14 PHP 2: Arrays, Functions and Forms

LEARNING OBJECTIVES

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

- Define and use numerical and associative arrays.
- List the various built-in array functions provided by PHP and use them in programs.
- Access the form data using in-built associate arrays: `$_GET[]`, `$_POST[]` and `$_REQUEST[]`
- Write user-defined functions in PHP.
- Access the details of client and server using `$_SERVER[]` array.
- Define and use the strings.
- List the in-built string functions and use them in programming.
- Describe the PHP features to handle files, encryption, Emails and file uploading.

Here, you learn how to further manipulate data within PHP and in the process get to grips with new functions, loop structures and the verification of data input through forms. Simple arrays to dynamic structures are discussed, along with the ability to manipulate strings through special functions. Attention is particularly given to how to enlist specific features of PHP when processing data and how these can be used to add security.

This chapter continues our exploration of PHP and we start to meet features that make it such a useful language for the Web. The language also includes aspects that are similar to, if not exactly the same as, a lot of programing languages that went before it and it builds on these. For example, in this chapter there are sections of developing your own functions, string manipulation and arrays. These are further enhanced by realizing features exist to make life a lot easier for programing on the Web.

PHP contains many of the structures and elements that JavaScript does. Another example of this is arrays, although there are some slight differences that you will pick up. Arrays can, for example, be treated in a dynamic sense (if you should wish to), exactly like a stack data structure with the accompanying functionality. Details of the visitor's software can be gathered and the browser controlled. Form information can be gathered easily and processed with advanced string manipulation. Information can be written to local server side files and encrypted if necessary. This kind of built-in functionality makes programing rapid and enjoyable without the need to reinvent the wheel!

14.1 Arrays

Arrays, as you may recall from JavaScript, are like variables but instead of just holding a single value they can hold many. To be able to hold more than one value and still be able to access them for changing and reading, you must use an index, with the first index value being 0, rather than 1.

Listing 1 shows an array being declared as `$myarray`, which is then initialized with string values. The contents of the array are then accessed and printed out. Note again that the concatenation (joining) operator is used to add the three strings together for output.

Scanned by CamScanner

```
<?php
    $myarray = array( );
    $myarray[0] = "This";
    $myarray[1] = " is ";
    $myarray[2] = " my array";
    echo( $myarray[0].$myarray[1].$myarray[2]);
?>
```

Listing 1 Using an array.

It's also possible to set the values at the start of the program rather than, as above, as separate operations:

```
$myarray = array("This", " is ", " my array");
```

The array values can be different types; the above values are all strings. You can, for example, mix them quite easily:

```
$myarray = array("one", 2, 3.5);
```

The last two values in this case are numerical, so you can perform numerical operations with them:

```
$total = $myarray[1] + $myarray[2];
```

Once you have assigned a value to an array element it's still possible to change it:

```
$preference = array("red", "white", "blue");
$preference[1] = "green";
```

The array will now contain red at index 0, green at index 1 and blue at index 2.

Arrays and Loops

PHP has some clever ways of accessing arrays within loops, so you can visit all or particular members of an array when you need to without complex programing.

```
<html><head></head>
<body><ol>
<?php
    $preferences = array ("red", "white", "blue","silver",
                   "aqua","cyan", "yellow");
    echo("The current preferences are");
    foreach($preferences as $value)
    {
        echo("<li>This preference is: $value </li>");
    }
?>
</ol></body></html>
```

Listing 2 The `foreach` loop.

Scanned by CamScanner

Listing 2 shows how PHP can be mixed with HTML to produce a fairly simple script. The various preferences are output to the browser, one at a time on a numbered line. The numbering is done using HTML to produce a list. The loop is formed with a `foreach` structure. This works by extracting each value one at a time from the stated array. As the loop progresses, these values are placed in the second variable, `$value`. The echo statement within the loop itself then outputs it. The exact output in this case will be:

```
1. This preference is: red
2. This preference is: white
3. This preference is: blue
4. This preference is: silver
5. This preference is: aqua
6. This preference is: cyan
7. This preference is: yellow
```

This is a little easier than using a `for` loop as you don't need to know how big the array is for its end point. If the `for` loop, or any of the other loops such as `while`, were used then you would need this information. This can be gained by using the `sizeof()` or `count()` functions on the array.

Listing 3 shows the same script but rewritten with the for loop. The terminating point is found with `sizeof()`.

```
<html><head></head>
<body><ol>
<?php
    $preferences = array ("red", "white" , "blue","silver",
                  "aqua","cyan", "yellow");
    echo("The current preferences are");
    for ($i=0; $i<sizeof($preferences); $i++) {
    {
        $value = $preferences[$i];
        echo("<li>This preference is: $value</li>");
    }
?>
</ol></body></html>
```

Listing 3 An equivalent `for` loop.

What if we wanted to add more values to the array? Well, in some languages and traditional static arrays this would not be possible, or at least it would be a little cumbersome to achieve! However, with PHP it is possible to add values to the beginning and end of an array using `array_unshift()` and `array_push()`. To add values to the beginning:

```
array_unshift($preferences, "black", "gold");
```

It's possible to add some at the end with:

```
array_push($preferences,"green", "pink");
```

You can probably think of how to remove items from the array; you could delete the item you want rid of and copy the others around it into new positions. PHP makes it easy by including more commands! To remove an item from the start of an array use:

```
array_shift($preferences);
```

Scanned by CamScanner

To remove items at the end:

```
array_pop($preferences);
```

If you have studied data structures you will notice that PHP is treating the dynamic array as a stack with `pop` and `push`. This can be visualized as a stack of plates: when you want to put a plate (the information) in the stack you place it on top (`push`) and when you want to take an item off you take the top one (`pop`). This is the well-known LIFO (Last In First Out) structure. Of course, the array can be accessed in other ways too, so there is only a similarity with that particular kind of command set.

