://filezilla-project.org/>), which runs on many operating systems.

## To SFTP your script to the server:

1. Open your SFTP application.
2. In the application's connection window, enter the information provided by your Web host **A**.

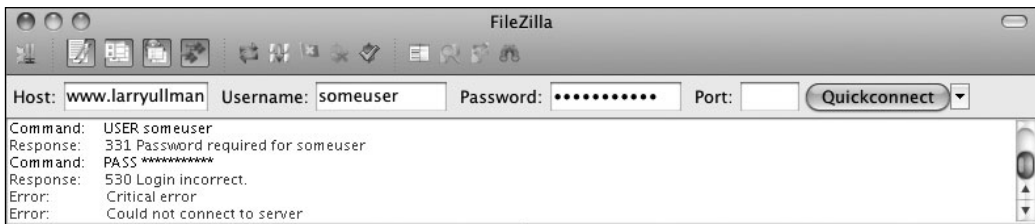
SFTP access requires a host (for example, the domain name or an IP address), username, and password.

3. Click Quickconnect (or your SFTP client's equivalent).

If you've provided the correct information, you should be able to connect. If not, you'll see error messages at the top of the FileZilla window **B**.



**A** The connection section of FileZilla's main window (as it appears on the Mac).



**B** The reported error says that the connection attempt was refused.

4. Navigate to the proper directory for your Web pages (for example, **www, htdocs,** or **httpdocs**).

The SFTP application won't necessarily drop you off in the appropriate directory. You may need to do some navigation to get to the *Web document root*. The Web document root is the directory on the server to which a URL directly points (for example, `www.larryullman.com`, as opposed to `www.larryullman.com/somedir/`). If you're unsure of what the Web document root is for your setup, see the documentation provided by the hosting company (or ask them for support).

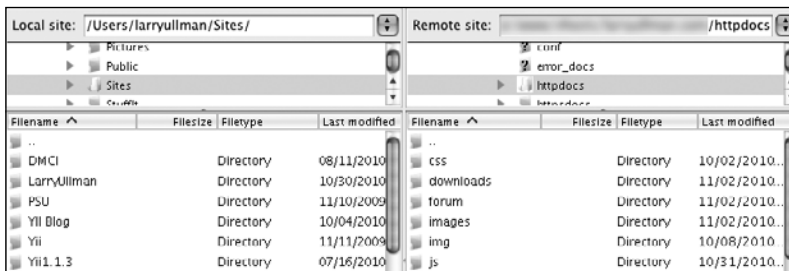
In FileZilla, the right-hand column represents the files and directories on the server; the left-hand column represents the files and directories on your computer **C**. Just double-click on folders to enter them.

5. Upload your script—**phpinfo.php**—to the server.

To do this in FileZilla, you just need to drag the file from the left column—your computer—to the right column—the server.

**TIP** Some text editors and IDEs have built-in SFTP capability, allowing you to save your scripts directly to the server. Other applications can run PHP scripts without leaving the application at all.

**TIP** You can also transfer files to your web server using version control software, such as Git (<https://git-scm.com>). Although this is an excellent route, it's well beyond the scope of a beginner's guide to PHP.



**C** I've successfully connected to the remote server and navigated into the **httpdocs** directory (aka the *Web document root*).

# Testing Your Script

Testing a PHP script is a two-step process. First you must put the PHP script in the appropriate directory for the Web server. Second, you run the PHP script in your Web browser by loading the correct URL.

If you're using a separate Web server, like one provided by a hosting company, you just need to use an SFTP application to upload your PHP script to it (as in the previous steps). If you have installed PHP on your personal computer, then you can test your PHP scripts by saving them in, or moving them to, the Web document root. This is normally

- **~/Sites** for Mac OS X users (where ~ stands for your home directory)
- **C:\Inetpub\wwwroot** for Windows users running IIS
- **C:\xampp\htdocs** for Windows users running XAMPP ([www.apachefriends.com](http://www.apachefriends.com))
- **/Applications/MAMP/htdocs** for Mac users running MAMP ([www.mamp.info](http://www.mamp.info))

If you're not sure what the Web document root for your setup is, see the documentation for the Web server application or operating system (if the Web server application is built-in).


