

Advance Core Python Programming

Begin your Journey to Master
the World of Python



MEENU KOHLI

bpb

Advance Core Python Programming

Begin your Journey to Master
the World of Python



MEENU KOHLI



Advance Core Python Programming

*Begin your Journey to Master
the World of Python*

Meenu Kohli



www.bpbonline.com

FIRST EDITION 2021

Copyright © BPB Publications, India

ISBN: 978-93-90684-06-9

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Distributors:

BPB PUBLICATIONS

20, Ansari Road, Darya Ganj

New Delhi-110002

Ph: 23254990/23254991

MICRO MEDIA

Shop No. 5, Mahendra Chambers,

150 DN Rd. Next to Capital Cinema,

V.T. (C.S.T.) Station, MUMBAI-400 001

Ph: 22078296/22078297

DECCAN AGENCIES

4-3-329, Bank Street,

Hyderabad-500195

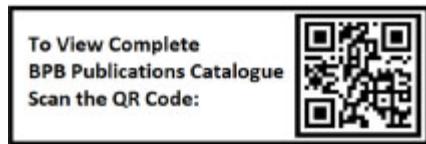
Ph: 24756967/24756400

BPB BOOK CENTRE

376 Old Lajpat Rai Market,

Delhi-110006

Ph: 23861747



Published by Manish Jain for BPB Publications, 20 Ansari Road, Darya Ganj, New Delhi-110002
and Printed by him at Repro India Ltd, Mumbai

www.bpbonline.com

Dedicated to

My Husband, Capt. Rishi Raj Kohli

*I dreamt an impossible dream; you gave me the wings to chase it,
and make it a reality.*

About the Author

Meenu Kohli, author of “Python Interview Questions” and “Basic Core Python Programming” is presenting her next venture – “Advance Core Python Programming”.

A BE (Electronics) from D.Y. Patil College of Engineering, Pune University, she has worked extensively as a Software Developer, Tester, and Trainer in reputed MNCs. She has experience of working on challenging projects across varied development environments such as Python, Java, EJB, C, C++, PHP, JSP, JavaScript, HTML, .NET, R, MySQL, Oracle, DB2.

She is a keen observer and passionate learner. In addition to software development, she is also a professionally trained software tester and a certified Six Sigma Green Belt from the National Institute of Industrial Engineering – Mumbai. She has experience of teaching Electronics and Computer Science undergraduate students and has also imparted her knowledge to several software-development courses. She later diversified into writing technical books primarily related to Python.

About the Reviewers

Prateek Gupta is a Data Enthusiast who loves data-driven technologies. Prateek has done his B.Tech in Computer Science & Engineering. He is currently working as a Data Scientist in an IT company. Prateek has 10 years of experience in the software industry, and is currently, working in the Computer Vision area. Prateek is also the author of the book “Practical Data Science with Jupyter”, 2nd Edition, published by the BPB Publications.

Shayank Jain is a software developer, a data analyst, and an Author. He is passionate about coding and architectural design. He has more than 7.5+ years of professional experience in developing scalable software solutions for various organizations. He has been programming since the age of 16 and has developed software for mobile, web, hardware gaming, and standalone applications. After getting his hands dirty in programming, he found many new ways to debug and deploy the code successfully, with minimal time constraints. After reading and implementation, he found out that many critical concepts can be implemented easily into programming with correct and focused thinking. His research interests include information security, cryptography, analysis, design, and implementation of algorithms. He has extensively worked with python and implemented new ideas on various projects in his free time. He is also active in the computer science and education community.

Acknowledgements

I would like to thank my husband and my two lovely children, Aananya and Ranbeer for all the help, support, and encouragement, without which I would not have been able to finish this book. When the lockdown was announced in early 2020, I thought my work would get affected, but my family ensured that I continue to work as per my schedule and finish the book on time.

Thank you to BPB for allowing me to write this book and thank you for your support.

Preface

There are several reasons why Python stands out from the other programming languages. The fact that you are right now holding a book on Advance level core Python Programming, indicates that you are already aware of the features that make Python so special.

In this book, you will get detailed information on the advanced topics related to Python programming. It starts with the chapter on Functions. I would, therefore, like to suggest that kindly brush up your knowledge on the Python Basic concepts before you start with this book. If you are an absolute beginner, I would suggest that you first read ‘Basic Core Python Programming’, which is also a BPB publication, so that you can follow the topics given in this book.

While working on this book, my main focus was on what is being taught in the most leading universities and what is in demand. I also focused on what problems the students and professionals face while learning to programme. By interacting with other programming enthusiasts, I realised that many people prefer self-study as they are busy with several other commitments or just cannot afford to join a course.

I have designed this book like a self-learning course, providing detailed steps and breaking complex problems into simpler ones that can be easily be coded and then put back together. I am confident that with this book, you would be able to think, design, and create your Python applications.

This book is divided into the following 14 chapters:

[**Chapter 1:**](#) Functions

[**Chapter 2:**](#) Classes, Objects, and inheritance

[**Chapter 3:**](#) Files

[**Chapter 4:**](#) MySQL for Python

[**Chapter 5:**](#) Python Threads

[**Chapter 6:**](#) Errors, Exceptions, Testing, and Debugging

[**Chapter 7:**](#) Data Visualization and Data Analysis

[Chapter 8:](#) Creating GUI Form and Adding Widgets

[Chapter 9:](#) My SQL and Python Graphical User Interface

[Chapter 10:](#) Stack, Queue, and Deque

[Chapter 11:](#) Linked Lists

[Chapter 12:](#) Trees

[Chapter 13:](#) Searching and Sorting

[Chapter 14:](#) Getting Started with Flask

I hope you enjoy reading it as much as I enjoyed working on it.

Happy reading and God bless!

Downloading the code bundle and coloured images:

Please follow the link to download the *Code Bundle* and the *Coloured Images* of the book:

<https://rebrand.ly/fed28b>

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePUB files available? You can upgrade to the eBook version at **www.bpbonline.com** and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at **business@bpbonline.com** for more details.

At **www.bpbonline.com**, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.



BPB is searching for authors like you

If you're interested in becoming an author for BPB, please visit www.bpbonline.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

The code bundle for the book is also hosted on GitHub at <https://github.com/bpbpublications/Advance-Core-Python-Programming>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/bpbpublications>. Check them out!

PIRACY

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at business@bpbonline.com with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com.

REVIEWS

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential

readers can then see and use your unbiased opinion to make purchase decisions, we at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Table of Contents

1. Functions and Recursion

Introduction

Structure

Objective

1.1 What are functions?

1.2 Creating Functions

 1.2.1 Simple Functions

 1.2.1.1 Definition of a function

 1.2.1.2 Function Body

 1.2.1.3 Calling a function

 1.2.2 Defining functions that take parameter

1.3 Functional arguments

 1.3.1 Positional arguments

 1.3.2 Default arguments

 1.3.3 Keyword arguments

 1.3.4 *args

 1.3.5 **kwargs

1.4 The return statement

1.5 Scopes and namespace

 1.5.1 Built-in namespace

 1.5.2 Global namespace

 1.5.3 Local namespace

1.6 Lambda functions

1.7 Recursion

 1.7.1 Finding factorial of a number using recursion

 1.7.2 Algorithm for finding factorial using recursion

 1.7.3 Types of recursion

 1.7.4 Advantages and disadvantages of recursion

1.8 Memoization

Points to remember

Conclusion

Short questions and answers

Coding questions and answers

Descriptive questions and answers

2. Classes, Objects, and Inheritance

Introduction

Structure

Objective

2.1 Classes and objects

 2.1.1 The Employee class

 2.2 Destructor `__del__()`

 2.3 Types of class variables

 2.4 Inheritance

Points to remember

Conclusion

Multiple choice questions

Answers

Short questions and answers

Coding questions and answers

Descriptive questions and answers

3. Files

Introduction

Structure

Objective

3.1 Advantages of storing data in files

3.2 Directory and file management with Python

3.3 Retrieve the directory current working directory using `getcwd()`

3.4 Checking contents of Python directory using `listdir()`

3.5 Create directory using `mkdir()`

3.6 Renaming a directory using `rename()`

3.7 Remove a directory using `rmdir()`

3.8 Working with files

 3.8.1 `open()` and `close()`

 3.8.2 Access mode in file writing

 3.8.3 Writing to a binary file using “wb” access mode

 3.8.4 Reading the contents of a binary file using “rb” access mode

 3.8.5 Write and read strings with binary files

 3.8.6 Other operations on binary file

3.8.7 File Attributes

3.9 Various File related methods

Conclusion

Multiple choice questions

Answer

Short questions and answers

Coding questions and answers

Descriptive questions and answers

4. MySQL for Python

Introduction

Structure

Objectives

4.1 Installing and configuring MySql on your system

 4.1.1 How to install MySQL

 4.1.2 Configuration of MySQL

4.2 Creating a database in MySQL using the command line tool

 4.2.1 Basic rules for writing SQL queries

4.3 Connect MySQL database to Python using MySQL connector

 4.3.1 Create a database in Mysql using Python

 4.3.2 Retrieve records from database using Python

4.4 Creating a database

 4.4.1 Creating database directly using MySQL

 4.4.2 Creating database using Python

4.5 Working with database using Python

 4.5.1 Inserting records

 4.5.2 Select records

 4.5.2.1 Fetching all records from a table in Python using fetchall().

 4.5.2.2 Fetching one record from a table in Python using fetchone().

 4.5.3 Fetching selected records from a table in Python using the 'WHERE' clause

 4.5.4 Using ORDER BY clause

 4.5.5 Deleting records using DELETE command

 4.5.6 Update clause

Conclusion

Questions and answers

5. Python Threads

Introduction

Structure

Objectives

5.1 Processes and threads

 5.1.1 Process

 5.1.2 Threads

5.2 How to create a thread

 5.2.1 Implementation of new thread using threading module

5.3 Thread synchronization with Lock and RLock

5.4 Applying lock

5.5 Deadlock

 5.5.1 Using locked().function to check if a resource is locked

 5.5.2 Resolving with RLOCK

5.6 Semaphore

5.7 Thread synchronization using an event object

5.8 Condition Class

5.9 Daemon and Non-DaemonThread

Conclusion

Questions and answers

6. Errors, Exceptions, Testing, and Debugging

Introduction

Structure

Objectives

6.1 What is an error?

 6.1.1 Syntax Errors

 6.1.2 Runtime error

 6.1.3 Logical error

6.2 Exception

6.3 Errors with respect to Python

 6.3.1 Try and Catch

 6.3.2 Catching exception in general

 6.3.3 Try ... Except...else Statement

 6.3.4 Try...Except...Finally...

6.3.5 Try and Finally

6.3.6 Raise an exception

6.4 How to debug a program?

6.5 ‘pdb’ debugger

6.6 Debugger at command line

6.7 Unit testing and test-driven development in Python

6.7.1 Why do we unit test?

6.7.2 Objectives of unit testing

6.8 Levels of testing

6.8.1 Unit testing

6.8.2 Integration testing

6.8.3 System-level testing

6.8.4 Performance testing

6.8.5 Unit testing in Python

6.8.6 Frequent errors encountered in Python programming

6.9 What is pytest?

6.10 The unittest module

6.11 Defining multiple test cases with unittest and pytest:

6.12 List of Assert methods available in the ‘unittest’ module.

Conclusion

Questions and answers

7. Data Visualization and Data Analysis

Introduction

Structure

Objectives

7.1 Introduction to Data Visualization

7.1.1 Why Data Visualization?

7.2 Matplotlib

7.2.1 Working with pyplot

7.2.2 Plotting a point

7.2.3 Plotting multiple points

7.2.4 Plotting a line

7.2.5 Labelling the x and y axis

7.3 numPy

7.3.1 Installing numpy

7.3.2 Shape of numpy Arrays

[7.3.3 How to get value of elements from Array](#)

[7.3.4 Creating numpy arrays](#)

[7.4 pandas](#)

[7.4.1 Creating dataframe from an Excel sheet](#)

[7.4.2 Creating dataframe from .csv file](#)

[7.4.3 Creating Dataframe from Dictionary](#)

[7.4.4 Creating Data Frame from list of tuples](#)

[7.5 Operations on DataFrame](#)

[7.5.1 Retrieving rows and columns](#)

[7.5.2 Working with columns](#)

[7.5.3 Retrieving data from multiple columns](#)

[7.5.4 Retrieving data based on conditions](#)

[7.5.5 Index range](#)

[7.5.6 Reset index](#)

[7.5.7 Sorting data](#)

[Conclusion](#)

8. Creating GUI Form and Adding Widgets

[Introduction](#)

[Structure](#)

[Objectives](#)

[8.1 Getting started](#)

[8.2 Introduction to widgets](#)

[8.2.1 Layout management](#)

[8.2.1.1 pack](#)

[8.2.1.2 Grid](#)

[8.2.1.3 Place](#)

[8.3 Working with buttons and Messagebox](#)

[8.3.1 Code for displaying a button](#)

[8.4 Canvas](#)

[8.4.1 Writing text on Canvas](#)

[8.5 Frame](#)

[8.6 Working with Labels](#)

[8.7 Mini Project - Stop Watch](#)

[8.8 List Box Widget](#)

[8.9 The Menu button and Menu](#)

[8.10 Radiobutton](#)

[8.11 Scrollbar and Sliders](#)

[8.12 Text](#)

[8.13 Spinbox](#)

[Points to remember](#)

[Questions and answers](#)

9. MySQL and Python Graphical User Interface

[Introduction](#)

[Structure](#)

[Objectives](#)

[9.1 MySQLdb Database](#)

[9.2 Creating a table using GUI](#)

[9.3 Insert data using GUI](#)

[9.4 Create a GUI to retrieve results](#)

[Activity](#)

[Conclusion](#)

10. Stack, Queue, and Deque

[Introduction](#)

[Structure](#)

[Objectives](#)

[10.1 Stack](#)

[10.1.1 Implementation of a stack in Python](#)

[10.2 Queue](#)

[10.2.1 Basic queue functions](#)

[10.2.2 Implementation of Queue](#)

[10.2.3 Implementation of a stack using single queue](#)

[10.2.4 Implementation of a queue using two stacks](#)

[10.3 Deque](#)

[10.3.1 Write a code to implement a deque](#)

11. Linked List

[Introduction](#)

[Structure](#)

[Objective](#)

[11.1 Introduction to Linked Lists](#)

[11.2 Implementation of Node class](#)

- 11.2.1 Traversing through a linked list
- 11.2.2 How to add a node at the beginning of a linked list
- 11.2.3 How to add a Node at the end of a linked list
- 11.2.4 Inserting a node between two nodes in a linked list
- 11.2.5 Removing a node from a linked list
- 11.2.6 Printing the values of the node in the centre of a linked list
- 11.2.7 Implementation of doubly linked list
- 11.2.8 Reversing a linked list

Conclusion

12. Trees

Introduction

Structure

Objective

12.1 Introduction

12.2 Simple tree representation

12.3 Representing a tree as list of lists

- 12.3.1 Tree traversal methods

12.3.1.1 Preorder Traversal

12.3.1.2 In Order Traversal

12.3.1.3 Post Order Traversal

12.4 Binary Heap

Conclusion

13. Searching and Sorting

Introduction

Structure

Objective

13.1 Sequential search

13.2 Binary search

13.3 Hash Tables

13.4 Bubble sort

13.5 Implementation of selection sort

13.6 Insertion sort

13.7 Shell sort

13.8 Quick Sort

Conclusion

14. Getting Started with Flask

Introduction

Structure

Objective

14.1 Introduction

14.2 Installation of virtual environment

14.3 “Hello world” Application with Flask

14.4 Debugging a flask application

Conclusion

Appendix

Index

CHAPTER 1

Functions and Recursion

Introduction

You have learnt about Python basics and Python data types and control structure. In this chapter, you will learn how to create **functions**. A function is block of reusable code that is defined to carry out one particular task. Once you understand the concept of function, you will learn about scenarios where a problem can be solved by a function making call to itself. This is known as **Recursion**.

Structure

- What are functions?
- Creating functions
- Simple functions
 - Definition of a function
 - Function body
 - Calling a function
- Functional arguments
 - Positional arguments
 - Default arguments
 - Keyword arguments
 - *args
 - **kwargs
- The return statement
- Scopes and namespace
 - Built-in namespace
 - Global namespace

- Local namespace
- Lambda
- Recursion
 - Finding factorial of a number using factorial
 - Algorithm for finding factorial using recursion
 - Types of recursion
 - Advantages and disadvantages of recursion
- Memoization

Objective

After reading this chapter, you will be able to:

- Create your own functions
- Work with Lambda functions
- Use recursion and memoization

1.1 What are functions?

In this chapter, you will learn about ‘*functions*’ which is the core topic for any object-oriented programming language. Functions form the most important aspect of programming in Python. The time has come to combine and use all the concepts you have learned in previous volume of this book (Basic Core Python Programming) to create reusable functions. Functions are organized blocks of reusable code. Functions are important because:

- They provide better readability and modularity*.
- Functions help save time and effort in designing and executing the code.
- Functions reduces duplication of code.
- They make code reusable.
- They Make code easy to maintain.
- With functions it becomes easier to understand how the code works.
- Functions help in information hiding.

Note: * The act of dividing a program or code into individual independent modules is called modularity.

In Python programming, functions can be classified into two types:

- **Built-in functions**

Built-in functions are functions that are provided by Python. They are readily available. All the functions that we have worked with till now are all built-in functions such as `max()`, `min()`, `len()`, and so on.

- **User-defined functions**

This chapter is actually all about user-defined functions. These functions are not provided by Python, but are created by programmers to perform a particular task.

1.2 Creating Functions

In the last section, you learnt about advantages of working with functions. Functions make code reusable. If there is a block of code that you need to execute, again and again, you can place that block of code inside a function, and call that function whenever you need to execute that specific task. There is a software development practice by the acronym ‘*DRY*’ that stands for “*Don’t Repeat Yourself*”. Functions help in keeping the code *DRY*. This is opposite to another coding acronym called *WET*, which stands for “*Write Everything Twice*”.

1.2.1 Simple Functions

In this section, you will create your first function.

For creating a function, you will have to follow certain rules regarding:

- How to define a function
- The function body
- Calling a function

1.2.1.1 Definition of a function

1. The definition of a function starts with the ‘*def*’ keyword.
2. The **def** keyword is followed by the name of the function.

3. The name of the function is followed by **parenthesis ()**.
4. After the parenthesis comes the colon : which marks the beginning of the function's block of code.

```
def helloWorldFunc():
```

1.2.1.2 Function Body

The block of code that comes after the function definition should be indented one level to the right, which is four spaces as per PEP-8. You have learnt about this while learning about Python basics ([Chapter 2, Basic Core Python Programming](#)). You have also implemented the same rule while working with the if...else statement, and loops such as the for and while loops.

```
def hello_world_func():

    """
    This is my first function.
    I am learning a lot and this looks like fun
    This function just prints a message.
    """

    print('My first function prints HELLO WORLD!!!')
```

1.2.1.3 Calling a function

You can call the function anytime by its name followed by parenthesis.

```
hello_world_func()
```

The function does not execute as long it is called **explicitly**.

Code:

```
def hello_world_func():

    """
    This is my first function.
    I am learning a lot and this looks like fun
    This function just prints a message.
    """

    print('My first function prints HELLO WORLD!!!')
    hello_world_func()
```

You may recall that strings in triple quotes are used for multi-line commenting, and if placed directly under the definition of the function, module or class it acts as docstring. The multi-line comment in triple quotes is highlighted in bold for your understanding. It is not mandatory to use docstring but it can be of great help in understanding complex functions as you will see in a short while.

The function has only one print statement. There are some points, which are not mandatory but are best practices to follow:

1. Use lowercase letters for function name, words can be separated by underscore. Some also prefer to use camel case.
2. It is recommended to have a docstring as the first part of the function. The docstring must emphasize on what the function does, and not on how it does it.
3. Place the code after docstring.

The output of the code is as follows:

Output:

```
My first function prints HELLO WORLD!!
>>>
```

Now, type the following command on the Python shell:

```
help(hello_world_func())
```

This command will give details about the structure of this function.

```
>>> help(hello_world_func())
My first function prints HELLO WORLD!!
Help on NoneType object:

class NoneType(object)
|   Methods defined here:

|   __bool__(self, /)
|       self != 0
|
|   __repr__(self, /)
|       Return repr(self).

|   -----
|   Static methods defined here:
|
```

```
| __new__(*args, **kwargs) from builtins.type
|     Create and return a new object. See help(type) for accurate signature.
```

To view the docstring, print the following command:

```
>>>print(hello_world_func.__doc__)

This is my first function.
I am learning a lot and this looks like fun
This function just prints a message.
>>>
```

1.2.2 Defining functions that take parameter

The previous section defined a simple function. In this section, we will modify the same function to take a parameter, that is the name of a person, and print a personalized message (name along with message).

```
def hello_world_func(name):

    """
    This is my first function.
    I am learning a lot and this looks like fun
    This function just prints a message.
    """

    print('My first function prints HELLO WORLD!! I am {}'.format(name))
hello_world_func('Meenu')
```

Please notice the function definition, the parenthesis has a parameter name. So, when you call `hello_world_func()`, you have to pass an argument that can be passed on to the function.

