Part 1

PROGRAMMING WITH PHP

The first part of this book is a thorough discussion of PHP as a programming language. You will be introduced to common concepts of computer science and how they are implemented in PHP. No prior programming experience beyond the use of simple mark-up languages is necessary. That is, you must be familiar with HTML. These chapters focus on building a foundation of understanding rather than on how to solve specific problems. If you have experience programming in a similar language, such as C or Perl, you may choose to read Chapter 1 and skim the rest, saving it as a reference. In most situations, PHP treats syntax much as these two languages do.

Chapter 1 is an introduction to PHP—how it began and what it looks like. It may be sufficient for experienced programmers, since it moves quickly through PHP's key features. If you are less experienced, I encourage you to treat this chapter as a first look. Don't worry too much about exactly how the examples work. I explain the concepts in depth in later chapters.

Chapter 2 introduces the concepts of variables, operators, and expressions. These are the building blocks of a PHP script. Essentially, a computer stores and manipulates data. Variables let you name values; operators and expressions let you manipulate them.

Chapter 3 examines the ways PHP allows you to control program execution. This includes conditional branches and loops.

Chapter 4 deals with functions, how they are called, and how to define them. Functions are packages of code that you can call upon repeatedly.

Chapter 5 is about arrays—collections of values that are identified by either numbers or names. Arrays are a very powerful way to store information and retrieve it efficiently.

Chapter 6 is about classes, presenting an object-oriented approach to grouping functions and data. Although not strictly an object-oriented language, PHP supports many features found in OO languages like Java.

Chapter 7 deals with how PHP sends and receives data. Files, network connections, and other means of communication are covered.

AN INTRODUCTION TO PHP

Topics in This Chapter

- The Origins of PHP
- What Makes PHP Better than Its Alternatives
- Interfaces to External Systems
- How PHP Works with the Web Server
- Hardware and Software Requirements
- Installation on Apache for UNIX
- Installation on 11S for Windows NT
- Editing Scripts
- Algorithms
- What a PHP Script Looks Like
- Saving Data for Later
- Receiving User Input
- Choosing between Alternatives
- Repeating Code
- Conclusion

Chapter 1

This chapter will introduce you to PHP. You will learn how it came about, what it looks like, and why it is the best server-side technology. You will also be exposed to the most important features of the language.

PHP began as a simple macro replacement tool. Like a nice pair of shoes, it got you where you needed to go, but you could go only so far. On the hyperspeed development track of the Internet, PHP has become the equivalent of a 1960s muscle car. It's cheap, it's fast, and there's plenty of room under the hood for you and your virtual wrench.

You probably don't need convincing that whether it's Internet, intranet, or extranet, the Web is no longer about plain HTML files. Web pages are being replaced with Web applications. The issue many Web engineers face is choosing among hundreds of technologies.

This chapter will let you poke around the PHP engine, get your hands a little dirty, and take it for a spin. There are lots of small examples you can try immediately. Like all the examples in this book, you can easily adapt them to provide real solutions. Don't be intimidated if you don't fully understand the PHP code at first. Later chapters will deal with all the issues in detail.

This chapter talks about some things that you already know, like what a computer is, just to make sure we're all on the same page. You may be a wizard with HTML, but not fully appreciate the alien way computers are put

together. Or you may find you learned all these things in a high school computer class. If you get too bored with the basics, skip to Chapter 2, "Variables, Operators, and Expressions."

The Origins of PHP

Wonderful things come from singular inspiration. PHP began life as a simple way to track visitors to Rasmus Lerdorf's online resume. It also could embed SQL queries in Web pages. But as often happens on the Web, admirers quickly asked for their own copies. As a proponent of the Internet's ethic of sharing, as well as a generally agreeable person, Rasmus unleashed upon an unsuspecting Web his Personal Home Page Tools version 1.0.

"Unleashed upon himself" may be more accurate. PHP became very popular. A consequence was a flood of suggestions. PHP 1.0 filtered input, replacing simple commands for HTML. As its popularity grew, people wondered if it couldn't do more. Loops, conditionals, rich data structures—all the conveniences of modern structured programming seemed like a next logical step. Rasmus studied language parsers, read about YACC and GNU Bison, and created PHP 2.0.

PHP 2.0 allowed developers to embed structured code inside HTML tags. PHP scripts could parse data submitted by HTML forms, communicate with databases, and make complex calculations on the fly. And it was very fast, because the freely available source code compiled into the Apache Web server. A PHP script executed as part of the Web server process and required no forking, often a criticism of Common Gateway Interface (CGI) scripts.

PHP was a legitimate development solution and began to be used for commercial Web sites. In 1996 Clear Ink created the SuperCuts site (www.supercuts.com) and used PHP to created a custom experience for the Web surfer. In January of 1999 the PHP Web site reported almost 100,000 Web servers were using PHP. By November that figure had climbed higher than 350,000!

