Chapter 5: PHP More Details[1]

Objectives:

After reading this chapter you should be entertained and learned:

1- **PHP Statements.**
2- **Variable Types in PHP.**
3- **Operators in PHP.**
4- **PHP Control Structures.**
5- **PHP Programs as Examples.**

5.1 Introduction:

In section 4.7, we started an application to pass parameters between a form and a program. The main purpose of that step was to train you on how to save those parameters into memory then retrieve them. Let us develop that step to save the values of those parameters on a file. The rest of this chapter is to demonstrate PHP statements in details.

5.1.1 The Order Form:

Figure 5.1 shows the source code of the input form. It has only one difference comparing with the program in figure 3.3. This difference is the name of the called program (line#2). The name of that program is process order file. There is no any difference in other statements.

```
 1  <html>
 2  <form action=processorderfile.php method=post>
 3  <table border=0>
 4  <tr bgcolor=#cccccc>
 5  <td width=150>Item</td>
 6  <td width=15>Quantity</td>
 7  <tr/>
 8  <tr>
 9  <td>Tires</td>
10  <td align=center>
11   <input type='text' name='tireqty' size=3 maxlength=3></td>
12  <tr/>
13  <tr>
14  <td>Oil</td>
15  <td align=center>
16  <input type='text' name='oilqty' size=3 maxlength=3></td>
17  <tr/>
18  <tr>
19  <td>Spark Plugs</td>
20  <td align=center>
21  <input type='text' name='sparkqty' size=3 maxlength=3></td>
22  <tr/>
23  <tr>
24  <td   colspan=2 align=right>
25  <input type=submit value='Submit'></td>
26  <tr/>
27  <table/>
28  <form/>
29  <html/>
```

Figure 5.1 Listing of the Form

[1] . . . This chapter was re-tailored from Luke Welling Book.

5.1.2 Saving Data for Later Use:

There are basically two ways you can store data: in flat files or in a database. A flat file can have many formats but, in general, when we refer to a *flat file*, we mean a simple text file. In this example, we'll write customer orders to a text file, one order per line. This is very simple to do, but also pretty limiting, as we'll see later in this chapter. If you're dealing with information of any reasonable volume, you'll probably want to use a database instead. However, flat files have their uses and there are some situations when you'll need to know how to use them. Writing to and reading from files in PHP is virtually identical to the way it's done in C. If you've done any C programming or UNIX shell scripting, this will all seem pretty familiar to you.

As you see, it is the same code as the program in figure 5.2. The only difference is the name of the file to be called when clicking the Submit button. (In this case the file name is processorderfile.php. Its listing is:

```
 1  <h1>Hesham's Auto Parts</h1>
 2  <h2>Order Results</h2>
 3  <?
 4  $tireqty=$_POST['tireqty'];
 5  $oilqty=$_POST['oilqty'];
 6  $sparkqty=$_POST['sparkqty'];
 7  print "<table border=10>
 8         <tr><th>Quantity</th><th>Type</th>
 9         <tr><td>$tireqty</td><td>tires</td>
10         <tr><td>$oilqty</td><td>oil can</td>
11         <tr><td>$sparkqty</td><td>sparks</td>
12         </table>";
13      $fp=fopen("order.txt","a");
14      $outst = $tireqty."\t".$oilqty."\t".$sparkqty."\n";
15      fwrite($fp, $outst);
16      fclose($fp);
17      print "A record has been written successfuly<br>
18             <a href=aab.php>New order</a>";
19  ?>
```

Figure 5.2 the processorderfile.php listing

The lines from #1 to #12 are the same as the program in figure 3.5. It gives the same output and then writes a record in the file order.txt.

Line #13 `$fp=fopen("order.txt","a");` opens the file order.txt. The type of open is "append".

Line #14 `$outst = $tireqty."\t".$oilqty."\t".$sparkqty."\n";` prepares the buffer to be written. It assigns the values of the variables `$tireqty, $oilqty, and $sparkqty after adding the tab ("\t") between the variables, then ends with the newline ("\n").`

Line #15 `fwrite($fp, $outst); It writes the record on the file.`

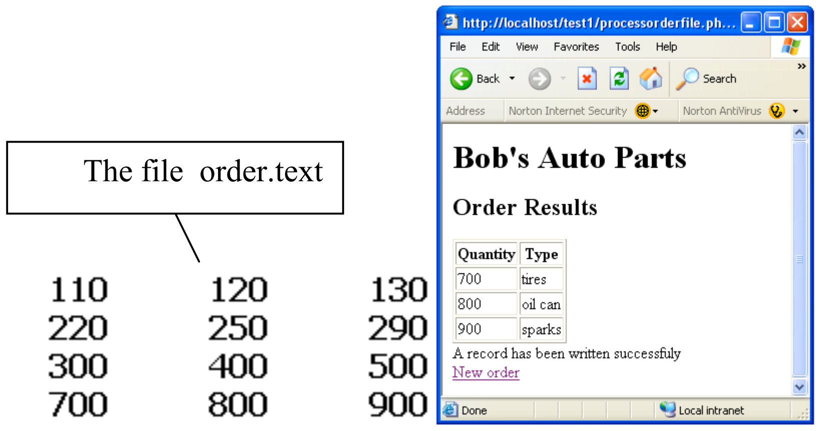
Line #16 `fclose($fp);` It closes the file.

Line #17 `print "A record has been written successfuly<br>` displays a message.

Line #18 `<a href=aab.php>New order</a>";` a link to get new record.

Line #19 `?>` PHP tag closing.

Figure 5.3 shows the output of the processoderfile.php program.


The file order.text
http://localhost/test1/processorderfile.ph...
File Edit View Favorites Tools Help
Back Search
Address Norton Internet Security Norton AntiVirus
Done
Local intranet

```
110    120    130
220    250    290
300    400    500
700    800    900
```

Bob's Auto Parts

Order Results

Quantity	Type
700	tires
800	oil can
900	sparks

A record has been written successfuly
New order

Figure 5.3 the outputs of the processorderfile.php

5.1.3 Printing the content of a file:

Figure 5.4 shows the listing of the printfile.php program.