The output of the preceding code is as follows:

```
My first function prints HELLO WORLD!! I am Meenu
>>>
```

Explanation:

To understand how the code works, it is important to first understand the difference between a parameter, and an argument.

```

Parameter

def hello_world_func(name):
    ...
    This is my first function.
    I am learning a lot and this looks like fun
    This function just prints a message.
    ...
    print('My first function prints HELLO WORLD!! I am {}'.format(name))

hello_world_func(Meenu)
Argument

```

Figure 1.1

Parameters are variables local to function, and are part of definition. Arguments are values that are passed on to the function. So, `name` is the parameter of the function `hello_world_func()` and “`Meenu`” is the argument that is passed on to the function `hello_world_func()` when it is called. When the call is made to the function, it maps the parameter “`name`” with the argument “`Meenu`” and uses that value for computation.

A function can have more than one parameter, and in that case order in which arguments are passed matters. For example, look at the following code:

```

def hello_world_func(name,age,profession):
    ...
    This is my first function.
    I am learning a lot and this looks like fun
    This function just prints a message.
    ...

    print('I am {} years old.'.format(age))
    print('My name is {}.'.format(name))
    print('I am {} by profession.'.format(profession))

hello_world_func(32,'Architect','Michael')

```

The function `hello_world_func()` takes three parameters in the following order `name`, `age`, and `profession` whereas the arguments passed to the function are in the order: `32, 'Architect', 'Michael'`.

As a result, the parameter `name` gets mapped to value `32`, `age` gets mapped to ‘`Architect`’ and `profession` gets mapped to ‘`Michael`’. So, when we execute the code following output is generated:

```

I am Architect years old.
My name is 32.
I am Michael by profession.

```

So, now let's call the function `hello_world_func()` and pass the arguments in the right order.

```
hello_world_func('Michael', 32, 'Architect')
```

Output:

```
I am 32 years old.  
My name is Michael.  
I am Architect by profession.
```

Python programmers often use parameters and arguments interchangeably as both are in a way quite similar but as a Python developer, you must understand the difference between the two. Parameters are declared in the function, and arguments are the values passed to a function when it is called.

1.3 Functional arguments

There are five types of functional arguments in Python.

- Positional arguments
- Default arguments
- Keyword arguments
- *args
- **kwargs

1.3.1 Positional arguments

All the examples that we have used till now are examples of positional arguments, that is, arguments are assigned to the parameters in the order in which they are passed or their position.

```
def sum_prod(num1, num2):  
    num_sum = num1 + num2  
    num_prod = num1 * num2  
    return num_sum, num_prod  
  
x = int(input('Enter the first number :'))  
y = int(input('Enter the second number :'))  
print(sum_prod(x,y))
```

Output:

```
Enter the first number :10
Enter the second number :20
(30, 200)
>>>
```

The positional argument looks at the position of the parameter where it will be assigned. So, the value of **x** gets mapped to **num1**, and value of **y** gets mapped to **num2** and these values are passed on to the function block code.

If you call a function with different number of parameters, an error will be generated.

1.3.2 Default arguments

You have the option of specifying default value of a parameter in the function definition. The positional argument for which a default value is defined, becomes optional and therefore known as **default argument**.

Code:

```
def sum_prod(num1,num2 =0):
    num_sum = num1 + num2
    num_prod = num1 * num2
    return num_sum,num_prod

print(sum_prod(2,5))
print(sum_prod(2))
```

Output:

```
(7, 10)
(2, 0)
>>>
```

The function shown above can be called with one or two arguments. If you omit the second argument, the function definition will pass on its default value, which is 0.

Let's take a look at another example.

In Python, a non-default argument **cannot** follow a default argument.

```
def sum_func(num1,num2 =0,num3):
    return num1+num2+num3
```

The preceding function will throw an error because `num3` which is a non-default argument follows `num2`, which is a default argument. So, if you type `sum_func(10, 20)`, the interpreter will not understand whether to assign 20 to `num2` or continue with the default value. The complexity will increase as the number of default arguments increase. In this scenario you will receive a Syntax Error: "non default argument follow default argument". The correct way of using default arguments is shown in the following code:

```
def sum_func(num1, num2 = 30, num3=40):
    return num1 + num2 + num3

print(sum_func(10))
print(sum_func(10, 20))
print(sum_func(10, 20, 30))
```

Output:

```
80
70
60
>>>
```

1.3.3 Keyword arguments

Keyword arguments allow you to ignore the order in which the parameters are entered in a function or even skip them when calling a function.

The function with keyword arguments are defined the same way as the function with positional arguments, but the difference is in the way they are called. Have a look at the following code:

```
def sum_func(num1, num2 = 30, num3=40):
    print("num1 = ", num1)
    print("num2 = ", num2)
    print("num3 = ", num3)
    return num1 + num2 + num3

print(sum_func(num3 = 10, num1 = 20))
```

Output:

```
num1 = 20
num2 = 30
num3 = 10
60
>>>
```

As you can see, the arguments are not passed in the desired order, but while passing the arguments, it is specified which argument belongs to which parameter. Since the default value of `num2` is zero, even if it is skipped, it does not matter. We wanted to use the default value of `num2` therefore only the value of `num1` and `num3` were specified. If that is not done, the output will be incorrect. As you can see in the following code, the value 20 is assigned to `num2` and default value of `num3` is taken as a result of which the result is completely different.

Code:

```
def sum_func(num1, num2 = 30, num3=40):
    print("num1 = ", num1)
    print("num2 = ", num2)
    print("num3 = ", num3)
    return num1 + num2 + num3

print(sum_func(10, 20))
```

Output:

```
num1 = 10
num2 = 20
num3 = 40
70
>>>
```

1.3.4 *args

`*args` is used when you don't have any idea about how many arguments you will use.

If you don't know how many parameter you require, then `* args` is the way to go. Suppose you have decided to go shopping, but you don't know how many items you are going to buy or how much you are going to spend. So, the expenditure is undecided. You have no idea about how many products you will buy so a function cannot be created with defined number of elements. When using `*args`, you are using a tuple with a potential of additional arguments. This tuple is initially empty, and no error is generated if no argument is provided.

Code:

```
def sum_func(a, *args):
    s = a + sum(args)
    print(s)

sum_func(10)
sum_func(10,20)
sum_func(10,20,30)
sum_func(10, 20, 30, 40)
```

Output:

```
10
30
60
100
>>>
```

1.3.5 **kwargs

****kwargs** stands for keyworded arguments(of variable length), and is used when you don't have any idea about how many keyword arguments you would be using. ****kwargs** builds a dictionary of key value pairs. These types of arguments are often used when working with different external modules and libraries. The double star ‘**’ in ****kwargs** allows any number of keyworded arguments to pass through. As the name suggests, in keyword argument a name is provided to the variable while passing it to the function similar to dictionary where keywords are associated with values.

```
def shopping(**kwargs):
    print(kwargs)
    if kwargs:
        print('you bought', kwargs['dress'])
        print('you bought', kwargs['food'])
        print('you bought', kwargs['Shampoo'])
shopping(dress = 'Frock', Shampoo = 'Dove', food = 'Pedigree Puppy')
```

Output:

```
{'dress': 'Frock', 'Shampoo': 'Dove', 'food': 'Pedigree Puppy'}
you bought Frock
you bought Pedigree Puppy
you bought Dove
```

It is important to note that since ****kwargs** is similar to dictionary, if you try to iterate over it then it may or may not print in the same order.

As far as the name is concerned, you can use any name instead of `args` or `kwargs`. These names are recommended, but are not mandatory. However, it is important to use `*` for positional arguments and `**` for keyword arguments. This is necessary.

1.4 The return statement

The `return` keyword is used at the end of the function when there is a need to send back the result of the function back to the caller. Software programming is not about printing results all the time. There are some calculations that must be performed behind the scene, hidden from the users. These values are further used in calculations to get the final output that the users desire. The value obtained from the `return` statement can be assigned to a variable and used further for calculations.

Look at the code given in the following box. The function `adding_numbers()` adds three numbers and returns the result which is assigned to variable `x`. The value of `x` is then displayed as output.

```
def adding_numbers(num1, num2, num3):
    print('Have to add three numbers')
    print('First Number = {}'.format(num1))
    print('Second Number = {}'.format(num2))
    print('Third Number = {}'.format(num3))
    return num1 + num2 + num3

x = adding_numbers(10, 20, 30)
print('The function returned a value of {}'.format(x))
```

Output:

```
Have to add three numbers
First Number = 10
Second Number = 20
Third Number = 30
The function returned a value of 60
```

So you can say that:

1. A `return` statement exits the function. It is the last statement of a function and any statement coming after that will not be executed.
2. When a function is not returning a value explicitly that means that indirectly or implicitly it is returning a value of None.

3. If a function has to return more than one value, then all the values will be returned as a tuple.

Example 1.1

Write a function that prompts the user to enter values for list. The function should return back the length of the list.

Answer:

```
def find_len(list1):
    return len(list1)

x = input('Enter the values separated by single space :')
x_list = x.split()
print(find_len(x_list))
```

Output:

```
Enter the values separated by single space :1 'Gmail' 'Google' 1.09 [2,3,45,9]
5
>>>
```

Explanation: On executing this code program, you will be prompted to enter values separated by single space. In this case, five values are passed:

1. 1
2. ‘Gmail’
3. ‘Google’
4. 1.09
5. [2,3,45,9]

So, the program counts the number of elements passed in as inputs, and displays the result of 5.

Example 1.2

The following code returns the value of sum and product of two numbers.

Code:

```
def sum_prod(num1,num2):
    num_sum = num1 + num2
    num_prod = num1 * num2
    return num_sum,num_prod
```

```
x = int(input('Enter the first number :'))
y = int(input('Enter the second number :'))
print(sum_prod(x,y))
```

Output:

```
Enter the first number :10
Enter the second number :20
(30, 200)
```

Example 1.3

Write code to find the HCF of two given numbers.

Answer:

HCF stands for *Highest Common Factor* or *Greatest Common Divisor* for two numbers. This means that it is the largest number within the range of 1 to smaller of the two given numbers that divides the two numbers perfectly giving the remainder as zero.

1. Define a function **hcf()** that takes two numbers as input.

```
def hcf(x,y) :
```

2. Find out which of the two numbers is greatest, the other one will be the smallest.

```
small_num = 0
if x > y:
    small_num = y
else:
    small_num = x
```

Set a for loop for the range 1 to **small_num+1**. (We take the upper limit as **small_num+1** because the for loop operates for one number less than the upper limit of the range). In this for loop, divide both the numbers with each number in the range, and if any number divides both, perfectly assign that value to hcf as shown in the following code:

```
for i in range(1,small_num+1):
    if (x % i == 0) and (y % i == 0):
        hcf = i
```

Suppose, the two numbers are 6 and 24, first both numbers are divisible by 2. So, $hcf = 2$, then both numbers will be divisible by 3. So, the value

of 3 will be assigned to 3. Then, the loop will encounter 6, which will again divide both the numbers equally. So, 6 will be assigned to hcf. Since the upper limit of the range has reached, the function will finally have hcf value of 6.

3. Return the value of hcf:

```
return hcf
```

Code:

```
def hcf(x,y):  
    small_num = 0  
    if x > y:  
        small_num = y  
    else:  
        small_num = x  
    for i in range(1,small_num+1):  
        if (x % i == 0) and (y % i == 0):  
            hcf = i  
    return hcf  
print(hcf(6,24))
```

OUTPUT:

```
6
```

1.5 Scopes and namespace

Namespace is a container that has all the names (of variables/functions/classes) that you define. You can define same names in different namespaces. A name or a variable exists in a specific area of the code which defines its scope. The information regarding binding between the variables/objects is stored in the namespace. There are three types of namespaces or scopes.

1. **Built-in Namespace:** These are in-built functions that are available across all files or modules.
2. **Global Namespace:** The global namespace has all the variables, functions, and classes that are available in a single file.
3. **Local Namespace:** The local namespace are variables defined within a function.

The scopes are nested, which means that the local namespace is nested within a global namespace which is nested within built-in namespace. Each scope has its namespace.

1.5.1.Built-in namespace

Built-in namespace are available across all the files, and module in Python. All functions that you see below `print()`, `tuple()`, `type()` are all built-in function, and belong to this namespace and are available across all files and modules in Python.

```
>>> list1 = [1,2,3,4,5,6]
>>> tup1 = tuple(list1)
>>> type(tup1)
<class 'tuple'>

>>> print("Hi")
Hi
>>>
```

1.5.2 Global namespace

Look at the following code:

```
x = 20
x += y
print(x)
```

When we execute this code, it generates a Name Error:

Output:

```
Traceback (most recent call last):
  File "F:\2020 - BPB\input.py", line 2, in <module>
    x += y
NameError: name 'y' is not defined
>>>
```

This is because Python looks for the name `y` in the global namespace, and fails to find it. It then looks for it in the built in namespace, and does not find it again. Hence, an error is generated. The following code works fine and does not produce any error because the statement `y = 5` created a global namespace:

```
x = 20
y = 5
```

```
x += y  
print(x)
```

Output:

```
25  
=>
```

1.5.3 Local namespace

Now, let's look at another example.

Code:

```
x = 20  
def print_x():  
    x = 10  
    print('Local variable x is equal to ',x)  
    print('Global variable x is equal to ',x)  
print_x()
```

Output:

```
Global variable x is equal to 20  
Local variable x is equal to 10  
=>
```

When a call is made to the function, the Python interpreter tries to locate the local variable called **x**. If that is not available, it will look for **x** at global namespace.

Code:

```
x = 20  
def print_x():  
    print('Local variable x is equal to ',x)  
  
    print('Global variable x is equal to ',x)  
print_x()
```

Output:

```
Global variable x is equal to 20  
Local variable x is equal to 20  
=>
```

Local variables, that is, the variables within a function are created when a call is made to that function. Whenever a call is made to a function, a new scope is created, and variables are assigned to that scope. Once the function has been executed, its scope is also gone. In the first example, when the `function print_x()` was called, it was able to find a local variable `x = 10` within the local namespace, and used it up. This value of `x` existed within the function, and vanishes with the function after its execution is over.

Sometimes, when we want to use the global variable inside our function namespace, we should use `global` keyword for that variable to make it clear that we want to use the global variable only. Look at the following code:

```
x = 20
def print_x():
    global x
    x = 10
    print('Local variable x is equal to ',x)

print('Global variable x is equal to ',x)
print_x()
print('Global variable x is equal to ',x)
```

The moment `global` keyword is used, the function `print_x()` comes to know that the global variable `x` will be used. In the next statement `x = 10`, the value 10 is assigned to `x` which is a global variable. Therefore, you will see in the output the value of global variable is 20 before the function is called, and 10 after the function is called. When you are using a `global` keyword with a variable name within a function, Python will not allow you to create another variable with the same name within the same function.

```

x = 20
def print_x():
    x = 9
    global x
    x = 10

    print('Local variable x is equal to ',x )

print('Global variable x is equal to ',x )
print_x()
print('Global variable x is equal to ',x )

```

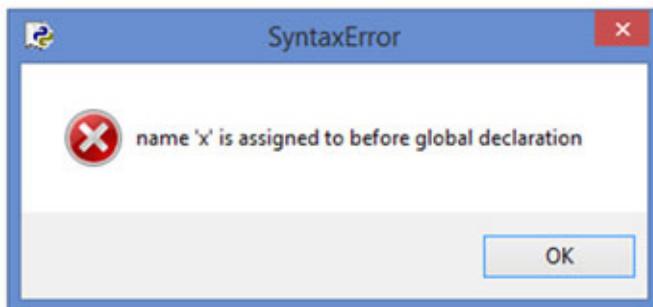


Figure 1.2: python does not allow you to create another variable within a function with the same name

1.6 Lambda functions

Lambda functions are Python's anonymous functions, that is, they are defined without a name. These functions are defined using `lambda` keyword instead of `def`. The syntax for lambda function is as follows:

```
lambda arguments: expression
```

The most interesting feature about lambda functions is that it can have any number of arguments but only one expression. The expression is evaluated, and the value is returned. Ideally, lambda functions are used if there is a requirement of function objects.

Note: Function Objects: Functions in Python can be passed as arguments to other functions. Functions can be also be assigned to variables or stored in a data structure as elements.

For example:

```
your_age = lambda yr_of_birth: 2021 - yr_of_birth
print(your_age(1956))
```

Output:

Three functions `map()`, `filter()`, and `reduce()` were created to facilitate functional approach in Python programming. In Python 3 `reduce()` has been discontinued. These functions can be replaced by List Comprehensions (Covered in [Chapter 5, List and Arrays, Basic Core Python Programming](#)) or loops. Example 1.4 demonstrate how to use lambda function with filter().

Example1.4: Use lambda function with `filter()` to print odd numbers from a given list.

Answer:

```
number = [1,2,3,4,5,6,13,7,8,9,0]
odd_number = list(filter(lambda x : (x%2!=0), number))
print(odd_number)
```

Similarly, `map()` function applies the same function to each element of a sequence and returns a modified list.

Suppose we need to square every number in list [1,2,3,4,5], the usual way of doing this would be:

```
list1 = [1,2,3,4,5]
for element in list1:
    print(element**2)
```

We can use `map()` along with lambda to produce the same result:

```
print(list(map(lambda x: x**2,list1)))
```

Example 1.5

Rewrite the following code using lambda function:

```
def comparisonFunc(a,b):
    if a>b:
        return a
    else:
        return b
print(comparisonFunc(10,3))
.
```

Answer:

```
comparisonFunc = lambda a,b: a if a > b else b
```

```
print(comparisonFunc(10,3))
```

Example 1.6

Rewrite the following code using lambda functions:

```
def addFunc(a,b):  
    return a+b  
print(addFunc(5,2))
```

Answer:

```
addFunc = lambda a,b: a+b  
print(addFunc(5,2))
```

Example 1.7

Given a list of countries as:

```
countries =  
['India', 'Mauritius', 'France', 'Turkey', 'Kenya', 'Hungary']
```

Use lambda function to print the length of each string in the list `countries`.

Answer:

```
countries = ['India', 'Mauritius', 'France', 'Turkey', 'Kenya', 'Hungary']  
print(list(map(lambda x: len(x),countries)))
```

Example 1.8

Explain how to use `sort()` with lambda. Also, explain the difference between `sort()` and `sorted()` functions.

Answer:

The syntax for sort is as follows:

```
list.sort(key = None, reverse= False)
```

The `sort` method uses the process of comparing the items to sort the elements of a list. The lambda functions allow key to become more versatile.

Suppose we have the list of country names:

```
countries =
['India', 'Mauritius', 'France', 'Turkey', 'Kenya', 'Hungary']
```

Now, if we want to sort these names alphabetically, we can use the procedure given as follows:

```
countries = ['India', 'Mauritius', 'France', 'Turkey', 'Kenya', 'Hungary']
countries.sort()
print(countries)
```

Or we can use the **sort()** function given as follows:

```
countries.sort(key = lambda x:x[0])
print(countries)
```

So basically, the lambda functions takes each **element(x)** and sorts with respect to first element of string.

Suppose we have a list of names, and we want to sort the names on the basis of surnames:

We can write:

```
>>> names = ['Mahatma Gandhi', 'Jawaharlal Nehru', 'Subhash Chandra bose', 'Rani Laxmi
Bai', 'Chandra Shekhar Azaad', 'Sarojini Naidu']

>>> names.sort(key = lambda x:x.split()[-1])

>>> print(names)

['Chandra Shekhar Azaad', 'Rani Laxmi Bai', 'Mahatma Gandhi', 'Sarojini Naidu',
'Jawaharlal Nehru', 'Subhash Chandra bose']
```

Here, **lambda x:x.split()[-1]**, **x.spilt()** breaks each element into individual words. The names (having two parts) is split into a list of two words, and the names having three parts are split into a list of three words. -1 is the index of the last element in the list. So, **lambda x:x.split()[-1]** is working on last word of each element in the list names. (Notice, the name ‘Subhash Chandra bose’ is placed at the end because the surname does not begin with capital ‘b’. Let’s change the surname to capital B and see the output.)

```
>>> names = ['Mahatma Gandhi', 'Jawaharlal Nehru', 'Subhash Chandra Bose', 'Rani Laxmi
Bai', 'Chandra Shekhar Azaad', 'Sarojini Naidu']

>>> names.sort(key = lambda x:x.split()[-1])

>>> print(names)
```

```
[ 'Chandra Shekhar Azaad', 'Rani Laxmi Bai', 'Subhash Chandra Bose', 'Mahatma Gandhi',
'Sarojini Naidu', 'Jawaharlal Nehru' ]
```

We can also get the same result using the **sorted()** function.

```
>>> names = ['Mahatma Gandhi','Jawaharlal Nehru','Subhash Chandra Bose','Rani Laxmi
Bai','Chandra Shekhar Azaad','Sarojini Naidu']

>>> print(list(sorted(names, key = lambda x:x.split() [-1])))

['Chandra Shekhar Azaad', 'Rani Laxmi Bai', 'Subhash Chandra Bose', 'Mahatma Gandhi',
'Sarojini Naidu', 'Jawaharlal Nehru']
```

The difference between the **sort()** and the **sorted()** function is that the **sort()** function modifies the list whereas the **sorted()** function provides a new list having the sorted values.

