Module 5: Networking

Contents

Introduction: Network Programming

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

This notes gives you understanding on most famous concept in Networking - Socket Programming.

What is Sockets?

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Sockets have their own vocabulary –

Sr. No.	Term & Description
1	**Domain** The family of protocols that is used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
2	**type** The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
3	**protocol** Typically zero, this may be used to identify a variant of a protocol within a domain and type.
4	**hostname** The identifier of a network interface – • A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation • A string "<broadcast>", which specifies an INADDR_BROADCAST address. • A zero-length string, which specifies INADDR_ANY, or An Integer, interpreted as a binary address in host byte order.
5	**port** Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The *socket* Module

To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax –

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters –

- **socket_family** – This is either AF_UNIX or AF_INET, as explained earlier.
- **socket_type** – This is either SOCK_STREAM or SOCK_DGRAM.
- **protocol** – This is usually left out, defaulting to 0.

Once you have *socket* object, then you can use required functions to create your client or server program. Following is the list of functions required –

Server Socket Methods

Sr.No.	Method & Description
1	**s.bind()** This method binds address (hostname, port number pair) to socket.
2	**s.listen()** This method sets up and start TCP listener.
3	**s.accept()** This passively accept TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods

Sr.No.	Method & Description
1	**s.connect()** This method actively initiates TCP server connection.

General Socket Methods

Sr.No.	Method & Description
1	**s.recv()** This method receives TCP message
2	**s.send()** This method transmits TCP message
3	**s.recvfrom()** This method receives UDP message
4	**s.sendto()**

	This method transmits UDP message
5	**s.close()** This method closes socket
6	**socket.gethostname()** Returns the hostname.

A Simple Server

To write Internet servers, we use the **socket** function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.

Now call **bind(hostname, port)** function to specify a *port* for your service on the given host.

Next, call the *accept* method of the returned object. This method waits until a client connects to the port you specified, and then returns a *connection* object that represents the connection to that client.

```
#!/usr/bin/python           # This is server.py file

import socket               # Import socket module

s = socket.socket()         # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345                # Reserve a port for your service.
s.bind((host, port))        # Bind to the port

s.listen(5)                 # Now wait for client connection.
while True:
   c, addr = s.accept()     # Establish connection with client.
   print 'Got connection from', addr
   c.send('Thank you for connecting')
   c.close()                # Close the connection
```

A Simple Client

Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's *socket* module function.

The **socket.connect(hosname, port)** opens a TCP connection to *hostname* on the *port*. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits –

```
#!/usr/bin/python           # This is client.py file

import socket               # Import socket module

s = socket.socket()         # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345                # Reserve a port for your service.

s.connect((host, port))
print s.recv(1024)
s.close()                     # Close the socket when done
```

Now run this server.py in background and then run above client.py to see the result.

```
# Following would start a server in background.
$ python server.py &

# Once server is started run client as follows:
$ python client.py
```

This would produce following result –

```
Got connection from ('127.0.0.1', 48437)
Thank you for connecting
```

Python Internet modules

A list of some important modules in Python Network/Internet programming.

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib

Introduction: Database Programming

The Python programming language has powerful features for database programming. Python supports various databases like SQLite, MySQL, Oracle, Sybase, PostgreSQL, etc. Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements. The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

Here is the list of available Python database interfaces: Python Database Interfaces and APIs. You must download a separate DB API module for each database you need to access.

In this topic we will see the use of SQLite database in python programming language. It is done by using python's inbuilt, sqlite3 module. You should first create a connection object that represents the database and then create some cursor objects to execute SQL statements.

Connect To Database

Following Python code shows how to connect to an existing database. If the database does not exist, then it will be created and finally a database object will be returned.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print "Opened database successfully";
```

Here, you can also supply database name as the special name **:memory:** to create a database in RAM. Now, let's run the above program to create our database **test.db** in the current directory. You can change your path as per your requirement. Keep the above code in sqlite.py file and execute it as shown below. If the database is successfully created, then it will display the following message.

