MT834
Unit 3
Server-side programming
Course team
Developer: Jenny Lim, Consultant
Designer: Chris Baker, OUHK
Coordinator: Dr Li Tak Sing, OUHK
Member: Dr Andrew Lui Kwok Fai, OUHK

External Course Assessor
Prof. Mingshu Li, Institute of Software, Chinese Academy of Sciences

Production
ETPU Publishing Team

Copyright © The Open University of Hong Kong, 2004.
All rights reserved.
No part of this material may be reproduced in any form by any means without permission in writing from the President, The Open University of Hong Kong.

The Open University of Hong Kong
30 Good Shepherd Street
Ho Man Tin, Kowloon
Hong Kong
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>1</td>
</tr>
<tr>
<td>Objectives</td>
<td>2</td>
</tr>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Submitting data to the server</td>
<td>5</td>
</tr>
<tr>
<td>Fill-in forms</td>
<td>5</td>
</tr>
<tr>
<td>GET versus POST</td>
<td>8</td>
</tr>
<tr>
<td>Executing applications</td>
<td>17</td>
</tr>
<tr>
<td>Passing data to the program</td>
<td>18</td>
</tr>
<tr>
<td>Programs versus scripts</td>
<td>19</td>
</tr>
<tr>
<td>Web server extension mechanisms</td>
<td>19</td>
</tr>
<tr>
<td>Developing applications</td>
<td>23</td>
</tr>
<tr>
<td>Tour of programming technologies</td>
<td>23</td>
</tr>
<tr>
<td>Sources of code</td>
<td>23</td>
</tr>
<tr>
<td>Writing applications in PHP</td>
<td>26</td>
</tr>
<tr>
<td>Tutorial</td>
<td>27</td>
</tr>
<tr>
<td>Implementing your applications</td>
<td>30</td>
</tr>
<tr>
<td>Programming considerations</td>
<td>35</td>
</tr>
<tr>
<td>Security issues</td>
<td>35</td>
</tr>
<tr>
<td>Debugging errors</td>
<td>37</td>
</tr>
<tr>
<td>Summary</td>
<td>41</td>
</tr>
<tr>
<td>Feedback to activities</td>
<td>43</td>
</tr>
<tr>
<td>Suggested answers to self-tests</td>
<td>44</td>
</tr>
<tr>
<td>References</td>
<td>49</td>
</tr>
</tbody>
</table>
Overview

Web server software is highly specialized at doing one thing: serving documents in response to HTTP requests. We often associate the term ‘documents’ with pre-built files that already exist in the Web server’s file system. In many cases, however, the Web server is required to go beyond merely serving such static documents.

Search engines, stock trading, online shopping and discussion forums cannot be built using static, pre-built pages alone. Dynamic and interactive applications such as these can only be made possible if the Web server is able to accept inputs from the user, process those inputs and generate the resulting documents dynamically.

This is achieved by implementing Web server extensions which allow the server to go beyond its basic functionality of serving documents. Web server extensions allow the Web server to execute other programs in response to incoming requests. In effect, the Web server acts as the middleman between the Web client and an executable program that runs on the server, hence, the term ‘server-side programs’. Server-side programs are also called ‘gateways’, because they open up an entire world of non-Web resources to users.

Server-side programming and HTML forms are the two major technologies used to build dynamic and interactive Web applications. We begin the unit by looking briefly at what server-side programs, scripts and forms can do. Next, we discuss the specific mechanism by which data is collected from HTML forms, packaged within an HTTP request and handed off to a server-side program for further processing.

Web servers have different ways of providing the context or environment in which programs can execute, and these will be covered next. There are also some options that you need to understand when choosing a programming language for the Web applications you develop. Then we move on to writing simple Web programs in PHP. You will need to set up your local Apache Web server with the capability to execute PHP scripts in order to complete some of the later activities. We will also direct you to online tutorials for PHP so that you can easily get up to speed on this language.

The final section of the unit discusses the hazards of executing server-side programs. We will examine some security precautions that you should take if your Web server allows users to request dynamic documents. We conclude by giving some hints on what to do when a Web program does not work correctly.

This unit should take you around four weeks. Please do not hesitate to contact your tutor if you have any problems. In addition, you should take advantage of the WebCT environment to communicate with other students and tutors as you work through the course.
Objectives

1. Describe the HTML constructs for developing online forms.

2. Develop and implement Web applications using a suitable programming language.

3. Analyse how the Web server executes server-side scripts.

4. Examine the meaning and structure of the HTTP messages generated when server-side scripts are invoked.

5. Identify the security hazards that come with running server-side programs, and explain how these risks can be minimized.

6. Introduce a wide range of alternative programming technologies for developing Web applications.
Introduction

When the World Wide Web emerged in the early 1990s, it included a simple but versatile communications protocol standard (HTTP), a universal, cross-platform client (the Web browser), and a set of reliable and well-written network servers (the httpd server). In addition, early Web servers were equipped with a server extension protocol called the Common Gateway Interface (CGI). This protocol extends the Web server’s basic functionality by giving it the capability to execute programs in response to specific user requests.

Furthermore, CGI was not restricted to any particular language. Programmers could choose among the command shell, PERL, Python, REXX, Visual Basic, or a traditional compiled language. A new breed of software developer, the ‘Web programmer’, appeared. Such programmers specialize in building server-side applications that run over the Web.

We use the term ‘server-side’ programming to differentiate this from Web programs that are executed within the Web browser, or the ‘client-side’. Server-side programs are also commonly called scripts, and the difference between programs and scripts will be discussed in a later section. However, they are equivalent in the sense that they both contain programming logic that can be executed on the Web server.

Web programs, or server-side programs, open up an entire world of resources to your users beyond the pre-built, static documents stored in the Web server’s file system. Aside from requesting the URL of a file, a Web browser can also request the URL of a resource that is really an executable program. In response to this, the server runs the requested program and returns the resulting HTML page to the client. The end user views the HTML result in the browser, perhaps without even realizing that it was constructed in real-time by an executable program on the server.

The Web server provides an environment for these programs to execute through server extension mechanisms. CGI is just one of the available mechanisms.

![Diagram of How a Web server processes a request for a dynamic webpage](image)

**Figure 3.1** How a Web server processes a request for a dynamic webpage

To introduce you to some Web applications which are made possible through Web server extensions and server-side programming, we’d like you to visit a few sites.
Activity 3.1

Visit and explore the following Web-based applications. Make a note of the file extensions that are used on the pages you visit. (Hint: Most of these file extensions won’t be .html anymore, which indicates that these pages are not static HTML pages.)

1. Try different criteria as you search for items using Ebay’s Advanced Search function:
   http://pages.ebay.com/search/items/search_adv.html

2. Take the online quiz on W3Schools to assess your knowledge of HTML:
   http://www.w3schools.com/html/html_quiz.asp

3. Search for movie listings and buy tickets on the Broadway Circuit’s e-commerce site:
   http://www.cinema.com.hk

4. Renew your library books and view items that you’ve currently checked out from the Hong Kong Public Library:
   http://www.hkpl.gov.hk

And of course, don’t forget our very own Online Learning Environment (OLE) here at the Open University!

By now, it must be quite obvious that the Web would be a much less useful and interesting place without the capability to run server-side programs. Do the following self-test to determine how well you appreciate the power of server-side programming.

Self-test 3.1

Identify the major functions that are performed by server-side programs or scripts on the websites you visited.
Submitting data to the server

Before we talk about writing programs that can generate dynamic Web documents like the ones we saw in Activity 3.1, we have to first understand how inputs are sent from the Web browser to a server-side program. For example, when you add an item to a shopping cart or enter a search term in a search engine, these inputs must be sent to a server-side program by the browser on your behalf.

If you’re thinking that it must somehow be done with HTTP, you’re quite right! You also need a mechanism for collecting the data in the Web browser, and this is done through HTML forms.