Once you've got the PHP script in the right place, use your browser to execute it.

## To test your script in the browser:

1. Open your favorite Web browser.  
For the most part, PHP doesn't behave differently on different browsers (because PHP runs on the server), so use whichever browser you prefer. In this book, you'll see that I primarily use Chrome, regardless of the operating system.
2. In the browser's address bar, enter the URL of the site where your script has been saved.  
In my case, I enter *[www.larryullman.com](http://www.larryullman.com)*, but your URL will certainly be different. If you're running PHP on your own computer, the URL is `http://localhost` (Windows); or `http://localhost/~username` (Mac OS X), where you should replace *username* with your username. Some all-in-one packages, such as MAMP and XAMPP, may also use a *port* as part of the URL: `http://localhost:8888`.  
If you're not sure what URL to use, see the documentation for the Web server application you installed.

If you placed the script within a subdirectory of the Web document root, you would add that subdirectory name to the URL as well (e.g., `/ch01/phpinfo.php`).

The page should load in your browser window **A**.

<div> <div>phpinfo0</div> <div> <div>PHP Version 5.3.2</div> <div>  </div> </div> </div>	
System	Darwin Lamy-Uilmans-iMac.local 10.4.1 Darwin Kernel Version 10.4.1: Fri Jul 16 23:04:20 PDT 2010; root:xnu-1504.7.51~1/RELEASE_I386 I386
Build Date	Mar 5 2010 16:41:09
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-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php5.3' '--exec-prefix=/Applications/MAMP/bin/php5.3' '--sysconfdir=/Applications/MAMP/conf/php5.3' '--with-soap' '--with-config-file-path=/Applications/MAMP/conf/php5.3' '--enable-track-vars' '--enable-bcmath' '--enable-gd-native-ttf' '--with-bz2=/usr' --with-ldap' --with-mysql=/Applications/MAMP/Library/bin/mysqllib' '--with-sqlite' '--with-ttf' '--with-t1lib=/Applications/MAMP/Library' '--enable-mbstring=all' '--with-curl=/Applications/MAMP/Library' '--enable-dbx' '--enable-sockets' '--enable-bcmath' '--with-imap=shared,/Applications/MAMP/Library/lib/imap-2007e' '--enable-soap' '--with-kerberos' '--enable-calendar' '--with-pgsql=shared,/Applications/MAMP/Library/pg' '--enable-dbase' '--enable-exif' '--with-libxml-dir=/Applications/MAMP/Library' '--with-gettext=shared,/Applications/MAMP/Library' --with-xml=/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'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/Applications/MAMP/conf/php5.3
Loaded Configuration File	/Library/Application Support/Apple/MAMP/PRO/conf/php.ini
Scan this dir for additional ini files	(none)
Additional ini files to parse	(none)
PHP API	20090626
PHP	20090626

## Your PHP version will be 7.X.X.

If you see the PHP code **B** or a blank page, it could mean many things:

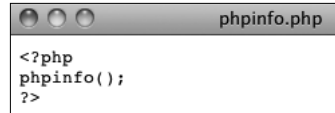
- ▶ You are not loading the PHP script through a URL (i.e., the address does not begin with *http*). Note that you may need to click the address bar to view the full URL, including the *http*, because many of today's browsers hide this by default.
- ▶ PHP has not been enabled on the server.
- ▶ You are not using the proper extension.

If you see a *file not found* or similar error **C**, it could be because

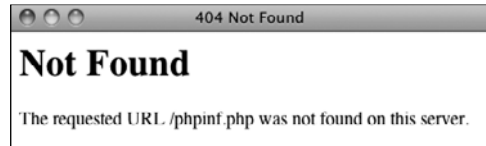
- ▶ You entered the incorrect URL.
- ▶ The PHP script is not in the proper directory.
- ▶ The PHP script does not have the correct name or extension.

**TIP** It's very important to remember that you can't open a PHP file directly in a Web browser as you would open HTML pages or files in other applications. PHP scripts must be processed by the Web server, which means you must access them via a URL (an address that starts with *http://*).