```
$chmod +x sqlite.py
$./sqlite.py
Open database successfully
```

Create a Table

Following Python program will be used to create a table in the previously created database.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute('''CREATE TABLE COMPANY
         (ID INT PRIMARY KEY     NOT NULL,
         NAME           TEXT    NOT NULL,
         AGE            INT     NOT NULL,
         ADDRESS        CHAR(50),
         SALARY         REAL);''')
print "Table created successfully";

conn.close()
```

When the above program is executed, it will create the COMPANY table in your **test.db** and it will display the following messages –

```
Opened database successfully
Table created successfully
```

Insert Operation

Following Python program shows how to create records in the COMPANY table created in the above example.

```python
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print "Records created successfully";
conn.close()
```

When the above program is executed, it will create the given records in the COMPANY table and it will display the following two lines –

```
Opened database successfully
Records created successfully
```

Select Operation

Following Python program shows how to fetch and display records from the COMPANY table created in the above example.

```python
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Update Operation

Following Python code shows how to use UPDATE statement to update any record and then fetch and display the updated records from the COMPANY table.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit
print "Total number of rows updated :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
Total number of rows updated : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 25000.0

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0
```

```
ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Delete Operation

Following Python code shows how to use DELETE statement to delete any record and then fetch and display the remaining records from the COMPANY table.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()
print "Total number of rows deleted :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

When the above program is executed, it will produce the following result.

```
Opened database successfully
Total number of rows deleted : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Introduction: CGI Programming

The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script.

What is CGI?

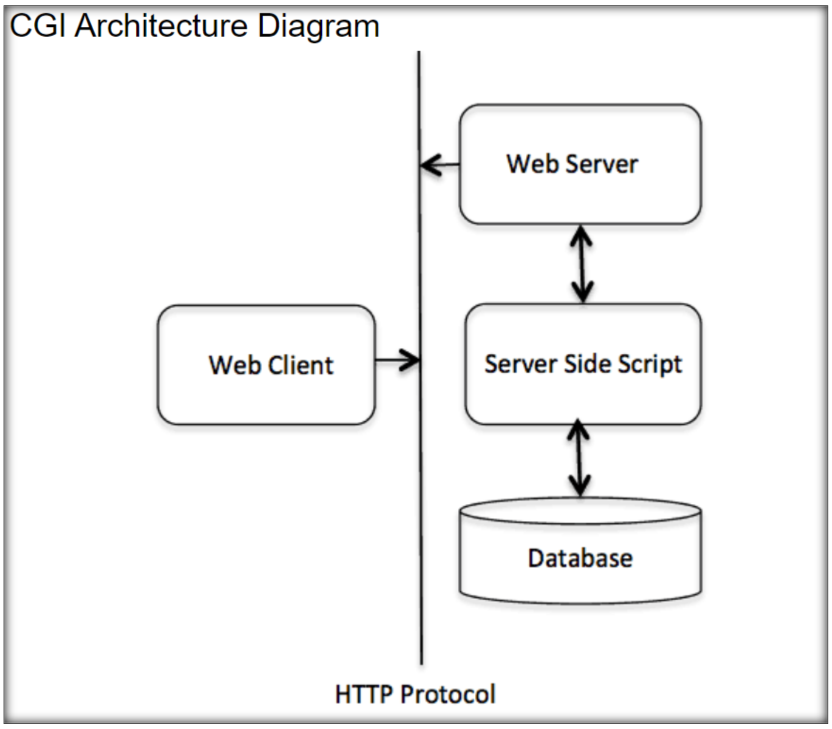
- The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.
- The current version is CGI/1.1 and CGI/1.2 is under progress.

Web Browsing

To understand the concept of CGI, let us see what happens when we click a hyper link to browse a particular web page or URL.

- Your browser contacts the HTTP web server and demands for the URL, i.e., filename.
- Web Server parses the URL and looks for the filename. If it finds that file then sends it back to the browser, otherwise sends an error message indicating that you requested a wrong file.
- Web browser takes response from web server and displays either the received file or error message.

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts. These CGI programs can be a Python Script, PERL Script, Shell Script, C or C++ program, etc.



Web Server Support and Configuration

Before you proceed with CGI Programming, make sure that your Web Server supports CGI and it is configured to handle CGI Programs. All the CGI Programs to be executed by the HTTP server are kept in a pre-configured directory. This directory is called CGI Directory and by convention it is named as /var/www/cgi-bin. By convention, CGI files have extension as. **cgi,** but you can keep your files with python extension **.py** as well.