A community of developers grew up around PHP. Feature requests were balanced by bug fixes and enhancements. Zeev Suraski and Andi Gutmans made a significant contribution by writing a new parser. They observed that the parser in PHP 2.0 was the source of many problems. Rasmus decided to begin work on PHP 3.0 and called for developers to commit to its creation. Along with Zeev and Andi, three others lent their support: Stig Bakken, Shane Caraveo, and Jim Winstead.

After seven months of developments, PHP version 3.0 was released on June 6, 1998. Work began immediately on the next version. Originally a 3.1 version was planned, but thanks to more revolutionary work by Zeev and Andi, work shifted to PHP 4.0, which used the new Zend library.

On January 4, 1999, Zeev and Andi announced a new framework that promised to increase dramatically the performance of PHP scripts. They named the new framework Zend, cleverly combining letters from their names. Early tests showed script execution times dropping by a factor of one hundred. In addition, new features for compiling scripts into binary, optimization, and profiling were planned.

Work on Zend and PHP 4.0 continued in parallel with bug fixes and enhancement to PHP 3.0. During 1999, eight incremental versions were released, and on December 29, 1999, PHP version 3.0.13 was announced. During the same year, Open Source projects written in PHP flourished. Projects like Phorum tackled long-time Internet tasks such as hosting online discussion. The PHPLib project provided a framework for handling user sessions that inspired new code in PHP. FreeTrade, a project I lead, offered a toolkit for building e-commerce sites.

Writing about PHP increased as well. More than twenty articles appeared on high-traffic sites such as webmonkey.com and techweb.com. Sites dedicated to supporting PHP developers were launched. The first two books about PHP were published in May 1999. Egon Schmid, Christian Cartus, and Richard Blume wrote a book in German called *PHP: Dynamische Webauftritte professionell realisieren.* Prentice Hall published the first edition of my book, *Core PHP Programming.* Since then several other books have been published and others planned.

PHP is not a shrink-wrapped product made by faceless drones or wizards in an ivory tower. PHP started as a simple tool brought into the bazaar described by Eric Raymond in his essay *The Cathedral and the Bazaar* <`http://www.tuxedo.org/~esr/writings/cathedral-bazaar/`>.

Once it was there, anyone could make improvements, and many did. Their aim seems to be to achieve solutions of direct, personal interest. If a client comes along that requires a project use a database not supported by PHP, you simply write an extension. Then you give it to the PHP project. Soon other people are fixing your bugs.

Yet, the vast majority of PHP users never write an extension. They happily find everything they need in the contributed works of others. Those who've contributed thousands of lines of code to PHP perhaps never consider themselves heroes. They don't trumpet their accomplishments. But because each

part of PHP came from a real person, I would like to point them out. When appropriate, I'll note who added a particular extension.

You can find an up-to-date list of credits on the PHP site <http://www.php.net/version4/credits.php>.

What Makes PHP Better than Its Alternatives

The skeptics are asking themselves, "Why should I learn PHP?" The days of static Web sites built with HTML files and a few CGI scripts are over: Today's sites must be dynamic. All the stale company brochures littering the streets of the Internet will transform into 24-hour virtual storefronts or be swept away. The toughest decision facing the creator of a Web application is choosing from hundreds of technologies.

Perl has adapted well to being a CGI solution and it has been used to drive complex Web technology like CyberCash and Excite's EWS search engine. Microsoft provides its Active Server Pages with Internet Information Server. Middleware like Allaire's Cold Fusion is yet another solution. ServerWatch.com lists hundreds of Web technologies, some costing tens of thousands of dollars. Why should you choose PHP over any of these alternatives?

The short answer is that PHP is better. It is faster to code and faster to execute. The same PHP code runs unaltered on different Web servers and different operation systems. Additionally, functionality that is standard with PHP is an add-on in other environments. A more detailed argument follows.

PHP is free. Anyone may visit the PHP Web site <http://www.php.net/> and download the complete source code. Binaries are also available for Windows. The result is easy entry into the experience. There is very little risk in trying PHP, and its license allows the code to be used to develop works with no royalties. This is unlike products such as Allaire's Cold Fusion or Everyware's Tango Enterprise that charge thousands of dollars for the software to interpret and serve scripts. Even commercial giants like Netscape and IBM now recognize the advantages of making source code available.

PHP runs on UNIX, Windows 98, Windows NT, and the Macintosh. PHP is designed to integrate with the Apache Web Server. Apache, another free technology, is the most popular Web server on the Internet and comes with source code for UNIX and Windows. Commercial flavors of Apache like WebTen and Stronghold support PHP, too. But PHP works with other Web servers,

including Microsoft's Internet Information Server. Scripts may be moved between server platforms without alteration. PHP supports ISAPI to allow for the performance benefits of tightly coupling with Microsoft Web servers.