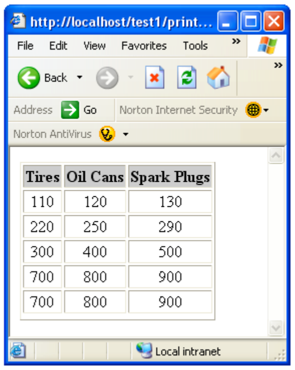
```
 1  <?
 2      $fp=fopen("order.txt","r");
 3
 4      print "<table border=1>
 5             <tr bgcolor=#cccccc><th>Tires</th>
 6                      <th>Oil Cans</th>
 7                      <th>Spark Plugs</th>";
 8
 9      while (!feof($fp))
10      {
11            $rec=fgets($fp,20);
12            $tireqty=substr($rec, 0,3);
13            $oilqty=substr($rec, 4,3);
14            $sparkqty=substr($rec, 8,3);
15            print "<tr><td align=center>$tireqty</td>
16                       <td align=center>$oilqty</td>
17                       <td align=center>$sparkqty</td>";
18       }
19
20       print "</table>";
21       fclose($fp);
22  ?>
```

Figure 5.4 printorder.php listing

`Line #2  $fp=fopen("order.txt","r");`	opens the data file order.txt. The type of open is "read".
`Line #4  thru Line #7`	print the table header.
`Line #9  while (!feof($fp))`	repeats the block from Line #10 to Line #18 till the end of file.
`!feof`	means not the end of data file (opened in line #2).
`Line #11  $rec=fgets($fp,20);`	gets a record from the data file (either of length 20 / till newline).
`Line #12, Line #13, Line #14`	get the value as a substring of index as first number and of length of the second number … index starts from zero for the first variable.
`Line #20  print "</table>";`	closes the table".

`Line #21  fclose($fp);`   closes the data file”.

Figure 5.5 shows the output of the printfile.php program.



Tires	Oil Cans	Spark Plugs
110	120	130
220	250	290
300	400	500
700	800	900
700	800	900

Figure 5.5 printorder.php output

5.2 PHP Statements:

We tell the PHP interpreter what to do by having PHP statements between our opening and closing tags. In this example, we used only one type of statement:

```
echo “<p>Order processed.”;
```

You will notice that a semicolon appears at the end of the echo statement. This is used to separate statements in PHP much like a period is used to separate sentences in English. If you have programmed in C or Java before, you will be familiar with using the semicolon in this way. Leaving the semicolon off is a common syntax error that is easily made. However, it’s equally easy to find and to correct.

5.2.1 Whitespace:

Spacing characters such as new lines (carriage returns), spaces and tabs are known as whitespace. I would combine the paragraph above and the one below and form one cohesive paragraph explaining how spacing characters (whitespace) is ignored in PHP and HTML. As you probably already know, browsers ignore whitespace in HTML. So does the PHP
engine. Consider these two HTML fragments:

```
<h1>Welcome to Hesham’s Auto Parts!</h1>
<p>What would you like to order today?   and
<h1>Welcome to Hesham’s
Auto Parts!</h1>
<p>What would you like to order today?
```

These two snippets of HTML code produce identical output because they appear the same to the browser. However, you can and are encouraged to use whitespace in your HTML as an aid to humans—to enhance the readability of your HTML code. The same is true for PHP. There is no need to have any whitespace between PHP statements, but it makes the code easier to read if we put each statement on a separate line. For example,

```
echo “hello”;
echo “world”;        and
echo “hello”;echo “world”;
```

are equivalent, but the first version is easier to read.

5.2.2 Comments:

Comments are used for documentation purpose. The PHP interpreter will ignore any text in a comment. Essentially the PHP parser skips over the comments that are equivalent to whitespace. PHP supports C, C++, and shell script style comments. This is a C-style, multi-line comment that might appear at the start of our PHP script:

```
/* Author: Hesham Elmahdy
Last modified: August 26th.
This script processes the customer orders.
*/
```

Multi-line comments should begin with a `/*` and end with `*/`. As in C, multi-line comments cannot be nested. You can also use single line comments, either in the C++ style:

```
echo "<p>Order processed."; // Start printing order
```

or in the shell script style:

```
echo "<p>Order processed."; # Start printing order
```

With both of these styles, everything after the comment symbol (# or //) is a comment until we reach the end of the line or the ending PHP tag, whichever comes first.

5.2.3 Adding Dynamic Content:

So far, we haven't used PHP to do anything we couldn't have done with plain HTML. The main reason for using a server-side scripting language is to be able to provide dynamic content to a site's users. This is an important application because content that changes according to a user's needs or over time will keep visitors coming back to a site. PHP allows us to do this easily. Let's start with a simple example. Replace the PHP in processorder.php with the following code:

```
<?
echo "<p>Order processed at ";
echo date("H:i, jS F");
echo "<br>";
?>
```

5.2.3 Calling Functions:

Look at the call to date(). This is the general form that function calls take. PHP has an extensive library of functions you can use when developing Web applications. Most of these functions need to have some data passed to them and return some data. Look at the function call:

```
date("H:i, jS F")
```

Notice that we are passing a string (text data) to the function inside a pair of parentheses. This is called the function's argument or parameter. These arguments are the input used by the function to output some specific results.

5.2.4 The date() Function

The date() function expects the argument you pass it to be a format string, representing the style of output you would like. Each of the letters in the string represents one part of the date and time. H is the hour in a twenty-hour hour format, i is the minutes with a leading zero where required, j is the day of the month without a leading zero, S represents the ordinal suffix (in this case “th”), and F is the year in four digit format.

5.2.5 Accessing Form Variables:

The whole point of using the order form is to collect the customer order. Getting the details of what the customer typed in is very easy in PHP. Within your PHP script, you can access each of the form fields as a variable with the same name as the form field.

5.2.6 Form Variables:

The data from the script will end up in PHP variables. You can recognize variable names in PHP because they all start with a dollar sign ($). (Forgetting the dollar sign is a common programming error.)

5.2.7 String Concatenation:

In the script, we used echo to print the value the user typed in each of the form fields, followed by some explanatory text. If you look closely at the echo statements, you will see that the variable name and following text have a period (.) between them, such as this:

```
echo $tireqty.” tires<br>”;
```

This is the string concatenation operator and is used to add strings (pieces of text) together. You will often use it when sending output to the browser with echo. This is used to avoid having to write multiple echo commands. You could alternatively write

```
echo “$tireqty tires<br>”;
```

This is equivalent to the first statement. Either format is valid, and which one you use is a matter of personal taste.

5.2.8 Variables and Literals:

The variable and string we concatenate together in each of the echo statements are different types of things. Variables are a symbol for data. The strings are data themselves. When we use a piece of raw data in a program like this, we call it a literal to distinguish it from a variable. $tireqty is a variable, a symbol which represents the data the customer typed in. On the other hand, “ tres” is a literal. It can be taken at face value.