**TIP** Even if you aren't a seasoned computer professional, you should consider installing PHP on your computer. Doing so isn't too difficult, and PHP is free. Again, see Appendix A for instructions.



**B** If you see the raw PHP code, then the PHP code is not being executed.



**C** This server response indicates a mismatch between the URL attempted and the files that actually exist on the server.

# Sending Text to the Browser

PHP wouldn't be very useful if all you could do was see that it works (although that confirmation is critical). You'll use PHP most frequently to send information to the browser in the form of plain text and HTML tags. To do so, use **print**:

```
print "something";
```

Just type the word **print**, followed by what you want to display: a simple message, the value of a variable, the result of a calculation, and so forth. In the previous example, the message is a string of text, so it must be surrounded with quotation marks (in comparison, numbers are not quoted).

PHP is case-insensitive when it comes to calling functions, such as **phpinfo()** and **print**. Using **print**, **Print** and **PRINT** nets the same results. Later in the book, you'll see examples where case makes a crucial difference.

To be clear, **print** doesn't actually *print* anything; it just outputs data. When a PHP script is run through a Web browser, that PHP output is received by the browser itself.

Also notice that the line is terminated with a semicolon (;). Every statement in PHP code must end with a semicolon, and forgetting this requirement is a common cause of errors. A *statement* in PHP is an executable line of code, like

```
print "something";
```

or

```
phpinfo();
```

Conversely, comments, PHP tags, control structures (conditionals, loops, and so on), and certain other constructs discussed in this book don't require semicolons.

Finally, you should know about a minor technicality: whereas `phpinfo()` is a *function*, **print** is actually a *language construct*. Although it's still standard to refer to **print** as a function, because **print** is a language construct, no parentheses are required when using it, as in the `phpinfo()` example.

## To print a simple message:

1. Begin a new HTML document in your text editor or IDE, to be named **hello1.php** (Script 1.3):

```
<!doctype html>
<html lang="en">
<head>

    <meta charset="utf-8">
→ <title>Hello, World!</title>
```

```
</head>
<body>
<p>The following was created
→ by PHP:
```

Most of this code is the standard HTML. The last line will be used to distinguish between the hard-coded HTML and the PHP-generated HTML.

2. On the next line, type `<?php` to create the initial PHP tag.
3. Add

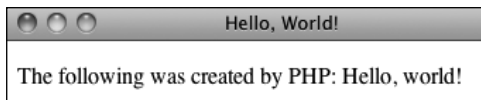
```
print "Hello, world!";
```

Printing the phrase *Hello, world!* is the first step most programming references teach. Even though it's a trivial reason to use PHP, you're not really a programmer until you've made at least one *Hello, world!* application.

**Script 1.3** By putting the **print** statement between the PHP tags, the server will dynamically send the *Hello, world!* greeting to the browser.

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8">
5     <title>Hello, World!</title>
6 </head>
7 <body>
8 <p>The following was created by PHP:
9 <?php
10 print "Hello, world!";
11 ?>
12 </p>
13 </body>
14 </html>
```





**A** A simple *Hello, world!* example: your first foray into PHP programming.

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

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

5. Save the file as **hello1.php**, place it on your PHP-enabled server, and test it in your browser **A**.

If you're running PHP on your own computer, remember that you can save the file to the proper directory and access the script via `http://localhost/`. If you see an error or a blank page instead of the results shown in the figure, see the debugging section at the end of this chapter.

**TIP** You can use other functions to send text to the browser, including `echo` and `printf()`, but this book primarily uses `print`.

**TIP** You can—and commonly will—use `print` over multiple lines:

```
print "This is a longer
sentence of text.";
```

The closing quotation mark terminates the message being printed and the semicolon is placed only at the end of that line.

# Using the PHP Manual

The PHP manual—accessible online at [www.php.net/manual](http://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).

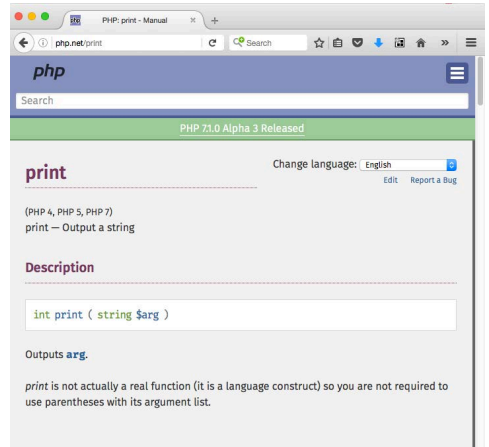
To quickly look up any function in the PHP manual, go to [www.php.net/functionname](http://www.php.net/functionname) in your Web browser (for example, [www.php.net/print](http://www.php.net/print)).

To understand how functions are described, look at the start of the **print** function's page **A**.

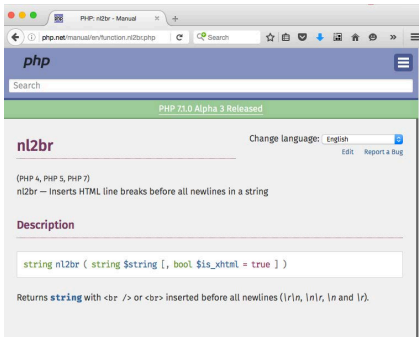
The first line is the name of the function itself, followed by the versions of PHP in which it's available. As the language grows, new functions are added and, occasionally, older functions are removed. Then there's a textual description of the function along with the function's basic usage. The usage is the most important and confusing part.

In this example, the first value, **int**, says that **print** returns an integer value (specifically, **print** returns 1, always).

Within the parentheses, **string \$arg** states that the function takes one required argument, which should be in the form of a string. You've already seen this in action.



**A** The PHP manual's page for the **print** construct.



**B** The PHP manual's page for the `nl2br()` function.

As a comparison, check out the manual's listing for the `nl2br()` function **B**. This function converts newlines found within text (the equivalent of pressing Return/Enter) into HTML break tags. This function, which returns a string, takes a string as its first argument and an optional Boolean (**true/false**) as its second. Whenever you see the square brackets, that indicates optional arguments, which are always listed last. When a function takes multiple arguments, they are separated by commas. Hence, this function can be called like so:

```
nl2br("Some text");  
nl2br("Some text", false);
```

As the definition also indicates, the second argument has a default value of **true**, meaning it'll create XHTML `<br />` tags unless the function is passed a second argument value of **false**. In that case, the function will create HTML `<br>` tags instead.

If you're ever confused by a function or how it is properly used, check the PHP manual's reference page for it.

## To look up a function definition:

1. Head to `www.php.net/functionname` in your Web browser.

If the PHP manual doesn't have a matching record for the function you tried, check the spelling or look at the recommended alternatives that the manual presents **C**.

2. Compare the versions of PHP that the function exists in with the version of PHP you're using.

Use the `phpinfo()` function, already demonstrated, to know for certain what version of PHP you are running. If a function was added in a later version of PHP, you'll either need to upgrade the version you have or use a different approach.

3. Examine what type of data the function returns.

Sometimes you may be having a problem with a function because it returns a different type of value than you expect it to.

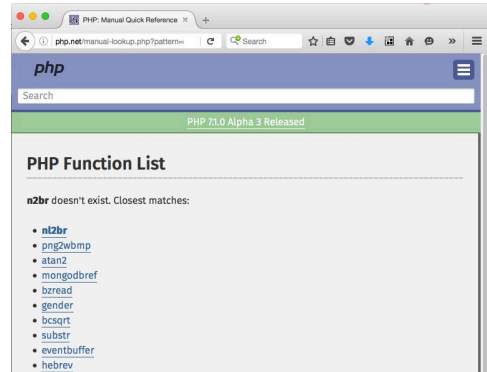
4. Examine how many and what types of arguments the function requires or can take.

The most common mistake when using functions is sending the wrong number or type of arguments when the function is called.

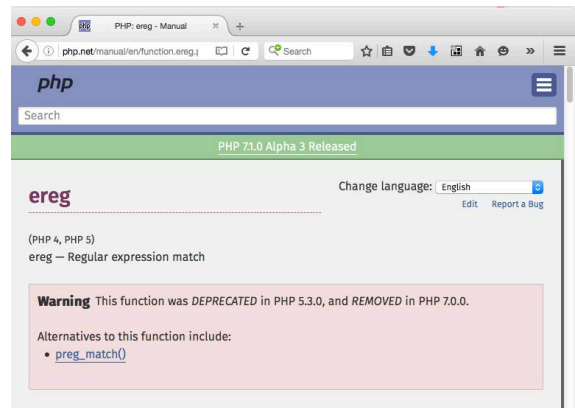
5. Read the user comments, when present, to learn more.

Sometimes the user comments can be quite helpful (other times not).

**TIP** If you see a message saying that a function has been deprecated **D**, that means the function will be dropped from future versions of PHP and you should start using the newer, better alternative (there almost always is one, and it will be identified).



**C** The manual will present alternative functions if the entered URL doesn't directly match a reference.



**D** Deprecated functions should be avoided in your code.

**Parse error:** syntax error, unexpected T\_STRING in /Users/larryullman/Sites/phpvqs4/hello.php on line 11

**A** Attempting to print double quotation marks will create errors, as they conflict with the **print** statement's primary double quotation marks.

**Script 1.4** With the **print** function, you can send HTML tags along with text to the browser, where the formatting will be applied.

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <title>Hello, World!</title>
6 <style type="text/css">
7 .bold {
8     font-weight: bolder;
9 }
10 </style>
11 </head>
12 <body>
13 <p>The following was created by PHP:
14 <?php
15     print "<span class='\"bold;\"'>Hello,
16         world!</span>";
17 ?>
18 </body>
19 </html>
```