Arrays can also be sorted using the `sort()` command, which will order them from lowest to highest:

```
sort($preferences);
```

If there is a set of strings stored in the array they will be sorted alphabetically. The type of sort applied can be chosen with a second optional parameter:

1. `SORT_REGULAR` – compare items normally (don't change types)
2. `SORT_NUMERIC` – compare items numerically
3. `SORT_STRING` – compare items as strings
4. `SORT_LOCALE_STRING` – compare items as strings, based on the current locale.

Alternatively, it's possible to mix them up:

```
shuffle($preferences);
```

It's also possible to join two arrays together:

```
$endArray = array_merge($prefs1, $prefs2);
```

Here the end array is made up of the joining of two other arrays, `$prefs1` and `$prefs2`.

An array can also be copied in sections:

```
$endArray = array_slice($prefs, 2, 6);
```

In this case the end array is made up of the `$prefs` array between 2 and 6.

There are a great many different useful functions for manipulating arrays, covering extracting sections of data, sorting, summing, ordering and merging. There is even a function for choosing random data from the array, if you wish!

Keys and Values

An important aspect of using arrays with PHP is the ability to have key–value pairs. This allows you to refer to an element in an array by a name rather than an index, as shown in Listing 4.

This shows how easy it is to remember where you put your data held in an array. The points to note here are the difference in quotes used, singular and double, when defining your keys and values along with the special association operator =>.

The key–value array format is also used with forms where the input name on the form is the key and the content is the value.

The HTML form is made up of the usual input elements consisting of text input and radio buttons. The difference to JavaScript or other processing is what happens when the submit button is clicked. As you can see from the action property it is sent to a PHP script called `check.php` via the post method.

Scanned by CamScanner

```
<?php
    $mydata = array ('name'=>"Charles Lutwidge Dodgson",
                     'age'=>28, 'occupation'=>"logician,
                     writer");
    echo("my name is ".$mydata['name']." my age is
       ".$mydata['age']);
?>
```

Listing 4 Using an associative array.

Forms and Associative Arrays

Accessing form values is quite easy with PHP and very useful! For example, Listing 5 is an HTML input form that captures data from a user, which it then sends to a PHP script.

```
<html><head></head>
<body>
<form action="check.php" method="post">
What is your customer number?
<input type="text" size="12" name="customer"><br>
<br>Please select your pizza: <br>
<input type="radio" size="40" name="pizza"
              value="Mushroom">Mushroom<br>
<input type="radio" size="40" name="pizza"
              value="Pepperoni">Pepperoni<br>
<input type="radio" size="40" name="pizza"
              value="Calzone">Calzone<br>
<input type="radio" size="40" name="pizza" value="Four
              Cheeses">Four Cheeses<br>
<input type="radio" size="40" name="pizza" value="Veg
              Special">Veg Special<br>
<input type="radio" size="40" name="pizza" value="Sea Food
              Special">Sea Food Special<br>
<br>Please Select base: <br>
<input type="radio" size="40" name="base"
              value="Thin">Thin<br>
<input type="radio" size="40" name="base"
              value="Crispy">Crispy<br>
<input type="radio" size="40" name="base"
              value="Deep">Deep<br>
<input type="radio" size="40" name="base"
              value="Stuffed">Stuffed<br>
<br>Extras: <br>
<input type="radio" size="40" name="drink"
              value="Coke">Coke<br>
<input type="radio" size="40" name="drink"
              value="Soda">Soda<br>
```

Scanned by CamScanner

```
<input type="radio" size="40" name="sidedish"
                value="salad">Salad<br><br>
<input type="submit" value="Submit your order!">
</form>
</body></html>
```

Listing 5 HTML input form for PHP processing.

Listing 6 is the `check.php` script that processes the HTML form. The first thing that happens in this script is the transfer of content from the form `POST` array into variables. Notice here, as mentioned, that the input labels become keys for the actual array.

```
<?php
$customer = $_POST['customer'];
$pizza = $_POST['pizza'];
$base = $_POST['base'];
$drink = $_POST['drink'];
$sidedish = $_POST['sidedish'];
if (empty($customer) || empty($pizza) || empty($base)) {
// stop execution of PHP script using die function!
die("<br>Please fill out the form carefully....<br>");
}
echo("Customer $customer has ordered a $pizza with a $base
                base ");
if (!empty($sidedish)) {
    echo(" they'd also like a $sidedish ");
}
if (!empty($drink)) {
echo(" and have chosen a $drink drink");
}
?>
```

Listing 6 PHP to process the HTML form.

