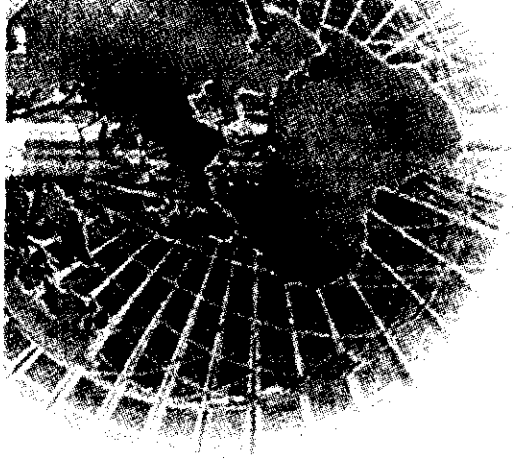


3

Introducing PHP

PHP: Hypertext Preprocessor (PHP) is a server side scripting language that is used to create dynamic Web pages. It was created by the Danish programmer, Rasmus Lerdorf, in the year 1995, and was originally called as Personal Home Page. Lerdorf initially created a set of Common Gateway Interface binaries, written in C language, to replace some Perl scripts in his homepage. He later combined these binaries with his Form Interpreter (FI) to create PHP/FI. Although, not officially launched, this PHP/FI was the first version of the PHP language. PHP/FI had the capability to communicate with database and develop dynamic Web pages. PHP is free software released under PHP License, which does not adhere to the GNU General Public License (GPL) norms.

In this chapter, you learn about the PHP language along with the advantages that it offers over other scripting languages. You also learn how to create and run a PHP script and error handling in PHP as well as the use of escape characters in PHP.

Versions of PHP

Depending upon the increasing and changing requirements, developers release new versions of a language with enhanced features and capabilities on a continuous basis. The following is the list of various versions of PHP released so far:

- **PHP Version 2**—Released on June 8, 1995 by Lerdorf. This version allowed to embed a PHP script in an (Hypertext Markup language) HTML file. It also provided the facility of form handling in Web pages.
- **PHP Version 3**—Released in the year 1997. The parser for PHP was rewritten by Zeev Surazki and Andy Gutaman, which formed the base for PHP 3. This was the time when PHP was given a new name, PHP: Hypertext Preprocessor. PHP/FI 2 was officially released in November 1997 and later on PHP version 3 was launched in the year 1998 after public testing..
- **PHP Version 4**—Released on May 22, 2000 with the Zend Engine 1.1. Various subversions of PHP version 4 are released till August 2008; however, further developments on this version have been stopped. The last released subversion of PHP 4 was PHP 4.4.9.
- **PHP Version 5**—Released on July 13, 2004 with an enhanced version of the Zend Engine, Zend Engine II. This version supported various advanced features, such as Object Oriented Programming, Data Object Extensions. The most recently released version of PHP 5 is PHP 5.2.10. Further developments under this version are still in progress.
- **PHP Version 6**—This version is still under development.

Features of PHP

As you know, PHP is a general-purpose scripting language that runs on a Web server, which is configured to take PHP code as input and create Web page content as output. It can be deployed on most of the Web servers and on almost every operating system and platform free of charge. The following are some of the important features of PHP:

- **Access control**—Provides a built-in Web-based configuration screen to handle access control configuration. Depending upon the client's domain, browser, e-mail address, or the referring document, various restrictions can be placed on the Web pages, such as password protected, completely restricted, and logging disabled.
- **File upload support**—Allows users to upload files to a Web server. PHP provides the Multipurpose Internet Mail Extensions (MIME) decoding process to upload the files onto a server.
- **HTTP-based authentication control**—Allows the user to create customized (Hypertext Transfer Protocol) HTTP-based authentication mechanisms for the Web server.
- **Variables, arrays, and associative arrays**—Support variables, arrays, and associative arrays that can be passed from one Web page to another using either the GET or POST method forms. You can learn about associative arrays in Chapter 4, *Working with Arrays*.

- **Conditional statements and loops**—Provide features similar to the C language and allows you to work with various conditional statements, such as if, elseif, else, as well as loops, such as for, foreach, while, and do while.
- **Extended regular expressions**—Support all common regular expression operations. Regular expressions are mainly used for pattern matching, pattern substitutions, and general string manipulation.
- **Raw HTTP header control**—Allows to transfer a Uniform Resource Locator (URL) from one client to another client. It also used to manipulate the latest updated header of the Web pages.
- **Access logging**—Allows you to record the number of times a Web page or a website is accessed. In addition, it also helps to generate footer on every page, displaying the access information.
- **Safe Mode support**—Allows multiple users to run PHP scripts on the same server simultaneously. The PHP safe mode helps to solve the shared-server security problem.
- **Open Source**—Allows the user to work with different software development languages. User can choose the desired software development language to create its own source code for different types of applications and distribute them on the World Wide Web (WWW) free of cost.
- **Third-Party application support**—Supports a wide range of different databases, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. For example, PHP 5.3 supports more than 15 database engines, and includes a common Application Program Interface (API) for database access.
- **PHP's extensible architecture**—Allows the user to read and write in various formats, such as Graphic Interchange Format(GIF), Joint Photographic Experts Group (JPEG), and Portable Network Graphics (PNG); send and receive e-mails using the Simple Mail Transfer Protocol SMTP, Internet Message Access Protocol (IMAP), and Post Office Protocol 3 (POP3) protocols. PHP also allows the user to access C libraries, Java classes, and Component Object Model (COM) objects as well as the program code written for these languages.

Introduction to HTML and XHTML

The PHP code needs to be embedded in either the HTML or Extensible Hypertext Markup Language (XHTML) language, as only these are supported by the Web browser. Apart from PHP, the JavaScript language can also be embedded in the HTML or XHTML languages to help perform certain tasks at the client side; thereby, reducing the burden on the Web server. Let's now discuss each of these languages.

HTML

HTML consists of a set of markup tags, which are used to display content on the Web pages. These markup tags (HTML tags) are enclosed between angular brackets and are usually written in pairs, such as <html>, </html>. The first tag is called the opening tag and the second tag is called the closing tag. A simple HTML document is used to create static Web pages; however, dynamic pages can also be created by embedding scripts, such as PHP and JavaScript in the HTML document. Listing 3.1 shows how to create an HTML page. You can find listing 3.1 in \Code\PHP\Chapter 03\Code for a Sample HTML page folder on the CD.

Listing 3.1: Showing the Code for a Sample HTML Page

```
<html>
<body>
<p>This is a HTML document</p>
</body>
</html>
```

In Listing 3.1, the <html>,<body> and are the opening tags; whereas, the ,</body> and </html> tags are the closing tags. The text written inside the and tags forms the content of the page and is displayed on the browser.

HTML documents are generally written in Notepad and saved with the .html or .htm extension. Nowadays, various HTML editors, such as Microsoft Office Front Page and Dreamweaver, are also available to create HTML pages.

XHTML

XHTML is a combination of HTML and Extensible Markup Language (XML); and is mainly created to enable the code to run on browsers with limited capabilities, such as mobile phones and other wireless devices.

The syntax of XHTML is almost similar to that of HTML; however, certain rules are to be followed to script a document in XHTML. Some of these rules are as follows:

- The <html>, <head>, and <body> tags are required in an XHTML document. The <html> tag must have an xmlns attribute with a value of http://www.w3.org/1999/xhtml
- All markup tags, once started must be closed. For example, an opening tag must have either an equal closing tag, such as <html> and </html>, or a self closing markup tag, such as <body />
- All tags must be written in lowercase
- All attribute values must be quoted with either single quotes or double quotes. For example, class=page is invalid but both class="page" and class='page' are correct
- All attributes must have values

The XHTML page shows errors if these rules are not followed; whereas, the same page would run in HTML. Listing 3.2 shows how to create an XHTML document. You can find listing 3.2 in \Code\PHP\Chapter 03\Code for a Sample XHTML Page folder on the CD

Listing 3.2: Showing the Code for a Sample XHTML Page

```
<! DOCTYPE html PUBLIC "-//W3C//DTD/XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml11-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>This is XHTML</title>
</head>
<body>
```

After learning to create the HTML and XHTML pages, let's discuss how to embed the PHP code in an XHTML page. The PHP code can be embedded in an XHTML page in any of the following three ways:

- The XML style
- The Short style
- The Script style

The XML style

The XML style is the most commonly used style of writing PHP scripts. Its syntax is as follows:

```
<?php
 //PHP code block
?>
```

The XML style method should be used in cases, where you need to embed the PHP code with the XML or XHTML documents.

The Short style

The Short style is the simplest style of writing PHP scripts. Its syntax is as follows:

```
<?
//PHP code block
?>
```

As the syntax of the Short style interferes with the XML document declarations, this style is not recommended for embedding the PHP code in XHTML.

The Script style

The Script style is mainly used while working with an HTML editor, which does not supports the other two styles.

```
<script language="php">
  //PHP Code Block
  </script>
```

This style is not much used these days.

Advantages of PHP over Other Scripting languages

PHP is one of the most popular server-side scripting languages used for creating dynamic Web pages. One of the reasons of its popularity is that it offers many advantages over most of the scripting languages that are currently in use. Now, let's discuss some of the advantages that the PHP offers over the following scripting languages:

- Active Server Pages (ASP)
- Cold Fusion
- Practical Extraction and Report Language (Perl)
- Java Server Pages (JSP)

Active Server Pages

Active Server pages (ASP) is a scripting language that is supported by only Microsoft Internet Information Server (IIS). This limits its availability to Win32 based servers; whereas, PHP scripting language is supported by almost all the Web servers. In addition, as compared to PHP, ASP is a much slower, less stable, and less secure language. PHP works with Apache, which has a proven record of speed, reliability, and hardened security.

Cold Fusion

PHP runs on almost every platform; whereas, Cold Fusion is only available on Win32, Solaris, Linux, and HP/UX. In addition, PHP is focused on programmers; whereas, Cold Fusion is designed for non-programmers. As compared to Cold Fusion, PHP is a faster, more efficient, and more stable language.

Practical Extraction and Report Language (Perl)

The main advantage of PHP over Perl is that PHP was designed for scripting for the Web; whereas, Perl was designed to do complex tasks, which makes it complicated. PHP is easier to integrate with the HTML language than Perl. As compared to Perl, PHP is relatively easy to learn and has a smaller learning curve. No prior knowledge of programming languages is required to learn PHP; whereas learning Perl requires some knowledge of C and shell scripting.

Java Server Pages (JSP)

PHP is supported on any platform that is equal or above 32 bits; whereas, JSP is supported only by those platforms that have a JVM. In addition, the performance of PHP is almost five times faster than JSP.

Creating a PHP Script

PHP scripts are plain-text files containing PHP instructions, JavaScript, and HTML language. Listing 3.3 shows how to create a PHP script. You can find listing 3.3 in \Code\PHP\Chapter 03\Code of a PHP Script folder on the CD.

Listing 3.3: Showing Code of a PHP Script

```
<html>
<body>
<?php
//this line of code displays a simple text
echo "Introduction to PHP ";
?>
</body>
</html>
```

Now, let's see how to run the script, shown in Listing 3.4, in a Web browser.

Running a PHP Script

Save the script, shown in Listing 3.4, to a location under your Web server root, and name it. In this case, we have named it as intro.php. Then, enter this URL in your Web browser, and click the ENTER key. The PHP script runs, as shown in Figure 3.1:

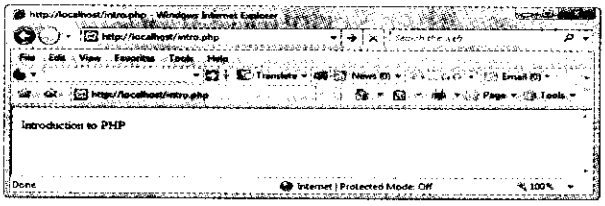


Figure 3.1: Showing the PHP Script Page

In Listing 3.4, first of all, you request the intro.php script. The Web server receives your request, recognizes that the file was a PHP script (by means of the .php file extension), and hands it over to the PHP parser and interpreter for further processing. This PHP interpreter then reads the instructions between the <?php . . . ?> tags, executes them, and passes the result to the back server, which in turn sends them back to your browser. The *echo* statement is used to display output to the user. The Print statement can also be used to display output instead of the echo statement.

The following points needs to be kept in mind while working with a PHP script:

- All PHP code must be enclosed within <?php . . . ?> tags
- Every PHP statement must end in a semicolon
- Blank lines within the PHP tags are ignored by the parser
- Single line comments must be preceded by the // characters, while multiline comments must be enclosed within a /* . . . */ comment block

Handling Errors in a PHP Script

Sometimes, the execution of the script halts in between due to occurrence of an error. Depending upon the severity of the error, either a warning message is displayed or the execution of the script stops at the point of error with a notification of error message. Now, let's generate an error in the script deliberately to have a better understanding of how to deal with an error.

Listing 3.5 shows that an error is generated by typing an extra semicolon after the echo keyword in the intro.php script. You can find listing 3.5 in \Code\PHP\Chapter 03\ Showing the Script with an Error folder on the CD.

Listing 3.5: Showing the Script with an Error

```
<html>
<body>
<?php
//this line of code displays a simple text
echo :"Introduction to PHP ";
?>
</body>
</html>
```

Now, run this script in the Web browser. The output is shown in Figure 3.2:

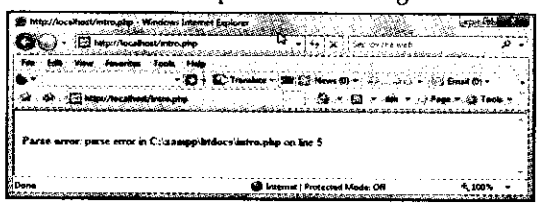


Figure 3.2: Showing the Error in Script

Figure 3.2 displays the error message generated by the PHP parser. The message tells you the nature and the location of the error.

Escape Characters

Escape characters are special functions that help to display certain characters, such as single or double quotes, the $ symbol in the script. The character to be displayed must be typed after the backslash (\) character, forming an escape sequence, such as \" and \$. These sequences signify that the character after the backslash should be displayed on the screen. You can also enter a new line or a tab space in the script by typing the letter n or t after the backslash character, respectively. Table 3.1 lists the escape characters in PHP:

Table 3.1: List of Escape Characters in PHP	
\"	Prints the next character as a double quote
\'	Prints the next character as a single quote
\n	Prints a new line character
\t	Prints a tab character
\r	Prints a carriage return
\$	Prints $ as the next character
\\	Prints \ as the next character

Listing 3.6 shows how to implement escape characters in a PHP script:

Listing 3.6: Showing Code to Implement Escape Characters

```
<?php
    $MyString = "This is an \"escaped\" string";
    $MySingleString = 'This \'will\' work';
    $MyNonVariable = "I have \$zilch in my pocket ";
    $MyNewline = "This ends with a line return\n";
    $MyFile = "c:\\windows\\system32\\myfile.txt";
    echo $MyString;
    echo $MySingleString;
    echo $MyNonVariable;
    echo $MyNewline;
    echo $MyFile;
?>
```

The output of Listing 1.6 is shown in Figure 3.3. You can find listing 3.6 in \Code\PHP\Chapter 03\ Showing Code to Implement Escape Characters on the CD.

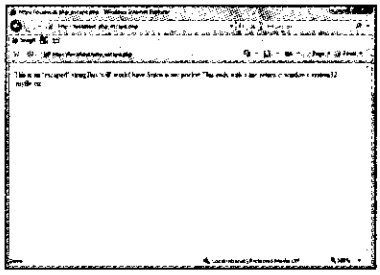


Figure 3.3: Displaying the Use of Escape Characters

Now, let's summarize what is discussed in this chapter.

Summary

In this chapter, you learned about the history, versions, and features of the PHP scripting language. PHP is one of the most widely used scripting languages in the world, as it can run on almost any Web server. The chapter

also discusses the advantages that PHP provides over other scripting languages, such as Cold Fusion, ASP, Perl, and JSP. You also learn to write and run a PHP script as well as how to track and handle errors in a script. The escape characters used while scripting are also discussed in detail.

In the next chapter, we will discuss about variables and constants in PHP.

Quick Revise

Q1. PHP stands for

a. **Personal Home Page** b. **PHP:Hypertext Preprocessor**

c. **Personal HTML Page** d. **None of these**

Ans: b

Q2. PHP was developed by

a. **Rasmus Lerdorf** b. **Robert Zend**

c. **The PHP Group** d. **None of these**

Ans: a

Q3. The parser for PHP was rewritten in the year 1997 by

a. **Zeev Suraski** b. **Andy Gutamen**

c. **Both** d. **None of these**

Ans: c

Q4. HTML stands for

a. **Hyper Text Markup Language** b. **High Text Markup Language**

c. **Hyper Transition Markup Language** d. **None of these**

Ans: a

Q5. JavaScript was developed by

a. **Microsoft Inc.** b. **SUN Microsystems**

c. **Texas Instrumentation** d. **Netscape**

Ans: d

Q6 \n is used to

a. **Print a new Tab character** b. **Print a carriage return**

c. **Print a new line character** d. **None of these**

Ans: c

Q7. Which of the following statements is true?

a. **The first official version of PHP was PHP/FI.**

b. **The HTML files are saved with .htm or .html extension.**

c. **JavaScript can be used for client side form validations.**

d. **PHP has a compiler.**

e. **XML stands for Extended Markup Language.**

f. **PHP adheres to GNU general public license norms.**

Ans: b, c

Q8. Which are the three styles that are recognized by the PHP parser?

Ans: The PHP code can be embedded in an XHTML page in any of the following three ways:

- **The XML style**—The XML style is the most commonly used style of writing PHP scripts. Its syntax is as follows:

```
<?php
 //PHP code block
?>
```

The XML style method should be used in cases, where you need to embed the PHP code with the XML or XHTML documents.

- **The Short style** – The Short style is the simplest style of writing PHP scripts. Its syntax is as follows:

```
<?
//PHP code block
?>
```

As the syntax of the Short style interferes with the XML document declarations, this style is not recommended for embedding the PHP code in XHTML.

- **The Script style** – The Script style is mainly used while working with an HTML editor, which does not supports the other two styles.

```
<script language="php">
//PHP Code Block
</script>
```

This style is not much used these days.

Q9. What are the advantages of PHP over other scripting languages?

Ans: PHP is one of the most popular server-side scripting languages used for creating dynamic Web pages. One of the reasons of its popularity is that it offers many advantages over most of the scripting languages that are currently in use. Now, let's discuss some of the advantages that the PHP offers over the following scripting languages:

- Active Server Pages (ASP)
- Cold Fusion
- Practical Extraction and Report Language (Perl)
- Java Server Pages (JSP)

Active Server pages (ASP) is a scripting language that is supported by only Microsoft Internet Information Server (IIS). This limits its availability to Win32 based servers; whereas, PHP scripting language is supported by almost all the Web servers. In addition, as compared to PHP, ASP is a much slower, less stable, and less secure language. PHP works with Apache, which has a proven record of speed, reliability, and hardened security.

PHP runs on almost every platform; whereas, Cold Fusion is only available on Win32, Solaris, Linux, and HP/UX. In addition, PHP is focused on programmers; whereas, Cold Fusion is designed for non-programmers. As compared to Cold Fusion, PHP is a faster, more efficient, and more stable language.

The main advantage of PHP over Perl is that PHP was designed for scripting for the Web; whereas, Perl was designed to do complex tasks, which makes it complicated. PHP is easier to integrate with the HTML language than Perl. As compared to Perl, PHP is relatively easy to learn and has a smaller learning curve. No prior knowledge of programming languages is required to learn PHP; whereas learning Perl requires some knowledge of C and shell scripting.

PHP is supported on any platform that is equal or above 32 bits; whereas, JSP is supported only by those platforms that have a Java Virtual Machine. In addition, the performance of PHP is almost five times faster than JSP.

