CHAPTER 1

INTRODUCTION

Python is a widely used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

In the late 1980s, history was about to be written. It was that time when working on Python started. Soon after that, Guido Van Rossum began doing its application based work in December of 1989 by at Centrum Wiskunde & Informatica (CWI) which is situated in Netherland. It was started firstly as a hobby project because he was looking for an interesting project to keep him occupied during Christmas. The programming language which Python is said to have succeeded is ABC Programming Language, which had the interfacing with the Amoeba Operating System and had the feature of exception handling. He had already helped to create ABC earlier in his career and he had seen some issues with ABC but liked most of the features. After that what he did as really very clever. He had taken the syntax of ABC, and some of its good features. It came with a lot of complaints too, so he fixed those issues completely and had created a good scripting language which had removed all the flaws. The inspiration for the name came from BBC's TV Show – 'Monty Python's Flying Circus', as he was a big fan of the TV show and also he wanted a short, unique and slightly mysterious name for his invention and hence he named it Python! He was the "Benevolent dictator for life" (BDFL) until he stepped down from the position as the leader on 12th July 2018. For quite some time he used to work for Google, but currently, he is working at Dropbox.

The language was finally released in 1991. When it was released, it used a lot fewer codes to express the concepts, when we compare it with Java, C++ & C. Its design philosophy was quite good too. Its main objective is to provide code readability and advanced developer productivity. When it was released it had more than enough capability to provide classes with inheritance, several core data types exception handling and functions.

The two of the most used versions has to Python 2.x & 3.x. There is a lot of competition between the two and both of them seem to have quite a number of different fanbase.

For various purposes such as developing, scripting, generation and software testing, this language is utilized. Due to its elegance and simplicity, top technology organizations like Dropbox, Google, Quora, Mozilla, Hewlett-Packard, Qualcomm, IBM, and Cisco have implemented Python.

Python has come a long way to become the most popular coding language in the world. Python has just turned 30, but it still has that unknown charm & X factor which can be clearly seen from the fact that Google users have consistently searched for Python much more than they have searched for Kim Kardashian, Donald Trump, Tom Cruise etc.

Python has been an inspiration for many other coding languages such as Ruby, Cobra, Boo, CoffeeScript,ECMAScript,Groovy,SwiftGo,OCaml,Julia.

