Ordering Information:

Python How to Program
The Complete Python Training Course

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ONLINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html**.

To learn more about our **Python programming courses** or any other Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training**, contact our Director of Corporate Training Programs at (978) 461-5880 or e-mail: **christi.kelsey@deitel.com.**

Note from the Authors: This article is an excerpt from Chapter 6, Sections 6.1 and 6.3 of *Python How to Program.* This article discusses the Common Gateway Interface (CGI), which allows Web servers to execute programs that generate dynamic content. We discuss the difference between client-side scripting and server-side scripting and the difference between static and dynamic Web content. We present the discussion in the context of two Python CGI programs. Readers should be familiar with basic python programming (e.g., functions, dictionaries and models) and XHTML. The code examples included in this article show readers programming examples using the DEITEL™ signature LIVE-CODE™ Approach, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

6.3 Simple CGI Script

Two types of scripting are used in Web-based applications: *server-side* and *client-side*. CGI scripts are an example of server-side scripts because they run on the server. Programmers have greater control over Web-page content when using server-side scripts, because server-side scripts can manipulate databases and other server resources. An example of client-side scripting is JavaScript. Client-side scripts can access the browser's features, manipulate browser documents, validate user input and much more.

Scripts executed on the server usually generate custom responses for clients. For example, a client might connect to an airline's Web server and request a list of all flights from Boston to San Antonio between September 19th and November 5th. The server queries the database, dynamically generates XHTML content containing the flight list and sends the XHTML to the client. This technology allows clients to obtain the most current flight information from the database by connecting to an airline's Web server.

Server-side scripting languages have a wider range of programmatic capabilities than their client-side equivalents. For example, server-side scripts can access the server's file directory structure, whereas client-side scripts cannot access the client's file directory structure.

Server-side scripts also have access to server-side software that extends server functionality. These pieces of software are called *COM components* for Microsoft Web servers and *modules* for Apache Web servers. Components and modules range from programming language support to counting the number of times a Web page has been visited (known as the number of *hits*).

Software Engineering Observation 6.1

Server-side scripts are not visible to the client; only the content the server delivers is visible to the client.

As long as a file on the server remains unchanged, its associated URL will display the same content in clients' browsers each time the file is accessed. For the content in the file to change (e.g., to include new links or the latest company news), someone must alter the file manually (probably with a text editor or Web-page design software) then load the changed file back onto the server.

Manually changing Web pages is not feasible for those who want to create interesting and dynamic Web pages. For example, if you want your Web page always to display the current date or weather, the page would require continuous updating. The alternative is to use server-side scripting to generate content dynamically, as the user visits the site. We can accomplish this with the *Common Gateway Interface (CGI)*.

CGI describes a set of protocols through which applications (commonly called *CGI programs* or *CGI scripts*) interact with Web servers and indirectly with Web browsers (e.g., client applications). A Web server is a specialized software application that responds to client application requests by providing resources (e.g. Web pages). CGI programs often generate Web content dynamically. A Web page is dynamic if a program on the Web server generates that page's content each time a user requests the page. For example, a form in a Web page could request that a user enter a zip code. When the user types and submits the zip code, the Web server can use a CGI program to create a page that displays information about the weather in that client's region. In contrast, *static* Web page content never changes unless the Web developer edits the document.

CGI is “common” because it is not specific to any operating system (e.g., Linux or Windows), to any programming language or to any Web server software. CGI can be used with virtually any programming or scripting language, such as C, Perl and Python.

The CGI protocol was developed in 1993 by the *National Center for Supercomputing Applications* (*NCSA*—**www.ncsa.uiuc.edu**), for use with its *HTTPd Web server*. NCSA developed CGI to be a simple tool to produce dynamic Web content. The simplicity of CGI resulted in its widespread use and in its adoption as an unofficial worldwide protocol. CGI was quickly incorporated into additional Web servers, such as Microsoft *Internet Information Services (IIS)* and Apache (**www.apache.org**).

Figure 6.3 illustrates the full program listing for a simple CGI script. Line 1

```
#!c:\Python\python.exe
```

is a *directive* (sometimes called the *pound-bang* or *sh-bang*) that specifies the location of the Python interpreter on the server. This directive must be the first line in a CGI script. The examples in this chapter are for Window users. For UNIX- or Linux-based machines, the directive typically is one of the following:

```
#!/usr/bin/python
#!/usr/local/bin/python
#!/usr/bin/env python
```

depending on the location of the Python interpreter. [*Note*: If you do not know where the Python interpreter resides, contact the server administrator.]

```
1  #!c:\Python\python.exe
2  # Fig. 6.3: fig06_03.py
3  # Displays current date and time in Web browser.
4
5  import time
6
7  def printHeader( title ):
8     print """Content-type: text/html
9
10 <?xml version = "1.0" encoding = "UTF-8"?>
11 <!DOCTYPE html PUBLIC
12    "-//W3C//DTD XHTML 1.0 Strict//EN"
13    "DTD/xhtml1-strict.dtd">
14 <html xmlns = "http://www.w3.org/1999/xhtml">
15 <head><title>%s</title></head>
16
17 <body>""" % title
18
19 printHeader( "Current date and time" )
20 print time.ctime( time.time() )
21 print "</body></html>"
```

