MODULE – 5

5.1 NETWORKED PROGRAMS

In this era of internet, it is a requirement in many situations to retrieve the data from web and to process it. In this section, we will discuss basics of network protocols and Python libraries available to extract data from web.

5.1.1 HyperText Transfer Protocol (HTTP)

HTTP (HyperText Transfer Protocol) is the media through which we can retrieve web-based data. The **HTTP** is an application protocol for distributed and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

Consider a situation:

- you try to read a socket, but the program on the other end of the socket has not sent any data, then you need to wait.
- If the programs on both ends of the socket simply wait for some data without sending anything, they will wait for a very long time.

So an important part of programs that communicate over the Internet is to have some sort of protocol. A protocol is a set of precise rules that determine

- Who will send request for what purpose
- What action to be taken
- What response to be given

To send request and to receive response, HTTP uses GET and POST methods.

NOTE: To test all the programs in this section, you must be connected to internet.

5.1.2 The World's Simplest Web Browser

The built-in module ***socket*** of Python facilitates the programmer to make network connections and to retrieve data over those sockets in a Python program. ***Socket*** is bidirectional data path to a remote system. **A *socket is much like a file, except that a single socket provides a two-way connection* between two programs.** You can both read from and write to the same socket. If you write something to a socket, it is sent to the application at the other end of the socket. If you read from the socket, you are given the data which the other application has sent.

Consider a simple program to retrieve the data from a web page. To understand the program given below, one should know the meaning of terminologies used there.

- **AF_INET** is an address family (IP) that is used to designate the type of addresses that your socket can communicate with. When you create a socket, you have to specify its address family, and then you can use only addresses of that type with the socket.

- **SOCK_STREAM** is a constant indicating the type of socket (TCP). It works as a file stream and is most reliable over the network.
- **Port** is a logical end-point. Port 80 is one of the most commonly used **port** numbers in the Transmission Control Protocol (TCP) suite.
- The command to retrieve the data must use CRLF(Carriage Return Line Feed) line endings, and it must end in \r\n\r\n (line break in protocol specification).
- ***encode()*** method applied on strings will return bytes-representation of the string. Instead of *encode()* method, one can attach a character *b* at the beginning of the string for the same effect.
- ***decode()*** method returns a string decoded from the given bytes.

A socket connection between the user program and the webpage is shown in Figure 5.1.

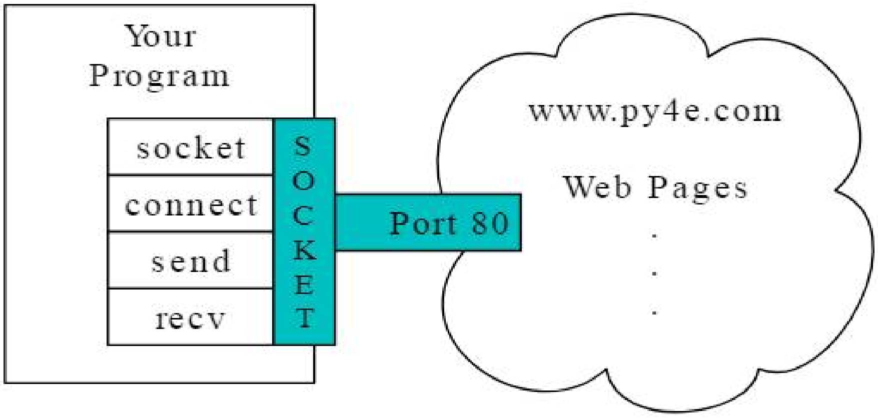

Figure 5.1 A Socket Connection

Now, observe the following program –

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd='GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break

    print(data.decode(),end='')

mysock.close()
```

When we run above program, we will get some information related to web-server of the website which we are trying to scrape. Then, we will get the data written in that web-page. In this program, we are extracting 512 bytes of data at a time. (One can use one's convenient number here). The extracted data is decoded and printed. When the length of data becomes less than one (that is, no more data left out on the web page), the loop is terminated.

5.1.3 Retrieving an Image over HTTP

In the previous section, we retrieved the text data from the webpage. Similar logic can be used to extract images on the webpage using HTTP. In the following program, we extract the image data in the chunks of 5120 bytes at a time, store that data in a string, trim off the headers and then store the image file on the disk.

```
import socket
import time

HOST = 'data.pr4e.org'         #host name
PORT = 80                      #port number

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((HOST, PORT))

mysock.sendall(b'GET http://data.pr4e.org/cover3.jpg HTTP/1.0\r\n\r\n')

count = 0
picture = b""                  #empty string in binary format

while True:
    data = mysock.recv(5120)  #retrieve 5120 bytes at a time
    if (len(data) < 1):
        break

    time.sleep(0.25)     #programmer can see data retrieval easily
    count = count + len(data)
    print(len(data), count)    #display cumulative data retrieved
    picture = picture + data

mysock.close()

pos = picture.find(b"\r\n\r\n") #find end of the header (2 CRLF)
print('Header length', pos)
print(picture[:pos].decode())

# Skip past the header and save the picture data
picture = picture[pos+4:]

fhand = open("stuff.jpg", "wb")    #image is stored as stuff.jpg
```

```
fhand.write(picture)
fhand.close()
```

When we run the above program, the amount of data (in bytes) retrieved from the internet is displayed in a cumulative format. At the end, the image file 'stuff.jpg' will be stored in the current working directory. (One has to verify it by looking at current working directory of the program).

5.1.4 Retrieving Web Pages with urllib

Python provides simpler way of webpage retrieval using the library ***urllib.*** Here, webpage is treated like a file. ***urllib*** handles all of the HTTP protocol and header details. Following is the code equivalent to the program given in Section 5.1.2.