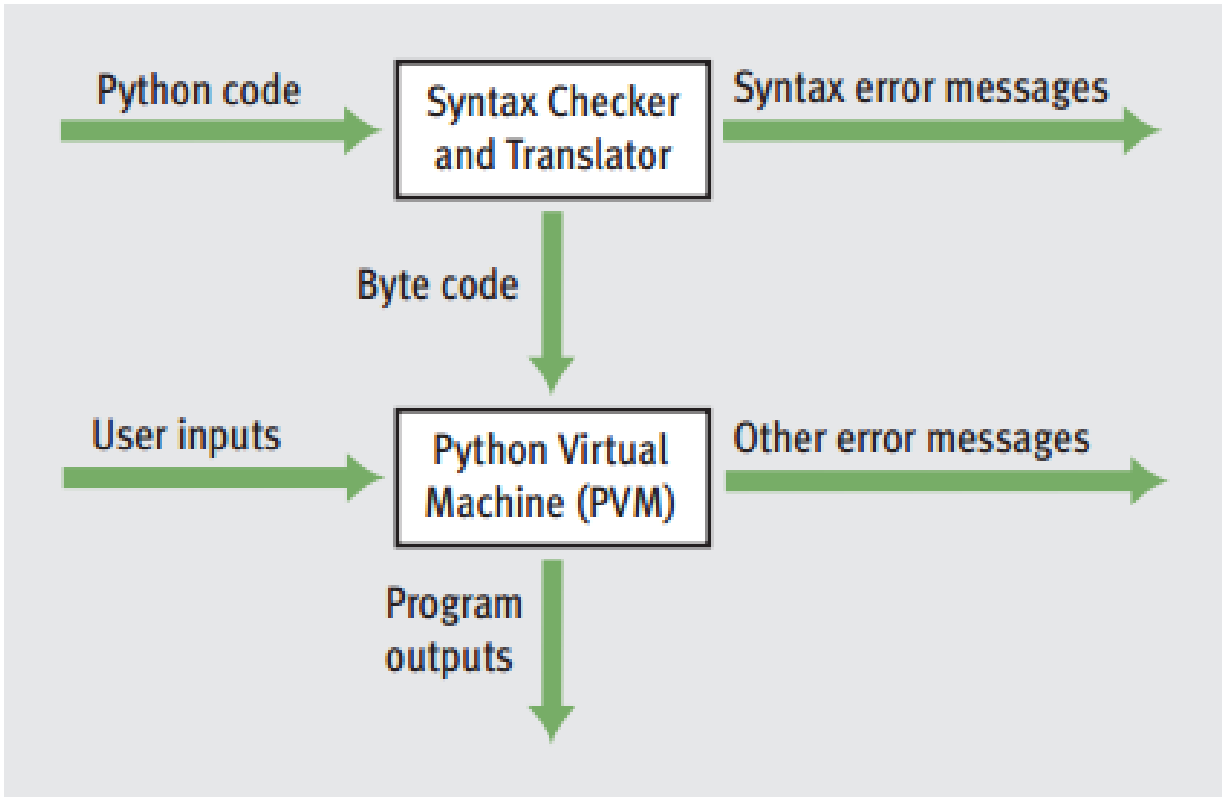

Fig 1.1 Python compiler Architecture

1.1 Features of Python.

- **Easy to Learn and Use -** Python is easy to learn and use. It is developer-friendly and high level programming language.
- **Expressive Language -** Python language is more expressive means that it is more understandable and readable.
- **Interpreted Language -** Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.
- **Cross-platform Language -** Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.
- **Free and Open Source -** Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.
- **Object-Oriented Language -** Python supports object oriented language and concepts of classes and objects come into existence.
- **Extensible -** It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.
- **Large Standard Library -** Python has a large and broad library and prvides rich set of module and functions for rapid application development.
- **GUI Programming Support -** Graphical user interfaces can be developed using Python.
- **Integrated** - It can be easily integrated with languages like C, C++, JAVA etc.

1.2 Applications of Python

- GUI based desktop applications
- Image processing and graphic design applications
- Scientific and computational applications
- Games
- Web frameworks and web applications
- Enterprise and business applications
- Operating systems
- Language development
- Prototyping

1.3 Benefits of Python

- **Presence of Third Party Modules** -The Python Package Index (PyPI) contains numerous third-party modules that make Python capable of interacting with most of the other languages and platforms.
- **Extensive Support Libraries** -Python provides a large standard library which includes areas like internet protocols, string operations, web services tools and operating system interfaces.
- **Open Source and Community Development** -Python language is developed under an OSI-approved open source license, which makes it free to use and distribute, including for commercial purposes.
- **Learning Ease and Support Available** -Python offers excellent readability and uncluttered simple-to-learn syntax which helps beginners to utilize this programming language.
- **User-friendly Data Structures** -Python has built-in list and dictionary data structures which can be used to construct fast runtime data structures. Python also provides the option of dynamic high-level data typing which reduces the length of support code that is needed.
- **Productivity and Speed** -Python has clean object-oriented design, provides enhanced process control capabilities, and possesses strong integration and text processing capabilities and its own unit testing framework, all of which contribute to the increase in its speed and productivity.

1.4 Installation of Python

- To get started working with Python 3, you'll need to have access to the Python interpreter. There are several common ways to accomplish this:
- Python can be obtained from the Python Software Foundation website at python.org. Typically, that involves downloading the appropriate installer for your operating system and running it on your machine.
- Some operating systems, notably Linux, provide a package manager that can be run to install Python.
- On macOS, the best way to install Python 3 involves installing a package manager called Homebrew. You'll see how to do this in the relevant section in the tutorial.

- On mobile operating systems like Android and iOS, you can install apps that provide a Python programming environment. This can be a great way to practice your coding skills on the go.
- Alternatively, there are several websites that allow you to access a Python interpreter online without installing anything on your computer at all.

1.5 Writing Program in Python

- When we want to write a program, we use a text editor to write the Python instructions into a file, which is called a script. By convention, Python scripts have names that end with .py.
- To execute the script, you have to tell the Python interpreter the name of the file. In a command window, you would type python hello.py as follows:
- $hello.py print('Hello world!') $ python hello.py Hello world!
- The “$” is the operating system prompt, and the “hello.py” is showing us that the file “hello.py” has a one-line Python program to print a string.
- We call the Python interpreter and tell it to read its source code from the file “hello.py” instead of prompting us for lines of Python code interactively.
- You will notice that there was no need to have quit() at the end of the Python program in the file. When Python is reading your source code from a file, it knows to stop when it reaches the end of the file.