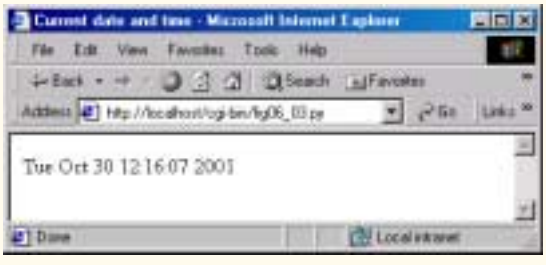

Fig. 6.3 CGI script displaying the date and time.

Common Programming Error 6.1

Forgetting to put the directive (`#!`) in the first line of a CGI script is an error if the Web server running the script does not understand the `.py` filename extension.

Line 5 **`import`**s module ***`time`***. This module obtains the current time on the Web server and displays it in the user's browser. Lines 7–17 define function **`printHeader`**. This function takes argument **`title`**, which corresponds to the title of the Web page. Line 8 contains the HTTP header. Notice that line 9 is blank, which denotes the end of the HTTP headers. The line that follows the last HTTP header must be a blank line, otherwise Web browsers cannot render the content properly. Lines 10–14 contain the XML declaration, document type declaration and opening **`<html>`** tag. For more information on XML, see Chapter 15 of the book *Python How to Program, First Edition*. Lines 15–17 contain the XHTML document header and title and begin the XHTML document body.

Common Programming Error 6.2

Failure to place a blank line after an HTTP header is an error.

Line 19 begins the main portion of the program by calling function **printHeader** and passing as an argument the title of the Web page. Line 20 calls two functions in module **time** to print the current time. Function ***time.time*** returns a floating-point value that represents the number of seconds since midnight, January 1, 1970 (called the *epoch*). Function ***time.ctime*** takes as an argument the number of seconds since the epoch and returns a human-readable string that represents the current time. We conclude the program by printing the XHTML body and document closing tags. For a complete list of functions in module **time**, visit

```
www.python.org/doc/current/lib/module-time.html
```

Note that the program consists almost entirely of **print** statements. The default target for **print** is *standard output*—an information stream presented to the user by an application. Typically, standard output is displayed on the screen, but it may be sent to a printer, written to a file, etc. When a Python program executes as a CGI script, the server redirects the standard output to the client Web browser. The browser interprets the headers and tags as if they were part of a normal server response to an XHTML document request.

Executing the program requires a properly configured server. [*Note*: In this book, we use the Apache Web server. For information on obtaining and configuring Apache, refer to our Python Web resources at **www.deitel.com**.] Once a server is available, the Web server site administrator specifies where CGI scripts can reside and what names are allowed for them. In our example, we place the Python file in the Web server's **cgi-bin** directory. For UNIX and Linux users, it also is necessary to set the permissions before executing the program. For example, UNIX and Linux command

```
chmod 755 fig06_03.py
```

gives the client the permission to read and execute **fig06_03.py**.

Assuming that the server is on the local computer, execute the program by typing

```
http://localhost/cgi-bin/fig06_03.py
```

in the browser's **Address** or **Location** field. If the server resides on a different computer, replace **localhost** with the server's hostname or IP address. [*Note*: The IP address of **localhost** is always **127.0.0.1**.] Requesting the document causes the server to execute the program and return the results.

Figure 6.4 illustrates the process of calling a CGI script. First, the client requests the resource named **fig06_03.py** from the server (Step 1). If the server has not been configured to handle CGI scripts, it might return the Python code as text to the client.

A properly configured Web server, however, recognizes that certain resources need to be processed differently. For example, when the resource is a CGI script, the script must be executed by the Web server. A resource usually is designated as a CGI script in one of two ways—either it has a special filename extension (such as **.cgi** or **.py**), or it is located in a specific directory (often **cgi-bin**). In addition, the server administrator must grant explicit permission for remote access and CGI-script execution.

The server recognizes that the resource is a Python script and invokes Python to execute the script (Step 2). The program executes, and the text sent to standard output is returned to the Web server (Step 3). Finally, the Web server prints an additional line to the output that indicates the status of the HTTP transaction (such as **`HTTP/1.1 200 OK`**, for success) and sends the whole body of text to the client (Step 4).