Once the information has been transferred for simplified access, the contents can be checked. Here, an initial check is made to see if the form has been filled out reasonably; that there is a customer number, and that a pizza has been chosen as well as a base. If these are not present, then the form stops. After printing out the customer's main details, checks are made on two optional items, the side salad and drink. It only prints these out if selected. You can check your variable contents as usual and verify that they match what you expect them to be. For example, PHP has various ways of doing broader checks on variables. As seen above an initial check may be on whether there is actually any content at all! This is done with the `empty()` function. It is also possible to see whether they have answered with a number with `is_numeric()`, although you can check any variable for various types including: `is_integer()`, `is_bool()`, `is_float()`, `is_array()`, `is_string()`, `is_null()` and `is_long()`.

Scanned by CamScanner

Checkpoint!

- An array has a single variable name but multiple accessible contents.
- Indexing begins at 0.
- Array items can be used in the same way as normal variables, applying various operations such as mathematical or string concatenation.
- An array can be initialized when it is declared.
- Useful structures allow iteration through an array such as `foreach`. Other loop structures can be used to index through the array too.
- Functions enable various operations to be performed on an array, such as `sizeof()`, `shuffle()` and `sort()`.
- An array can also be treated as a dynamic data structure such as a LIFO stack.
- Associative arrays allow access through keys.
- Associative arrays are used to access many important values in PHP, such as values from server (via `$_SERVER`) and forms (via `$_POST`).
- You should verify that returned values from forms are as expected, using PHP functions.

Concept Check

- Attempt to use an array to store information (perhaps about a music collection) and output the data to a Web page.
- Try the various functions on this data, for example `shuffle()` and `sort()`.
- As a slightly longer exercise, attempt to implement a stack with a Web interface, utilizing the array functions `array_pop()` and `array_push()`.

14.2 Functions

The idea of a function is to separate a section of code that can be used frequently, possibly applied to different data for example. You have already encountered many built-in functions present in PHP. These are recognizable by the rounded brackets after the function name; they can also contain *parameters*. Functions can also be user-made too, using the function keyword:

```
function test1( ) {
    echo("Hey this is inside your function<hr>");
}

function test2($name) {
    echo("Hey, $name, this is the second function");
}

functions test3($num1, $num2, $num3 = 10) {
  $someOp = $num1 + $num2 +num3;
  return $someOp;
}
```

The functions can be placed anywhere in the Web page as long as they are included in the usual `<?php ?>` brackets. They can then be called just like the internal PHP functions:

```
<?php
    test1( );
```

Scanned by CamScanner

```
        test2("Bert Fry");
        $result = test3(4, 5, 6);
    ?>
```

In function `test3` there are 3 arguments (or parameters); the last variable has a default value of 10 if the user does not supply one. This function also has a value that is returned, in this case to the result variable.

14.3 Browser Control

PHP can control various features of a browser. This is important as often there is a need to reload the same page (maybe if a page form needs re-attempting) or redirect the user to another one.

Some of these features are accessed by controlling the information sent out in the HTTP header to the browser. This uses the `header()` command such as:

```
header("Location: index.html");
```

In this case there would be a redirection to the `index.html` page. It's also possible to reload the current page:

```
<?php
header("Cache-Control:no-cache");
$self = $SERVER['PHP_SELF'];
...
<form action = "<?php $self ?>" method ="post">
```

This code fragment shows a common technique where information is processed to some degree on the same page as it is collected. Initially, header information is sent, instructing to always reload the page rather than relying on a cached version. The second part, which sets the `$self` variable, defines it as having the value of the URL of this page. Later in the script a form can utilize the `$self` variable to actually point at this page, so the data collected can be checked and, if it is not correct, the form can be re-displayed. This tends to keep the scripting quite compact and together on one page rather than chaining another script.

The main thing to remember when using the `header()` function is that it should be sent as the first output to a page or an error will be given such as 'headers already sent' which, of course, they would have been!

14.4 Browser Detection

The range of devices with browsers is increasing so it is becoming more important to know which browser and other details you are dealing with in order to appropriately render the Web page.

The browser that the server is dealing with can be identified using:

```
$browser_ID = $_SERVER['HTTP_USER_AGENT'];
```

`$_SERVER` is a global array with lots of useful information stored in it about the server's current status. The `HTTP_USER_AGENT` is an environment variable in the table that contains this information.

Typical responses to querying the variable are as follows:

```
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/312.1
            (KHTML, like Gecko) Safari/312
```

This shows the Safari browser running on Mac OS X.

```
Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.7.6)
            Gecko/20050225 Firefox/1.0.1
```

Scanned by CamScanner

This shows Firefox running on Mac OS X.

```
Mozilla/5.0 (compatible; Konqueror/3.4; Linux) KHTML/3.4.0 (Like
            Gecko)
```

This shows Konqueror running on (Slackware) Linux.

So, all you need to do is check the returned string containing the browser information (and OS if you like!).

14.5 String Manipulation with PHP

A lot of what is done in PHP is based around the manipulation of strings, whether it be input from the user, databases or files that have been written.

To begin with, it's possible to think of a string in PHP as an array of characters so it's possible to say:

```
$mystring = "this is a test";
print $mystring[0]; // which will print 't'
print $mystring[3]; // which will print 's'
```

This uses an index (a number in square brackets) as an offset from the beginning of the string, starting at 0.

Often, there are specific things that need to be done to a string, such as reversing, extracting parts of it, finding a match to part, or changing case. You could do these manually with the aid of loops but, as usual with PHP, many of these things are already done for you in the form of functions.