Q10. Discuss the various versions of PHP

Ans: Depending upon the increasing and changing requirements, developers release new versions of a language with enhanced features and capabilities on a continuous basis. The following is the list of various versions of PHP released so far:

- **PHP Version 2** – Released on June 8, 1995 by Lerdorf. This version allowed to embed a PHP script in an (Hypertext Markup language) HTML file. It also provided the facility of form handling in Web pages.
- **PHP Version 3** – Released in the year 1997. The parser for PHP was rewritten by Zeev Surazki and Andy Gutaman, which formed the base for PHP 3. This was the time when PHP was given a new

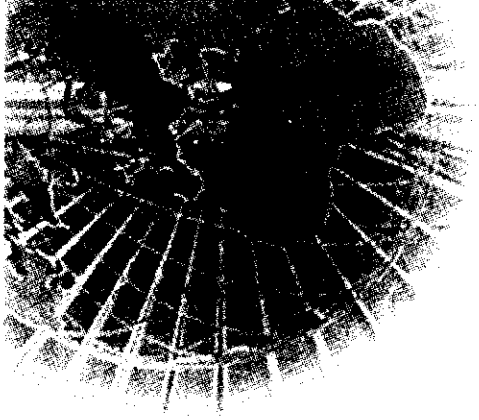
name, PHP: Hypertext Preprocessor. PHP/FI 2 was officially released in November 1997 and later on PHP version 3 was launched in the year 1998 after public testing.

- **PHP Version 4**—Released on May 22, 2000 with the Zend Engine 1.1. Various subversions of PHP version 4 are released till August 2008; however, further developments on this version have been stopped. The last released subversion of PHP 4 was PHP 4.4.9.
- **PHP Version 5**—Released on July 13, 2004 with an enhanced version of the Zend Engine, Zend Engine II. This version supported various advanced features, such as Object Oriented Programming, Data Object Extensions. The most recently released version of PHP 5 is PHP 5.2.10. Further developments under this version are still in progress.
- **PHP Version 6**—This version is still under development.

Q11. List the various features of PHP

Ans: The following are some of the important features of PHP:

- **Access control**—Provides a built-in Web-based configuration screen to handle access control configuration. Depending upon the client's domain, browser, e-mail address, or the referring document, various restrictions can be placed on the Web pages, such as password protected, completely restricted, and logging disabled.
- **File upload support**—Allows users to upload files to a Web server. PHP provides the Multipurpose Internet Mail Extensions (MIME) decoding process to upload the files onto a server.
- **HTTP-based authentication control**—Allows the user to create customized (Hypertext Transfer Protocol) HTTP-based authentication mechanisms for the Web server.
- **Variables, arrays, and associative arrays**—Support variables, arrays, and associative arrays that can be passed from one Web page to another using either the GET or POST method forms.
- **Conditional statements and loops**—Provide features similar to the C language and allows you to work with various conditional statements, such as if, elseif, else, as well as loops, such as for, foreach, while, and do while.
- **Extended regular expressions**—Support all common regular expression operations. Regular expressions are mainly used for pattern matching, pattern substitutions, and general string manipulation.
- **Raw HTTP header control**—Allows to transfer a Uniform Resource Locator (URL) from one client to another client. It also used to manipulate the latest updated header of the Web pages.
- **Access logging**—Allows you to record the number of times a Web page or a website is accessed. In addition, it also helps to generate footer on every page, displaying the access information.
- **Safe Mode support**—Allows multiple users to run PHP scripts on the same server simultaneously. The PHP safe mode helps to solve the shared-server security problem.
- **Open Source**—Allows the user to work with different software development languages. User can choose the desired software development language to create its own source code for different types of applications and distribute them on the World Wide Web (WWW) free of cost.
- **Third-Party application support**—Supports a wide range of different databases, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. For example, PHP 5.3 supports more than 15 database **engines**, and includes a common Application Program Interface (API) for database access.
- **PHP's extensible architecture**—Allows the user to read and write in various formats, such as Graphic Interchange Format(GIF), Joint Photographic Experts Group (JPEG), and Portable Network Graphics (PNG); send and receive e-mails using the Simple Mail Transfer Protocol SMTP, Internet Message Access Protocol (IMAP), and Post Office Protocol 3 (POP3) protocols. PHP also allows the user to access C libraries, Java classes, and Component Object Model (COM) objects as well as the program code written for these languages.



4

Working with Variables and Constants

In PHP: Hypertext Preprocessor (PHP), variables and constants are used to store data so that programs can process correctly and efficiently. A variable can be defined as a symbolic name associated with a value, which can be changed during the execution of a program. The value associated with a variable can be numeric or alpha-numeric. On the other hand, Constants are used to store values that can not be changed during execution.

In this chapter, you learn about variables and constants in which you learn how to declare variables, assign values to variables and destroy them. Next, you learn about various data types available in PHP, such as integer, float, and object. In the end, you learn about different types of operators available in PHP, such as assignment operators, arithmetic operators, and comparison operators.

Let's start by learning about variables.

Using Variables

As discussed, a variable can be considered as a labeled container that stores a value which can be changed during the execution of a program. As a general rule, variable names should be easy to understand. Let's discuss the naming rules and conventions for a variable.

Naming Rules and Conventions

Every programming language has a set of rules for naming variables. The naming rules for PHP variables are as follows:

- Every variable must be preceded by the $ sign. For example, `$var` is valid but `var` is invalid.
- A variable name must start with a letter or an underscore "_". For example, `$_var`, `$var` are valid variable names but `$1var` is invalid.
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _). For example, `$my_var` is valid but *$23#* is invalid.
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore or capitalization. For example, `$my_var` and `$myVar` are valid variable names but `$my var` is invalid.
- PHP has no limit on the length of variable name as in the case of other programming languages.

Let's now learn how to assign values to variables.

Assigning Values to Variables

In PHP, value is assigned to a variable using the equality (=) sign, which is also known as the assignment operator in PHP. This operator assigns the value on the right side of the equation to the variable on the left side. Listing 4.1 shows how to assign values to variables. You can find listing 4.1 in \Code\PHP\Chapter 04\Assigning Values to Variables on the CD.

Listing 4.1: Assigning Values to Variables

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head><title >PHP</title></head>
<body>
<?php
$name = 'PHP DEMO'; // assigning value to variable
echo $name; //calling the variable
?>
</body>
</html>
```

In Listing 4.1, the PHP DEMO value is assigned to the $name variable and the `echo` statement is used to print its value.

NOTE

In the previous example, you can also use the print statement instead of the echo statement to print the value of the $name variable. Though both the statements are used to print the values; however, they are different from each other. The echo statement can take multiple arguments separated by commas whereas the print statement cannot take multiple arguments. For example,

```
echo $x,$y; //valid statement
print $x,$y;//invalid statement
print $x;//valid statement
```

The print statement returns a value whereas the echo statement does not return a value. You can display the formatted output, such as you can print some letters in capital and some in small by using the print statement which is not possible with the echo statement.

Variables can be used for various purposes, such as you can assign a value of a variable to other variables or perform various calculations on them. Listing 4.2 shows how to perform various operations on variables. You can find listing 4.2 in \Code\PHP\Chapter 04\Performing Various Operations on a Variables folder on the CD.

Listing 4.2: Performing Various Operations On a Variable

```
<?php
$current_value = 10; // assigning value to variable
$value = $current_value; // assigning variable to another variable
$next_value = $value + 1; // perform calculation
echo "$value comes after $current_value"; // output: '11 comes after 10'
?>
```

In Listing 4.2, the `$current_value` variable is assigned value, 10. The `$current_value` variable is then assigned to the $value variable. The next statement increments the value of the $value variable by 1 and stores it in the `$next_value` variable. The value of the `$next_value` is then printed by using the `echo` statement. You can also assign values to variables by using the assign by reference method. In this method, the new variable points to another variable and changes done in any of the variables affect both the variables. In this method, the variable name must be preceded by an ampersand(&) sign before passing its reference to another variable. Listing 4.3 shows an example of assigning a value to a variable by assign by reference method.

Listing 4.3: Assigning a Value to a Variable Using the Assign by Reference Method

```
<?php
$name = 'Kogent India';          // Assign the value 'Kogent India' to $name
$alt_name = &$name;              // Reference $name via $alt_name.
$alt_name = "We are $alt_name"; // Alter $alt_name
echo $alt_name;
echo $name;                      // $name is altered too.
?>
```

In Listing 4.3, the `$name` variable is assigned a string value, Kogent India, and the `$alt_name` variable is assigned the reference of `$name`. In the next statement, the `$alt_name` variable is now assigned a string We are `$alt_name`. The value of the `$alt_name` and `$name` variables are then printed using the `echo` statement. You must note that when the value of `$alt_name` is changed, then the value of $name is also changed, as in this case both the variables print the same value. You can find listing 4.3 in \Code\PHP\Chapter 04\ Using the Reference Method folder on the CD

NOTE

It is important to note that you cannot assign expressions to variables by the assign by reference method.

```
<?php
$name = 'KogentIndia';
$alt_name = &$name; // Valid
$alt_name = &(1+2); // Invalid
?>
```

PHP allows you to set the name of a variable dynamically. In this case, a value assigned to a variable itself becomes a variable. An example of assigning a variable name dynamically is shown in Listing 4.4:

Listing 4.4: Assigning Variable Name Dynamically

```
<?php
$name = 'kogent'; // create a new variable
${$name} = 'PHP book'; // its name comes dynamically from the value of $name
echo $kogent;
?>
```

In Listing 4.4, the $name variable is assigned the value, kogent. In the next statement, the value assigned to `$name` itself becomes a variable (`$kogent`) that holds a value, PHP book. The value of the `$kogent` variable is then printed using the `echo` statement.

Let's now learn how to destroy variables.

Destroying Variables

PHP allows you to destroy variables or an element from an array when they are not required. You can destroy variables or elements from an array by using the `unset()` function. Listing 4.5 shows how to destroy a variable using the `unset()` function. You can find listing 4.5 in \Code\PHP\Chapter 04\ Using unset function folder on the CD.

Listing 4.5: Destroying a Variable Using the `unset()` Function

```
<?php
$name = 'we are Kogent India'; // assign value to variable
echo "Before destroying, $name";// print variable value
unset($name); // destroy variable
echo "After detroying, $name";// print variable value
?>
```

In Listing 4.5, the `$name` variable is assigned a string, we are Kogent India and then the `echo` statement prints 'Before destroying, we are Kogent India'. In the next statement, the `unset()` function is used to destroy the `$name` variable and now the `echo` statement shows an error message as the `$name` variable has been destroyed. However, it depends on the PHP error reporting configuration whether the error is printed or not.

After learning about variables, let's now learn about constants in PHP.

Using Constants

Constants are identifiers that store values, which cannot be changed during the execution of the script. You can store data, such as configuration settings, whose value does not change during the execution of the script. Constants are also used to represent integer values with special meanings in a particular context, such as error codes and flags. It is important to note that constants are case sensitive. By convention, the constant identifier name is always in uppercase. Constants follow same naming rules as variables; however, unlike variables the constant names do not start with the `$` sign. A valid constant name starts with a letter or underscore, followed by any number of letters, digits, or underscores.

The syntax to create a constant is as follows:

```
define ( "CONSTANT_NAME", constant_value );
```

In the preceding syntax, `CONSTANT_NAME` refers to the name of the constant and constant value refers to the value that the constant holds. For example,

```
define("HELLO", hello)
define("USER_NAME", kogent)
```

Listing 4.6 shows how to define constants in PHP scripts. You can find listing 4.6 in \Code\PHP\Chapter 04\Creating Constants in PHP Scripts folder on the CD.

Listing 4.6: Creating Constants in PHP Scripts

```
<?php
define("HELLO",Hello);
define("KOGENT",Kogent);
echo "HELLO";
echo "KOGENT";
?>
```

In Listing 4.6, two constants HELLO and KOGENT are defined that store values, Hello and Kogent. The values of these variables are printed using the `echo` statement.

NOTE

If a constant is redefined in the script, it might show an error message or prints the value that is assigned to it at the very first time. However, it depends on the error reporting configuration whether an error is printed or not.

In PHP, constants can only hold scalar values, such as numbers, strings, and boolean values. Unlike variables, they cannot hold arrays or objects.

PHP also includes a set of pre-defined built-in constants. Some of these built-in constants are M_PI, PHP_VERSION. While M_PI displays the value of mathematical Pi, `PHP_VERSION` displays the current version of PHP. List of other pre-defined constants is available on www.php.net

Let's now discuss about the data types available in PHP.

Exploring Data Types in PHP

Data type describes categories of values that a programming language can use. A data type refers to the type of data that a variable can hold. PHP includes eight data types, which are as follows:

- Integers
- Floating point numbers
- Strings
- Booleans
- Arrays
- Objects
- Resources
- Null

In PHP, it is not necessary to specify the data type of a variable before using it. PHP automatically decides the data type according to the value assigned to the variable. PHP data types can be broadly divided into two categories: *scalar* or *primitive* data types and *compound* data types. The scalar data type, such as integers, floating point numbers, strings, and Booleans can hold only a single value. The compound data type includes objects and arrays that can store a collection of values. All the data types of PHP are described as follows:

- **Integers**—Represents a whole number with no fractional components, such as -21,0,67. The permissible range of the integer data type varies. The range is decided by the operating system, generally a range of -2,147,483,648 to +2,147,483,647 is used. Integers can be written in decimal (1, 786, -32, +1063856), octal (01, 0123, +05, -0123456), or hexadecimal (0x1, 0xff, 0x1a3, +0x6, -0x1a234d) format.

NOTE

Hexadecimal numbers are preceded by a leading zero and X.

- **Floating point numbers**—Represents real numbers that include decimal place. PHP includes two types of floating point numbers. The first is a simple numeric literal with a decimal point. The second is a floating-point number written in scientific notation. Scientific notation is in the form of [*number*]E[*exponent*]. For example, 1.23, 0.003, -2.13, 0.214E2, -3.14E-3.
- **Strings**—Represents text literals of arbitrary length. In PHP, strings are enclosed within single quotes or double quotes. Strings inside double quotes are parsed, while strings inside single quotes are not. It means that if variables or special characters are enclosed in double-quotes with strings, then the values of variables are printed with the specified string. When variable names and special characters are enclosed in single-quotes, then the output is printed in the same way as you typed them. Listing 4.7 shows an example of using single and double quotes in strings. You can find listing 4.7 in \Code\PHP\Chapter 04\Using Single and Double Quotes in Strings folder on the CD.

Listing 4.7: Using Single and Double Quotes in strings

```
<?php
$name="Kogent India";//value assigned to $name
echo "We are $name";//prints  'We are Kogent India'
echo 'We are $name';//prints 'We are $name'
?>
```

In Listing 4.7, the `$name` variable is assigned a string, Kogent India. In the next statement, when the string, We are, and the $name variable is enclosed in double quotes then the `echo` statement prints We are Kogent India. Now when the same string is enclosed in single quotes, the `echo` statement prints We are `$name`. The last statement prints the variable name instead of its value in the string.

- **Booleans**—Represents a true or false value. All conditions return a true/false boolean value based on the condition being tested. Some of the statements always return a false value, which are as follows:
  - The keyword literal false.
  - The integer 0.
  - The floating-point number 0.0.
  - The empty string ("").
  - The string "0" (zero).
  - An object with no values or methods.
  - The null value.
- **Array**—Represents a variable that stores a collection of related data elements. Each individual element of an array can be accessed by referring to its index position. The position is either specified numerically or alphabetically. You learn more about arrays in chapter 4 Working with Functions and Arrays.
- **Object**—Allows you to store data as well as information to process that data. The data elements stored within an object are referred to as its properties or attributes of the object. To declare objects, first you must declare a *class* of object. Then you need to instantiate the object. Objects also allow you to create your own data types. You can define the data type in the object class, and then use the data type in instances of that class.
- **Resource**—Represents a special data type, which stores references to functions and resources external to PHP. The most common example of the resource data type is a database call.
- **NULL**—Represents a special data type that can have only one value, null. Null is not only a data type, but also a keyword literal. A variable of the null data type is a variable that has no value assigned to it. When no value is assigned to a variable, it is automatically assigned a value, null.

Let's now learn how to determine data type of a variable.

Determining Variable Data Type

PHP automatically determines the data type of variables from the value that the variable holds. If the value of a variable changes in the program, PHP automatically sets the appropriate new data type. Listing 4.8 shows an example of determining a variable data type using the `gettype()` function. You can find listing 4.8 in \Code\PHP\Chapter 04\Determining a Data Type using the gettype function folder on the CD.

Listing 4.8: Determining a Data Type Using the `gettype()` Function

```
<?php
$we_are = 'Kogent'; // define string variable
echo gettype($we_are); // output: 'string'
$we_are = 99.8; // assign new integer value to variable
echo gettype($we_are); // output: 'double'
unset($we_are); // destroy variable
echo gettype($we_are); // output: 'NULL'
?>
```

In Listing 4.8, the `gettype()` function is used to determine the datatype of a particular variable. The `$we_are` variable is assigned the value, Kogent. Therefore, PHP sets the datatype of the `$we_are` as string. Further, the

value of `$we_are` is changed to 99.8 which makes it a floating point variable. Next, the variable is destroyed using the `unset()` function which makes it a NULL type variable.

In addition to `gettype()`, PHP uses a number of specialized functions to check the data type of a variable, which are listed in Table 4.1:

Table 4.1: List of specialized functions

Functions	Purpose
is_bool()	Tests if a variable holds a Boolean value
is_numeric()	Tests if a variable holds a numeric value
is_int()	Tests if a variable holds an integer
is_float()	Tests if a variable holds a floating-point value
is_string()	Tests if a variable holds a string value
is_null()	Tests if a variable holds a NULL value
is_array()	Tests if a variable is an array
is_object()	Tests if a variable is an object

Let's now learn how to use the type casting method.

Using Type Casting

In PHP, you can convert the data type of a variable to another data type. For example, if a variable is of type integer and you want to store string value in it then you can convert the integer data type to the string data type. This form of converting data type of a variable to another data type is known as type casting. PHP does not require explicit type definition in variable declaration. An example automatic type conversion of PHP is the addition (+) operator. If either operand is a float, then both operands are evaluated as floats, and the result will be a float. Otherwise, the operands are interpreted as integers, and the result is also an integer. Note that this does **not** change the data types of the operands. Listing 4.9 shows an example of type casting. You can find listing 4.9 in \Code\PHP\Chapter 04\Showing an Example of Type Casting folder on the CD.

Listing 4.9: Showing an Example of Type Casting

```
<?php
$x = "3";  // $x is string (ASCII 48)
$x += 3;   // $x is now an integer (3)
$x = $x + 1.3;  // $x is now a float (3.3)?>
```

In Listing 4.9, the `$x` variable is assigned the value, 3, as string. In the next statement, the value of `$x` is incremented by 3 and the data type of the variable changes to integer implicitly. In the next statement, `$x` is incremented by 1.3 and the data type of `$x` is now changed to float. You can explicitly type cast a variable by enclosing the name of the desired type in parentheses before the variable which is to be cast. The syntax to use explicitly type cast is as follows:

```
variable=(target_type)variable
```

where, variable refers to a variable whose data type you want to convert target_type refers to the target data type to which you want to convert the data type of a variable

For Example:

```
$x=(boolean)$x; //converts data type of $x to boolean
$x=(float)$x; // converts data type of $x to float
$x=(binary)$x; // can also be used in place of binary; converts data type of $x to binary
```

Another way to type cast a variable is to use the `settype()` function. The syntax to use the function is as follows:

```
settype(variable_name,"target_type")
```

where, variable refers to a variable whose data type you want to convert

`target_type` refers to the target data type to which you want to convert the data type of a variable

For Example:

```
settype($x, "array"); //converts data type of $x to array
settype($x, "bool"); //converts data type of $x to Boolean
```

Let's now discuss about the operators available in PHP.

Exploring Operators in PHP

You can use operators in PHP to perform various operations, such as assign, multiplication, addition, subtraction, and concatenation, on variables and values.

Operators in PHP work with operands, which specify the variables and values that are to be used in a particular operation. PHP offers different types of operators, which are as follows:

- Assignment Operators
- Arithmetic Operators
- String Operators
- Comparison Operators
- Logical Operators
- Increment/Decrement Operators
- Arithmetic Assignment Operators
- Operator Precedence

Let's discuss all these in detail.

Assignment Operators

Assignment operator is used to assign values to variables or assign one variable to another as a value. Such an assignment of value is done with the equal (=) operator.

For Example:

```
$my_var = 9;
$another_var = $my_var;
```

Both the `$my_var` and `$another_var` variables contain the value, 9. Assignment operators can also be used in conjunction with arithmetic operators.

Let's now learn about arithmetic operators.

Arithmetic Operators

Arithmetic operators are used to perform basic mathematical operations, such as addition, subtraction, multiplication, division, and modulus. Table 4.2 lists the arithmetic operators used in PHP:

Table 4.2: List of Arithmetic Operators

Example	Name	Result
-$x	Negation	Displays opposite of $x
$x+$y	Addition	Displays addition of $x and $y
$x-$y	Subtraction	Displays difference of $x and $y
$x*$y	Multiplication	Displays product of $x and $y
$x/$y	Division	Displays quotient after dividing $x by $y
$x%$y	Modulus	Displays remainder after dividing $x by $y

Let's now learn about string operators.

String Operators

The String concatenation operator is used to combine values to create a string. The string concatenation operator is represented by a period (.) and can be used to build a string from other strings, variables containing non-strings (such as numbers) and even constants. Listing 4.10 shows an example of concatenating strings using the string concatenation operator. You can find listing 4.10 in \Code\PHP\Chapter 04\Using the String Concatenation Operator folder on the CD.

Listing 4.10: Using the String Concatenation Operator

```
<?php
echo 'We are'.'Kogent India';
?>
```

In Listing 4.10, the string concatenation operator is used to concatenate two strings. The second string is appended to the end of the first string, which gives the output, We are Kogent India. You can also use the string concatenation operator to concatenate the values of two variables. An example of concatenating the values of variables is shown in Listing 4.11. You can find listing 4.11 in \Code\PHP\Chapter 04\Using the Concatenation Operator folder on the CD.

Listing 4.11: Using the Concatenation Operator with Variables

```
<?php
$myColor='black';
echo 'My favorite color is'.$myColor;
?>
```

In Listing 4.11, the value of the `$myColor` variable is appended at the end of the string, My favorite color is, by using the concatenation operator. The `echo` statement now prints the string, My favorite color is black.

Let's now learn about comparison operators.

Comparison Operators

The comparison operators are used to compare one value with another and return either a true or false depending on the status of the match. For example, you can use a comparison operator to check if a variable value matches a particular number or whether one string is identical to another or not. PHP provides a wide selection of comparison operators.

The comparison operators are used with two operands, one to the left and one to the right of the operator. Table 4.3 lists the comparison operators used in PHP:

Table 4.3: List of Comparison Operators

Operator	Type	Description
==	Equal to	Returns true if first operand is equal to second
!=	Not equal to	Returns true if first operand is not equal to second
<>	Not equal to	Returns true if first operand is not equal to second
===	Identical to	Returns true if first operand is equal to second in both value and type
!==	Not identical to	Returns true if first operand is not equal to second in both value and type
<	Less than	Returns true if first operand is less than the second operand
>	Greater than	Returns true if first operand is greater than second
<=	Less than or equal to	Returns true if first operand is less than or equal to second
>=	Greater than or equal to	Returns true if the first operand is greater than or equal to the second operand

Let's now learn about logical operators.

Logical Operators

Logical Operators are also known as boolean operators because they evaluate parts of an expression and return either true or false. Table 4.4 lists the logical operators used in PHP:

Table 4.4: List of Logical Operators

Operator	Type	Description
and	AND	Performs a logical AND operation
or	OR	Performs a logical OR operation
xor	XOR	Performs a logical XOR(exclusive OR) operation
&&	AND	Performs a logical AND operation
||	OR	Performs a logical OR operation

Let's now learn about increment/decrement operators.

Increment/Decrement Operators

Other useful operators used in PHP are: Increment/Decrement Operators. Table 4.5 shows the List of Increment/decrement operators:

Table 4.5: List of Increment/Decrement Operators

Example	Name	Effect
$x++	Post Increment	Returns the value of *$a*, and then increments its value by one
++$x	Pre Increment	Increments the value of *$a* by one, then returns the value of *$a*.
$x–	Post Decrement	Returns the value of *$a*, and then decrements its value by one.
–$x	Pre Decrement	Decrements the value of *$a* by one, then returns the value of $a

You can use increment/decrement operators with variables. An example of using increment/decrement operators with variables is shown in Listing 4.12. You can find listing 4.12 in \Code\PHP\Chapter 04\Using Increment and Decrement folder on the CD.

Listing 4.12: Using Increment/Decrement Operators

```
<?php
$x = 19; // define variable
$y=$x++; // post increment
echo $y; // output: 19
$y=++$x; // pre increment
echo $y;// output:21
$y=$x--; // post decrement
echo $y; // output: 21
$y=--$x; // pre decrement
echo $y; //output:19
?>
```

In Listing 4.12, the `$x` variable is assigned the value, 19, and then `$x` is assigned to the `$y` variable. Therefore, now `$y` has value 19 and then the value of `$x` is incremented by 1. Now, the value of `$x` is 20. The `echo` statement prints the value of `$y` as 19. This is known as post increment. In pre increment, the value of `$x` is first incremented by 1 and then its incremented value, 21, is assigned to `$y`. Therefore, now both `$x` and `$y` contains value, 21. The `echo` statement now prints the value of `$y` as 21. Similarly, the post and pre decrement operations are performed in the next statements.

Let's now learn about arithmetic assignment operators.

Arithmetic Assignment Operators

The arithmetic assignment operators are a combination of arithmetic and assignment operators. They first perform the basic arithmetic operations on variables and then assign the resultant value to a variable itself. Table 4.6 shows the list of some arithmetic assignment operators:

Table 4.6: List of Arithmetic Assignment Operators

Operator	Description
+=	Adds the value and assigns it to a variable
-=	Subtracts the value and assigns it to a variable
*=	Multiplies the value and assigns it to a variable
/=	Divides the value and assigns it to a variable
%=	Divides and assigns the modulus to a variable
.=	Concatenates and assigns the value(for strings only) to a variable

You can use arithmetic assignment operators to perform various operations, as shown in Listing 4.13:

Listing 4.13: Using the Arithmetic Assignment Operators

```
<?php
$x=9;
$x+=2; //similar to $x=$x+2
echo $x; //displays  11
$x-=3; //similar to $x=$x-2
 echo $x;//displays 8
$x*=2; // similar to $x=$x*2
echo $x;// displays 16
$x/=2; //similar to $x=$x/2
echo $x;// displays 8
$x%=3;// similar to $x=$x%3
echo $x;// displays 2
$x= 'We are';
$x.='Kogent India';
echo $x; // displays 'We are Kogent India'
?>
```

In Listing 4.13, the `$x` variable is assigned a value, 9, and in the next statement, its value is incremented by 2 and assigned to itself. Therefore, now its value is 11, which is printed using the `echo` statement. Similarly, other assignment operators are used in next statements. You can find listing 4.13 in \Code\PHP\Chapter 04\Using the Arithmetic Operators folder on the CD. Let's now learn about operator precedence.

Operator Precedence

Operator precedence refers to the strength with which the two operators are bound together. For example, in the expression 1 + 5 * 3, the answer is 16 and not 18 because the multiplication (*) operator has a higher precedence than the addition (+) operator. Listing 4.14 shows an example of operator precedence.

Listing 4.14: Showing an Example of Operator Precedence

```
<?php
echo 5*2+1;// displays 11
echo 5+2*1;// displays 7
echo 5*(2+1);// displays 15
?>
```

In Listing 4.14, the first `echo` statement prints 11 while the next `echo` statement prints 7. This is because the precedence of the * operator is higher than the + operator; therefore, in the first statement the expression is evaluated as (5*2)+1 and in the second

case it is evaluated as 5+(2*1). You can find listing 4.14 in \Code\PHP\Chapter 04\Showing an Example of Operator Precedence folder on the CD

Parenthesis is used to force precedence which is seen in the third `echo` statement where (2+1) is evaluated first and its result is then multiplied with 5 and the output is 15. If the operators have same precedence then left to right associativity decides the order of precedence. Listing 4.15 shows an example of left to right associativity:

Listing 4.15: Showing the left to right associativity

```
<?php
echo 3*3/2;//evaluates (3*3)/2, displays 4.5
echo 3/3*2;//evaluates (3/3)*2 ,displays 2
?>
```

In Listing 4.15, both (*) and the(/) operator has the same precedence hence the expressions are evaluated on the basis of left to right precedence. Table 4.7 lists the precedence of operators with the highest-precedence operators listed at the top of the table and operators on the same line have equal precedence:

Table 4.7: Precedence of Operators

Operator Precedence		
Associativity	Operators	Additional Information
Right	!	Logical
Left	*/ %	Arithmetic
Left	+ - .	Arithmetic and string
Left	&	Bitwise and references
Left	|	Bitwise
Left	&&	Logical
Left	||	Logical
Right	=+=-=*=/=.=%=&=|=^=<< = >>=	Assignment
Left	and	Logical
Left	xor	Logical
Left	or	Logical
Left	,	Many uses

With this, we come to the end of the chapter. Let's now summarize the main topics learned in this chapter.

Summary

In this chapter, you have learned about variables that are used to store values. In PHP, every variable name starts with the $ sign and has certain rules and conventions that it follows. The unset() method is used to destroy a variable. You have also learned about various data types in PHP. In addition, you have learned how to determine a data type of a variable and convert one data type to another. In the end, you have learned about various operators, such as arithmetic operators, comparison operators, and logical operators, which are used to manipulate the variables to perform various arithmetic, logical and other important operations.

In the next chapter, we will discuss about controlling program flow in PHP.

Quick Revise

Q1. Which of the following is/are the valid variable name/names in PHP?

a. $96

b. $_my_name56

c. $a_@

d. $PHP_variable

e. $a b

Ans: b, d

Q2. Which of the following function/functions is/are used to destroy a variable in PHP?

a. destroy_var()

b. unset_var()

c. unset()

d. var_unset()

Ans: c

Q3. Which of the following is/are the correct method to declare a constant?

a. define("MESSAGE", HI)

b. define("my_message",hello)

c. define("123",123)

d. define('MY_NAME', PHP)

Ans: a, b

Q4. Which of the following is the correct way to define a variable in PHP?

a. int $x = 1

b. $int x= 1

c. $int_x=1

d. $x=1

Ans: d

Q5. Which of the following are the scalar data types?

a. NULL

b. Booleans

c. Arrays

d. Resources

Ans: b

Q6. Which of the following functions return a Boolean value?

a. is_boolean()

b. gettype()

c. settype()

d. is_not_int()

Ans: a

Q7. Which of the following is not an arithmetic operator?

a. ?

b. +

c. -

d. %

Ans: a

Q8. || operator is a

a. Arithmetic Assignment operator

b. Logical operator

c. Increment/Decrement operator

d. comparison operator

Ans: b

Q9. Which of the following operators returns true if first operand is equal to the second in both value and type?

a. <>

b. =

c. !=

d. ===

Ans: d

Q10. Which of the following operators have the same precedence as the - operator

a. +

b. !

c. *

d. %

Ans: a

Q11. Which of the following statements are true?

a. Integer data type in PHP has a range of -2,147,483,648 to +2,147,483,647 in all operating systems.

b. -0123956 is a valid octal integer in PHP.

c. 0x1a3 is a valid hexadecimal integer in PHP.

d. -4.19, 0.194E2 are valid floating point numbers in PHP.

e. NULL data type can take two values NULL and NOT_NULL

f. PHP automatically determines the data type of a variable.

g. ++$x is same as $x=$x+1

h. $x="0" returns true.

i. Value of a constant can be changed during the execution of a script.

Ans: c, d, f, g, h

Q12. Write the output of the following scripts

```
<?php
define("name",kogent_india);
echo "name";
?>
```

Ans: kogent_india

```
<?php
$x = 21%2;
$y=$x++;
echo $x;
$y=++$x;
echo $y;
$y=$x--;
echo $y;
$y=--$x;
echo $y;
?>
```

Ans: 2 3 3 1

```
<?php
$x=17;
$x+=2;
```

```
$x--;
$x*=5;
$x/=2;
$x%=3;
echo $x;
?>
```

Ans: 0

```
<?php
echo 5+8-2*6/4+5;
?>
```

Ans: 15

```
<?php
    $x=false;
 $x=(int)$x;
echo $x;
?>
```

Ans: 0

Q13. What is a variable? List the rules and conventions used to declare a variable in PHP.

Ans: A variable can be defined as a symbolic name associated with a value. The associated value can be changed during the execution of script. The value associated with a variable can be numeric or alpha-numeric.

Naming Rules and Conventions in PHP—The following are the rules and conventions to name a variable in PHP

- Every variable must be preceded by a "$" sign. `$var` is valid but `var` is invalid.
- A variable name must start with a letter or an underscore "_" . For example, `$_var`, `$var` are valid but `$1var` is invalid.
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _). For example `$my_var` is valid but *$23#* is invalid.
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore or with capitalization. For example, `$my_var` and `$myVar` is valid but `$my var` is invalid.
- PHP has no limit on the length of variable name as in the case of other programming languages.

Q14. How is type casting done in PHP?

Ans: PHP does not require explicit data type definition in variable declaration and the data type conversion is done implicitly. However, there are two methods for explicitly convert the data type of a variable. One way to type cast is to put the name of the desired type in parentheses before the variable which is to be cast.

```
variable=(target_type)variable
```

Some examples are:

```
$x=(boolean)$x;
$x=(float)$x;
$x=(int)$x;
```

Another way to type cast is by using the settype() function. The syntax to use the function as follows:

```
settype(variable_name,"target_type")
```

Some examples are:

```
settype($x, "array");
settype($x, "bool");
settype($x, "boolean");
settype($x, "float");
                                settype($x, "int");
```

Q15. Explain String operator with example

Ans: The PHP String operator is used to combine values to create a string. The String concatenation operator is represented by a period/full stop (.) and can be used to build a string from other strings, variables containing non-strings (such as numbers) and even constants. It is possible to concatenate a string and a number using the String operator. Examples of using String operator are as follows:

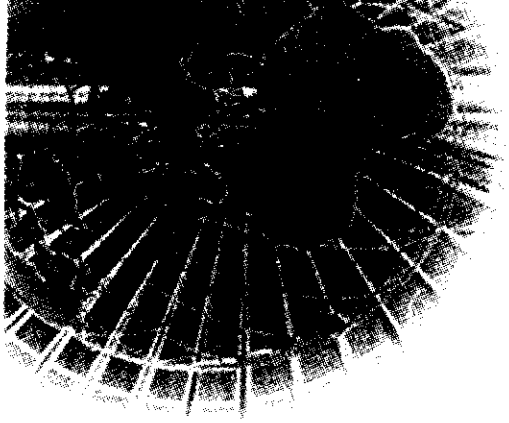
```
echo 'We are'.'Kogent India';
echo 13.'is unlucky';
echo 6+7.'is unlucky';
echo 'The unlucky number is'.(6+7);
```

Q16. How is variable name set dynamically in PHP?

Ans: PHP allows setting the name of the variable dynamically. In this case, value stored in a variable itself becomes a variable. The example is as follows:

```
<?php
$we_are = 'kogent';
${$we_are} = 'kogent india';
echo $kogent;
?>
```

In the preceding example, the string kogent is stored in the `$we_are` variable. Next, the string that is stored in the `$we_are` variable itself becomes a variable and a new string kogent india is stored in the dynamically created variable. In the next statement, the string stored in the dynamically created variable `$kogent` is printed using the `echo` statement.



5

Controlling Program Flow

In every programming or scripting language, by default, the program executes in a sequential order. However, you can change the flow of the program by using conditional and looping statements. These statements are used when you want to specify some conditions or parameters in the program. The program executes only if the specified conditions are fulfilled. You can use the conditional and looping statements in the following conditions:

- Specified condition or value is true
- Specified condition or value is not true
- Execute a program or block of code repeatedly till the condition is fulfilled
- Transfer control of the program from one block to another

Let's explore these statements in detail.

Conditional Statements

Conditional statements help the program to decide whether the condition given in the scrip is true and false. The conditions provided in the statements can be single or multiple And it allows to branch the path of execution in a script based on whether a single, or multiple conditions, evaluate to true or false.

Exploring Different Types of Conditional Statements

In PHP, there are four types of conditional statements, which are:

- The if Statement
- The if-else Statement
- The if-else if-else Statement
- The switch-case Statement

The if Statements

The If Statements always begin with if and followed by a condition provided in parentheses. If the condition is evaluated and found true, the statement or statements immediately following the condition are executed. On the other hand, if the condition is found false, no change takes place and you get a blank browser window when the script is run. The syntax for an if statement is as follows:

```
if (expression)
statement
```

Listing 5.1 shows an example to understand the script of the if statement. You can find listing 5.1 in \Code\PHP\Chapter 05\Script of the if Statement folder on the CD.

Listing 5.1: Script of the if Statement

```
<?php
$x=1;
if($x==1)
print'$xisequalto1';
?>
```

Output:

```
$x is equal to 1
```

In Listing 5.1, if the condition stated in the if block is true , then the message $x is equal to 1 gets displayed. However, if the condition is false, then the output gets the blank Web page. When you have more than one statement to be executed within a condition, it is necessary to provide them in brackets as shown in listing 5.2.

Listing 5.2 shows the if condition having more than one statement. You can find listing 5.2 in \Code\PHP\Chapter 05\Showing the if Condition with more than One Statement folder on the CD.

Listing 5.2: Showing the if Condition with More than One Statement

```
<?php
$x=1;
if ($x == 1)
```

```
{
  print '$x is equal to 1 <br>';
  $x++;
  print 'now $x is equal to 2';
}
?>
```

Output:

```
$x is equal to 1
Now $x is equal to 2
```

In Listing 5.2, if the condition stated in the if block is true, then two messages, $ x is equal to 1 and $ x is equal to 2, get displayed. First, the PHP parser executes the $ x is equal to 1 statement, then increments the value of $ x by 1, and then executes the Now $x is equal to 2 statement. However, if the condition is false, then only a blank Web page is shown as the output. You can write the conditional statements in different ways. Listing 5.3 shows how to write an if statement in multiple ways. You can find listing 5.3 in \Code\PHP\Chapter 05\Showing the if Statement in Different Ways folder on the CD.

Listing 5.3: Showing the if Statement in Different Ways

```
<?php
$x=1;
if ($x == 1) print '$x is equal to 1 <br>';
if ($x == 1) { print '$x is equal to 1<br>'; }
if ($x == 1) {
print '$x is equal to 1<br>'; }
?>
```

Output:

```
$x is equal to 1
$x is equal to 1
$x is equal to 1
```

In Listing 5.3, PHP parser checks whether or not the condition stated in the if block is true. In case the condition is true, the message gets displayed on the Web Page .

The if-else Statements

The if-else statements are used to make two types of decisions, such as true and false, based on a condition. When the condition is true, the if statement is executed and when the condition is false, the else statement is executed.

The syntax for the if-else statement is as follows:

```
if (condition)
  {
      statements_1
  }
  else
  {
      statements_2
  }
```

Listing 5.4 shows an example to understand the script of the if-else statement:

Listing 5.4: Showing the Script of the if-else Statement

```
<?php
$a=34;
$b=23;
if ($a > $b)
{
  echo "a is greater than b";
```

```
}
else
{
  echo "A is not greater than b";
}
?>
```

Output:

```
a is greater than b
```

In Listing 5.4 , the parser checks whether or not the condition($a>$b) stated in the if block is true. In case the condition is true, the block of statement executes displaying the message, **a is greater than b**. However, if the condition is false, then the control transfers to the else block. In this case, the block of code inside the else block is executed, displaying the message, **a is not greater than b**.

The if-elseif-else Statement

A combination of if-elseif-else statements is evaluated in a sequence. When the condition within the if statement is false then the elseif condition is checked and if it is found true, elseif statements are executed. However, else condition is executed when all the elseif conditions are false. The syntax for if-elseif-else statement is as follows:

```
if (condition_1) {
  statement_1
}
elseif (condition_2) {
  statement_2
}...
elseif (condition_n_1) {
  statement_n_1
}
else {statement_n
}
```

Listing 5.5 shows an example to understand the script of the if-elseif-else statement. You can find listing 5.5 in \Code\PHP\Chapter 05\Script of the if-elseif-else Statement folder on the CD.

Listing 5.5: Script of the if-elseif-else Statement

```
<?php
$result = 70;
if ($result >= 75)
{
    echo "Passed -> Grade A <br />";
}
elseif ($result >= 60)
{
    echo "Passed-> Grade B <br />";
}
elseif ($result >= 45)
{
    echo "Passed -> Grade C <br />";
}
else
{
 echo "Failed <br />";
}
?>
```

Output:

```
Passed -> Grade B
```

In Listing 5.5, the value 70 is assigned to the $ result variable. The parser checks the condition stated in the if and elseif blocks. If the condition is true, then the associated block of statement is executed, and the message Passed-> Grade B is displayed.

The switch Statements

A switch statement allows a program to evaluate a condition and match the condition's value to the statements given in case labels shown in the following syntax. If a match is found, the program executes the associated statement. The working of the switch statements is similar to the if statements.

The syntax for the switch statement is as follows:

```
switch (condition)
{   case label_1:
      statements_1
      break;
   case label_2:
      statements_2
          break;
   ...
   default:
      statements_n
          break;
}
```

First the program looks for a statement in the case labels that matches with the value of condition and execute the associated statements. If no matching label is found, the program looks for the optional default statement. However, in the absence of default statement, the program executes the statement given in the end of the switch statements.

Listing 5.6 shows an example to understand the script of the switch statement. You can find listing 5.6 in \Code\PHP\Chapter 05\Scripting of the switch Statement folder on the CD.

Listing 5.6: Script of the switch Statement

```
<?php
$flower = "rose";
switch ($flower)
{  case "rose" :
      echo $flower." costs $2.50";
      break;
   case "daisy" :
      echo $flower." costs $1.25";
      break;
   case "lily" :
      echo $flower." costs $1.50";
      break;
   default :
      echo "there is no such flower in our shop";
      break;
}
echo" no match is found";
?>
```

Output:

```
rose cost $ 2.50
```

In Listing 5.6, the $flower variable is assigned a value, rose, and then the switch condition (value of $flower) is matched with the case value given in the switch block. If the case value matches with the switch condition, then the control flow is transferred to the case-statement. The statement within the case block is executed and the message, Great! Ready to make calculations, is displayed as output.

In addition to match the conditions to the case labels, you can test for other the conditions, such as greater than and less than relationships. The conditions against which the expression is tested must be repeated in the case statement. Listing 5.7 shows an example to understand the script of the switch statement. You can find listing 5.7 in \Code\PHP\Chapter 05\Script of the switch statement folder on the CD.

Listing 5.7: Script of the switch Statement:

```
<?php
$myNumber = 5;
switch ($myNumber)
 {  case 0:
    echo "zero is not a valid value.";
    break;
  case $myNumber < 0:
    echo "Negative numbers are not allowed.";
    break;
  default:
    echo "Great! Ready to make calculations.";
    break;
}
?>
```

Output:

```
Great! Ready to make calculations.
```

In Listing 5.7, the $myNumber variable is assigned a value, 5. The switch condition (value of $myNumber) is matched with the case value given in the switch block. If the case value matches with the switch condition, then the control flow is transferred to the case statement. The case statement is executed and the message is displayed. However, if the switch condition is not matched with any case value, then the control flow is transferred to the default keyword. The statement within the default keyword is executed and the Great! Ready to make calculations message is displayed as the output. If an expression matches with the values specified in more than one case statement, only the first one should be executed. When a program makes a match, PHP stops looking for more matches.

Defining the Nested if-else Statements

One useful thing about conditional statements is that they can be nested within each other. The succeeding example shows the process of writing a discount program by using nested if-else statements.

The syntax for nested if-else is as follows:

```
if (condition)
{
statement;
  if (condition)
  {
       statement;
  }
else
{
statement;
}
else
{
statement;
}
```

Listing 5.8 shows an example to understand the script of the nested if else statement. You can find listing 5.8 in \Code\PHP\Chapter 05\Script of the Nested if-else Statement folder on the CD.

Listing 5.8: Script of the Nested if-else Statement

```
<?php
$age=30;
```

```
$price=3.00;
if($age>65)
{
$discount=.90;
echo "You have received our senior's  discount ,Your price is $".$price*$discount;
}
else
{
if($age<18)
{
$discount=.95;
echo "You have received our student's discount ,Your price is $".$price*$discount;
}
else
{
echo "Sorry you do not qualify for a discount , Your price is $".$price;
}
}
?>
```

Output:

```
Sorry you do not qualify for a discount, Your price is $ 3
```

In Listing 5.8, the $age and $ price variables are assigned the 30 and 3.00 values, respectively. The parser checks whether or not the if condition (value of $age) is true. If it is true, the associated block of statements is executed and the message is displayed. However, if the condition is not true, the control is transferred to the else block, executing the statements within the else block. This entire process is repeated for the next if-else block that is nested in the else block.

Combining Conditional Statements

You can also combine conditional statements by using logical operator, such as the && or || operators. You learned about logical operator in the earlier chapter. Listing 5.9 shows an example to understand the script of the combining conditional statement. You can find listing 5.9 in \Code\PHP\Chapter 05\Scripting of the Combining Conditional Statements folder on the CD.

Listing 5.9: Script of the Combining Conditional Statements

```
<?php
$year=2008;
// leap year are divisible by 400 or by 4 but not 100
if (($year % 400==0) || ($year % 100!=0) && ($year % 4==0))
{
echo  "$year is a leap year.";
}
else {
echo " $year is not a leap year.";
} ?>
```

Output:

```
2008 is a leap year.
```

In Listing 5.9, the value 2008 is assigned to the $year variable. The parser checks whether or not the if condition (value of $year) is true. The block of statements under the if statement gets executed in case the condition is true; otherwise, the control is passed to the else block. In this listing, the if condition is true and the message, 2008 is a leap year, is displayed as an output.

Looping Statements

In programming, it is often necessary to repeat the same block of statements a given number of times, or until certain conditions are fulfilled. This can be accomplished by using looping statements. Loops execute a block of statements a specified number of times provided it is true.

Exploring Different Types of Looping Statements

In PHP, there are four types of looping statements:

- The while loop
- The do-while loop
- The for loop
- The foreach loop

The while loop

The while loops are the simplest type of loops in PHP. It executes the statements repeatedly, as long as the *while* condition is evaluated and found **TRUE**. The value of this condition is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statements, execution does not stop until the end of the process. Sometimes, if the *while* condition is evaluated and found **FALSE** from the very beginning, the statements are not executed. The syntax for while loop is as follows:

```
while (condition)
{
 statement;
}
```

Listing 5.10 shows an example to understand the script of the while loop. You can find listing 5.10 in \Code\PHP\Chapter 05\Script of the while loop folder on the CD.

Listing 5.10: Script of the while Loop

```
<?php
$i=1;
while($i<=5)
{
echo "The number is ".$i."<br />";
$i++;
}
?>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

In Listing 5.10, the *while* condition evaluates whether $i is less than or equal to 5. The loop starts with i=1. The value of i gets increased by 1 each time the loop runs. When the value of i becomes greater than 5, the statement in the braces is skipped and the loop exits without performing any tasks. However, the value of i is not greater than 5, the statement in the braces is executed and the loop returns to the while statement. The process repeats until $i is greater than 5.

The do-while loop

The do-.while loops always executes the block of statements at least once and then checks the condition, and repeats the loop if the condition is true. The syntax for the do-while loop is as follows:

```
do
 {
 statement;
 }
while (condition);
```

Listing 5.11 shows an example to understand the script of the do-while loop. You can find listing 5.11 in \Code\PHP\Chapter 05\Script of the do-while loop folder on the CD.

Listing 5.11: Script of the do-while Loop

```
<?php
$i=1;
do
{
$i++;
echo "The number is".$i."<br />";}
while ($i<=5);
?>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

In Listing 5.11, the value 1 is assigned to the $i variable. The value of i is incremented by 1 each time the loop runs. If the value of i is already greater than 5, then the loop exits. If $i is not greater than 5, the statement in the braces executes and the loop returns to the do-while statement. The process repeats until $i is greater than 5, and the message is display as an output.

NOTE

The while loop evaluates a condition before executing the statement contained in the body of the loop. If the condition is evaluated and found false on the first check then the statement is not executed. The do-while loop, on the other hand, is provided for situations where you know that the statements contained in the body of the loop is always needs to be executed at least once when the condition is false.

The for loop

The for loop is used when you know how many times you want to execute a statement or a list of statements. For this reason, the for loop is also known as definite loop. The syntax of for loops is a bit more complex as compared to while loop and do while loop. The for loop syntax is as follows:

```
for (initialization; condition; increment/decrement)
{
statement;
}
```

The for statement takes three expressions inside its parentheses, separated by semi-colons, which are as follows: .

1. The initializing expression is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the for loop terminates.
3. The update expression increment/decrement executes.
4. The statements execute, and control returns to step 2.

Listing 5.12 shows an example to understand the script of the for loop. You can find listing 5.12 in \Code\PHP\Chapter 05\ Script of the for Loop folder on the CD.

Listing 5.12: Showing the Script of the for Loop

```
<?php
for ($i=1; $i<=5; $i++)
{
echo "The number is".$i."<br />";
}
?>
```

Output:

```
The number is 1
```

```
The number is 2
The number is 3
The number is 4
The number is 5
```

In Listing 3.12, the value 1 is assigned to the $i variable. The loop iterates as long as the value of the $i variable is less than or equal to 5. After each iteration, the value of the $i variable is incremented by 1. The statement inside the loop is executed until the condition is true.

The foreach loop

The foreach loop is a variation of the for loop and allows you to iterate over elements in an array. There are two different versions of the foreach loop, which are given in the syntax. The foreach loop syntaxes are as follows:

```
foreach (array as value)
  {
  statement;
  }

foreach (array as key=>value)
  {
  statement;
  }
```

Listing 5.13 shows an example to understand the script of the foreach loop. You can find listing 5.13 in \Code\PHP\Chapter 05\Showing the Script of the foreach Loop folder on the CD.

Listing 5.13: Showing the Script of the foreach Loop

```
<?php
$email= array('yogendra gupta','sulabh dixit');
foreach ($email as $value)
{
echo "Processing  ".$value."<br />";
}
?>
```

Output:

```
Processing  yogendra gupta
Processing  sulabh dixit
```

In Listing 5.13, the array elements are assigned to the $email variable. The foreach loop executes continuously until all the elements of the array are traced. The statement of the foreach block is executed after all the elements are traced, and the message is displayed as the output. Listing 5.14 shows an example of the foreach loop. You can find listing 5.14 in \Code\PHP\Chapter 05\ Script of the foreach Loop folder on the CD.

Listing 5.14: Script of the foreach Loop

```
<?php
$person = array('Name' => 'Sulabh', 'Age' => 23, 'Address' => 'Delhi');
foreach ($person as $key => $value)
{
echo $key." is ".$value."<br />";
}
?>
```

Output:

```
Name is Sulabh
Age     is 23
Address is Delhi
```

In Listing 5.14, the array element, (name=sulabh, age=23, address=delhi), is assigned to the $person variable. The foreach loop executes continuously until all the elements of the array are traced. The statement of the foreach block is executed after all the elements are traced, and the message is displayed as the output.

Defining the Nested Looping Statement

The placement of one loop inside the body of another loop is called nesting. When you nest two loops, the outer loop controls the number of iterations of the inner loop. You can nest all types of loops. Listing 5.15 shows an example to understand the script of the nested looping statement. You can find listing 5.15 in \Code\PHP\Chapter 05\Showing the Script of the Nested Looping Statement folder on the CD.

Listing 5.15: Showing the Script of the Nested Looping Statement

```
<?php
for($num2 = 0; $num2 <= 2;  $num2++)
{
   for($num1 = 0; $num1 <= 1; $num1++)
   {
      echo  $num2." And ".$num1."<br/>";
   }
}
?>
```

Inner Loop

Outer Loop

In Listing 5.15, two for loops are used, outer and inner. In the outer for loop, the $num2 variable is initialized as zero and a condition is set to evaluate whether the value of the $num2 variable is less than or equal to 2. If the specified condition is true, that is, if the value of the $num2 variable is less than or equal to 2, the inner for loop is executed. In the inner for loop, the $num1 variable is initialized as zero and a condition is set to check whether the value of $num1 is less than or equal to one. If the condition of inner for loop is true, then the value of $num1 is incremented by 1 and the echo statement of the inner for loop displays the result to the user.

When the program is executed for the first time, the initial values of the $num2 (0) and $num1 (0) variables are displayed to the user. The inner for loop is executed until the value of the $num1 variable does not exceed beyond 1. Once the value of $num1 exceeds 1, the control of the program is transferred to the outer for loop and checks the condition. The process of checking the condition of the outer for loop and transferring control to the inner for loop continues as long as the value of the $num2 variable remains less than or equal to 2.

Output:

```
0 and 0
0 and 1
1 and 0
1 and 1
2 and 0
2 and 1
```

Break, Continue, and Exit Statements

Sometimes the loops start without any condition, and allow the statements inside the brackets to decide when to exit the loop. There are three special statements that can be used inside a loop, which are:

- Break
- Continue
- Exit

Break statement

The break statement, when enter in a loop skips the remaining statements in the loop body and breaks the loop. Optionally, you can put a number after the Break keyword indicating how many times you want to break the loop. The syntax for break statement is as follows:

```
for (condition)
{
if (condition)
}
break;
statement;
statement;
```

```
statement;
}
```

Listing 5.16 shows an example to understand the script of the break statement. You can find listing 5.16 in \Code\PHP\Chapter 05\Showing the Script of the break Statement folder on the CD.

Listing 5.16: Showing the Script of the break Statement

```
<?php
for ($i=0; $i<=10; $i++)
{
  if ($i==3)
{
break;
}
echo "The number is ".$i;
echo "<br />";
}
echo " This is break statement";
?>
```

Output:

```
The number is 0
The number is 1
The number is 2
This is break statement
```

In Listing 5.16, the 0 value is assigned to the $i variable. The loop iterates as long as the value of the $i variable remains less than or equal to 5. After each iteration, the value of the $i variable is incremented by 1. In this listing, the for loop consists of the if condition, which contains a break statement. The execution of the loop terminates and the message is displayed as the output, if the condition is true.

Continue Statements

The continue statement also skips all remaining statements in the loop for the current iteration, but returns to the top of the loop and allows it to continue running. The syntax for continue statement is as follow:

```
for (condition)
{
if (condition)
}
continue;
statement;
statement;
}
```

Listing 5.17 shows an example to understand the script of the continue statement. You can find listing 5.17 in \Code\PHP\Chapter 05\Showing the Script of the continue Statement on the CD.

Listing 5.17: Showing the Script of the continue Statement

```
<?php
for ($i = 0; $i < 5; ++$i)
{
 if ($i == 2)
{
continue;
}
echo "The number is: " . $i;
echo "<br>";}
?>
```

Output:

```
The number is: 0
The number is: 1
The number is: 3
The number is: 4
```

In Listing 5.17, the 0 value is assigned to the $i variable. The loop iterates as long as the value of the $i variable remains less than or equal to 5. After each iteration, the value of the $i variable is incremented by 1. In this listing, the for loop consists of the if condition, which contains a continue statement. The current iteration of the loop is terminated, if the condition is true. The control returns to the top of the loop, restarting the cycle once again.

Exit statement

The exit statement is used when you want to stop a program from running. It can block infinite looping statements in the program. The syntax for exit statement is as follows:

```
for (condition)
{
if (condition)
}
exit;
statement;
statement;
  }
```

Listing 5.18 shows an example to understand the script of the exit statement. You can find listing 5.18 in \Code\PHP\Chapter 05\Showing the Script of the exit Statement folder on the CD.

Listing 5.18: Showing the Script of the exit Statement

```
<?php
for ($i = 0; $i < 5; ++$i)
{
if ($i == 2)
{
exit;
}
echo "The number is: ". $i;
echo "<br>";
}
echo " This is exit statement";
?>
```

Output:

```
The number is: 0
The number is: 1
```

In Listing 5.18, the 0 value is assigned to the $i variable. The loop iterates as long as the value of the $i variable remains less than or equal to 5. After each iteration, the value of the $i variable is incremented by 1. In this listing, the for loop consists of the if condition, which contains an exit statement. The execution of the script terminates and the message is displayed as the output, if the condition is true.

Summary

In this chapter, you have learned about the conditional and looping statements that are used to control the program flow. You also learned if-else statements and the process of combining conditional statements. In addition, you learned about break, continue, and exit statements.

In the next chapter, you learn about function and array.

Quick Revise

Q1. True/False:

a. if-elseif-else statement is evaluated in sequence. **True**

b. Combine conditional statements are used assignment operator. **False**

c. Inner loop controls the number of iterations of outer loop. **False**

d. The If Statements always begin with if and followed a condition provided in parentheses. **True**

e. A switch statement allows a program to evaluate a condition and match the condition's value to the statements given in case labels. **True**

f. The execution of the script terminates by using the break statement. **False**

g. The break statement, when enter in a loop skips the remaining statements in the loop body and breaks the loop. **True**

h. While loop evaluates a condition after executing the statement. **False**

Q2. The if statement is an example of a

a. Sequence Structure

b. Decision Structure

c. Pathway Structure

d. Class Structure

Ans: b

Q3. Which type of structure has a value of either true or false?

a. Binary Expression

b. Unconditional Expression

c. Decision Expression

d. Boolean Expression

Ans: c

Q4. To create a block of statements, you enclose the statements in

a. Parentheses ( )

b. Angled Bracket < >

c. Square Bracket [ ]

d. Braces { }

Ans: d

Q5. Which of these is an if statement that appears inside another if statement?

a. Structured if statement

b. Nested if statement

c. Tiered statement

d. None of these

Ans: b

Q6. Which section of a switch statement is branched to if none of the case expressions match the switch expression?

a. Default

b. Else

c. Case

d. None of these

Ans: a

Q7. What is the each repetition of a loop is known as?

a. **Cycle**

b. **Revolution**

c. **Iteration**

d. **Orbit**

Ans: c

Q8. This type of loop always executes at least one time.

a. **While**

b. **Do-while**

c. **For**

d. **Foreach**

Ans: b

Q9. In the expression i++, the ++ operator is in what mode?

a. **Postfix**

b. **Prefix**

c. **Pretest**

d. **Posttest**

Ans: a

Q10. This type of loop has no way of ending and repeats until the program is interrupted.

a. **Indeterminate**

b. **Interminable**

c. **Infinite**

d. **Timeless**

Ans: c

Q11 What will the echo statement in the following program segment display?

```
$x=5;
echo ++$x;
```

a. **5**

b. **0**

c. **6**

d. **None of these**

Ans: c

Q12. Explain what the meant by the term conditionally executed?

Ans: The program statements are executes in a sequential order. Conditional statements are used when you want to specify some conditions in the program. The program executes only if the specified conditions are fulfilled.

Q13. Explain how many types of conditional statements are used in PHP?

Ans: There are four types of conditional statements, which are:

- The if Statement
- The if-else Statement
- The if-elseif-else Statement
- The switch-case Statement

Q14. What is nested if-else statement? Write the code of the nested if-else statement.

Ans: One useful thing about conditional statements is that they can be nested within each other. The succeeding example shows the process of writing a discount program by using nested if-else statements.

The syntax for nested if-else is as follows:

```
if (condition)
{
  statement
  if (condition)
  {
        statement
  }
  else
  {
        statement
  }
  else
  {
  statement
}
```

Script of the nested if-else statement is:

```
<?php
$x=1;
if($x==0)
{
echo $x ." i equal to 0";
}
else
{
if($x>0)
{
echo $x." is greater than 0";
}
else
{
echo $x." is less than 0";
}
}
?>
```

Q15. Describe the difference between while loop and do-while loop?

Ans: The while loop evaluates a condition before executing the statement contained in the body of the loop. If the condition is evaluated and found false on the first check then the statement is not executed. The do-while loop, on the other hand, is provided for situations where you know that the statements contained in the body of the loop is always needs to be executed at least once when the condition is false.

Q16: Write a for loop that display the following set of number:

0, 10, 20, 30, 40, 50 500

Ans: The script of the following set of number is:

```
<?php
      for($i=0; $i<=500; $i=$i+10)
            {
                  echo $i."  ";
            }
?>
```

The output of the following script is:

```
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230
240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400 410 420 430 440 450
460 470 480 490 500
```

Q17. Write a nested loop that displays 5 rows of @ character. There should be 10 @ characters in each row.

Ans: The script is:

```
<?php
for($i=0; $i<5; $i++)
      {
            for($y=0; $y<10; $y++)
                  {
                        echo "@ ";
                  }
            echo "<br>";
      }
?>
```

The output of the following script is:

```
@ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @ @
```

Q18. Find the output of the following codes:

a.