Example 1.9

Given $\text{list1} = [(1, 2), (4, 1), (9, 10), (13, -3)]$. Write, the code to get an output of:

1. $[(1, 2), (4, 1), (9, 10), (13, -3)]$
2. $[(13, -3), (4, 1), (1, 2), (9, 10)]$
3. $[(1, 2), (4, 1), (9, 10), (13, -3)]$
4. $[(9, 10), (1, 2), (4, 1), (13, -3)]$

Answer:

In the solution given below, you should know that you are sorting a list consisting of tuples. **x[0]** means you are sorting on the basis of element at position 0 and **x[1]** means sorting on the basis of element present on position 1 of tuple. By default, the sorting is done in the ascending order. To sort in the descending order, set reverse to True.

1.

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
list1.sort(key = lambda x:x[0])
print(list1)
```

Or

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
print(list(sorted(list1, key = lambda x:x[0])))
```

Output:

```
[ (1, 2), (4, 1), (9, 10), (13, -3) ]
```

2.

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
list1.sort(key = lambda x:x[1])
print(list1)
```

Or

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
print(list(sorted(list1, key = lambda x:x[1])))
```

Output:

```
[ (13, -3), (4, 1), (1, 2), (9, 10) ]
```

3.

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
list1.sort(key = lambda x:x[0], reverse = True)
print(list1)
```

Or

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
print(list(sorted(list1, key = lambda x:x[0], reverse = True)))
```

Output:

```
[ (13, -3), (9, 10), (4, 1), (1, 2) ]
```

4.

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
list1.sort(key = lambda x:x[1], reverse = True)
print(list1)
```

Or

```
list1 = [(1, 2), (4, 1), (9, 10), (13, -3)]
print(list(sorted(list1, key = lambda x:x[1], reverse = True)))
```

Output:

```
[ (9, 10), (1, 2), (4, 1), (13, -3) ]
```

Example 1.10

Function `func` is defined as follows:

```
def func(x,y):  
    return (x+y)/2
```

Rewrite the function using lambda expression.

Answer:

```
func1 = lambda x,y: (x+y)/2
```

Example 1.11

Alex's score is given as follows:

```
marks = [ {'Subject':'Maths','Score':90},{'Subject':'Science','Score': 100},  
{'Subject':'Geography','Score':83} ]
```

Sort the dictionary `marks` by descending order of score using lambda expression.

Answer:

```
marks = [ {'Subject':'Maths','Score':90},{'Subject':'Science','Score': 100},  
{'Subject':'Geography','Score':83} ]  
print(list(sorted(marks, key = lambda x:x['Score']), reverse= True)))
```

Output:

```
[{'Subject': 'Science', 'Score': 100}, {'Subject': 'Maths', 'Score': 90}, {'Subject':  
'Geography', 'Score': 83}]
```

Example 1.12

Given:

```
list1 = ['AR-MO-UR', 'O-F', 'G-O-D']
```

use `map()` function along with lambda expression to produce the following output:

```
[ 'ARMOUR' , 'OF' , 'GOD' ]
```

Answer:

```
>>> list1 = ['AR-MO-UR','O-F','G-O-D']
>>>print(list(map(lambda x: ''.join(x.split('-')),list1)))
['ARMOUR', 'OF', 'GOD']
```

Example 1.13

`list1 = [10,30,50,70], list2 = [20,40,60,80]`. Write a code that displays as list that has sum of `list1[i]+list2[i]`. where ‘i’ is the index of the element.

Answer:

```
>>> list1 = [10,30,50,70]
>>> list2 = [20,40,60,80]
>>>print(list(map(lambda x,y: x+y,list1,list2)))
[30, 70, 110, 150]
```

1.7 Recursion

Programming is all about solving complex problems, and a good programmer knows that there can be more than one way of solving a problem. He should also be aware of different techniques that can be applied to get the solution problem. In this chapter, you will learn about recursion which is a fundamental technique of computer programming used to resolve a particular type of complex problems.

Recursion in programming can be applied to solve a problem whose solution depends on the solutions to smaller instances of the same problem.

We can therefore say that Recursion refers to a function, which can calculate the right answer by first solving a smaller version of its own self and then using that result along with some more computation to get the final answer. The definition of recursion will become clearer as you go through the examples given in this chapter.

Let’s have a look at our first example - finding the factorial of a number.

1.7.1 Finding factorial of a number using recursion

A factorial of a number is the product of that number and all positive integers below it. So, factorial of 5 or $5!$ can also be represented as follows:

$$5! = 5 * 4 * 3 * 2 * 1 \quad \dots\dots(1)$$

Now, look at the preceding statement carefully. Since a factorial of a number is product of that number and all positive numbers below it we can say that $4 * 3 * 2 * 1 = 4!$. Hence, statement (1) can be rewritten as:

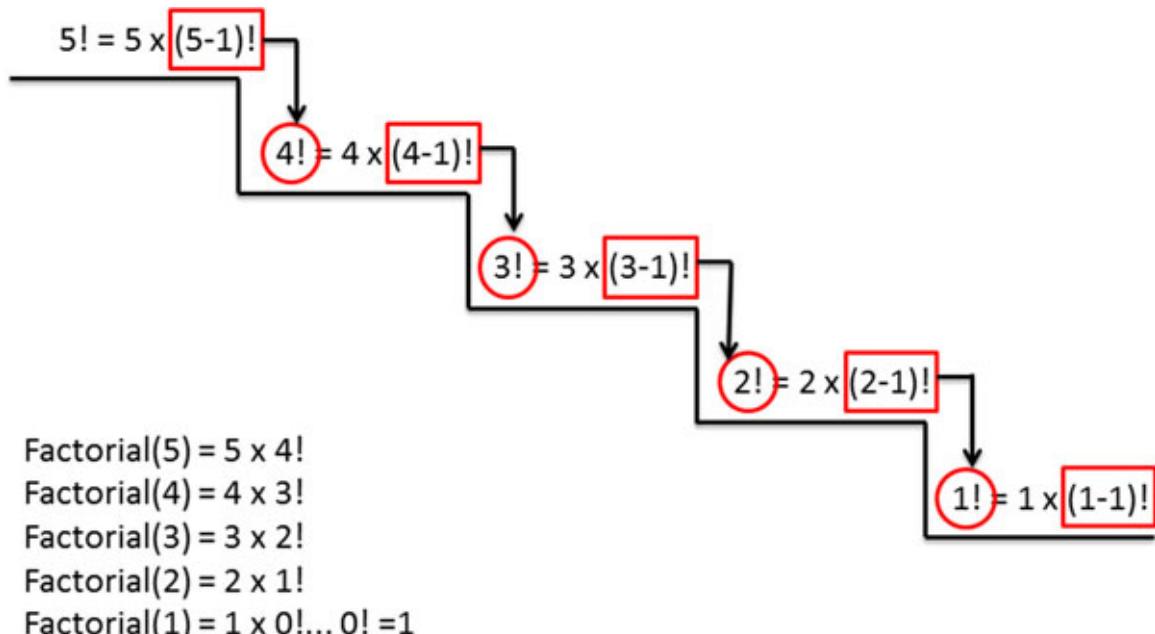
$$5! = 5 * 4 * 3 * 2 * 1 = 5 * 4! \quad \dots\dots(2)$$

Following the definition of factorial, we can now write (2) as:

$$5! = 5 * 4 * 3 * 2 * 1 = 5 * 4 * 3! \quad \dots\dots(3)$$

Similarly,

$$5! = 5 * 4 * 3 * 2 * 1 = 5 * 4 * 3 * 2! \quad \dots\dots(4)$$



$$\text{Factorial}(n) = n \times (\text{Factorial}(n-1))$$

Figure 1.3: Calculating factorial of a number

So, if we define a method `find_factorial(n)` to find factorial of number `n`, it would mean that `find_factorial(n)` is same as `n * find_factorial(n-1)` and `find_factorial(n-1)` is same as `(n-1) * find_factorial(n-2)`, and so on.

Therefore, as we start writing the code for recursion, the following may seem like the right way to start. However, the following code is not complete yet.

```
def find_factorial(n):
    return n*find_factorial(n-1)
```

At this point, it is important to understand base case or terminal case. Every problem in recursion has a base case. It is one or more special values for which a function can be evaluated without recursion or it is that part of the recursion problem, which cannot be defined in smaller instances of itself. *Without a base case a recursive function will never stop executing.* A recursive function makes calls to itself, which step by step takes the function to the base case, and the function then stops executing.

So, the preceding code is not complete because it has no base case.

As per the definition of factorial, we say that factorial of a number is the product of itself with all the positive numbers below it. This means that the last number to be multiplied is always 1. Thus, the function should stop executing when a call is made to it with value `n= 1`. This will be the base case for our `find_factorial()` function.

```
def find_factorial(n):
    if(n==1):
        return 1
    return n*find_factorial(n-1)

print (find_factorial (5))
```

1.7.2 Algorithm for finding factorial using recursion

Function `find_factorial(n):`

Step 1: Read the number `n` provided for finding factorial.

Step 2: Check whether the value provided is equal to 1. If true, return 1.

Step 3: Else return value of `n*find_factorial(n-1)`.

Before going ahead with recursion, let's learn few more things about recursion.

1.7.3 Types of recursion

Recursion can be classified as direct recursion or indirect recursion.

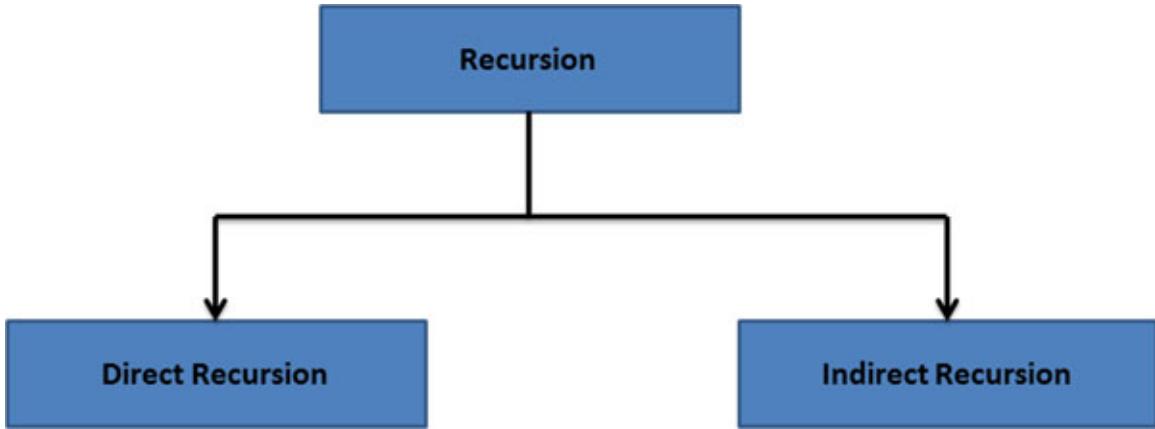


Figure 1.4: Types of Recursion

If a function makes a call to itself, then it is an example of **direct recursion**. For example: in the factorial function, the `find_factorial()` function made a call to itself. Therefore, we can call it an example of direct recursion. However, sometimes there are scenarios where a function `f()` may call another function `f1()`, which in return makes a call back to function `f()`. This is known as **Indirect Recursion**. As an example, have a look at the following code:

```

def happy_new_year(n=1):
    if(n<=0):
        how_many_times()

    for i in range(n):
        print("Happy New Year")

def how_many_times():
    val = input("How many times should I print?:")
    if val=='':
        happy_new_year()
    else:
        happy_new_year(int(val))

how_many_times()

```

In the preceding example, there is a function called `happy_new_year()` that prints the “*Happy New Year*” message as many number of times as you want. If no value is provided, then it takes the default value as 1, and prints the message once. If the value is invalid, that is, 0 or below, it calls a function `how_many_times()` which prompts the user again to provide another value and makes a call back to `happy_new_year()` with the new value. Have a look at the following figure:

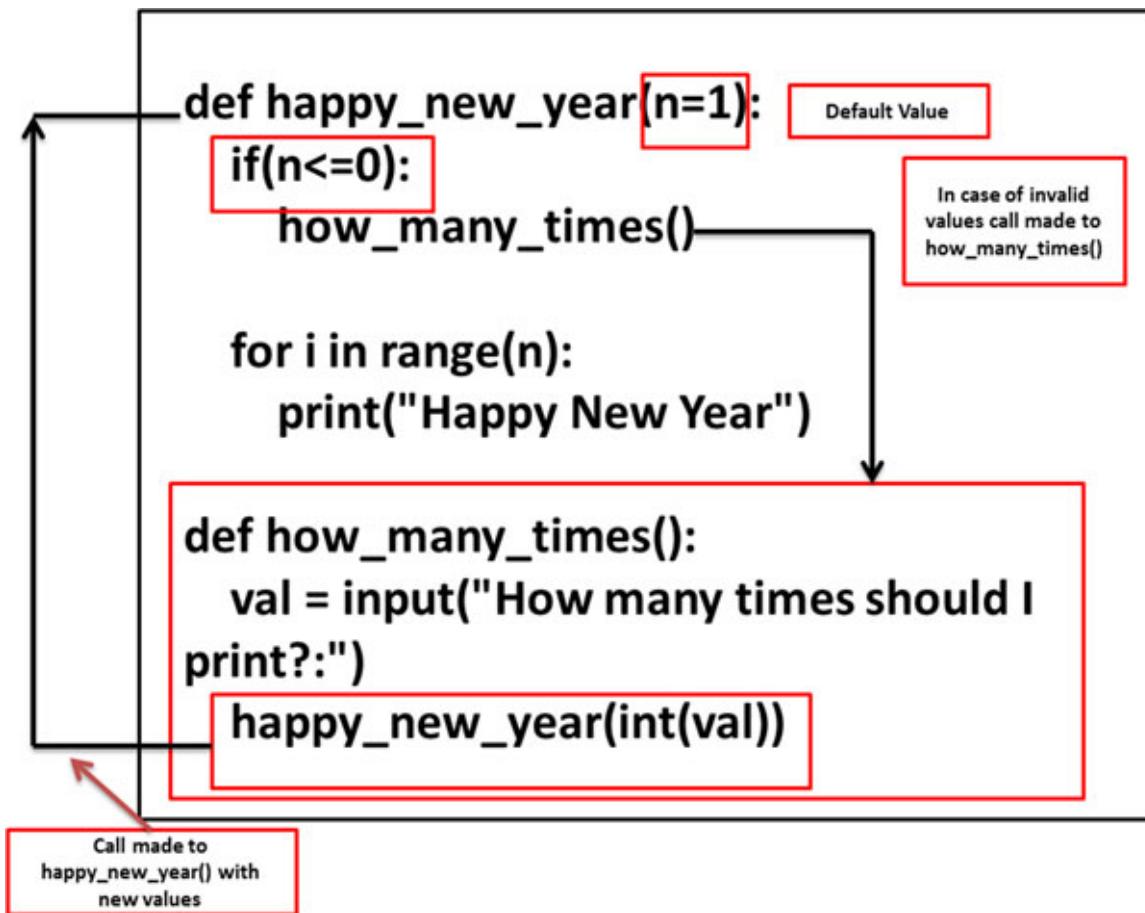


Figure 1.5: Indirect Recursion

Let's have a look at some interesting output for this code:

Case 1: Call is made to `happy_new_year()` with no input value.

Output:

Happy New Year

Explanation: Since no value was provided the message got printed only once as that is the default value.

Case 2: Call is made to `happy_new_year()` for `n=0`.

Output: Since `n = 0`, `how_many_times()` is called again till a value of `n` greater than 0 is provided.

```

How many times should I print?:0
How many times should I print?:0
How many times should I print?:7
Happy New Year
Happy New Year

```

```
Happy New Year
>>>
```

Explanation: The user provided $n = 0$ twice as a result of which the function `happy_new_year()` called `how_many_times()` back each time till the user decided to provide a value greater than 0 which in this case is 7. So, for $n \leq 0$, the function prompts the user to provide another value, and the function `happy_new_year()` is called again with the new input value.

Case 2: Call made with invalid values again and again.

Output:

```
happy_new_year(-2)
How many times should I print?:-3
How many times should I print?:-8
How many times should I print?:0
How many times should I print?:2
Happy New Year
Happy New Year
```

Explanation: An invalid value is passed on to `happy_new_year()` function, which then calls the `how_many_times()` function which prompts users to provide a valid number. However, the user continues to provide invalid values so both the functions keep on calling each other till a valid value is provided.

1.7.4 Advantages and disadvantages of recursion

Let's have a look at advantages and disadvantages of recursion:

Advantages of recursion

- Requires fewer lines of code. The code looks clean.
- Allows you to break a complex task into simpler tasks.

Disadvantages of recursion

- Forming the logic for recursion can sometimes be difficult.
- Debugging a recursive function can be difficult.

- Recursive functions consume more memory and time.

Example 1.14 Recursion for Fibonacci Numbers

Fibonacci series have integers arranged in the following sequence:

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots$

Before we start writing the code, it is important to understand how the sequence is created. Look at the following illustration:

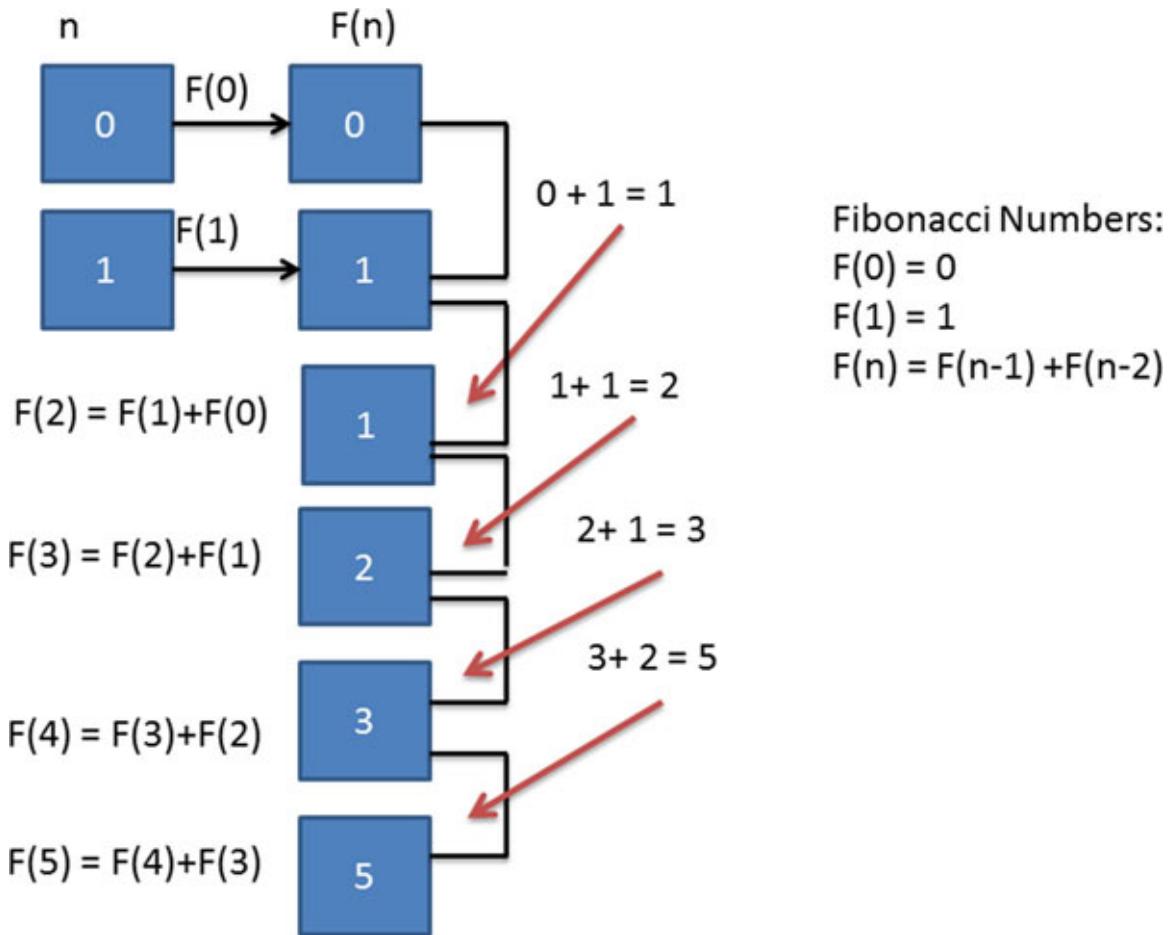


Figure 1.6: Fibonacci numbers

From the logic shown in the preceding figure, you can easily calculate that:

$$F(6) = F(5) + F(4) = 5 + 3 = 8$$

$$F(7) = F(6) + F(5) = 8 + 5 = 13$$

$$\text{Therefore, } F(n) = F(n-1) + F(n-2)$$

So, we need to follow the following steps to create a sequence of Fibonacci numbers:

Function fib(n):

Step 1: Read the number n.

Step 2: If n=0, return 0

Step 3: If n=1, return 1

Step 4: Else return fib(n-1)+fib(n-2)

The code for Fibonacci numbers would be as follows:

```
def fib(n):
    if(n==0):
        return 0
    elif(n==1):
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

1.8 Memoization

Basically in memoization we maintain a look up table where solutions are stored so that we don't have to solve the same sub problem again and again. Instead we solve it once, and store the values so that they can be reused.