Table 1 shows some of the string functions available within PHP. Often you need to process input from the user and check they have filled a form out correctly. For example, the following code fragments are for the registering of a user to a Web site, which in this case uses a plain text file. A database or *encrypted* file would normally be used.

The first thing you may want to do is strip out leading and trailing spaces to get at the actual string you want to process from the incoming form data:

```
// collect the data from the html form
$name_in = $_POST[$name1];
$email_in = $_POST[$email1];
$ident_in = $_POST[$ident1];

// check if filled out
if (empty($name_in) || empty($email_in) || empty($ident_in)) {
        die("Please fill out all of the required fields");
}

// trim whitespace either end of the string
$name = trim($name_in);
$email = trim($email_in);
$ident = trim($ident_in);
```

If you wanted to limit their registration to a specific email address group, possibly belonging to a particular organization, you could use a check on the email address supplied:

```
// check contains organization tail
    if (!strstr($email,"@mdx.ac.uk")) {

        die("Must have a valid mdx address!");
    }
```

Scanned by CamScanner

Notice here, a check is being performed with the `strstr()` function to see if the email part exists in the entered string. Sometimes it's useful to check the length of a string:

```
// check length of id
if (strlen($ident)!=7) {
    die("Please fill in the correct details");
}
```

The `strlen()` function is used to check the length of an identity code that should be a specific length.

Table 1 Some string functions in PHP

String Function	*Purpose*
`strlen($string)`	Returns length of string
`strstr($string1, $string2)`	Finds string 2 inside string 1; if not found returns false, otherwise returns the portion of string 1 that contains it
`strpos($string1, $string2)`	Finds string 2 inside string 1; if not found returns false, otherwise returns the index position where the substring begins
`substr($string, startpos)` `substr($string, startpos, endpos)`	Returns string from either start position to end or the section given by `startpos` to `endpos`
`strtok($string, $delimiters)` `strtok($delimiters)`	Splits a string up into tokens. Initial call contains both string and delimiters, further calls only require delimiters as string is stored

`trim($string)`
`rtrim($string)`
`ltrim($string)`
	Trims away white space, including tabs, newlines and spaces, from both beginning and end of a string. `ltrim` is for the start of a string only and `rtrim` is for the end of a string only

`strip_tags($string, $tags)`	Strips out HTML tags within a string, leaving only those within `$tags` intact
`substr_replace($string1, $string2, start, end)`	Similar to `substr` but replaces the substring with string 2 at start through to optional end point. Returns transformed string
`str_replace($search, $replace, $string, $count)`	Looks for `$search` within `$string` and replaces with `$replace`, returning the number of times this is done in `$count`
`strtolower($string)`	Converts all characters in `$string` to lowercase
`strtoupper($string)`	Converts all characters in `$string` to uppercase
`ucwords($string)`	Converts all first letters in a string to uppercase
`explode($delimiters, $string)`	Similar to `strtok` but breaks string up into an array at the points marked by the delimiters
`stripslashes($string)`	Strips out inserted backslashes

Scanned by CamScanner

Checkpoint!

- It's possible to write your own functions using the `function` keyword.
- The `header()` function allows control over the visitor's browser via the HTTP header information. It is possible to reload or redirect, for example. Make sure that this is the first output to the page though!
- When developing your own applications and Web pages it's important to know what kind of browser your visitor is using, and also the device itself, to render display information correctly. The browser may be one for a mobile device such as pico, for example.
- There are a great many functions built into PHP for the manipulation of strings.

14.6 Files

If all our checks were okay, so far, we may like to see if that user identity is already in a list we have stored. If we have a list that contains:

```
username: identity : email : other_data
```

the following code fragment could be used to check if the identity is already present:

```
$result = -1;

// reads the members list into an array, $data
$data = file("members.txt");

// iterate through file
foreach ($data as $line)
{
    $arr = explode(":", $line);

    if ($arr[1] == $num)
    {
    $result=1;
    break;
    }
}
```

Firstly, a flag variable is set to -1, and then a text file is read in. The text file is then looped through line by line. When a line is extracted, it is then broken up into an array using the `explode` function, which breaks each part of the line up when it finds a colon. The array entry at position 1 (remembering that arrays begin at 0!) is the identity code. If they match then `$result` is set to 1, otherwise it will remain at -1 when it comes out of the loop. Also notice here that a `break` command is used to stop the loop going through any more lines of data.

Once the identity number has been checked to see if it is already in the list and is not present, you would probably want to add the new user details in. We know the order of our data is username, identity, email ... separated by colons. This is the format that the data needs to follow, and to do this the concatenation operator is needed:

```
$newreg = $name.":".$ident.":".$email;
```

Scanned by CamScanner

To write this back into our list of users we could add it to the $data array or append it back to the file:

```
$filename ="members.txt";

// open for read/write, create if doesn't exist
$file = fopen($filename, "a");

$newreg = $student.":".$studid.":".$email."\n";

fwrite($file, $newreg);
fclose($file);
```

The file name is the same as before but this time we are using the `fopen()` function with an append attribute, `"a"`, as the *mode* string. The `fopen()` function can be used to write, `"w"`, read, `"r"` and append, `"a"`. Placing a `"+"` with these, such as `"a+"`, allows reading and writing to take place. If a file does not exist for a write operation then it is created. It is also possible to specify whether a binary file (such as a gif picture) is being written or a text file (as is here). To do this, include `"b"` for binary or `"t"` for text within the mode string, e.g. `"wb"`, write binary. In this case we want to append, or add to the end of the file. The new line is then made up and written. Finally, the file is closed.