You are probably very familiar with forms, whether paper-based or computerized. Forms are everywhere in our lives, from birth certificates to school report cards, from credit card applications to income tax returns. Forms are also widely used to collect data on the Web.

This section describes how data can be collected from end-users via HTML fill-in forms. Next, it describes how the collected data is included within an HTTP request message sent to the server.

Fill-in forms

HTML forms provide a rich set of data input fields and features that are very similar to those found on common paper forms. In fact, an HTML form looks just like any other webpage, except it has places where the user can enter data or areas to click on.

![SEARCH FORM](image)

**Figure 3.2** A simple form with one text field and one submit button

HTML forms are easy to build. User controls such as buttons, text fields, radio buttons, check boxes and drop-down lists can be included in your webpages through HTML tags. The next reading presents an overview of the tags that are used to construct HTML forms.
To recap from the previous reading, the FORM tag is used to specify a fill-out form within an HTML document. Data fields are specified within a form using the INPUT, SELECT and TEXTAREA tags. Within the FORM tag itself, these are the attributes that you should take note of:

- The METHOD used to send the user input to the server. The most common choices are GET and POST. GET is the default used if the METHOD attribute is not provided.

- The ACTION which specifies the URL to which the user input will be sent. The URL points to the executable program that will serve the request:

  ```html
  <FORM METHOD="POST" ACTION="inquiry.php">
  <!-- CONTENTS OF FORM WILL BE PLACED HERE -->
  </FORM>
  ```

Now let’s look more closely at the INPUT tag. Each INPUT tag has a name, type and value attribute. The name is the variable name associated with the data by the server-side program, type describes the kind of data field used, and value contains the default value to be shown in this field, if any.

```html
<FORM METHOD="GET" ACTION="inquiry.php">
<input type="text" name="txtAuthor">
<input type="submit" name="submit" value="GO">
</FORM>
```

Figure 3.3  The form from figure 3.2 and the HTML code behind it
It’s a good practice to assign meaningful names to your form fields since these are the same names which will be used by the server-side program to retrieve their values. By doing so, your programs can be made self-documenting and easier to understand. For example, a good name for a text field which accepts email addresses is email or txtEmail (e.g., txt is a commonly used prefix for text variables).

Now that you know how HTML forms and the fields within them are constructed, you will be asked to create a fill-in form for ABC Books in the next activity. Please take some time to select the most appropriate form control for each entry.

**Activity 3.2**

**Part 1** — Create an inquiry form for ABC Books that accepts the following inputs:

1. Title — choices are Mr, Mrs, Miss and Dr — required
2. Last Name — required
3. First Name — required
4. Phone Number
5. Email — required
6. Type of Inquiry — choices are Bookseller, International Orders, Corporate Orders, Copyright Permissions, and General Web Site Inquiry.
7. Text of Inquiry

**PART 2** — Test that the form works, by going through the following steps:

1. View your form within the Web browser to make sure it displays correctly.
2. Enter data or click on each of the input fields individually.
3. Test that the **RESET** button is able to reset the form fields to their original values.
4. Finally, click on the **SUBMIT** button. At this point, the form should merely reload itself since we haven’t told it where to send the data yet.

You may view the model answer on the course website.
GET VERSUS POST

It’s actually possible to send data to a server-side program without using any forms. This is done by attaching the information to the end of the URL of the requested program as a series of name-value pairs. A question mark (?) separates the URL from the submitted data.

The term ‘query string’ refers to the name-value pairs that are sent to the server-side program for processing, and its format is as follows:

name1=value1&name2=value2&name3=value3……and so on….

Here’s an example of a URL which sends the query string txtAuthor equals Twain to the script inquiry.php:

http://plbpc001.ouhk.edu.hk/abc/inquiry.php?txtAuthor=Twain

In the next example, the query string consists of two input values: bookID and quantity. Note how the name-value pairs are separated by the ampersand (&) character.


Since we are typing the URL and query string ourselves, the browser uses the GET method by default. In fact, you also use the GET method whenever you click on a hyperlink or type a URL (without a query string) into a browser.

If you’ll recall from Unit 2, an HTTP GET message includes the URL of the requested resource in the starting line, along with the method (e.g., GET) and the HTTP version used.

In the next activity, you will try out this method of submitting data to a server-side program.

Activity 3.3

We will be using the online HTTP viewer at http://www.rexswain.com/httpview.html.

1. Enter the following URL into the online HTTP viewer form:


2. Examine the resulting HTTP request that is sent to the server.

You’ve now seen one of the simplest ways to submit data to the server: attaching the name-value pairs to the end of a URL and sending this combined string within an HTTP GET request. However, this approach
can quickly become unwieldy if you have to submit a lot of name-value pairs to the server-side program.

GET and POST can both be used to send data which is entered via fill-in forms. Just indicate which method you want to use within the METHOD attribute on the <FORM> tag. The main difference between the two is in how the form data is included within the HTTP request message sent to the server. Just like the query string, the form data will be encoded as a series of name-value pairs, separated by the ampersand (&) character:

name1=value1&name2=value2&name3=value3&….

When the GET method is used, the data entered into the form is always appended to the URL for that request. Here’s how the initial line in an HTTP request would look when the form from figure 3.3 is submitted by the user:

GET inquiry.php?authorID=51005&submit=GO HTTP/1.1

The first part of the URL contains the name (inquiry.php) of the script to be executed. Two name-value pairs are sent to the script: the author ID entered by the user and the value of the submit button.

The HTTP specification does not state any formal limits to the number of name-value pairs that may be included in a query string, but in actual practice, Web servers are not able to handle exceptionally long query strings (e.g., length > 4096 bytes).

Now let’s look at the POST method. With POST, form inputs are sent within the entity body of the HTTP request. In other words, the form data is sent as a document from the browser to the server, rather than as part of the URL. This allows bigger blocks of data to be sent to the server more conveniently. In cases when binary data (e.g., an image file) needs to be sent to the server, the POST method is your best choice.

Here’s the same request shown in figure 3.3, but reconstructed using the POST method:

POST inquiry.php HTTP/1.1
Host: plbpc001.ouhk.edu.hk
Content-type: application/x-www-form-urlencoded
Content-length: 25

authorID=51005&submit=GO

Activity 3.4 gives you a closer look at how an HTTP request containing user inputs is formed. We will use a little program called Spy to listen in on the communication between your browser and your local Web server. The following activity will guide you through installing the Spy program and setting up your browser to make it work.
Activity 3.4

Part 1 — Install the Spy program and configure your Web browser to use it.

1. Download the Spy program from the course website:
   http://plbpc001.ouhk.edu.hk/~mt834

2. Decompress the program using Winzip or an equivalent. The program is written in Java, so your computer must be equipped with the Java Runtime Environment (JRE) in order to execute it. If necessary, you can download the JRE (Java 2 Standard Edition) from http://java.sun.com/j2se/1.4.2/download.html.

3. After you have done this, open a DOS prompt. Navigate to the directory where you extracted the files from Spy.zip. Run the program by typing java Spy.

The following window should appear.

![Figure 3.4 Configuring Spy](image)

The Spy program will listen for HTTP requests at port 8123 and then redirect these requests to the corresponding sites, acting as a proxy. Spy will intercept all the HTTP messages passing through it and display them within its output window.

4. In order to use this program, you should configure your browser to use Spy as an HTTP proxy. Step 5 describes the steps for IE, and Step 6 describes the steps for Netscape Navigator.

5. If you are using IE, the proxy can be set using the following menu options:

   Tools → Internet Options → Connections → LAN Settings
If your browser is already using another proxy server, write down the port and address of the proxy server on a piece of paper so you can restore this setting later on.

Tick the checkbox for the **Proxy server**, and enter **localhost** in the **Address** and **8123** in the **Port** fields.

![Local Area Network (LAN) Settings](image)

**Figure 3.5** Local area network settings

If you were using a proxy server originally, you should configure Spy to use this proxy server as well. Go back to the Spy window shown in figure 3.4, then enter the address and port of the original proxy server you were using.

