Unit – Four

The Server Tier

There are lots of great reasons to write computer programs in PHP. Maybe you want to learn PHP because you need to put together a small web site for yourself that has some interactive elements. Perhaps PHP is being used where you work and you have to get up to speed. This chapter provides context for how PHP fits into the puzzle of web site construction: *what it can do and why it's so good at what it does. You'll also get your first look at the PHP language and see it in action.*

PHP's Place in the Web World: (Web Server and Creating Dynamic Content)

PHP is a programming language that's used mostly for building web sites. Instead of a PHP program running on a desktop computer for the use of one person, it typically runs on a web server and is accessed by lots of people using web browsers on their own computers. This section explains how PHP fits into the interaction between a web browser and a web server. When you sit down at your computer and pull up a web page using a browser such as Internet Explorer or Mozilla, you cause a little conversation to happen over the Internet between your computer and another computer. This conversation and how it makes a web page appear on your screen is illustrated in Figure below.

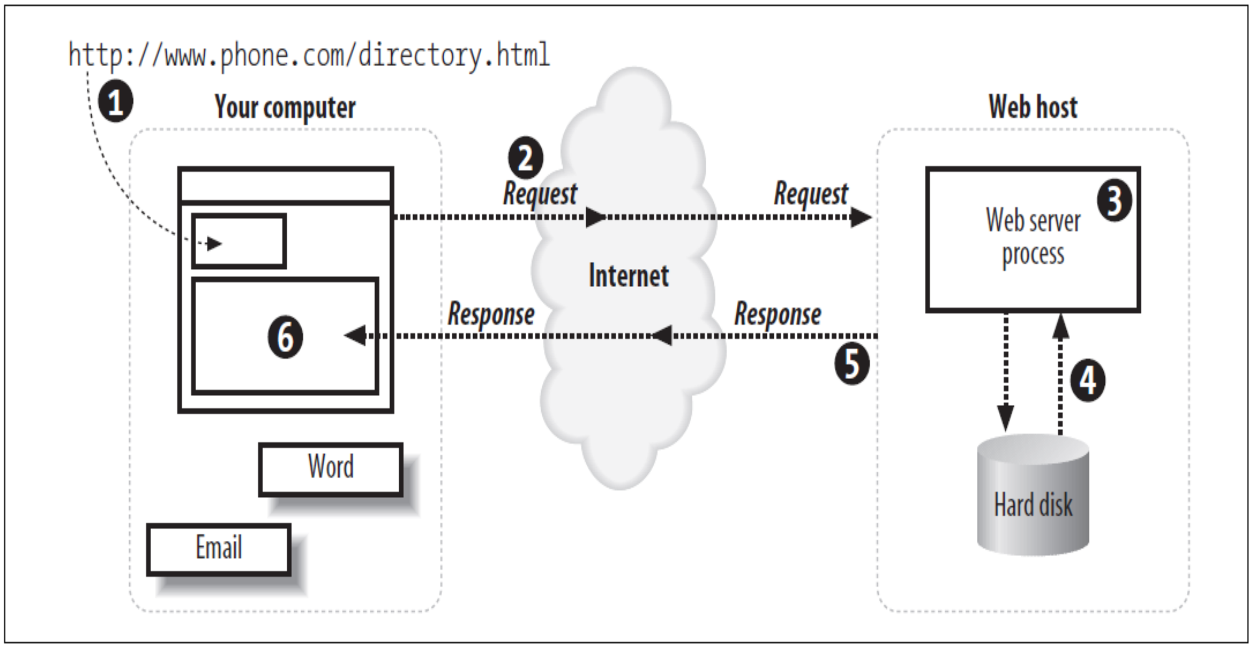

Figure 1-1 While the user only types in a URL and hits Enter, there are several steps that occur

Here’s the breakdown of the figure:

1. You enter a web page address in your browser’s location bar.

2. Your browser breaks apart that address and sends the name of the page to the web server. For example, *http://www.phone.com/directory.html* would request the page *directory.html* from *www.phone.com*.