## Sending HTML to the Browser

As those who first learned HTML quickly discovered, viewing plain text in a Web browser leaves a lot to be desired. Indeed, HTML was created to make plain text more appealing and useful. Because HTML works by adding tags to text, you can use PHP to also send HTML tags to the browser, along with other data:

```
print "<b>Hello, world!</b>";
```

There is one situation where you have to be careful, though. HTML tags that require double quotation marks, like `<a href="page.php">link</a>`, will cause problems when printed by PHP, because the **print** function uses quotation marks as well **A**:

```
print "<a href='\"page.php\"'>link</a>";
```

One workaround is to *escape* the quotation marks within the HTML by preceding them with a backslash (\):

```
print "<a href='\"page.php\"'>link</a>";
```

By escaping each quotation mark within the **print** statement, you tell PHP to print the mark itself instead of treating the quotation mark as either the beginning or end of the string to be printed.

### To send HTML to the browser:

1. Open the hello1.php script (Script 1.3) in your text editor or IDE, if it is not already open.
2. Within the HTML head, declare a CSS class (Script 1.4):

```
<style type="text/css">
.bold {
    font-weight: bolder;
}
</style>
```

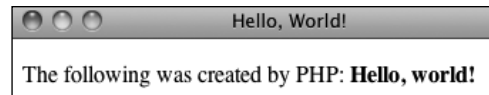
This CSS code declares a class named **bold**, which will be used to add emphasis to text. This is obviously a fairly trivial use of CSS, but by declaring this as a class, it can easily be updated, perhaps to change the color of the text or the size, along with its weight.

3. Edit Hello, world! message by adding the HTML tags, making it read as follows:

```
print "<span class='bold'>  
    Hello, world!</span>";
```

To make the PHP-generated part of the message stand out, CSS styling will embolden the greeting. For this to work, you must escape the quotation marks within the **span** tag so that they don't conflict with the **print** statement's quotation mark.

4. Save the script as **hello2.php**, place it on your PHP-enabled server, and run the page in your browser **B**.



**B** The new version of the *Hello, world!* page, with a little more decoration and appeal.

## Using White Space

When you're programming in PHP, you should understand that white space is generally (but not universally) ignored. Any blank line (just one or several in a row) in PHP code is irrelevant to the end result. Likewise, tabs and spaces are normally inconsequential to PHP. And as PHP code is not visible in the Web browser (unless there's a problem with the server), white space in your PHP files has no impact on what the end user sees.