6 If you are using Netscape, the proxy can be set by using the following link:

   **Edit → Preferences → Advanced → Proxies**

Choose manual proxy configuration and click the **View** button to display the window where you will configure Netscape to use Spy as the proxy server.
Figure 3.6  Local area network settings

If you were using a proxy server originally, you should configure Spy to use this proxy server as well. Go back to the Spy window shown in figure 3.4, then enter the address and port of the original proxy server you were using.

The next figure describes how Spy sits between your Web browser and Web server, and how HTTP requests and responses pass through Spy first before arriving at their final destination.

Figure 3.7  The Spy program is a proxy server listening to port 8123 on your machine. Note that if you were using another proxy server prior to Spy, then this proxy server will be located between Spy and the Web server in the above diagram.

Now type the following URL into your browser to see the HTTP messages that are being exchanged:

http://plbpc001.ouhk.edu.hk
You should see that three connections were made by the browser to the Web server order to retrieve all the files needed to render this webpage. The three requests are:

```
GET http://plbpc001.ouhk.edu.hk/ HTTP/1.0
GET http://plbpc001.ouhk.edu.hk/icons/apache_pb.gif HTTP/1.0
GET http://plbpc001.ouhk.edu.hk/powered_by.gif HTTP/1.0
```

Spy displays each HTTP exchange (i.e., request and response) in a new tabbed window. When retrieving the page above, three new tabbed windows are shown, confirming that there were three separate requests and connections needed to build a single webpage. As discussed in Unit 2, HTTP/1.1 now supports persistent connections, which allow multiple files to be retrieved in a single connection. The version of HTTP used will depend on the outcome of the browser and server negotiation.

**Part 2** — Examine the HTTP messages when submitting data via GET.

1. A working version of ABC Books is available at:
   
   `http://plbpc001.ouhk.edu.hk/abc/unit3`

   Open the homepage in your Web browser, then click on **Authors** from the menu.

2. The author page contains a list of all the authors published by ABC Books. Click on any of the author names and examine the HTTP messages shown by the Spy program.

**Part 3** — Examine the HTTP messages when submitting data via POST.

1. From the ABC Books homepage (`http://plbpc001.ouhk.edu.hk/abc/unit3`), click on **Contact** from the menu.

2. The contact page includes an HTML form where users can enter and send their inquiries to ABC Books. Fill in the contact page and submit the form to the server. Once again, examine the HTTP messages shown by the Spy program.

*Note:* There is feedback on this activity at the back of the unit.

Table 3.1 shows a comparison between the GET and POST methods. You will note that the only advantage GET has over POST is that it can be invoked without a form. It is therefore the only choice if you want to submit data to a server-side program without using a form.
### Table 3.1 Comparison between GET and POST methods

<table>
<thead>
<tr>
<th></th>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit on the length of query string</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Can only be invoked inside a form</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Will the query string be shown on the browser?</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

You should use POST when sending binary (e.g., non-ASCII) data or large amounts of data. In addition, both Internet Explorer and Netscape display the URL when they try to retrieve the corresponding document. If the GET method is used, the query string attached to the URL can be seen by the user. For security and privacy reasons, it may not always be desirable for submitted data to be viewed in this manner. Imagine what would happen if users could see their usernames and passwords in the browser’s URL after they log in!

Another difference between GET and POST lies in how the server-side program will eventually receive the data from the Web server. This will be discussed in a later section called ‘Executing applications’.

### URI-encoding

You’ve seen that ampersands (&) have a special meaning within a query string because they are used to separate the different input items being sent to the server. But what would happen if an ampersand also appears as part of the names or values in a query string? Clearly, there must be a way to differentiate them from ampersands that are merely acting as separators.

There are other characters which have special meaning within a URL as well, such as colon (:), forward slash (/), question mark (?), and percent (%). All these characters may appear either as part of the URL or as part of the query string. How does the Web server tell the difference?

The differentiation is done via URI-encoding, which is performed by the Web browser on your behalf. URI-encoding replaces any special characters that may appear within the names or values coming from an HTML form. This is done automatically by the Web browser when the form is submitted or when the URL is entered. Here are the rules followed in URI-encoding:

- The ASCII characters a through z, A through Z, and 0 through 9, the symbol @, the asterisk (*), the underscore symbol (_), the hyphen (-), and the full stop (.) remain the same.

- The space character ( ) is converted into a plus sign (+).

- ASCII characters which are not allowed in the URL may be represented by a percent sign (%) immediately followed by two hexadecimal digits (0 to 9, A to F) giving the ISO Latin 1 code for
that character, and are converted into the character string %xy, where xy is the two-digit hexadecimal representation of the lower 8-bits of the character.

Table 3.2 Common characters that must be URI-encoded

<table>
<thead>
<tr>
<th>ASCII character</th>
<th>URI-encoded hexadecimal equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>%25</td>
</tr>
<tr>
<td>&amp;</td>
<td>%26</td>
</tr>
<tr>
<td>(</td>
<td>%28</td>
</tr>
<tr>
<td>)</td>
<td>%29</td>
</tr>
<tr>
<td>\</td>
<td>%2C</td>
</tr>
<tr>
<td>.</td>
<td>%2E</td>
</tr>
<tr>
<td>=</td>
<td>%3D</td>
</tr>
<tr>
<td>@</td>
<td>%40</td>
</tr>
</tbody>
</table>

You may have realized by now that URI-encoded characters will also need to be decoded on the other end. That is, when the server-side program accepts the query string, it first needs to split the data by the ampersands. Then, for each name=value pair that it gets, the program must URI-decode the name, and then the value, before it can begin processing. Fortunately, most server-side languages already have built-in routines or programming objects that retrieve the name-value pairs and decode them for you.

In fact, if you revisit the HTTP request generated in Part 3 of Activity 3.4, you’ll see that some of your form inputs may contain URI encoding within them.

Take the following self-test to assess your understanding of fill-in forms, the GET and POST HTTP methods, and URI-encoding.

**Self-test 3.2**

1. Identify and describe the different types of data fields that can be contained within an HTML form.
2. List some advantages and disadvantages of the GET method.
3. List some advantages and disadvantages of the POST method.
4. How would the following query string look after it has gone through URI-encoding?

   ```
   name=Melissa Joy O’Connor
   company = Melissa Joy & Associates
   email=melissa_joy@hotmail.com
   ```
We have now finished our discussion on how query strings and requests for Web scripts are formed. Let’s now move on to look at how such a request is executed by the server when it arrives.
Executing applications

Despite the wide range of tasks that programs can perform, they all work the same way from the point of view of the Web server. The httpd daemon starts the script and passes data from the browser to the server-side program. After the server-side program executes, it passes the output (usually an HTML document) to the server, and the server then sends this document out over the network to the browser.

Here are the basic steps that must be performed by the httpd daemon:

1. determine that the request is for an executable program rather than a document;
2. locate the program and verify that its access permissions allow it to be executed by a Web user;
3. start the program and pass any inputs from the Web client to it;
4. read the output from the program and pass it back to the client (if something goes wrong while the program is executing, an error message should be sent to the client); and
5. close the network connection properly when the program finishes execution.

Now let’s go through some of the highlights in more detail. For example, how does the Web server know whether an HTTP request is for a document or an executable program?

The httpd daemon usually requires scripts to be located in a particular directory or set of directories. For example, the Apache Web server is often configured so that the directory cgi-bin contains executable programs. Here is how the directive may look within the Apache configuration file (httpd.conf):

```
ScriptAlias /cgi-bin/ "/*/var/www/cgi-bin/"
```

The server can also be configured to allow any file with a particular file extension, such as files ending in .cgi, to be executed:

```
AddHandler cgi-script .cgi
```

Aside from the server configuration options above, server-side programs also need to have execute permission turned on. This is done differently depending on the operating system used. The chmod command is used to make programs executable within the UNIX environment. Access permissions are set for the owner of the file, the user group that the owner belongs to, and everyone else.