Well, almost. Remember the second example previously? PHP replaced the variable name $tireqty in the string with the value stored in the variable. There are actually two kinds of strings in PHP—ones with double quotes and ones with single quotes. PHP will try and evaluate strings in double quotes, resulting in the behavior we saw earlier. Single-quoted strings will be treated as true literals.

5.2.9 Identifiers:

Identifiers are the names of variables. (The names of functions and classes are also identifiers.) There are some simple rules about identifiers:

- Identifiers can be of any length and can consist of letters, numbers, underscores, and dollar signs. However, you should be careful when using dollar signs in identifiers. You’ll see why in the section called, “Variable Variables.”
- Identifiers cannot begin with a digit.
- In PHP, identifiers are case sensitive. $tireqty is not the same as $TireQty. Trying to use these interchangeably is a common programming error. PHP’s built-in functions are an exception to this rule—their names can be used in any case.

- Identifiers for variables can have the same name as a built-in function. This is confusing, however, and should be avoided. Also, you cannot create a function with the same identifier as a built-in function.

5.2.10 User-Declared Variables:

You can declare and use your own variables in addition to the variables you are passed from the HTML form. One of the features of PHP is that it does not require you to declare variables before using them. A variable will be created when you first assign a value to it—see the next section for details.

5.2.11 Assigning Values to Variables:

You assign values to variables using the assignment operator, =. On Bob's site, we want to work out the total number of items ordered and the total amount payable. We can create two variables to store these numbers. To begin with, we'll initialize each of these variables to zero. Add these lines to the bottom of your PHP script:

```
$totalqty = 0;
$totalamount = 0.00;
```

Each of these two lines creates a variable and assigns a literal value to it. You can also assign variable values to variables, for example:

```
$totalqty = 0;
$totalamount = $totalqty;
```

5.3 Variable Types:

A variable's type refers to the kind of data that is stored in it.

5.3.1 PHP's Data Types:

PHP supports the following data types:

- Integer—Used for whole numbers
- Double—Used for real numbers
- String—Used for strings of characters
- Array—Used to store multiple data items of the same type (see Chapter 3, "Using
- Arrays")
- Object—Used for storing instances of classes (see Chapter 6, "Object Oriented PHP")
- PHP also supports the pdfdoc and pdfinfo types if it has been installed with PDF (Portable Document Format) support. We will discuss using PDF in PHP in Chapter 29, "Generating Personalized Documents in Portable Document Format."

5.3.2 Type Strength:

PHP is a very weakly typed language. In most programming languages, variables can only hold one type of data, and that type must be declared before the variable can be used, as in C. In PHP, the type of a variable is determined by the value assigned to it. For example, when we created $totalqty and

```
$totalamount, their initial types were determined, as follows:
$totalqty = 0;
$totalamount = 0.00;
```

Because we assigned 0, an integer, to $totalqty, this is now an integer type variable. Similarly, $totalamount is now of type double.

Strangely enough, we could now add a line to our script as follows:

```
   $totalamount = “Hello”;
```

The variable $totalamount would then be of type string. PHP changes the variable type according to what is stored in it at any given time.

This ability to change types transparently on-the-fly can be extremely useful. Remember PHP “automagically” knows what data type you put into your variable. It will return the data with the same data type once you retrieve it from the variable.

5.3.3Type Casting:

You can pretend that a variable or value is of a different type by using a type cast. These work identically to the way they work in C. You simply put the temporary type in brackets in front of the variable you want to cast.

For example, we could have declared the two variables above using a cast.

```
$totalqty = 0;
$totalamount = (double)$totalqty;
```

The second line means “Take the value stored in $totalqty, interpret it as a double, and store it in $totalamount.” The $totalamount variable will be of type double. The cast variable does not change types, so $totalqty remains of type integer.

5.3.4 Variable Variables:

PHP provides one other type of variable—the variable variable. Variable variables enable us to change the name of a variable dynamically. (As you can see, PHP allows a lot of freedom in this area—all languages will let you change the value of a variable, but not many will allow you to change the variable’s type, and even fewer will let you change the variable’s name.) The way these work is to use the value of one variable as the name of another. For example, we could set

```
$varname = “tireqty”;
```

We can then use $$varname in place of $tireqty. For example, we can set the value of

```
$tireqty:
$$varname = 5;
This is exactly equivalent to
$tireqty = 5;
```

This might seem a little obscure, but we’ll revisit its use later. Instead of having to list and use each form variable separately, we can use a loop and a variable to process them all automatically. There’s an example illustrating this in the section on for loops.

5.3.5 Constants:

As you saw previously, we can change the value stored in a variable. We can also declare constants. A constant stores a value such as a variable, but its value is set once and then cannot be changed elsewhere in the script. In our sample application, we might store the prices for each of the items on sale as constants. You can define these constants using the define function:

```
define(“TIREPRICE”, 100);
define(“OILPRICE”, 10);
define(“SPARKPRICE”, 4);
```

Add these lines of code to your script.

You will notice that the names of the constants are all in uppercase. This is a convention borrowed from C that makes it easy to distinguish between variables and constants at a glance. This convention is not required but will make your code easier to read and maintain.

We now have three constants that can be used to calculate the total of the customer's order. One important difference between constants and variables is that when you refer to a constant, it does not have a dollar sign in front of it. If you want to use the value of a constant, use its name only. For example, to use one of the constants we have just created, we could type:

```
echo TIREPRICE;
```

As well as the constants you define, PHP sets a large number of its own. An easy way to get an overview of these is to run the phpinfo() command:

```
phpinfo();
```

This will provide a list of PHP's predefined variables and constants, among other useful information. We will discuss some of these as we go along.

5.3.6 Variable Scope:

The term *scope* refers to the places within a script where a particular variable is visible. The three basic types of scope in PHP are as follows:

- Global variables declared in a script are visible throughout that script, but *not inside functions*.
- Variables used inside functions are local to the function.
- Variables used inside functions that are declared as global refer to the global variable of the same name.

We will cover scope in more detail when we discuss functions. For the time being, all the variables we use will be global by default.

5.4 Operators:

Operators are symbols that you can use to manipulate values and variables by performing an operation on them. We'll need to use some of these operators to work out the totals and tax on the customer's order. We've already mentioned two operators: the assignment operator, =, and ., the string concatenation operator. Now we'll look at the complete list.

In general, operators can take one, two, or three arguments, with the majority taking two. For example, the assignment operator takes two—the storage location on the left-hand side of the = symbol, and an expression on the right-hand side. These arguments are called *operands*, that is, the things that are being operated upon.

5.5.1 Arithmetic Operators:

Arithmetic operators are very straightforward—they are just the normal mathematical operators. The arithmetic operators are shown in Table 5.1.