The spacing of HTML code shows up in the HTML source of a Web page but has only a minimal effect on what's viewed in the Web browser. For example, all of a Web page's HTML source code could be placed on one line without changing what the end user sees. If you had to hunt for a problem in the HTML source, however, you would not like the long, single line of HTML. You can affect the spacing of dynamically generated HTML code by printing it in PHP over multiple lines, or by using the newline character (**\n**) within double quotation marks:

```
print "Line 1\nLine 2";
```

Again, use of the newline character affects the *HTML source code* of the Web page, not what the end user will see rendered in the browser.

To adjust the spacing in the rendered Web page, you'll use CSS, plus paragraph, div, and break tags, among others.

5. View the HTML page source to see the code that was sent to the browser **C**.

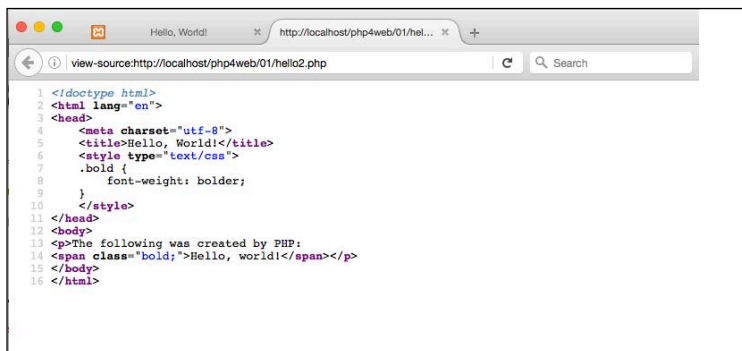
How you do this depends upon the browser: Select View > Developer > View Source in Chrome, View > Page Source in Firefox, View > Source in Internet Explorer.

This is a step you'll want to be in the habit of taking, particularly when problems occur. Remember that PHP is primarily used to generate HTML, sent to and interpreted by the Web browser. Often, confirming what was sent to the Web browser (by viewing the source) will help explain the problem you're seeing in the browser's interpretation (or visible result).

**TIP** Understanding the role of quotation marks and how to escape problematic characters is crucial to programming with PHP. These topics will be covered in more detail in the next two chapters.

**TIP** The HTML you send to the Web browser from PHP doesn't need to be this simple. You can create tables, JavaScript, and much, much more.

**TIP** Remember that any HTML outside of the PHP tags will automatically go to the browser. Within the PHP tags, `print` statements are used to send HTML to the Web browser.



```
1 <!doctype html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8">
5     <title>Hello, World!</title>
6     <style type="text/css">
7         .bold {
8             font-weight: bolder;
9         }
10    </style>
11 </head>
12 <body>
13 <p>The following was created by PHP:
14 <span class="bold">Hello, world!</span></p>
15 </body>
16 </html>
```

- C** The resulting HTML source code of `hello2.php` **B**.

# Adding Comments to Scripts

Comments are integral to programming, not because they do anything but because they help you remember why *you* did something. The computer ignores these comments when it processes the script. Furthermore, PHP comments are never sent to the Web browser and therefore remain your secret.

PHP supports three ways of adding comments. You can comment out one line of code by putting either `//` or `#` at the beginning of the line you want ignored:

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

You can also use `//` or `#` to begin a comment at the end of a PHP line, like so:

```
print "Hello"; // Just a greeting.
```

Although it's largely a stylistic issue, `//` is much more commonly used in PHP than `#`.

You can comment out multiple lines by using `/*` to begin the comment and `*/` to conclude it:

```
/* This is a  
multi-line comment. */
```

Some programmers also prefer this comment style as it contains both open and closing “tags,” providing demarcation for where the comment begins and ends.



**Script 1.5** PHP and HTML comments are added to the script to document it and to render a line of PHP code inert.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>Hello, World!</title>
6    <style type="text/css">
7      .bold {
8        font-weight: bolder;
9      }
10   </style>
11 </head>
12 <body>
13 <p>The following was created by PHP: <br>
14 <?php
15  /*
16   * Filename: hello3.php
17   * Book reference: Script 1.5
18   * Created by: Larry Ullman
19   */
20
21  //print "<span class=\"bold\">
22  Hello, world!</span>";
23  ?>
24  <!-- This is an HTML comment. -->
25  </p>
26  </body>
27  </html>
```

## To add comments to a script:

1. Open the hello2.php (Script 1.4) in your text editor or IDE.
2. After the initial PHP tag, add some comments to your script (Script 1.5):