We know that Fibonacci sequence is:

$$\begin{aligned} F(n) &= F(n-1)+F(n-2) \text{ if } n>1 \\ &= n \text{ if } n=0,1 \end{aligned}$$

So,

```
F(n):
    if n<1:
        return n
    else :
        return F(n-1)+F(n-2)
```

Here, we are making two recursive calls, and adding them up, and the value is returned.

Look at the following diagram:

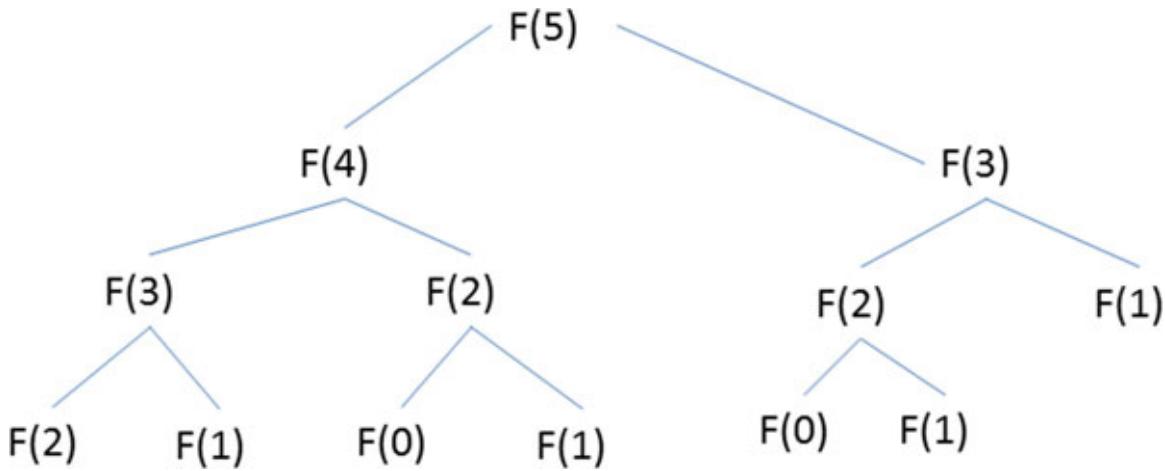


Figure 1.7: recursive calls in Fibonacci numbers

Observe that just to find $\text{fibonacci}(5)$, $\text{fibonacci}(2)$ is computed three times and $\text{fibonacci}(3)$ is computed two times. So, as n increases, fibonacci function's($f(n)$) performance goes down. The consumption of time and space would increase exponentially with increase in n . We can save time, by following a simple approach, that is to save a value when it is computed for the first time. So, we can save the values of $F(1)$, $F(2)$, $F(3)$ and $F(4)$ when they are computed for the first time, same way with $F(3)$, $F(4)$...so on. So, we can say that:

```

F(n) :
if n=<1:
    return n
elif F(n) exist :
    return F(n-1)
else:
    F(n) = F(n-1)+F(n-2)
    Save F(n)
    Return F(n)

```

In the following code:

1. The function `fibonacci()` takes a number and creates a list, `fib_num` of size `num+1`. This is because the Fibonacci series start from 0.
2. It calls the function `fib_calculate()`, which takes the number `num` and list `fib_num` as a parameter.
3. We have saved -1 at all index in the list:
 - a. If `fib_num[num]` is >0 , that means Fibonacci for this number already exists, and we need not compute it again, and the number

can be returned.

- b. If $num \leq 1$, then return num.
- c. Else if $num \geq 2$, calculate `fib_calculate(num - 1, fib_num) + fib_calculate(num - 2, fib_num)`. The value calculated must be stored in list `fib_num` at index num so that there is no need to calculate it again.

Code:

```
def fibonacci(num):  
    fib_num = [-1] * (num + 1)  
    return fib_calculate(num, fib_num)  
def fib_calculate(num, fib_num):  
    if fib_num[num] >= 0:  
        return fib_num[num]  
    if (num <= 1):  
        fnum = num  
        return fnum  
    else:  
        fnum = fib_calculate(num - 1, fib_num) + fib_calculate(num - 2, fib_num)  
        fib_num[num] = fnum  
    return fnum  
  
num = int(input('Enter the number: '))  
print("Answer = ", fibonacci(num))
```

Execution:

```
num = int(input('Enter the number: '))  
print("Answer = ", fibonacci(num))
```

Output:

```
Enter the number: 15  
Answer = 610  
>>>
```

Points to remember

- Functions are organized blocks of reusable code.
- Functions simplify coding, and have many advantages.
- Every function is dedicated to perform single-related action.
- In Python programming functions can be classified as:

1. Built-in functions that are provided by Python.
 2. User-defined functions are developed by programmers to perform a particular task. These functions are not provided by Python.
- A function block begins with the keyword `def` and is followed by the function name and `parenthesis()` after which a colon (`:`) is placed to mark the beginning of the function code block.
 - The code block must be properly indented as per PEP-8.
 - You can call the function anytime by its name followed by parenthesis.
 - The function does not execute as long it is called explicitly.
 - You can use docstring for documenting important features of a function
 - Functions may or may not take parameters it depends on the function definition.
 - Parameters are variables local to function and are part of definition.
 - There are five types of functional arguments in Python.
 1. Positional arguments
 2. Default arguments
 3. Keyword arguments
 4. `*args`
 5. `**kwargs`
 - Return statement is used for exiting a function by passing back an expression to caller.
 - It is possible for a program to return nothing(`None`).
 - Namespace is a space that has all the names(of variables/functions/classes) that you define in a program.
 - Namespace ensure that the names are unique so that there is no conflict.
 - In Python, namespace is implemented in form of dictionary.
 - The namespace maintains a name to object mapping where the name acts as a key and objects are values.
 - A name or a variable exists in a specific area of the code, which defines its scope. A variable can reach anywhere in the area that is defined by its scope. Three types of namespace or scopes are as follows:

1. **Built-in Namespace:** These are in-built functions that are available across all files or modules.
 2. **Global Namespace:** The global namespace has all the variables, functions, and classes that are available inside a single file.
 3. **Local Namespace:** The local namespace are variables defined within a function.
- Recursion is defining anything in terms of itself. Recursion is when a function makes a call to itself again and again till a base condition is met that stops the function from executing further.
 - When a function makes a call to itself, it is called **recursion**. The same sets of instructions are repeated again and again for new values, and it is important to decide when the recursive call must end.
 - Recursion can be of two types: (1) direct and (2) indirect
 - Recursive functions make the code look neat.
 - Recursion can consume more memory and time and can be hard to debug.
 - A recursive function consists of two things:
 1. Base case or terminating case
 2. Recursive case
 - Every function that uses recursion can be written using iteration and vice versa.

Conclusion

In this chapter, you learnt how to create your own functions. You now have all the knowledge that is required to start working with object-oriented programming. You will learn how to create classes in Python in the next chapter.

Short questions and answers

1. A named block of code that can be reused and called by its name is called a _____.

Answer: Function

2. Why are functions required?

Answer: Many times in a program, a certain set of instructions may be called more than once. Instead of writing the same piece of code again and again whenever it is required, it is better to define a function, and place the code in it. This function can be called whenever there is a need. This saves time and effort, and the program can be developed easily. Functions help in organizing coding work, and testing of code also becomes easy.

3. What is the difference between passing immutable and mutable objects as argument to a function?

Answer: If immutable arguments such as strings, integers, or tuples are passed to a function, the object reference is passed but the value of these parameters cannot be changed. It acts like pass by value call. Mutable objects too are passed by object reference, but their values can be changed.

4. In-built functions can be used easily by importing the modules in which they are defined in your code.

- a. True
- b. False

Answer: False

5. Functions provide better modularity.

- a. True
- b. False

Answer: True

Coding questions and answers

1. Write code to find all possible palindromic partitions in a string.

Answer:

The code to find all possible palindromic partitions in a string will involve the following steps:

1. Create a list of all possible substrings.
2. Substrings are created by slicing the strings as all possible levels using for loop.
3. Every substring is then checked to see whether it is a palindrome.

4. The substring is converted to a list of single characters.
5. In the reverse order, the characters from the list are added to a string.
6. If the resultant string matches the original string, then it is a palindrome.

Code:

```

def create_substrings(x):
    substrings = []
    for i in range(len(x)):
        for j in range(i, len(x)+1):
            if x[i:j] != '':
                substrings.append(x[i:j])
    for i in substrings:
        check_palin(i)
def check_palin(x):
    palin_str = ''
    palin_list = list(x)
    y = len(x)-1
    while y>=0:
        palin_str = palin_str + palin_list[y]
        y = y-1
    if(palin_str == x):
        print("String ", x, " is a palindrome")
x = "malayalam"
create_substrings(x)

```

Execution:

```

x = "malayalam"
create_substrings(x)

```

Output:

```

String m is a palindrome
String malayalam is a palindrome
String a is a palindrome
String ala is a palindrome
String alayala is a palindrome
String l is a palindrome
String layal is a palindrome
String a is a palindrome
String aya is a palindrome
String y is a palindrome
String a is a palindrome
String ala is a palindrome
String l is a palindrome

```

```
String a is a palindrome  
String m is a palindrome
```

2. What will be the output of the following function?

```
def happyBirthday():  
    print("Happy Birthday")  
a = happyBirthday()  
print(a)
```

Answer:

```
Happy Birthday  
None
```

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

```
def outerWishes():  
    global wishes  
    wishes = "Happy New Year"  
def innerWishes():  
    global wishes  
    wishes = "Have a great year ahead"  
    print('wishes =', wishes)  
wishes = "Happiness and Prosperity Always"  
outerWishes()  
print('wishes =', wishes)
```

Answer: The output will be as follows:

```
wishes = Happy New Year
```

4. What would be the output for the following code?

```
total = 0  
def add(a,b):  
    total = a+b  
    print("inside total = ",total)  
add(6,7)  
print("outside total = ",total)
```

Answer:

```
inside total = 13  
outside total = 0
```

5. Write a function that calculates HCF using Euclidean Algorithm, and returns that value.

Answer:

The following figure shows two ways to find HCF:

On the left hand side, you can see the traditional way of finding the HCF. On the right hand side is the implementation of **Euclidean** algorithm to find HCF.

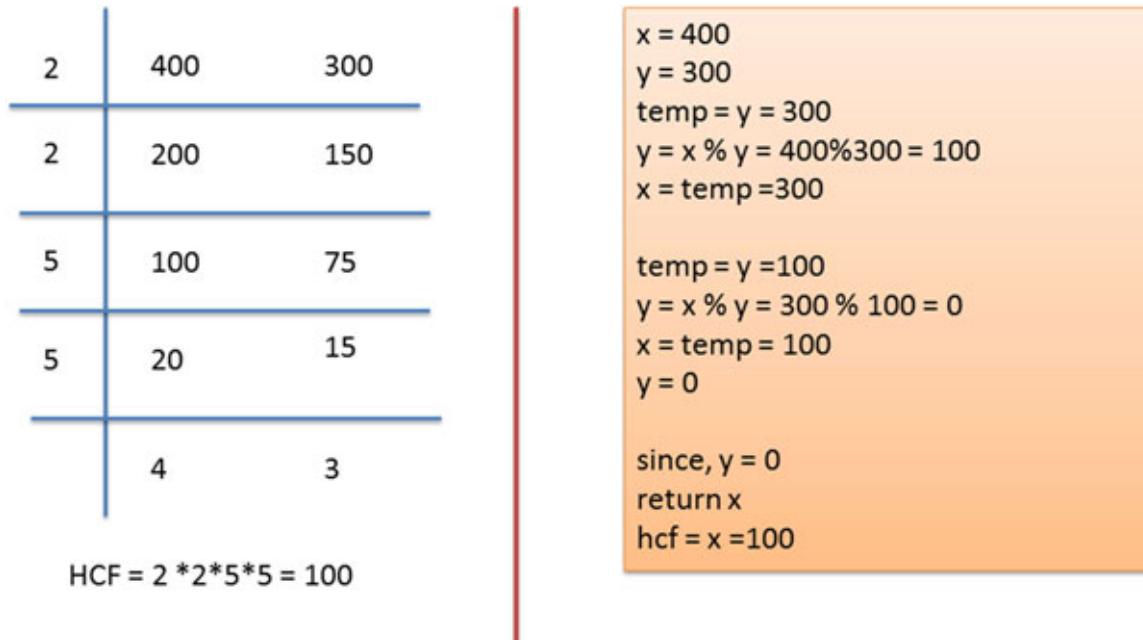


Figure 1.8: Finding HCF using normal method and Euclidean Algorithm

Code:

```
def hcf(x,y):  
    small_num = 0  
    greater_num = 0  
    temp = 0  
    if x > y:  
        small_num = y  
        greater_num = x  
    else:  
        small_num = x  
        greater_num = y  
    while small_num> 0:  
        temp = small_num  
        small_num = greater_num % small_num  
        greater_num = temp  
    return temp
```

Execution:

```
print("HCF of 6 and 24 = ",hcf(6,24))
print("HCF of 400 and 300 = ",hcf(400,300))
```

Output:

```
HCF of 6 and 24 = 6
HCF of 400 and 300 = 100
```

6. A function that does not return explicitly returns implicitly none. Prove.

Answer:

```
def find_len(list1):
    print("List Recieved")
    x = input('Enter the values separated by single space :')
    x_list = x.split()
    print(find_len(x_list))
```

Output:

```
Enter the values separated by single space :1 'Gmail' 'Google' 1.09 [2,3,45,9]
5
List Recieved
None
>>>
```

Explanation: The function `find_len()` does not return the length of the list; hence, the print statement displays none. Hence proved that when a function does not return explicitly, it implicitly returns none.

7. Determine the output of the following:

```
x = 20
def print_x():
    global x
    x = 10
    return x
print(print_x())
```

Answer: 10

8. Determine the output of the following:

```
x = 20
def function_1():
    global x
    x = 10
    return x
def function_2():
```

```
    return x
print(function_2())
```

Answer:

20

9. Write a function to find factorial of a number using **for** loop.

Answer:

The code for finding a factorial using **for** loop will be as follows:

Code:

```
def factorial(number):
    j = 1
    if number==0|number==1:
        print(j)
    else:
        for i in range (1, number+1):
            print(j, " * ",i," = ",j*i)
            j = j*i
    print(j)
factorial(5)
```

Execution:

```
factorial(5)
```

Output:

```
1 * 1 = 1
1 * 2 = 2
2 * 3 = 6
6 * 4 = 24
24 * 5 = 120
120
```

10. Write a function for Fibonacci series using a for loop:

Answer:

Fibonacci series: 0,1,1,2,3,5,8....

We take three variables:

i,j, and k:

- o If i = 0, j = 0, k = 0

- o If $i=1, j=1, k=0$

- o If $i>1$:

$\text{temp} = j$

$j = j + k$

$k = \text{temp}$

The calculations are as shown as follows:

i	k	j
0	0	0
1	0	1
2	0	$\text{temp} = j = 1$ $j = j + k = 1 + 0 = 1$ $k = \text{temp} = 1$
3	1	$\text{temp} = j = 1$ $j = j + k = 1 + 1 = 2$ $k = \text{temp} = 1$
4	1	$\text{temp} = j = 2$ $j = j + k = 2 + 1 = 3$ $k = \text{temp} = 2$
5	2	$\text{temp} = j = 3$ $j = j + k = 3 + 2 = 5$ $k = \text{temp} = 3$
6	3	$\text{temp} = j = 5$ $j = j + k = 5 + 3 = 8$ $k = \text{temp} = 5$

Table 1.1

Code:

```
def fibonacci_seq(num):
    i = 0
    j = 0
    k = 0
    for i in range(num):
        if i==0:
            print(j)
        elif i==1:
            j = 1
            print(j)
        else:
            temp = j
            j = j+k
            k = temp
```

```
k = temp  
print(j)
```

Execution:

```
fibonacci_seq(10)
```

Output:

```
0  
1  
1  
2  
3  
5  
8  
13  
21  
34
```

11. How would you write the following code using a while loop?

```
def test_function(i,j):  
    if i == 0:  
        return j  
    else:  
        return test_function(i-1,j+1)  
print(test_function(6,7))
```

Answer:

```
def test_function(i,j):  
    while i > 0:  
        i = i-1  
        j = j+1  
    return j  
print(test_function(6,7))
```

12. What will be the output of following code?

```
total = 0  
def add(a,b):  
    global total  
    total = a+b  
print("inside total = ",total)  
add(6,7)  
print("outside total = ",total)
```

Answer:

The output will be as follows:

```
inside total = 13
outside total = 13
```

13. Write code to find the sum of natural numbers from 0 to the given number using recursion.

Answer:

i	Result
0	0
1	$1 + 0 = i(1) + i(0) = 1$
2	$2 + 1 = i(2) + i(1) = 3$
3	$3 + 3 = i(3) + i(2) = 6$
4	$4 + 6 = i(4) + i(3) = 10$
5	$5 + 10 = i(5) + i(4) = 15$

Observe for $i = 0$, the result is 0, thereafter result = $i(n)+i(n-1)$.

Code:

```
def natural_sum(num):
    if num == 0:
        return 0
    else:
        return (num + natural_sum(num-1))
```

Execution:

```
print(natural_sum(10))
```

Output:

```
55
```

14. What would be the output for the following code?

```
def funny(x, y):
    if y == 1:
        return x[0]
    else:
```

```

    a = funny(x, y-1)
    if a > x[y-1]:
        return a
    else:
        return x[y-1]
x = [1,5,3,6,7]
y = 3
print(funny(x,y))

```

Answer: 5

Explanation:

For better understanding, insert print statement as shown in the box here, and execute the code again. We can see the sequence in which the code executes:

```

Def funny(x,y):
    print("calling funny, y = ",y)
    if y == 1:
        return x[0]
    else:
        print("inside else loop because y = ", y)
        a = funny(x, y-1)
        print("a = ",a)
        if a > x[y-1]:
            print("a = ",a, " Therefore a > ",x[y-1])
            return a
        else:
            print("a = ",a, " Therefore a < ",x[y-1])
            return x[y-1]
x = [1,5,3,6,7]
y = 3
print(funny(x,y))

```

Output:

```

calling funny, y = 3
inside else loop because y = 3
calling funny, y = 2
inside else loop because y = 2
calling funny, y = 1
a = 1
a = 1 Therefore a < 5
a = 5
a = 5 Therefore a > 3
5
The answer is 5

```

15. What would be the output of the following code?

```

def funny(x):
    if (x%2 == 1):
        return x+1
    else:
        return funny(x-1)
print(funny(7))
print(funny(6))

```

Answer:

For x = 7

1. x = 7
2. x % 2 is 1
3. return 7 + 1 = 8

For x = 6

1. x = 6
2. x%2 = 0
3. Return funny(5)
4. x = 5
5. x%2 = 1
6. Return x+1 = 6

16. Write Fibonacci sequence using recursion.

Answer:

The Fibonacci sequence = 0,1,2,3,5,8,13.....

i	Result
0	0
1	1
2	$1+0 = i(0) + i(1) = 1$
3	$1+1 = i(2) + i(1) = 2$
4	$2+1 = i(3) + i(2) = 3$
5	$3+2 = i(4) + i(3) = 5$

Observe for $i = 0$, the result is 0 and for $i = 1$, the result is 1. Thereafter, the value of $i(n) = i(n-1) + i(n-2)$. We implement the same, when we try to find Fibonacci code using recursion.

- o The `fibonacci_seq(num)`, takes a number as argument.
- o If `num = 0`, result is 0
- o If `num = 1`, result is 1
- o Else result is `fibonacci_seq(num-1) + Fibonacci_seq(num-2)`
- o If you want to find Fibonacci Sequence for 10 then:
- o For elements 0 to 10
- o Call the `fibonacci_seq()` function
- o $fibonacci_seq(0) = 0$
- o $fibonacci_seq(1) = 1$
- o $fibonacci_seq(2) = fibonacci_seq(1) + fibonacci_seq(0)$
- o $fibonacci_seq(3) = fibonacci_seq(2) + fibonacci_seq(1)$

Code:

```
def fibonacci_seq(num):
    if num<0:
        print("Please provide a positive integer value")
    if num == 0:
        return 0
    elif num == 1:
        return 1
    else:
        return (fibonacci_seq(num-1)+fibonacci_seq(num-2))
```

Execution:

```
for i in range(10):
    print(fibonacci_seq(i))
```

Output:

```
0
1
1
2
3
5
8
13
21
34
```

17. What would be the output of the following program?

```

def test_function(i,j):
    if i == 0:
        return j
    else:
        return test_function(i-1,j+1)
print(test_function(6,7))

```

Answer: 13

Explanation:

i	j	i == 0 ?	return
6	7	No	test_function(5,8)
5	8	No	test_function(4,9)
4	9	No	test_function(3,10)
3	10	No	test_function(2,11)
2	11	No	test_function(1,12)
1	12	No	test_function(0,13)
0	13	Yes	13

The output will be 13.