TABLE 5.1 PHP's Arithmetic Operators

Operator	*Name*	*Example*
+	Addition	$a + $b
-	Subtraction	$a - $b
*	Multiplication	$a * $b
/	Division	$a / $b
%	Modulus	$a % $b

Addition and subtraction work as you would expect. The result of these operators is to add or subtract, respectively, the values stored in the $a and $b variables. You can also use the subtraction symbol, -, as a unary operator (that is, an operator that takes one argument or operand) to indicate negative numbers. For example

```
$a = -1;
```

Multiplication and division also work much as you would expect. Note the use of the asterisk as the multiplication operator, rather than the regular multiplication symbol, and the forward slash as the division operator, rather than the regular division symbol. The modulus operator returns the remainder of dividing the $a variable by the $b variable. Consider this code fragment:

```
$a = 27;
$b = 10;
$result = $a%$b;
```

The value stored in the $result variable is the remainder when we divide 27 by 10; that is, 7. You should note that arithmetic operators are usually applied to integers or doubles. If you apply them to strings, PHP will try and convert the string to a number. If it contains an "e" or an "E", it will be converted to a double; otherwise it will be converted to an int. PHP will look for digits at the start of the string and use those as the value—if there are none, the value of the string will be zero.

5.5.2 String Operators:

We've already seen and used the only string operator. You can use the string concatenation operator to add two strings and to generate and store a result much as you would use the addition operator to add two numbers.

```
$a = "Hesham's ";
$b = "Auto Parts";
$result = $a.$b;
```

The $result variable will now contain the string "Hesham's Auto Parts".

5.5.3 Assignment Operators:

We've already seen =, the basic assignment operator. Always refer to this as the assignment operator, and read it as "is set to." For example

```
$totalqty = 0;
```

This should be read as "$totalqty is set to zero". We'll talk about why when we discuss the comparison operators later in this chapter.

5.5.4 Returning Values from Assignment:

Using the assignment operator returns an overall value similar to other operators. If you write $ a + $b the value of this expression is the result of adding the $a and $b variables together. Similarly, you can write

```
$a = 0;
```

The value of this whole expression is zero. This enables you to do things such as

```
$b = 6 + ($a = 5);
```

This will set the value of the $b variable to 11. This is generally true of assignments: The value of the whole assignment statement is the value that is assigned to the left-hand operand. When working out the value of an expression, parentheses can be used to increase the precedence of a subexpression as we have done here. This works exactly the same way as in mathematics.

5.5.5 Combination Assignment Operators:

In addition to the simple assignment, there is a set of combined assignment operators. Each of these is a shorthand way of doing another operation on a variable and assigning the result back to that variable. For example

```
$a += 5;
```

This is equivalent to writing

```
$a = $a + 5;
```

Combined assignment operators exist for each of the arithmetic operators and for the string concatenation operator. A summary of all the combined assignment operators and their effects is shown in Table 5.2.

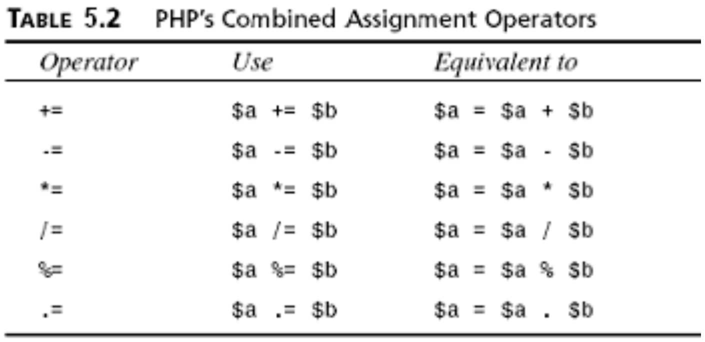


TABLE 5.2 PHP's Combined Assignment Operators

Operator	*Use*	*Equivalent to*
+=	$a += $b	$a = $a + $b
-=	$a -= $b	$a = $a - $b
*=	$a *= $b	$a = $a * $b
/=	$a /= $b	$a = $a / $b
%=	$a %= $b	$a = $a % $b
.=	$a .= $b	$a = $a . $b

5.5.6 Pre- and Post-Increment and Decrement:

The pre- and post- increment (++) and decrement (--) operators are similar to the += and -= operators, but with a couple of twists. All the increment operators have two effects—they increment and assign a value. Consider the following:

```
$a=4;
echo ++$a;
```

The second line uses the pre-increment operator, so called because the ++ appears before the $a. This has the effect of first, incrementing $a by 1, and second, returning the incremented value. In this case, $a is incremented to 5 and then the value 5 is returned and printed. The value of this whole expression is 5. (Notice that the actual value stored in $a is changed: We are not just returning $a + 1.) However, if the ++ is after the $a, we are using the post-increment operator. This has a different effect. Consider the following:

```
$a=4;
echo $a++;
```

In this case, the effects are reversed. That is, first, the value of $a is returned and printed, and second, it is incremented. The value of this whole expression is 5. This is the value that will be printed. However, the value of $a after this statement is executed is 5. As you can probably guess, the behavior is similar for the -- operator. However, the value of $a is decremented instead of being incremented.

5.5.7 References Operator:

A new addition in PHP 4 is the reference operator, & (ampersand), which can be used in conjunction with assignment. Normally when one variable is assigned to another, a copy is made of the first variable and stored elsewhere in memory. For example

```
$a = 5;
$b = $a;
```

These lines of code make a second copy of the value in $a and store it in $b. If we subsequently change the value of $a, $b will not change:

```
$a = 7; // $b will still be 5
```

You can avoid making a copy by using the reference operator, &. For example

```
$a = 5;
$b = &$a;
$a = 7; // $a and $b are now both 7
```

5.5.8 Comparison Operators:

The comparison operators are used to compare two values. Expressions using these operators return either of the logical values true or false depending on the result of the comparison.

5.5.9 The Equals Operator:

The equals comparison operator, == (two equal signs) enables you to test if two values are equal. For example, we might use the expression

```
$a == $b
```

to test if the values stored in $a and $b are the same. The result returned by this expression will be true if they are equal, or false if they are not. It is easy to confuse this with =, the assignment operator. This will work without giving an error, but generally will not give you the result you wanted. In general, non-zero values evaluate to true and zero values to false. Say that you have initialized two variables as follows:

```
$a = 5;
$b = 7;
```

If you then test $a = $b, the result will be true. Why? The value of $a = $b is the value assigned to the left-hand side, which in this case is 7. This is a non-zero value, so the expression evaluates to true. If you intended to test $a == $b, which evaluates to false, you have introduced a logic error in your code that can be extremely difficult to find. Always check your use of these two operators, and check that you have used the one you intended to use. This is an easy mistake to make, and you will probably make it many times in your programming career.