PHP is modifiable. PHP has been designed to allow for future extension of functionality. PHP is coded in C and provides a well-defined Application Programming Interface (API). Capable programmers may add new functionality easily. The rich set of functions available in PHP are evidence that they often do. Even if you aren't interested in changing the source code, it's comforting to know you can inspect it. Doing so may give you greater confidence in PHP's robustness.

PHP was written for Web page creation. Perl, C, and Java are very good general languages and are certainly capable of driving Web applications. The unfortunate sacrifice these alternatives make is the ease of communication with the Web experience. PHP applications may be rapidly and easily developed because the code is encapsulated in the Web pages themselves.

Support for PHP is free and readily available. Queries to the PHP mailing list are often answered within hours. A custom bug-tracking system on the PHP site shows each problem along with its resolution. Numerous sites, such as phpbuilder.com and zend.com, offer original content to PHP developers.

PHP is popular. Internet service providers find PHP to be an attractive way to allow their customers to code Web applications without the risks exposed by CGIs. Developers worldwide offer PHP programming. Sites coded in PHP will have the option of moving from one host to another as well as a choice of developers to add functionality.

Programming skills developed in other structured languages can be applied to PHP. PHP takes inspiration from both Perl and C. Experienced Perl and C programmers learn PHP very quickly. Likewise, programmers who learn PHP as a first language may apply their knowledge toward not only Perl and C, but other C-like languages such as Java. This is very different from learning to code in a visual editor such as Microsoft Visual InterDev.

Interfaces to External Systems

PHP is somewhat famous for interfacing with many different database systems, but it also has support for other external systems. Support comes in the form of modules called extensions. They either compile directly into PHP or are loaded dynamically. New extensions are added to the PHP project regularly. The extensions expose groups of functions for using these external

systems. As I've said, some of these are databases. PHP offers functions for talking natively with most popular database systems, as well as providing access to ODBC drivers. Other extensions give you the ability to send messages using a particular network protocol, such as LDAP or IMAP. These functions are described in detail in Section Two, but you might find the highlights listed here interesting. Because PHP developers are enthusiastic and industrious, you will undoubtedly find more extensions have been added since I wrote this.

Aspell is a system for checking spelling. An extension provides support for numbers of arbitrary precision. There is an extension for dealing with various calendar systems. An extension provides support for DBM-style databases. You can read from filePro databases. You can interact with Hyperwave. You can use the ICAP, IMAP, and LDAP protocols. The Interbase and Informix databases are supported natively, as are mSQL, Mysql, MS SQL, Sybase, Oracle, and Postgres. You can also parse XML or create WDDX packets.

How PHP Works with the Web Server

The normal process a Web server goes through to deliver a page to a browser is as follows. It all begins when a browser makes a request for a Web page. Based on the URL, the browser resolves the address of the Web server, identifies the page it would like, and gives any other information the Web server may need. Some of this information is about the browser itself, like its name (Mozilla), its version (4.08), or the operating system (Linux). Other information given the Web server could include text the user typed into form fields.

If the request is for an HTML file, the Web server will simply find the file, tell the browser to expect some HTML text, and then send the contents of the file. The browser gets the contents and begins rendering the page based on the HTML code. If you have been programming HTML for any length of time, this will be clear to you.

Hopefully you have also had some experience with CGI scripts. When a Web server gets a request for a CGI, it can't just send the contents of the file. It must execute the script first. The script will generate some HTML code, which then gets sent to the browser. As far as the browser is concerned, it's just getting HTML. The Web server does a bunch of work that it

gets very little recognition for, but Web servers rarely get the respect they deserve. The medium is definitely not the message.

When a PHP page is requested, it is processed exactly like a CGI, at least to the extent that the script is not simply sent to the browser. It is first passed through the PHP engine, which gives the Web server HTML text.

What happens when the user clicks the stop button before the page finishes downloading? The Web server detects this situation and usually terminates the PHP script. It is possible to force a script to finish despite an aborted connection. You may also allow the script to terminate but execute special code first. The functions to allow this functionality are listed in Chapter 8, "I/O Functions," and Chapter 11, "Time Date, and Configuration Functions."

Hardware and Software Requirements

One great advantage of Open Source software is that it provides the opportunity for adaptation to new environments. This is true of PHP. Although originally intended as a module for the Apache Web server, PHP has since embraced the ISAPI standard, which allows it to work equally well with Microsoft's Internet Information Server. With regard to hardware requirements, I have personally witnessed PHP running on 100-MHz Pentium machines running Slackware Linux and Windows NT, respectively. Performance was fine for use as a personal development environment. A site expected to receive thousands of requests a day would need faster hardware, of course. Although more resources are needed when comparing a PHP-powered site to a flat HTML site, the requirements are not dramatically different. Despite my example, you are not limited to Intel hardware. PHP works equally well on PowerPC and Sparc CPUs.