```
import urllib.request

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')

for line in fhand:
    print(line.decode().strip())
```

Once the web page has been opened with `urllib.urlopen`, we can treat it like a file and read through it using a for-loop. When the program runs, we only see the output of the contents of the file. The headers are still sent, but the `urllib` code consumes the headers and only returns the data to us.

Following is the program to retrieve the data from the file romeo.txt which is residing at www.data.pr4e.org, and then to count number of words in it.

```
import urllib.request
fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
counts = dict()

for line in fhand:
     words = line.decode().split()
     for word in words:
        counts[word] = counts.get(word, 0) + 1

print(counts)
```

5.1.5 Reading Binary Files using urllib

Sometimes you want to retrieve a non-text (or binary) file such as an image or video file. The data in these files is generally not useful to print out, but you can easily make a copy of a URL to a local file on your hard disk using ***urllib***. In Section 5.1.3, we have seen how to retrieve image file from the web using sockets. Now, here is an equivalent program using ***urllib***.

```
import urllib.request
img=urllib.request.urlopen('http://data.pr4e.org/cover3.jpg').read()
fhand = open('cover3.jpg', 'wb')
fhand.write(img)
fhand.close()
```

Once we execute the above program, we can see a file `cover3.jpg` in the current working directory in our computer.

The program reads all of the data in at once across the network and stores it in the variable ***img*** in the main memory of your computer, then opens the file `cover.jpg` and writes the data out to your disk. This will work if the size of the file is less than the size of the memory (RAM) of your computer. However, if this is a large audio or video file, this program may crash or at least run extremely slowly when your computer runs out of memory. In order to avoid memory overflow, we retrieve the data in blocks (or buffers) and then write each block to your disk before retrieving the next block. This way the program can read any size file without using up all of the memory you have in your computer.

Following is another version of above program, where data is read in chunks and then stored onto the disk.

```
import urllib.request

img=urllib.request.urlopen('http://data.pr4e.org/cover3.jpg')
fhand = open('cover3.jpg', 'wb')
size = 0

while True:
    info = img.read(100000)
    if len(info) < 1:
        break
    size = size + len(info)
    fhand.write(info)

print(size, 'characters copied.')
fhand.close()
```

Once we run the above program, an image file `cover3.jpg` will be stored on to the current working directory.

5.1.6 Parsing HTML and Scraping the Web

One of the common uses of the urllib capability in Python is to *scrape the web.* Web scraping is when we write a program that pretends to be a web browser and retrieves pages, then examines the data in those pages looking for patterns. Example: a search engine such as Google will look at the source of one web page and extract the links to other pages and retrieve those pages, extracting links, and so on. Using this technique,

Google *spiders its way through nearly all of the* pages on the web. Google also uses the frequency of links from pages it finds to a particular page as one measure of how "important" a page is and how high the page should appear in its search results.

5.1.7 Parsing HTML using Regular Expressions

Sometimes, we may need to parse the data on the web which matches a particular pattern. For this purpose, we can use regular expressions. Now, we will consider a program that extracts all the hyperlinks given in a particular webpage. To understand the Python program for this purpose, one has to know the pattern of an HTML file. Here is a simple HTML file –

```
<h1>The First Page</h1>
<p>
     If you like, you can switch to the
       <a href="http://www.dr-chuck.com/page2.htm">
     Second Page</a>.
</p>
```

Here,
`<h1>` and `</h1>` are the beginning and end of header tags
`<p>` and `</p>` are the beginning and end of paragraph tags
`<a>` and `</a>` are the beginning and end of anchor tag which is used for giving links
`href` is the attribute for anchor tag which takes the value as the link for another page.

The above information clearly indicates that if we want to extract all the hyperlinks in a webpage, we need a regular expression which matches the `href` attribute. Thus, we can create a regular expression as –

```
     href="http://.+?"
```

Here, the question mark in `.+?` indicate that the match should find smallest possible matching string.

Now, consider a Python program that uses the above regular expression to extract all hyperlinks from the webpage given as input.

```
     import urllib.request
     import re

     url = input('Enter - ')  #give URL of any website

     html = urllib.request.urlopen(url).read()
     links = re.findall(b'href="(http://.*?)"', html)

     for link in links:
          print(link.decode())
```

When we run this program, it prompts for user input. We need to give a valid URL of any website. Then all the hyperlinks on that website will be displayed.

5.1.8 Parsing HTML using BeautifulSoup

There are a number of Python libraries which can help you parse HTML and extract data from the pages. Each of the libraries has its strengths and weaknesses and you can pick one based on your needs. ***BeautifulSoup** library* is one of the simplest libraries available for parsing. To use this, *download and install the BeautifulSoup code* from:

http://www.crummy.com/software/

Consider the following program which uses `urllib` to read the page and uses BeautifulSoup to extract `href` attribute from the anchor tag.