```
<?php
$employeeAge;
$employeeAge ["Ram"] = "28";
$employeeAge ["Tanu"] = "23";
$employeeAge ["Sandeep"] = "35";
$employeeAge ["Gopal"] = "46";
$employeeAge ["Neeraj"] = "34";
foreach($employeeAge as $key => $value)
{
        echo "Name: $key, Age: $value <br />";
}
?>
```

b.

```
<?php
for($x=1, $y=1;$x<=5; $x++,$y++)
{
echo $x."multiply".$y."equals".($x*$y)."<br>";
}
?>
```

c.

```
<?php
$x=1;
switch($x)
{
case 1;
echo"The number is 1 <br>";
case 2:
echo "The number is 2 <br>";
case 4:
echo "The number is 4 <br>";
default;
echo "Default number <br>";
```

```
}
?>
```

d.

```
?php
$marks=74;
if($marks > 90)
echo"Excelent <br>";
elseif($marks > 75)
echo "Very Good<br>";
elseif($marks > 60)
echo "Good <br>";
elseif($marks > 45)
echo "Fine <br>";
else
echo "Fail";
?>
```

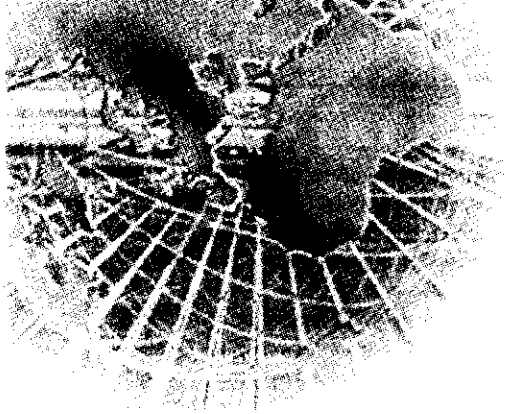
Ans: The output of the following scripts is:

a. Name: Ram, Age: 28
   Name: Tanu, Age: 23
   Name: Sandeep, Age: 35
   Name: Gopal, Age: 46
   Name: Neeraj, Age: 34

b. 1 multiply 1 equals 1
   2 multiply 2 equals 4
   3 multiply 3 equals 9
   4 multiply 4 equals 16
   5 multiply 5 equals 25

c. The number is 1
   The number is 2
   The number is 4
   Default number

d. Good



6

Working with Functions, Arrays, Files, and Directories

Introduction

Functions and arrays are two useful programming concepts in any programming as well as scripting language. In simple terms, an array is a special variable with a unique name, which can hold more than one value, at a time. On the other hand a function is a block of code with a unique name, which can be reused again and again. If required a function can be called by another function. A function comprises of statements, such as looping statements, conditional statements, and statements that define variables and arrays.

In this chapter you learn to use functions and arrays in PHP. The first section of this chapter introduces the process of creating and invoking user-defined functions within a PHP script. This section also discusses about some built-in functions and their usage in PHP. The second part of this chapter introduces arrays, the process of creating arrays, and their implementation within the PHP script. Finally, some useful built-in functions are discussed for array manipulation.

PHP: Hypertext Preprocessor (PHP) comes with a powerful and flexible file manipulation Application Programming Interface (API) that allows a developer to perform various functions with the help of files and directories. A computer file is a collection of related data or program records that are stored as a unit with a single name. A file type is identified by a dot followed by a series of letters, with the name of the file appearing before the dot, such as name.doc or pic.jpg. The letters after the dot define the type of the file, such as .doc refers to a text file; whereas, the .jpg refers to a picture file. The storage place where the computer files are stored in a logical order is known as a directory. A directory can also contain other directories, which are called subdirectories of the main directory. A system may contain many directories with innumerable files and subdirectories in them.

User-Defined Functions in PHP

In PHP, there are more than **700 built-in functions**. But PHP also allows us to write our own functions. **Reusability** is the primary advantage of user-defined functions. Functions make it easy to use the same piece of code several times in your application without the need to write it again and again. The syntax to create user-defined functions is as follows:

```
function <function-name> (argument 1, argument 2…, argument n) {
  //code to execute
  return;
}
```

A function is declared using function keyword, followed by the name of the function and the parenthesis that may contain variable names as arguments. The actual function code is enclosed in curly braces. The last line of the code block should be a return statement, which returns control back to the called function.

Naming Conventions

Following are some basic rules, which should be followed while providing names to the functions:

- Function names are not case sensitive. Therefore, `fname()`, `Fname()`, and `FNAME()` all refer to the same function.
- Function names can contain only letters from the ASCII character set, digits, and underscores.
- Function names cannot begin with a digit.
- Two functions cannot have the same name as PHP does not support function overloading.
- Reserved keywords cannot be used as function names.

Creating and invoking a Function

You can create a function using **function** keyword and invoke or call it by its name. The name needs to be followed by a set of parentheses that contain zero or more values to be passed to the function. Listing 6.1 shows a user-defined function to print the book name. You can find listing 6.1 in \Code\PHP\Chapter 06\Creating and Invoking a Function folder on the CD.

Listing 6.1: Creating and Invoking a Function

```
php
 //function definition
function book_Name($name) {
```

```
echo   "I have a book named". $name;
}
//function invocation
book_Name("programming in PHP");
?>
```

Output:

```
programming in PHP
```

In the preceding listing, the function is named `book_Name()`, that has a single argument `$name` and its body contains a single statement, which uses **echo** to print the name of the book.

Returning Value from a Function

You can return a value to the caller function, using the `return $value` statement. Execution control is transferred to the caller immediately after the return statement. If there are other statements in the function after the return statement, they will not be executed. Listing 6.2 shows how to return a value from a function. You can find listing 6.2 in \Code\PHP\Chapter 06\Returning a value from a Function folder on the CD.

Listing 6.2: Returning value from a Function

```
<?php
//function definition
function get_Year()
{
$year=date("Y");
return  $year;  //value returns to the calling function in our case echo function
}
echo " This year is: ". get_Year();
?>
```

Output:

```
This year is 2009
```

In the preceding listing, we declare a function `get_year()` that returns the output This year is 2009 to the caller function.

Understanding the Variable Scope

A key concept related to user-defined function in PHP is variable scope; the extent of a variable's visibility within the space of a PHP program. By default, variables used within a function are local. A local variable is restricted to the function space alone, and it can't be viewed or manipulated outside the function in which it is declared. Listing 6.3 shows the scope of a local variable. You can find listing 6.3 in \Code\PHP\Chapter 06\Defining the Variable Scope folder on the CD.

Listing 6.3: Defining the Variable Scope

```
<?php
//function definition
//change the value of $score
function change_Score() {
$score=20;              //local variable
}
//Define a variable in the main program
$score=40;
echo "Score is: ".$score;                    //Output: Score is: 40
//run the change_Score() function
change_Score();
//print $score again
echo "Score is: ".$score                     //output: Score is: 40
?>
```

Here, the variable, `$score` is defined in the main program, and the `change_Score()` function contains code to change the value of this variable. However, after running this function, the value of $score remains at its original setting, because the changes made to $score within the `change_Score()` function remain "local" to the function and do not reflect in the program. The global keyword helps use the same variable everywhere in your code, even inside the functions. Listing 6.4 shows the use of a global variable. You can find listing 6.4 in \Code\PHP\Chapter 06\Using the Global Variable folder on the CD.

Listing 6.4: Using the Global Variable

```
<?php
//function definition
//change the value of $score
function change_Score()
{
global $score=20;                                    //global variable
}
//Define a variable in the main program
$score=40;
echo "Score is: ".$score;                            //Output: Score is: 40
//run the change_Score() function
change_Score();
//print $score again
echo "Score is: ".$score                             //output: Score is: 20
?>
```

Here, the `global` keyword changes the scope of the variable, `$score` increasing its scope to encompass the entire program. As a result, changes made to the variable within the `change_score()` function reflect in the main program.

Passing Arguments by Reference

An alternative to returning a result or using a `global` variable is to pass a reference to a variable as an argument to the function. This means that any change made to the variable within the function affects the original variable. Listing 6.5 shows the use of a reference variable. You can find listing 6.5 in \Code\PHP\Chapter 06\Passing Arguments by Reference folder on the CD.

Listing 6.5: Passing Arguments by Reference

```
<?php
//function definition
//change the value of $score
function change_Score(&$score)                       //$score as a reference argument {
 $score=$score +20;
}
//Define a variable in the main program
$score=40;
echo "Score is: ".$score;                            //Output: Score is: 40
//run the change_Score() function
change_Score($score);
//print $score again
echo "Score is: ".$score                             //output: Score is: 60
```

The difference between the Listing 6.4 and 6.5 is that the parameter `$score` of the `change_Score()` function is prefixed with an ampersand (&) character. The ampersand means that a reference of the original variable is passed as the parameter, not the value of the variable. The result is that the changes made to **$score** in the `change_Score()` function affect the original variable `$score` outside the function.

Built-in Functions in PHP

PHP offers several built-in functions that can be used in your code. These functions are well documented and very helpful in achieving programming goals.

String Manipulation Functions

Strings are one of the most important parts of the PHP language. Most textual data are represented as strings, and many scripts are dedicated to processing, cleaning, escaping, parsing, and transforming text. Let's discuss some built-in string manipulation functions.

Getting the Length of a String

- **strlen()**—Returns the length of the string. Following syntax is used to define the `strlen()` function:
  Syntax: `int strlen (string $string)`
- **Parameters**—string - Required. The string being measured for length.
- **Return Value**—The length of the string on success, and 0 if the string is empty.

Listing 6.6 shows how to determine the length of a string using the `strlen()` function. You can find listing 6.7 in \Code\PHP\Chapter 06\Splitting the String into an Array folder on the CD.

Listing 6.6: Getting the Length of a String

```
<?php
$string="kogent";
echo "The length of string is:".strlen($string);  //Output: The length of string is:6
```

In the preceding listing, the `strlen()` function returns the length of the string, kogent.

Splitting the String into an Array

- **explode()**—Breaks a string in to an array. Following syntax is used to define the `explode()` function:

  ```
  Syntax:     array explode(separator, string $string , limit)
  ```

- **Parameters:**
  - Separator - Required. Specifies where to break the string.
  - string - Required. Specifies the string to be split.
  - limit - Optional. Specifies the maximum number to return.
- **Return Value**—If separator is an empty string ("") `explode()` returns `false`. If separator is not contained in the string then an empty array is returned.

Listing 6.7 shows how to break a string into an array using `explode()` function:

Listing 6.7: Splitting the String into an Array

```
<?php
$string="Kogent Learning Solutions";
$element=explode(" ", $string);
echo $element[0]."<br/>";
echo $element[1]."<br/>";
echo  $element[2];
?>
```

Output:

```
Kogent
Learning
Solutions
```

In the preceding listing, the `explode()` function breaks the Kogent Learning Solution into an array.

Joining Array Elements into a Single String

- **implode ()**: Joins array elements with a string. Following syntax is used to define the `implode()` function:

  Syntax:

  ```
  string implode (separator , array)
  ```

- **Parameters:**
  - separator - Optional. Specifies what to put between the array elements, default is an empty string.
  - array - Required. The array to join to a string.
- **Return Value**—Returns a string containing a string representation of all the array elements in the same order, with separator string between each element.

Listing 6.8 shows with a string how to join an array elements using `implode()` function. You can find listing 6.8 in \Code\PHP\Chapter 06\Joining Array Elements into a Single String folder on the CD.

Listing 6.8: Joining Array Elements into a Single String

```
<?php
$string=array('kogent' , 'is' , 'a' , 'learning' , 'solution');
echo implode( " " , $string);
?>
Output: kogent is a learning solution
```

In the preceding listing, the `implode()` function joins the array elements into a string, kogent learning solution.

Finding the Position of a String in another String

- **strpos ()**—Finds the position of first occurrence of a string in another string (case-sensitive). Following syntax is used to define the `strops()` function:

Syntax:

```
int strpos (string, find, start)
```

- **Parameters:**
  - string- Required. Specifies the string to search.
  - find - Required. Specifies the string to find.
  - start - Optional. Specifies where to begin the search.
- **Return Value**—Returns the position as an integer. If it is not found, `strpos()` will return boolean `false`.

Listing 6.9 shows a string how to join array elements using `implode()` function. You can find listing 6.9 in \Code\PHP\Chapter 06\Joining Array Elements into a Single String folder on the CD.

Listing 6.9: Finding the Position of a String in another String

```
<?php
$string="kogent solutions";
echo "The position of so in kogent solution is:".strpos($string , "so");      //output:7
```

In the preceding listing, the `strpos()` function finds the position of the string, so in kogent solution.

Repeating the Same String Many Times

- **str_repeat ()**—Repeats a string. Following syntax is used to define the `str_repeat()` function:

Syntax:

```
string str_repeat (string $string , repeat)
```

- **Parameters:**
  - string -Required. Specifies the string to be repeated.
  - repeat -Required. Specifies the number of times the string is repeated. The number must be greater or equal to 0.
- **Return Value**—Returns the repeated string.

Listing 6.10 shows how to repeat a string using the `str_repeat()` function. You can find listing 6.10 in \Code\PHP\Chapter 06\Repeating the Same String Many Times folder on the CD.

Listing 6.10: Repeating the Same String Many Times

```
<?php
$string="kogent-";
echo  str_repeat($string , 5);
?>
```

Output:

```
kogent-kogent-kogent-kogent-kogent-
```

In the preceding listing, the `str_repeat()` function repeats the string, kogent- 5 times.

Reversing a String

- **strrev ()**—Reverses a string. Following syntax is used to define the `strrev()` function:

Syntax:

```
int strrev (string $string)
```

- **Parameters**—The string to be reversed.
- **Return Value**—Returns the reversed string.

Listing 6.11 shows how to reverse a string using the `strrev()` function. You can find listing 6.12 in \Code\PHP\Chapter 06\Reversing a String folder on the CD.

Listing 6.11: Reversing a String

```
<?php
$string="kogent";
echo "The reversed string is:".strrev($string);
```

Output:

```
The reversed string is: tnegok
```

In the preceding listing, the `strrev()` function reverses the string, kogent as tnegok.

Table 6.1 lists some commonly used string manipulation functions:

Table 6.1: String Manipulation Functions

Function	Description	Version
bin2hex()	Converts a string of ASCII characters to hexadecimal values	3
chr()	Returns a character from a specified ASCII value	3
chunk_split()	Splits a string into a series of smaller parts	3
convert_cyr_string()	Converts a string from one Cyrillic character-set to another	3
count_chars()	Returns how many times an ASCII character occurs within a string and returns the information	4
echo()	Outputs strings	3
fprintf()	Writes a formatted string to a specified output stream	5
html_entity_decode()	Converts HTML entities to characters	4
htmlentities()	Converts characters to HTML entities	3
ltrim()	Deletes whitespace from the left side of a string	3
money_format()	Returns a string formatted as a currency string	4
print()	Outputs a string	3
printf()	Outputs a formatted string	3
rtrim()	Deletes whitespace from the right side of a string	3
sscanf()	Parses input from a string according to a format	4
str_ireplace()	Replaces some characters in a string (case-insensitive)	5
str_replace()	Replaces some characters in a string (case-sensitive)	3
str_split()	Splits a string into an array	5
str_word_count()	Counts the number of words in a string	4
strcasecmp()	Compares two strings (case-insensitive)	3
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())	3
strcmp()	Compares two strings (case-sensitive)	3
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)	5
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)	3
strrchr()	Finds the last occurrence of a string inside another string	3
strripos()	Finds the position of the last occurrence of a string inside another string (case-insensitive)	5
strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)	3
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)	3
strtolower()	Converts a string to lowercase letters	3
strtoupper()	Converts a string to uppercase letters	3
substr_count()	Counts the number of times a substring occurs in a string	4
substr_replace()	Replaces a part of a string with another string	4
trim()	Deletes whitespace from both sides of a string	3
ucfirst()	Converts the first character of a string to uppercase	3
ucwords()	Converts the first character of each word in a string to uppercase	3
vfprintf()	Writes a formatted string to a specified output stream	5
vprintf()	Outputs a formatted string	4
wordwrap()	Wraps a string to a given number of characters	4

Date and Time Functions in PHP

The date and time built-in function in PHP provides the ability to read the system time in various formats and helps to manipulate the date information. Let's learn to use some of the most common date and time functions.

Finding the Current Date and Time

- **date ()** – Formats the local time and date. Following syntax is used to define the `date()` function:

  Syntax:

  ```
  date (format , timestamp)
  ```

- **Parameters:**
  - format- Required. Specifies how to return the result:
    - d - Specifies the day of the month (from 01 to 31)
    - D - Specifies a textual representation of a day (three letters)
    - F - specifies a full textual representation of a month (January through December)
    - t - Specifies the number of days in the given month
    - g - Specifies 12-hour format of an hour (1 to 12)
    - L - Specifies whether it's a leap year (1 if it is a leap year, 0 otherwise)
    - l - Specifies the full textual representation of a day
    - h - Specifies 12-hour format of an hour (01 to 12)
    - i - Specifies minutes with leading zeros (00 to 59)
    - s- Specifies seconds, with leading zeros (00 to 59)
    - Y - specifies a four digit representation of a year
    - a - Specifies a lowercase am or pm
    - S - Specifies the English ordinal suffix for the day of the month (st, nd, rd or th)
- **Return Value** – Returns the formatted date and time

Listing 6.12 shows how to display current date and time in a specified format using the `date()` function. You can find listing 6.12 in \Code\PHP\Chapter 06\Finding the Current Data and Time folder on the CD.

Listing 6.12: Finding the Current Date and Time

```
<?php
echo date ("l")."<br/>";
echo (date ("l dS \of F Y h: i: s a"). " "); ?>
```

Output:

```
Thursday Thursday 16th of July 2009 11:37:03 pm
```

Validating a Given Date

- **checkdate ()** – Validates a date. Following syntax is used to define the `checkdate()` function:

  Syntax:

  ```
  checkdate (int month ,int day ,int year)
  ```

- **Parameters:**
  - month-required. Specifies the month
  - day -required. Specifies the day
  - year -required. Specifies the year
- **Return Value** – Returns **true** if the specified date is valid; else returns false.

Listing 6.13 shows how to validate a given date using the `checkdate()` function. You can find listing 6.13 in \Code\PHP\Chapter 06\Validating a Given Date folder on the CD.

Listing 6.13: Validating a Given Date

```
<?php
var_dump(checkdate(12, 31, 2000));
var_dump(checkdate(2, 29, 2001));
?>
```

Output:

```
bool(true) bool(false)
```

NOTE

The var_dump function displays structured information about expressions that includes its type and value.

In the preceding listing, the `checkdate()` function checks whether the given date is correct or not and returns true or false accordingly.

Returning the Unix Timestamp for a Date

- **mktime ()** -- Returns the Unix timestamp for a date. Following syntax is used to define the `mktime()` function:

  Syntax:

  ```
  int mktime (hour, minute, second, month, day, year, is_dst)
  ```

- **Parameters:**
  - hour - Optional. Specifies the hour
  - minute - Optional. Specifies the minute
  - second - Optional. Specifies the second
  - month - Optional. Specifies the numerical month
  - day - Optional. Specifies the day
  - year - Optional. Specifies the year
  - is_dst - Optional. Set this parameter to 1 if the time is during daylight savings time (DST)
- **Return Value**—Returns the Unix timestamp of the given Date

Listing 6.14 shows how to convert a date into Unix timestamp using the `mktime()` function. You can find listing 6.41 in \Code\PHP\Chapter 06\Returning the Unix Timestamp for a Date folder on the CD.

Listing 6.14: Returning the Unix Timestamp for a Date

```
<?php
echo(date("M-d-Y",mktime(0,0,0,12,36,2001))."<br/>");
echo(date("M-d-Y",mktime(0,0,0,14,1,2001))."<br/>");
echo(date("M-d-Y",mktime(0,0,0,1,1,2001))."<br/>");
echo(date("M-d-Y",mktime(0,0,0,1,1,99))."<br />");
?>
```

Output:

```
Jan-05-2002
Feb-01-2002
Jan-01-2001
Jan-01-1999
```

In the preceding listing, the `mktime()` function returns the Unix timestamp of the given date in the specified format.

Parsing English Textual Datetime Description into a Unix Timestamp

- **strtotime ()** -- Parses an English textual date or time into a Unix timestamp. Following syntax is used to define the strtotime() function:

  Syntax:

  ```
  int strtotime (time , now)
  ```

- **Parameters:**
  - time- Required. Specifies the time string to parse
  - now - Optional. Specifies the timestamp used to calculate the returned value
- **Return Value**—Returns a timestamp on success, false otherwise.

Listing 6.15 shows how to convert the textual date or time into the Unix timestamp using the `strtotime()` function. You can find listing 6.15 in \Code\PHP\Chapter 06\Parsing English Datetime Description into a Unix Timestamp folder on the CD.

Listing 6.15: Parsing English Textual Datetime Description into a Unix Timestamp

```
<?php
echo (strtotime ("now")."<br/>");
echo (strtotime ("3 October2005"). "<br />");
echo (strtotime ("+5 hours"). "<br />");
echo (strtotime ("+1 week"). "<br />");
echo (strtotime ("+1 week 3 days 7 hours 5 seconds"). "<br />");
echo (strtotime ("next Monday"). "<br />");
echo(strtotime("last Sunday"));
?>
```

Output:

```
1138614504
1128290400
1138632504
1139219304
1139503709
1139180400
1138489200
```

In the preceding listing, the `strtotime()` function returns the Unix timestamp of the given date or time.

Table 6.2 lists some commonly used date and time functions:

Table 6.2: Date and Time Functions

Function	Description	Version
`date_default_timezone_get()`	Returns the default time zone	5
`date_default_timezone_set()`	Sets the default time zone	5
`date_sunrise()`	Returns the time of sunrise for a given day / location	5
`date_sunset()`	Returns the time of sunset for a given day / location	5
`getdate()`	Returns an array that contains date and time information for a Unix timestamp	3
`gettimeofday()`	Returns an array that contains current time information	3
`gmdate()`	Formats a GMT/UTC date/time	3
`gmmktime()`	Returns the Unix timestamp for a GMT date	3
`idate()`	Formats a local time/date as integer	5
`localtime()`	Returns an array that contains the time components of a Unix timestamp	4
`microtime()`	Returns microseconds for the current time	3
`strftime()`	Formats a local time/date according to locale settings	3
`strptime()`	Parses a time/date generated with strftime()	5
`time()`	Returns the current time as a Unix timestamp	3

Mathematical Functions

The mathematical functions can handle values within the range of integer and float types. These functions are very helpful in addition, subtractions, multiplication and many other mathematical calculations, such as, calculating logarithmic values and rounded values. Let's learn to use some of the most common mathematical functions.

Generating Random Numbers

- **rand()** – Generates a random integer. Following syntax is used to define the rand() function:

  Syntax:

  ```
  int rand (void)  OR  int rand(int $min , int $max)
  ```

- **Parameters**:

- min, max - Optional. Specifies the range of the random numbers.

- **Return Value**—If this function is called without parameters, it returns a random integer between 0 and RAND_MAX.

Listing 6.16 shows how to get a random integer within the specified range using the `rand()` function:

Listing 6.16: Generating Random Numbers

```
<?php
echo(rand()."<br/>");
echo(rand()."<br/>");
echo(rand(10,100));
?>
```

Output:

```
14385
2876
45
```

In the preceding listing, the first two echo statements display the random integers between 0 and RAND_MAX while the third echo statement displays the random integer between 10 and 100.

Calculating Logarithmic Values

- **log()**—Returns the natural logarithm. Following syntax is used to define the `log()` function:

  Syntax:

  ```
  log($arg , $base)
  ```

- **Parameters**:
  - arg - Required. The value to calculate the logarithm for
  - base - Optional. If the base parameter is specified, `log()` returns $\log_{base} x$.
- **Return Value**—The logarithm of arg to base, if given, or the natural logarithm

Listing 6.17 shows how to return the logarithm using the `log()` function.

Listing 6.17: Calculating Logarithmic Values

```
<?php
echo (log(2.7183)."<br/>");
echo (log (2)."<br/>");
echo (log (1)."<br/>");
echo (log (0). "<br />");
?>
```

Output:

```
1.00000668491
0.69314718056
0
-INF
```

In the preceding listing, the `log()` function returns the logarithm of a given value.

Rounding Floating Point Numbers

- **round()**—Rounds a float number. Following syntax is used to define the round() function:

  Syntax:

  ```
  round( float $val , int $precision)
  ```

- **Parameters**:
  - val - Required. The value to round
  - precision - Optional. The number of digits after the decimal point
- **Return Value**—The rounded value

Listing 6.18 shows how to find the rounding value of a float number using the `round()` function. You can find listing 6.18 in \Code\PHP\Chapter 06\Rounding Floating Point Numbers folder on the CD.

Listing 6.18: Rounding Floating Point Numbers

```
<?php
echo round(3.4)." ";
 echo round(3.5)." ";
echo round (3.6)." ";
echo round (3.6, 0)." ";
echo round (1.95583, 2)." ";
echo round(1241757, -3)." ";
echo round (5.045, 2)." ";
echo round (5.055, 2)." ";
?>
```

Output:

```
3 4 4 4 1.96 1242000 5.05 5.06
```

In the preceding listing, the `round()` function returns the rounded value of a given value.

- **ceil()**—Rounds up the fractions number. Following syntax is used to define the `ceil()` function:
  Syntax: `float ceil( float $value )`
- **Parameters:**
  - value - Required. The value to be rounded.
- **Return Value**—Rounds up the value to the next highest integer. The return value of `ceil()` is still of type float as the value range of float is usually bigger than that of integer.

Listing 6.19 shows how to find the next highest integer value of a float number using the `ceil()` function:

Listing 6.19: Rounding Float Value to the Next Highest Integer

```
<?php
echo ceil(4.3)." ";
echo ceil(9.999)." ";
echo ceil(-3.14)." ";
?>
```

Output:

```
5 10 -3
```

In the preceding listing, the `ceil()` function returns the rounded value of a given value. Table 6.3 lists some commonly used math functions:

Table 6.3: Math Functions

Function	Description	Version
`abs()`	Returns the absolute value of a number	3
`base_convert()`	Converts a number from one base to another	3
`bindec()`	Converts a binary number to a decimal number	3
`ceil()`	Returns the value of a number rounded upwards to the nearest integer	3
`cos()`	Returns the cosine of a number	3
`cosh()`	Returns the hyperbolic cosine of a number	4
`decbin()`	Converts a decimal number to a binary number	3
`dechex()`	Converts a decimal number to a hexadecimal number	3
`exp()`	Returns the value of E^x	3
`expm1()`	Returns the value of $E^x - 1$	4
`floor()`	Returns the value of a number rounded downwards to the nearest integer	3
`fmod()`	Returns the remainder (modulo) of the division of the arguments	4
`hexdec()`	Converts a hexadecimal number to a decimal number	3
`is_infinite()`	Returns true if a value is an infinite number	4
`is_nan()`	Returns true if a value is not a number	4

Table 6.3: Math Functions

Function	Description	Version
max()	Returns the number with the highest value of two specified numbers	3
min()	Returns the number with the lowest value of two specified numbers	3
octdec()	Converts an octal number to a decimal number	3
pi()	Returns the value of PI	3
pow()	Returns the value of x to the power of y	3
sin()	Returns the sine of a number	3
sqrt()	Returns the square root of a number	3
tan()	Returns the tangent of an angle	3

Recursive, Variable, and Callback Functions

Recursive function is a function that calls itself repeatedly for a specified condition. Using recursion, you can break a complex problem into simpler parts. It increases the flexibility and adaptability of a function in the same way as a loop increases the flexibility and adaptability of a static statement or block of code. Recursive functions are particularly useful for navigating dynamic data structures such as linked lists and trees. Following are some points, which must be keep in mind while using recursion:

- You must have an exit condition (the part of the function that can be calculated without further calls to that function).
- Each recursive call must be different than the one before it.

Let's take an example to understand how recursion works.

Example: Factorial (The factorial for any given number is equal to that number multiplied by the factorial of the number one lower. 6! =6*5*4*3*2*1=720).

Listing 6.20 shows how to use recursion to calculate factorial of a given number. You can find listing 6.20 in \Code\PHP\Chapter 06\Calculating Factorial folder on the CD.

Listing 6.20: Calculating the Factorial Using Recursion

```
<? php
function factorial($number) {
 if ($number < 2) {
        return 1;
    } else {
        return ($number * factorial($number-1));
    }
}
echo factorial(6);
?>
```

Output:

```
720
```

In the preceding listing, The `factorial()` function calls itself again and again on a smaller version of the input and multiplies it by the result of recursive call until it reaches the base condition. PHP supports the concept of variable functions. If a variable name has parentheses appended to it, PHP searches for a function with the same name as the variable's name. and attempts to execute it. Variable functions can be used to implement callbacks, function tables, and so forth. Listing 6.21 shows how to use the variable function. You can find listing 6.21 in \Code\PHP\Chapter 06\Variable Function folder on the CD.

Listing 6.21: Variable Function

```
<?php
  function myfunc($msg) {
    echo $msg."<br />";
}
  $func_name = 'myfunc';
```

```
    $func_name("This is a variable function calling");
?>
```

Output:

```
This is a variable function calling
```

In the preceding listing, we call the function `myfunc ()`, through the use of variable `$func_name`, that contains the actual name of the function and our message "This is a variable function calling".

NOTE

Variable functions do not work with language constructs such as echo (), print (), unset (), isset(), empty(), include(), and require().

Callback function refers to a function that is passed to another function so that the second function can call it. This is the simple way of making the second function more flexible. By passing different callback functions you can get different behavior. Listing 6.22 shows how to use the callback function. You can find listing 6.22 in \Code\PHP\Chapter 06\Callback Function folder on the CD.

Listing 6.22: Callback Function

```
<? php
// This function ses a callback function.
 function doIt($callback) {
   $data = "This is my data";
   $callback($data);
}
// This is a sample callback function for doIt ().
function myCallback($data)
{
  echo $data. "\n";
 }
 doIt ('myCallback');
// Call doIt() and pass our sample callback function's name.
?>
```

Output:

```
This is my data
```

In the preceding listing, we make a function named, `doIt()` with an argument `$callback` and then we use this variable `$callback` as a variable function. We make another function named `myCallback()`. Now, when we call our `doIt()` function by passing the 'myCallback' as an argument, PHP looks for `myCallback()` function and attempts to execute it. As a result, "this is my data" is printed on the screen. In this section, you have learned to create and invoke the user-defined functions and use the built-in functions. In the next section, you learn to work with a special type of variable, array.

Introducing Arrays

You have already learnt that a variable is a storage area holding numbers and text. The problem is that a variable can hold only one value at a time while an array is a special variable, which can hold more than one value at a time. An array can hold all variable values under a single name. You can access the values of an array by referring to the array name.

Creating an Array

Creating an array in PHP is a simple task. An `array()` construct is used in PHP to create an array. Let's start with creating an empty array. That is, an array with no index or values. Listing 6.23 shows how to create an empty array using the `array()` construct. You can find listing 6.23 in \Code\PHP\Chapter 06\Creating an Empty Array folder on the CD.

Listing 6.23: Creating an Empty Array

```
<?php
//create an empty array
$array=array();
```

```
?>
```

In the preceding listing, we have created a variable named $array, which is an empty array using `array()` construct. PHP provides a `is_array()` function to check if a variable is an array or not. The `is_array()` function takes a variable as its only argument and returns a Boolean TRUE if the variable is an array else it returns FALSE. Listing 6.24 checks whether the variable, $array is an array or not using the `is_array()` function. You can find listing 6.24 in \Code\PHP\Chapter 06\Checking Variable folder on the CD.

Listing 6.24: Checking the $array Variable

```
<?php
 $array = array();                          // create an empty array
 if(is_array($array))                       //check if $array is an array
 {                                          // if it is an array
        echo "Variable is an array";  }
 else {                                        // if it is not an array
        echo "Variable is not an array"; }
?>
```

Output:

```
Variable is an Array
```

In the preceding listing, we first create an empty variable, `$array` and then we check whether this variable is an array or not using the `is_array()` function. The output variable is an array shows that the variable, $array is an array. As mentioned above an array is map of key (index) and value pairs. Each key is assigned a value. The value can be in the form of variables, arrays, or objects. Listing 6.25 shows how to create an array with elements. You can find listing 6.25 in \Code\PHP\Chapter 06\Creating an Array with Elements folder on the CD.

Listing 6.25: Creating an Array with Elements

```
<?php
  // create an array with some element
  $cars = array('Lio', 'Spark', 'Scorpio', 'Safari');
?>
```

In the preceding listing, we have not specified any keys (indexes). If no key is specified, PHP automatically assigns a numeric value to the key, thus creating a numeric array. The count of the index begins at zero (0).You have learned to create an array with elements, lets learn to print the elements of an array. Listing 6.26 shows how to print the elements of an array using the `print_r()` function. You can find listing 6.26 in \Code\PHP\Chapter 06\Printing the Array Elements folder on the CD.

Listing 6.26: Printing the Array Elements

```
<?php
// create an array with some element
  $cars = array('Lio', 'Spark', 'Scorpio', 'Safari');
 //print the array contents
  print_r( $cars );
?>
```

Output:

```
Array ( [0] => Lio [1] => Spark [2] => Scorpio [3] => Safari )
```

In the preceding listing, first we create array variable $cars with elements Lio, Spark, Scorpio, Safari and then print the elements using the `print_r()` function.

NOTE

The print_r() function outputs human-readable information about a variable and is an excellent tool for quickly debugging or viewing array contents.

Accessing Array Elements

PHP allows direct access to array values by specifying the array name and the index. Listing 6.27 shows how to access a specific array element using the index number. You can find listing 6.27 in \Code\PHP\Chapter 06\Using Index Number folder on the CD.

Listing 6.27: Accessing Array Elements Using the Index Number

```
<?php
  // create an array with some element
  $cars = array('Lio', 'Spark', 'Scorpio', 'Safari');
  echo  "The second element of the array is:".$cars[1];
?>
```

Output:

```
The second element of the array is : Spark
```

In the preceding listing, we first create an array variable $cars with elements Lio, Spark, Scorpio, Safari and then access its second element. The array index of 1 (one) has been used within the array index operators [ ], and the output prints the value of the array element with the index 1, Spark.

Types of Arrays

Following are three types of arrays in PHP:

- **Numeric Array** – Defines an array with a numeric ID key
- **Associative Array** – Defines an array where each ID key is associated with a value
- **Multidimensional Array** – Defines an array containing one or more arrays

Now let's discuss different types of arrays in details.

Numeric Arrays

Numeric arrays use integer values as their index number to identify each item of the array. The array examples discussed in previous listings are numeric arrays as they have integer values as index numbers for each item. Listing 6.28 shows the numeric array. You can find listing 6.28 in \Code\PHP\Chapter 06\ Numeric Array folder on the CD.

Listing 6.28: Using Numeric Array

```
<?php
$colors = array('white','black','blue');
print_r($colors);
?>
```

Output:

```
Array( [0] => white [1] => black [2] => blue [3])
```

The output of the preceding listing shows the index numbers for white, black and blue are 0, 1, 2 respectively, which are numeric values. Hence we call such arrays numeric arrays.

Associative Arrays

In an associative array an index is associated with a value. For example, if you wanted to store the salaries of employees you can use the employees names as the keys in an associative array and the output value would be their respective salary. Listing 6.29 shows how to use values as index numbers. You can find listing 6.29 in \Code\PHP\Chapter 06\Using Values as Index Number folder on the CD.

Listing 6.29: Using Values as Index Numbers

```
<?php
$salaries["Bob"] = 2000;
$salaries["Sally"] = 4000;
$salaries["Charlie"] = 600;
$salaries["Clare"] = 0;
echo "Bob is being paid - $" . $salaries["Bob"] . "<br />";
echo "Sally is being paid - $" . $salaries["Sally"] . "<br />";
echo "Charlie is being paid - $" . $salaries["Charlie"] . "<br />";
echo "Clare is being paid - $" . $salaries["Clare"];
?>
```

Output:

```
Bob is being paid - $2000
Sally is being paid - $4000
Charlie is being paid - $600
Clare is being paid - $0
```

The preceding listing shows how we can use names Bob, Sally, Charlie, Clare as index to access their respective salaries instead of using the integers in an array. We assign different salaries using array variable, $salaries to different employees and after that we are accessing the salary values using the name of the employees.

Multidimensional Arrays

A multidimensional array is an array that can contain sub arrays and the sub arrays can further contain sub arrays within them. You can understand the concept of multidimensional arrays using the following example. David has two sons Richie and Mason. Richie has two daughters Sue and Natasha while Mason has three daughters Nichole, Salma and Amber. Figure 6.1 shows the family tree structure of David's family:

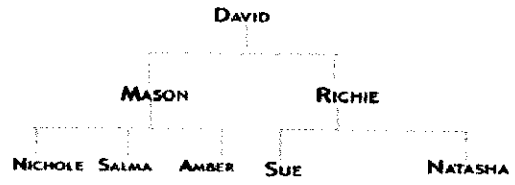


Figure 6.1: Family Tree Structure of David's Family

Listing 6.30 defines a multidimensional array to show the family tree structure of David's family. You can find listing 6.30 in \Code\PHP\Chapter 06\Using Multidimensional folder on the CD.

Listing 6.30: Using Multidimensional Array

```
<?php
   // define a multidimensional array
   $david = array
   (
      'richie'=>array('Sue', 'Natasha'),
      'Mason'=>array ('Nichole', 'Salma', 'Amber')
   );
print_r($david);
?>
```

Output:

```
Array ( [richie] => Array ( [0] => Sue [1] => Natasha ) [Mason] => Array ( [0] =>
  Nichole [1] => Salma [2] => Amber ) )
```

In the preceding listing, we use a multidimensional array $david to show the family tree structure of his family. Array $david has two sub arrays richie and Mason. Array richie has two elements Sue and Natasha and array Mason has three elements Nichole, Salma, and Amber.

Traversing Arrays using Loops and ArrayIterators

Array traversal is useful to work on each element of an array, such as, sending a mail to each element in an array of addresses and updating each file in an array of filenames. There are several ways to traverse an array in PHP, and the one you choose depends on data and the task you are performing.

The foreach Construct

PHP 4 has introduced a foreach construct. It provides an easy way to iterate over arrays. foreach works only on arrays, and issues an error when you try to use it on a variable with a different data type or an uninitialized variable. There are two syntaxes for define the foreach loop. Following syntax is used to define the foreach loop:

Syntax:

```
First:   foreach (array_expression as $value) {
               //statement
         }
Second: Foreach (array_expression as $key => $value) {
               //statement
         }
```

In the preceding syntax, first forms a loop over the array given by `array_expression`. On each loop, the value of the current element is assigned to $value and the internal array pointer is increased by one. The second form does the same thing, except that the current element's index is assigned to the variable $key on each loop. Listing 6.31 shows the use of foreach loop to traverse an array. You can find listing 6.31 in \Code\PHP\Chapter 06\Using the foreach Loop folder on the CD.

Listing 6.31: Using the foreach Loop