3. A program on the web server, called the *web server process*, takes the request for *directory.html* and looks for this specific file.

4. The web server reads the *directory.html* file from the web server’s hard drive.

5. The web server returns the contents of *directory.html* to your browser.

6. Your web browser uses the HTML markup that was returned from the web server to build the rendition of the web page on your computer screen.

The HTML file called *directory.html* (requested in Figure 1-1) is called a ***static web page because everyone who requests the directory.html page gets exactly the same page.***

For the web server to customize the returned page, PHP and MySQL are added to the mix. Figure 1-2 illustrates the extra steps that occur in the chain of events on the web host.

Each step in the chain is listed here:

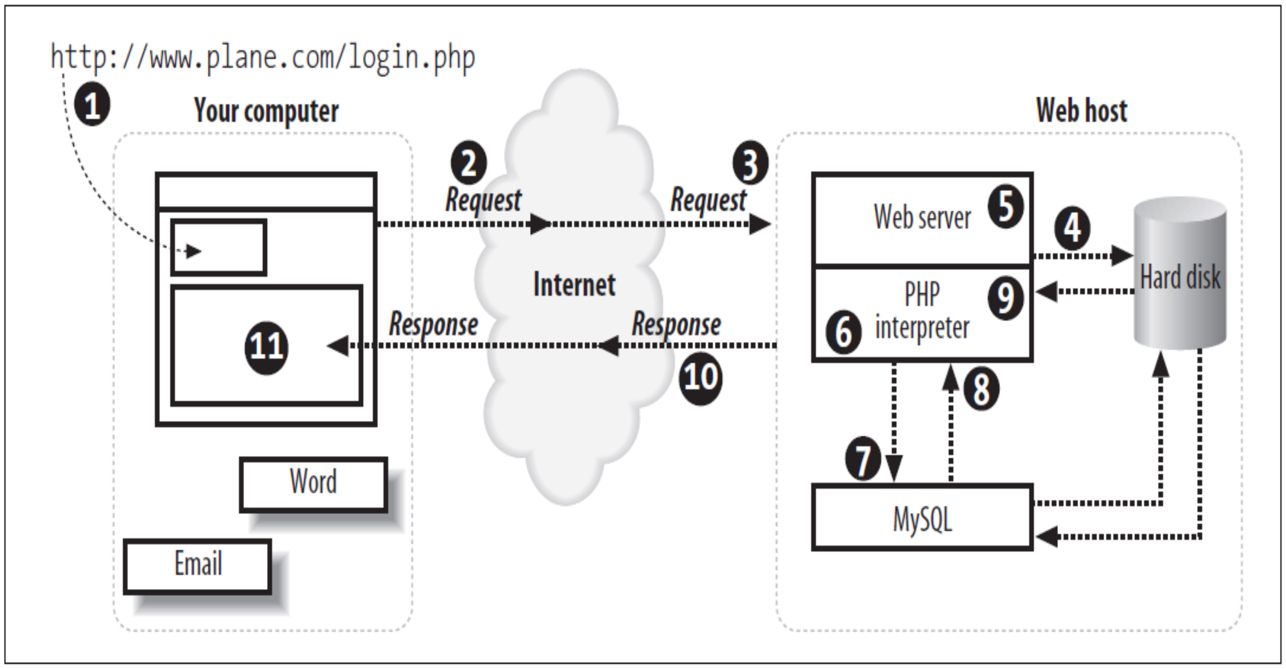

Figure 2-2 The PHP interpreter, MySQL, and the web server cooperate to return the page

1. You enter a web page address in your browser's location bar.

2. Your browser breaks apart that address and sends the name of the page to the host. For example, *http://www.phone.com/login.php* requests the page *login.php* from *www.phone.com*.

3. The web server process on the host receives the request for *login.php*.

4. The web server reads the *login.php* file from the host's hard drive.

5. The web server detects that the PHP file isn't just a plain HTML file, so it asks another process—the PHP interpreter—to process the file.