When choosing an operating system, you have the general choice between Windows and a UNIX-like OS. PHP will run on Windows 95 and 98, although these operating systems aren't suited for high-traffic Web servers. It will also run on Windows NT and its successor, Windows 2000. For UNIX operating systems, PHP works well with Linux and Solaris, as well as others. If you have chosen a PPC-based system, such as a Macintosh, you may choose LinuxPPC, a version of Linux. You may pursue the commercial WebTen Web server that runs in the Macintosh OS. Chad Cunningham has

contributed patches for compiling PHP in Apple's OS X. In 1999 Brian Havard added support for IBM's OS/2.

PHP still works best with the Apache Web server. But it now works very well with IIS. It also compiles as a module for the fhttpd Web server. You can make PHP work with almost any Web server using the CGI version, but I don't recommend this setup for production Web sites. If you are using UNIX, I recommend compiling PHP as an Apache module. If you are using Windows NT, pursue IIS.

Installation on Apache for UNIX

If you are using Linux, you can easily find an RPM for Apache and PHP, but this installation may not include every PHP feature you want. I recommend this route as a very quick start. You can always pursue compiling Apache and PHP from scratch later. PHP will compile on most versions of UNIX-like operating systems, including Solaris and Linux. If you have ever compiled software you've found on the Net, you will have little trouble with this installation. If you don't have experience extracting files from a tar archive and executing make files, you may wish to rely on your sysadmin or someone else more experienced. You will need to have root privileges to completely install PHP.

The first step is to download the tar files and unpack them. The CDROM that accompanies this book has recent versions of both PHP and Apache, but you may wish to check online for the newest versions, <`http://www.php.net/`> and <`http://www.apache.org/`>, respectively.

After unpacking the tar file, the first step is to configure Apache. This is done by running the configure script inside the Apache directory:

```
./configure –prefix=/www
```

The script will examine your system and prepare a make file for Apache. The `prefix` directive will cause a directory to be created in the root of your file system.

Next, configure and compile PHP:

```
./configure –with-apache=/usr/local/src/apache_1.3.9 –enable-track-vars
make
make install
```

This is done within the PHP directory. The –with-apache and –enable-track-vars options are minimal. You might add –with-mysql if you have the MySQL database installed. PHP can usually find the MySQL libraries on its own. Appendix E, "Compile-Time Configuration" lists the compile-time configuration directives. Running make will create the PHP library, and make install will prepare Apache for including the PHP module. Notice that the call to configure includes a path to your Apache source code directory. This can be a relative path, as you may have put the Apache source code parallel to the PHP source code. However, do not make the mistake of using relative paths for any of the other directives.

Next, you will need to reconfigure Apache and run make. Return to the Apache source code directory and run configure again, this time with an option that tells Apache to include the PHP module:

```
./configure –prefix=/www –activate-module=src/modules/php4/libphp4.a
make
make install
```

This will create a new make file and then run it. The new httpd binary will be installed in the /www/bin directory, or wherever you specified the files should be installed.

To supply additional configuration options PHP uses a file called php.ini. This file should reside in /usr/local/lib, so copy it from the PHP source directory:

```
cp php.ini-dist /usr/local/lib/php.ini
```

It is not likely you will need to edit this file, but if you do, there are instructive comments inside.

The last step is to associate a file extension with PHP. This is done by editing the httpd.conf file. It can be found in Apache's conf directory, /www/conf/httpd.conf, for example. Add the following line:

```
AddType application/x-httpd-php .php
```

This causes all files with the extension .php to be executed as PHP scripts. You may choose another, such as phtml. You may also wish to insert index.php as a default document. When the Apache server is started, it will process PHP scripts. The documentation for Apache has hints for starting Apache automatically. If you have been running Apache previously, you will need to restart it, not just use a kill –HUP command.

Installation on IIS for Windows NT

The first step is to install PHP. You do not need to compile PHP for Windows. A binary distribution is available on the Web site. Download the zip file and expand it wherever you wish. I put mine in `c:\php4`. Next, copy the file `php.ini-dist` into your system root directory, which is probably `c:\winnt`. Rename it `php.ini`. When PHP is invoked, it looks first for `php.ini` in this directory. Although you don't need to, you may wish to edit it to change configuration parameters, including automatically loading extensions. Comments in the file explain the purpose of each configuration directive.

The next step is to make sure the required DLL files are in your path. One way is to copy the two required files to your system directory, such as `c:\winnt\system32`. Alternatively, you can click on the `system` icon in the control panel and add your PHP directory to the system path.

You need to tell IIS that files ending with a particular extension, such as `.php`, should be processed with PHP. IIS calls this process an ISAPI filter. Open the Management Console that allows you to configure all aspects of IIS. One of the tabs for editing a Web server allows you to edit ISAPI filters. Add one. You should call it PHP, and point to the `php4isapi.dll` file, which should be with the rest of the files you installed with PHP. This file is really small, but it loads the PHP core from another library, `php4ts.dll`.