18. What will be the output for the following code:

```

def even(k):
    if k <= 0:
        print("please enter a positive value")
    elif k == 1:
        return 0
    else:
        return even(k-1) + 2
print(even(6))

```

Answer: 10

Explanation:

k	k <= 0	k == 1	result
6	No	no	even(5)+2
5	No	no	Even(4)+2+2
4	No	no	Even(3)+2+2+2
3	No	no	Even(2)+2+2+2+2
2	No	no	Even(1)+2+2+2+2+2

1	No	yes	0+2+2+2+2+2 = 10
---	----	-----	------------------

19. Write a code to find nth power of 3 using recursion.

Answer:

1. Define function `n_power(n)`, it takes the value of power as parameter(`n`).
2. If `n = 0` then return 1 because any number raise to power 0 is 1.
3. Else return (`n_power(n-1)`).

N	<code>n < 0</code>	<code>n == 0</code>	result
4	No	no	<code>n_power(3)*3</code>
3	No	no	<code>n_power(2)*3*3</code>
2	No	no	<code>n_power(1)*3*3*3</code>
1	No	no	<code>n_power(0)*3*3*3*3</code>
0	No	yes	<code>1*3*3*3*3</code>

Code:

```
def n_power(n):
    if n < 0:
        print("please enter a positive value")
    elif n == 0:
        return 1
    else:
        return n_power(n-1)*3
print(n_power(4))
```

Execution:

```
print(n_power(4))
```

Output:

```
81
```

Descriptive questions and answers

1. Write a short note on function.

Answer:

The answer to this question should include:

1. Definition of function.
 2. Importance of functions in Python coding.
 3. Classification or types of functions in Python programming along with examples.
2. Write a short note on how a function is called?

Answer:

The syntax for calling a function is as follows:

```
function_name(parameters)
```

Where, **function_name** is the name of the function that you want to call and **parameters** are the values that the function requires as per its definition, all the parameters must be separated by commas and must be enclosed in parenthesis. When the function is called the arguments are mapped to the respective parameters and the block of code is executed accordingly.

3. What is the difference between formal arguments and actual arguments?

Answer:

Formal arguments, also known as parameters are used in function definition where they are placed inside the parenthesis. These parameters receive the actual values when a function is called.

Actual arguments are the arguments that exist in the function call. These values are passed on to the function. It is mandatory to pass values as per the function definition. The formal arguments get substituted by the functional arguments when a call is made.

4. What are the different types of functions in Python?

Answer:

There are two types of functions in Python:

1. Built-in functions: library functions in Python.
2. User-defined functions: defined by the developer.

5. What is a function header?

Answer:

The first line of function definition that starts with def and ends with a colon(:) is called a **function header**.

6. When does a function execute?

Answer:

A function executes when a call is made to it. It can be called directly from the Python prompt or from another function.

7. What is a parameter? What is the difference between a parameter and argument?

Answer:

A parameter is a variable that is defined in a function definition whereas an argument is an actual value that is passed on to the function. The data carried in the argument is passed on to the parameters. An argument can be passed on as a literal or as a name.

```
def function_name(param):
```

In the preceding statement, param is a parameter. Now, take a look at the following statement, it shows how a function is called:

```
function_name(arg):
```

arg is the data that we pass on while calling a function. In this statement, **arg** is an argument.

So, a parameter is simply a variable in method definition, and an argument is the data passed on the method's parameter when a function is called.

8. What is the difference between a function that prints a value and a function that returns a value.

Answer:

When a function returns a value, it sends back a value that can be captured or in other words assigned to a variable, which is not the case with the function that prints the value.

9. What is the use of the return statement?

Answer: The return statement exits function, and hands back value to the function's caller. You can see in the code given as follows. The function `func()` returns sum of two numbers. This value assigned to “total” and then the value of total is printed.

```
def func(a,b):  
    return a+b  
  
total = func(5,9)  
print(total)
```

10. Local scope is nested within the global scope, which is nested inside the built-in scope. What is the meaning of this statement?

Answer:

Scope of a variable can be used to know which program can be used from which section of a code. The scope of a variable can be local or global.

Local variables are defined inside a function, and global functions are defined outside a function. Local variables can be accessed only within the function in which they are defined. Global variable can be accessed throughout the program by all functions.

```
total = 0 # Global variable  
def add(a,b):  
    sumtotal = a+b #Local variable  
    print("inside total = ",total)
```

When a reference is made to a variable, Python will look for that variable first within its own scope, and if it does not find it there, it will search up the chain of all enclosing scope and namespace.

11. What are local and global variables?

Answer:

Local Variables are defined and used within a function. The interpreter does not recognize any variable defined within a function outside that function, which means that the variable is recognized only when the function in which it is defined is executed, and its identity is lost the moment the function has finished its task.

A global variable just like its name can be used anywhere in a file. It can be used inside or outside a function without declaring it again.

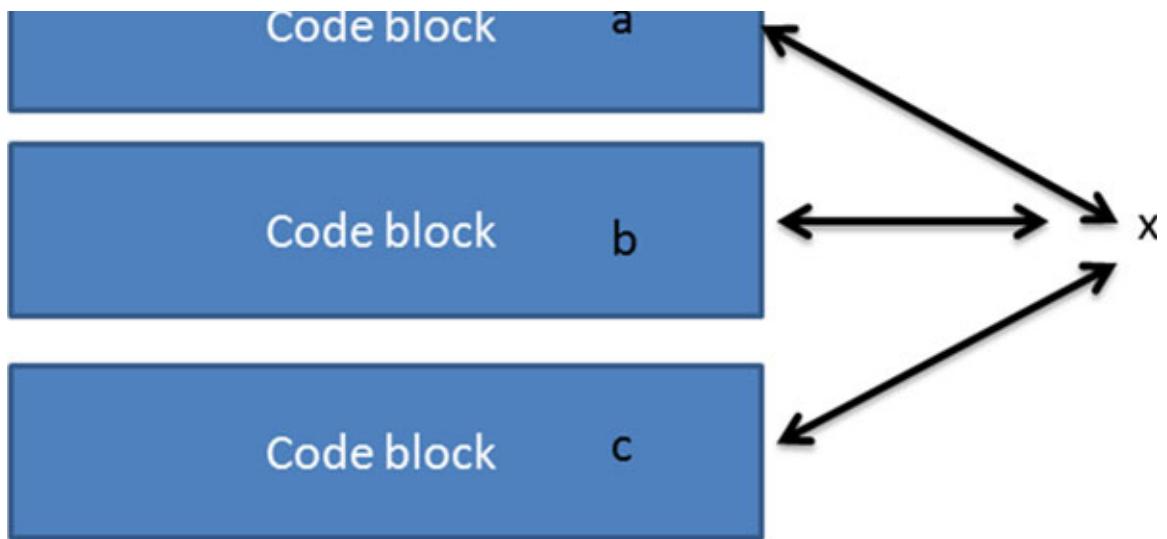


Figure 1.9: Local and Global variables

Variable **x** is a global variable and can be used in any block of code in the file, but that is not the same with variable **a**, **b**, and **c** as they can only be used in the code block in which they are defined. Variables **a**, **b**, and **c** are local variables.

12. What is a default parameter?

Answer:

Default parameter is also known as **optional parameter**. While defining a function if a parameter has a default value provided to it, then it is called a **default parameter**. If while calling a function the user does not provide any value for this parameter, then the function will consider the default value assigned to it in the function definition.

13. Define any three types of function arguments in Python along with example?

Answer:

The are three types of function arguments in Python:

1. **Default Arguments:** Assumes a default value, if no value is provided by the user.

```
def func(name = "Angel"):
    print("Happy Birthday ", name)
```

```
func()  
Happy Birthday Angel
```

You can see that the default value for name is “*Angel*” and since the user has not provided any argument for it, it uses the default value.

2. **Keyword Arguments:** We can call a function, and pass on values irrespective of their positions provided we use the name of the parameter and assign them values while calling the function.

```
def func(name1, name2):  
    print("Happy Birthday", name1, " and ", name2, "!!!")  
  
func(name2 = "Richard", name1 = "Marlin")  
  
Output:  
Happy Birthday Marlin and Richard !!!
```

3. **Variable-length Arguments:** If there is an uncertainty about how many arguments might be required for processing a function, we can make use of variable-length arguments. In the function definition, if a single ‘*’ is placed before parameter, then all positional arguments from this point to the end are taken as a tuple. On the other hand if “**” is placed before the parameter name, then all positional arguments from that point to the end are collected as a dictionary.

```
def func(*name, **age):  
    print(name)  
    print(age)  
func("Lucy", "Aron", "Alex", Lucy = "10", Aron = "15", Alex = "12")  
  
Output:  
('Lucy', 'Aron', 'Alex')  
{'Lucy': '10', 'Aron': '15', 'Alex': '12'}
```

14. What is a fruitful, and non-fruitful function?

Answer:

Fruitful function is a function that returns a value, and a non-fruitful function does not return a value. Non-fruitful functions are also known as **void** function. **Question:** What are anonymous functions?

Answer: Lambda facility in Python can be used for creating function that have no names. Such functions are also known as anonymous functions. Lambda functions are very small functions that have just one line in function body. It requires no return statement.

```
total = lambda a, b: a + b
total(10,50)
60
```

CHAPTER 2

Classes, Objects, and Inheritance

Introduction

In this chapter, you will learn about object oriented programming also known as OOP. In all the previous chapters, you have worked on several examples that followed imperative/procedural style of programming. So, all that we did up till now was use variables, functions, and flow control to create set of instructions that were followed in a predefined fashion. In this chapter, you will see how Python is also an Object-oriented programming language that can make use of classes, objects, attributes, and methods. Actually, Python is a multi-paradigm programming language, it supports imperative, procedural, functional, and object oriented paradigm. Hence, it offers a lot of flexibility and developers have a choice which approach they would like to follow for a specific case.

Structure

In this chapter we will cover the following topics:

- Classes and Objects
- Destructor `__del__()`
- Types of class Variables
- Inheritance

Objective

After reading this chapter, you will be able to:

- Create your own classes.
- Implement the concept of inheritance for advance programming.

Object Oriented Programming follows an approach of modelling real world objects that have some data associated with them and then objects are related to each other the way they are associated in real world. So, a program on school management system can have objects like class, student, teacher, syllabus, subject and so on, and then we can associate them with each other the same way as they are associated with each other in the real world school system. With object oriented programming developers can create their own real world data types.

2.1 Classes and objects

Object-oriented programming allows us to design our code as per our way of thinking or as how we observe a system. While dealing with small problem,, it does not matter how we approach a problem but in the real world software development industry, the projects are generally very complex. If you don't follow the correct approach, it can be very difficult to incorporate each and every possibility and association in your code. However, if we break up the actual scenarios into smaller and simpler parts, focus on one part at a time, and then relate these parts to each other to get the final outcome, it would be much easier to solve the problem and you will be able to deliver a code that has all features incorporated in it. These smaller parts are called **objects**. This concept of coding is very similar to how people think. As a project designer you pay attention to what an entity can do, how it behaves, and what we can do with it. Every object has a state and behaviour. In order to understand the concept of objects properly, let's try to understand the concept of class and instance.

A class is just a definition or blueprint that is required to create an object. So, let's say that we are creating a management system for a company. The details of employees seem to be the most important part of this system. So, if we create a class by the name of **Employee**, then that would act as the blueprint for creating **Employee** objects. You would notice that the term '**instance**' and '**object**' are often used interchangeably in the real world but it is important to understand the difference between the two. An object indicates the memory address of a class and an instance is just a virtual copy of a class at a particular time. Suppose we have a class **Employee** and we have information about 6 employees. Every employee is a different object. Each employee has unique data associated with him or her such as

name, email id, age etc. Every instance is unique actual manifestation of one single object. The object uses the instance to do things. Object is generic but instance is specific. Every instance is assigned a value and this differentiates it from the other other instances. So, we can say that *An object is generic but an instance is an actual manifestation of the class. The instance represents a specific object and it represents a single object that has been created in the memory.* This will be explained in detail in the next section.

In Python, everything is an object and everything is an instance of a class. *Class is a data type that is defined by a developer.* When we create an object of that data type, we call it an instance of that class.

```
>>> x = 1
>>> type(x)
<class 'int'>
>>> y = "String"
>>> type(y)
<class 'str'>
```

So, what is the difference between a class and an object? *An object is a set of instances where as an instance refers to one specific representation.* For example, look at the employee table ([Table 2.1](#)):

Name	Email	Department	Age	Salary
Alex	alex@company_name.com	Finance	42	5,00,000
Evan	evan@company_name.com	HR	34	3,00,000
Maria	maria@company_name.com	HR	30	3,50,000
Pradeep	pradeep@company_name.com	IT	28	7,00,000
Simon	simon@company_name.com	Finance	40	5,00,000
Venkatesh	venkatesh@company_name.com	Sles	25	4,00,000

Table 2.1: Employee

It is obvious that an employee will have much more details like phone number, address, gender, marital status, bank account, and so on, but we are keeping the example simple here to get a grasp of the subject. Now, if we try to create a class `Employee` from the table given above, then all employees (Alex, Evan, Maria, Pradeep, Simon, Venkatesh) created using this class are objects. However, when we talk about a given Employee, then that specific representation of that employee is an instance. Therefore, every row of the table is an instance.

Now, let's pay attention to each column of the table: Name, Email, Department, Age, Salary. These columns hold information related to employee. The variables that hold this information data related to the object are called attributes.

2.1.1 The Employee class

Now, let's start by creating an `Employee` class, having following attributes:

- name
- email
- department
- age
- salary

In Python, in order to create a class we make use of class keyword followed by the class name.

Syntax for creating a class:

```
class class_name:
```

The `class` keyword works like any other block definition and whatever is contained within the class block becomes a part of it. In this scenario, we are talking about Employees. So, we can define the Employee class as follows:

Class Employee:

Note: As a convention the class name is camel case so that we are able to distinguish them from the standard library classes that are

already provided by Python. By camel case means that each word in the class name should begin with the capital letter for example, Employee, MyEmployee, and so on.

You must know that as in case of functions, a class must at least contain one instruction. So, even if you want to create an empty class there should be at least one instruction in the class block. Therefore, an empty class `Employee` would look like:

```
class Employee:  
    pass
```

'`pass`' is an empty instruction; it does nothing but can be used to avoid syntax error.

After the class definition, we can define the attributes and methods of the class. A method is a function defined within the class and it works with the given object. A method requires access to the object on which it is made to work. For this reason, the first argument for every method in the class is the current instance which by convention is called `self`. '`self`' can be used inside the method to access the data stored in an object.

Python has defined some special methods the names of which starts and ends with double underscore. The most commonly used method is `__init__()`. This method is used to initialize the instance of the class or you can say it is a `class constructor`. Whenever an instance of the class is created in Python, it will automatically look for the `__init__()` method. So, we just have to define it but there is no need to call it explicitly. You should also know that `__init__()` is not mandatory for a class. It is optional. So, let's start with defining the `__init__()` function.

```
class Employee:  
    def __init__(self, name, email, department, age, salary):  
        self.name = name  
        self.email = email  
        self.department = department  
        self.age = age  
        self.salary = salary
```

Let us analyse each statement one by one:

```
def __init__(self, name, email, department, age, salary):
```

- `__init__` is a special function, which is constructor of the class.
- The first argument of this function is `self`.
- The `self` connects the method to the instance of the class (which is the object that calls the method).
- The arguments that follows the `self` are values that are method parameters.
- Once the method has access to the object, the values of method parameters are passed on to the object attributes.

Now, let's have a look at the function block.

```
self.name = name
self.email = email
self.department = department
self.age = age
self.salary = salary
```

As you can see, on the left handside of the '=' sign, we have the object attributes, and on the right hand side, we have the method parameters. By convention, we use the same name for the object attributes and method arguments. The method arguments can have different names but that would change the code also. Look at the following block of code:

```
class Employee:
    def __init__(self, a, b, c, d, e):
        self.name = a
        self.email = b
        self.department = c
        self.age = d
        self.salary = e
```

However, we will stick to convention, and in all the examples that we follow, there will not be any deviation from the convention.

Now, let's have a look at how `__init__()` works. If you have a look at the table again, the name of the first employee is "Alex." His e-mail ID is alex@company_name.com. He works in the Finance department; his age is 42, and he has an income of 5,00,000.

Therefore, we create the class instance as follows:

```
e1 = Employee("Alex","alex@company_name.com","Finance",42,500000)
```

This statement calls the `__init__()` function. The method parameters are as follows:

1. '`self`' which connects the method to the instance `e1` of the class.
2. "Alex"
3. alex@company_name.com
4. "Finance"
5. 42
6. 500000

Now, in the `__init__()` method, the values passed on as method parameters are assigned to the object attributes as follows:

```
self.name = "Alex"  
self.email = "alex@company\_name.com"  
self.department = "Finance"  
self.age = 42  
self.salary = 500000
```

With this a separate entity, an instance has been created.

So far, the class looks like this:

```
class Employee:  
    def __init__(self, name, email, department, age, salary):  
        self.name = name  
        self.email = email  
        self.department = department  
        self.age = age  
        self.salary = salary  
  
e1 = Employee("Alex","alex@company_name.com","Finance",42,500000)
```

If you execute this code, you will not receive any output because we have created an object but we have not used it to do anything.

Let's create our next method that prints the values of the object.

```
def print_employee(self):  
    print("Name : ",self.name)  
    print("Email : ",self.email)  
    print("Department : ",self.department)
```

```
print("Age : ",self.age)
print("Salary : ",self.salary)
```

The `print_employee()` method only has `self` as an argument which gives it access to the object (which in our case is `e1`).

A call can be made to the method in conjunction with the object as follows:

```
object_name.method_name()
```

In our case, the object name is `e1` and the method name is `print_employee()`. So, we can call this method using:

```
e1.print_employee()
```

Our code now looks like this:

```
class Employee:
    def __init__(self, name, email, department, age, salary):
        self.name = name
        self.email = email
        self.department = department
        self.age = age
        self.salary = salary

    def print_employee(self):
        print("Name : ",self.name)
        print("Email : ",self.email)
        print("Department : ",self.department)
        print("Age : ",self.age)
        print("Salary : ",self.salary)

e1 = Employee("Alex","alex@company_name.com","Finance",42,500000)
e1.print_employee()
```

Look at the last two lines:

```
e1 = Employee("Alex","alex@company_name.com","Finance",42,500000)
```

This statement creates an instance `e1` of the class `Employee`.

```
e1.print_employee()
```

This statement calls the `print_employee()` method. The argument `self` provides access to `e1`. It then prints the value of all the object attributes. If you execute the code, the output will be as follows:

```

Name : Alex
Email : alex@company_name.com
Department : Finance
Age : 42
Salary : 500000

```

Now, you can create instances to feed in the rest of the values in the table, and call **`print_employee()`** method to print the values.

```

class Employee:
    def __init__(self, name, email, department, age, salary):
        self.name = name
        self.email = email
        self.department = department
        self.age = age
        self.salary = salary

    def print_employee(self):
        print("Name : ",self.name)
        print("Email : ",self.email)
        print("Department : ",self.department)
        print("Age : ",self.age)
        print("Salary : ",self.salary)

#Create instances of class 'Employee' and call the print_employee() method
e1 = Employee("Alex","alex@company_name.com","Finance",42,500000)
e1.print_employee()

e2 = Employee("Evan","evan@company_name.com","HR",34,300000)
e2.print_employee()

e3 = Employee("Maria","maria@company_name.com","HR",30,350000)
e3.print_employee()

e4 = Employee("Pradeep","pradeep@company_name.com","IT",28,700000)
e4.print_employee()

e5 = Employee("Simon","simon@company_name.com","Finance",40,500000)
e5.print_employee()

e6 = Employee("Venkatesh","venkatesh@company_name.com","Sales",25, 200000)
e6.print_employee()

```

2.2 Destructor `del_0`

You have learnt about the instructor `__init__()` of the Python class and it is very natural to wonder if there is a destructor also available in Python as in case of other languages such as C++. Destructor is a special function that is called when the lifecycle of an object gets over. The purpose of this

method is to free the resources that the object had occupied in its lifetime. In Python, the need for destructor is less because Python follows the concept of **garbage collection** for **memory management**. However, Python does have a destructor which actually does not require explicit definition. The destructor is `__del__()` method. When not explicitly defined, it is called automatically for destroying the object and clear the memory resources that were occupied by the object.

```
class Students:
    student_id = 50

    # This is a special function that Python calls when you create new instance of
    # the class
    def __init__(self, name, age):
        self.name = name
        self.age = age
        Students.student_id = Students.student_id + 1
        print('Student Created')

    # This method keeps a count of total number of students
    def Total_no_of_students(self):
        print("Total Number of students in the school are :
              "+str(Students.student_id))

    def __del__(self):
        print('object {} life cycle is over. '.format(self.name))

# create an object stul
stul = Students('Paris', 12)
stul.Total_no_of_students()

# destroy object stul
del stul

#Checking if the object still exists
stul.Total_no_of_students()
```