6. The PHP interpreter executes the PHP code that it finds in the text it received from the web server process. Included in that code are calls to the MySQL database.

7. PHP asks the MySQL database process to execute the database calls.

8. The MySQL database process returns the results of the database query.

9. The PHP interpreter completes execution of the PHP code with the data from the database and returns the results to the web server process.

10. The web server returns the results in the form of HTML text to your browser.

11. Your web browser uses the returned HTML text to build the web page on your screen.

This may seem like a lot of steps, but all of this processing happens automatically every time a web page with PHP code is requested. In fact, this process may happen several times for a single web page, since a web page can contain many image files and the CSS definition, which must all be retrieved from the web server. When developing dynamic web pages, you work with a variety of variables and server components, which are all important to having an attractive, easy-to-navigate, and maintainable web site.

What's So Great About PHP?

You may be attracted to PHP because it's free, because it's easy to learn, or because your boss told you that you need to start working on a PHP project next week. Since you're going to use PHP, you need to know a little bit about what makes it special. The next time someone asks you "What's so great about PHP?", use this section as the basis for your answer.

The answer are as follows:

1. **PHP Is Free (as in Money):**
   You don't have to pay anyone to use PHP. Whether you run the PHP interpreter on a beat-up 10-year-old PC in your basement or in a room full of million dollar "enterprise-class" servers, there are no licensing fees, support fees, maintenance fees, upgrade fees, or any other kind of charge. Most Linux distributions come with PHP already installed. If yours doesn't, or you are using another operating system such as Windows, you can download PHP from http://www.php.net/. Appendix A has detailed instructions on how to install PHP.

2. **PHP Is Free (as In Speech):**
   As an open-source project, PHP makes its innards available for anyone to inspect. If it doesn't do what you want, or you're just curious about why a feature works the way it does, you can poke around in the guts of the PHP interpreter (written in the C programming language) to see what's what. Even if you don't have the technical expertise to do that, you can get someone who does to do the investigating for you. Most people can't fix their own cars, but it's nice to be able to take your car to a mechanic who can pop open the hood and fix it.

3. **PHP Is Cross-Platform:**
   You can use PHP with a web server computer that runs Windows, Mac OS X, Linux, Solaris, and many other versions of Unix. Plus, if you switch web server operating systems, you generally don't have to change any of your PHP programs. Just copy them from your Windows server to your Unix server, and they will still work. While Apache is the most popular web server program used with PHP, you can also use Microsoft Internet Information Server and any other web server that supports the CGI standard. PHP also works with a large number of databases including MySQL, Oracle, Microsoft SQL Server, Sybase, and PostgreSQL. In addition, it supports the ODBC standard for database interaction. If all the acronyms in the last paragraph freak you out, don't worry. It boils down to this: whatever system you're using, PHP probably runs on it just fine and works with whatever database you are already using.

4. **PHP Is Widely Used:**
   As of March 2023, PHP is installed on more than 500 million different web sites, from countless tiny personal home pages to giants like Yahoo!. There are many books, magazines, and web sites devoted to teaching PHP and exploring what you can do with it. There are companies that provide support and training for PHP. In short, if you are a PHP user, you are not alone.

5. **PHP Hides Its Complexity:**
   You can build powerful e-commerce engines in PHP that handle millions of customers. You can also build a small site that automatically maintains links to a changing list of articles or press releases. When you're using PHP for a simpler project, it doesn't get in your way with concerns that are only relevant in a massive system. When you need advanced features, such as caching, custom libraries, or dynamic image generation, they are available. If you don't need them, you don't have to worry about them. You can just focus on the basics of handling user input and displaying output.

6. **PHP is Built for Web Programming:**
   Unlike most other programming languages, PHP was created from the ground up for generating web pages. This means that common web programming tasks, such as accessing form submissions and talking to a database, are often easier in PHP. PHP comes with the capability to format HTML, manipulate dates and times, and manage web

cookies tasks that are often available only as add-on libraries in other programming languages.