CHAPTER 2

ORGANIZATION OVERVIEW

PHYTEC Embedded Systems Pvt. Ltd. is the largest entity within the PHYTEC Technologies Holding AG. PHYTEC develops and produces microprocessor-based solutions for the global, industrial embedded market at its headquarters in Mainz. Our range of products and services include System on Modules, Single Board Computer and custom products as well as housing design and der complete assembly. PHYTEC also offers Internet of Things solutions, Embedded Imaging as well as hardware specific software. The majority family-owned enterprise PHYTEC employs more than 200 people in 5 locations worldwide. PHYTEC solutions have been deployed in thousands of systems across a wide range of industries and applications, such as control and automation, medical, test and measurement, automotive, energy, transportation for more. Typical series production quantities range between 100 and several tens of thousands of products per year. We provide end to end solutions for companies seeking full product development support from a single source, enabling them to shorten time-to-market, reduce development costs and avoid substantial design risk in bringing products to market. PHYTEC designs and manufactures System on Module subassemblies, Single Board Computers and Rapid Development Kits that accelerate microprocessor-based embedded developments with about 200 employees at five company sites – Mainz, Seattle, Le Mans, Bangalore and Shenzhen. Developing and producing at our corporate headquarters in Mainz has been a conscious decision. All main departments such as purchasing, development, manufacturing, logistics, sales, marketing and even bookkeeping are located at headquarters. PHYTEC's advantage is increased flexibility due to high level of communication between development and manufacturing teams. Our customers are at the center of our business. Still today PHYTEC is a family business. We place a strong emphasis on employee satisfaction, a healthy company growth as well as socially responsible corporate management. PHYTEC fosters a flat organizational hierarchy as well as an open book management approach that is part of our lean management philosophy. Turnover is very low, which attests to our corporate culture. PHYTEC remains a majority family-owned enterprise, with minority ownership shares held by the managers of our operational divisions in Germany, North America, France, and India.

PHYTEC

Fig 2.1 Logo of the company

2.1 Company Structure

Phytec was Founded in 1986, PHYTEC Mess technik GmbH is the largest entity within the PHYTEC Technologies Holding AG. PHYTEC remains a majority family-owned enterprise, with minority ownership shares held by the managers of our operational divisions in Germany, North America, France, and India. With no debt or external financing, PHYTEC has funded its continued growth and expanding international presence entirely through operations.

2.2 Company Independence

As a global family business, we take responsibility for all stakeholders. With no debt and an equity ratio of over 68% (in 2013) PHYTEC is financially independent and remains a majority family-owned enterprise with minority ownership shares held by the managers of our operational divisions. This safeguards our success, renders PHYTEC a highly stable employer, and makes us a reliable and crisis-proof partner for our customers. Even during the economic turmoil of recent years we have demonstrated that our goals and values are long-term and provided superior short-term stability. The basis for this sustained growth is our mission to provide our customers with better performance and sustainable competitiveness by consistently offering products and services based on the newest embedded technologies.

2.3 Company Financials and Growth

PHYTEC has achieved positive operating results during the past 15 years. PHYTEC pursues a strategy of expansion based on continued healthy, organic growth.

Our engaged employees anticipate market trends, find innovative solutions and secure decisive competitive advantages for our customers as well as PHYTEC. Optimized use of the newest embedded technology advances leads to prolonged product life-cycles and maximized returns on development investments. PHYTEC has doubled its overall financial performance over the last five years. In 2014 worldwide sales have exceeded 29 million Euro.

CHAPTER 3

BASIC CONCEPTS OF PYTHON

3.1 Variables

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

Python supports four different numerical types :

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Table 3.1 Shows examples of Variable types -

Int	Long	Float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.2	45.j
-786	0xDEFABCECBDAECBFBAEI	-21.9	9.322e-36j
080	535633629843L	32.3+e18	.876j
-0490	-052318172735L	-90	-.6545+0j
-0x69	-4721885298529L	-32.54e100	3e+26j

3.2 String

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable. For example −

```
var = 'Hello World!'
```

3.2.1 Accessing values in strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring. To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example –

```
var = 'Hello World!'
print "var[0]: ", var[0]
```

When the above code is executed, it produces the following result –

```
var[0]:  H
```

3.2.2 Updating string

Using "update" we can add or delete an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. For example –

```
var = 'Hello World!'
print "Updated String :- ", var[:6] + 'Python'
```

When the above code is executed, it produces the following result –

```
Updated String :  Hello Python
```

3.2.3 String operation

The string consists of characters, where various operations can be performed on a string and the below table gives the list of string operations.