Code:

```
Student Created
Total Number of students in the school are : 51
object Paris life cycle is over.
Traceback (most recent call last):
  File "F:\2020 \input.py", line 23, in <module>
    stul.Total_no_of_students()
NameError: name 'stul' is not defined
>>>
```

2.3 Types of class variables

A class can have two types of variables:

1. Instance variable
2. Class variable or static variable

Class and instance variables are defined as follows:

```
class Class_name:  
    class_variable_name = static_value  
  
    def __init__(instance_variable_val):  
        Instance_variable_name = instance_variable_val
```

Till now, you have learnt how to create instance attributes using the constructor or the `__init__()` attribute. There is another way to create attributes using common methods. For example, look at the `setGuarantee()` method as follows:

```
class Table:  
    total_tables = 0  
  
    def __init__(self, wood_used, year):  
        self.wood_used = wood_used  
        self.year = year  
        Table.total_tables = Table.total_tables + 1  
        print('Table made of {} wood has been created.'.format (self.wood_used))  
  
    def setGuarantee(self,guarantee):  
        self.guarantee = 25  
t1 = Table('Ebony',2020)
```

When the object `t1` is created, it has only two attributes `wood_used` and `year`. You can also check this using the `__dict__()` method.

```
class Table:  
    total_tables = 0  
  
    def __init__(self, wood_used, year):  
        self.wood_used = wood_used  
        self.year = year  
        Table.total_tables = Table.total_tables + 1  
        print('Table made of {} wood has been created.'.format (self.wood_used))  
  
    def setGuarantee(self,guarantee):  
        self.guarantee = 25
```

```
t1 = Table('Ebony',2020)
print(t1.__dict__)
```

Output:

```
Table made of Ebony wood has been created.
{'wood_used': 'Ebony', 'year': 2020}
```

So, you can see that as of now the object has no attribute by the name of **guarantee**. Now, let's call the **setGuarantee()** method and check again.

```
class Table:
    total_tables = 0
    def __init__(self, wood_used, year):
        self.wood_used = wood_used
        self.year = year
        Table.total_tables = Table.total_tables + 1
        print('Table made of {} wood has been created.'.format (self.wood_used))

    def setGuarantee(self,guarantee):
        self.guarantee = guarantee

t1 = Table('Ebony',2020)
print("Attributes before calling setGaurantee() method.",t1.__dict__)
t1.setGuarantee(25)
print("Attributes after calling setGaurantee() method.",t1.__dict__)
```

Output:

```
Attributes before calling setGaurantee() method. {'wood_used': 'Ebony', 'year': 2020}
Attributes after calling setGaurantee() method. {'wood_used': 'Ebony', 'year': 2020, 'guarantee': 25}
>>>
```

You can also use the **getattr()** method instead of **__dict__** as follows:

```
print("Value of guarantee after calling setGaurantee() method.",getattr
(t1,'guarantee'))
```

Output:

```
Value of guarantee after calling setGaurantee() method. 25
```

All these attributes that we have worked so far are actually **instance attributes** because they have been created in the **__init__()** method or in

other instance method. These attributes have unique data for each instance, and can be accessed as follows:

```
objectName.attributeName
```

Class attributes on the other hand are shared by all instances of a class, and has same value for all objects. They are defined at the class level, which means outside the class methods. You can access it either by `objectName.attributeName` OR `className.attributeName`. The class attribute can be overwritten by instance attribute.

So, if you look carefully at the code for class `Table`, it has an attribute called `total_tables` that increases by 1 every time a table object is created.

```
class Table:  
    total_tables = 0  
  
    def __init__(self, wood_used, year):  
        self.wood_used = wood_used  
        self.year = year  
        Table.total_tables = Table.total_tables + 1  
        print('Table No.{0} : Table made of {1} wood has been  
              created.'.format(Table.total_tables, self.wood_used))  
  
    def setGuarantee(self,guarantee):  
        self.guarantee = guarantee  
  
t1 = Table('Ebony',2020)  
t2 = Table('Teak',2019)  
t3 = Table('Mango wood',2018)
```

Output:

```
Table No.1 : Table made of Ebony wood has been created.  
Table No.2 : Table made of Teak wood has been created.  
Table No.3 : Table made of Mango wood wood has been created.  
>>>
```

Some common methods to access attributes are given in the table as follows:

Description	Format
Create an attribute for an object.	<code>objectname.attr = value</code>
To check if an attribute exists or not.	<code>hasattr(obj, name)</code>

Access an attribute of an object.	<code>getattr(obj, name[, default])</code>
Set an attribute or create it if it does not exist.	<code>setattr(obj, name, value)</code>
Delete an attribute.	<code>The delattr(obj, name)</code>

Table 2.2

2.4 Inheritance

Object oriented languages allow us to reuse the code. Inheritance is one such way that takes code reusability to another level all together. In inheritance, we have a **superclass**, and a **subclass**. The **subclass** will have attributes that are not present in superclass. So, imagine that we are making a software program for a dog Kennel. For this, we can have a dog class that has features that are common in all dogs. However, when we move on to specific breeds, there will be differences in each breed. So, we can now create classes for each breed. These classes will inherit common features of the dog class, and to those features, it will add its own attributes that makes one breed different from the other. Now, let's try something. Let's go step by step. Create a class and then create it's subclass to see how things work. Let's use a simple example so that it is easy for you to understand the mechanism behind it.

Step 1:

Let's first define a class using “*Class*” Keyword as follows:

```
class dog():
```

Step 2:

Now that a class has been created, we can create a method for it. For this example, we create one simple method which when invoked prints a simple message *I belong to a family of Dogs*.

```
def family(self):
    print("I belong to family of Dogs")
```

The code so far is as follows:

```
class dog():
    def family(self):
```

```
print("I belong to family of Dogs")
```

Step 3:

In this step, we create an object of the class **dog** as shown in the following code:

```
c = dog()
```

Step 4:

The object of the class can be used to invoke the method **family()** using dot ‘.’ operator as shown in the following code:

```
c.family()
```

At the end of step 4, the code would be as follows:

```
class dog():
    def family(self):
        print("I belong to family of Dogs")
c = dog()
c.family()
```

When we execute the program, we get the following output:

```
I belong to family of Dogs
```

From here, we move on to implementation of the concept of **inheritance**. It is widely used in object oriented programming. By using the concept of inheritance, you can create a new class without making any modification to the existing class. The existing class is called the **base** and the new class that inherits it will be called **derived class**. The features of the base class will be accessible to the derived class.

We can now create a class **germanShepherd** that inherits the class **dog** as shown in the following code:

```
class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")
```

The object of class **germanshepherd** can be used to invoke methods of class **dog** as shown in the following code:

```

class dog():
    def family(self):
        print("I belong to family of Dogs")

class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")

c = germanShepherd()
c.family()
c.breed()

```

Output

```

I belong to family of Dogs
I am a German Shepherd

```

If you look at the preceding code, you can see that object of class **germanShepherd** can be used to invoke the method of class.

Here are few things that you need to know about **inheritance**.

Any number of classes can be derived from a class using inheritance.

In the following code, we create another derived class **Husky**. Both the classes **germanShepherd** and **husky**, call the family method of **dog** class and **breed** method of their own class.

```

class dog():
    def family(self):
        print("I belong to family of Dogs")

class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")

class husky(dog):
    def breed(self):
        print("I am a husky")

g = germanShepherd()
g.family()
g.breed()
h = husky()
h.family()
h.breed()

```

Output:

```
I belong to family of Dogs
I am a German Shepherd
I belong to family of Dogs
I am a husky
```

A derived class can override any method of its base class.

```
class dog():
    def family(self):
        print("I belong to family of Dogs")

class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")

class husky(dog):
    def breed(self):
        print("I am a husky")

    def family(self):
        print("I am class apart")

g = germanShepherd()
g.family()
g.breed()

h = husky()
h.family()
h.breed()
```

Output:

```
I belong to family of Dogs
I am a German Shepherd
I am class apart
I am a husky
```

A method can call a method of the base class with the same name.

Look at the following code, the class **husky** has a method **family()**, which call the **family()** method of the base class and adds its own code after that.

```
class dog():
    def family(self):
        print("I belong to family of Dogs")

class germanShepherd(dog):
    def breed(self):
        print("I am a German Shepherd")
```

```
class husky(dog):
    def breed(self):
        print("I am a husky")
    def family(self):
        super().family()

        print("but I am class apart")

g = germanShepherd()
g.family()
g.breed()

h = husky()
h.family()
h.breed()
```

Output:

```
I belong to family of Dogs
I am a German Shepherd
I belong to family of Dogs
but I am class apart
I am a husky
```

Points to remember

- Python classes and objects.
- Python is an object-oriented language.
- An object or an instance is a data structure and it is defined by its class.
- A class tells about the characteristics of the object.
- A class consists of attributes, data members, and methods that can be accessed by the dot(.) notation.
- A class is the basic building block for Python.
- A class is created to define all the parameters that are required to create an instance of that class.
- It logically groups all the functions related to an entity.
- Technically a class can be created in any manner, but while programming it is preferred to assign one class to one real world entity.

- Functions defined within a class are also called methods.
- Methods are functions that are defined within a class.
- It makes use of instance variables, class variables and methods to define an object.
- A class is defined as:

```
class Class_name:  
    code (consisting of functions, variables etc)
```

- Class instantiation requires function notation.

```
object_name=class_name()
```

- `class_name()` creates an instance of the class, and it is assigned to local variable `object_name`.
- Instance objects are created by calling the class name and passing the arguments that are required by its `__init__` function.
- Python has a destructor, which does not require explicit definition.
- The destructor is known as the `__del__()` method.
- When `__del__()` is not explicitly defined, it is called automatically for destroying the object so that the memory resources that were occupied by the object are released.
- Class variables are variables can be shared by all instances of the class. Data members hold data associated with class and it's objects.
- Instance variables are defined inside the methods of the class and are associated only with the current instance of the class.
- You can access the attributes of an object by using a dot operator, and if you want to access a class variable, then you will have to use a dot operator with the class name.
- The object of a class supports attribute reference and instantiation. In case of attribute reference the attributes can be referred in the following manner:

```
object_name.object_attribute.
```

Conclusion

Now that you have learnt about how to create our own classes, and work with objects, it is time that you create a small project, and implement the concepts that you have learnt so far. This will help you strengthen the foundations. The next chapter will familiarize you how to work with files in Python.

Multiple choice questions

1. How will you create a new instance of the following class named **Table** follows?

```
class Table:  
    total_tables = 0  
    def __init__(self, wood_used, year):  
        self.wood_used = wood_used  
        self.year = year  
        Table.total_tables = Table.total_tables + 1  
        print('Table made of {} wood has been created.'.format (self.wood_used))
```

- a. t1 = Table('Ebony',2020)
- b. t1 = Table()
- c. t1 = __init__()
- d. t1 = __init__('Ebony',2020)

2. Determine the output of the following code:

```
class Table:  
    total_tables = 0  
    def __init__(self, wood_used, year):  
        self.wood_used = wood_used  
        self.year = year  
        Table.total_tables = Table.total_tables + 1  
        print('Table made of {} wood has been created.'.format (self.wood_used))  
  
    def __del__(self):  
        print('Beautiful Table made of {} wood has been  
destroyed.'.format(self.wood_used))  
  
t1 = Table('Ebony',2020)
```

- a. Table made of Ebony wood has been created.
- b. Beautiful Table made of Ebony wood has been destroyed.
- c. Table made of Ebony wood has been created.

- d. Beautiful Table made of Ebony wood has been destroyed.
- e. Table made of Ebony wood has been created.

Answers

- 1. a
- 2. d

Short questions and answers

- 1. Instance of a class is called as

Answer: Object

- 2. Write a short note on self.

Answer:

- The self is similar to this in Java or pointers in C++.
- All functions in Python have one extra first parameter (the '`self`') in function definition, even when any function is invoked no value is passed for this parameter.
- If there a function that takes no arguments, we will have to still mention one parameter – the "self" in the function definition.

- 3. What are components of a class?

Answer: A class would consist of the following components:

- class Keyword
- Instance and class attributes
- self keyword
- `_init_` function

- 4. The _____ -- method is similar to constructor in Java. It is called as soon as an object is instantiated. It is used to initialize an object.

Answer: `__init__()`

- 5. What is multiple inheritance?

Answer: If a class is derived from more than one class, it is called **multiple inheritance**.

6. A is a subclass of B. How can one invoke the `__init__` function in B from A?

Answer: The `__init__` function in B can be invoked from A by any of the two methods:

- `super().__init__()`
- `__init__(self)`

7. How in Python can you define a relationship between a bird and a parrot?

Answer: Inheritance. Parrot is a subclass of bird.

8. What would be the relationship between a train and a window?

Answer: Composition

9. What is the relationship between a student and a subject?

Answer: Association

10. What would be the relationship between a school and a teacher?

Answer: Composition

11. Write code for creating an empty class Bank?

Answer:

```
class Bank:  
    pass
```

12. Methods that their names starting and ending with double underscore are called _____.

Answer: Special methods.

13. How do you call the special method `__init__()` during object creation?

Answer: The special method `__init__()` is not called explicitly. It is called automatically when an object of a class is created.

14. Variable defined inside a method of a class is called _____.

Answer: Instance variable

15. Variable defined outside the methods of a class are called _____.

Answer: class variables

16. _____ remain same for all objects of the class.

Answer: Class attributes

17. _____ are parameters of `__init__()` method. These values are different for different objects.

Answer: instance variables

18. Identify which is a feature of a class variable and which is a feature of instance variable.

Defined within class construction.	Class variables
Can be accessed using dot operator along with the class name.	Class variables
They are shared by all instances in class.	Class variables
They are owned by class itself.	Class variables
Are owned by instances.	instance variables
Generally have same value for every instance.	Class variables
They are defined right under the class header.	Class variables
Different values for different instances.	instance variables
Important to create an instance of the class to access these variables.	instance variables

Table 2.3

Coding questions and answers

1. Determine the output of the following code:

```
class Kite:  
    def fly(self):  
        print('is meant to fly')  
k = Kite()  
k.fly()
```

Answer:

```
is meant to fly
```

2. Determine the output of the following code:

```
class classLevel:  
    def class1(self):  
        print('There are 20 students in this class')  
    def class2(self):  
        print('There are 15 students in this class')  
    def class3(self):  
        print('There are 41 students in this class')  
    def class4(self):  
        print('There are 30 students in this class')  
  
cl = classLevel()  
cl.class3()
```

Answer: There are 41 students in this class.

3. What will be the output for the following code:

```
class Twice_multiply:  
    def __init__(self):  
        self.calculate(500)  
  
    def calculate(self, num):  
        self.num = 2 * num;  
  
class Thrice_multiply(Twice_multiply):  
    def __init__(self):  
        super().__init__()  
        print("num from Thrice_multiply is", self.num)  
  
    def calculate(self, num):  
        self.num = 3 * num;  
  
tm = Thrice_multiply()
```

Answer:

```
num from Thrice_multiply is 1500  
=>
```

4. For the following code, is there any method to verify whether tm is an object of **Thrice_multiply** class?

```
class Twice_multiply:  
    def __init__(self):  
        self.calculate(500)  
  
    def calculate(self, num):  
        self.num = 2 * num;
```

```

class Thrice_multiply(Twice_multiply):
    def __init__(self):
        super().__init__()
        print("num from Thrice_multiply is", self.num)

    def calculate(self, num):
        self.num = 3 * num;

tm = Thrice_multiply()

```

Answer: Yes, one can check whether an instance belongs to a class or not using `isinstance()` function.

```
isinstance(tm,Thrice_multiply)
```

5. Determine the output of the following code:

```

class Table:
    def features(x):
        print('This table has {} legs.'.format(x))
Table.features(4)

```

Answer: This table has 4 legs.

Descriptive questions and answers

1. Explain the difference between bounded, unbounded, and static methods.

Answer: The **bounded methods** have '`self`' as the first argument. These methods are dependent on the instance and only an instance can be used to access it.

The **unbounded methods** do not take 'self' as an argument, and have not been a part of the language from Python 3.0 onwards.

The **static methods** are bound to the class, and not to the object, and therefore, they are accessed using class names.

Static methods may be used for purposes that do not require the instances to alter. These methods are used as utility functions. For example, if we have to find a factorial of a number it can be done using maths class.

```
>>>math.factorial(8)
```

You can see that to find the factorial we need not create an object to call the function. To make a method static, you can use the `@staticmethod` decorator.

```
class classLevel:  
    def class1(self,string_a):  
        print(string_a)  
  
    def class2(self):  
        print('There are 15 students in this class')  
  
cl = classLevel("Hello")  
cl.class1()  
cl.class2()
```

Output:

```
Traceback (most recent call last):  
  File "F:\2020 - BPB\Python for Undergraduates\code\input.py", line 8, in  
    <module>  
      cl = classLevel("Hello")  
TypeError: classLevel() takes no arguments  
>>>
```

Using Static decorator

```
class Foo:  
    @staticmethod  
    def bar():  
        print('Static method bar()')  
  
    @staticmethod  
    def stat():  
        print("Static method stat()")  
  
    def class1(self,string_a):  
        print(string_a)  
  
# Calling static method bar()  
Foo.bar()  
  
# Calling static method stat()  
Foo.stat()  
#calling bounded method class1()  
f = Foo()  
#calling bounded method class1()  
f = Foo()  
f.class1('bounded method')
```

Output:

```
Static method bar()
Static method stat()
bounded method
>>>
```

2. What are magic methods? Explain.

Answer: **Magic methods** are special methods or dunder (dunder stands for double underscore) methods. They are called special methods because they are not invoked directly. Every operator such as addition, subtraction, and so on, has a magic method associated with itself. There are many methods defined in Python. Some examples of magic methods are as follows:

1. `__init__()`: Initializing an object.
2. `__add__()`: This method is called when we use operator ‘+’.
3. `__str__()`: This method is called automatically while printing an object.
4. `__lt__()`: Is called when we use less than (<) operator.

The invocation of these methods happens behind the scenes. So, when you say `2 + 3` the `__add__()` method is invoked. The biggest advantage of using magic methods is that it makes objects behave like built-in types.

```
>>> 10 + 10
20
>>> '10'+'10'
'1010'
>>>
```

The ‘+’ operator when used with two integer values 10 and 10 returns an integer value 20. The same operator when used with two strings ‘10’ and ‘10’ returns a string value of ‘1010’. This is because here the ‘+’ is a short hand for magic method `__add__()`, which is automatically called. When the operand on the left-hand side is numeric, + works as an addition operator, and if the left operand is a string, it works as a string concatenation operator.

3. Explain operator overloading.

Answer: The assignment of more than one function to a particular operator is called as **operator overloading**. You have seen how magic methods work. Now, let's suppose you and your friend have a bank accounts in same bank.

```
class Bank:

    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    b1 = Bank('customer1','10000')
    b2 = Bank('friend','10000')

    #print values of b1 and b2
    print(b1)
    print(b2)

    #print values of b1 and b2
    b1+b2
```

Suppose you try to print values of these two objects, you will get memory location of these objects rather than the information. Also, if you try to add two objects to know the total available balance, an error will be generated because addition for two Bank objects is not defined.

Output:

```
<__main__.Bank object at 0x03228E20>
<__main__.Bank object at 0x034AB3D0>
Traceback (most recent call last):
  File "F:\2020 - BPB\Python for Undergraduates\code\input.py", line 15, in
    <module>
      b1+b2
TypeError: unsupported operand type(s) for +: 'Bank' and 'Bank'
```

We can define functionality for these operators by overloading `__str__()` and `__add__()` method.

```
class Bank:

    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def __str__(self):
        str1 = 'name: '+self.name+' balance: '+self.balance+'.'
        return str1
```

```
def __add__(self, other):
    balance = int(self.balance) + int(other.balance)
    return balance

b1 = Bank('customer1', '10000')
b2 = Bank('friend', '10000')

#print values of b1 and b2
print(b1)
print(b2)

#print values of b1 and b2
print(b1 + b2)
```

Output:

```
name: customer1 balance: 10000.
name: friend balance: 10000.
20000
>>>
```

CHAPTER 3

Files

Introduction

"**File**" is a very common name that you may keep on hearing every now and then. Your dad may be maintaining separate files for bank documents, office papers, car documents, and so on. In the same way, you must be maintaining different files for different projects in your school or college. So, basically, you need files for storing important information, which is the same in the case of computers as well. Important information can be stored in files on your computer. It is not wise to maintain hardcopy of a huge amount of data, instead you can store all the information that you require in a file on your system.

Structure

- Advantages of storing data in files
- Directory and file management with Python
- Retrieve the directory where Python is installed using `getcwd()`
- Checking contents of Python directory using `listdir()`
- Create directory using `mkdir()`
- Renaming a directory using `rename()`
- Remove a directory using `rmdir()`
- Working with files
 - `Open()` and `Close()`
 - Access mode in file writing
 - Writing to a binary file using the “wb” access mode
 - Reading the contents of a binary file using the “rb” access mode
 - Write and Read strings with binary files

- Other operations in binary files
- File attributes

Objective

After reading this chapter, you will be able to:

- Manage directory and file with Python
- Retrieve the directory current working directory
- Check contents of Python directory
- Create your own directory using `mkdir()`
- Rename a directory using `rename()`
- Remove a directory using `rmdir()`
- Work with files

File is a contiguous set of bytes that is used to store data. Data can be in a text file or an exe file. However, when the computer has to process these files data must be converted to binary format. Let's learn more about files.