```
<?php
$students = array();
//assigning values
$students[0] = "david";
$students[1] = "kevin";
$students[2] = "julie";
$students[3] = "nayyar";
// now we will use the foreach loop to display all the students names i.e the array alues
   in one go foreach ( $students as $std_name )
{
echo $std_name . "\n";}
?>
```

Output:

```
david kevin julie nayyar
```

In the above code the `$std_name` works as a temporary variable to get each value of array. When the loop executes, the next available value of the array overwrites the existing value of the `$std_name` and then `$std_name` points to the current fetched value. It works just as walking through your array values one by one.

The ArrayIterator

Every PHP array keeps track of the current element you are working on; the pointer to the current element is known as the `iterator`. PHP has a ready-made extensible tool to loop over the array elements called `ArrayIterator` (new in PHP 5.0) to set, move, and reset the `iterator`. The `ArrayIterator` functions are:

- **current()** – Returns the element currently pointed by the `iterator`
- **reset()** – Moves the `iterator` to the first element in the array and returns its value
- **next()** – Moves the `iterator` to the next element in the array and returns its value
- **prev()** – Moves the `iterator` to the previous element in the array and returns its value
- **end()** – Moves the `iterator` to the last element in the array and returns its value
- **each()** – Returns the index and value of the current element as an array and moves the `iterator` to the next element in the array
- **key()** – Returns the index of the current element

Listing 6.32 shows the use of the `ArrayIterator` class to traverse an array. You can find listing 6.32 in \Code\PHP\Chapter 06\Using the ArrayIterator folder on the CD.

Listing 6.32: Using the `ArrayIterator`

```
<?php
//define array
$cities=array("United Kingdom"=>"London", "United State"=>"Washington", "France"=>"Paris",
   "India"=>"Delhi");
//create an ArrayIterator object
$iterator= new ArrayIterator($cities);
//rewind to beginning of array
$iterator->rewind();
// iterator over array
// print each value
while($iterator->valid()) {
print $iterator->current()." is in ".$iterator->key()."</br>";
$iterator->next();
}
?>
```

Output:

```
London is in United Kingdom
Washington is in United State
Paris is in France
Delhi is in India
```

In the preceding listing, an `ArrayIterator` object is initialized with an array variable, and the object's `rewind()` method is used to reset the internal array pointer to the first element of the array. A while loop executes until the `valid()` function returns true. Individual array keys are retrieved with the `current()` function. The `next()` function moves the internal array pointer forward to the next array element.

Built-in Array Functions

Array functions allow us to manipulate arrays. PHP supports both simple and multi-dimensional arrays. There are some specific functions for populating arrays from results retrieved from the database queries. Following are some commonly used array functions to manipulate arrays in PHP:

Getting the Size of an Array

- **count()**—Counts all elements in an array, or properties in an object. Syntax to define the `count()` function is as follows:

  ```
  Syntax: int count (array, mode)
  ```

- **Parameters**:
  - array -Required. Specifies the array or object to count.
  - mode - Optional. Specifies the mode of the function. Possible value are 0 or 1.

Listing 6.33 shows the use of `count()` function to count the number of elements in an array. You can find listing 6.33 in \Code\PHP\Chapter 06\Using count Function folder on the CD.

Listing 6.33: Using `count()` Function

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
$result = count($people);
echo $result;
?>
```

Output:

```
4
```

In the preceding listing, we create an array, people, which has elements Peter, Joe, Glenn, Cleveland. The count function counts the array, people and returns the output 4.

Searching For the Occurrence of a Given Value

- **array_search()**—Searches an array for a value and returns the index. Syntax to define the array_search() function is:

  ```
  Syntax: array_search(value, array, strict)
  ```

- **Parameter**:
  - value- Required. Specifies the value to search for
  - array- Required. Specifies the array to search in
  - strict- Optional. Possible values are true and false.

Listing 6.34 shows how to use an `array_search()` function. You can find listing 6.34 in \Code\PHP\Chapter 06\Using array search function folder on the CD.

Listing 6.34: Using the `array_search()` Function

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
echo array_search("Dog",$a);
?>
```

Output:

```
a
```

In the preceding listing, we create an array, `$a` and at index a, b, and c and assign values Dog, Cat, and Horse respectively. When we print the index of dog in array $a using `array_search()` function, it returns a as the output.

Sorting an Array

- **sort()**—Sorts an array. Following syntax is used to define the sort() function:

  Syntax:

  ```
  sort(array, sorttype)
  ```

- **Parameter**:
  - array- Required. Specifies an array to sort
  - sorttype- Optional. Specifies how to sort the array values. Possible values are:
    - SORT_REGULAR - Default. Treat values as they are (don't change types).
    - SORT_NUMERIC - Specifies values numerically.
    - SORT_STRING - Specifies values as strings.
    - SORT_LOCALE_STRING - Specifies values as strings, based on local settings.

Listing 6.35 shows the use of `sort()` function to sort array elements in alphabetical order. You can find listing 6.35 in \Code\PHP\Chapter 06\Using the sort Function folder on the CD.

Listing 6.35: Using the `sort()` Function

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");
sort($my_array);
print_r($my_array);
?>
```

Output:

```
Array([0] => Cat[1] => Dog[2] => Horse)
```

Table 6.4 lists some commonly used array functions:

Table 6.4: Array Functions

Function	Description	Version
`array_fill()`	Fills an array with values	4
`array_intersect()`	Compares array values, and returns the matches	4
`array_merge()`	Merges one or more arrays into one array	4
`array_multisort()`	Sorts multiple or multi-dimensional arrays	4
`array_pop()`	Deletes the last element of an array	4
`array_product()`	Calculates the product of the values in an array	5
`array_push()`	Inserts one or more elements at the end of an array	4
`array_reverse()`	Returns an array in the reverse order	4
`array_shift()`	Removes the first element from an array, and returns the value of the deleted element	4
`array_sum()`	Returns the sum of the values in an array	4
`array_unique()`	Removes duplicate values from an array	4
`array_values()`	Returns all the values of an array	4
`extract()`	Imports variables into the current symbol table from an array	3
`in_array()`	Checks if a specified value exists in an array	4
`krsort()`	Sorts an array by index in reverse order	3
`ksort()`	Sorts an array by index	3

Table 6.4: Array Functions

Function	Description	Version
`list()`	Assigns variables as if they were an array	3
`prev()`	Rewinds the internal array pointer	3
`range()`	Creates an array containing a range of elements	3
`reset()`	Sets the internal pointer of an array to its first element	3
`rsort()`	Sorts an array in reverse order	3
`uksort()`	Sorts an array by index using a user-defined function	3
`usort()`	Sorts an array by values using a user-defined function	3

Working with Files and Directories

PHP: Hypertext Preprocessor (PHP) comes with a powerful and flexible file manipulation Application Programming Interface (API) that allows a developer to perform various functions with the help of files and directories. A computer file is a collection of related data or program records that are stored as a unit with a single name. A file type is identified by a dot followed by a series of letters, with the name of the file appearing before the dot, such as name.doc or pic.jpg. The letters after the dot define the type of the file, such as .doc refers to a text file; whereas, the .jpg refers to a picture file. The storage place where the computer files are stored in a logical order is known as a directory. A directory can also contain other directories, which are called subdirectories of the main directory. A system may contain many directories with innumerable files and subdirectories in them.

In this chapter, you learn about various functions provided by the PHP language to work with files and directories. You also learn how to perform various operations, using these functions, such as read and write files, view and modify file attributes, read and list directory contents, alter file permissions, and retrieve file contents. Let's start with discussing how to work with files.

Working with Files

As already discussed, a file is a block of information, containing related data, which is stored on a computer system. The following are some of the operations that can be performed on a file using various functions:

- Opening a file
- Closing a file
- Reading a file
- Writing a file
- Retrieving File Status
- Locking File with the flock() Function

Opening a File

The `fopen()` function is used to open a file in PHP. The syntax of using the `fopen()` function is:

```
fopen(string filename, string mode)
```

In the preceding syntax, the filename parameter refers to the name of the file or the path of the file in the file system that needs to be opened. The filename is written in string format. The second parameter in the `fopen()` function is the mode in which the file is opened. The two basic modes are "r"(read) and "w"(write). Table 6.5 lists various modes that are used to open a file:

Table 6.5: List of Modes to Open a File

Modes	Description
r	Refers to the Read only mode. It starts at the beginning of the file.
r+	Refers to the Read/Write mode. It starts at the beginning of the file.
w	Refers to the Write only mode. It allows the user to clear the content and write in the file.

Table 6.5: List of Modes to Open a File	
w+	Refers to the Read/Write mode. It allows the user to read as well as write in the file. Similar to the Write only mode, this mode also clears the content of the file.
a	Refers to the Append mode. It adds content at the end of an existing file. It also creates a new file to add content, if no file already exists.
a+	Refers to the Read/Append mode. It adds content at the end of an existing file.
x	Refers to the Write only mode. It creates a new file to write content in it. This mode works on a new file and returns FALSE and an error, if a file already exists.
x+	Refers to the Read/Write mode. It creates a new file to read and write content in it. This mode works on a new file and returns FALSE and an error if a file already exists.

NOTE

If the fopen() function is unable to open the specified file, it returns 0 (false) And The path of the file should always be given in single quotes

You can find listing 6.36 in \Code\PHP\Chapter 06\Using fopen function folder on the CD.

Listing 6.36: Showing the Use of the `fopen()` Function

```
<?php
  $file=fopen("test.txt","r") or die("Unable to open file");
  echo "File successfully opened";
?>
```

The code, shown in Listing 6.36, shows how to open an already created file, test.txt, in the Read only mode. If the file is successfully opened, the echo statement prints "File successfully opened"; otherwise, it prints "Unable to open file". You can also access a file from a remote location by typing its (Uniform Resource Locator) URL in the `fopen()` function. The syntax to access a file by entering its URL address is:

```
fopen('http://www.example.com/', "r");
fopen('ftp://ftp.example.com/incoming/test.txt',"r");
```

In the preceding syntax, http://www.example.com and ftp://ftp.example.com/incoming/test.txt are the URLs that can be used to access a file.

NOTE

The test.txt file used in listings exists in the directory of PHP. However, you may specify your own path for the file.

Now, let's see how to close a file.

Closing a File

After working and saving the changes in the file, you can close it By using the `fclose()` function The syntax of using the `fclose()` function is:

```
fclose(handle)
```

In the preceding syntax, the handle parameter refers to the variable that holds the reference of the file. You can find listing 6.37 in \Code\PHP\Chapter 06\Using fclose Function folder on the CD.

Listing 6.37: Showing the Use of the `fclose()` Function

```
<?php
  $file=fopen("test.txt","r") or die("Unable to open file");
  echo "File successfully opened";
  fclose($file) or die("Unable to close file");
  echo "File successfully closed";
?>
```

Listing 6.37 shows that the test.txt file, which is opened in the Read only mode, is closed using the `fclose()` function. If the file is successfully closed, the echo statement prints "File successfully closed"; otherwise, it prints "Unable to close file".

Reading a File

In PHP, a file can be read using different functions. The following are some of the functions that can be used to read a file:

- The `fgets()` function
- The `fread()` function
- The `file()` function
- The `file_get_contents()` function
- The `readfile()` function
- The `fgetc()` function

The fgets() Function

The `fgets()` function is used to read a file line by line. The syntax to use the `fgets()` function is:

```
fgets (handle, length)
```

In the preceding syntax, the handle parameter refers to the variable that holds the reference of the file and *length* refers to the size of data that is extracted at a time. Note that it is not mandatory to provide the length parameter; by default, the function reads only 1024 bytes at a time. Listing 6.38 shows how to use the `fgets()` function to read the content of a file. You can find listing 6.38 in \Code\PHP\Chapter 06\Using fgets function folder on the CD.

Listing 6.38: Showing the Use of the `fgets()` Function

```
<?php
   $file=fopen("test.txt","r") or die("Unable to open file");
   echo "File successfully opened <br>";
   while (!feof($file))
   {
         $data=fgets($file,512);
         echo $data."<br>";
   }
   fclose($file) or die("Unable to close file");
   echo "<br>File successfully closed";
?>
```

The code, shown in Listing 6.38, shows that the test.txt file is opened using the `fopen()` function. The $file variable acts as the handle referencing the file. The while loop is used to print the contents of the file using the `fgets()` function. You can notice that a new function, `feof()`, is used in the while loop. The `feof()` function returns Boolean value and is used to test whether or not the end of a file has been reached. You can also see that in this case, the size of the data extracted at a time is 512 bytes.

The fread() Function

The `fread()` function reads data from a file and reports it back to the PHP file. It reads only the specified number of bytes. The syntax to use the `fread()` function is:

```
fread(handle, length)
```

In the preceding syntax, the handle parameter refers to the variable holding the reference of the file and length refers to the size of the data that is extracted at a time. The length is measured in bytes. The difference between the `fgets()` and `fread()` functions is that the `fgets()` function reads up to length-1 bytes; whereas, the `fread()` function reads up to length bytes.

The file() Function

The `file()` function retrieves data from a file and puts it into an array. The array stores data in form of lines. The count of these lines begins from 0. The syntax to use the `file()` function to retrieve data is:

```
file(filename)
```

In the preceding syntax, filename is a parameter that refers to the name of the file. Listing 6.39 shows how to use the `file()` function to read the content of a file. You can find listing 6.39 in \Code\PHP\Chapter 06\File Function folder on the CD.

Listing 6.39: Showing the Use of the file () Function

```
<?php
  $lines = file('test.txt');
  foreach ($lines as $line_num => $line)
  {
        print "Line".$line_num. " : " . $line . "<br />\n";
  }
?>
```

In Listing 6.39, the test.txt file is opened in the form of an array using the `file()` function. This array is then passed to the $lines variable. Next, the foreach loop is used to print the contents of the file showing the line numbers.

The file_get_contents() Function

The `file_get_contents()` function works similar to the `file()` function except that it returns a string instead of an array. This function retrieves data from an external file and then puts it into a string for further use. The syntax to retrieve data using the `file_get_contents()` function is:

```
file_get_contents(filename)
```

In the preceding syntax, the filename parameter refers to the name of the file that you want to read. Listing 6.40 shows how to use the `file_get_contents()` function to retrieve data from a file:

Listing 6.40: Showing the Use of the `file_get_contents()` function

```
<?php
  $file = file_get_contents ('test.txt');
  echo $file;
?>
```

The `file_get_contents()` function, shown in Listing 7.5, prints the content of a file in a single line of string.

The readfile() Function

The readfile() function first takes a filename as its single argument and reads the whole file. It then displays the content on the standard output. The syntax to read a file using the `readfile()` function is:

```
readfile(filename)
```

In the preceding syntax, the filename parameter refers to the name of the file that you want to read. Listing 6.41 show how to read data from a file using the `readfile()` function. You can find listing 6.41 in \Code\PHP\Chapter 06\Using readfile Function folder on the CD.

Listing 6.41: Showing the Use of the `readfile()` Function

```
<?php
  echo readfile("test.txt");
?>
```

The `readfile()` function, shown in Listing 7.6, displays the content of the file as the output.

The fgetc() Function

The `fgetc()` function is used to read one character at a time from a file. In case the location of the file pointer is placed at the end of file (EOF), the FALSE value is returned. The syntax to read a file using the `fgetc()` function is:

```
fgetc(filename)
```

In the preceding syntax, the filename parameter refers to the name of the file that you want to read. Listing 6.42 shows how to use the `fgetc()` function to read the content of a file:

Listing 6.42: Showing the Use of the `fgetc()` Function

```
<?php
  $file = fopen("test.txt", "r");
  while (! feof ($file))
  {
        echo fgetc ($file);
  }
  fclose ($file);
```

```
?>
```

The `fgetc()` function, shown in Listing 6.42, prints the content of the file, character by character.

NOTE

The fgetc() function should not be used to read large amounts of data. However, in cases where you need to read a large file, one character at a time, use the fgets() function to read the data one line at a time, and then process each line, one character at a time with a looping construct, such as while loop or for loop.

Writing a File

The process of writing a file in PHP is similar to that of reading a file. The following are some of the functions that can be used to write in a file:

- The `fwrite()` function
- The `fputs()` function

The fwrite() Function

The `fwrite()` function is used to write content in an already opened file. This function, if successful, returns the number of bytes that are written in a file; otherwise returns a FALSE value. The syntax for the `fwrite()` function is:

```
fwrite(filename/handle,string,length)
```

In the preceding syntax, the filename/handle parameter refers to the name of the file in which you need to write content. The second parameter, *string*, specifies the content, in the form of a string, which you want to write in the file. The next parameter, length (optional), specifies the maximum number of bytes to write to the file.Listing 6.43 shows how to use the `fwrite()` function to write the content in a file. You can find listing 6.43 in \Code\PHP\Chapter 06\Using the fwrite Function folder on the CD.

Listing 6.43: Showing the Use of the `fwrite()` Function

```
<?php
   $file=fopen("test.txt","a+") or die("Unable to open file");
   $data="New text is added";
   fwrite($file,$data,3) or die("Could not write");
   echo "Data successfully written to file";
   fclose($file) or die("File not closed");
?>
```

Listing 6.43 shows that the test.txt file is opened in the Read/Append (a+) mode. The string to be written is stored in the `$data` variable. In this example, the handle, the string, and the length of the string are passed as parameters.

The fputs() Function

The `fputs()` function is used to write in an already opened file. The function stops at the end of the file or when it reaches the specified length, whichever comes first. If successful, this function returns the number of bytes of the written content; otherwise, returns FALSE. The syntax of the `fputs()` function is:

```
fputs(filename/handle,string,length)
```

In the preceding syntax, the filename/handle parameter refers to the open file in which you want to write your content. The second parameter, string, specifies the string to be written in the open file. The length parameter is optional and used to specify the maximum number of bytes to write to the open file. Listing 6.44 shows how to use the `fputs()` function to write content in a file. You can find listing 6.44 in \Code\PHP\Chapter 06\Using the fputs Function folder on the CD.

Listing 6.44: Showing the Use of the `fputs()` Function

```
<?php
   $file=fopen("test.txt","a+") or die("Unable to open file");
   $data="New text is added";
   fputs($file,$data,3) or die("Could not write");
   echo "Data successfully written to file";
   fclose($file) or die("File not closed");
?>
```

The code, shown in Listing 6.44, shows that the `test.txt` file opens in the Read/Append (a+) mode, and a string New text is added is stored in the `$data` variable. In the next statement, the `$file`, `$data`, and 3 are used as parameters with the `fputs()` function to write in the `test.txt` file. However, only first three characters of the string, New text is added, is written in the `test.txt` file, as the length parameter in the `fputs()` function is 3. Finally, the test.txt file is closed.

Retrieving File Status

Sometimes, due to your unawareness of the current status of the file, you may perform an illegal operation that may generate an error. To avoid such situations, it is always advisable to retrieve the status of a file before you start working on it. The following are some of the functions that are used to retrieve the status of a file in PHP:

- The `file_exists()` function
- The `is_executable()` function
- The `is_file()` function
- The `is_link()` function
- The `is_readable()` function
- The `is_writable()` function

The file_exists() Function

The `file_exists()` function is used to check whether or not a file exists. This function returns the TRUE value, if the file exists; otherwise, it returns FALSE. The syntax of using the `file_exists()` function is:

```
file_exists(path)
```

In the preceding syntax, the path parameter refers to the path of the file. Listing 6.45 shows how to use the `file_exists()` function to check the existence of a file. You can find listing 6.45 in \Code\PHP\Chapter 06\File exists folder on the CD.

Listing 6.45: Showing the Use of the `file_exists()` Function

```
<?php
  if (file_exists("test.txt"))
  {
        print "test.txt exists!\n";
  }
  else
  {
        print "test.txt does not exist!\n";
  }
?>
```

The code, shown in Listing 6.45, checks the existence of the specified file and prints the test.txt exists! text, if the specified file is found; else, it prints test.txt does not exist!.