Table 3.2 Shows string operations

Operator	Description
+	Concatenation - Adds values on either side of the operator
*	Repetition - Creates new strings, concatenating multiple copies of the same string
[]	Slice - Gives the character from the given index
[:]	Range Slice - Gives the characters from the given range
In	Membership - Returns true if a character exists in the given string
not in	Membership - Returns true if a character does not exist in the given string
t/R	Raw String - Suppresses actual meaning of Escape characters.
%	Format - Performs String formatting

3.3 Files

When we want to read or write a file (say on your hard drive), we first must open the file. Opening the file communicates with your operating system, which knows where the data for each file is stored. When you open a file, you are asking the operating system to find the file by name and make sure the file exists. In this example, we open the file mbox.txt, which should be stored in the same folder that you are in when you start Python.

3.3.1 Open a file

When we want to read or write a file (say on your hard drive), we first must open the file. Opening the file communicates with your operating system, which knows where the data for each file is stored. When you open a file, you are asking the operating system to find the file by name and make sure the file exists. Before you can read or write a file, you have to open it using Python's built-in open() function.

Syntax:

fileObject = open(file_name [	access_mode][	buffering])

The parameter details are −

- File_name − The file_name argument is a string value that contains the name of the file that you want to access.
- access_mode − The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- buffering − If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Table 3.3 Shows list of different modes of opening a file

Sr.No	Modes	Description
1	r	Opens a file for reading only.
2	r+	Opens a file for both reading and writing.
3	w	Opens a file for writing only.
4	w+	Opens a file for both writing and reading.
5	a	Opens a file for appending.
6	a+	Opens a file for both appending and reading.

3.3.2 Close a file

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax:

```
fileObject. close()
```

3.3.3 Reading a file

The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax –

```
fileObject.read([count])
```

3.3.4 Writing a file

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text. The write() method does not add a newline character ('\n') to the end of the string.

Syntax –

```
fileObject. Write(string)
```

3.4 Lists

A string, a list is a sequence of values. In a string, the values are characters; in a list, they can be any type. The values in list are called elements or sometimes items. The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

3.4.1 List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Table 3.4 Shows list operation

Expression	Description	Result
len([1,2,3])	Length	3
[1,2,3]+[4,5,6]	Concatenation	[1,2,3,4,5,6]
[“Hi!”]*4	Repetition	[“Hi!”,”Hi!”,”Hi!”,”Hi!”]

3.4.2 List functions

Python has some built in functions which can be used on list to perform the functions. The sum() function only works when the list elements are numbers. The other functions (max(), len(), etc.) work with lists of strings and other types that can be comparable.

Table 3.5 Shows functions of list

Sr.No	Function	Description
1	cmp(list)	Compares elements of both lists.
2	len(list)	Gives the total length of the list.
3	max(list)	Returns item from the list with max value.
4	min(list)	Returns item from the list with min value.
5	list(seq)	Converts a tuple into list.

3.4.3 List Methods

Python provides methods that operate on lists.

Table 3.6 shows list methods

Sr.No	Method	Description
1	list.append(obj)	Appends objects obj to list
2	list.count(obj)	Returns count of how many times obj occurs in list
3	list.exend(seq)	Appends the contents of seq to list
4	list.index(obj)	Returns the lowest index in list
5	list.insert(index,obj)	Inserts object obj into list
6	list.pop(obj=list[-1])	Removes the element in list
7	list.remove(obj)	Removes object from list
8	list.reverse(obj)	Removes object of list in place
9	list.sort([func])	Sorts object of list

3.4.4 Deleting elements in the list

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know. For example

```
list = ['physics', 'chemistry', 1997, 2000];

print list

del list[2];

print "After deleting value at index 2 : "

print list
```

