13 PHP 1: Starting to Script on the Server Side

LEARNING OBJECTIVES

At the end of this chapter, you will be able to:

- List the features of PHP as server side scripting language for dynamic Web development.
- Describe the basic syntax for writing PHP scripts.
- Define and use PHP variables for handling various types of data.
- Explain how errors are handled that occurs while running PHP script.
- Access the user input in PHP script send using HTML forms.
- Describe the decision and looping statements and use them to write the PHP scripts.
- Develop the dynamic Web pages using PHP and access them using browser.

This chapter gives a basic introduction to PHP and dynamic programing on the server side. You will learn how to develop simple PHP, how to structure your programs and embed script within HTML.

In this chapter we discover how to write scripts using PHP, a good choice for Web applications that need to interact and generate pages as required, depending on the user or processed data. PHP is an acronym meaning 'PHP: Hypertext Pre-Processor'; you could call this *recursive* in that it refers to itself! This is a naming style borrowed from Unix's GNU – GNU's Not Unix.

First of all let's work out why we need a dynamic programing language such as PHP. As there are many different programing languages around why learn and develop in yet another? PHP is a modern, Web-targeted language. It's modern because the people who have developed it have learnt from other languages and particularly suited it toward Web development. You may have picked up ideas and concepts from other languages such as C, C++ and Java that are in keeping with PHP's philosophy and implementation. Many of the programing structures and concepts shown in the JavaScript chapter are used here.

PHP is well suited for Web development particularly because of its ability to allow dynamic interaction with a Web page, building it up as real-time data is processed. This might be input from the user, database material, date sensitivity or other information which changes.

Concept Check

- Find out how PHP started by doing some research on the Web. What was its original name?

13.1 Starting to Script with PHP

PHP is a scripting language that is combined with HTML, either by embedding it within a Web document or by using it as a file that is processed alongside it, but on the server side!

Listing 1 shows an HTML page that has embedded PHP code. The code is sitting within HTML inside a special tag beginning with <?php and ending with ?>. Any code inside this tag is interpreted as PHP and must follow the syntax that is legal within a PHP program.

```
<html>
<head>
<title>Listing 1</title>
</head>
<body> Today's date is <?php echo date("l F d, Y"); ?>
</body>
</html>
```

Listing 1 Script using the `date` function.

When embedding code in this way it is particularly important to pay attention to formatting the script to make it easy to follow. It would be possible for example to improve the above program as in Listing 2. Notice here there are several differences:

1. Code is indented.
2. Comments are added.
3. More white space is added.

PHP can be indented and have white space (spacing) as required for your style and it is good programing practice. This improves readability and therefore adds a style that is quicker to follow.

Notice how adding white space and indentation does not interfere with the program's running or output. The only way such additions would be visible is if there were changes to the text within the print statement. In previous scripts echo was used instead of print, so what's the difference? Not much, except that print actually is a function that always returns 1 as a value, but it can be ignored. Echo is marginally faster as it doesn't do this.

```
<html>
<head>
<title>Listing 2</title>
</head>
<body> Today's date is
<?php
  /*
  * We can print today's date using PHP
  *
  *
  */
  print(date("l F d, Y"));
?>
</body>
</html>
```

Listing 2 Tidying the script using the `date` function.

Good Programing Practices Apply Here!

In the program the HTML body contains text that is output to the Web page and then PHP code follows. This is sent to the PHP engine or interpreter within the Web browser. Any output from the PHP code will follow the text from the HTML. In this case the output from PHP is sent by the print command and utilizes the `date` function, which allows the current date to be output in a format determined by a special set of characters. In this case we have selected the weekday name, the full month name, the day of the month and the four-digit year.

Another way to improve readability and quick understanding of your program is to add comments. In PHP this is similar to Java, C and C++. Comments can take up one line only and use a double forward slash, or they can take up several lines.

```
// This is a comment

/*
*  And so is this...!
*  but this one is spread
*  over several lines
*/
```

A multi-line comment stretches over several lines by being enclosed within a starting /* and ending with */.

It is possible you want the output of the program to be purely from the PHP, rather than the HTML. The way to do this would be to place the actual text within the print statement along with the `date` function.

Listing 3 shows how the output can be placed directly within the PHP code. Notice though, the Web page generated is exactly the same. The way text output is combined with a function is by using a full-stop or period character. This is slightly different to other languages where concatenation (the joining of two items such as strings) is performed by characters such as addition or ampersand. In this example you will notice that the output of the year is slightly different as it is in a two-digit format. This has happened because Listing 3 has a lowercase 'y', rather than an uppercase one as in the previous program, meaning that there should be a four-digit year.

```
<html>
<head>
<title>Listing 3</title>
</head>
<body>
<?php
  /*
  * We can print today's date using PHP
  *
  *
  */

  print("Today's date is ".date("l F d, y"));
?>
</body>
</html>
```

Listing 3 Using concatenation.