5.5.10 Other Comparison Operators:

PHP also supports a number of other comparison operators. A summary of all the comparison operators is shown in Table 5.3. One to note is the new identical operator, ===, introduced in PHP 4, which returns true only if the two operands are both equal and of the same type.

TABLE 5.3 PHP's Comparison Operators

Operator	*Name*	*Use*
`==`	equals	`$a == $b`
`===`	identical	`$a === $b`
`!=`	not equal	`$a != $b`
`<>`	not equal	`$a <> $b`
`<`	less than	`$a < $b`
`>`	greater than	`$a > $b`
`<=`	less than or equal to	`$a <= $b`
`>=`	greater than or equal to	`$a != $b`

5.5.11 Logical Operators:

The logical operators are used to combine the results of logical conditions. For example, we might be interested in a case where the value of a variable, $a, is between 0 and 100. We would need to test the conditions $a >= 0 and $a <= 100, using the AND operator, as follows $a >= 0 && $a <=100 PHP supports logical AND, OR, XOR (exclusive or), and NOT. The set of logical operators and their use is summarized in Table 5.5.

TABLE 5.4 PHP's Logical Operators

Operator	*Name*	*Use*	*Result*
`!`	NOT	`!$b`	Returns `true` if `$b` is `false`, and vice versa
`&&`	AND	`$a && $b`	Returns `true` if both `$a` and `$b` are `true`; otherwise `false`
`||`	OR	`$a || $b`	Returns `true` if either `$a` or `$b` or both are `true`; otherwise `false`
`and`	AND	`$a and $b`	Same as `&&`, but with lower precedence
`or`	OR	`$a or $b`	Same as `||`, but with lower precedence

The and and or operators have lower precedence than the && and || operators. We will cover precedence in more detail later in this chapter.

5.5.12 Bitwise Operators:

The bitwise operators enable you to treat an integer as the series of bits used to represent it. You probably will not find a lot of use for these in PHP, but a summary of bitwise operators is shown in Table 5.5.

TABLE 5.5 PHP's Bitwise Operators

Operator	*Name*	*Use*	*Result*
`&`	bitwise AND	`$a & $b`	Bits set in `$a` and `$b` are set in the result
`|`	bitwise OR	`$a | $b`	Bits set in `$a` or `$b` are set in the result
`~`	bitwise NOT	`~$a`	Bits set in `$a` are not set in the result, and vice versa
`^`	bitwise XOR	$a ^ $b	Bits set in `$a` or `$b` but not in both are set in the result
<<	left shift	$a << $b	Shifts `$a` left `$b` bits
>>	right shift	$a >> $b	Shifts `$a` right `$b` bits

5.6 Control Structures:

Control structures are the structures within a language that allow us to control the flow of execution through a program or script As mentioned in figure 4.5 (previous chapter), programs can be reduced to into three main categories: sequence, selection, and repetition.

5.6.1 Sequence:

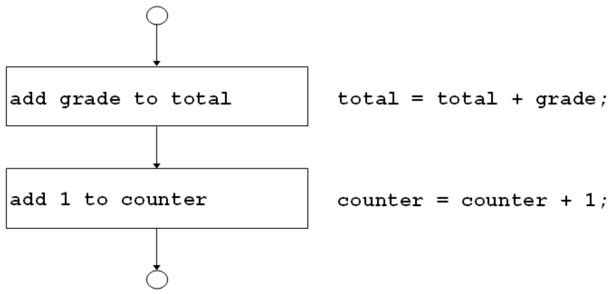


Figure 5.6 Sequence

5.6.2 Selection:

If we want to sensibly respond to our user's input, our code needs to be able to make decisions. The constructs that tell our program to make decisions are called *conditionals*.

5.6.2.1 if Statement:

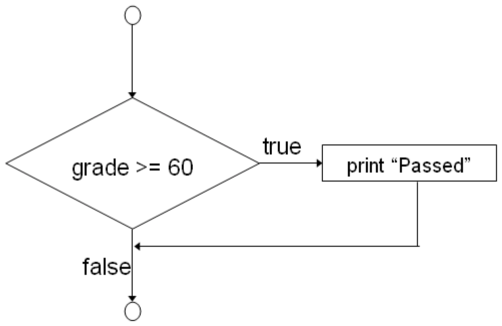


Figure 5.7 if Statement

We can use an if statement to make a decision. You should give the if statement a condition to use. If the condition is true, the following block of code will be executed. Conditions in "if statement" must be surrounded by brackets ().

For example, if we order no tires, no bottles of oil, and no spark plugs from Bob, it is probably because we accidentally pressed the Submit button. Rather than telling us "Order processed," the page could give us a more useful message. When the visitor orders no items, we might like to say, "You did not order anything on the previous page!" We can do this easily with the following if statement:

```
If ( $totalqty == 0 )
   {echo "You did not order anything on the previous page!<br>";}
```

The condition we are using is $totalqty == 0. Remember that the equals operator (==) behaves differently from the assignment operator (=). The condition $totalqty == 0 will be true if $totalqty is equal to zero. If $totalqty is not equal to zero, the condition will be false. When the condition is true, the echo statement will be executed.

5.6.2.1.1 Code Blocks:

Often we have more than one statement we want executed inside a conditional statement such as if. There is no need to place a new if statement before each. Instead, we can group a number of statements together as a block. To declare a block, enclose it in curly braces:

```
If ( $totalqty == 0 )
{
   echo "<font color=red>";
   echo "You did not order anything on the previous page!<br>";
   echo "</font>";
}
```

The three lines of code enclosed in curly braces are now a block of code. When the condition is true, all three lines will be executed. When the condition is false, all three lines will be ignored.

5.6.2.1.2 A Side Note: Indenting Your Code:

As already mentioned, PHP does not care how you lay out your code. You should indent your code for readability purposes. Indenting is generally used to enable us to see at a glance which lines will only be executed if conditions are met, which statements are grouped into blocks, and which statements are part of loops or functions. You can see in the previous examples that the statement which depends on the "if statement" and the statements which make up the block are indented.