```
import urllib.request
from bs4 import BeautifulSoup
import ssl                       #Secure Socket Layer

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url,context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')
tags = soup('a')

for tag in tags:
    print(tag.get('href', None))
```

A sample output would be –

```
     Enter - http://www.dr-chuck.com/page1.htm
     http://www.dr-chuck.com/page2.htm
```

The above program prompts for a web address, then opens the web page, reads the data and passes the data to the BeautifulSoup parser, and then retrieves all of the anchor tags and prints out the href attribute for each tag.

The BeautifulSoup can be used to extract various parts of each tag as shown below –

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import ssl

ctx = ssl.create_default_context()
```

```
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')

html = urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, "html.parser")
tags = soup('a')
for tag in tags:
     print('TAG:', tag)
     print('URL:', tag.get('href', None))
     print('Contents:', tag.contents[0])
     print('Attrs:', tag.attrs)
```

The sample output would be –

```
     Enter - http://www.dr-chuck.com/page1.htm
     TAG: <a href="http://www.dr-chuck.com/page2.htm">
     Second Page</a>
     URL: http://www.dr-chuck.com/page2.htm
     Contents:
     Second Page
     Attrs: {'href': 'http://www.dr-chuck.com/page2.htm'}
```

5.2 USING WEB SERVICES

There are two common formats that are used while exchanging data across the web. One is HTML and the other is XML (eXtensible Markup Language). In the previous section we have seen how to retrieve the data from a web-page which is in the form of HTML. Now, we will discuss the retrieval of data from web-page designed using XML.

XML is best suited for exchanging document-style data. When programs just want to exchange dictionaries, lists, or other internal information with each other, they use JavaScript Object Notation or JSON (refer www.json.org). We will look at both formats.

5.2.1 eXtensible Markup Language (XML)

XML looks very similar to HTML, but XML is more structured than HTML. Here is a sample of an XML document:

```
     <person>
          <name>Chuck</name>
          <phone type="intl">
               +1 734 303 4456
          </phone>
          <email hide="yes"/>
     </person>
```

Often it is helpful to think of an XML document as a tree structure where there is a top tag `person` and other tags such as `phone` are drawn as children of their parent nodes. Figure 5.2 is the tree structure for above given XML code.

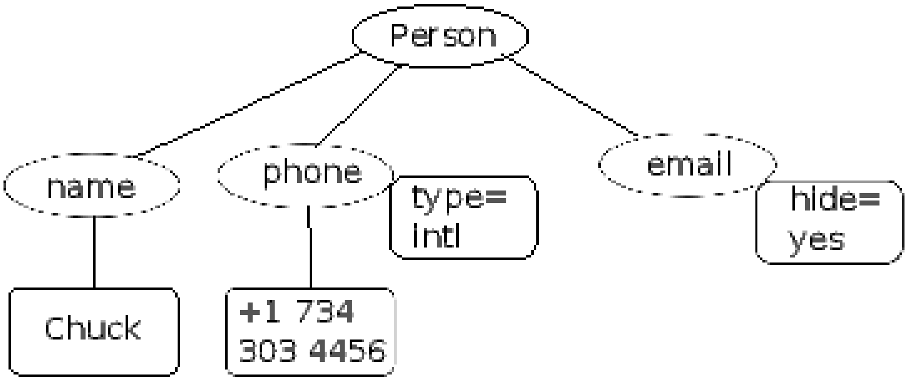

Figure 5.2 Tree Representation of XML

5.2.2 Parsing XML

Python provides library `xml.etree.ElementTree` to parse the data from XML files. One has to provide XML code as a string to built-in method `fromstring()` of `ElementTree` class. `ElementTree` acts as a parser and provides a set of relevant methods to extract the data. Hence, the programmer need not know the rules and the format of XML document syntax. The `fromstring()`method will convert XML code into a tree-structure of XML nodes. When the XML is in a tree format, Python provides several methods to extract data from XML. Consider the following program.

```
import xml.etree.ElementTree as ET

#XML code embedded in a string format
data = '''
<person>
    <name>Chuck</name>
    <phone type="intl">
        +1 734 303 4456
    </phone>
    <email hide="yes"/>
</person>'''

tree = ET.fromstring(data)
print('Attribute for tag email:', tree.find('email').get('hide'))
print('Attribute for tag phone:', tree.find('phone').get('type'))
```

The output would be –

```
     Name: Chuck
     Attribute for the tag email: yes
     Attribute for the tag phone: intl
```

In the above example, `fromstring()` is used to convert XML code into a tree. The `find()` method searches XML tree and retrieves a node that matches the specified tag. The `get()` method retrieves the value associated with the specified attribute of that tag. Each node can have some text, some attributes (like hide), and some “child” nodes. Each node can be the parent for a tree of nodes.

5.2.3 Looping Through Nodes

Most of the times, XML documents are hierarchical and contain multiple nodes. To process all the nodes, we need to loop through all those nodes. Consider following example as an illustration.

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
        </user>
    </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))

for item in lst:
     print('Name', item.find('name').text)
     print('Id', item.find('id').text)
     print('Attribute', item.get("x"))
```

The output would be –

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

The `findall()` method retrieves a Python list of subtrees that represent the `user` structures in the XML tree. Then we can write a for-loop that extracts each of the `user` nodes, and prints the `name` and `id,` which are text elements as well as the attribute `x` from the `user` node.

5.2.4 JavaScript Object Notation (JSON)

The JSON format was inspired by the object and array format used in the JavaScript language. But since Python was invented before JavaScript, Python's syntax for dictionaries and lists influenced the syntax of JSON. So the format of JSON is a combination of Python lists and dictionaries. Following is the JSON encoding that is roughly equivalent to the XML code (the string `data`) given in the program of Section 5.2.2.