Concept Check

- You've encountered the `date` function but what other useful built-in functions exist? Look on the Web!
- Make a list of features you would consider to be good programing practices that you can apply to your PHP scripts.

13.2 Variables

As described earlier in the chapter on JavaScript, programing languages often need somewhere to keep values that are being worked on. PHP, just like other languages, uses variables to do this.

When a variable is encountered for the first time a memory space is set aside for the contents. Unlike a lot of programing languages such as C and Java, you don't have to state explicitly what type will fit in that variable, it will work it out from the assignment statement. This language feature is called being 'loosely typed'. Listing 4 contains several variables with different kinds of content. In PHP all variables are prefaced with a dollar sign. The first two in the listing, `$filmName` and `$cinema`, contain strings that can be any textual information built up of characters. The next variable, `$screenNumber`, contains an integer-type

```
<html>
  <head>
  <title>Listing 4</title>
  </head>
  <body>
  <?php
    /*
    * This program shows how to output variable
    * data and mix it with other textual output
    *
    */

    // first declare the variables
    $filmName = "Minority Report";
    $cinema = "Blue Screens Multiplex";
    $screenNumber = 5;
    $ticketCost = 7.50;
    $todaysDate = date("l, F d, Y");

    // now output the complete message
    print("Today, ".$todaysDate.", at the ".$cinema);
    print(" in screen ".$screenNumber." is ");
    print($filmName." which costs ".$ticketCost);
?>
</body>
</html>
```

Listing 4 Utilizing variables and output.

number while `$ticketCost`, because it is a monetary value, has a floating point type. Finally in this listing we have the assignment of a variable with the value of a function, in this case the date function. When the interpreter of the PHP encounters this it will first work out the function result and place that within the variable memory. It needs to do this first so it can allocate the correct memory space for the result.

The next section of the program outputs the variable contents. Notice here that although the output to the Web page is on one continuous line the program statements to do this are actually on three. The program is joining the text together, complete with variable output through the concatenation operator.

Errors in PHP

What happens if you miss out a dollar sign in front of a variable name? If you do this when the variable is being set up and declared then you will receive something like Fig. 1.

```
Parse error: parse error, unexpected '=' in /homepages/
42/d33119297/htdocs/book/list12_4.php on line 1
```

Figure 1 Error in variable name.

If the error is lower down in the program after its declaration, then the program may run but with strange results. The output from Listing 4 with a missing dollar sign on the `$cinema` variable in the print statement is shown in Fig. 2. If you look carefully at the output you will notice that instead of the cinema's name there is the word 'cinema'. In other words, the variable name itself has been printed rather than its contents.

```
Today, Monday August 02, 2004, at the cinema in screen 5 is
Minority Report which costs 7.5
```

Figure 2 Error in variable at print out.

You can also define constants; names that are associated with a value using the define statement. They don't need the dollar sign to precede them and can't be changed once set up but otherwise they are used as variables:

```
define("pi", 3.14159);
```

Checkpoint!

We now know how to:

- Set up a variable by declaring it and assigning a value.
- Create a constant.
- Output the value stored in a variable.
- Join text and variables together for output.
- Use a simple built-in function.
- Format for readability and good programing practice.

Concept Check

- Write a small program to output an HTML document via the PHP print command. Don't forget you can use all the usual format commands available in a Web document.

13.3 Getting Some Input

The next stage is to learn how to receive input from a user; after all, one of the benefits of using PHP is being able to dynamically build Web pages based on user data!

Listing 5 is an HTML document that calls a separate PHP file with collected input from the user. First of all there are the usual starting tags and title. Within the body is a form that has several parts. We can see there are a few input points consisting of a label such as 'Name:', 'Age:' and 'Date of Birth:', together with some input variables and associated types. The first and last lines of the form concern chaining the separate PHP file when the submit button is pressed. The `action` statement points to the program that will be able to access the user input through the various defined variables.

Listing 6 is a more complex program. It is not purely PHP but contains a mix of HTML and PHP. The listing begins with an insert outside of the main HTML code. This shows that it does not have to be contained within the HTML. It also shows that you can have PHP in separate fragments throughout a listing. This code sets the variable with the date function as before. The next section of the code is the usual HTML starting point, which leads into the main program.

```
<html>
<head>
<title>Listing 5</title>
</head>
<body>
<form action="list6.php" method="post">
  Name: <input type="text" name="aName"><br>
  Age: <input type="text" name="anAge"><br>
  Date of Birth: <input type="text" name="dob"><br>
<input type="submit" value="Send Data">
</form>
</body>
</html>
```