By default, the Linux server is configured to run only the scripts in the cgi-bin directory in /var/www. If you want to specify any other directory to run your CGI scripts, comment the following lines in the httpd.conf file –

```
<Directory "/var/www/cgi-bin">
   AllowOverride None
   Options ExecCGI
   Order allow,deny
   Allow from all
</Directory>

<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

N.B: Your system should have Web Server up and running successfully and system should able to run any other CGI program like Perl or Shell, etc.

First CGI Program

Here is a simple link, which is linked to a CGI script called hello.py. This file is kept in /var/www/cgi-bin directory and it has following content. Before running your CGI program, make sure you have change mode of file using **chmod 755 hello.py** UNIX command to make file executable.

```
#!/usr/bin/python

print "Content-type:text/html\r\n\r\n"
print '<html>'
print '<head>'
print '<title>Hello World - First CGI Program</title>'
print '</head>'
print '<body>'
print '<h2>Hello World! This is my first CGI program</h2>'
print '</body>'
print '</html>'
```

If you click hello.py, then this produces the following output –

```
Hello World! This is my first CGI program
```

This hello.py script is a simple Python script, which writes its output on STDOUT file, i.e., screen. There is one important and extra feature available which is first line to be printed **Content-type:text/html\r\n\r\n**. This line is sent back to the browser and it specifies the content type to be displayed on the browser screen.

By now you must have understood basic concept of CGI and you can write many complicated CGI programs using Python. This script can interact with any other external system also to exchange information such as RDBMS.

HTTP Header

The line **Content-type:text/html\r\n\r\n** is part of HTTP header which is sent to the browser to understand the content. All the HTTP header will be in the following form –

```
HTTP Field Name: Field Content

For Example
Content-type: text/html\r\n\r\n
```

There are few other important HTTP headers, which you will use frequently in your CGI Programming.

Sr.No.	Header & Description
1	**Content-type:** A MIME string defining the format of the file being returned. Example is Content-type:text/html
2	**Expires: Date** The date the information becomes invalid. It is used by the browser to decide when a page needs to be refreshed. A valid date string is in the format 01 Jan 1998 12:00:00 GMT.
3	**Location: URL** The URL that is returned instead of the URL requested. You can use this field to redirect a request to any file.
4	**Last-modified: Date** The date of last modification of the resource.
5	**Content-length: N** The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file.
6	**Set-Cookie: String** Set the cookie passed through the *string*

CGI Environment Variables

All the CGI programs have access to the following environment variables. These variables play an important role while writing any CGI program.

Sr.No.	Variable Name & Description
1	**CONTENT_TYPE** The data type of the content. Used when the client is sending attached content to the server. For example, file upload.
2	**CONTENT_LENGTH** The length of the query information. It is available only for POST requests.
3	**HTTP_COOKIE** Returns the set cookies in the form of key & value pair.
4	**HTTP_USER_AGENT**

	The User-Agent request-header field contains information about the user agent originating the request. It is name of the web browser.
5	**PATH_INFO** The path for the CGI script.
6	**QUERY_STRING** The URL-encoded information that is sent with GET method request.
7	**REMOTE_ADDR** The IP address of the remote host making the request. This is useful logging or for authentication.
8	**REMOTE_HOST** The fully qualified name of the host making the request. If this information is not available, then REMOTE_ADDR can be used to get IR address.
9	**REQUEST_METHOD** The method used to make the request. The most common methods are GET and POST.
10	**SCRIPT_FILENAME** The full path to the CGI script.
11	**SCRIPT_NAME** The name of the CGI script.
12	**SERVER_NAME** The server's hostname or IP Address
13	**SERVER_SOFTWARE** The name and version of the software the server is running.

Here is small CGI program to list out all the CGI variables. Click this link to see the result Get Environment.

```
#!/usr/bin/python

import os

print "Content-type: text/html\r\n\r\n";
print "<font size=+1>Environment</font><\br>";
for param in os.environ.keys():
   print "<b>%20s</b>: %s<\br>" % (param, os.environ[param])
```

GET and POST Methods

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program. Most frequently, browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.

Passing Information using GET method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the “?” character as follows –