```
/*
 *Filename: hello3.php
 *Book reference: Script 1.5
 *Created by: Larry Ullman
 */
```

This is just a sample of the kind of comments you can write. You should document what the script does, what information it relies on, who created it, when, and so forth. Stylistically, such comments are often placed at the top of a script (as the first thing within the PHP section, that is), using formatting like this. The extra asterisks aren't required; they just draw attention to the comments.

3. On line 21, in front of the **print** statement, type **//**.  
By preceding the **print** statement with two slashes, the function call is "commented out," meaning it will never be executed.
4. After the closing PHP tag (on line 23), add an HTML comment:  
**<!-- This is an HTML comment. -->**  
This line of code will help you distinguish among the different comment types and where they appear. This comment will only appear within the HTML source code.
5. Save the script as **hello3.php**, place it on your PHP-enabled server, and run the page in your Web browser **A**.

6. View the source of the page to see the HTML comment **B**.

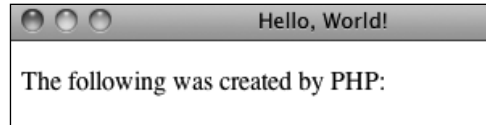
**TIP** You can comment out just one line of code or several using the `/*` and `*/` method. With `//` or `#`, you can negate only one line at a time.

**TIP** Different programmers prefer to comment code in different ways. The important thing is to find a system that works for you and stick to it.

**TIP** Note that you cannot use HTML comment characters (`<!--` and `-->`) within PHP to comment out code. You could have PHP print those tags to the browser, but in that case you'd create a comment that appeared in the HTML source code on the client's computer (but not in the browser window). PHP comments never make it as far as a user's computer.

**TIP** Despite my strong belief that you can't over-comment your scripts, the scripts in this book aren't as documented as they should be, in order to save space. But the book will document each script's name and number, for cross-reference purposes.

**TIP** When you change a script's code, be certain to update its comments as well. It's quite confusing to see a comment that suggests a script or a line of code does something other than what it actually does.



**A** With the `print` statement commented out, the page looks just as it would if the `print` function weren't there.



**B** HTML comments don't appear in the Web browser but are in the HTML source. PHP comments remain in the PHP script on the server, not visible inside the HTML source.

# 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.

## 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 through **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 the **phpinfo.php** file (Script 1.2) to confirm the version of PHP in use.

- **Make sure `display_errors` is on.** This is a basic PHP configuration setting (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 and Chapter 3, "HTML Forms and PHP."

- **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 my 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.

**TIP** These are just some general debugging techniques, specifically tailored to the beginning PHP programmer. They should suffice for now, though, as the examples in this book are relatively simple. More complex coding requires more advanced debugging techniques, so my *PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide, Fourth Edition* (Peachpit Press, 2012) dedicates a whole chapter to this subject.

# Review and Pursue

Each chapter ends with a “Review and Pursue” section. In these sections you’ll find

- Questions regarding the material just covered
- 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 ([www.LarryUllman.com/forum/](http://www.LarryUllman.com/forum/)).

## Review

- What is HTML? What is the current version of HTML?
- What encoding is your text editor or IDE set to use? Does that match the encoding specified in your generated HTML pages? Why does the encoding matter?
- What is CSS and what is it used for?
- What file extension should PHP scripts have for your particular server?
- What is meant by “web root directory”? What is the web root directory for your server?
- How do you test PHP scripts? What happens when PHP scripts are not run through a URL?
- Name two ways comments can be added to PHP code. Identify some ways you would use comments.

## Pursue

- If you have access to more than one server, confirm what version of PHP is running on another server.
- Create a static HTML page that displays some information. Then replace some of the static content with content created by PHP.
- Create a template to use for your own work. The template should contain the HTML shell, the opening and closing PHP tags, and some basic comments.
- Confirm, using the `phpinfo()` function, that **display\_errors** is enabled on your server. If it's not, change your server's configuration to enable it (see Chapter 3 and Appendix A).
- In subsequent chapters, occasionally check the PHP manual's page when a new function is mentioned in the book.