```
{
     "name" : "Chuck",
     "phone": {"type" : "intl", "number" : "+1 734 303 4456"},
     "email": {"hide" : "yes"}
}
```

Observe the differences between XML code and JSON code:

- In XML, we can add attributes like "intl" to the "phone" tag. In JSON, we simply have key-value pairs.
- XML uses tag "person", which is replaced by a set of outer curly braces in JSON.

In general, JSON structures are simpler than XML because JSON has fewer capabilities than XML. But JSON has the advantage that it maps *directly* to some combination of dictionaries and lists. And since nearly all programming languages have something equivalent to Python's dictionaries and lists, JSON is a very natural format to have two compatible programs exchange data. JSON is quickly becoming the format of choice for nearly all data exchange between applications because of its relative simplicity compared to XML.

5.2.5 Parsing JSON

Python provides a module `json` to parse the data in JSON pages. Consider the following program which uses JSON equivalent of XML string written in Section 5.2.3. Note that, the JSON string has to embed a list of dictionaries.

```
import json

data = '''
[
    { "id" : "001",
      "x" : "2",
      "name" : "Chuck"
    } ,
    { "id" : "009",
      "x" : "7",
      "name" : "Chuck"
    }
]'''

info = json.loads(data)
print('User count:', len(info))

for item in info:
     print('Name', item['name'])
     print('Id', item['id'])
     print('Attribute', item['x'])
```

The output would be –

```
     User count: 2
     Name Chuck
     Id 001
     Attribute 2
     Name Chuck
     Id 009
     Attribute 7
```

Here, the string `data` contains a list of users, where each user is a key-value pair. The method `loads()` in the `json` module converts the string into a list of dictionaries. Now onwards, we don't need anything from `json`, because the parsed data is available in Python native structures. Using a for-loop, we can iterate through the list of dictionaries and extract every element (in the form of key-value pair) as if it is a dictionary object. That is, we use index operator (a pair of square brackets) to extract value for a particular key.

NOTE: Current IT industry trend is to use JSON for web services rather than XML. Because, JSON is simpler than XML and it directly maps to native data structures we already have in the programming languages. This makes parsing and data extraction simpler compared to XML. But XML is more self descriptive than JSON and so there are some applications where XML retains an advantage. For example, most word processors store documents internally using XML rather than JSON.

5.2.6 Application Programming Interface (API)

Till now, we have discussed how to exchange data between applications using HTTP, XML and JSON. The next step is to understand API. Application Programming Interface defines and documents the contracts between the applications. When we use an API, generally one program makes a set of services available for use by other applications and publishes the APIs (i.e., the "rules") that must be followed to access the services provided by the program.

When we begin to build our programs where the functionality of our program includes access to services provided by other programs, we call the approach a ***Service-Oriented Architecture***(SOA). A SOA approach is one where our overall application makes use of the services of other applications. A non-SOA approach is where the application is a single stand-alone application which contains all of the code necessary to implement the application.

Consider an example of SOA: Through a single website, we can book flight tickets and hotels. The data related to hotels is not stored in the airline servers. Instead, airline servers contact the services on hotel servers and retrieve the data from there and present it to the user. When the user agrees to make a hotel reservation using the airline site, the airline site uses another web service on the hotel systems to actually make the reservation. Similarly, to reach airport, we may book a cab through a cab rental service. And when it comes time to charge your credit card for the whole transaction, still other computers become involved in the process. This process is depicted in Figure 5.3.

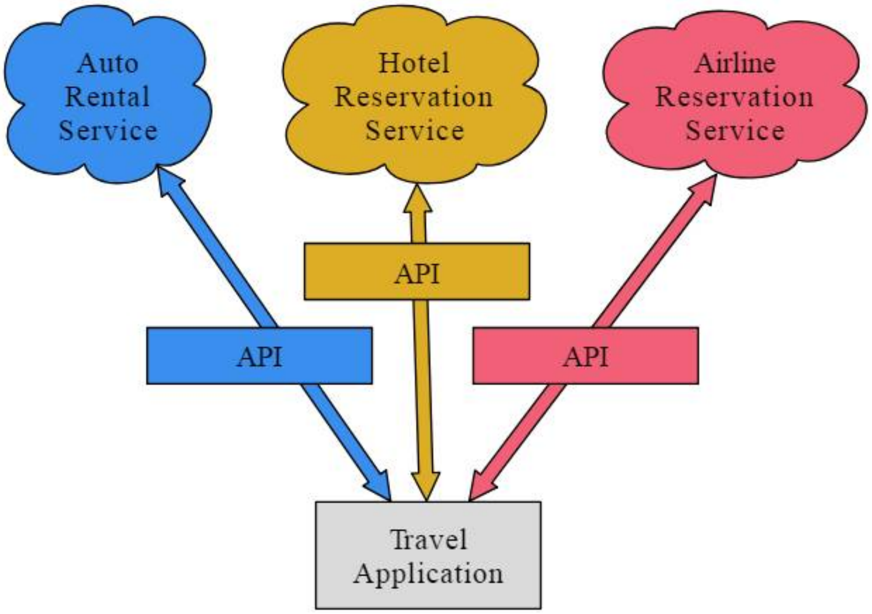

Figure 5.3 Server Oriented Architecture

SOA has following major advantages:

- we always maintain only one copy of data (this is particularly important for things like hotel reservations where we do not want to over-commit)
- the owners of the data can set the rules about the use of their data.

With these advantages, an SOA system must be carefully designed to have good performance and meet the user's needs. When an application makes a set of services in its API available over the web, then it is called as ***web services.***

5.2.7 Google Geocoding Web Service

Google has a very good web service which allows anybody to use their large database of geographic information. We can submit a geographic search string like "Rajarajeshwari Nagar" to their geocoding API. Then Google returns the location details of the string submitted.

The following program asks the user to provide the name of a location to be searched for. Then, it will call Google geocoding API and extracts the information from the returned JSON.