Listing 5 Setting up a form.

```
<?php
     $Today = date("l F d, Y");
?>
<html>
<head>
<title>Listing 6</title>
</head>
<body>
Today's date:
<?php
     /*
     ** print today's date
     */
```

```
    print("<h3>$Today</h3>");
    /*
    ** print greeting message
    */
    print($_REQUEST['aName'] . ", you are ");
    print($_REQUEST['anAge']);
    print(" years old and your date of birth is ");
    print($_REQUEST['dob']."<br>");

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

Listing 6 Accessing the form variables.

The main section initially prints out the date set up at the start of the listing. This is output through the print statement as before but this time it is formatted using HTML within the actual string. In this case the <h3> style is used as usual for a Web page. The best way to think about this is to imagine, correctly, that you are writing a Web page with the print output rather than, as you would with other languages such as C, outputting to a screen.

The following section of code accesses the variables set up in Listing 5 at runtime with `$_REQUEST['variable_name']`. When the submit button is pressed on the HTML form page the form fields are sent to the script noted in the `action` attribute. PHP takes these values and places them in an array called `_REQUEST`. An array is simply a series of memory locations accessed under one name. In this case each stored form field is placed in a memory location and accessed by the programer by placing the name of the required field in the square brackets. You may notice that when the form fields are described in the HTML code there is a type attached to each; if you enter a string where a number is expected then PHP will enter a zero for that field.

Another point to note in this listing is the use of newline and line breaks. This allows any output to continue on a line until you wish to output a break of some type. Remember all HTML formatting commands are available for use in your print.

Checkpoint!

This section has covered several key points:

- Input via variables.
- Transfer control to another script at runtime.
- Accessing the variables.
- Processing the user-entered data.
- Formatting output via HTML commands.

Concept Check

- Write two scripts, one to pick up the details of a library customer and the other to store the details (in variables) and display them.

13.4 Decisions

When writing scripts and programs very often we need certain sections of code to execute only if certain conditions have been met. The idea is the same as was met in the JavaScript chapter. In the case of PHP, the ability to generate Web pages in certain formats or with particular features may be the outcome of such decisions.

Listing 7 shows how to control the flow of execution for your script based on certain conditions being present. First of all the program prints today's date using the date function as before but notice this time that the variable is embedded within the actual output string. If you are used to other programing languages you would probably think that the actual variable name itself would be printed out but in the case of PHP any reference to a known variable within such a string results in the contents, not the name, being printed out.

```
<html>
<head>
<title>Listing 7</title>
</head>
<body>
<h1>
<?php
     $Today = date("l F d, Y");
     print("Today is $Today -");

     /*
     ** Get this years leap year status!
     */
     $Today = date("L");
     if($Today == 1)
     {
          print("This year is a leap year!");
     }
     else
     {
          print("This year is not a leap year");
     }
?>
</h1>
</body>
</html>
```

Listing 7 Controlling execution flow.

The next section of code again accesses the `date` function and re-uses the same variable, this time accessing the `date` function's leap year status value.

To access this value an 'L' is placed in the string. When this is done the return value is either a 1 if the current year is a leap year, or 0 if not. If you want to react in a program to whichever case is true then you could use the condition structure shown.

This particular method is the same as that described in the chapter on JavaScript. It has a structure that begins with the `if` statement itself, followed by brackets that contain a logical condition. The case here is

Table 1 Comparison operators

Operator	*Operation Performed*	*Example*
<	Less than	`$num < 12`
>	Greater than	`$num > 32`
<=	Less than or equal to	`$num <= 23`
>=	Greater than or equal to	`$num >= 78`
==	Equal to	`$num == 3`
===	Identical	`$num === NULL`
!=	Not equal to	`$num !=20`
!==	Not identical to	`$num !== FALSE`
AND, &&	Logical and	`$num1 AND $num2` `$num1 && $num2`
OR, ||	Logical or	`$num1 OR $num2` `$num1 || $num2`
XOR	Exclusive or	`$num1 XOR $num2`
!	Not	`! $num`

that a variable is tested to see if it contains 1. Note that the test is done with double equal signs rather than a single equals, which is used for assigning values.

Table 1 shows other logical operators that are available; some of which are fairly obvious, others are more subtle.

The condition or test is performed and the result, if true, allows the next block of code to be executed. The `if` statement can also contain a block of code to be completed if the condition is false, contained after an `else` statement.

In this way the flow of execution can be controlled, depending on user input, events arising or processing of data taking place.

In Listing 8 we find another way of controlling the flow of a script, depending on the value contained in a variable. Again, this is exactly the same as described for JavaScript, using the `case` statement and allows a straightforward means of reacting to multiple possibilities. In this example the date is first of all collected and printed out. The date is again collected from the function on its own as a single number and placed in `$diaryDate`. The `case` structure begins with a `switch` statement, followed by the variable you want to check, in this case the `$diaryDate` variable. Within the next block, marked out by curly brackets, is a set of executable lines, each starting with a `case` statement. This contains the choice for that particular response. So, if it was 3 August 2004 then `case 03` would be activated and the code starting at that point executed until it hit the `break` statement at the end. Once the execution has got to this point the code begins executing outside of the code block for the structure.

Finally, if all the cases have been checked and none match the date contained in `$diaryDate`, then it is possible to add in a `default` statement, which starts a section of code to be executed in that instance. Default means, in this case, if all else fails, run this code. So, if it was 7 August 2004 the default code would run and it would report 'You have no booked events today'.

Now that we have seen how to use both `if` statements and `case` statements it is possible to compare them. In Listing 9 there is another version of the same program but using `if else` rather than `case`.