```
http://www.test.com/cgi-bin/hello.py?key1=value1&key2=value2
```

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be sent in a request string. The GET method sends information using QUERY_STRING header and will be accessible in your CGI Program through QUERY_STRING environment variable.

You can pass information by simply concatenating key and value pairs along with any URL or you can use HTML <FORM> tags to pass information using GET method.

Simple URL Example:Get Method

Here is a simple URL, which passes two values to hello_get.py program using GET method.

/cgi-bin/hello_get.py?first_name=ZARA&last_name=ALI

Below is **hello_get.py** script to handle input given by web browser. We are going to use **cgi** module, which makes it very easy to access passed information –

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

This would generate the following result –

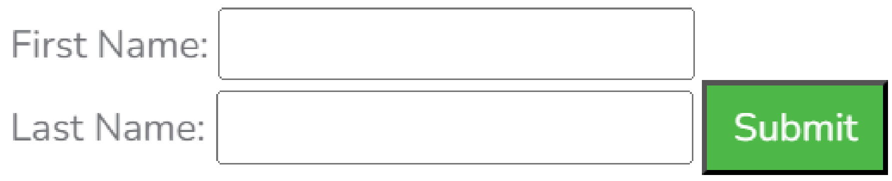
```
Hello ZARA ALI
```

Simple FORM Example: GET Method

This example passes two values using HTML FORM and submit button. We use same CGI script hello_get.py to handle this input.

```
<form action = "/cgi-bin/hello_get.py" method = "get">
First Name: <input type = "text" name = "first_name">  <br />

Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
```

Here is the actual output of the above form, you enter First and Last Name and then click submit button to see the result.



Passing Information Using POST Method

A generally more reliable method of passing information to a CGI program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes into the CGI script in the form of the standard input.

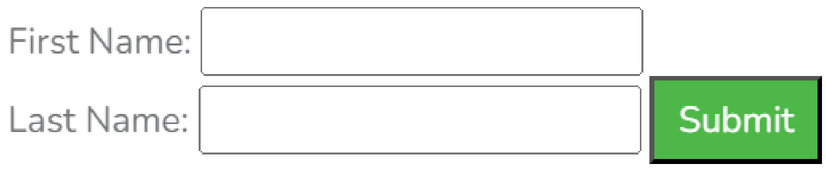
Below is same hello_get.py script which handles GET as well as POST method.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

Let us take again same example as above which passes two values using HTML FORM and submit button. We use same CGI script hello_get.py to handle this input.

```
<form action = "/cgi-bin/hello_get.py" method = "post">
First Name: <input type = "text" name = "first_name"><br />
Last Name: <input type = "text" name = "last_name" />

<input type = "submit" value = "Submit" />
</form>
```

Here is the actual output of the above form. You enter First and Last Name and then click submit button to see the result.



Passing Checkbox Data to CGI Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code for a form with two checkboxes –

```
<form action = "/cgi-bin/checkbox.cgi" method = "POST" target = "_blank">
<input type = "checkbox" name = "maths" value = "on" /> Maths
<input type = "checkbox" name = "physics" value = "on" /> Physics
<input type = "submit" value = "Select Subject" />
</form>
```

The result of this code is the following form –

Below is checkbox.cgi script to handle input given by web browser for checkbox button.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()
```

```
# Get data from fields
if form.getvalue('maths'):
   math_flag = "ON"
else:
   math_flag = "OFF"

if form.getvalue('physics'):
   physics_flag = "ON"
else:
   physics_flag = "OFF"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Checkbox - Third CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> CheckBox Maths is : %s</h2>" % math_flag
print "<h2> CheckBox Physics is : %s</h2>" % physics_flag
print "</body>"
print "</html>"
```

Passing Radio Button Data to CGI Program

Radio Buttons are used when only one option is required to be selected.

Here is example HTML code for a form with two radio buttons –

```
<form action = "/cgi-bin/radiobutton.py" method = "post" target = "_blank">
<input type = "radio" name = "subject" value = "maths" /> Maths
<input type = "radio" name = "subject" value = "physics" /> Physics
<input type = "submit" value = "Select Subject" />
</form>
```

The result of this code is the following form –

Below is radiobutton.py script to handle input given by web browser for radio button –

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('subject'):
   subject = form.getvalue('subject')
else:
   subject = "Not set"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Radio - Fourth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Selected Subject is %s</h2>" % subject
print "</body>"
print "</html>"
```

Passing Text Area Data to CGI Program

TEXTAREA element is used when multiline text has to be passed to the CGI Program.

Here is example HTML code for a form with a TEXTAREA box –