```
import urllib.request, urllib.parse, urllib.error
import json

serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

address = input('Enter location: ')
if len(address) < 1:
    exit()

url = serviceurl + urllib.parse.urlencode({'address': address})
print('Retrieving', url)
uh = urllib.request.urlopen(url)
data = uh.read().decode()
print('Retrieved', len(data), 'characters')

try:
    js = json.loads(data)
except:
    js = None

if not js or 'status' not in js or js['status'] != 'OK':
    print('==== Failure To Retrieve ====')
    print(data)


print(json.dumps(js, indent=4))
lat = js["results"][0]["geometry"]["location"]["lat"]
lng = js["results"][0]["geometry"]["location"]["lng"]
print('lat', lat, 'lng', lng)
location = js['results'][0]['formatted_address']
print(location)
```

(Students are advised to run the above program and check the output, which will contain several lines of Google geographical data).

The above program retrieves the search string and then encodes it. This encoded string along with Google API link is treated as a URL to fetch the data from the internet. The data retrieved from the internet will be now passed to JSON to put it in JSON object format. If the input string (which must be an existing geographical location like Channasandra, Malleshwaram etc!!) cannot be located by Google API either due to bad internet or due to unknown location, we just display the message as 'Failure to Retrieve'. If Google successfully identifies the location, then we will dump that data in JSON object. Then, using indexing on JSON (as JSON will be in the form of dictionary), we can retrieve the location address, longitude, latitude etc.

5.2.8 Security and API Usage

Public APIs can be used by anyone without any problem. But, if the API is set up by some private vendor, then one must have ***API key*** to use that API. If API key is available, then it can be included as a part of POST method or as a parameter on the URL while calling API.

Sometimes, vendor wants more security and expects the user to provide cryptographically signed messages using shared keys and secrets. The most common protocol used in the internet for signing requests is ***OAuth.***

As the Twitter API became increasingly valuable, Twitter went from an open and public API to an API that required the use of OAuth signatures on each API request. But, there are still a number of convenient and free OAuth libraries so you can avoid writing an OAuth implementation from scratch by reading the specification. These libraries are of varying complexity and have varying degrees of richness. The OAuth web site has information about various OAuth libraries.

5.3 USING DATABASES AND SQL

A structured set of data stored in a permanent storage is called as ***database***. Most of the databases are organized like a dictionary – that is, they map keys to values. Unlike dictionaries, databases can store huge set of data as they reside on permanent storage like hard disk of the computer.

There are many database management softwares like Oracle, MySQL, Microsoft SQL Server, PostgreSQL, SQLite etc. They are designed to insert and retrieve data very fast, however big the dataset is. Database software builds *indexes* as data is added to the database so as to provider quicker access to particular entry.

In this course of study, SQLite is used because it is already built into Python. SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a non-standard variant of the SQL query language. SQLite is designed to be *embedded* into other applications to provide database support within the application. For example, the Firefox browser also uses the SQLite database internally. SQLite is well suited to some of the data manipulation problems in Informatics such as the Twitter spidering application etc.

5.3.1 Database Concepts

For the first look, database seems to be a spreadsheet consisting of multiple sheets. The primary data structures in a database are ***tables, rows*** and ***columns.*** In a relational database terminology, tables, rows and columns are referred as ***relation, tuple*** and ***attribute*** respectively. Typical structure of a database table is as shown below. Each table may consist of n number of attributes and m number of tuples (or records). Every tuple gives the information about one individual. Every cell(i, j) in the table indicates value of j^{th} attribute for i^{th} tuple.

	Attribute1	**Attribute2**	**………………**	**Attribute_n**
Tuple1	V11	V12	………………	V1n
Tuple2	V21	V22	………………	V2n
…………..	………	…….	………………	……….
………….	…………	……….	……………..	………..
Tuple_m	Vm1	Vm2	……………..	Vmn

Consider the problem of storing details of students in a database table. The format may look like –

	RollNo	**Name**	**DoB**	**Marks**
Student1	1	Ram	22/10/2001	82.5
Student2	2	Shyam	20/12/2000	81.3
…………..	………	…….	………………	……….
………….	…………	……….	……………..	………..
Student_m	…………..	……….	…………….	…………

Thus, table columns indicate the type of information to be stored, and table rows gives record pertaining to every student. We can create one more table say *addressTable* consisting of attributes like DoorNo, StreetName, Locality, City, PinCode. To relate this table with a respective student stored in *studentTable,* we need to store RollNo also in *addressTable* (Note that, RollNo will be unique for every student, and hence there won't be any confusion). Thus, there is a relationship between two tables in a single database. There are softwares that can maintain proper relationships between multiple tables in a single database and are known as Relational Database Management Systems (RDBMS). The detailed discussion on RDBMS is out of the scope of this study.

5.3.2 Structured Query Language (SQL) Summary

To perform operations on databases, one should use structured query language. SQL is a standard language for storing, manipulating and retrieving data in databases. Irrespective of RDBMS software (like Oracle, MySQL, MS Access, SQLite etc) being used, the syntax of SQL remains the same. The usage of SQL commands may vary from one RDBMS to the other and there may be little syntactical difference. Also, when we are using some programming language like Python as a front-end to perform database applications, the way we embed SQL commands inside the program source-code is as per the syntax of respective programming language. Still, the underlying SQL commands remain the same. Hence, it is essential to understand basic commands of SQL.

There are some *clauses* like FROM, WHERE, ORDER BY, INNER JOIN etc. that are used with SQL commands, which we will study in a due course. The following table gives few of the SQL commands.

Command	Meaning
CREATE DATABASE	creates a new database
ALTER DATABASE	modifies a database
CREATE TABLE	creates a new table
ALTER TABLE	modifies a table
DROP TABLE	deletes a table
SELECT	extracts data from a database
INSERT INTO	inserts new data into a database
UPDATE	updates data in a database
DELETE	deletes data from a database

As mentioned earlier, every RDBMS has its own way of storing the data in tables. Each of RDBMS uses its own set of data types for the attribute values to be used. SQLite uses the data types as mentioned in the following table –

Data Type	Description
NULL	The value is a NULL value.
INTEGER	The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
REAL	The value is a floating point value, stored as an 8-byte floating point number
TEXT	The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)
BLOB	The value is a blob (Binary Large Object) of data, stored exactly as it was input

Note that, SQL commands are case-insensitive. But, it is a common practice to write commands and clauses in uppercase alphabets just to differentiate them from table name and attribute names.

Now, let us see some of the examples to understand the usage of SQL statements –

- `CREATE  TABLE Tracks (title TEXT, plays INTEGER)`

This command creates a table called as `Tracks` with the attributes `title` and `plays` where `title` can store data of type `TEXT` and `plays` can store data of type `INTEGER.`

- `INSERT INTO Tracks (title, plays) VALUES ('My Way', 15)`

  This command inserts one record into the table `Tracks` where values for the attributes `title` and `plays` are 'My Way' and 15 respectively.

- `SELECT * FROM Tracks`

  Retrieves all the records from the table `Tracks`

- `SELECT * FROM Tracks WHERE title = 'My Way'`

  Retrieves the records from the table `Tracks` having the value of attribute `title` as 'My Way'

- `SELECT title, plays FROM Tracks ORDER BY title`

  The values of attributes `title` and `plays` are retrieved from the table `Tracks` with the records ordered in ascending order of `title`.

- `UPDATE Tracks SET plays = 16 WHERE title = 'My Way'`

  Whenever we would like to modify the value of any particular attribute in the table, we can use UPDATE command. Here, the value of attribute `plays` is assigned to a new value for the record having value of `title` as 'My Way'.

- `DELETE FROM Tracks WHERE title = 'My Way'`

  A particular record can be deleted from the table using DELETE command. Here, the record with value of attribute `title` as 'My Way' is deleted from the table `Tracks`.

5.3.3 Database Browser for SQLite

Many of the operations on SQLite database files can be easily done with the help of software called *Database Browser for SQLite* which is freely available from:

http://sqlitebrowser.org/

Using this browser, one can easily create tables, insert data, edit data, or run simple SQL queries on the data in the database. This database browser is similar to a text editor when working with text files. When you want to do one or very few operations on a text file, you can just open it in a text editor and make the changes you want. When you have many changes that you need to do to a text file, often you will write a simple Python program. You will find the same pattern when working with databases. You will do simple operations in the database manager and more complex operations will be most conveniently done in Python.

5.3.4 Creating a Database Table

When we try to create a database table, we must specify the names of table columns and the type of data to be stored in those columns. When the database software knows the type of data in each column, it can choose the most efficient way to store and look up the data based on the type of data. Here is the simple code to create a database file and a table named Tracks with two columns in the database:

Ex1.