```
<html>
<head>
<title>listing 8</title>
</head>
<body>
<?php
      $Today = date("l F d, Y");
      print("Today is $Today, I will check your
                  diary...<br>\n");

      $diaryDate = date("d");

      switch($diaryDate)
      {
        case 3 : print("you have a dinner date"); break;
        case 10 : print("dentist appointment today"); break;
        case 23 : print("have the day off!"); break;
        case 29 : print("go to conference"); break;
        default : print("You have no booked events today!");
      }
?>
</body></html>
```

Listing 8 Using the `switch-case` structure.

While it responds in the same way as the version with `case` it is perhaps a little messier! It is very similar to the `case` version but contains repetition of the check and variable name that it is acting on. Again, there is a default option that is activated should the execution not follow one of the paths above it.

```
<html>
<head>
<title>Listing 9</title>
</head>
<body>
<?php
      $Today = date("l F d, Y");
      print("Today is $Today, I will check your
                  diary...<br>\n");
      $diaryDate = date("d");
      if ($diaryDate==3) print("you have a dinner date");
      else
      if ($diaryDate==10) print("dentist appointment
                  today");
      else
      if ($diaryDate==23) print("have the day off!");
      else
```

```
        if ($diaryDate==29) print("go to conference");
        else
        print("You have no booked events today!");
?>
</body>
</html>
```

Listing 9 Script using the `date` function.

13.5 Looping

We may also want a script to repeat a certain number of times. This may be to process data, collect input or output some results, for example.

The `for` Loop

If you know how many times you'd like a section of code to repeat, you can use a `for` statement. Listing 10 shows how this is done. After an initial piece of code using HTML, the PHP section launches straight into a `for` loop. A `for` loop contains the statement itself followed by brackets containing controlling parameters. The first parameter is executed once only before the start of the loop; this is usually a variable that needs initializing. The next parameter is a test that will stop the loop. In this case the test is acting on the `$count` variable and says 'while the `$count` variable is less than or equal to 12 then do the following loop'. The final parameter is executed every time the loop has been run through once. Usually this is a statement that will increment the loop counter.

```
<html>
<head>
<title>Listing 10</title>
</head>
<body>
<h1>I must learn my 7 times table</h1><br>

<?php
  for($count = 1; $count <= 12; $count++)
  {
    print("7 * $count =".(7*$count)."<br>");
  }
?>
</h1>
</body>
</html>
```

Listing 10 Introducing the `for` loop.

So, the idea is

```
for (INITIALIZE; END CONDITION; UPDATE COUNTER) {
        BODY
}
```

In this program there is some formatting done in HTML, such as the selection of styles, while the rest is done in PHP. In this case we can see that there is a linefeed required to drop down to the next line. As in Listing 9 the variable value is included in the actual output string.

When the script executes the loop it runs through each time incrementing (adding 1 to) the `$count` variable. The first time the loop is entered `$count` is set at 1 and is printed out and calculated. After one execution `$count` is incremented again and so on. Finally, the loop completes its twelfth execution and the counter variable becomes 13. At this point the second parameter in the `for` statement becomes false and the loop is exited.

The `while` Loop

Another way to do the same thing is to use a `while` loop. In this approach you have to do more yourself: for example there is only one parameter as opposed to the three in the `for` statement. The only parameter is the check to see if the counter has reached its limit. When the script in Listing 11 is run the counter is set outside the loop to 1. The loop checks the value of `$count` on entry. If the count is already above 12 then the loop will not be entered at all! In this case the counter has to be incremented within the loop block itself. Note that if the count loop were not altered within the loop block then it would run forever!

```
<html>
<head>
<title>Listing 11</title>
</head>
<body>
<h1>I must learn my 7 times table</h1><br>
<?php
      $count=1;
      while( $count<=12 )
      {
        print("7 * $count =".(7*$count)."<br>");
        $count++;
      }
?>
</body>
</html>
```

Listing 11 Script using `while`.

Listing 12 contains yet another way of writing the same program. This time it is made with a `do . . . while` loop but what is the difference this time? The difference here is that the loop is *always* executed at least once as the check is at the end rather than the beginning. This can be useful for certain types of problem, so it is worth considering carefully what type of loop is required.

```
<html>
<head>
<title>Listing 12</title>
</head>
```

```
<body>
<h1>I must learn my 7 times table</h1><br>
<?php
    $count=1;

    do {
      print("7 * $count =".(7*$count)."<br>");

      $count++;

    } while( $count<=12 )

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

Listing 12 Yet another loop structure.