The following command will allow everyone to execute inquiry.php, including Web scripts:

```
chmod a+x inquiry.php
```
Passing data to the program

Once the Web server delegates a request to another executable program, it passes information to the program within a set of environment variables. The environment variables contain the query string sent by the user (if the HTTP GET method is used), the HTTP request header values, and some additional values from the Web server.

Here’s the partial output of a server-side program which reads in and prints out all the environment variables passed to it.

```
<address>Apache/2.0.49 (Red Hat Linux) Server at <a href="mailto:tili@ouhk.edu.hk">tili@ouhk.edu.hk</a> Port 80</address>
```

Aside from environment variables, data is also sent via the standard input stream when the HTTP POST method is used. This is another difference between GET and POST: instead of reading the query string from an environment variable, the program must read the CONTENT-LENGTH environment variable, and then read in this number of bytes from the standard input to get the submitted data.

Fortunately, popular Web programming languages such as PHP already come with built-in, pre-defined variables that contain user inputs and relevant information from the Web server environment. You won’t have to read data from the input stream later on when you write your PHP scripts; the PHP engine does all this for you beforehand.
Programs versus scripts

A program is simply a set of instructions given to a computer to perform. Scripts are similar to programs because they are also made up of instructions or code to be executed on a computer. However, there is a basic difference in the way programs and scripts are executed.

Programs and scripts need to be converted from human-readable into machine-readable form before they can be executed by your computer’s processor or CPU. In the case of programs written in C, C++ or Java, the conversion process is performed before the program runs, by a highly specialized software called a compiler. The compiler creates the machine language version of the program in a separate file. On the Windows platform, these machine language files often end with the .exe extension. Microsoft Word and Internet Explorer are popular examples of compiled (.exe) programs.

With scripts, there is no need to translate the entire program into machine language first before it can be executed. The translation process takes place every time the script is run, by another specialized software called an interpreter. The interpreter is installed on the Web server, and it is invoked by the server to execute requested scripts as necessary. Interpreted programs run more slowly than compiled programs because the code is translated to machine language line-by-line as they are being executed.

Table 3.3  Popular examples of compiled and interpreted languages

<table>
<thead>
<tr>
<th>Programs (compiled)</th>
<th>Scripts (interpreted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java, C, C++, Cobol, Fortran, Visual Basic</td>
<td>JavaScript, Perl, PHP, VisualBasic Script (VBScript)</td>
</tr>
</tbody>
</table>

Compiled programming languages have a more complex syntax, and require more strict programming practices. However, they execute faster since the code has been entirely translated into machine language beforehand. They also come with a wider and more powerful set of instructions and functions, so most sophisticated applications are still developed using compiled languages.

We’ll concentrate on scripting languages in MT834 since they’re easier to learn and do not require a compilation step. They are also adequate for implementing most of the functions that exist behind dynamic websites.

Web server extension mechanisms

The Web server software provides the environment in which server-side programs can execute, accept user inputs and pass back the generated responses. The specific mechanism used to achieve this varies among different Web servers, but here are the most common models:
1 Common Gateway Interface (CGI);
2 Application Programming Interface (API);
3 embedded interpreter; and
4 Java servlets.

**Common Gateway Interface (CGI)**

CGI defines a standard way for how external programs are to be started by the Web server, and how data is passed between the Web server and the program. It was one of the earliest mechanisms for interfacing Web server software with external gateway programs. Due to its simplicity and flexibility, CGI is universally supported in most Web servers.

The Web server passes information from the Web client to CGI programs via environment variables and the standard input stream (`stdin`). CGI scripts send the results of their processing back to the server by writing to standard output. CGI is not a programming language, rather, it defines a standard interface that can be used by Web servers and external programs to communicate with each other. Therefore, CGI programs can be written in any language that is available on the system where they run.

The primary disadvantage of CGI is that it starts a new, separate process every time a script or program is requested. For example, if ten users are simultaneously accessing `inquiry.php`, the Web server has to launch ten separate processes to serve each request, all running the same program. Every process consumes resources such as disk space, processor time, and main memory, ultimately competing with the Web server software itself. The system must also keep track of all the resources used by each process and free up these resources when the process terminates. Creating, tracking, and deleting processes involves lots of system overhead.

It’s also resource-intensive for CGI scripts to be unloaded after just one execution. This means that common tasks, such as establishing and closing database connections, must be performed for every request.

**Application Programming Interface (API)**

Web server APIs were created as an early alternative to CGI. APIs allow Web programmers to extend the Web server’s functionality by linking new modules directly to the Web server’s executable program. In effect, these additional modules run as part of the Web server itself, not as a separate process. Depending on the literature and computing platform, Web server extensions using APIs are also called ‘server plug-ins’, ‘Dynamic Link Libraries’ (DLLs), or ‘modules’.

Three of the most popular Web server APIs are ISAPI (Internet Server Application Programming Interface), NSAPI (Netscape Server Application Programming Interface) and the Apache server API. Programs written to a server API are obviously more efficient because they are loaded into the Web server’s process space and they remain
resident in memory. They can also access more powerful functions due to their tight coupling with the Web server and the operating system.

The downside is that API executables are more complex to program. API extensions are multithreaded and have to pay close attention to memory protection. When errors occur, they can also potentially crash the Web server itself since they share the same process space. API modules are also platform dependent. For example, ISAPI extensions will run only on Microsoft’s Web server, and Apache modules will run only on Apache. Programs written for a specific server API will require extensive revisions in order to work on a different server platform.

Despite the drawbacks, Web server APIs are your best choice if you want to get the most power and performance out of your programs.

**Embedded interpreter**

Many Web developers prefer to use scripting languages because they are easier to learn and use. However, running server-side scripts using CGI can place an enormous load on the server.

With CGI, the Web server launches a new process which runs the interpreter, which in turn executes the code on a line-by-line basis. And don’t forget, the interpreter has to be loaded into and removed from memory every time the script is executed. This is a tremendous waste of resources!

In order to get better performance from Web servers, there are now API modules or DLLs which allow embedded interpreters to run as part of the Web server. This means the interpreter is loaded into main memory only once, and it remains active for as long as the Web server itself is up and running. This saves the overhead of starting and stopping the interpreter every time.

Embedded interpreters often come with other features designed to improve performance. For example, the PHP embedded interpreter also offers code caching: scripts are loaded and compiled only once, when the Web server is first started. The scripts are served from the Web server’s cache for subsequent requests. Running pre-compiled code results in faster execution times than running interpreted code.

The `mod_perl` and `mod_php` Apache modules are examples of embedded interpreters. Whenever possible, it’s recommended to use embedded interpreters over plain CGI when running server-side scripts. This is the approach we shall use for ABC Books, our case study website.

**Servlets**

Just like CGI scripts, servlets are programs running on the Web server which can respond to HTTP requests. Servlets are written in the Java programming language, and their main advantage over CGI is that only
one copy of a servlet class is needed to service multiple requests. Servlets are multithreaded, which means that the processor handles new, incoming requests by launching lightweight Java threads instead of a separate operating system process. Servlets also remain in memory after they finish processing, so they can store information in between requests, such as results of previous computations.

Unlike CGI, support for Java servlets is not universal. For example, Microsoft’s IIS does not include native support for Java servlets. However, there are add-on engines available that let IIS support Java Servlets. Java servlets can also be executed using a stand-alone server, such as Apache Tomcat.

---

**Self-test 3.3**

1. What kind of data is passed by the Web server to the server-side program?

2. Compare the advantages/disadvantages of using interpreted versus compiled languages for server programming.

3. Compare the advantages/disadvantages of the different Web server extension mechanisms we just discussed.
Developing applications

We’ve spent most of this unit discussing the inner workings of server-side programs: how data is submitted to them, how the Web server provides an execution environment for them, and how they send outputs back to the client. Some of you may be wondering if you’ll ever get a chance to write your own code 😊. You’ll have plenty of chances to do this soon. I promise!