5.6.2.1.3 else Statements:

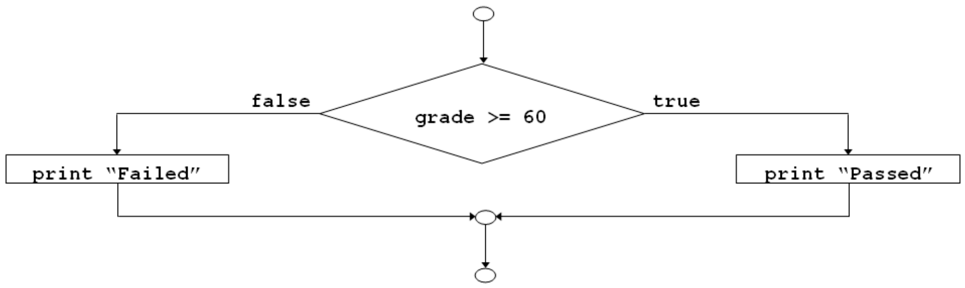


Figure 5.8 if – else Statement

You will often want to decide not only if you want an action performed, but also which of a set of possible actions you want performed. An else statement allows you to define an alternative action to be taken when the condition in an if statement is false. We want to warn Bob's customers when they do not order anything. On the other hand, if they do make an order, instead of a warning, we want to show them what they ordered. If we rearrange our code and add an else statement, we can display either a warning or a summary.

```
If ( $totalqty == 0 )
{
   echo "You did not order anything on the previous page!<br>";
}
else
{
   echo $tireqty." tires<br>";
   echo $oilqty." bottles of oil<br>";
   echo $sparkqty." spark plugs<br>";
}
```

We can build more complicated logical processes by nesting if statements within each other. In the following code, not only will the summary only be displayed if the condition $totalqty == 0 is true, but also each line in the summary will only be displayed if its own condition is met.

```
If ( $totalqty == 0)
{
   echo "You did not order anything on the previous page!<br>";
}
else
{
   if ( $tireqty>0 )
      echo $tireqty.” tires<br>”;
   if ( $oilqty>0 )
      echo $oilqty.” bottles of oil<br>”;
   if ( $sparkqty>0 )
      echo $sparkqty.” spark plugs<br>”;
}
```

5.6.2.1.4 elseif Statements:

For many of the decisions we make, there are more than two options. We can create a sequence of many options using the elseif statement. The elseif statement is a combination of an else and an if statement. By providing a sequence of conditions, the program can check each until it finds one that is true. Bob provides a discount for large orders of tires. The discount scheme works like this:

- Less than 10 tires purchased—no discount
- 10-49 tires purchased—5% discount
- 50-99 tires purchased—10% discount
- 100 or more tires purchased—15% discount

We can create code to calculate the discount using conditions and if and elseif statements. We need to use the AND operator (&&) to combine two conditions into one.

```
If ( $tireqty < 10 )
   $discount = 0;
elseif ( $tireqty >= 10 && $tireqty <= 49 )
       $discount = 5;
       elseif ( $tireqty >= 50 && $tireqty <= 99 )
              $discount = 10;
              elseif ( $tireqty > 100 )
                     $discount = 15;
```

Note that you are free to type elseif or else if—with and without a space are both correct. If you are going to write a cascading set of elseif statements, you should be aware that only one of the blocks or statements will be executed. It did not matter in this example because all the conditions were mutually exclusive—only one can be true at a time. If we wrote our conditions in a way that more than one could be true at the same time, only the block or statement following the first true condition would be executed.

5.6.2.2 switch Statements:

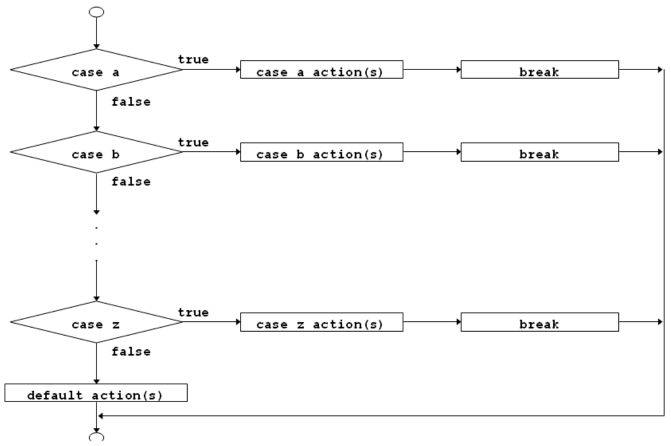
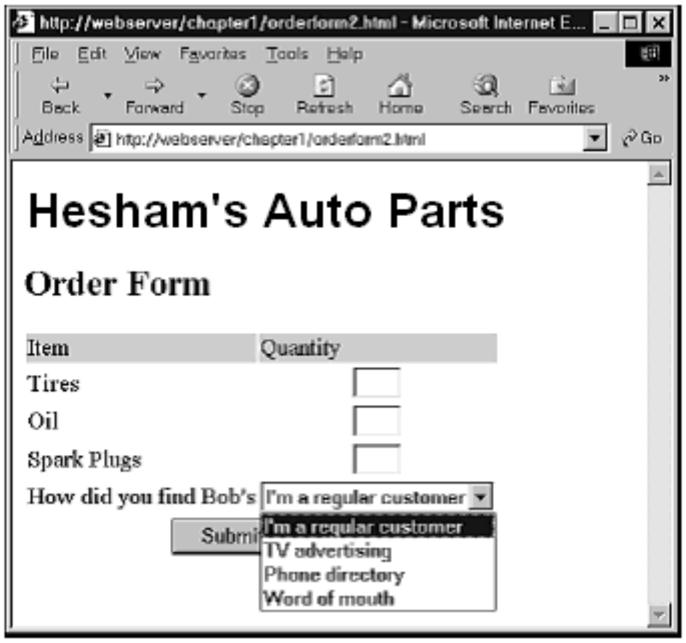


Figure 5.8 switch Statement

The switch statement works in a similar way to the if statement, but allows the condition to take more than two values. In an if statement, the condition can be either true or false. In a switch statement, the condition can take any number of different values, as long as it evaluates to a simple type (integer, string, or double). You need to provide a case statement to handle each value you want to react to and, optionally, a default case to handle any that you do not provide a specific case statement for. He wants to know what forms of advertising are working for him. We can add a question to our order form.

```
<tr>
<td>How did you find Hesham’s</td>
<td><select name=”find”>
<option value = “a”>I’m a regular customer
<option value = “b”>TV advertising
<option value = “c”>Phone directory
<option value = “d”>Word of mouth
</select>
</td>
</tr>
```


http://webserver/chapter1/orderform2.html - Microsoft Internet E...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://webserver/chapter1/orderform2.html
Go
Submit