Now that you've added the filter, you need to associate it with an extension. Look for the home directory configuration button in the properties dialog. Add a new entry to the list of application mappings. Choose .php for the extension, and find your `php4isapi.dll` file again. Leave the text box labeled "method exclusions" blank, and check the script engine checkbox.

The last step is to restart the Web server. Stopping it from the management console is not sufficient. You must stop the service itself either from the command line with `net stop w3svc`, or by using the services control panel. After stopping it, restart it.

Editing Scripts

PHP scripts are just text files, and you can edit and create them just as you would HTML files. Certainly, you can telnet into your Web server and start creating files with `vi`. Or you can create files with notepad and use `ftp` to

upload them one by one. But these aren't ideal experiences. One handy feature of newer editors is built-in FTP. These editors can open files on a remote Web server as if they were on a local drive. A single click saves them back to the remote Web server. Another feature you may enjoy is syntax highlighting. This causes PHP keywords to be colored in order to help you read the code faster.

Everyone has a favorite editor for PHP scripts. I use UltraEdit <http://www.ultraedit.com/>. I know many Windows users prefer Macromedia's Dreamweaver

<http://www.macromedia.com/software/dreamweaver/>

and Allaire's HomeSite

<http://www.allaire.com/products/homesite/>

to edit PHP scripts. Quad Systems offers a free package called phpWeave that allows Dreamweaver to build some PHP scripts automatically <http://phpweave.quad-sys.com/>. The Macintosh users I know prefer BBedit <http://www.barebones.com/products/bbedit/bbedit.html>.

On a UNIX operating system, you may prefer emacs or vi, of course. You might also consider nEdit <ftp://ftp.fnal.gov/pub/nedit/>. A module for PHP is available in the `contrib` directory. The topic of which editor is best appears frequently on the PHP mailing list. Reading the archives can be amusing and informative

<http://www.progressive-comp.com/Lists/?l=php3-general>.

Algorithms

Whenever we interact with a computer, we are instructing it to perform some action. When you drag an icon into the waste basket on your desktop, you are asking the computer to remove the file from your hard disk. When you write an HTML file, you are instructing the computer in the proper way to display some information. There are usually many incremental steps to any process the computer performs. It may first clear the screen with the color you specified in the `body` tag. Then it may begin writing some text in a particular color and typeface. As you use a computer, you may not be entirely aware of each tiny step it takes, but you are giving it a list of ordered instructions that you expect it to follow.

Instructions for baking a cake are called a recipe. Instructions for making a movie are called a screenplay. Instructions for a computer are called a program. Each of these is written in its own language, a concrete realization of an abstract set of instructions. Borrowing from mathematics, computer science calls the abstract instructions an algorithm.

You may at this moment have in mind an algorithm that you'd like to implement. Perhaps you wish to display information in a Web browser that changes frequently. Imagine something simple, such as displaying today's date. You could edit a plain HTML file once a day. You could even write out a set of instructions to help remind you of each step. But you cannot perform the task with HTML alone. There's no tag that stands for the current date.

PHP is a language that allows you to express algorithms for creating HTML files. With PHP, you can write instructions for displaying the current date inside an HTML document. You write your instructions in a file called a script. The language of the script is PHP, a language that both you and the computer can understand.

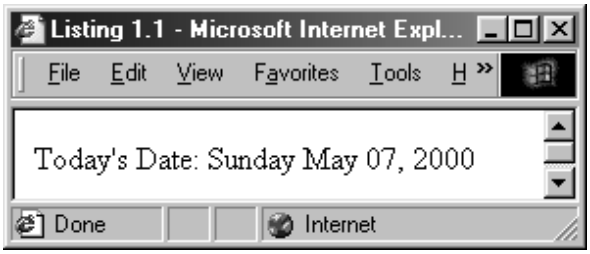
What a PHP Script Looks Like

PHP exists as a tag inside an HTML file. Like all HTML tags, it begins with a less-than symbol, or opening angle bracket (<) and ends with a greater than symbol, or closing angle bracket (>). To distinguish it from other tags, the PHP tag has a question mark (?) following the opening angle bracket and preceding the closing angle bracket. All text outside the PHP tag is simply passed through to the browser. Text inside the tag is expected to be PHP code and is parsed.

To accommodate XML and some picky editors such as Microsoft's Front Page, PHP offers three other ways to mark code. Putting `php` after the opening question mark makes PHP code friendly to XML parsers. Alternatively, you may use a script tag as if you were writing JavaScript. Lastly, you can use tags that appear like ASP, using `<%` to start blocks of code. Appendix D explains how these alternatives work. I use the simple `<?` and `?>` method for all my examples.

Listing 1.1 shows an ordinary HTML page with one remarkable difference: the PHP code between the `<?` and the `?>`. When this page is passed through the PHP module, it will replace the PHP code with today's date. It might read something like, `Friday May 1, 1999`.