```
import sqlite3
conn = sqlite3.connect('music.sqlite')
cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS Tracks')
cur.execute('CREATE TABLE Tracks (title TEXT, plays INTEGER)')
conn.close()
```

The `connect()` method of `sqlite3` makes a "connection" to the database stored in the file `music.sqlite3` in the current directory. If the file does not exist, it will be created. Sometimes, the database is stored on a different database server from the server on which we are running our program. But, all the examples that we consider here will be local file in the current working directory of Python code.

A `cursor()` is like a file handle that we can use to perform operations on the data stored in the database. Calling `cursor()` is very similar conceptually to calling `open()` when dealing with text files. Hence, once we get a cursor, we can execute the commands on the contents of database using `execute()` method.

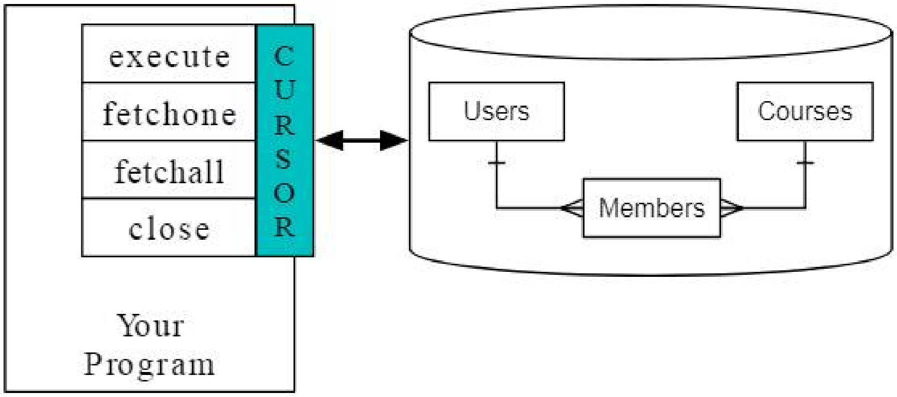

Figure 5.4 A Database Cursor

In the above program, we are trying to remove the database table `Tracks`, if at all it existed in the current working directory. The DROP TABLE command deletes the table along with all its columns and rows. This procedure will help to avoid a possible error of trying to create a table with same name. Then, we are creating a table with name `Tracks` which has two columns viz. `title`, which can take TEXT type data and `plays`, which can take INTEGER type data. Once our job with the database is over, we need to close the connection using close() method.

In the previous example, we have just created a table, but not inserted any records into it. So, consider below given program, which will create a table and then inserts two rows and finally delete records based on some condition.

Ex2.