```
<form action = "/cgi-bin/textarea.py" method = "post" target = "_blank">
<textarea name = "textcontent" cols = "40" rows = "4">
Type your text here...
</textarea>
<input type = "submit" value = "Submit" />
</form>
```

The result of this code is the following form –

Below is textarea.cgi script to handle input given by web browser –

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('textcontent'):
   text_content = form.getvalue('textcontent')
else:
   text_content = "Not entered"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>";
print "<title>Text Area - Fifth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Entered Text Content is %s</h2>" % text_content
print "</body>"
```

Passing Drop Down Box Data to CGI Program

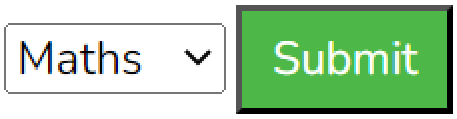
Drop Down Box is used when we have many options available but only one or two will be selected.

Here is example HTML code for a form with one drop down box –

```
<form action = "/cgi-bin/dropdown.py" method = "post" target = "_blank">
<select name = "dropdown">
<option value = "Maths" selected>Maths</option>
<option value = "Physics">Physics</option>
```

```
</select>
<input type = "submit" value = "Submit"/>
</form>
```

The result of this code is the following form –



Below is dropdown.py script to handle input given by web browser.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('dropdown'):
   subject = form.getvalue('dropdown')
else:
   subject = "Not entered"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Dropdown Box - Sixth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Selected Subject is %s</h2>" % subject
print "</body>"
print "</html>"
```

Using Cookies in CGI

HTTP protocol is a stateless protocol. For a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. How to maintain user's session information across all the web pages?

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How It Works?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the cookie is available for retrieval. Once retrieved, your server knows/remembers what was stored.

Cookies are a plain text data record of 5 variable-length fields –

- **Expires** – The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** – The domain name of your site.
- **Path** – The path to the directory or web page that sets the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure** – If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** – Cookies are set and retrieved in the form of key and value pairs.

Setting up Cookies

It is very easy to send cookies to browser. These cookies are sent along with HTTP Header before to Content-type field. Assuming you want to set UserID and Password as cookies. Setting the cookies is done as follows –

```
#!/usr/bin/python

print "Set-Cookie:UserID = XYZ;\r\n"
print "Set-Cookie:Password = XYZ123;\r\n"
print "Set-Cookie:Expires = Tuesday, 31-Dec-2007 23:12:40 GMT";\r\n"
print "Set-Cookie:Domain = www.tutorialspoint.com;\r\n"
print "Set-Cookie:Path = /perl;\n"
print "Content-type:text/html\r\n\r\n"
...........Rest of the HTML Content....
```

From this example, you must have understood how to set cookies. We use **Set-Cookie** HTTP header to set cookies.

It is optional to set cookies attributes like Expires, Domain, and Path. It is notable that cookies are set before sending magic line **"Content-type:text/html\r\n\r\n**.

Retrieving Cookies

It is very easy to retrieve all the set cookies. Cookies are stored in CGI environment variable HTTP_COOKIE and they will have following form –

```
key1 = value1;key2 = value2;key3 = value3....
```

Here is an example of how to retrieve cookies.

```
#!/usr/bin/python

# Import modules for CGI handling
from os import environ
import cgi, cgitb

if environ.has_key('HTTP_COOKIE'):
   for cookie in map(strip, split(environ['HTTP_COOKIE'], ';')):
      (key, value ) = split(cookie, '=');
      if key == "UserID":
         user_id = value

      if key == "Password":
         password = value

print "User ID  = %s" % user_id
print "Password = %s" % password
```

This produces the following result for the cookies set by above script –

```
User ID = XYZ
Password = XYZ123
```

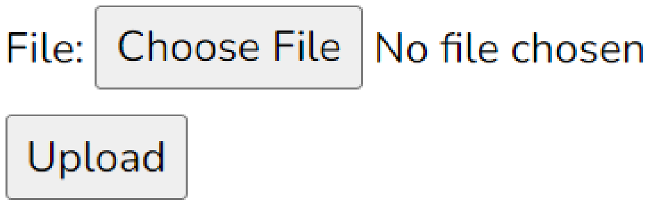
File Upload Example

To upload a file, the HTML form must have the enctype attribute set to **multipart/form-data**. The input tag with the file type creates a "Browse" button.