When the above code is executed, it produces following result –

```
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

CHAPTER 4

NETWORKED 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. 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.

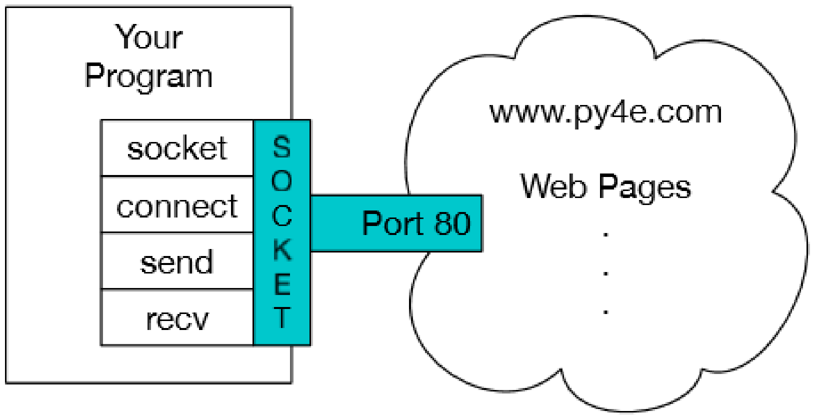

Fig 4.1 A Socket connection

The output starts with headers which the web server sends to describe the document. For example, the Content-Type header indicates that the document is a plain text document (text/plain). After the server sends us the headers, it adds a blank line to indicate the end of the headers, and then sends the actual data of the file romeo.txt. This example shows how to make a low-level network connection with sockets. Sockets can be used to communicate with a web server or with a mail server or many other kinds of servers.

All that is needed is to find the document which describes the protocol and write the code to send and receive the data according to the protocol. However, since the protocol that we use most commonly is the HTTP web protocol, Python has a special library specifically designed to support the HTTP protocol for the retrieval of documents and data over the web. One of the requirements for using the HTTP protocol is the need to send and receive data as bytes objects, instead of strings. In the preceding example, the encode() and decode() methods convert strings.

4.1 Using web service

Once it became easy to retrieve documents and parse documents over HTTP using programs, it did not take long to develop an approach where we started producing documents that were specifically designed to be consumed by other programs (i.e., not HTML to be displayed in a browser). There are two common formats that we use when exchanging data across the web. eXtensible Markup Language (XML) has been in use for a very long time and 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 (JSON) . We will look at both formats. XML looks very similar to HTML, but XML is more structured than HTML.

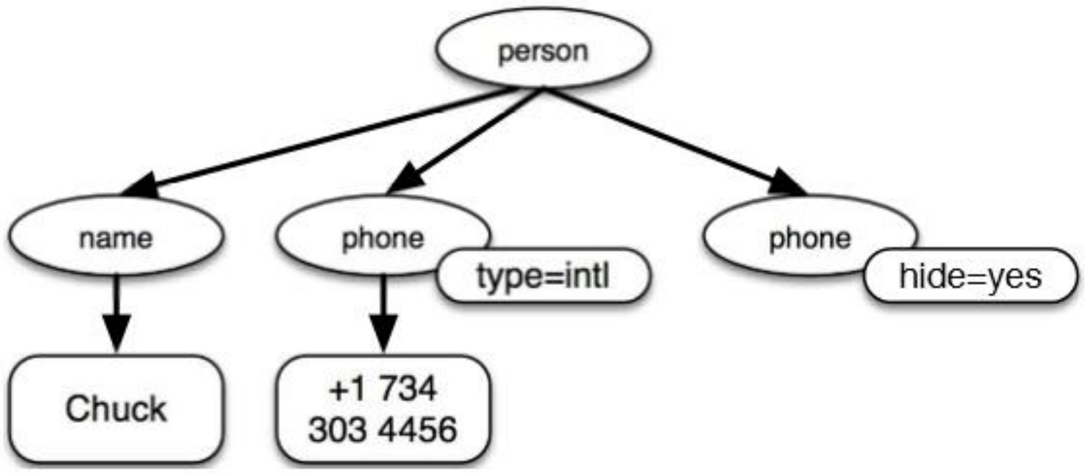

Fig 4.2 Tree representation of XML

4.1.1 Parsing XML

XML stands for eXtensible Markup Language. It was designed to store and transport data. It was designed to be both human- and machine-readable. That’s why, the design goals of XML emphasize simplicity, generality, and usability across the Internet. Using an XML parser such as ElementTree has the advantage that while the XML in this example is quite simple, it turns out there are many rules regarding valid XML, and using ElementTree allows us to extract data from XML without worrying about the rules of XML syntax.

Here is a simple application that parses some XML and extracts some data elements from the XML:

```
import xml.etree.ElementTree as ET

data = '''

<person>

   <name>Chuck</name>

     <phone type="intl"> +1 734 303 4456 </phone>

       <email hide="yes" />

</person>'''

tree = ET.fromstring(data)

print('Name:', tree.find('name').text)

print('Attr:', tree.find('email').get('hide'))
```

The triple single quote ('''), as well as the triple double quote ("""), allow for the creation of strings that span multiple lines. Calling from string converts the string representation of the XML into a “tree” of XML elements. When the XML is in a tree, we have a series of methods we can call to extract portions of data from the XML string. The find function searches through the XML tree and retrieves the element that matches the specified tag.

```
Name: Chuck

Attr: yes
```

4.1.2 Parsing 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 nearly identical to a combination of Python lists and dictionaries. 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 cooperating 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. The JSON is more succinct (an advantage) but also is less self-describing (a disadvantage).

```
import json
data = ' ' '
[
  {
       "id" : "001",
        "x" : "2",
         "name" : "Chuck"
       } ,
      {
         "id" : "009",
          "x" : "7",
           "name" : "Brent"
         }
         ]' ' '
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 of this program is exactly the same as the XML version above.

```
User count: 2

Name: Chuck

 Id: 001

Attribute: 2

Name: Brent

Id: 009

Attribute: 7
```

4.2 Object Oriented Programming

Python is a multi-paradigm programming language. Meaning, it supports different programming approach. One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP). Object oriented programming is a way to arrange your code so that you can zoom into 50 lines of the code and understand it while ignoring the other 999,950 lines of code for the moment. Like many aspects of programming, it is necessary to learn the concepts of object oriented programming before you can use them effectively. You should approach this chapter as a way to learn some terms and concepts and work through a few simple examples to lay a foundation for future learning. The key outcome of this chapter is to have a basic understanding of how objects are constructed and how they function and most importantly how we make use of the capabilities of objects that are provided to us by Python and Python libraries.

4.2.1 Using Object

we define a class (template), use that class to create an instance of that class (object), and then use the instance. When the program finishes, all of the variables are discarded. Usually, we don’t think much about the creation and destruction of variables, but often as our objects become more complex, we need to take some action within the object to set things up as the object is constructed and possibly clean things up as the object is discarded.

If we want our object to be aware of these moments of construction and destruction, we add specially named methods to our object:

```
class PartyAnimal:
x = 0
def __init__(self):
print('I am constructed')
def party(self) :
self.x = self.x + 1
print('So far',self.x)
def __del__(self):
print('I am destructed', self.x)
an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

When this program executes, it produces the following output:

```
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

As Python constructs our object, it calls our __init__ method to give us a chance to set up some default or initial values for the object. When Python encounters the line:

```
an = 42
```

It actually “throws our object away” so it can reuse the an variable to store the value 42. Just at the moment when our an object is being “destroyed” our destructor code (__del__) is called. We cannot stop our variable from being destroyed, but we can do any necessary cleanup right before our object no longer exists. When developing objects, it is quite common to add a constructor to an object to set up initial values for the object. It is relatively rare to need a destructor for an object.

4.2.2 Multithreading

A Thread or a Thread of Execution is defined in computer science as the smallest unit that can be scheduled in an operating system. Threads are normally created by a fork of a computer script or program in two or more parallel (which is implemented on a single processor by multitasking) tasks. Threads are usually contained in processes. More than one thread can exist within the same process. These threads share the memory and the state of the process. In other words: They share the code or instructions and the values of its variables. Multithreaded programs can run faster on computer systems with multiple CPUs, because theses threads can be executed truly concurrent. A program can remain responsive to input. This is true both on single and on multiple CPU. Threads of a process can share the memory of global variables. If a global variable is changed in one thread, this change is valid for all threads. A thread can have local variables.