Hesham's Auto Parts

Order Form

Item	Quantity
Tires	
Oil	
Spark Plugs	

How did you find Bob's: I'm a regular customer

I'm a regular customer
TV advertising
Phone directory
Word of mouth

Figure 5.9 combo box

This HTML code has added a new form variable whose value will be “a”, “b”, “c”, or “d”. We could handle this new variable with a series of if and elseif statements like this:

```
if ($find == “a”)
   echo “<P>Regular customer.”;
elseif ($find == “b”)
       echo “<P>Customer referred by TV advert.”;
       elseif ($find == “c”)
              echo “<P>Customer referred by phone directory.”;
              elseif ($find == “d”)
                     echo “<P>Customer referred by word of mouth.”;
```

Alternatively, we could write a switch statement:

```
switch ($find)
{
       case “a” :
            echo “<P>Regular customer.”;
            break;
       case “b” :
            echo “<P>Customer referred by TV advert.”;
            break;
       case “c” :
```

```
                echo "<P>Customer referred by phone directory.";
                break;
           case "d" :
                echo "<P>Customer referred by word of mouth.";
                break;
           default :
                echo "<P>We do not know how this customer found us.";
                break;
      }
```

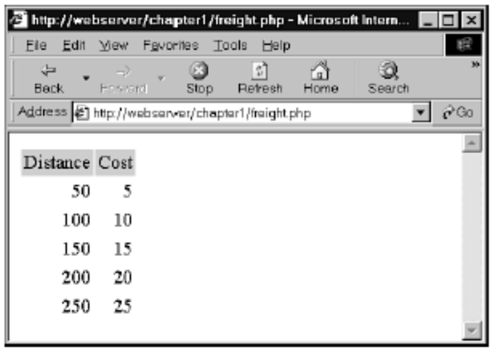
The switch statement behaves a little differently from an if or elseif statement. An if statement affects only one statement unless you deliberately use curly braces to create a block of statements. A switch behaves in the opposite way. When a case in a switch is activated, PHP will execute statements until it reaches a break statement. Without break statements, a switch would execute all the code following the case that was true. When a break statement is reached, the next line of code after the switch statement will be executed.

5.6.2.3 Comparing the Different Conditionals:

If you are not familiar with these statements, you might be asking, "Which one is the best?" That is not really a question we can answer. There is nothing that you can do with one or more else, elseif, or switch statements that you cannot do with a set of if statements. You should

5.6.3 Iteration: Repeating Actions:

One thing that computers have always been very good at is automating repetitive tasks. If there is something that you need done the same way a number of times, you can use a loop to repeat some parts of your program. Hesham wants a table displaying the freight cost that will be added to a customer's order (shown in figure 5.10-followed by its source code without any repetition). With the courier Hesham uses, the cost of freight depends on the distance the parcel is being shipped. The cost can be worked out with a simple formula.


http://webserver/chapter1/freight.php - Microsoft Intern...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://webserver/chapter1/freight.php
Go

Distance	Cost
50	5
100	10
150	15
200	20
250	25

Figure 5.10 freight cost

```
<table border = 0 cellpadding = 3>
<tr>
    <td bgcolor = "#CCCCCC" align = center>Distance</td>
    <td bgcolor = "#CCCCCC" align = center>Cost</td>
</tr>
<tr>
    <td align = right>50</td>
    <td align = right>5</td>
</tr>
<tr>
    <td align = right>100</td>
    <td align = right>10</td>
</tr>
```

```
<tr>
    <td align = right>150</td>
    <td align = right>15</td>
</tr>
<tr>
    <td align = right>200</td>
    <td align = right>20</td>
</tr>
<tr>
    <td align = right>250</td>
    <td align = right>25</td>
</tr>
</table>
```

It would be helpful if, rather than requiring an easily bored human—who must be paid for his time—to type the HTML, a cheap and tireless computer could do it. Loop statements tell PHP to execute a statement or block repeatedly.

5.6.3.1 while Loops:

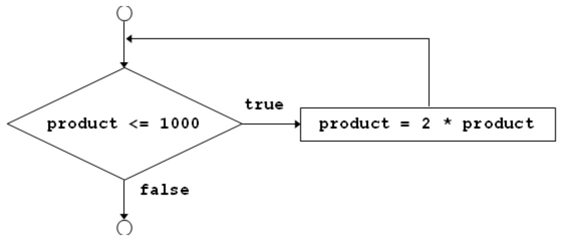


Figure 5.11 while loop

The simplest kind of loop in PHP is the while loop (shown in figure 5.11. Like an if statement, it relies on a condition. The difference between a while loop and an if statement is that an if statement executes the following block of code once if the condition is true. A while loop executes the block repeatedly for as long as the condition is true.

You generally use a while loop when you don't know how many iterations will be required to make the condition true. If you require a fixed number of iterations, consider using a for loop. The basic structure of a while loop is while(condition) expression;

The following while loop will display the numbers from 1 to 5.

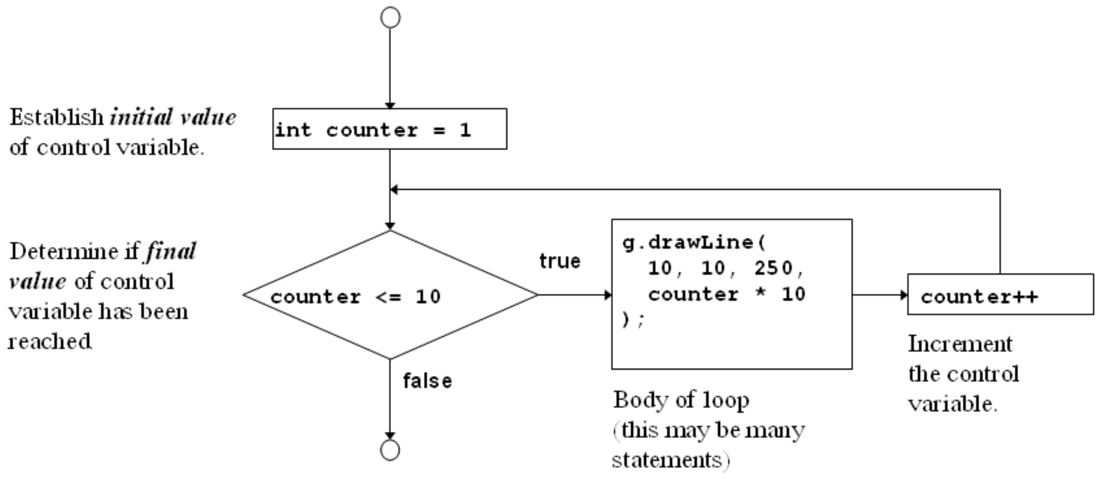
```
$num = 1;
while ($num <= 5 )
{
   echo $num.”<BR>”;
   $num++;
}
```