The Components of a PHP Application:

In order to process and develop dynamic web pages, you'll need to use and understand several technologies. There are three main components of creating dynamic web pages: a web server, a server-side programming language, and a database. It's a good idea to have an understanding of these three basic components for web development using PHP. We'll start with some rudimentary understanding of the history and purpose of Apache (your web server), PHP (your server-side programming language), and MySQL (your database). This can help you to understand how they fit into the web development picture.
Remember that dynamic web pages pull information from several sources simultaneously, including Apache, PHP, MySQL, and Cascading Style Sheets (CSS), which we'll talk about later.

PHP:

PHP grew out of a need for people to develop and maintain web sites containing dynamic client-server functionality. In 1994, Rasmus Lerdorf created a collection of open-source Perl scripts for his personal use, and these eventually were rewritten in C and turned into what PHP is today. By 1998, PHP was released in its third version, turning it into a web development tool that could compete with similar products such as Microsoft's Active Server Pages (ASP) and Sun's Java Server Pages (JSP). PHP also is an interpreted language, rather than a compiled one. The real beauty of PHP is simplicity coupled with power.

Note: *Compiled languages create a binary file such as an .exe, while interpreted languages work directly with the source code when executing, as opposed to creating a standalone file.*

PHP is ubiquitous and compatible with all major operating systems. It is also easy to learn, making it an ideal tool for web programming beginners. Additionally, you get to take advantage of a community's effort to make web development easier for everyone. The creators of PHP developed an infrastructure that allows experienced C programmers to extend PHP's abilities. As a result, PHP now integrates with advanced technologies like XML, XSL, and Microsoft's Component Object Model Technologies (COM).

Apache:

Apache is a web server that turns browser requests into resulting web pages and knows how to process PHP code. PHP is only a programming language, so without the power of a web server like Apache behind it, there would be no way for web users to reach your pages containing the PHP language code. Apache is not the only web server available. Another popular web server is Microsoft's Internet Information Services (IIS), which is supplied with Windows 2000 and all later versions. Apache has the decided advantages of being free, providing full source code, and using an unrestricted license. Apache 2.0 is the current version you would most likely be using, though 1.3 is often still used. IIS is easier to integrate with Active Directory, Microsoft's latest authentication system, but this applies mostly to internal company web sites.

Note: *According to the Netcraft web server survey, Apache has been the most popular web server on the Internet since April 1996.*

Because web servers like Apache and IIS are designed to serve up HTML files, they need a way to know how to process PHP code. Apache uses *modules* to load extensions into its functionality. IIS uses a similar concept called Internet Server Application Program Interface (ISAPI). These both allow for faster processing of the PHP code than the old-school process of calling PHP as a separate executable each time the web server had a request for a page containing PHP.

SQL and Relational Databases:

Structured Query Language (SQL) is the most popular language used to create, retrieve, update, and delete data from relational database management systems. A *relational* database conforms to the relational model and refers to a database's data and schema. The *schema* is the database's structure of how data is arranged. Common usage of the term "Relational Database Management System" technically refers to the software used to create a relational database, such as Oracle or Microsoft SQL Server.
A relational database is a collection of tables, but other items are frequently considered part of the database, as they help organize and structure the data in addition to forcing the database to conform to a set of requirements.

MySQL:

MySQL is a free yet full-featured relational database. MySQL was developed in the 1990s to fill the ever-growing need for computers to manage information intelligently. The original core MySQL developers were trying to solve their needs for a database by using mSQL, a small and simple database. It became clear that mSQL couldn't solve all the problems they wanted it to, so they created a more robust database that turned into MySQL.

MySQL supports several different *database engines*. Database engines determine how MySQL handles the actual storage and querying of the data. Because of that, each storage engine has its own set of abilities and strengths. Over time, the database engines available are becoming more advanced and faster.