Syntax -

thread.start_new_thread (function	args[	kwargs])

4.2.3 Inheritance

Inheritance is the core feature of object-oriented programming which extends the functionality of an existing class by adding new features. You may compare it with real-life situations when a child inherits the property of his parents in addition to adding his own. He may even derive the surname (the second name) from his parents. By using the inheritance feature, we can have a new blueprint with old attributes but without making any changes to the original one. We refer to the new class as the derived or child class whereas the old one becomes the base or parent class. For this example, we move our PartyAnimal class into its own file. Then, we can ‘import’ the PartyAnimal class in a new file and extend it, as follows:

```
from party import PartyAnimal
class CricketFan(PartyAnimal):
points = 0 def six(self):
self.points = self.points + 6
self.party()
print(self.name,"points",self.points)
s=PartyAnimal("Sally")
```

```
s.party()
j=CricketFan("Jim")
j.party()
j.six()
print(dir(j))
```

As the program executes, we create s and j as independent instances of PartyAnimal and CricketFan. The j object has additional capabilities beyond the s object.

```
Sally constructed
Sally party count 1
Jim constructed
Jim party count 1
Jim party count 2
Jim points 6
['__class__', '__delattr__', ... '__weakref__', 'name', 'party', 'points', 'six', 'x’]
```

CHAPTER 5

USING DATABASE AND SQL

A database is a file that is organized for storing data. Most databases are organized like a dictionary in the sense that they map from keys to values. The biggest difference is that the database is on disk (or other permanent storage), so it persists after the program ends. Because a database is stored on permanent storage, it can store far more data than a dictionary, which is limited to the size of the memory in the computer. Like a dictionary, database software is designed to keep the inserting and accessing of data very fast, even for large amounts of data. Database software maintains its performance by building indexes as data is added to the database to allow the computer to jump quickly to a particular entry. There are many different database systems which are used for a wide variety of purposes including: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, and SQLite. We focus on SQLite in this book because it is a very common database and is already built into Python. 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 as do many other products. SQLite is well suited to some of the data manipulation problems that we see in Informatics such as the Twitter spidering application. In technical descriptions of relational databases the concepts of table, row, and column are more formally referred to as relation, tuple, and attribute, respectively.