14.7 Passwords

A password system can be fairly easy to set up. There are a couple of approaches to this: asking the user for a password and confirming it, or emailing them with an initial password that they can change. The second approach has the benefit of checking their email address is valid as they won't be able to log in if they have given someone else's address.

If passwords are involved then it is not a good idea to have them just written clearly in plain text; luckily PHP has a good choice of mechanisms to encrypt. Probably the easiest way is the `crypt()` function:

```
$encrypted = crypt($password);
```

This is a one–way process because the original password cannot be derived, even if given the encrypted password, which could be stored in a file. To authorize a person to access the Web site using a password encrypted in this way:

```
// $passwd = encrypted password from file
// $tstpass = incoming password attempt from user input
result = -1;
if (crypt($tstpass,$passwd) == $passwd) {
        // there is a match!
   $result = 1
}
```

is simply tests the encrypted, stored password against the user input from a form. If there is a match then `result` variable is changed. Other encryption options exist such as `md5()`.

Scanned by CamScanner

14.8 Email

An email can be sent to the address the user supplies to verify the user's desire to sign up to a Web site or newsletter; it could, after all, be a bogus registration. In PHP this is quite easy to do, although there must be a mail server running on the host! Here is an example code fragment of how to send an email:

```
#recipient's email address
$to = $email;

#subject of the message
$re = "Purr-fect Cats Web Registration";

#message from the feedback form
$msg = "Hello, you are now registered to the Purr-fect
     Cats Website.\nInitial password: $passwd\n\n";

#set the From header
$headers = "From: webmaster@Purr-fect-Cats.co.uk";

#send the email now...
mail($to,$re,$msg, $headers);
```

14.9 Uploading

Once you have a site up and running you may like to have the capability for files to be uploaded. The first stage of this could be a simple HTML form:

```
<form name="subform" action="upload.php" method="post"
                enctype="multipart/form-data">
<input type="hidden" name="max_file_size" value="40000">
<br><br>Select the file to upload:<br>
<input type="file" name="file" size=50> <br>
<input type="submit" value="Upload File">
</form>
```

This form attempts to set a maximum file size but this technique should not be relied on, as it is fairly easy to bypass, it being only HTML! The input type of file enables a file to be selected. Once the submit button is clicked on, the file is copied up to the server. The PHP can then process it further before it is copied from a temporary upload area into wherever the PHP may want it. This is a good time to check that it's the kind of file you want uploading to the server so make sure your various checks are done, such as file extension/type and size. Here is the basic fragment for uploading:

```
// checks if upload via POST
if (is_uploaded_file($_FILES['file']['tmp_name']))
{

  //Get the Size of the File
  $size = $_FILES['file']['size'];
```

Scanned by CamScanner

```
    //Make sure that $size is not too big
    if ($size > 1000000)
    {
        echo 'File Too Large.';
        exit( );
    }
    if($file_name !="")
    {
        $tot_name = "uploads/$file_name";
        // actually put the file in the correct place
        // from the temporary upload area
        copy ("$file",$tot_name ) or die("Could not copy file");
    }
    else
    {
        die("No file specified");
    }
    } else {
        die("File upload error");
}
```

Note how the initial name of the file and file size are taken from an associative array (with key–value) `$_FILE`. The actual upload is completed when the copy statement has been successfully executed. Three variables are created by the HTML form when a file is submitted: `file_name; file_size` and `file_type`. These are accessible in your PHP by the same names with the usual dollar sign in front.

What if the person uploading the file actually sends a server side script? They could execute it and possibly gain access to all of your system. For example, it's relatively easy to write a script that will display all the files in the Web area and delete or modify them if they want! There are a few things you can do to help stop any uploading data being executed with evil intent:

1. Don't let anonymous people upload at all; make people have an account or register and keep a log of activities/uploads.
2. Allow only certain kinds of specific file, check file extensions and run special functions to disable any script that may be present.
3. Load files to a non-obvious named directory, hopefully inaccessible from the outside world (unless that's what you want!).
4. Check the size of the file being uploaded.

Checking file extensions is fairly simple:

```
if (strstr($file_name, ".exe"))  {
    die("Not accepted: must not be an exe file");
}
```

Using the function `strip_tags`, as described in Table 1, will delete any embedded HTML code, such as that which allows script to be embedded. You may like to add other ways of checking any uploaded code, especially if it can be viewed afterwards.

Scanned by CamScanner

Checkpoint!

- Always verify and process incoming data from a user. Check for white space around the information or simply use trim. Does it conform to the data you expect – if you are expecting an email address does it look like one? If it is an identity code is it the right length and does it consist of the correct characters?
- PHP allows file handling and manipulation.
- If you are using a file, remember to maintain the order of data.
- Any security-critical information stored in such a file should be encrypted using PHP's various functions such as `crypt()` or `md5()`.
- It's possible to communicate with a visitor through email within PHP as long as a mail server is set up on the same host.
- PHP will control file uploads from a user. As with any information being uploaded or passed to the server this should be carefully processed to check it is what you expect. Simple checking may include file size restrictions, format and type.
- The variables `$file_name`, `$file_size` and `$file_type` are automatically created when you upload a file.