Using break and continue

If you used one of these loops and were processing information – for example inputting data – and wanted to end early (before the natural ending you had set up) how would you do it? Usually, you would have some kind of check or in the case of the `for` loop, an exact number. In the case of the check you could make the check fall through deliberately early. For example, in the multiplication table script we could have said (although this is not very elegant):

```
if ($count==8) $count=12;
```

If this was positioned correctly in one of the loops it would have created an early termination before the count had reached 12. Another way to do this is to use the `break` statement:

```
if ($count==8) break;
```

This forces execution to begin again outside the current loop block when 8 has been reached. This may be useful if an abnormal condition came about in the course of data collection, for example. Another interesting inclusion in the PHP language is the `continue` statement.

Listing 13 shows the `continue` statement in use. The loop would normally just print out numbers from 0 to 5 but instead misses out 2. This is because the effect of the `continue` statement makes the loop miss out that particular run through of the loop and skip to the next. In this case the print statement is not executed and the loop begins at the top again. If the continue statement was swapped for a `break`, the loop would terminate at 2.

```
<html>
<head>
<title>Listing 13</title>
</head>
<body>
<h1>Doing the loop</h1><br>

<?php
  for ($i = 0; $i <= 5; $i++) {
```

```
        if ($i == 2) continue;
        print "$i<br>";
    }
?>

</body>
</html>
```

Listing 13 Skipping an iteration.

In this section you have seen how to control the flow of your program by making decisions with the `if` statement and repeating with the `for` statement.

Checkpoint!

We now know:

- How to control the manner in which a program executes, depending on conditions.
- Two ways to do this: `if` and `case`.
- That there is more than one way to achieve the same result.
- How to repeat a code a specific number of times.

13.6 Examples

We know that PHP script runs under the Web server and is used to generate the Web pages dynamically on the fly. The PHP code inside your Web page is interpreted and replaced by the HTML output. The main aim of the server side scripting is to provide the services to clients. The examples in this section give you an idea about how both the above purposes can be achieved with PHP.

Listing 14 contains the PHP code which generates the HTML header tags in all sizes (h1, h2, h3, h4, h5, h6) having the text "Learning PHP is fun".

```
<html>
<body>

<?php
    //display text in all header sizes
    for($i=1;$i<=6;$i++) {
        print("<h".$i.">");
        print("Learning PHP is fun");
        print("</h".$i.">");
    }
?>

</body>
</html>
```

Listing 14 Generate header tags.