```
import sqlite3

conn = sqlite3.connect('music.sqlite')
cur = conn.cursor()
cur.execute('DROP TABLE IF EXISTS Tracks')
cur.execute('CREATE TABLE Tracks (title TEXT, plays INTEGER)')

cur.execute(“INSERT INTO Tracks (title, plays) VALUES
                                   ('Thunderstruck', 20)”)
cur.execute(“INSERT INTO Tracks (title, plays) VALUES (?, ?)”,
                                             ('My Way', 15))
conn.commit()

print('Tracks:')
cur.execute('SELECT title, plays FROM Tracks')
for row in cur:
     print(row)

cur.execute('DELETE FROM Tracks WHERE plays < 100')
cur.close()
```

In the above program, we are inserting first record with the SQL command –

```
“INSERT INTO Tracks (title, plays) VALUES('Thunderstruck', 20)”
```

Note that, execute() requires SQL command to be in string format. But, if the value to be store in the table is also a string (TEXT type), then there may be a conflict of string representation using quotes. Hence, in this example, the entire SQL is mentioned within double-quotes and the value to be inserted in single quotes. If we would like to use either single quote or double quote everywhere, then we need to use escape-sequences like \’ or \”.

While inserting second row in a table, SQL statement is used with a little different syntax –

```
“INSERT INTO Tracks (title, plays) VALUES (?, ?)”,('My Way', 15)
```

Here, the question mark acts as a place-holder for particular value. This type of syntax is useful when we would like to pass user-input values into database table.

After inserting two rows, we must use `commit()` method to store the inserted records permanently on the database table. If this method is not applied, then the insertion (or any other statement execution) will be temporary and will affect only the current run of the program.

Later, we use SELECT command to retrieve the data from the table and then use for-loop to display all records. When data is retrieved from database using SELECT command, the cursor object gets those data as a list of records. Hence, we can use for-loop on the cursor object. Finally, we have used a DELETE command to delete all the records WHERE `plays` is less than 100.

Let us consider few more examples –

Ex3.

```
import sqlite3
from sqlite3 import Error

def create_connection():
    """ create a database connection to a database that resides
        in the memory
    """
    try:
        conn = sqlite3.connect(':memory:')
        print("SQLite Version:",sqlite3.version)
    except Error as e:
        print(e)
    finally:
        conn.close()

create_connection()
```

Few points about above program:

- Whenever we try to establish a connection with database, there is a possibility of error due to non-existing database, authentication issues etc. So, it is always better to put the code for connection inside try-except block.
- While developing real time projects, we may need to create database connection and close it every now-and-then. Instead of writing the code for it repeatedly, it is better to write a separate function for establishing connection and call that function whenever and wherever required.
- If we give the term `:memory:` as an argument to `connect()` method, then the further operations (like table creation, insertion into tables etc) will be on memory (RAM) of the computer, but not on the hard disk.

Ex4. Write a program to create a Student database with a table consisting of student name and age. Read n records from the user and insert them into database. Write queries to display all records and to display the students whose age is 20.

```
import sqlite3
conn=sqlite3.connect('StudentDB.db')
c=conn.cursor()
c.execute('CREATE TABLE tblStudent(name text, age Integer)')

n=int(input(“Enter number of records:”))
for i in range(n):
    nm=input("Enter Name:")
    ag=int(input("Enter age:"))
    c.execute("INSERT INTO tblStudent VALUES(?,?)",(nm,ag))

conn.commit()
c.execute("select * from tblStudent ")
print(c.fetchall())

c.execute("select * from tblStudent where age=20")
print(c.fetchall())