3.1 Advantages of storing data in files

Before you start working with files, you need to understand what are the advantages of storing data in files:

1. Data remains stored in the file permanently, that is, it remains in the system even when you have switched off your computer.
2. You can update the file easily as and when required.
3. Once the data is stored in a computer, it can be used by various applications for complex calculations.
4. Files can be used to store any amount of data, which may not be possible physically. Also, once the data is on a file, it can be used for analysis and you can immediately search for any information.

3.2 Directory and file management with Python

Files have three main parts as shown in [*figure 3.1*](#).

1. Header
2. Data
3. End of File

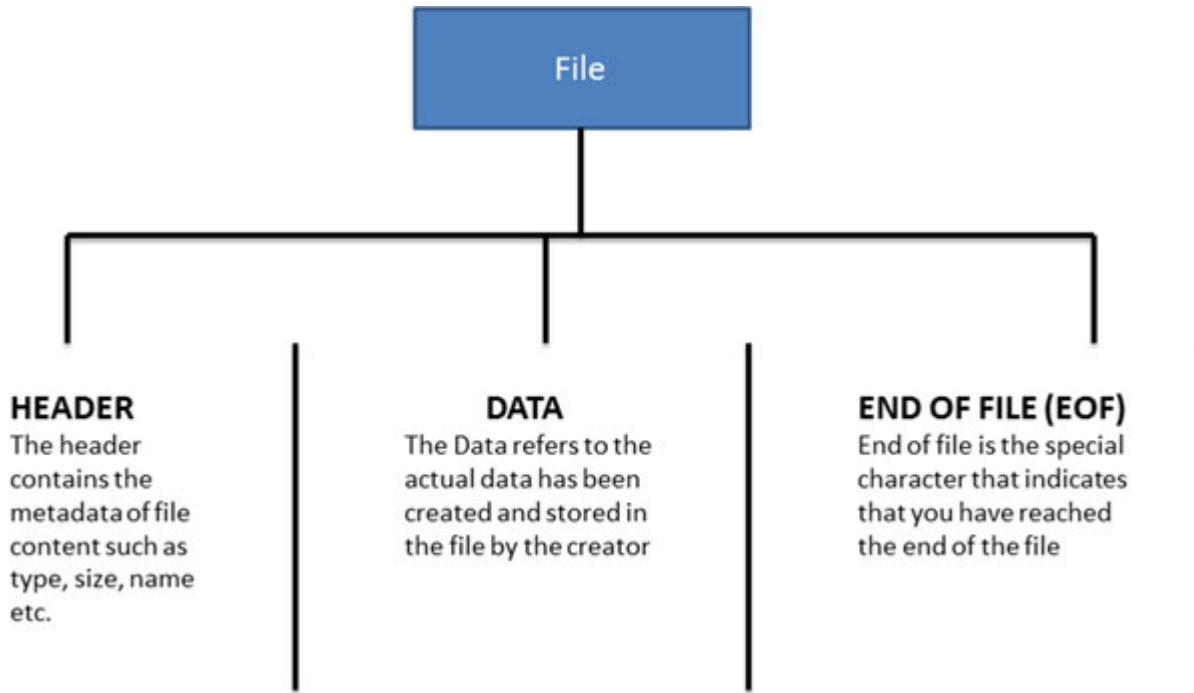


Figure 3.1: Parts of a File

Although we can work with files of any type, but for now, let's restrict ourselves to `.txt` and `.csv` files.

Before we start working on files, let's have a look at the important aspects of Python directory and file management. In a computer files are organized in a directory. To work with the directory, we need to import the `os` module.

```
import os
```

3.3 Retrieve the directory current working directory using `getcwd()`

If you want to retrieve the absolute path of current working directory(`cwd`), you will have to import the `os` module and call its `getcwd()` method as follows:

```
>>>import os
```

```
>>>os.getcwd()
'C:\\\\Users\\\\MYPC\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38-32'
```

To render the directory name using print command, it would be like:

```
>>> import os
>>> print(os.getcwd())
C:\\Users\\MYPC\\AppData\\Local\\Programs\\Python\\Python38-32
```

If you recall in [table 4.1](#) (Escape Characters of [Chapter 4, Basic Core Programming](#)), you learnt about escape characters when you `print('\\\\')` the output is '\\'.

3.4 Checking contents of Python directory using `listdir()`

You can check the contents of the directory with the help of `listdir()` function. This would display the list of all the contents in the directory in a python list.

```
>>> import os
>>> os.listdir()
['comment.py', 'DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt', 'NEWS.txt',
'python.exe', 'python3.dll', 'python38.dll', 'pythonw.exe', 'Scripts', 'tcl',
'Tools', 'vcruntime140.dll']
```

3.5 Create directory using `mkdir()`

In this section, you will learn how to use `mkdir()` to create a directory. So, let's make a folder to keep our python files.

```
>>> import os
>>>os.mkdir("Learning Python")
```

If you give `listdir()` command, you will see the name of the new folder there:

```
>>>os.listdir()
['comment.py', 'DLLs', 'Doc', 'include', 'Learning Python', 'Lib', 'libs',
'LICENSE.txt', 'NEWS.txt', 'python.exe', 'python3.dll', 'python38.dll',
'pythonw.exe', 'Scripts', 'tcl', 'Tools', 'vcruntime140.dll']
```

You can also check in the window explorer. If you go to the same path in the windows explorer, which in this case is ‘C:\Users\MYPC\AppData\Local\Programs\Python\Python38-32’, you will see the newly created folder there:

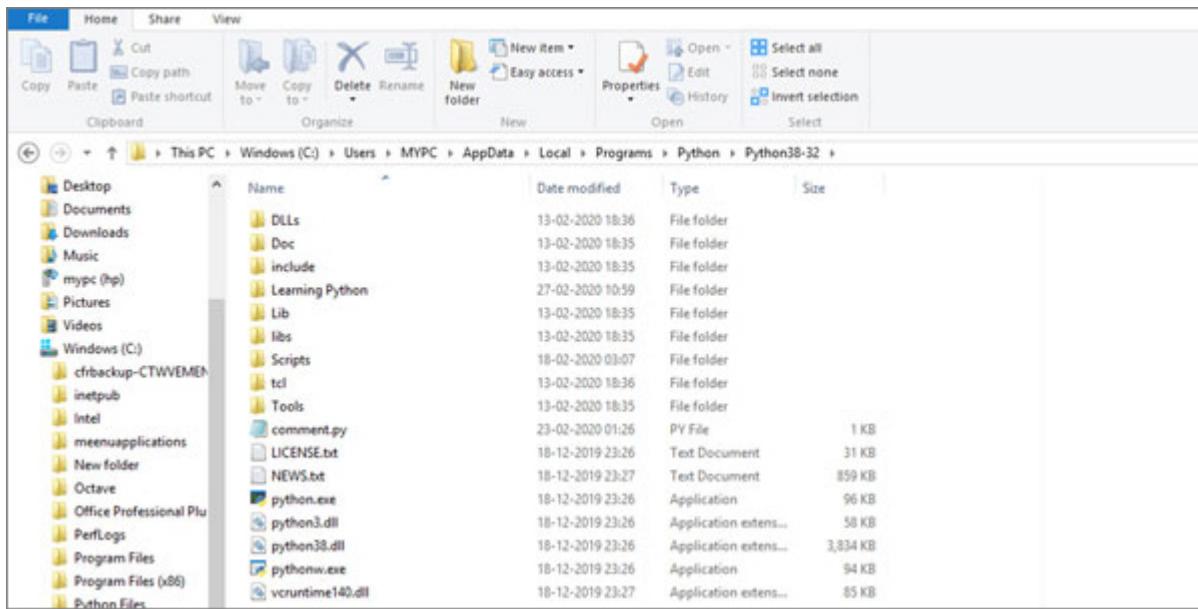


Figure 3.2: Creating new folder

3.6 Renaming a directory using rename()

You can rename the folder “Learning Python” to “Understanding Python” using the `rename()` function.

```
>>> import os
>>>os.rename("Learning Python","Understanding Python")
>>>os.listdir()
['comment.py', 'DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt', 'NEWS.txt',
'python.exe', 'python3.dll', 'python38.dll', 'pythonw.exe', 'Scripts', 'tcl',
'Tools', 'Understanding Python', 'vcruntime140.dll']
```

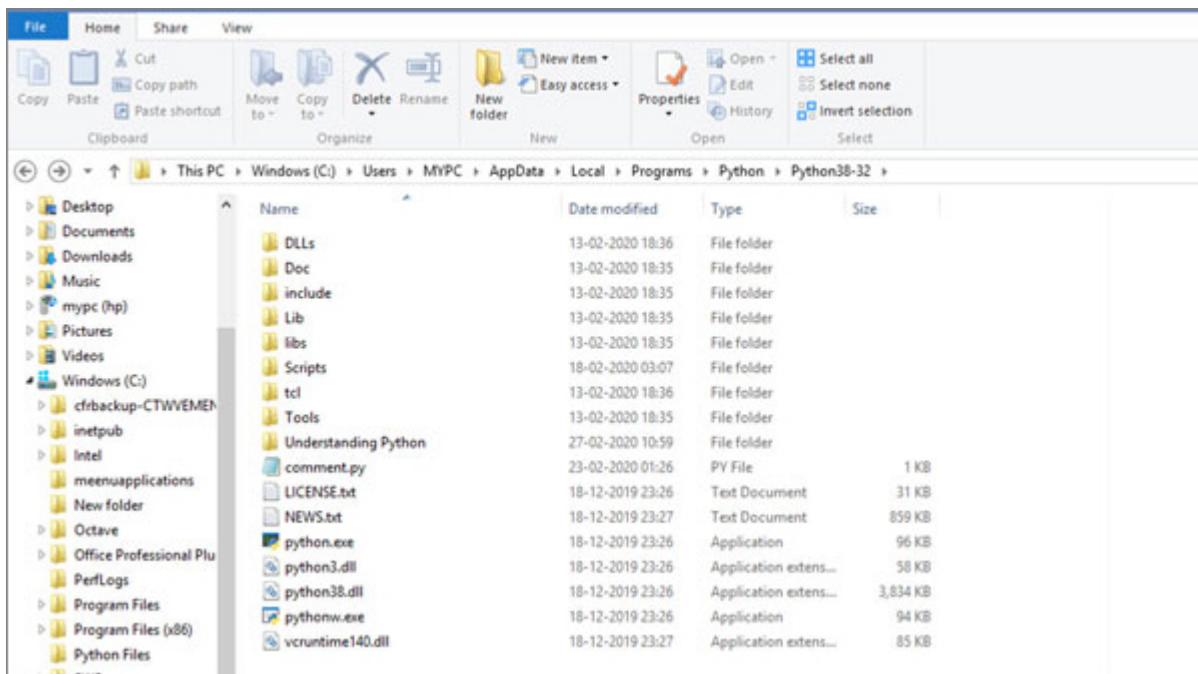


Figure 3.3: renaming using rename()

3.7 Remove a directory using rmdir()

You can remove the directory using `rmdir()`.

In the same directory, create a new text file by the name `delete_file.txt`.

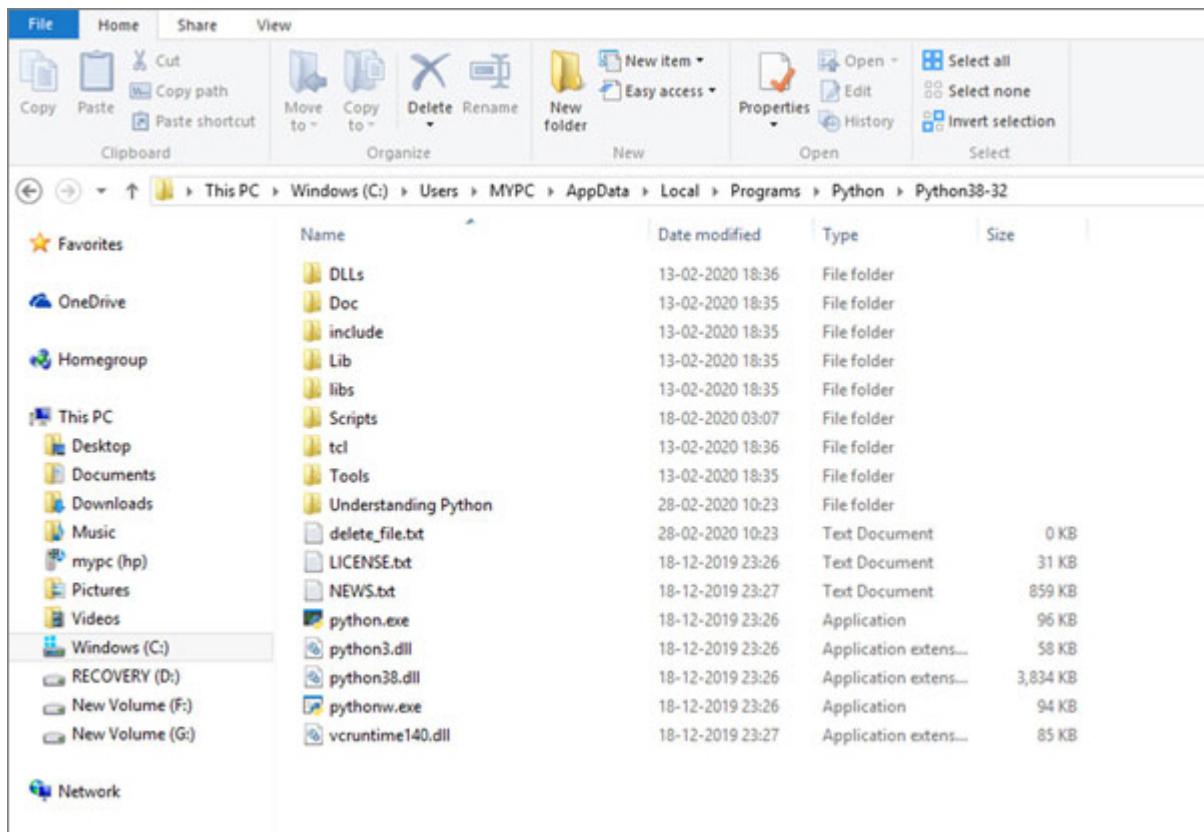


Figure 3.4: Create delete_file.txt

Now, let's see how to remove this file from the directory.

```
>>> import os  
>>>os.remove("delete_file.txt")
```

The file is nowhere to be seen.

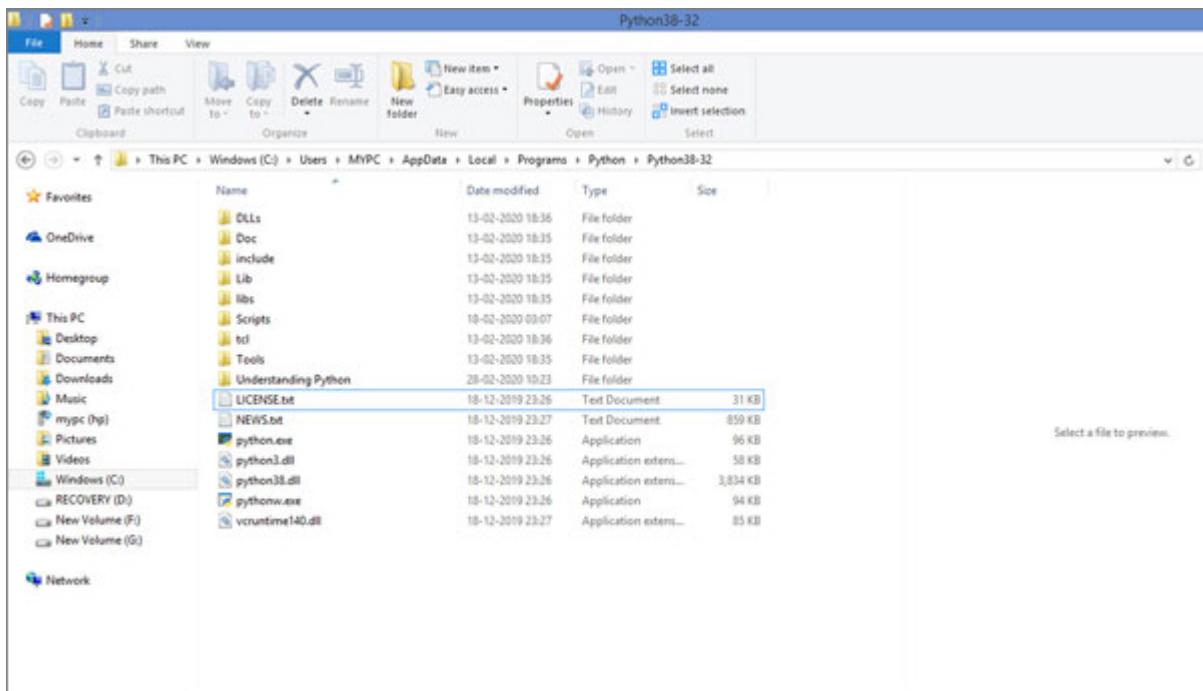


Figure 3.5: After removing the file

Now let's check whether a directory exists with `os.path.exists()`.

```
>>> import os
>>> os.path.exists('C:\\\\Users\\\\MYPC\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38-
32')
True
```

3.8 Working with files

In this section, you will learn how to work with files using Python. In this section you will learn::

- How to open and close a file
- The access mode for writing into a file
- Working with binary files
- File attributes

3.8.1 open() and close()

In order to work with a file, you will have to first open it. This would be the first step. After you have finished working on the file, it is important to

close it. All files that have been opened must be closed.

To demonstrate how we can work with files, let's first create a text file "**first_file.txt**" as shown in [*figure 3.6*](#).

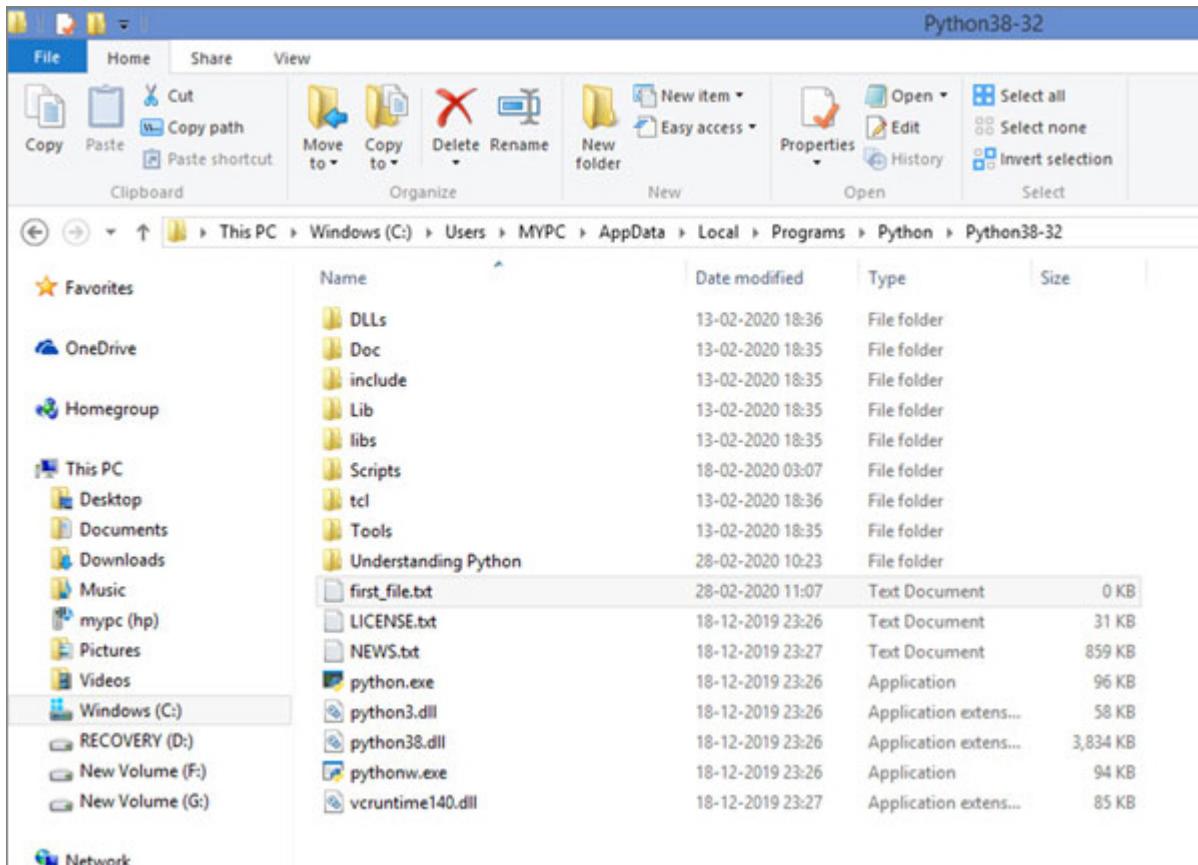


Figure 3.6

The content of the file is as follows:

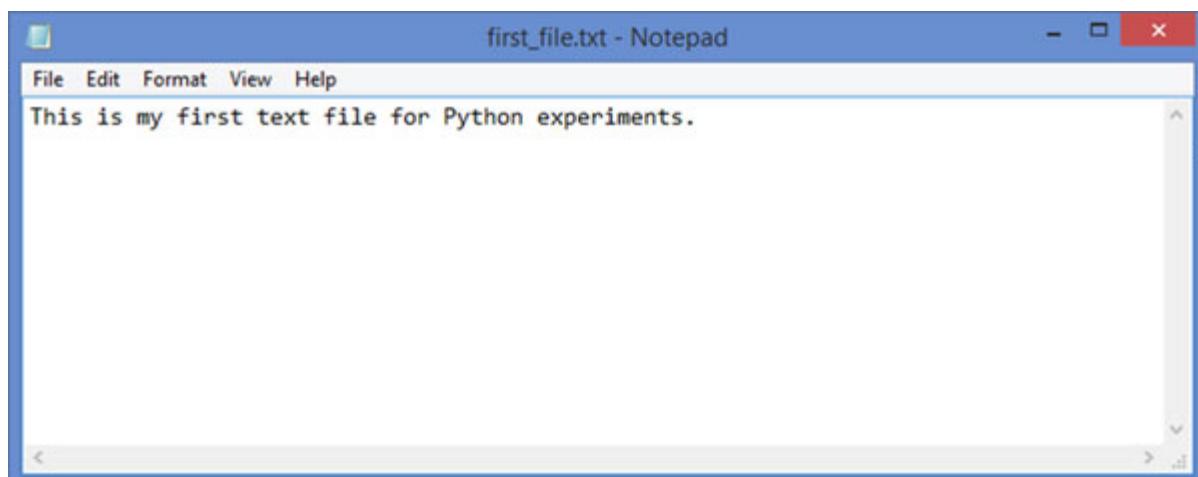


Figure 3.7

To open a file, you can use the `open()` function.