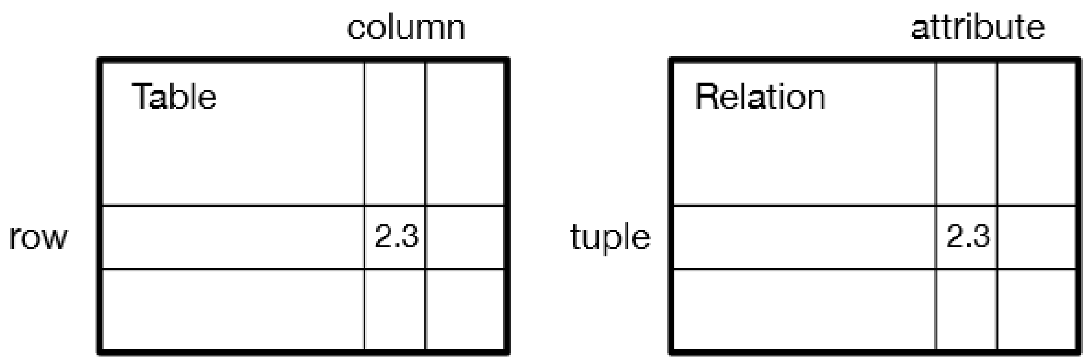

Fig 5.1 Relational database

5.1 Creating a database table

Databases require more defined structure than Python lists or dictionaries1. When we create a database table we must tell the database in advance the names of each of the columns in the table and the type of data which we are planning to store in each column. 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. Defining structure for your data up front may seem inconvenient at the beginning, but the payoff is fast access to your data even when the database contains a large amount of data. The code to create a database file and a table named Tracks with two columns in the database is as follows:

```
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 operation makes a "connection" to the database stored in the file music.sqlite in the current directory. If the file does not exist, it will be created. The reason this is called a "connection" is that sometimes the database is stored on a separate "database server" from the server on which we are running our application. In our simple examples the database will just be a local file in the same directory as the Python code we are running. 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.

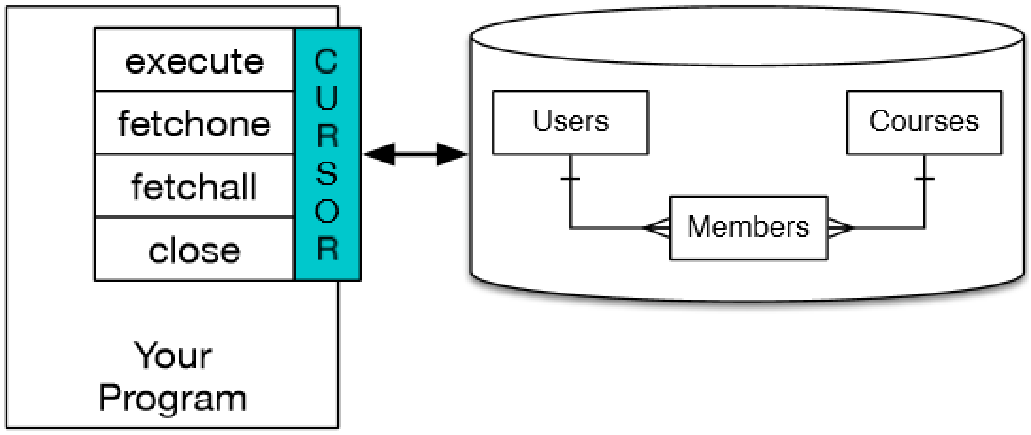

Fig 5.2 Shows a database cursor

5.2 Inserting into table

Python allows us to insert data into the database through SQL. The user can insert name, id, phone etc into the database. It is required when you want to create your records into a database table. You can add new rows to an existing table of SQLite using the INSERT INTO statement. In this, you need to specify the name of the table, column names, and values. While inserting records using the INSERT INTO statement, if you skip any columns names, this record will be inserted leaving empty spaces at columns which you have skipped. You can also insert records into a table without specifying the column names, if the order of values you pass is same as their respective column names in the table. Create a connection object using the connect() method by passing the name of the database as a parameter to it. The cursor() method returns a cursor object using which you can communicate with SQLite3. Create a cursor object by invoking the cursor() object on the (above created) Connection object. Then, invoke the execute() method on the cursor object, by passing an INSERT statement as a parameter to it. For example –

```
import MySQLdb
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
LAST_NAME, AGE, SEX, INCOME)
VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
cursor.execute(sql)
db.commit()
except:
db.rollback()
db.close()
```

5.3 Delete database in table

DELETE operation is required when you want to delete some records from your database. To delete records from a MySQL table, you need to use the DELETE FROM statement. To remove specific records, you need to use WHERE clause along with it. It is considered a good practice to escape the values of any query, also in delete statements. This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database. The mysql.connector module uses the placeholder %s to escape values in the delete statement. For example –

```
import MySQLdb
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
cursor.execute(sql)
db.commit()
except:
db.rollback()
db.close()
```

5.4 Spidering twitter using database

One of the problems of any kind of spidering program is that it needs to be able to be stopped and restarted many times and you do not want to lose the data that you have retrieved so far. You don’t want to always restart your data retrieval at the very beginning so we want to store data as we retrieve it so our program can start back up and pick up where it left off. We will start by retrieving one person’s Twitter friends and their statuses, looping through the list of friends, and adding each of the friends to a database to be retrieved in the future. After we process one person’s Twitter friends, we check in our database and retrieve one of the friends of the friend. We do this over and over, picking an “unvisited” person, retrieving their friend list, and adding friends we have not seen to our list for a future visit. We also track how many times we have seen a particular friend in the database to get some sense of their “popularity”. By storing our list of known accounts and whether we have retrieved the account or not, anhow popular the account is in a database on the disk of the computer, we can stop and restart our program as many times as we like. This program is a bit complex. It is based on the code from the exercise earlier in the book that uses the Twitter API.

CONCLUSION

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. often the quickest way to debug a program is to add a few print statements to the source. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Python language supports both object oriented and procedure oriented approach. Python is designed to be a highly extensible language. Python works on the principle of “there is only one obvious way to do a task” rather than “there is more than one way to solve a particular problem”. Python is very easy to learn and implement. The simpler syntax, uncomplicated semantics and approach with which Python has been developed makes it very easier to learn. A large number of python implementations and extensions have been developed since its inception.

REFERENCES

[1] Charles R. Severance, “Python for Everybody: Exploring Data Using Python 3”, 1st Edition, CreateSpace Independent Publishing Platform, 2016.

[2] Allen B. Downey, "Think Python: How to Think Like a Computer Scientist”, 2ndEdition, Green Tea Press, 2015.

[3] G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995."