conn.close()
```

In the above program we take a for-loop to get user-input for student’s name and age. These data are inserted into the table. Observe the question mark acting as a placeholder for user-input variables. Later we use a method fetchall() that is used to display all the records form the table in the form of a list of tuples. Here, each tuple is one record from the table.

5.3.5 Three Kinds of Keys

Sometimes, we need to build a data model by putting our data into multiple linked tables and linking the rows of those tables using some ***keys***. There are three types of keys used in database model:

- A ***logical key*** is a key that the “real world” might use to look up a row. It defines the relationship between primary keys and foreign keys. Most of the times, a UNIQUE constraint is added to a logical key. Since the logical key is how we look up a row from the outside world, it makes little sense to allow multiple rows with the same value in the table.
- A ***primary key*** is usually a number that is assigned automatically by the database. It generally has no meaning outside the program and is only used to link rows from different tables together. When we want to look up a row in a table, usually searching for the row using the primary key is the fastest way to find the row. Since primary keys are integer numbers, they take up very little storage and can be compared or sorted very quickly.

- A ***foreign key*** is usually a number that points to the primary key of an associated row in a different table.

Consider a table consisting of student details like RollNo, name, age, semester and address as shown below –

RollNo	Name	Age	Sem	Address
1	Ram	29	6	Bangalore
2	Shyam	21	8	Mysore
3	Vanita	19	4	Sirsi
4	Kriti	20	6	Tumkur

In this table, RollNo can be considered as a primary key because it is unique for every student in that table. Consider another table that is used for storing marks of students in all the three tests as below –

RollNo	Sem	M1	M2	M3
1	6	34	45	42.5
2	6	42.3	44	25
3	4	38	44	41.5
4	6	39.4	43	40
2	8	37	42	41

To save the memory, this table can have just RollNo and marks in all the tests. There is no need to store the information like name, age etc of the students as these information can be retrieved from first table. Now, RollNo is treated as a foreign key in the second table.

5.3.6 Basic Data Modeling

The relational database management system (RDBMS) has the power of linking multiple tables. The act of deciding how to break up your application data into multiple tables and establishing the relationships between the tables is called ***data modeling***. The design document that shows the tables and their relationships is called a ***data model***. Data modeling is a relatively sophisticated skill. The data modeling is based on the concept of ***database normalization*** which has certain set of rules. In a raw-sense, we can mention one of the basic rules as never put the same string data in the database more than once. If we need the data more than once, we create a numeric ***key*** (primary key) for the data and reference the actual data using this key. This is because string requires more space on the disk compared to integer, and data retrieval (by comparing) using strings is difficult compared to that with integer.

Consider the example of Student database discussed in Section 5.3.5. We can create a table using following SQL command –

```
CREATE TABLE tblStudent
        (RollNo INTEGER PRIMARY KEY, Name TEXT, age INTEGER, sem INTEGER, address TEXT)
```

Here, RollNo is a primary key and by default it will be unique in one table. Now, another take can be created as –

```
CREATE TABLE tblMarks
     (RollNo INTEGER, sem INTEGER, m1 REAL, m2 REAL, m3 REAL, UNIQUE(RollNo,sem))
```

Now, in the tblMarks consisting of marks of 3 tests of all the students, RollNo and sem are together unique. Because, in one semester, only one student can be there having a particular RollNo. Whereas in another semester, same RollNo may be there.

Such types of relationships are established between various tables in RDBMS and that will help better management of time and space.

5.3.7 Using JOIN to Retrieve Data

When we follow the rules of database normalization and have data separated into multiple tables, linked together using primary and foreign keys, we need to be able to build a SELECT that reassembles the data across the tables. SQL uses the JOIN clause to reconnect these tables. In the JOIN clause you specify the fields that are used to reconnect the rows between the tables.

Consider the following program which creates two tables tblStudent and tblMarks as discussed in the previous section. Few records are inserted into both the tables. Then we extract the marks of students who are studying in 6th semester.

```
import sqlite3

conn=sqlite3.connect('StudentDB.db')
c=conn.cursor()

c.execute('CREATE TABLE tblStudent
     (RollNo INTEGER PRIMARY KEY, Name TEXT, age INTEGER, sem
     INTEGER, address TEXT)')

c.execute('CREATE TABLE tblMarks
     (RollNo INTEGER, sem INTEGER, m1 REAL, m2 REAL, m3 REAL,
     UNIQUE(RollNo,sem))')

c.execute("INSERT INTO tblstudent VALUES(?,?,?,?,?)",
                                        (1,'Ram',20,6,'Bangalore'))
c.execute("INSERT INTO tblstudent VALUES(?,?,?,?,?)",
                                        (2,'Shyam',21,8,'Mysore'))
c.execute("INSERT INTO tblstudent VALUES(?,?,?,?,?)",
                                        (3,'Vanita',19,4,'Sirsi'))
c.execute("INSERT INTO tblstudent VALUES(?,?,?,?,?)",
                                        (4,'Kriti',20,6,'Tumkur'))
```

```
c.execute("INSERT INTO tblMarks VALUES(?,?,?,?,?)",(1,6,34,45,42.5))
c.execute("INSERT INTO tblMarks VALUES(?,?,?,?,?)",(2,6,42.3,44,25))
c.execute("INSERT INTO tblMarks VALUES(?,?,?,?,?)",(3,4,38,44,41.5))
c.execute("INSERT INTO tblMarks VALUES(?,?,?,?,?)",(4,6,39.4,43,40))
c.execute("INSERT INTO tblMarks VALUES(?,?,?,?,?)",(2,8,37,42,41))

conn.commit()

query="SELECT tblStudent.RollNo, tblStudent.Name, tblMarks.sem,
     tblMarks.m1, tblMarks.m2, tblMarks.m3
     FROM tblStudent JOIN tblMarks ON
     tblStudent.sem  = tblMarks.sem  AND
     tblStudent.RollNo = tblMarks.RollNo
     WHERE tblStudent.sem=6"

c.execute(query)
for row in c:
    print(row)

conn.close()
```

The output would be –

```
     (1, 'Ram', 6, 34.0, 45.0, 42.5)
     (4, 'Kriti', 6, 39.4, 43.0, 40.0)
```

The query joins two tables and extracts the records where RollNo and sem matches in both the tables, and sem must be 6.