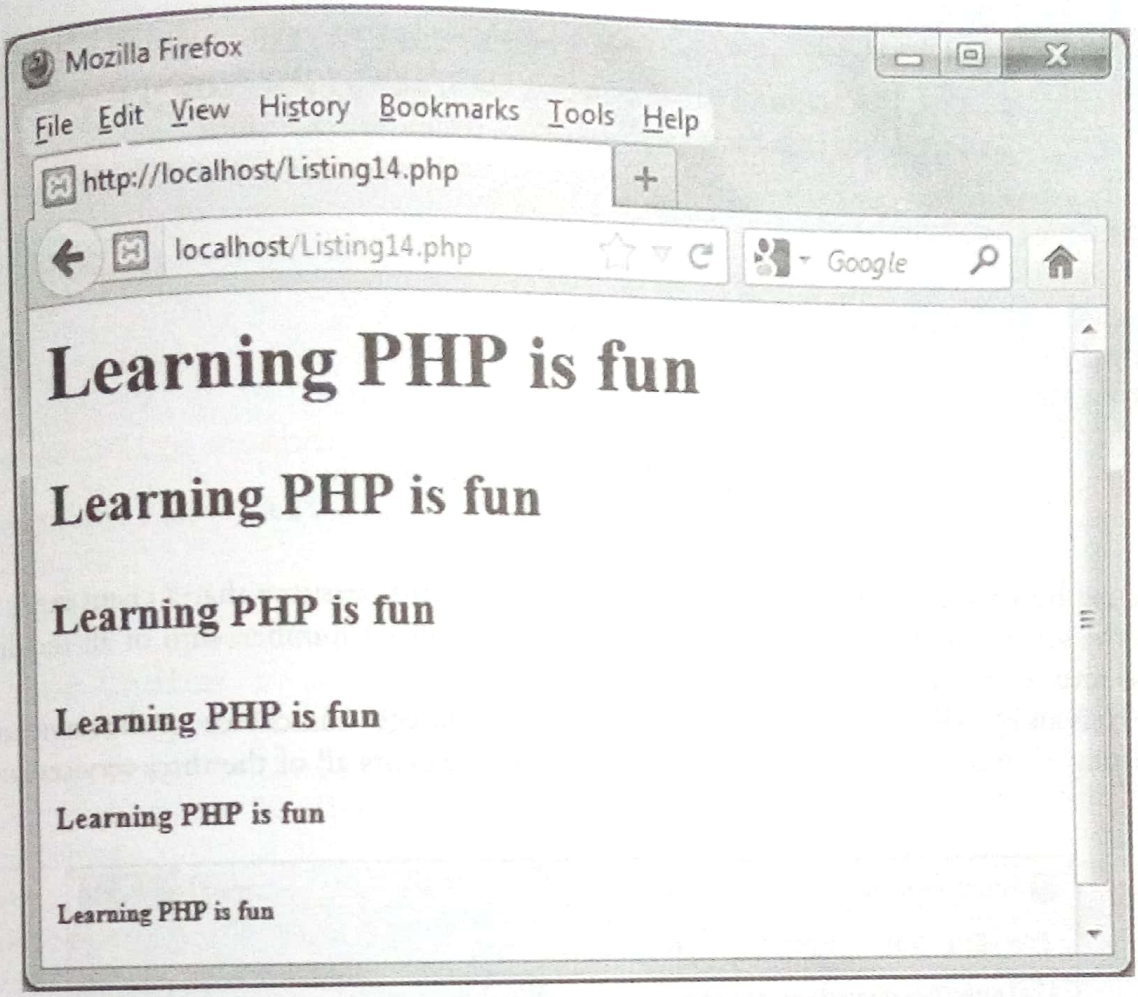

Figure 3 Web page generated by Listing 14.

To view the resulting HTML page, assume that the above code is stored in `Listing14.php` file. Store this file in `C:\xampp\htdocs` directory assuming that XAMPP server is installed in `C:\xampp` directory. To run the `Listing14.php`, use the following URL:

```
http://localhost/Listing14.php
```

The Web page generated is shown in Fig. 3. View the source code of the page from your browser facility to see that how PHP code is replaced by HTML tags.

Listing 15 is an example that uses the HTML table tag to generate a table of squares for numbers 1 to 10. The output of Listing 15 is shown in Fig. 4. You can see the source of the Web page to see how the table is generated.

```
<html>
<body>

<?php
   print("<h1>Table of Squares: 1 to 10</h1>");
   print("<table border=1>");
   print("<tr><th>Number</th><th>Square</th></tr>");
   for($i=1;$i<=10;$i++){
        print("<tr>");
```

```
        print("<td align='center'>".$i."</td>");
        print("<td align='center'>".$i*$i."</td>");
        print("</tr>");
    }
  print("</table>");
?>

</body>
</html>
```

Listing 15 Display table of squares.

Let us now see how we can use PHP to create services for clients. Assume that a client sends an integer number to the server offering three services: largest digit of a given number, sum of all the digits of the number and reverse of a given number based on client choice.

Listing 16 is an HTML form used by the client to send an integer and choice for the service using radio buttons. Listing 17 is a server side PHP program which implements all of the three services and executes