Compatibility:

Web browsers such as Safari, Firefox, Netscape, and Internet Explorer are made to process HTML, so it doesn't matter which operating system a web server runs on. Apache, PHP, and MySQL support a wide range of operating systems (OS), so you aren't restricted to a specific OS on either the server or the client. While you don't have to worry much about software compatibility, the sheer variety of file formats and different languages that all come together does take some getting used to.

Note: Installation tutorial

PHP in Action:

Ready for your first taste of PHP? This section contains a few program listings and explanations of what they do. If you don't understand everything going on in each listing, don't worry! That's what the rest of the book is for. Read these listings to get a sense of what PHP programs look like and an outline of how they work. Don't sweat the details yet.

When given a program to run, the PHP interpreter pays attention only to the parts of the program between PHP start and end tags. Whatever's outside those tags is printed with no modification. This makes it easy to embed small bits of PHP in pages that mostly contain HTML. The PHP interpreter runs the commands between <?php (the PHP start tag) and ?> (the PHP end tag). PHP pages typically live in files whose names end in *.php*.

Example: (Printing Hello World)

```
<html>
<head>
    <title> PHP Says Hello </title>
</head>
<body>
     <b>
         <?php
              echo “Hello World”;
         ?>
     </b>
</body>
</html>
```

Basic Rules of PHP Programs:

This section lays out some ground rules about the structure of PHP programs. More foundational than the basics such as "how do I print something" or "how do I add two numbers", these proto-basics are the equivalent of someone telling you that you should read pages in this book from top to bottom and left to right, or that what's important on the page are the black squiggles, not the large white areas.

Rules are:

1. **Start and End Tags:**

   Each of the examples you've already seen in this chapter uses <?php as the PHP start tag and ?> as the PHP end tag. The PHP interpreter ignores anything outside of those tags. Text before the start tag or after the end tag is printed with no interference from the PHP interpreter.

   A PHP program can have multiple start and end tag pairs, as shown in following example

```
Five plus five is:
 <?php echo 5 + 5; ?>
    <p>
        Four plus four is:
        <?php
                echo 4 + 4;
         ?>
     </p>
    <img src="vacation.jpg" alt="My Vacation">
```

2. **Whitespace and Case-Sensitivity:**
   Like all PHP programs, the examples in this section consist of a series of statements, each of which end with a semicolon. You can put multiple PHP statements on the same line of a program as long as they are separated with a semicolon. You can put as many blank lines between statements as you want. The PHP interpreter ignores them. The semicolon tells the interpreter that one statement is over and another is about to begin. No whitespace at all or lots and lots of whitespace between statements doesn't affect the program's execution. (*Whitespace* is programmer-speak for blank-looking characters such as space, tab, and newline.)

   In practice, it's good style to put one statement on a line and to put blank lines between statements only when it improves the readability of your source code. The spacing in Examples Example 1 and Example 2 is bad. Instead, format your code as in Example 3.

   Example 1: This PHP is too cramped

```
<?php echo "Hello"; echo " World!"; ?>
```

   Example 2: This PHP is too sprawling

```
<?php



    echo "Hello";

   echo " World!";

?>
```

Example 3: This PHP is just right

```
<?php
    echo "Hello";
    echo " World!";
?>
```

In addition to ignoring whitespace between lines, the PHP interpreter also ignores whitespace between language keywords and values. You can have zero spaces, one space, or a hundred spaces between echo and "Hello, World!" and again between "Hello, World!" and the semicolon at the end of the line.

Good coding style is to put one space between print and the value being printed and then to follow the value immediately with a semicolon. Example 4 shows three lines, one with too much spacing, one with too little, and one with just the right amount.

Example 4: Spacing

```
<?php
echo                    "Too many spaces";
echo"Too few spaces";
echo "Just the right number of spaces";
?>
```

3. **Keywords and function names are case-insensitive:**
   Language keywords (such as print) and function names (such as number_format) are not case-sensitive. The PHP interpreter doesn't care whether you use uppercase letters, lowercase letters, or both when you put these keywords and function names in your programs. The statements in Example 5 are identical from the interpreter's perspective.