The is_executable() Function

The `is_executable()` function is used to check whether or not the specified file is executable. This function returns TRUE if the file is executable. The syntax of using the `is_executable()` function is:

```
is_executable(path)
```

Listing 6.46 shows how to use the `is_executable()` function. You can find listing 6.46 in \Code\PHP\Chapter 06\Is executable Function folder on the CD.

Listing 6.46: Showing the Use of the `is_executable()` Function

```
<?php
  if (is_executable("test.txt"))
  {
        print "test.txt is executable!\n";
  }
  else
  {
```

```
        print "test.txt is not executable!\n";
    }
?>
```

Listing 6.46, prints test.txt is executable!, if the file is executable; else, it prints test.txt is not executable!

The is_file() Function

The is_file() function is used to check whether or not the specified file is a regular file, such as .doc files and .txt files. This function returns TRUE, if the specified file is a regular file. The syntax of using the `is_file()` function is:

```
is_file(path)
```

In the preceding syntax, the path parameter refers to the path of the file. Listing 6.47 shows how to use the `is_file()` function to determine whether or not a file is regular. You can find listing 6.47 in \Code\PHP\Chapter 06\Using the is file Function folder on the CD.

Listing 6.47: Showing the Use of the `is_file()` Function

```
<?php
        if (is_file("test.txt"))
    {
        print "test.txt is a file!\n";
    }
    else
    {
        print "test.txt is not a file!\n";
    }
?>
```

The code, shown in Listing 6.47, prints the test.txt is a file text, if the specified file is a regular file.

The is_link() Function

The `is_link()` function is used to check whether or not the specified file is a link. This function returns TRUE, if the file is a link; else, it returns FALSE. The syntax of using the `is_link()` function is:

```
is_link(path)
```

In the preceding syntax, the path parameter refers to the path of the file. Listing 6.48 shows how to use the `is_link()` function to determine whether or not a file is a link. You can find listing 6.48 in \Code\PHP\Chapter 06\Using is link folder on the CD.

Listing 6.48: Showing the Use of the `is_link()` Function

```
<?php
        if (is_link("test.txt"))
    {
        print "test.txt is a link!\n";
    }
    else
    {
        print "test.txt is not a link!\n";
    }
?>
```

The code, shown in Listing 6.48, prints the test.txt is a link! text, if the specified file is a link.

The is_readable() Function

The is_readable() function is used to check whether or not the specified file is readable. This function returns TRUE, if the file is readable; else, it returns FALSE. The syntax of using the `is_readable()` function is:

```
is_readable(path)
```

In the preceding syntax, the path parameter refers to the path of the file.

Listing 6.49 shows how to use the `is_readable()` function to check whether or not the file is readable:

Listing 6.49: Showing the Use of the `is_readable()` Function

```
<?php
    if (is_readable("test.txt"))
    {
        print "test.txt is readable!\n";
    }
    else
    {
        print "test.txt is not readable!\n";
    }
?>
```

The code, shown in Listing 6.49, prints the test.txt is readable! text, if the test.txt file is readable; else, it prints test.txt is not readable!.

The is_writable() Function

The `is_writable()` function is used to check whether or not the specified file is writable. This function returns TRUE, if the file is writeable. The syntax of using the `is_writable()` function is:

```
is_writable(path)
```

Listing 6.50 shows how to use the `is_writable()` function to check whether or not a file is writable. You can find listing 6.50 in \Code\PHP\Chapter 06\Using the is writable Function folder on the CD.

Listing 6.50: Showing the Use of the `is_writable()` Function

```
<?php
    if (is_writable("test.txt"))
    {
        print "test.txt is writable!\n";
    }
    else
    {
        print "test.txt is not writable!\n";
    }
?>
```

The code, shown in Listing 6.50, prints the test.txt is writable! text, if the test.txt file is writable; else, it prints test.txt is not writable!.

Locking File Using the flock() Function

PHP provides the facility of locking a file that prevents the simultaneous access of a file by multiple users. The `flock()` function is used to lock a file. The syntax for using the `flock()` function is:

```
flock(handle,lock,block)
```

In the preceding syntax, handle, lock, and block are parameters. Table 6.6 lists the description of these parameters of the `flock()` function:

Parameter	Description
handle	Specifies the reference of a file that is to be locked. This is a mandatory parameter.
lock	Specifies the mode of lock to be applied on the file. This is a mandatory parameter.
block	Blocks two or more users to lock a file at the same time. This is an optional parameter.

Depending upon your requirements, you can lock a file in various modes. Table 6.7 lists the modes provided by PHP to lock a file:

Table 6.7: Lock Modes and their Description	
LOCK_SH	Refers to the shared lock. Allows other users to access the file in the Read only mode.
LOCK_EX	Refers to the exclusive lock. Prevents other users from accessing the locked file.
LOCK_UN	Unlocks the file that is locked by the shared or exclusive lock.
LOCK_NB	Prevents other users to apply for a locked file. The `flock()` function returns a FALSE value immediately, if the requested file is found to be locked.

Listing 6.51 shows how to use the `flock()`function to lock a file. You can find listing 6.51 in \Code\PHP\Chapter 06\Using the flock Function folder on the CD.

Listing 6.51: Showing the Use of the `flock()` Function

```
<?php
  $file = fopen("test.txt","w+");
  if (flock($file,LOCK_EX)) // exclusive lock {
  fwrite($file,"Kogent India");
  flock($file,LOCK_UN); // release lock
}
  else {
  echo "Could not lock file";
}
  fclose($file);
?>
```

In Listing 6.51, the test.txt file is opened in the Read/Write mode. If the exclusive lock is available for the file, the file is locked in the exclusive mode and Kogent India is written in the file; else, it prints Could not lock file. You can also see that the file is unlocked using the `LOCK_UN` mode. This is necessary for making the file available to other users. After discussing how to perform various functions on a file, let's now proceed to discuss the file system directories.

Working with Directories

File system directories refer to the storage places where thousands of files can be stored in a logical order. PHP provides you with a variety of functions to read and manipulate directories and directory entries. Some of the common directory manipulation functions are:

- Creating a directory
- Deleting a directory
- Locating the current working directory
- Changing a directory
- Listing files in a directory

Creating a Directory

A new directory can be created in PHP using the `mkdir()` function. The syntax of using the `mkdir()` function is:

```
mkdir(path/ directory_name, permission)
```

In the preceding syntax, the `path/ directory_name` parameter refers to the location where you want to create the directory. The permission parameter specifies the functions that can be performed on the directory, such as making the directory Read only or Read/ Write. Listing 6.52 show how to use the `mkdir()` function to create a directory. You can find listing 6.52 in \Code\PHP\Chapter 06\Using the mkdir Function folder on the CD.

Listing 6.52: Showing the Use of the `mkdir()` Function

```
<?php
  $result=mkdir("C:\\test", "0777") or die("Unable to create directory");
  echo "Directory created";
?>
```

The code, shown in Listing 6.52, creates a directory in the C drive of the file system with all permissions (0777 mode). Now, let's see how to delete a directory.

Deleting a Directory

An existing directory can be deleted in PHP using the `rmdir()` function. The syntax of using the `rmdir()` function is:

```
rmdir(path/directory_name)
```

In the preceding syntax, the path/ directory_name parameter refers to the directory, which is to be deleted. The delete operation succeeds only if the directory is empty. If the directory contains files or other subdirectories, the deletion cannot be performed until those files and subdirectories are deleted.

Listing 6.53 shows how to use the `rmdir()` function to delete a directory:

Listing 6.53: Showing the Use of the `rmdir()` Function

```
<?php
  $result=rmdir("C:\\test") or die("Unable to delete directory");
  echo "Directory deleted";
?>
```

The code, shown in Listing 6.53, deletes the directory from the C drive, if the directory is empty.

Locating the Current Working Directory

Now, let's see how to locate the directory in which you are currently working. The current working directory can be located using the `getCwd()` function. This function takes no parameters. Listing 6.54 shows how to use the `getCwd()` function to locate the current working directory. You can find listing 6.54 in \Code\PHP\Chapter 06\Using the getCwd Function folder on the CD.

Listing 6.54: Showing the Use of the `getCwd()` Function

```
<?php
  $current_dir = getCwd();
  echo "Current directory is $current_dir";
?>
```

The code, shown in Listing 6.54, displays the current working directory.

Changing a Directory

PHP allows you to change the directory in which you are currently working. The `chdir()` function is used to change the current working directory. The syntax of the `chdir()` function is:

```
chdir(directory_path)
```

Listing 6.55 shows how to use the `chdir()` function to change the current working directory. You can find listing 6.55 in \Code\PHP\Chapter 06\Using the chdir Function folder on the CD.

Listing 6.55: Showing the Use of the `chdir()` Function

```
<?php
  $current_dir = getCwd();
  echo "Current directory is $current_dir <br>";
  chdir ("C:\\test");
  $current_dir = getCwd();
  echo "Current directory is now $current_dir <br>";
?>
```

The code, shown in Listing 6.55, first prints the name of the current directory, and then changes the current working directory using the `chdir()` function. The name of the new current working directory is printed in the next statement.

Listing Files in a Directory

PHP provides various functions to read the contents of a directory. The following functions are most commonly used to read the directory content:

- The `readdir()` function
- The `scandir()` function

Let's discuss each of them in detail.

The readdir() Function

The syntax of the `readdir()` function is:

```
readdir(dir_stream)
```

In the preceding syntax, the `dir_stream` parameter specifies the handle of the directory. The `readdir()` function returns an entry from a directory handle, opened by the `opendir()` function. Listing 6.56 shows how to use the `readdir()` function to read the content of a directory. You can find listing 6.56 in \Code\PHP\Chapter 06\Using the readdir Function folder on the CD.

Listing 6.56: Showing the Use of the `readdir()` Function

```
<?php
$dir = opendir("C:\\test");//Open images directory
while (($file = readdir($dir))) //List files in images directory {
  echo "filename: " . $file . "<br />";
}
closedir($dir);
?>
```

In Listing 6.56, first, the `opendir()` function is used to open a directory, and then the content of the directory is printed using the `readdir()` function. The directory is then closed using the `closedir()` function.

The scandir() Function

Another function that can be used to read the contents of a directory is the `scandir()` function. The syntax of the `scandir()` function is:

```
scandir(directory,sort,context)
```

In the preceding syntax, the directory parameter refers to the name of the directory whose content you want to read. The sort parameter is used to sort the file names in either ascending or descending order. The context parameter is used to define the behavior of the streams.

Listing 6.57 shows how to use the `scandir()` function to read the content of a directory:

Listing 6.57: Showing the Use of the `scandir()` Function

```
<?php
  $file=scandir("C:\\test");
  print_r($file);
?>
```

The code, shown in Listing 6.57, displays the content of the directory. Note that a new function, `print_r ()`, is used to print the contents. This function displays the information about the $file variable that, in this case, is acting as a handle for the `scandir()` function.

Let's summarize the key points learned in this chapter.

Summary

You have learned to create and invoke user-defined functions, return a value from the function and the scope of a variable. You have learned to use the built-in function, string manipulation functions, mathematical functions, and date/time functions. In addition you have also learned recursive, variable, and callback functions. Next, you have learned to create an array, access the elements of an array, types of array. You have learned to traverse the array using the foreach loop and the ArrayIterator class. Finally, you have learned to use the built-in array functions. In this chapter, you have learned about various functions, such as fgets(), readfile(), and readdir(), provided by PHP to work with files and directories. These functions are used to perform different types of operations on files and directories. All these functions and operations are explained in the chapter, in detail, with the help of listings. You also learned about the flock () function and various lock modes used to lock a file.

In this chapter, you have learned about various functions, such as fgets(), readfile(), and readdir(), provided by PHP to work with files and directories. These functions are used to perform different types of operations on files and directories. All these functions and operations are explained in the chapter, in detail, with the help of listings. You also learned about the flock () function and various lock modes used to lock a file.

In the next chapter, we will discuss about forms and database in PHP.

Quick Revise

Q1. **True/False:**

a. Reusability is the primary advantage of functions. **True**

b. A function is declared using keyword function. **True**

c. PHP function names are case-sensitive. **False**

d. The exp() function returns the value of $e^x - 1$. **False**

e. The octdec() function converts a decimal number to an octal number. **False**

f. The sqrt() function returns the squre root of a number. **True**

g. Recursive function is a function that calls itself repeatedly. **True**

h. An array is a special variable, which can hold more than one value, at a time. **True**

i. An array() construct is used in PHP to create an array. **True**

j. In an associative array a key is associated with an integer. **False**

Q2. **What is the correct way to create a function in PHP?**

a. new_function myfunction()

b. function myFunction()

c, Create myfunction()

d. make myfunction()

Ans: b

Q3. **Given a comma separated list of values in a string, which function from the given list can create an array of each individual value with a single call?**

a. strstr()

b. extract()

c. explode()

d. strtok()

Ans: c

Q4. **What is the all purpose way of compairing two strings?**

a. Using strops()

b. Using the == operator

c. Using strcasecmp()

d. Using strcmp()

Ans: d

Q5. **Which of the following function do not return a timestamp?**

a. time()

b. date()

c. localtime()

d. strtotime()

Ans: b

Q6. _____Checks a date for numeric validity.

a. Check_date()
b. Verifydate()
c. Verify_date()
d. Checkdate()

Ans: a

Q7 The_____ function parses an English textual date or time into a Unix timestamp.

a. strtodate()
b. stroftime()
c. strtotime()
d. str_to_time()

Ans: c

Q8. What function is used to return the value of a number rounded upwards to the nearest integer?

a. ceil()
b. floor()
c. round()
d. fmode()

Ans: a

Q9. There are three different kind of arrays?

a. Numeric Array, String Array, Multidimensional Array
b. Numeric Array, Associative Array, Dimensional Array
c. Numeric Array, Associative Array, Multidimensional Array
d. Const Array, Associative Array, Numeric Array

Ans: c

Q10. Array values are keyed by ____values (called numeric arrays) or using _____values (called associative arrays).Of course, these key methods can be combined as well.

a. Float, string
b. Positive number, negative number
c. String, Boolean
d. Integer, string

Ans: d

Q11. What function counts the elements in an array?

a. Count()
b. count(), sizeof()
c. array_count()
d. count_array()

Ans: b

Q12. Write a PHP script to reverse a string 'kogent'?

Ans:

```
<?php
$string="kogent";
echo "The reversed string is:".strrev($string);
?>
```

Q13. Write the Output of following PHP script?

```
<?php
$string="kogent solutions";
echo "The position of so in kogent solution is:".strpos($string , "so");
?>
```

Ans: The position of **so** in kogent solution is: 7

Q14 Write a PHP script to print an Array element?

Ans:

```
<?php
$cars = array('Lio', 'Spark', 'Scorpio', 'Safari');
print_r( $cars );
?>
```

Q15. What are numeric arrays define with one example?

Ans: Numeric arrays use integer values as their index number to identify each item of the array.

Example:

```
<?php
$colors = array('white','black','blue');
print_r($colors);
?>
```

Output: Array([0] => white [1] => black [2] => blue [3])

The output of the preceding example shows the index numbers for white, black and blue are 0, 1, 2 respectively, which are numeric values. Hence we call such arrays numeric arrays.

Q16. If the fopen() function is unable to open a file it returns

a. True
b. 0
c. Error
d. Depends on the Operating System

Ans: b

Q17. Which of the following modes is used to write at the end of the file?

a. r
b. r+
c. a+
d. w+

Ans. c

Q18. Which of the following functions fetches a file in the form of array?

a. fgets()
b. file_get_contents()
c. file()
d. fgetc()

Ans: c

Q19. Which of the following functions read a file line by line?

a. fgets()
b. readfile()
c. fgetc()
d. fputs()

Ans: a

Q20. Which of the following function return a Boolean value?

a. is_executable()
b. is_writable()
c. flock()
d. All of these

Ans: d

Q21. Which of the following modes is used with the flock() function to unlock a file?

a. LOCK_NB
b. LOCK_UN
c. LOCK_SH
d. LOCK_EX

Ans. b

Q22. **Which of the following functions is used to determine the current working directory?**

a. getcwd() **b. is_cwd()**

c. get_cwd() **d. get_curr_wd()**

Ans: a

Q23. **The fgets() function reads _____ bytes of data by default**

a. 512 **b. 256**

c. 1024 **d. 64**

Ans: c

Q24. **Which of the following functions is used to check whether a file is a link or not**

a. is_hyperlink() **b. is_link()**

c. is_file_link() **d. none of these**

Ans: b

Q25. **Which of the following functions is used to change the current working directory?**

a. chdir() **b. chngdir()**

c. change_dir() **d. chcwd()**

Ans: a

Q26. **Which of the following statements are true?**

a. The "w" mode of opening a file allows user to write as well as read.

b. The feof() functions checks for the end of file.

c. The file_get_contents() function returns file in the form of string.

d. A directory containing sub-directories can be deleted.

e. The path of the file to be opened or read can be given in single quotes.

f. LOCK_NB when used in flock() function allows other processes to access the file.

g. feof() function does not return Boolean value.

h. The fgets() function is alias of fread() function.

i. The fputs() function is an alias of the fwrite() function.

Ans: b, c, f, i

Q27. **Explain the fopen() function? Also, list the various modes of opening a file**

Ans: The fopen() function is used to open a file. If it is unable to open file it returns 0. The syntax of the fopen() function is as follows:

```
fopen(string filename, string mode)
```

The filename parameter refers to the name of the file to be opened. The mode parameter refers to mode in which the file is opened.

Modes	Description
r	Refers to the Read only mode. It starts at the beginning of the file.
r+	Refers to the Read/Write mode. It starts at the beginning of the file.
w	Refers to the Write only mode. It allows the user to clear the content and write in the file.
w+	Refers to the Read/Write mode. It allows the user to read as well as write in the file. Similar to the Write only mode, this mode also clears the content of the file.
a	Refers to the Append mode. It adds content at the end of an existing file. It also creates a new file to add content, if no file already exists.

Modes	Description
a+	Refers to the Read/Append mode. It adds content at the end of an existing file.
x	Refers to the Write only mode. It creates a new file to write content in it. This mode works on a new file and returns FALSE and an error, if a file already exists.
x+	Refers to the Read/Write mode. It creates a new file to read and write content in it. This mode works on a new file and returns FALSE and an error if a file already exists.

Q28. List some of the functions in PHP used to read a file. Also write their syntax

Ans: The functions used to read a file in PHP are as follows:

- The fgets() function
- The fread() function
- The file() function
- The file_get_contents() function
- The readfile() function
- The fgetc() function

Syntax of the fgets() function is as follows:

```
fgets (handle, length)
```

The handle parameter refers to the variable that holds the reference of the file and *length* refers to the size of data that is extracted at a time. The length parameter is not mandatory to use.

Syntax of the fread() function is as follows:

```
fread(handle, length)
```

The *handle* parameter refers to the variable that holds the reference of the file and *length* refers to the size of data that is extracted at a time. The length parameter is not mandatory to use.

Syntax of the file() function is as follows:

```
file(filename)
```

The filename is a parameter that refers to the name of the file.

Syntax of the file_get_contents() function is as follows:

```
file_get_contents(filename)
```

The filename parameter refers to the name of the file that you want to read.

Syntax of the readfile() function is as follows:

```
readfile(filename)
```

The filename parameter refers to the name of the file that you want to read.

Syntax of the fgetc() function is as follows:

```
fgetc(filename)
```

The filename parameter refers to the name of the file that you want to read.

Q29. Explain the fwrite() function.

Ans: The fwrite() function is used to write content in an already opened file. This function, if successful, returns the number of bytes that are written in a file; otherwise returns a FALSE value. The syntax to write in a file using the fwrite() function is:

```
fwrite(filename/handle,string,length)
```

In the preceding syntax, the *filename/handle* parameter refers to the name of the file in which you need to write content. The second parameter, *string*, specifies the content, in the form of a string, which you want to write in the file. The next parameter, *length* (optional), specifies the maximum number of bytes to write to the file.