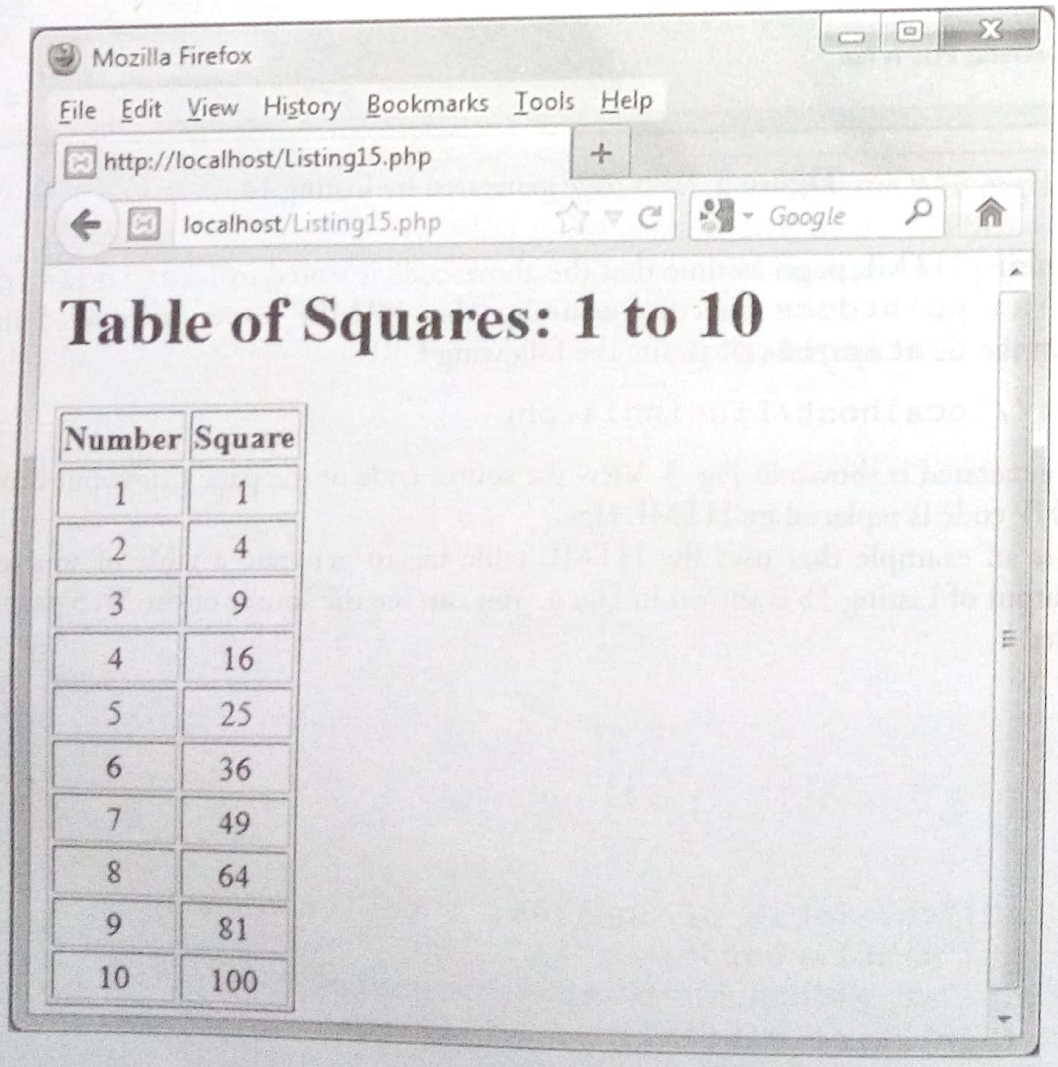

Figure 4 Web page generated by Listing 15.

one of them on the integer number sent using case statement. Note that Listing 17 uses the mathematical function `floor( )` in case 1 to truncate the decimal part after diving n by 10 whereas type casting is used in cases 2 and 3.

```
<html>
<body>
<h1>Three services are available</h1>
<fieldset>
<legend>Give number and choose service</legend>
<form action="Listing17.php" method="post">
  <br />
  Give an Integer > 0 :
  <input type="text" name="num"><br /><br />
  Your Choice :
  <input type="radio" name="choice" value="1" />Largest Digit
  <input type="radio" name="choice" value="2" />Sum of Digits
  <input type="radio" name="choice" value="3" />Reverse of
  number
  <br /><br/>
  <input type="submit" value="Submit">
</form>
</fieldset>
</body>
</html>
```

Listing 16 HTML form used by client.

```
<html>
<body>
<?php
    $n = $_REQUEST['num'];
    $ch = $_REQUEST['choice'];
    switch($ch)
    {
     case 1: $max = 0;
             do {
             $last_digit = $n % 10;
             if($last_digit > $max)
               $max = $last_digit;
             $n = floor($n/10);
             }
             while($n > 0);
```

```
                print("<h1>Largest digit is : ".$max."</h1>");
                break;
        case 2: $sum = 0;
                do {
                $last_digit = $n % 10;
                $sum += $last_digit;
                $n = (integer)($n/10);
                }
                while($n > 0);
                print("<h1>Sum of digits is : ".$sum."</h1>");
                break;
        case 3: $n_rev = 0;
                do {
                $last_digit = $n % 10;
                $n_rev = $n_rev*10+$last_digit;
                $n = (integer)($n/10);
                }
                 while($n > 0);
                 print("<h1>Reverse number is : ".$n_rev."</h1>");
                 break;
         default:print("<h1>Error</h1>");
                 break;
    }
?>
</body>
</html>
```

Listing 17 Implementing services using PHP.

Assume that Listing 16 is stored in `Listing16.html` and Listing 17 is stored in `Listing17.php`. Executing `Listing16.html` (`http://localhost/Listing16.html`) displays the Web page shown in Fig. 5. It shows the page after entering the number 4691 and choice for reversing the number. Once the "Submit" button is pressed `Listing17.php` runs and produces the Web page as shown in Fig. 6.

Checkpoint!

We now know:

- PHP code contained in Web page is interpreted at server side and replaced by the HTML tags.
- Web server after executing PHP code sends the resulting page (having only HTML code) to the Web Browser for display.
- PHP is used to develop the services running at server side which are accessed by the clients using proper interface.