Listing 1.1 Printing Today's Date




```
<HTML>
<HEAD>
<TITLE>Listing 1.1</TITLE>
</HEAD>
<BODY>
Today's Date: <? print(Date("l F d, Y")); ?>
</BODY>
</HTML>
```

Whitespace, that is spaces, tabs, and carriage returns, is ignored by PHP. Used judiciously, it can enhance the readability of your code. Listing 1.2 is functionally the same as the previous example, though you may notice more easily that it contains PHP code.

Listing 1.2 Reformatting for Readability

```
<HTML>
<HEAD>
<TITLE>Listing 1-2</TITLE>
</HEAD>
<BODY>
Today's Date:
<?
       /*
       ** print today's date
       */
       print(Date("l F d, Y"));
?>
</BODY>
</HTML>
```

You may also notice that in Listing 1.2 there is a line of code that begins with a slash followed by an asterisk. This is a comment. Everything between the /* and the */ is equivalent to whitespace. It is ignored. Comments can be used to document how your code works. Even if you maintain your own code you will find comments necessary for all but simple scripts.

In addition to the opening and closing comment statements, PHP provides two ways to build a single-line comment. Double-slashes or a pound sign will cause everything after them to the end of the line to be ignored by the parser.

After skipping over the whitespace and the comment in Listing 1.2, the PHP parser encounters the first word: `print`. This is one of PHP's functions. A function collects code into a unit you may invoke with its name. The `print` function sends text to the browser. The contents of the parentheses will be evaluated, and if it produces output, `print` will pass it along to the browser.

Where does the line end? Unlike BASIC and JavaScript, which use a line break to denote the end of a line, PHP uses a semicolon. On this issue PHP takes inspiration from C.

The contents of the line between `print` and `;` is a call to a function named `date`. The text between the opening and closing parentheses is the parameter passed to `date`. The parameter tells `date` in what form you want the date to appear. In this case we've used the codes for the weekday name, the full month name, the day of the month, and the four-digit year. The current date is formatted and passed back to the `print` function.

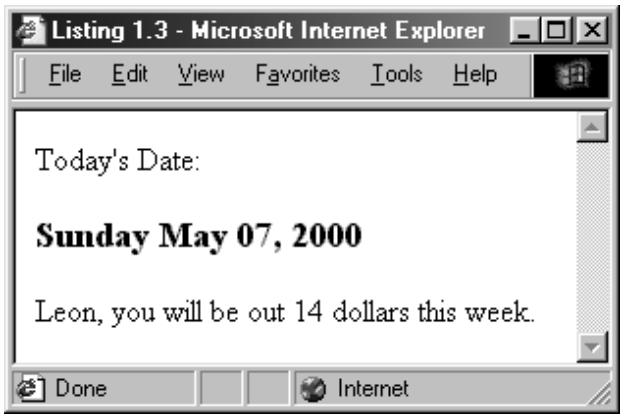
The string of characters beginning and ending with double quotes is called a string constant or string literal. PHP knows that when quotes surround characters you intend them to be treated as text. Without the quotes, PHP will assume you are naming a function or some other part of the language itself. In other words, the first quote is telling PHP to keep hands off until it finds another quote.

Notice that `print` is typed completely in lowercase letters, yet `date` has a leading uppercase letter. I did this to illustrate that PHP takes a very lenient attitude toward the names of its built-in functions. `Print`, `PRINT`, and `PrInT` are all valid calls to the same function. However, for the sake of readability, it is customary to write PHP's built-in functions using lowercase letters only.

Saving Data for Later

Often it is necessary to save information for later use. PHP, like most programming languages, offers the concept of variables. Variables give a name to the information you want to save and manipulate. Listing 1.3 expands on our example by using variables.

Listing 1.3 Assigning Values to Variables




```
<?
       $YourName = "Leon";
       $Today = date("l F d, Y");
       $CostOfLunch = 3.50;
       $DaysBuyingLunch = 4;
?>
<HTML>
<HEAD>
<TITLE>Listing 1-3</TITLE>
</HEAD>
<BODY>
Today's Date:
<?
       /*
       ** print today's date
       */
       print("<H3>$Today</H3>\n");

       /*
       ** print message about lunch cost
       */
       print("$YourName, you will be out ");
       print($CostOfLunch * $DaysBuyingLunch);
       print(" dollars this week.<BR>\n");
?>
</BODY>
</HTML>
```

The first block of PHP code puts values into some variables. The four variables are `YourName`, `Today`, `CostOfLunch`, and `DaysBuyingLunch`. PHP knows they are variables because they are preceded by a dollar sign (`$`). The first time you use a variable in a PHP script, some memory is set aside to store the information you wish to save. You don't need to tell PHP what kind of information you expect to be saved in the variable; PHP can figure this out on its own.