Concept Check

- Attempt to develop a registration page for a Web site that allows a user to enter their details, process them and write details to a file.
- Look in one of the PHP online manuals and find out how to use encryption such as `md5()`.
- Add a password and use appropriate encryption.
- Add a log-in page that verifies a user by simply checking the name and password.
- Finally, add the ability to upload `'.txt'` files only.

14.10 Examples

This section provides examples based on PHP concepts and features covered earlier in this chapter for readers to understand how to apply them in real situations. The attempt is also done to provide enhanced part of some features which are felt necessary in present Web development scenarios.

Arrays

Listing 7 gives the example of creating a numeric array (array that uses integer indices) and then applying the PHP built-in array functions on that.

```
<html>
<body>
<?php
   //Program to demonstrate array functions
   $data = array(34,23,12);
   print("The original array :");
   print_r($data);
   array_push($data,35);
   print("<br />The array after adding value at end :");
```

Scanned by CamScanner

```
    print_r($data);
    array_shift($data);
    print("<br />The array after removing value from start :");
    print_r($data);
    sort($data, SORT_NUMERIC);
    print("<br />The array after sorting :");
    print_r($data);
    shuffle($data);
    print("<br />The array after shuffling :");
    print_r($data);
    $pos = array_search(12,$data);
    print("<br />The value found at index : $pos");
    $sum = array_sum($data);
    print("<br />The sum of all the elements : $sum");
?>
</body>
</html>
```

Listing 7 Using array functions with numeric array.

The above PHP program produces the following output:

```
The original array : Array ( [0] => 34 [1] => 23 [2] => 12 )
The array after adding value at end : Array ( [0] => 34 [1] => 23 [2] =>
12 [3] => 35 )
The array after removing value from start : Array ( [0] => 23 [1] => 12
[2] => 35 )
The array after sorting : Array ( [0] => 12 [1] => 23 [2] => 35 )
The array after shuffling : Array ( [0] => 12 [1] => 23 [2] => 35 )
The value found at index : 0
The sum of all the elements : 70
```

Listing 8 shows how to create and use associative array (array with key-value pairs). The example also demonstrates the use of `ArrayIterator` (introduced in PHP 5.0) object. The object contains methods: `current()`, `reset()`, `next()`, `prev()`, `end()`, `each()`, `key()`, `valid()`, `rewind()`.

```
<html>
<body>
<?php
    //Program to demonstrate array functions
    $data = array('k1'=>10,'k2'=>20,'k3'=>30);
    print("The original array :");
    print_r($data);
    print("<br />The value of key 'k2' is :");
    print($data['k2']);
    //using array iterator
```

Scanned by CamScanner

```
    $iterator = new ArrayIterator($data);
    while($iterator->valid()) {
       print("<br />");
       print("The value of key ".$iterator->key()." is ".
       $iterator->current());
       $iterator->next();
    }
?>
</body>
</html>
```

Listing 8 Using associative array.

The above program produces the following output:

```
The original array : Array ( [k1] => 10 [k2] => 20 [k3] => 30 )
The value of key 'k2' is : 20
The value of key k1 is 10
The value of key k2 is 20
The value of key k3 is 30
```

Let us see how we can create two-dimensional arrays in PHP. A two-dimensional array is an array of arrays. Listing 9 creates a two-dimensionl array with first row having 3 elements and second row having 2 elements. Then the array is printed using nested for each loop. Finally, the largest from the array is found and displayed.

```
<html>
<body>
<?php
   //create 2D array
   $a = array(array(8,12,15),array(5,14));
   //print the array
   print("<h3>Array is</h3>");
   foreach($a as $r) {
      foreach($r as $c) {
         print($c);
         print(" ");
      }
      print("<br />");
   }
   //find largest
   $max = $a[0][0];
   foreach($a as $r) {
      foreach($r as $c) {
         if($c >= $max)
            $max = $c;
      }
```

Scanned by CamScanner

```
    }
    print("<h3>Largest value : ".$max."</h3>");
?>
</body>
</html>
```

Listing 9 Working with two-dimensional array.

The above program produces following output:

```
Array is
8 12 15
5 14
Largest value : 15
```

Remember that all the array functions can only be applied to two-dimensional array as also to individual rows as it is array of arrays. For example, to add one more element in second row at the end we can use

```
array_push($a[1], 7);
```

Functions

The functions in PHP are almost similar to C functions. Listing 10 shows how we can write recursive function and call it.

```
<html>
<body>
<?php
   //recursive fucntion to find factorial
   function fact($n)
   {
      if($n == 0 || $n == 1)
      return 1;
      else
        return $n*fact($n - 1);
   }
   //caling function fact()
   print("Factoral of 5 is : ".fact(5));
?>
</body>
</html>
```

Listing 10 Factorial using recursive function.

The above program produces the following output:

```
Factoral of 5 is : 120
```

Listing 11 shows how we can pass arguments to a function as references. Using reference arguments function can change original variables. They are useful to return more than one value from the function.

Scanned by CamScanner

```
<html>
<body>
<?php
   //function to exchange numbers
   function exch(&$a, &$b)
   {
      $temp = $a;
      $a = $b;
      $b = $temp;
   }
   //caling function exch() with reference
   $x = 10;
   $y = 20;
   print("Original numbers: X = ".$x." Y = ".$y);
   exch($x,$y);
   print("<br />");
   print("After exchange: X = ".$x." Y = ".$y);
?>
</body>
</html>
```

Listing 11 Using reference parameters.