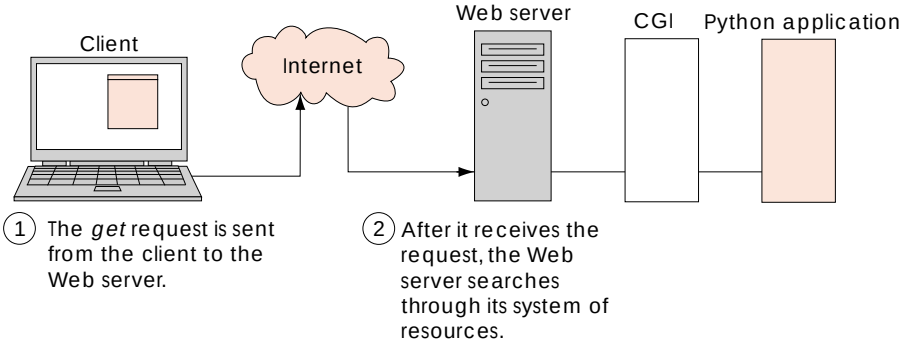

Fig. 6.4 Step 1: The **`GET`** request, **`GET /cgi-bin/fig06_02.py HTTP/1.1`**. (Part 1 of 4.)

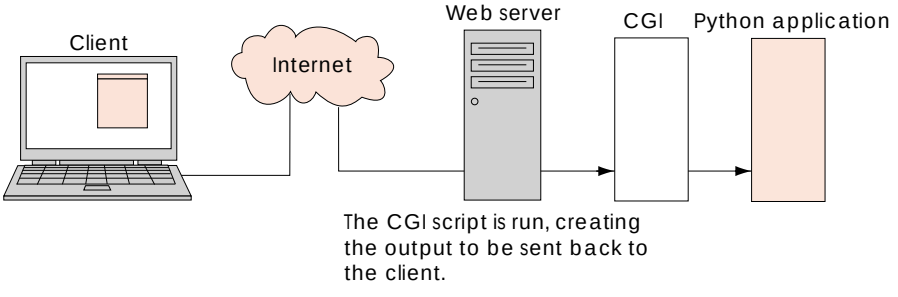

Fig. 6.4 Step 2: The Web server starts the CGI script. (Part 2 of 4.)

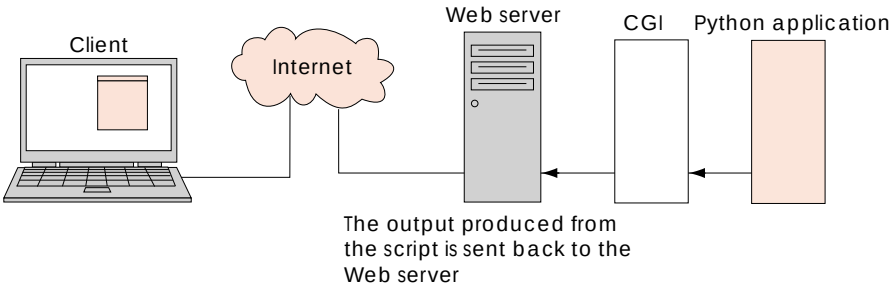

Fig. 6.4 Step 3: The output of the script is sent to the Web server. (Part 3 of 4.)

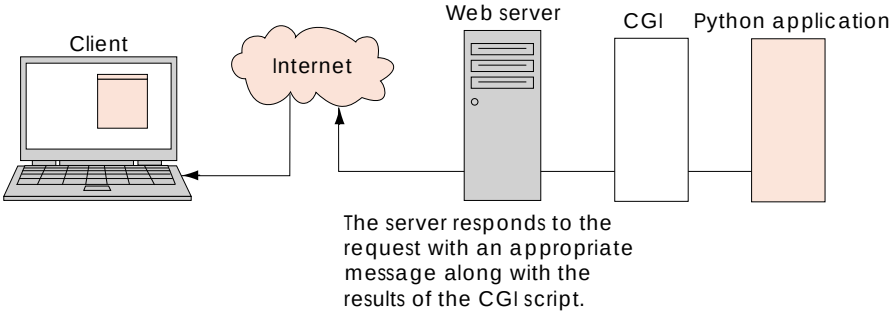

Fig. 6.4 Step 4: The HTTP response, **`HTTP/1.1 200 OK`**. (Part 4 of 4.)

The browser on the client side then processes the XHTML output and displays the results. It is important to note that the browser does not know about the work the server has done to execute the CGI script and return XHTML output. As far as the browser is concerned, it is requesting a resource like any other and receiving a response like any other. The client computer is not required to have a Python interpreter installed, because the script executes on the server. The client simply receives and processes the script's output.