```
<html>
<body>
  <form enctype = "multipart/form-data"
           action = "save_file.py" method = "post">
  <p>File: <input type = "file" name = "filename" /></p>
  <p><input type = "submit" value = "Upload" /></p>
  </form>
```

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

The result of this code is the following form –



Above example has been disabled intentionally to save people uploading file on our server, but you can try above code with your server.

Here is the script **save_file.py** to handle file upload –

```
#!/usr/bin/python

import cgi, os
import cgitb; cgitb.enable()

form = cgi.FieldStorage()

# Get filename here.
fileitem = form['filename']

# Test if the file was uploaded
if fileitem.filename:
  # strip leading path from file name to avoid
  # directory traversal attacks
  fn = os.path.basename(fileitem.filename)
  open('/tmp/' + fn, 'wb').write(fileitem.file.read())

  message = 'The file "' + fn + '" was uploaded successfully'

else:
  message = 'No file was uploaded'

print """\
Content-Type: text/html\n
<html>
<body>
  <p>%s</p>
</body>
</html>
""" % (message,)
```

If you run the above script on Unix/Linux, then you need to take care of replacing file separator as follows, otherwise on your windows machine above open() statement should work fine.

```
fn = os.path.basename(fileitem.filename.replace("\\", "/" ))
```

How To Raise a "File Download" Dialog Box?

Sometimes, it is desired that you want to give option where a user can click a link and it will pop up a "File Download" dialogue box to the user instead of displaying actual content. This is very easy and can be achieved through HTTP header. This HTTP header is be different from the header mentioned in previous section.

For example, if you want make a **FileName** file downloadable from a given link, then its syntax is as follows –

```
#!/usr/bin/python

# HTTP Header
print "Content-Type:application/octet-stream; name = \"FileName\"\r\n";
print "Content-Disposition: attachment; filename = \"FileName\"\r\n\n";

# Actual File Content will go here.
fo = open("foo.txt", "rb")

str = fo.read();
print str

# Close opend file
fo.close()
```

Introduction: GUI Programming

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

- **Tkinter** – Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this chapter.
- **wxPython** – This is an open-source Python interface for wxWindows http://wxpython.org.
- **JPython** – JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine http://www.jython.org.

There are many other interfaces available, which you can find them on the net.

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

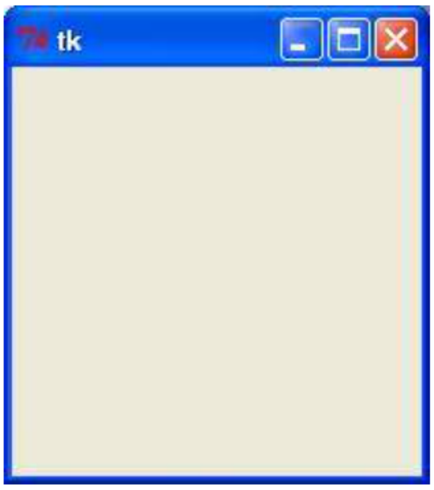
- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.

- Enter the main event loop to take action against each event triggered by the user.

Example

```
#!/usr/bin/python

import Tkinter
top = Tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

This would create a following window –



Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table –

Sr.No.	Operator & Description
1	Button The Button widget is used to display buttons in your application.
2	Canvas The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
3	Checkbutton The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4	Entry The Entry widget is used to display a single-line text field for accepting values from a user.
5	Frame The Frame widget is used as a container widget to organize other widgets.
6	Label

	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
7	Listbox The Listbox widget is used to provide a list of options to a user.
8	Menubutton The Menubutton widget is used to display menus in your application.
9	Menu The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
10	Message The Message widget is used to display multiline text fields for accepting values from a user.
11	Radiobutton The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12	Scale The Scale widget is used to provide a slider widget.
13	Scrollbar The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14	Text The Text widget is used to display text in multiple lines.
15	Toplevel The Toplevel widget is used to provide a separate window container.
16	Spinbox The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
17	PanedWindow A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18	LabelFrame A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19	tkMessageBox This module is used to display message boxes in your applications.

Standard attributes

Let us take a look at how some of their common attributes.such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles

- Bitmaps
- Cursors

Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

- The *pack()* Method – This geometry manager organizes widgets in blocks before placing them in the parent widget.
- The *grid()* Method – This geometry manager organizes widgets in a table-like structure in the parent widget.
- The *place()* Method – This geometry manager organizes widgets by placing them in a specific position in the parent widget.

References:

1. Python How to Program, DEITEL, Pearson.

2. Website: https://www.tutorialspoint.com/index.htm