In the meantime, let’s take a tour of the numerous competing server-side programming technologies that are available. We’ll also explore the vast archives of code that are accessible online. This code can either be customized or incorporated into your own programs for free, saving you lots of time and effort in your development work.

Tour of programming technologies

Although we will focus on PHP in this unit, there are many other widely-used technologies that allow a Web server to serve dynamic data. The next reading presents you with a good overview of the most popular server-side technologies on the market: CGI/Perl, Active Server Pages (ASP), Cold Fusion (CFM), PHP, and Java.

Just a few notes on the reading before you proceed. Perl is often associated with CGI because a large number of CGI scripts are written in Perl, but if you’ll recall, CGI is a standard interface which is not restricted to just Perl. CGI scripts can be implemented in many different languages. Also, Perl can now be executed via CGI and via an embedded interpreter in the Web server, so the performance problems linked to Perl in the reading are not entirely true anymore.

Reading 3.2

Yank, K (2001) ‘Which server-side language is right for you?’

Note: This is a relatively long reading, so if you just want the highlights, you can read ‘Advantages’, ‘Disadvantages’ and ‘Drawbacks’ at the end of each page first. You can always go back and read the details at a later time.

Sources of code

While developing your own server-side programs, it’s a big help to know that there are many online repositories from which you can borrow or purchase pre-written code. In fact, there’s hardly any reason to write Web programs from scratch nowadays.
The code within these repositories is provided by programmers who want to generate extra income, to publicize their skills, or to simply help others. The available programs are usually classified according to their use. For example, if you’re looking for PHP code to display the current date/time on a webpage, you can be more efficient by restricting your search within a category that reflects this.

![Figure 3.9 Categories of PHP code from Hotscripts.com](image)

We’ll explore some online sources of code in the next activity. As you go through these websites, you’ll see that there’s a wide variety of downloadable code on offer, ranging from simple validation code to entire applications such as discussion boards and shopping carts.

And don’t forget, aside from the Web, you can also build your applications using code samples from Web programming books and tutorials.

**Activity 3.5**

1. Here are three online code repositories for PHP. Spend some time viewing the available categories and reading the descriptions of some downloadable code packages:

• PHP Builder at http://www.phpbuilder.com/snippet; and

• PEAR at http://pear.php.net (click on **Browse All Packages**).

2 Most of the code on PEAR is object-oriented. When using one of their class packages, you will normally instantiate an object for that particular class and call its methods within your code. We will take a closer look at using the PEAR DB package to perform database access in **Unit 5**.

For Hotscripts and PHPBuilder, the code is mostly of the kind that you can cut and paste into your programs or modify for your own use.

3 Now let’s download some code that will display a random quote on a webpage. This is a nice feature that we will add later on to the ABC Books homepage. Here’s the URL of the script:


4 Open the ZIP file that you just downloaded and skim through the files included within. There is a file called `quote.dat` which contains the quotes which will be randomly displayed. We want to make use of quotes from authors who are published by ABC Books instead, so download and replace this with `quote.dat` from the course website.

5 Remember where you stored these files since you’ll be using this in a later activity.
Writing applications in PHP

PHP is a powerful server-side scripting language whose syntax draws upon C, UNIX shell scripting languages, and Perl. It is also an open-source technology with a very active support community.

PHP was designed purely for the creation of dynamic webpages. For this reason, it is incredibly easy to do common, Web-related tasks in PHP. Things such as reading user inputs from the Web server, displaying output to the Web browser, and sending out email can be coded in just a few lines.

Furthermore, it is an embedded language. This means that the PHP code is embedded directly into the HTML, wrapped in a simple `<?php ....... ?>` tag. Here’s a very basic PHP script which displays the message *Hello World* in the Web browser.

```html
<html>
<head>
  <title>PHP Test</title>
</head>
<body>
<?php echo "<p>Hello World</p>"; ?>
</body>
</html>
```

*Figure 3.10  Code behind hello.php*

When the PHP interpreter parses `hello.php`, it looks for special tags which tell it where the PHP code is located within the page. As shown in figure 3.10, the opening tag used is `<?php`. The parser then executes all the code it finds, up until the next PHP closing tag, which is `?>`.

There can be several blocks of PHP code embedded within an HTML page. The parser continues executing any PHP code that it finds, and the results are written back into the HTML page. Everything outside the PHP tags is left alone, while everything inside is executed as code.

PHP has grown in popularity among Web programmers because of its power and ease of use. As of this writing, PHP usage is growing at a rate of 15% each month, and is in use on at least seven million domains (Netcraft Survey). This is about 20% of all domains registered so far. And this figure does not yet include PHP installations on intranets and private Web servers!

If you have not done so yet, you will install PHP on your local Apache Web server in the next activity.
Activity 3.6


2. Follow the steps for installing PHP on your local Web server.

3. Now test that your installation works by running phpinfo.php (code can be downloaded from the course website). Remember to make the script executable first before running it. If all goes well, this script will display information regarding your PHP installation and the Web server environment.

4. Once you’ve confirmed that phpinfo.php runs correctly, create a folder called abc in your DocumentRoot directory. This is the folder where you will install the ABC Books website.

5. Download abcbooks_v1.zip from the course website and unzip the files into the abc folder. The website is structured as follows:

![Website Structure Diagram]

**Figure 3.11** ABC Books website structure

6. Visit all the pages on the site to make sure they display correctly. These files all come with the file extension .php, but if you’ll view the source code behind them, you’ll see that these pages are all made up of HTML. There is no programming code in them as of yet; you will add this later on.

Tutorial

Now that you’ve got PHP installed and running on your Web server, let’s get started on learning PHP. If you’ve got a programming background in another language, specially C, UNIX shell scripting or Perl, then you should be able to pick up PHP fairly easily. In most of the activities we do, you will be provided with pre-written code that you just need to customize or tweak slightly.
Although you will not be required to write code from scratch in MT834, it helps a great deal if you get some exposure to basic PHP syntax early on. You may skim through the next reading which covers the following fundamental topics:

1. basic syntax;
2. types;
3. variables;
4. constants;
5. expressions;
6. operators;
7. control structures; and
8. functions.

**Reading 3.3**


*Note*: Skim-read the sections on ‘Basic syntax’ up to ‘Functions’. You may also find it useful to bookmark this page as a reference.

Now answer the following questions to test your grasp of PHP’s fundamentals.

**Self-test 3.4**

1. Aside from `<?php . . ?>`, what else can we use to denote blocks of PHP code within an HTML page?

2. What is the output of the following code?

   ```php
   $lastname = "Lim";
   $firstname = "Jenny";
   echo "Your name is $lastname, $firstname.";
   ```

3. Define a PHP array which contains the values of the seven days of the week, starting from Sunday.

4. What will be the output of the following code?

   ```php
   function foo ($string1, $string2)
   { 
   ```
return $string1 . $string2;
}　　
echo foo("Merry Christmas !");

5 Rewrite the following if statement as a switch statement.

```php
if ($fruit == "apple") {
    print "You ordered Number 1";
} elseif ($fruit == "blueberry") {
    print "You ordered Number 2";
} elseif ($fruit == "orange") {
    print "You ordered Number 3";
}
```

6 Which two PHP functions are used to write HTML output? Is there any other way to include HTML output in a PHP script aside from this?

7 List three superglobal arrays used by the PHP engine to pass data to a PHP script.

In the next tutorial, you will create some basic PHP scripts to perform common, everyday tasks such as displaying output to the browser and accepting data from input forms and the Web server environment. If any of the syntax used is unclear to you, you can always use Reading 3.3 as a reference.

---

**Activity 3.7**

This activity is divided into three parts. For each part, go through the assigned reading first, and then build a small program that uses the principles learned from the reading.