The above program produces the following output:

```
Original numbers: X = 10 Y = 20
After exchange: X = 20 Y = 10
```

Strings

The PHP supports rich set of string functions. Listing 12 demonstrates the use of some of the important string functions:

```
<html>
<body>
<?php
   //Program to demonstate string functions
   $str = "Learning PHP is fun";
   print("String: ".$str);
   print("<br />Length of string : ".strlen($str));
   $str1 = str_replace("fun", "easy", $str, $count);
   print("<br />Replaced 'fun' by 'easy' ".$count." times");
   print("<br />String: ".$str1);
   $str2 = strtoupper($str);
   print("<br />Uppercase String: ".$str2);
   $pos = strpos($str, "PHP");
```

Scanned by CamScanner

```
    print("<br />'PHP' found at position: ".$pos);
?>
</body>
</html>
```

Listing 12 Using string functions.

The above program produces the following output:

```
String : Learning PHP is fun
Length of string : 19
Replaced 'fun' by 'easy' 1 times
String : Learning PHP is easy
Uppercase String : LEARNING PHP IS FUN
'PHP' found at position : 9
```

Processing Headers and Form Data

The PHP gets the user input using HTML form which collects the data from user and passes it to the server using either HTTP get or post method. Apart from user data, each request also contains headers containing useful information. The PHP contains in-built associative arrays to store these details. The `$_SERVER[]` array contains the information regarding headers, paths and script locations. We can access these data using the key names as index to the `$_SERVER[]` array. Listing 13 reads some details from `$_SERVER[]` array and prints it.

```
<html>
<body>
<?php
   //get the details
   print("Script Running: ".$_SERVER['PHP_SELF']."<br />");
   print("Browser: ".$_SERVER['HTTP_USER_AGENT']."<br />");
   print("Server Machine: ".$_SERVER['SERVER_NAME']."<br />");
   print("Server Software: ".$_SERVER['SERVER_SOFTWARE']);
   print("<br />");
   print("Protocol : ".$_SERVER['SERVER_PROTOCOL']."<br />");
 ?>
</body>
</html>
```

Listing 13 Using `$_SERVER[]` array.

The above program produces the following output:

```
Script Running : /Listing13.php
Browser : Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0)
Gecko/20100101 Firefox/13.0.1
Server Machine : localhost
```

Scanned by CamScanner

```
Web Server Software : Apache/2.4.2 (Win32) OpenSSL/1.0.1c PHP/5.4.4
Protocol : HTTP/1.1
```

The user data passed using get or post method can be accessed using `$_GET[]` or `$_POST[]` array, respectively. The `$_REQUEST[]` array works for both get and post methods. Listing 14 (`Listing14.html`) contains the HTML from collecting the data about user for example, name, age and location.

```
<html>
<body>
<h1>Give your details</h1>
<form action="Listing15.php" method="get">
    Name :
    <input type="text" name="name" /><br /><br />
    Age :
    <input type="text" name="age" /><br /><br />
    Location :
    <input type="text" name="loc" /><br /><br />
    <input type="submit" value="Submit" /><br />
</form>
</body>
</html>
```

Listing 14 HTML form with get method.

As can be seen from above HTML form, the method used is get. The output of the `Listing14.html` is shown in Fig. 1 after entering the data.

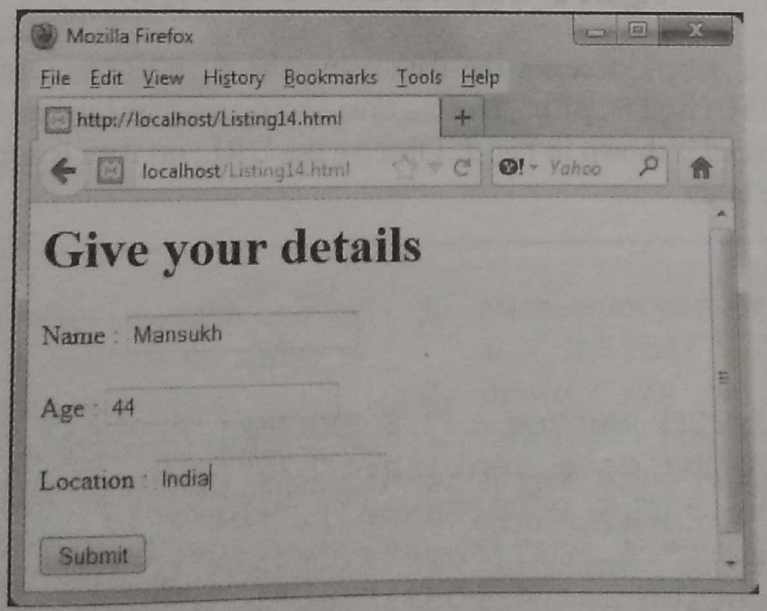

Figure 1 HTML form displayed by `Listing14.html`.

Listing 15 (`Listing15.php`) which is target program for the form in Listing 14 uses `$_GET[]` array to access the data sent.