The script first puts a character string into the variable `YourName`. As I noted earlier, PHP knows it's textual data because I put quotes around it. Likewise I put today's date into a variable named `Today`. In this case PHP knows to put text into the variable because the `date` function returns text. This type of data is referred to as a string, which is shorthand for character string. A character is a single letter, number, or any other mark you make by typing a single key on your keyboard.

Notice that there is an equal sign (`=`) separating the variable and the value you put into it. This is the assignment operator. Everything to its right is put into a variable named to its left.

The third and fourth assignments are putting numerical data into variables. The value 3.5 is a floating-point, or fractional, number. PHP calls this type a double, showing some of its C heritage. The value 4 in the next assignment is an integer, or whole number.

After printing some HTML code, another PHP code block is opened. First the script prints today's date as a level-three header. Notice that the script passes some new types of information to the `print` function. You can give string literals or string variables to `print` and they will be sent to the browser.

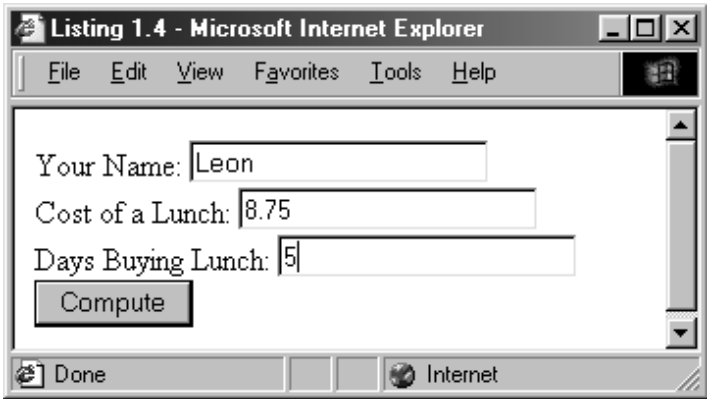
When it comes to variables, PHP is not so lenient with case. `Today` and `today` are two different variables. Since PHP doesn't require you to declare variables before you use them, you can accidentally type `today` when you mean `Today` and no error will be generated. If variables are unexpectedly empty, check your case.

The script next prints `Leon, you will be out 14.00 dollars this week`. The line that prints the total has to calculate it with multiplication using the `*` operator.

Receiving User Input

Manipulating variables that you set within your script is somewhat interesting, but hardly anything to rave about. Scripts become much more useful when they use input from the user. When you call PHP from an HTML

Listing 1.4 HTML Form for Lunch Information

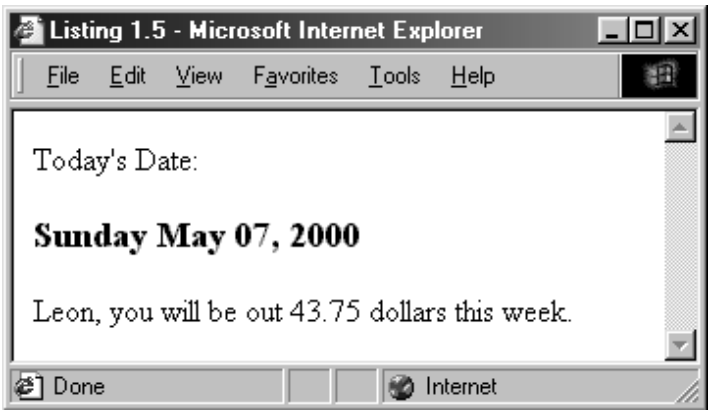



```
<HTML>
<HEAD>
<TITLE>Listing 1-4</TITLE>
</HEAD>
<BODY>
<FORM ACTION="1-5.php" METHOD="post">
Your Name:
<INPUT TYPE="text" NAME="YourName"><BR>
Cost of a Lunch:
<INPUT TYPE="text" NAME="CostOfLunch"><BR>
Days Buying Lunch:
<INPUT TYPE="text" NAME="DaysBuyingLunch"><BR>
<INPUT TYPE="submit" NAME="x" VALUE="Compute">
</FORM>
</BODY>
</HTML>
```

form, the form fields are turned into variables. Listing 1.4 is a form that calls Listing 1.5, a further modification of our example script.

Listing 1.4 is a standard HTML form. If you have dealt at all with CGIs, it will look familiar. There are three form fields that match up with the variables from our previous example. Instead of simply putting data into the variables, we will provide a form and use the information the user types. When the user presses the submit button, the script named in the `ACTION` attribute will receive the three form fields and PHP will convert them into variables.

Listing 1.5 Computing the Cost of Lunch from a Form




```
<?
       $Today = date("l F d, Y");
?>
<HTML>
<HEAD>
<TITLE>Listing 1-5</TITLE>
</HEAD>
<BODY>
Today's Date:
<?
       /*
       ** print today's date
       */
       print("<H3>$Today</H3>\n");

       /*
       ** print message about lunch cost
       */
       print("$YourName, you will be out ");
       print($CostOfLunch * $DaysBuyingLunch);
       print(" dollars this week.<BR>\n");
?>
</BODY>
</HTML>
```

Notice that in the first segment of the PHP script, I have eliminated the lines setting the variables, except for today's date. The rest of the script is unchanged. The script assumes there will be data in the variables. Try experimenting with the scripts by entering nonsense in the form fields.