**Part 1**


After completing the reading, create and test a PHP script which uses the `echo` function to display an HTML level-four heading (`<H4>`) containing the text *I Love To Program!*

**Part 2**


Create and test a PHP script based on Example 2-5 of the tutorial you just read. This script will detect whether a user is using the Internet Explorer browser or not.
Part 3


Create an HTML form based on Example 2-6 and a PHP script based on Example 2-7 of the tutorial, ‘Dealing with forms’.

The form must use the POST method to send its outputs to your PHP script. Test that the PHP script is able to display the inputs sent by the user.

Implementing your applications

Now that you’ve warmed up to PHP, let’s bring the power of server-side programming to our case study website, ABC Books. Here are the three applications which we shall add to the site, which consists entirely of static pages at the moment:

1. display a random quote from an ABC Books Author on the homepage;
2. process and accept inquiries on the Contact Page; and
3. search for books by a specific author from the Author Page.

The next sections will guide you as you work through these applications.

Random quote

ABC Books wants to display short excerpts taken from their books on the homepage as a way of attracting customers. They also want different quotes to be displayed every time the user views the page, to give the impression that the site is dynamic and updated frequently.

We will adapt the code that you downloaded in Activity 3.5 in order to perform this function. Here’s an overview of the steps that will take place:

1. All the possible quotes that can be displayed will be stored in a text file on the Web server. Each quote occupies one line in the file.

2. The script reads all the quotes into an array, and then randomly selects one of the elements of the array to be displayed.

The next activity will walk you through these steps in more detail.
Activity 3.8

1 Back in Activity 3.5, you were asked to download an existing PHP script which displays random quotes on a webpage. The script comes with a text file called quotes.dat which contains all the possible quotes that can be shown. Make sure you have replaced the original version of this file with our own version, also called quotes.dat. You can download this from the course website if you haven’t done so yet.

2 Open index.php (ABC Books homepage) in a text editor. Search for an HTML comment which says Insert random quote here. Add this line of code right after the comment.

```php
<?php
    require('quotes.php);
?>
```

This line will execute the code from quotes.php and display its output on the same spot. Save the new version of index.php.

3 Make quotes.php executable through the CHMOD command:

   CHMOD a+x quotes.php

4 Make quotes.dat readable through the CHMOD command:

   CHMOD a+r quotes.php

5 Now reload the homepage and if all works well, you should see a quote from a randomly selected authors displayed on the left-hand margin. Reload the page a few more times to verify that the quotes are changing.

6 You can download the model answer from the course website. You can also view a working URL of ABC Books at http://plbpc001.ouhk.edu.hk/abc/unit3.

Accepting inquiries

In this section, we’ll build a program that will accept and process the online inquiries submitted on the site. Here are the general steps that will take place:

1 The HTML form is displayed in the browser with the required fields marked accordingly.

2 The user fills in the form and clicks on the submit button to send the data to the server.

3 The data is validated by the server-side script. If there are validation errors, send back an error message and ask the user to re-enter any
invalid entries. Fields that have already been filled in by the user should redisplay the provided values.

4 If there are no validation errors, save the user inputs into a file on the Web server.

5 Send out an email to the user confirming that their inquiry has been accepted.

6 Display a confirmation message in the browser.

I've already created the initial version of this program, and you will make further changes to it in the succeeding activity.

---

**Activity 3.9**

1 Download `contact.zip` from the course website and unzip this file into your `abc` folder. This ZIP archive contains only one file — `process_contact.php` — which already has part of the code needed to process the Contact Form.

2 Open `process_contact.php` in a text editor and read through the comments within the code to understand what it’s doing. This script contains both the code for displaying the input form and also the code for processing it.

You’ll see that some parts have been intentionally left blank, and there are comments explaining how you will complete these parts. Complete, step-by-step instructions can be found on the course website.

3 The following logic has been left for you to do. It’s a good idea for you to implement and test each change separately, building the program bit by bit as you go along. This also makes debugging much easier!

   - Validate that First Name, Last Name and Email Address have been provided.

   - Create a record containing the information that you’ve just collected. The record is made up of eight fields: the Unique ID or record key, Title, First Name, Last Name, Company Name, Phone, Email, Type of Inquiry and the Inquiry itself. The fields should be separated by the pipe (|) character.

   - Change the recipient’s email address to your own in the confirmation email message.

4 You can download the model answer from the course website. You can also view a working URL of ABC Books at http://plbpc001.ouhk.edu.hk/abc/unit3.
Simple search

Now that you’ve got your Contact Form working, let’s implement a simple search application which allows users to view all the books by a particular author. Here are the high-level steps behind this function:

1. The list of authors published by ABC Books can be found by viewing author.php. The URL for each author includes the author’s ID number within the query string. This data will be sent to the server-side script when the author’s name is clicked on.

2. Clicking on an author’s name will launch a new browser window which displays the titles written by this author. Note: This page is blank right now, and you will complete the programming logic needed to make it work.

3. The script will search for all books written by the author from the file abc_catalog.txt. A page displaying a list of these books will be sent back to the browser.

You will create this application in the next activity. Just like before, I will provide you with a partially written version of this program, and you will be given instructions on how to complete the code.

Activity 3.10

1. Download inquiry.zip from the course website and unzip this file into your abc folder. This ZIP archive contains only one file — inquiry.php — which already has part of the code needed to display the list of books by an author.

2. Open inquiry.php in a text editor and read through the comments within the code to understand what it’s doing. You’ll see that some parts have been intentionally left blank, and there are comments explaining how you will complete these parts. Complete, step-by-step instructions can be found on the course website.

3. The following logic has been left for you to do. Once again, implement and test each of the changes separately, so that you’re building on top of previously tested and working code.

   - Display an error message if the file abc_catalog.txt is not found or cannot be opened.

   - Split a record read from the abc_catalog.txt file into its component fields, using the pipe (|) symbol as the delimiter. The fields are as follows: Book ID, Book Title, Author ID, Category, Photo, Price and Summary of Contents.
• Output any book titles which match the given Author ID to the webpage. If not a single book was found, display the message *No Titles Found* instead.

4 You can download the model answer from the course website. You can also view a working URL of ABC Books at http://plbpc001.ouhk.edu.hk/abc/unit3.
Programming considerations

Server extensibility provides your Web server software with capabilities beyond merely serving static documents. However, there is a price that must be paid for this ability.

First, you should accept the possibility that the security of your Web server could be completely compromised. That’s because server-side programs may have security flaws of their own. These loopholes may give outsiders access to the host, or even allow unauthorized changes or the removal of critical files from your system.

Next, you should realize that faulty and poorly tested programs could slow down the Web server or, in the worst case, bring the entire server down. This is why Web programmers should test and debug server-side programs very carefully and thoroughly.

Web programmers tend to focus mostly on the development work. As a result, there’s not always enough time and effort allocated to testing and optimizing the programs they produce. In this section, we will stress the importance of debugging your applications and making them as secure as possible.

Security issues

Web server extensions may come in the form of scripts, programs, and plug-ins. These mechanisms all have the potential to open up security loopholes when accessing confidential information outside the Web directory hierarchy or when they interface with other external applications, such as email programs and database servers.

The next reading urges you to consider the security issues that come with installing and running external programs on your Web server.

Reading 3.4


Note: You may begin reading from the section called ‘Why this is important’.

Reading 3.4 raises a very important point — Web developers must be very, very familiar with what their programs are doing. This is doubly important when you are running or adapting code that has been downloaded from the Internet. You must never trust your sources...
explicitly. It’s still necessary to walk through the code, examine it closely and stay alert for any possible security flaws.

Here’s another reading which shows you some code, explains why running this code could be dangerous to your Web server’s health, and how the code could be made more secure. Please note that even though the examples used are written in PHP, these practices can be adapted to server-side programs in any language.

**Reading 3.5**


Note: Start from the section ‘Protecting the server’ and read up to and including ‘Enough validation’.