Scanned by CamScanner

```
<html>
<body>
<?php
   print("<h3>HTTP Method : ".$_SERVER['REQUEST_METHOD']."</h3>");
   print("<h3>The data sent is:</h3>");
   print("Name : ".$_GET["name"]."<br />");
   print("Age : ".$_GET["age"]."<br />");
   print("Location : ".$_GET["loc"]."</br />");
?>
</body>
</html>
```

Listing 15 Accessing data using $_GET[] array.

Once the form in Fig. 1 is filled up and Submit button is pressed, the `Listing15.php` is executed whose output is shown in Fig. 2. Observe the URL in address bar having data appended as query string as method is get.

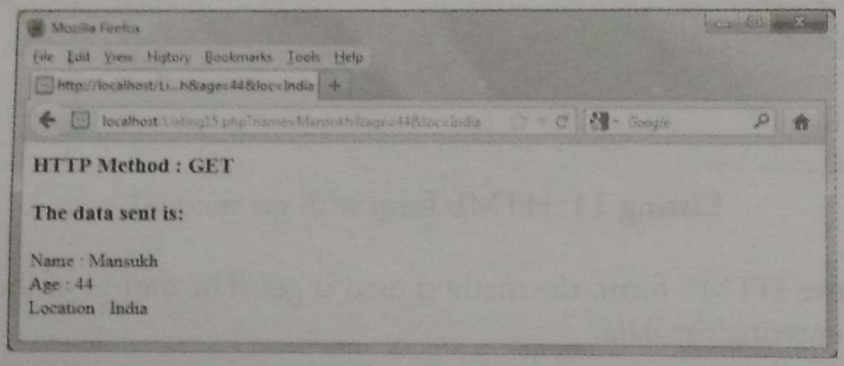

Figure 2 Output of `Listing15.php`.

Listing 16 (`Listing16.php`) accesses the data using the `$_POST[]` array. To execute it, change attributes `action="Listing16.php"` and `method="post"` in Listing 14 and then run and press Submit. The output produced is shown in Fig. 3. Observe the URL; no data is appended, as in post method it is passed in the message body.

```
<html>
<body>
<?php
   print("<h3>HTTP Method : ".$_SERVER['REQUEST_METHOD']."</h3>");
   print("<h3>The data sent is:</h3>");
   print("Name : ".$_POST["name"]."<br/>");
   print("Age : ".$_POST["age"]."<br/>");
   print("Location : ".$_POST["loc"]."</br/>");
?>
</body>
</html>
```

Listing 16 Accessing data using `$_POST[]` array.

Scanned by CamScanner

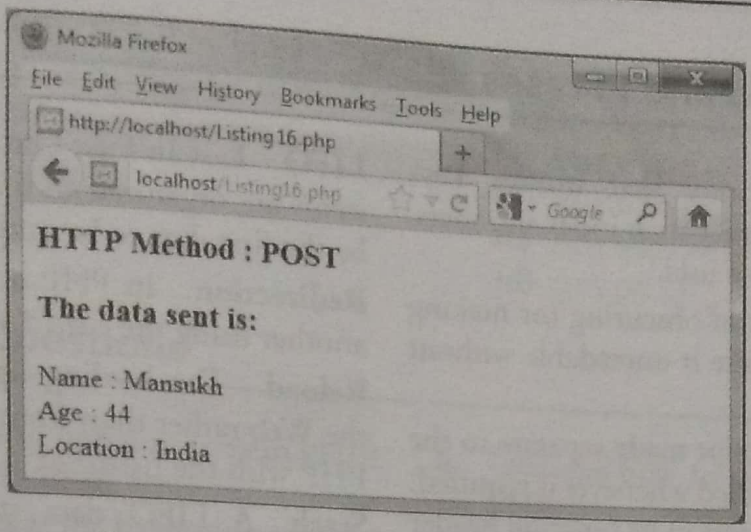

Figure 3 Output of `Listing16.php`.

An issue in using `$_GET[]` or `$_POST[]` method is that the PHP script must use one of them depending upon whichever method is used by the HTML form for collecting and sending the data. The PHP uses `$_REQUEST[]` array to solve this problem as it works with both the get and post methods. If you replace `$_GET[]` in Listing 15 and `$_POST[]` in Listing16 by `$_REQUEST[]`, they work fine.

Checkpoint!

We now know:

- PHP also supports multi dimensional arrays.
- PHP functions are similar to C functions and support recursion as well as reference parameters.
- PHP provides rich set of built-in functions for processing strings.
- PHP provides in-built global array `$_SERVER[]` to store the headers, paths and script locations.
- The user data sent using HTML form with either get or post method are accessed using `$_GET[]` and `$_POST[]` arrays, respectively.
- The `$_REQUEST[]` array works with both get and post methods.

Summary

1. Arrays are very useful data structures and are made even more so in PHP by the ability to use associative arrays with key-values.
2. There are several useful functions available to manipulate arrays, which include the ability to treat the structure as a stack.
3. Loop structures exist, such as `foreach`, which allow iteration through arrays.
4. Form information is easy to access, process and verify.
5. User-made functions can be made using the function keyword.
6. Mechanisms exist for browser recognition such as `$_SERVER['HTTP_USER_AGENT']`.
7. Browsers can be controlled through the `header()` function.
8. A great many string functions exist to allow manipulation.
9. Specialized functions exist, such as `encrypt()` and `md5()`, for encryption.
10. It is possible to allow files to be uploaded from the user but, as with any incoming information, they should be carefully verified to check they are what is expected.

Scanned by CamScanner