One thing you should notice is that if you put words where the script expects numbers, PHP seems to just assign them values of zero. The variables are set with a text string, and when the script tries to treat it as a number, PHP does its best to convert the information. Entering `10 Little Indians` for the cost of lunch will be interpreted as 10.

Listing 1.6 Conditional Daily Message


```
<HTML>
<HEAD>
<TITLE>Listing 1-6</TITLE>
</HEAD>
<BODY>
<H1>
<?
       /*
       ** get today's day of the week
       */
       $Today = date("l");

       if($Today == "Friday")
       {
         print("Thank Goodness It's Friday!");
       }
       else
       {
         print("Today is $Today.");
       }
?>
</H1>
</BODY>
</HTML>
```

Choosing between Alternatives

PHP allows you to test conditions and execute certain code based on the result of the test. The simplest form of this is the `if` statement. Listing 1.6 shows how you can customize the content of a page based on the value of a variable.

The `Today` variable is set with the name of today's weekday. The `if` statement evaluates the expression inside the parentheses as either true or false. The `==` operator compares the left side to the right side. If `Today` contains the word `Friday`, the block of code surrounded by curly braces (`{` and `}`) is executed. In all other cases the block of code associated with the `else` statement is executed.

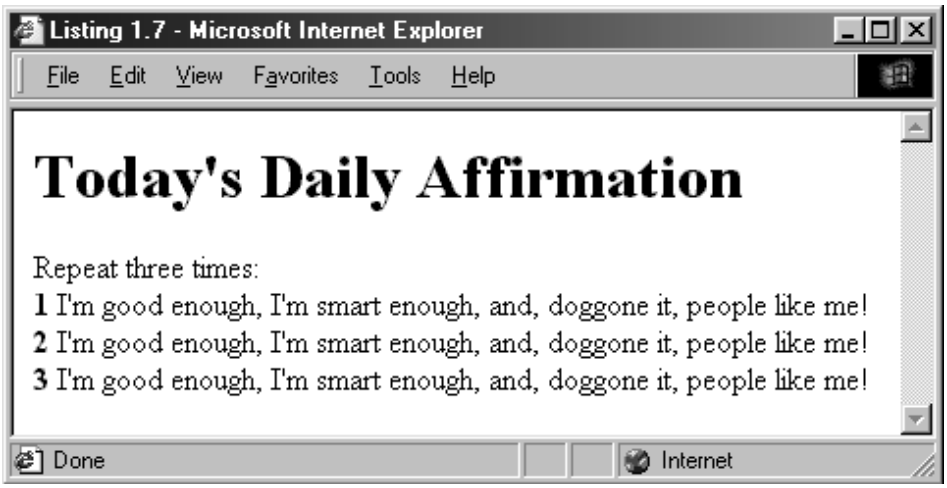
Repeating Code

The last type of functionality in this brief introduction is looping. Looping allows you to repeat the execution of code. Listing 1.7 is an example of a `for` loop. The `for` statement expects three parameters separated by semicolons. The first parameter is executed once before the loop begins. It usually initializes a variable. The second parameter makes a test. This is usually a test against the variable named in the first parameter. The third parameter is executed every time the end of the loop is reached.

The `for` loop in Listing 1.7 will execute three times. The initialization code sets the variable `count` to be one. Then the testing code compares the value of count to three. Since one is less than or equal to three, the code inside the loop executes. Notice that the script prints the value of `count`. When you run this script you will find that `count` will progress from one to three. The reason is that the third part of the `for` statement is adding one to `count` each time through the loop. The `++` operator increments the variable immediately to its left.

The first time through the loop `count` is one, not two. This is because the increment of `count` doesn't occur until we reach the closing curly brace. After the third time through the loop, `count` will be incremented to four, but at that point four will not be less than or equal to three, so the loop will end. Execution continues at the command following the loop code block.

Listing 1.7 Today's Daily Affirmation




```
<HTML>
<HEAD>
<TITLE>Listing 1-7</TITLE>
</HEAD>
<BODY>
<H1>Today's Daily Affirmation</H1>
Repeat three times:<BR>
<?
       for($count = 1; $count <= 3; $count++)
       {
         print("<B>$count</B> I'm good enough, ;");
         print("I'm smart enough, ");
         print("and, doggone it, people like me!<BR>\n");
       }
?>
</BODY>
</HTML>
```

Conclusion

Hopefully this chapter has convinced you of the power of PHP. You have seen some of the major features of the language and read arguments why PHP is better than its many competitors. The rest of the book will look at PHP in more detail.