We now consider a more involved CGI program. Figure 6.5 organizes all *CGI environment variables* and their corresponding values in an XHTML table, which is then displayed in a Web browser. Environment variables contain information about the execution environment in which script is being run. Such information includes the current user name and the name of the operating system. A CGI program uses environment variables to obtain information about the client (e.g., the client's IP address, operating system type, browser type, etc.) or to obtain information passed from the client to the CGI program.

Line 6 **`import`**s module ***`cgi`***. This module provides several CGI-related capabilities, including text-formatting, form-processing and URL parsing. In this example, we use module **`cgi`** to format XHTML text.

```
1  #!c:\Python\python.exe
2  # Fig. 6.5: fig06_05.py
3  # Program displaying CGI environment variables.
4
5  import os
6  import cgi
7
8  def printHeader( title ):
9     print """Content-type: text/html
10
11 <?xml version = "1.0" encoding = "UTF-8"?>
12 <!DOCTYPE html PUBLIC
13    "-//W3C//DTD XHTML 1.0 Strict//EN"
14    "DTD/xhtml1-strict.dtd">
15 <html xmlns = "http://www.w3.org/1999/xhtml">
16 <head><title>%s</title></head>
17
18 <body>""" % title
19
20 rowNumber = 0
21 backgroundColor = "white"
22
23 printHeader( "Environment Variables" )
24 print """<table style = "border: 0">"""
25
26 # print table of cgi variables and values
27 for item in os.environ.keys():
28    rowNumber += 1
29
30    if rowNumber % 2 == 0:          # even row numbers are white
31       backgroundColor = "white"
32    else:                           # odd row numbers are grey
33       backgroundColor = "lightgrey"
34
35    print """<tr style = "background-color: %s">
36    <td>%s</td><td>%s</td></tr>""" % ( backgroundColor,
37       cgi.escape( item ), cgi.escape( os.environ[ item ] ) )
38
39 print """</table></body></html>"""
```

Fig. 6.5 CGI program to display environment variables. (Part 1 of 2.)

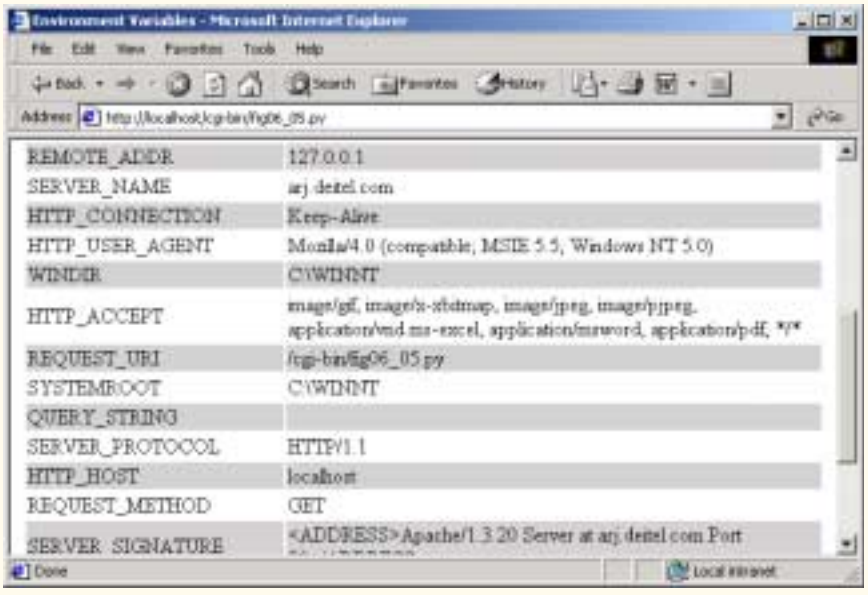

Fig. 6.5 CGI program to display environment variables. (Part 2 of 2.)

Lines 8–18 define function **`printHeader`**, which is identical to the function we defined in the previous example. The main program prints an XHTML table that contains the environment variables (lines 24–39). The ***`os.environ`*** data member holds all the environment variables (line 27). This data member acts like a dictionary; therefore, we can access its keys via the **`keys`** method and its values via the **`[]`** operator. Lines 30–33 set the background color for each row. For each environment variable, lines 35–37 create a new row in the table containing that key and the corresponding value.

Note that line 37 calls function ***`cgi.escape`*** and passes as values each environment variable name and value. This function takes a string and returns a properly formatted XHTML string. Proper formatting means that special XHTML characters, such as the less-than and greater-than signs (**`<`** and **`>`**), are "escaped." For example, function **`escape`** returns a string where "**`<`**" is replaced by "**`<`**", "**`>`**" is replaced by "**`>`**" and "**`&`**" is replaced by "**`&`**". The replacement signifies that the browser should display a character instead of treating the character as markup. After we have printed all the environment variables, we close the **`table`**, **`body`** and **`html`** tags.