Example 5:

```
// These four lines all do the same thing
echo number_format(285266237);
ECHO Number_Format(285266237);
Echo number_format(285266237);
eChO NUMBER_FORMAT(285266237);
```

4. **Comments:**

As you've seen in some of the examples in this chapter, comments are a way to explain to other people how your program works. Comments in source code are an essential part of any program. When you're coding, what you are writing may seem crystal clear to you at the time. A few months later, however, when you need to go back and modify the program, your brilliant logic may not be so obvious. That's where comments come in. By explaining in plain language how the programs work, comments make programs much more understandable.

Comments are even more important when the person who needs to modify the program isn't the original author. Do yourself and anyone else who might have occasion to read your source code a favor and fill your programs with a lot of comments.

Perhaps because they're so important, PHP provides many ways to put comments in your programs. One syntax you've seen already is to begin a line with //. This tells the PHP interpreter to treat everything on that line as a comment. After the end of the line, the code is treated normally. This style of comment is also used in other programming languages such as C++, JavaScript, and Java. You can also put // on a line after a statement to have the remainder of the line treated as a comment. PHP also supports the Perland shell-style single-line comments. These are lines that begin with #. You can use # to start a comment in the same places that you can use //, but the modern style prefers // over #.

Comments used are as follows:

a) Single-line comments (// or #) e.g. //comment goes here
b) Multiline comments (/* comment goes here */)

Working with Text and Numbers

PHP can work with different types of data. In this chapter, you'll learn about individual values such as numbers and single pieces of text. You'll learn how to put text and numbers in your programs, as well as some of the limitations the PHP interpreter puts on those values and some common tricks for manipulating them.

Most PHP programs spend a lot of time handling text because they spend a lot of time generating HTML and working with information in a database. HTML is just a specially formatted kind of text, and information in a database, such as a username, a product description, or an address is a piece of text, too. Slicing and dicing text easily means you can build dynamic web pages easily.

In Chapter 1, you saw variables in action, but this chapter teaches you more about them. A variable is a named container that holds a value. The value that a variable hold can change as a program runs. When you access data submitted from a form or exchange data with a database, you use variables. In real life, a variable is something such as your checking account balance. As time goes on, the value that the phrase "checking account balance" refers to fluctuates. In a PHP program, a variable might hold the value of a submitted form parameter. Each time the program runs, the value of the submitted form parameter can be different. But whatever the value, you can always refer to it by the same name. This chapter also explains in more detail what variables are: how you create them and do things such as change their values or print
them.

2.1 Text

When they're used in computer programs, pieces of text are called *strings*. This is because they consist of individual characters, strung together. Strings can contain letters, numbers, punctuation, spaces, tabs, or any other characters. Some examples of strings are I would like 1 bowl of soup, and "Is it too hot?" he asked, and There's no spoon! A string can even contain the contents of a binary file such as an image or a sound. The only limit to the length of a string in a PHP program is the amount of memory your computer has.

2.1.1 Defining Text Strings

There are a few ways to indicate a string in a PHP program. The simplest is to surround the string with single quotes:

```
echo 'I would like a bowl of soup.';
echo 'chicken';
echo '06520';
echo '"I am eating dinner," he growled.';
```

The single quotes aren't part of the string. They are *delimiters*, which tell the PHP interpreter where the start and end of the string is. If you want to include a single quote inside a string surrounded with single quotes, put a backslash (\) before the single quote inside the string:

```
echo  'We\'ll each have a bowl of soup.';
```

The \' sequence is turned into ' inside the string, so what is printed is:

```
We'll each have a bowl of soup.
```

The backslash tells the PHP interpreter to treat the following character as a literal single quote instead of the single quote that means "end of string." This is called *escaping*, and the backslash is called the *escape character*. An escape character tells the system to do something special with the character that comes after it. Inside a single-quoted string, a single quote usually means "end of string." Preceding the single quote with a backslash changes its meaning to a literal single quote character.