Most of the dangers described above are due to the fact that the script interfaces with the UNIX command shell using the `exec()` and `popen()` functions. `exec()` will execute any command that is passed to it as if it had been entered via the command line. `popen()` also executes commands just like `exec()` but it forks a new process in order to do so. Allowing users to enter strings that will be executed by your Web server’s command shell is very dangerous and should be avoided at all costs!

Here is a summary of the recommended practices you’ve seen from these two readings:

1. Ensure that programs are only able to perform the functions they were intended for. Attackers are always looking for ways to ‘trick’ server-side programs into revealing information or executing actions that they were never meant to do.

2. Run the programs in a restricted environment. Even if attackers are able to take over your programs, you can still limit the damage that can be done. For example, if your Web server processes all run as ‘root’ or superuser, a non-authorized user who gains access to these processes might also gain root privileges.

3. Do not store any sensitive data within the Web document tree where it can be accessed via a Web server request.

In the next activity, you will modify the code behind the ABC Books Contact Form in order to make it more secure.
**Activity 3.11**

We will validate the inputs provided on the ABC Books Contact Form for any suspicious or illegal characters. For example, the pipe (|), less than (<), greater than (>) and semicolon (;) characters should never be entered as part of someone’s Last Name or First Name.

You can continue working on the code `contact.php` that you produced from Activity 3.10, or you can download the model answer from the course website first and work on this one. Complete, step-by-step instructions can be found on the course website.

Here are the validation steps we will perform:

1. Use the `preg_replace` function to strip out any illegal characters from First Name, Last Name, Phone Number, and Inquiry text. Inform the users that these characters have been removed if any are found.

2. Use the `ereg` function to validate that the email address is in the right format (i.e., somebody@somewhere.com).

**Debugging errors**

No matter what the nature of your Web application may be, it’s almost certain that your program may fail to execute properly under certain conditions. There are many possible reasons for this: a missing semicolon (;) at the end of a statement, a programming loop that doesn’t terminate, or something more subtle, such as code that doesn’t work the way you thought it would. Errors may also arise when programs interface with other applications, such as file I/O and database servers.

Whatever the cause, well-designed and well-written programs should always prepare for the unexpected. They must be able to detect error conditions on their own and handle these conditions as gracefully as possible. When an error is encountered, the program should decide whether to stop execution or to perform an alternative course of action. In all cases, errors should be logged so they can be investigated further. Adequate feedback must be given to users if necessary. There is nothing more disconcerting to a user than seeing an *HTTP 500 Server Error* message!

**Testing your programs**

It’s more convenient to test your programs from the command line first rather than from a Web browser. Fortunately, the input and output interfaces of a Web script are defined very clearly. Therefore, you can simulate the environment under which a Web server invokes the script. This means that you will feed the inputs directly to the program and accept the outputs from it yourself rather than go through the Web server.
For example, if the Web script needs to accept user inputs via the GET or POST method, you can set meaningful values in the \_GET or \_POST superglobal array within your code. You may also need to set any necessary environment variables in the \_SERVER superglobal array that are used in your Web script.

The output of the Web script is sent to standard output, usually your screen display. You can redirect the results of a PHP script to a file instead through following command:

```
php aprogram > afile
```

This command invokes the PHP interpreter to execute the program aprogram and write the HTML page that is produced to the file called afile. You should replace aprogram and afile with your actual script and file names, of course.

When testing and debugging your programs, it helps to know what kinds of errors you should be watching out for. Here are the common types of errors that you will find:

1. Syntax errors — errors which violate the spelling and grammar rules for a programming language. Syntax errors are caught when the compiler or interpreter parses the code, and they need to be fixed before the program can be executed.

   Examples: mismatched apostrophe, misspelled keyword

   /*There is a mismatched double quote.*/
   echo "A leap year has 366 days';
   /*The keyword echo has been misspelled.*/
   eco "The quick brown fox jumps over the dog";

2. Semantic errors — there are programs which observe proper syntax but contain programming constructs whose meaning is not understood by the computer.

   Examples: using the wrong operator, function that passes different number of arguments

   $a = 5.0;
   $b = "pie";
   /*Semantic error: Why do you want to multiply a string and a number ?*/
   $c = $a * $b;

3. Logic errors — errors which do not violate the syntax or semantics of a programming language, rather, they occur when a programmer writes code that behaves differently from what is expected.

   Examples: misspelled variables, wrong boundary conditions
/*Logic error: Programmer expects this loop to execute 10 times, but it will only execute for 9 times. The correct boundary condition should be $count <= 10. */
for ($count = 1; $count < 10; $count++) {
    /*do something here*/
}

4 Environmental errors — caused by unexpected conditions in the program’s external operating environment.
Examples: reading or writing files to disk

/*Environmental error: A program tries to read a file's contents without first checking that the file was opened successfully. */
$file = @fopen("nonexistentfile.txt", "r");
while ($buffer = fread($file, 100)) {
    /* Process the data read from the file. But what if the file was not opened successfully? */
}

It’s also important to test all possible paths of execution within your program. Let’s say you have the following code snippet:

$tax = .00825;
if ($tax > 0) {
    echo ("Total: " . ($_POST["price"] * $tax));
} else {
    echo ("Total: " ($_POST["price"] * "1.00"));
}

You should test the case when tax is greater than zero and also when it is less than or equal to zero in order to test all possible actions that may be taken by the program. If some portions of your code remain untested, there is always the danger that there are still errors lurking within them.

In the next activity, you will download and test some pre-written PHP scripts in your local Web server. You will then fix any errors found and re-run the corrected versions. For each one of them, identify which type of programming error was found.
Activity 3.12

Download a PHP script from the course website. Read the description of its expected behaviour from the comments within the code and from the brief summary posted on the site.

Run the program to test whether it works accordingly or not. You will use the `error_log` function to record any problems in a file for further investigation. Debug and fix the problems you find.

Complete, step-by-step instructions can be found on the course website.
Summary

In this unit, you learned how server-side programs can build webpages in real time in response to specific user requests. Server-side programming provides the following advantages:

- the ability to generate customized webpages; and
- the ability to convert content from non-Web sources (e.g., databases, external software applications) to HTML format and then display them in a browser.

HTML fill-in forms are the most popular way of collecting user information on the Web. Forms can contain different types of input fields which are familiar to most of us in our daily lives: text fields, radio buttons, drop-down menus, buttons and check boxes, etc.

Two HTTP methods, GET and POST, are used to send data to a server-side script for processing. The difference between the two lies in how the data is included within the HTTP request. The GET method appends the data to the URL of the script, while POST includes the data as a document within the HTTP request body.

There are different Web server extension mechanisms available. In this unit, we looked at Common Gateway Interface (CGI), Web Server API, embedded interpreters and Java servlets. Some mechanisms require programs to run in a separate process and others allow them to run in the same process as the Web server. Our practical work in this unit focused on using the PHP language running via an embedded interpreter.

Despite the benefits derived from Web server extensions, there is an inherent danger in allowing users to execute programs on the Web server. Here are some of the recommended precautions:

- do not run the Web server process as root or superuser;
- avoid executing other programs via a command shell within a Web script, specially where the command string is entered by the user; and
- always validate your user’s inputs and watch out for illegal or suspicious characters in their data.

Finally, developers should never underestimate the amount of effort involved in testing and debugging their programs. They should test how their programs perform, both in expected and unexpected conditions. Errors should be written to a log file so documentation exists on what went wrong. Adequate feedback should also be provided to users in the form of simple, friendly error messages.

On the practical side of things, we’ve added some useful and interesting functionality to our case study website, ABC Books. Now that you’ve got a few weeks of PHP experience behind you, you will build on this knowledge to implement online searching of their product catalog in
Unit 4. You will also learn how search engines work and how you can optimize a website’s listing on a search engine.
Feedback to activities

Activity 3.4

Part 2 — Here is the HTTP request message generated when clicking on the Author link for Sir Arthur Conan Doyle. Take note of how the user inputs are attached to the URL as a query string.