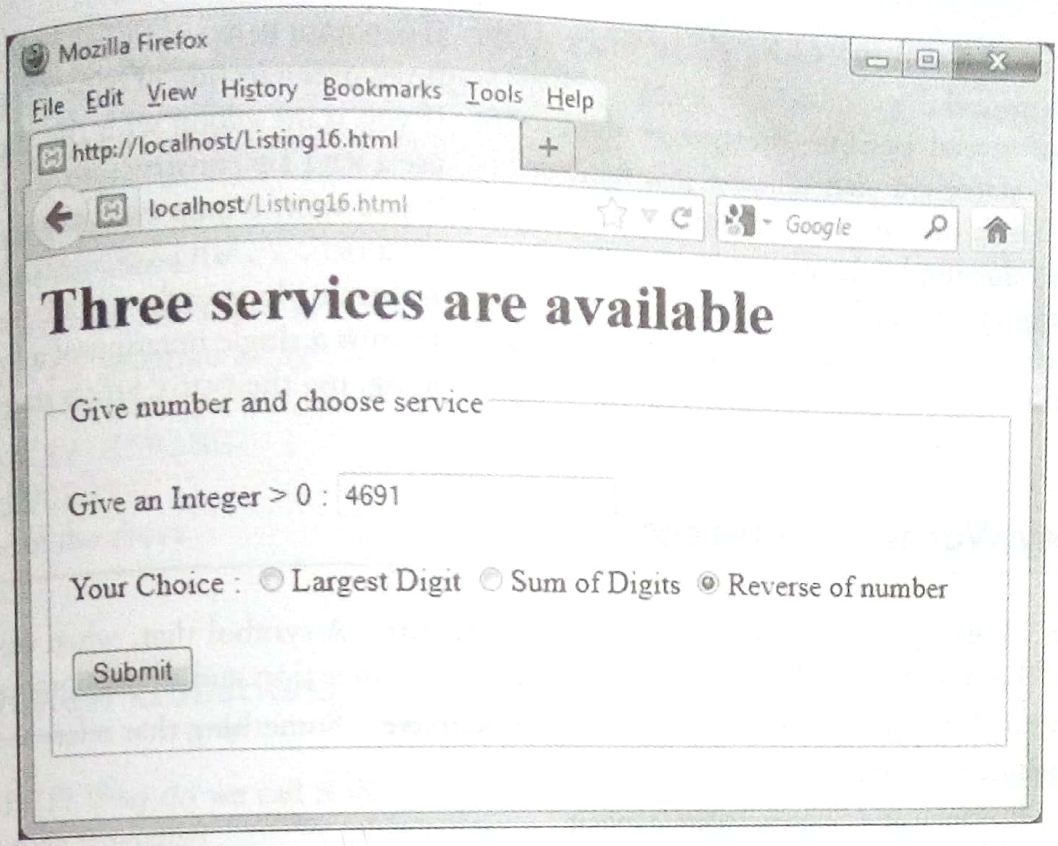

Figure 5 Web page for client.

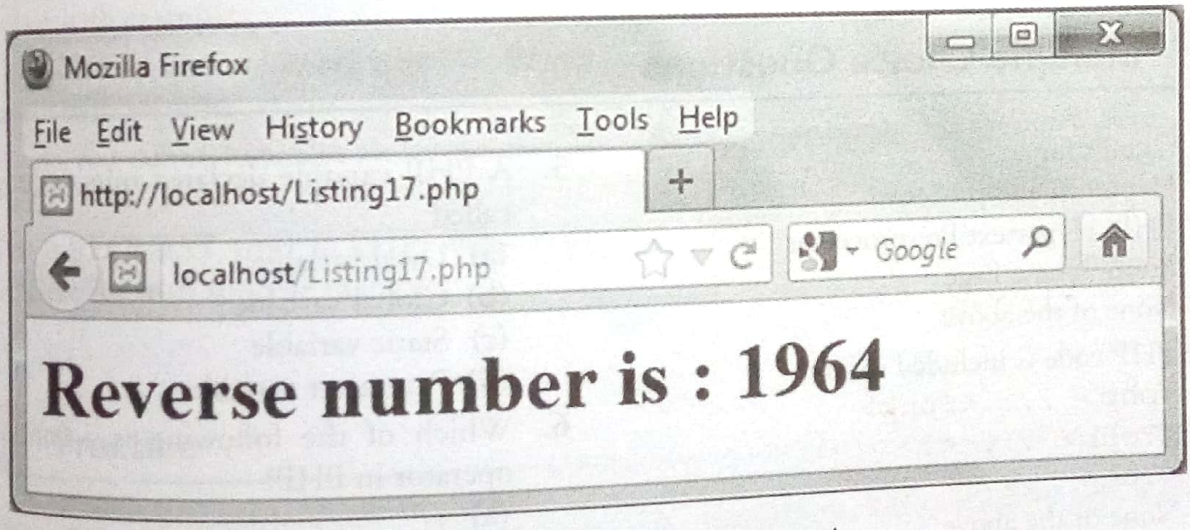

Figure 6 Web page after getting service.

Summary

1. PHP can be either embedded in an HTML file or in a separate file.
2. Good programing practice should be maintained by ensuring readability – add structure, spacing and comments.
3. There are lots of in-built functions in PHP; make sure you check them before writing your own to do the same thing!
4. Choose good variable names and remember, in PHP, they begin with the dollar symbol. Also, they are loosely typed.
5. You can access form variables set up in an HTML script from within a separate PHP file.
6. `If` statements can check that certain conditions are present and then execute appropriate code.