Note: The biggest difference between single-quoted and double-quoted strings is that when you include variable names inside a double-quoted string, the value of the variable is substituted into the string, which doesn't happen with single quoted strings. For example, if the variable $user held the value Bill, then 'Hi $user' is just that: Hi $user. However, "Hi $user" is Hi Bill.

Example on Here Document and printing a here document refer to code

2.1.2 Manipulating Text

PHP has a number of built-in functions that are useful when working with strings. This section introduces the functions that are most helpful for two common tasks: validation and formatting. The "Strings" chapter of the PHP online manual, at http://www.php.net/strings, has information on other built in string handling functions.

2.1.2.1 Validating Strings

Validation is the process of checking that input coming from an external source conforms to an expected format or meaning. It's making sure that a user really entered a ZIP Code in the "ZIP Code" box of a form or a reasonable email address in the appropriate place. Coming chapter on form delves into all the aspects of form handling, but since submitted form data is provided to your PHP programs as strings, this section discusses how to validate those strings. The trim() function removes whitespace from the beginning and end of a string. Combined with strlen( ), which tells you the length of a string, you can find out the length of a submitted value while ignoring any leading or trailing spaces.

2.1.2.2 Formatting Text

The printf( ) function gives you more control (compared to print) over how the output looks. You pass printf( ) a format string and a bunch of items to print. Each rule in the format string is replaced by one item. Example below shows printf( ) in action.

Example:
$price = 5; $tax = 0.075;
printf('The dish costs $%.2f', $price * (1 + $tax));

This prints:
The dish costs $5.38

Example: Changing case (Refer to code)

- strtolower()
- strtoupper()

Example: Prettifying name with: (Refer to code)

- ucwords()
- ucfirtst()
- lcfirst()

Example: Substring (Refer to code)

- Truncating with substr()
- Extracting the end of string substr()
- Using str_replace()

2.2 Numbers

Numbers in PHP are expressed using familiar notation, although you can't use commas or any other characters to group thousands. You don't have to do anything special to use a number with a decimal part as compared to an integer.

Examples: Refer to code

2.2.1 Using Different Kinds of Numbers

Internally, the PHP interpreter makes a distinction between numbers with a decimal part and those without one. The former is called *floating-point* numbers and the latter are called *integers*. Floating-point numbers take their name from the fact that the decimal point can "float" around to represent different amounts of precision.

The PHP interpreter uses the math facilities of your operating system to represent numbers so the largest and smallest numbers you can use, as well as the number of decimal places you can have in a floating-point number, vary on different systems.

One distinction between the PHP interpreter's internal representation of integers and floating-point numbers is the exactness of how they're stored. The integer 47 is stored as exactly 47. The floating-point number 46.3 could be stored as 46.2999999.

2.2.2 Arithmetic Operators

Doing math in PHP is a lot like doing math in elementary school, except it's much faster. Some basic operations between numbers are shown in example below:

Example 2-16. Math operations

```
echo  2 + 2; //4
echo 17 - 3.5; //13.5
echo 10 / 3; //3.333333
echo 6 * 9; //54
```

In addition to the plus sign (+) for addition, the minus sign (-) for subtraction, the forward slash (/) for division, and the asterisk (*) for multiplication, PHP also supports the percent sign (%) for modulus division. This returns the remainder of a division operation:

```
echo 17 % 3; //2
```

2.3 Variables

Variables hold the data that your program manipulates while it runs, such as information about a user that you've loaded from a database or entries that have been typed into an HTML form. In PHP, variables are denoted by $ followed by the variable's name. To assign a value to a variable, use an equal’s sign (=). This is known as the assignment operator.

Example:

```
$plates = 5;
$dinner = 'Chicken Chow-min';
$cost_of_dinner = 8.95;
```

Note: More about variables are discussed in code file.