At the beginning of each iteration, the condition is tested. If the condition is false, the block will not be executed and the loop will end. The next statement after the loop will then be executed. freight.php—Generating Hesham's Freight Table with PHP

```
<table border = 0 cellpadding = 3>
   <tr>
       <td bgcolor = “#CCCCCC” align = center>Distance</td>
       <td bgcolor = “#CCCCCC” align = center>Cost</td>
   </tr>
<?
$distance = 50;
while ($distance <= 250 )
{
   echo "<tr>\n <td align = right>$distance</td>\n";
   echo " <td align = right>". $distance / 10 ."</td>\n</tr>\n";
   $distance += 50;
}
?>
</table>
```

5.6.3.2 for Loops:

The way that we used the while loops previously is very common. We set a counter to begin with. Before each iteration, we tested the counter in a condition. At the end of each iteration, we modified the counter (shown in figure 5.12).



Establish *initial value* of control variable.
int counter = 1
Determine if *final value* of control variable has been reached
counter <= 10
true
false
g.drawLine(10, 10, 250, counter * 10);
counter++
Body of loop (this may be many statements)
Increment the control variable.

Figure 5.12 for loop

We can write this style of loop in a more compact form using a for loop. The basic structure of a for loop is:

for(*expression1*; *condition*; *expression2*)
expression3;

- *expression1* is executed once at the start. Here you will usually set the initial value of a counter.
- The *condition* expression is tested before each iteration. If the expression returns false, iteration stops. Here you will usually test the counter against a limit.
- *expression2* is executed at the end of each iteration. Here you will usually adjust the
- value of the counter.
- *expression3* is executed once per iteration. This expression is usually a block of code and will contain the bulk of the loop code.

We can rewrite the while loop example in Listing 1.4 as a for loop. The PHP code will become

```
<?
for ($distance = 50; $distance <= 250; $distance += 50)
{
   echo "<tr>\n <td align = right>$distance</td>\n";
   echo " <td align = right>". $distance / 10 ."</td>\n</tr>\n";
}
?>
```

Both the while version and the for version are functionally identical. The for loop is somewhat more compact, saving two lines.

Both these loop types are equivalent—neither is better or worse than the other. In a given situation, you can use whichever you find more intuitive.

As a side note, you can combine variable variables with a for loop to iterate through a series of repetitive form fields. If, for example, you have form fields with names such as name1, name2, name3, and so on, you can process them like this:

```
for ($i=1; $i <= $numnames; $i++)
{
   $temp= "name$i";
   echo $$temp."<br>"; // or whatever processing you want to do
}
```

By dynamically creating the names of the variables, we can access each of the fields in turn.

5.6.3.3 do..while Loops:

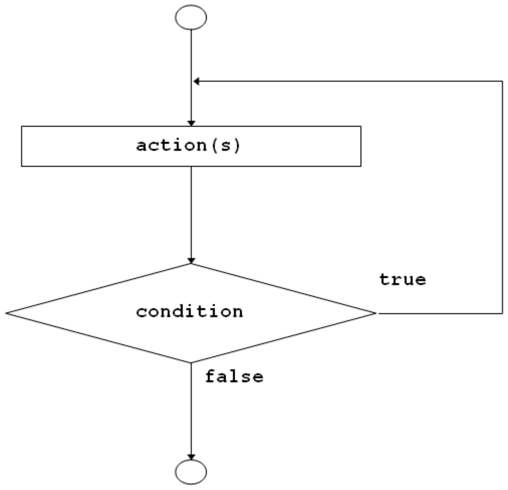


Figure 5.12 do..while loop

The final loop type we will mention behaves slightly differently. The general structure of a do..while statement is

do
expression;
while(*condition*);

A do..while loop differs from a while loop because the condition is tested at the end. This means that in a do..while loop, the statement or block within the loop is always executed at least once.

Even if we take this example in which the condition will be false at the start and can never become true, the loop will be executed once before checking the condition and ending.

```
$num = 100;
do
{
   echo $num."<BR>";
   $num--;
}
while ($num < 1 );
```

5.6.3.4 Breaking Out of a Control Structure or Script:

If you want to stop executing a piece of code, there are three approaches, depending on the effect you are trying to achieve. If you want to stop executing a loop, you can use the break statement as previously discussed in the section on switch. If you use the break statement in a loop, execution of the script will continue at the next line of the script after the loop.

If you want to jump to the next loop iteration, you can instead use the continue statement. If you want to finish executing the entire PHP script, you can use exit. This is typically useful when performing error checking. For example, we could modify our earlier example as follows:

```
if( $totalqty == 0)
{
echo "You did not order anything on the previous page!<br>";
exit;
}
```

5.7 A Sample of Questions:

Part 1: . . . .

Fill in the circle that represents the correct choice for each question in the given answer sheet (more than one choice for any question would be considered wrong answer).

An example of a correct answer:

a	b	c	d
●	○	○	○

Examples of wrong answers:

a	b	c	d
●	●	○	○

a	b	c	d
○	✓	○	○

a	b	c	d
○	◎	○	○

1- If you want to stop executing a piece of code, there . . . .
   a- is one approach   b- are two approaches
   c- are three approaches   d- are four approaches

2- Control structures are the structures within a language that allow us to control the flow of execution through a program or script As mentioned in figure 3.5 (previous chapter), programs can be reduced to into the main categories:
   a- sequence   b- selection
   c- repetition   d- All of the above

Part 2 : . . . .

Fill in the circle that represents the correct choice for each question in the given answer sheet (more than one choice for any question would be considered wrong answer).

1) A do..while loop differs from a while loop because the condition is tested at the end.
   a. True

b. False

2) Both these loop types are equivalent—neither is better or worse than the other. In a given situation, you can use whichever you find more intuitive.
    a. True
    b. False

3) Both the while version and the for version are functionally identical. The while loop is somewhat more compact, saving two lines.
    a. True
    b. False

Part 3 : ....

This part consists of fill in The Blank questions. Given below a table of words to choose from. On the answer sheet, put the number of the appropriate word in the space available for that question.

1 while	2 for	3 if	4 else	5 exit
6 switch	7 do	8 assign	9 continue	10 break

1. If you want to jump to the next loop iteration, you can instead use the . . .9 . statement.
2. A . . . 1. loop executes the block repeatedly for as long as the condition is true.
3. The . . .6 . statement works in a similar way to the if statement.
4. An else statement allows you to define an alternative action to be taken when the condition in an . . .3 . statement is false.