GET

GET http://plbpc001.ouhk.edu.hk/abc/styles/main.css HTTP/1.0

GET http://plbpc001.ouhk.edu.hk/abc/images/abcheader.jpg HTTP/1.0

Part 3 — Here is the HTTP request message generated when submitting the Contact Form. Take note of how the form inputs are passed within the HTTP entity body.

POST http://plbpc001.ouhk.edu.hk/abc/process_contact.php HTTP/1.0

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*

Referer: http://plbpc001.ouhk.edu.hk/abc/process_contact.php

Accept-Language: en-us

Content-Type: application/x-www-form-urlencoded

Proxy-Connection: Keep-Alive

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: plbpc001.ouhk.edu.hk

Content-Length: 182

Pragma: no-cache

selTitle=Mrs&txtFirstName=Jenny&txtLastName=Lim&txtPhone=5555-1212&txtEmail=jennylim@verdana.com&optInquiry=2&txtInquiry=What+are+your+requirements+for+book+authors+%3F&submit=Submit
Suggested answers to self-tests

Self-test 3.1

Server-side programs or scripts can perform the following functions:

• access information from non-Web sources and services from external applications, such as databases, email, news, and online transaction systems;

• allow dynamic, ongoing conversation between the user and the server. This is the true essence of interactivity, allowing users to shop and chat online, search databases, pay bills and many other possibilities; and

• build customized documents dynamically at the time they are requested. Customized documents reflect the circumstances of a particular user rather than returning the same version of a document every time.

Self-test 3.2

1 Aside from INPUT tags, drop-down lists and multiline text fields can also be included within forms using the SELECT tags and TEXTAREA tags, respectively.

This table lists and describes the most common data fields found in HTML forms.

<table>
<thead>
<tr>
<th>Input fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Accepts a line of text. Ideal for entering names, phone numbers, addresses, etc.</td>
</tr>
<tr>
<td>Password</td>
<td>Identical to a text field, except that characters typed into the field are displayed as asterisks in the browser.</td>
</tr>
<tr>
<td>Radio</td>
<td>Multiple-choice element which allows only one value to be selected. Radio buttons belonging to the same set of choices all have the same name.</td>
</tr>
<tr>
<td>Checkbox</td>
<td>Multiple-choice element which allows several values to be checked at once. Checkboxes belonging to the same set of choices all have the same name.</td>
</tr>
<tr>
<td>Hidden</td>
<td>Prevents an input element from being displayed. Any value information is still submitted with the form. Hidden fields can be used in a script-generated form to store information about the state of a transaction.</td>
</tr>
<tr>
<td>Submit</td>
<td>Submits the form’s contents for processing when clicked on.</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets all the form fields to their original values when the reset button is clicked on. Allows the user to undo any values they have already typed in and start all over again.</td>
</tr>
</tbody>
</table>
Image Displays an image-based button rather than the standard grey button.

Select Allows users to choose values from a drop-down list or menu. Select boxes can be configured to show only one line or multiple lines at once. They can also allow the user to make a single choice or multiple choices.

Textarea Accepts multiple lines of text. Ideal for entering free-format text such as comments, inquiries, and messages.

2 GET method:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can be used to pass data to the server without using a form.</td>
<td>The length of the query string is limited. The query string is displayed in the browser, which isn’t too suitable if you’re passing confidential data to the server.</td>
</tr>
</tbody>
</table>

3 POST method:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longer blocks of data are allowed because data is passed as a document, not as a string. The query string is not shown within the browser. Allows submission of non-ASCII data.</td>
<td>Data must be entered through a form.</td>
</tr>
</tbody>
</table>

4 After URI-encoding, the query string would look like this:

```
name=Melissa%20Joy%20O’Connor&company=Melissa%20Joy%20%26Associates&email=melissa_joy@hotmail.com
```

Self-test 3.3

1 The Web server can pass the following data to the server-side script:

- server environment variables (i.e., Web server software and version used, Web server’s address, remote client’s user agent or browser, remote client’s IP address, etc.);
- data attached to the end of the URL (i.e., via the GET method); and
- data within the document body of the HTTP request (i.e., via the POST method).
2 Interpreted languages:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to learn.</td>
<td>Performance may be slower because code has to be interpreted and executed line-by-line. However, if embedded interpreters are used, the performance may be almost as good as that of compiled languages.</td>
</tr>
<tr>
<td>Feature set and instructions are usually adequate for the needs of dynamic websites.</td>
<td></td>
</tr>
<tr>
<td>Embedded interpreters can now be used to decrease the overhead of launching the interpreter each time the script must be executed.</td>
<td></td>
</tr>
</tbody>
</table>

Compiled languages:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster execution time.</td>
<td>Steeper learning curve because they have more complex syntax and require more strict programming practices (i.e., such as requiring all variables to be declared).</td>
</tr>
<tr>
<td>Come with a wider and more powerful set of instructions and functions.</td>
<td></td>
</tr>
<tr>
<td>Web server extension mechanisms</td>
<td>Advantages</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CGI</td>
<td>Simple to implement.</td>
</tr>
<tr>
<td></td>
<td>Supported by most Web servers.</td>
</tr>
<tr>
<td></td>
<td>Not restricted to any programming language, so developers can use a language of their choice.</td>
</tr>
<tr>
<td>Web server API</td>
<td>Programs run as part of the Web server, leading to better performance.</td>
</tr>
<tr>
<td></td>
<td>Strongly tied to the Web server and operating system, so they can make use of more powerful and advanced features and functions.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedded interpreter</td>
<td>Interpreter runs as part of the Web server process, so it doesn’t have to be launched each time it is needed.</td>
</tr>
<tr>
<td></td>
<td>Often includes other optimization features such as interpreting a script only once, when it is initially requested.</td>
</tr>
<tr>
<td></td>
<td>Subsequent requests are served from a cached version of the interpreted code.</td>
</tr>
<tr>
<td>Java servlets</td>
<td>Servlets are only compiled once, and they remain in memory as long as the Web server is running.</td>
</tr>
<tr>
<td></td>
<td>Due to their multithreaded nature, only one copy of the servlet is needed to service all incoming requests for it.</td>
</tr>
</tbody>
</table>

Self-test 3.4

1. Aside from `<?php ... ?>`, we can use the following to denote blocks of PHP code within an HTML page:

   ```html
   <script language="php">
   //...put code here...//
   </script>
   ```

2. The output of the following code will be *Your name is Lim, Jenny.* followed by a line feed. Note that the line feed will not show up within a Web browser unless you use the `<BR>` tag.
$lastname = “Lim”;
$firstname = “Jenny”;
echo “Your name is $lastname, $firstname.”;

3 Here is a PHP array which contains the values of the seven days of the week, starting from Sunday:


4 The following code will produce an error because the function `foo` expects to receive two arguments:

```php
function foo ($string1, $string2)
{
    return $string1 . $string2;
}

echo foo(“Merry Christmas !”);
```

5 Here is the `if` statement rewritten as a `switch` statement:

```php
switch ($fruit) {
    case “apple”:
        print “You ordered Number 1”;
        break;
    case “blueberry”:
        print “You ordered Number 2”;
        break;
    case “orange”:
        print “You ordered Number 3”;
        break;
}
```

6 The `echo` and `print` functions can both be used to produce HTML output. Another way to include HTML output in a PHP script is to write the HTML directly outside of a PHP code block. Anything outside of a code block will be ignored by the parser and outputted as HTML.

7 The following table shows three superglobal arrays used by PHP to pass pre-defined variables to a script.

<table>
<thead>
<tr>
<th>_SERVER</th>
<th>Environment variables.</th>
</tr>
</thead>
<tbody>
<tr>
<td>_GET</td>
<td>Data submitted via HTTP GET method.</td>
</tr>
<tr>
<td>_POST</td>
<td>Data submitted via HTTP POST method.</td>
</tr>
</tbody>
</table>
References