Syntax:

```
file_obj = open(file_name, access_mode, buffering)
```

Where:

file_name: Name of the file that you want to access; it is a mandatory parameter.

access_mode: Optional parameter; the default mode is `read(r)`. It specifies the mode in which it has to open. [Table 3.1](#) provides various types of access modes.

buffering: Optional parameter, 1 means buffering will be performed using the indicated buffer size, any value less than 1 means that the buffer size is the system default.

```
file_handler = open("first_file.txt")
```

Before we start working with files we should learn how to close a file. Any file that has been opened must be closed in order to avoid any unwanted behaviour. A file left open may or may not cause problems while executing a program. To be on the safe side, the file must be closed once you have finished working on it. To close the file, you can use the `close()` function.

```
file_handler.close()
```

To ensure that a file is properly closed even if there is any error, we can use the **try-finally** block.

```
file_handler = open("first_file.txt")
try:
    #code
    .....
finally:
    file_handler.close()
```

Or

You can use the '`with`' keyword shown as follows:

```
>>> with open("first_file.txt") as file_handler:
```

3.8.2 Access mode in file writing

We will now be working with `first_file.txt`.

Table 3.1

File Mode	Name	Description
r	Read	The file opens in read mode, and the file pointer is placed at the beginning of the file. "r" is also considered as the default mode.

```
# Use read() function to extract all the contents of the file in String format.  
>>> file_handle = open("first_file.txt","r")  
>>> try:  
    file_handle.read()  
finally:  
    file_handle.close()
```

Output:

"This is my first text file for Python experiments.\nThere is a lot that can be done with files in Python.\nLet's get Started."

OR

```
# Use read() function to extract first few (in this case 10) characters of the  
file  
>>>try:  
    file_handle.read(10)  
finally:  
    file_handle.close()
```

Output:

'This is my'

OR

```
#Print every line one by one  
  
>>>file_handle = open("first_file.txt","r")  
>>>try:  
    for each in file_handle:  
        print(each)  
finally:  
    file_handle.close()
```

Output:

This is my first text file for Python experiments.

There is a lot that can be done with files in Python.
Let's get Started.

r+	Read and write on existing file.	This access mode opens the file for both read and write. The file pointer is placed at the beginning of the file.
----	----------------------------------	---

(I) Reading a file using r+ mode:

```
>>>file_handle = open("first_file.txt","r+")
>>>try:
    file_handle.read()
finally:
    file_handle.close()
```

Output:

```
"This is my first text file for Python experiments.\nThere is a lot that can be
done with files in Python.\nLet's get Started.\n"
```

(II) Reading and writing with r+ mode:

```
file_handle = open("first_file.txt",'r+')
>>>try:
    file_handle.write("Now I am using r+ mode")
    file_handle.read()
finally:
    file_handle.close()
```

Output:

```
22
"file for Python experiments.\nThere is a lot that can be done with files in
Python.\nLet's get Started.\n"
If you open the file the content would be as follows:
```

Now I am using r+ mode file for Python experiments.
There is a lot that can be done with files in Python.
Let's get Started.

Figure 3.8

The number 22 shows that it starts reading at location 22. If you see a total of 22 characters were inserted.

So, you see that “r+” mode inserts the content at the beginning of the file. However, there may be times when you do not desire this. To insert text at any other position, we need to make use of the `seek()` function. The `seek()` function takes two parameters, the first parameter is the position of the read/writer pointer in the file. The second parameter is optional and defaults to 0, which means beginning of the file. To insert content in the current location, the value of the second parameter should be changed to 2.

(III) Inserting text at the end of the file

```
>>>file_handle = open("first_file.txt",'r+')
>>>try:
file_handle.seek(0,2)
file_handle.write("\n Now I am using r+ mode. Inserting at the end of file")
file_handle.read()
finally:
    file_handle.close()
```

This is my first text file for Python experiments.
There is a lot that can be done with files in Python.
Let's get Started.

Now I am using r+ mode. Inserting at the end of file

Figure 3.9

In order to use “r+” mode, it is important for the file to exist on the system. Else, an error will be displayed.

```
>>> file_handle = open("my_file.txt",'r+')
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
    file_handle = open("my_file.txt",'r+')
FileNotFoundException: [Errno 2] No such file or directory: 'my_file.txt'
>>>
```

Figure 3.10

W	Write	This access mode opens the file in write mode only. If the file exists, then it overwrites the content of the file, and if does not exist then a new file is created, and contents are written on that file.
---	-------	--

```
>>>file_handle = open("first_file.txt","w")
>>> try:
    file_handle.write("Hi There")
finally:
    file_handle.close()
```

The preceding piece of code replaces the content of the file with “Hi There”.

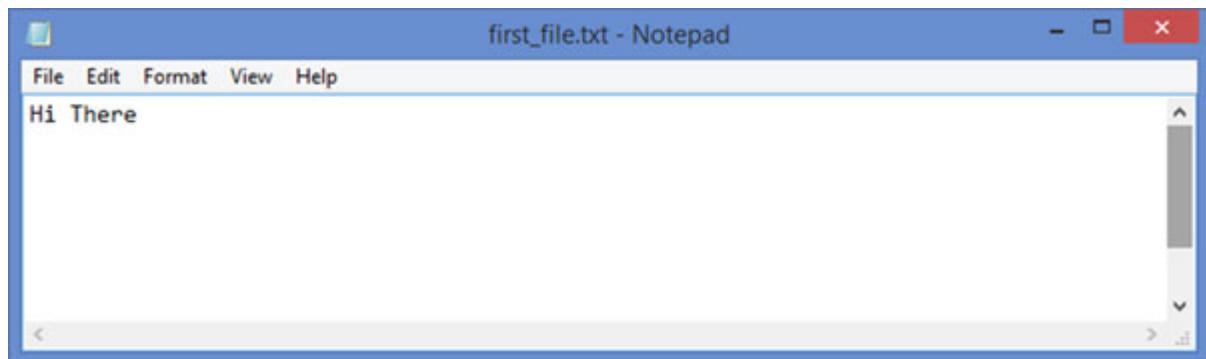


Figure 3.11

If the file does not exist, then a file will be created by the given name, and contents will be added.

```
>>>file_handle = open("first_file1.0.txt","w")
>>>try:
    file_handle.write("Hi There")
finally:
    file_handle.close()
```

Since, no file by the name `first_file1.0.txt` existed, therefore it was immediately created.

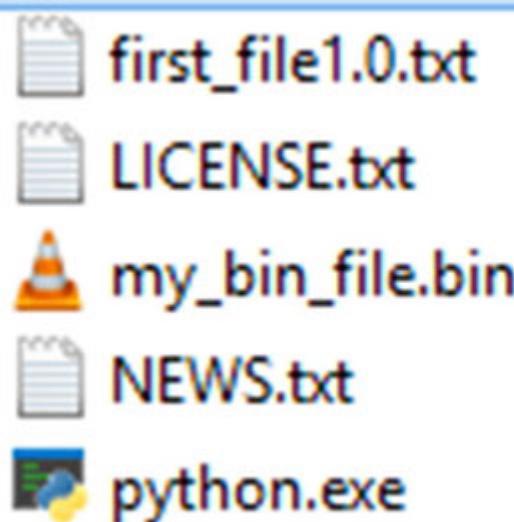


Figure 3.12

The contents of the file were as follows:

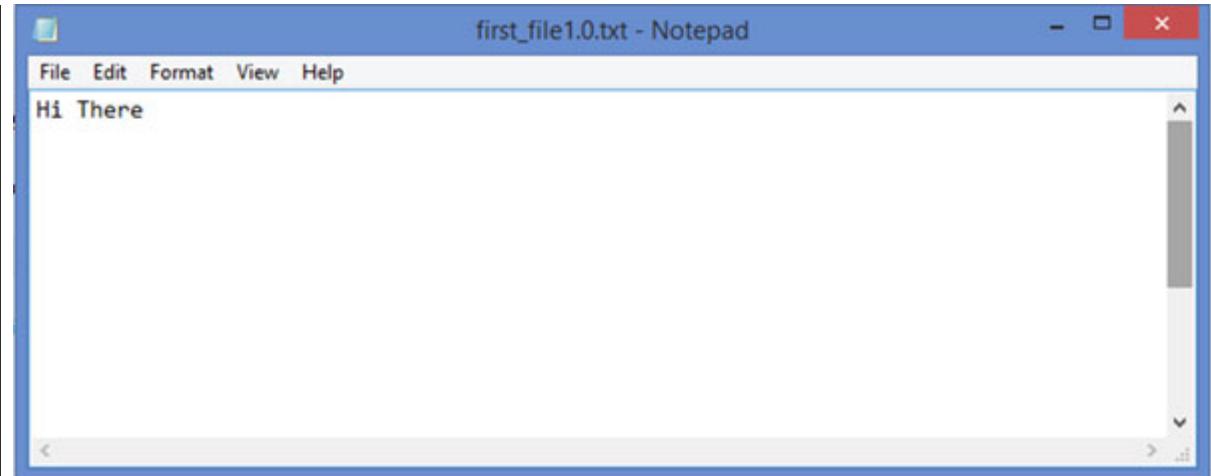


Figure 3.13

When the access mode is “w”, you cannot perform any read operations.

```
>>>file_handle = open("first_file.txt","w")
>>>try:
    file_handle.read()
finally:
    file_handle.close()

Traceback (most recent call last):
  File "<pyshell#17>", line 2, in <module>
    file_handle.read()
io.UnsupportedOperation: not readable
```

w+	Read and write	This opens the file in both read and write mode and if the file does not exist, then a new file will be created with the same name for reading and writing
----	----------------	--

```
>>>file_handle = open("first_file.txt","w+")
>>> try:
    file_handle.write("Hi")
finally:
    file_handle.close()
```

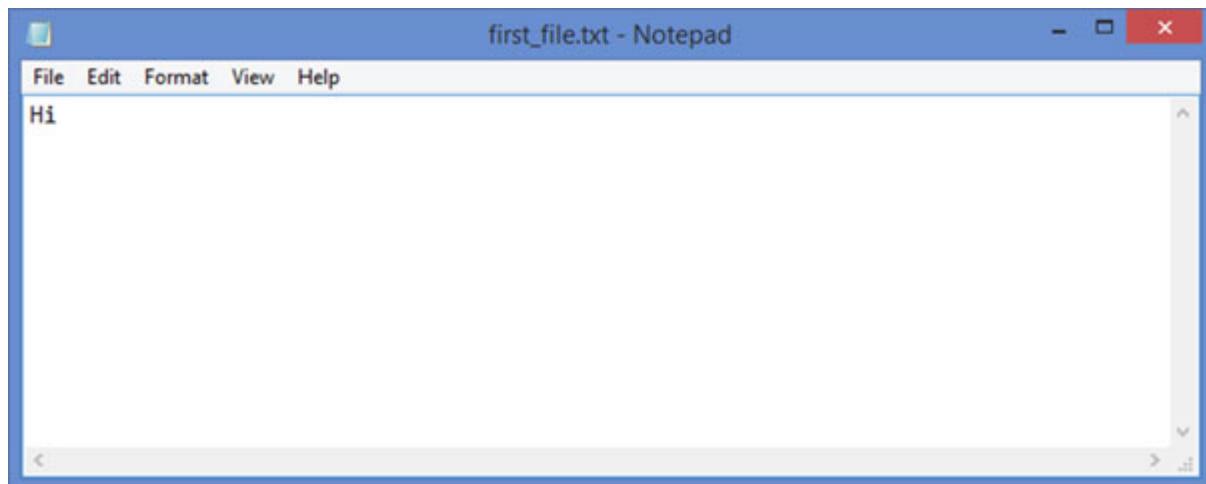


Figure 3.14

a	Append	It adds contents at the end of the file, and if the file does not exist, then a new file is created and content is added to that file.
---	--------	--

```
>>>file_handle = open("first_file.txt","a")
>>>try:
    file_handle.write("\n This is so exciting.")
finally:
    file_handle.close()
```

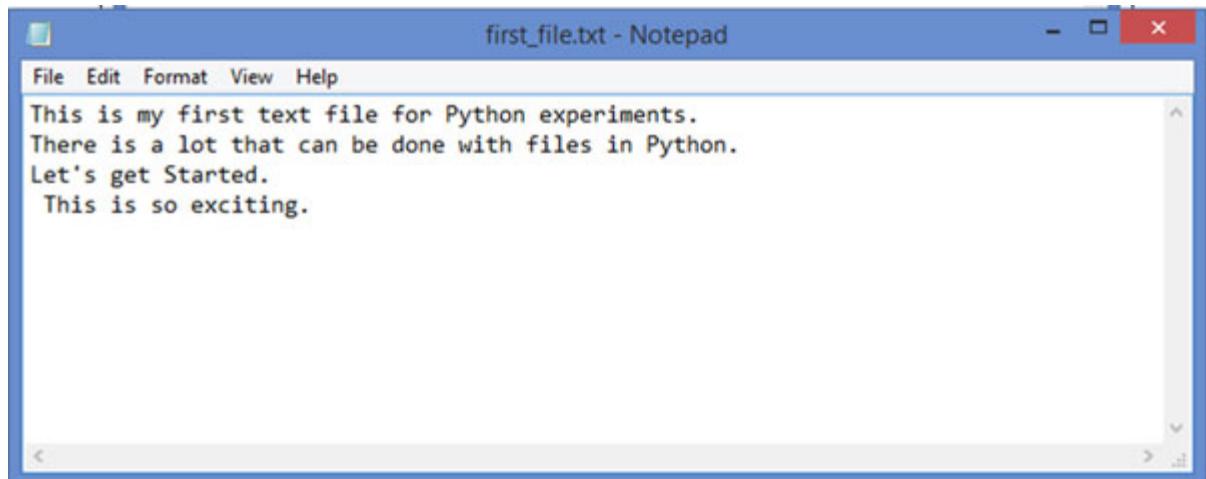


Figure 3.15

a+	Read and append	Opens a file in the read and append mode, and if a file does not exist, it will be created.
----	-----------------	---

3.8.3 Writing to a binary file using “wb” access mode

The following code explains how you can write to a binary file using “wb” access mode.

```
#open the file, in this case, the file by this name does not exist hence it has  
been created
```

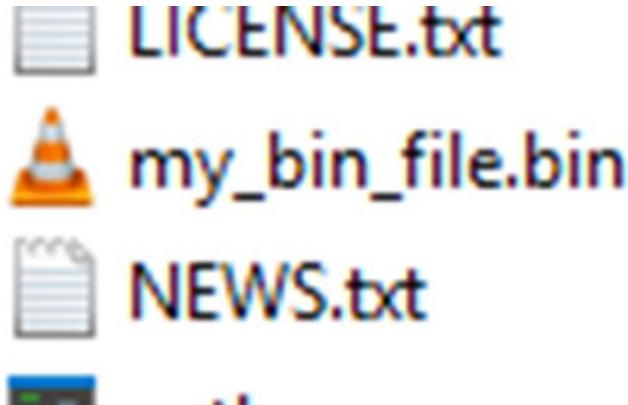


Figure 3.16

```
>>>f =open("my_bin_file.bin", "wb")  
>>>num = [1,2,3,4,5]  
# The list that you want to store must be first converted to byte array to get  
byte representation that can be added to the binary file.  
>>>arr =bytearray(num)  
  
# Just checking how the byte representation looks like  
>>>print(arr)  
bytearray(b'\x01\x02\x03\x04\x05')  
>>>f.write(arr)  
5  
>>>f.close()
```

3.8.4 Reading the contents of a binary file using “rb” access mode

You can read the contents of a binary file using “rb” access mode.

```
>>> f =open("my_bin_file.bin", "rb")  
>>> number = list(f.read())  
>>> print(number)  
[1, 2, 3, 4, 5]
```

3.8.5 Write and read strings with binary files

The following piece of code explains how you can read and write strings in binary files.

```
#Writing
file_handle = open("my_bin_file.bin", "wb")
msg = "Hi there"
arr = bytearray(msg.encode())
file_handle.write(arr)
file_handle.close()

#Reading
file_handle = open("my_bin_file.bin", "rb")
msg = file_handle.read()
print(msg)
print(msg.decode())
```

Output:

```
b'Hi there'
Hi there
```

3.8.6 Other operations on binary file

File Mode	Name	Description
rb+	Read and write on an existing file	Allows you to perform read and write operations on an existing binary file.
wb+	Read and write	Allows you to perform read and write operation on a binary file. If the file does not exist, then a new file will be created with the same name.
ab	Append	Appends content to the end of the file.
ab+	Read and Append	Read and append a file. New content are added at the end of the file. If the file does not exist, then a new file will be created with the same name.

Table 3.2

3.8.7 File Attributes

The following code displays how you can use file attributes to check whether a file is open or closed, its mode and name.

```
file_handle = open("first_file.txt", "w+")
print('Closed or Not - ', file_handle.closed)
print('Mode : ', file_handle.mode)
print('Name : ', file_handle.name)
```

Output:

```
Closed or Not - False
Mode : w+
Name : first_file.txt
```

3.9 Various File related methods

The [Table 3.3](#) lists down various methods available in order to work with files:

Method	Description
<code>close()</code>	Close the file.
<code>flush()</code>	Flush the internal buffer memory.
<code>fileno()</code>	Returns the integer.
<code>next()</code>	Returns next line from the file.
<code>read()</code>	Reads the file.
<code>readline()</code>	Reads an entire line.
<code>readlines()</code>	Returns line by line till the end of ie and then returns a list of lines in the file.
<code>seek()</code>	To change the position.
<code>tell()</code>	Tells the file's current position of the file pointer within the file.
<code>truncate()</code>	Truncates a file.
<code>write()</code>	Writes to a file.
<code>writelines()</code>	Writes sequence of strings to a file.

Table 3.3

Example 3.1

```
read_line()
```

```
>>>file_handle = open("first_file.txt","r")
>>> try:
    file_handle.readline()
finally:
    file_handle.close()

'This is my first text file for Python experiments.\n'
```

Example 3.2

```
readlines()
```

```
File_handle = open("first_file.txt","r")
>>> try:
    file_handle.readlines()
finally:
    file_handle.close()

['This is my first text file for Python experiments.\n', 'There is a lot that can
be done with files in Python.\n', "Let's get Started."]
```

Example 3.3

```
writelines()
```

```
>>>file_handle = open("first_file.txt","w")
>>> try:
    file_handle.writelines(['Hi\n','How are you \n','Good to see you :')])
finally:
    file_handle.close()
```

Conclusion

With this, you have come to the end of this chapter. Files are used to store data. Data can also be stored in a tabular form in a database. In the next chapter, you will learn about MySQL database, and how Python can be used to access a database and execute all types of SQL queries.

Multiple choice questions

1. Which of the following allows you to perform read and write operations on an existing binary file?

- a. rb+
 - b. wb+
 - c. ab
 - d. ab+
2. Allows you to perform read and write operation on a binary file. If the file does not exist, then a new file will be created with the same name.
- a. rb+
 - b. wb+
 - c. ab
 - d. ab+
3. Appends content to the end of the file.
- a. rb+
 - b. wb+
 - c. ab
 - d. ab+
4. Read and append a file. New content are added at the end of the file. If the file does not exist, then a new file will be created with the same name.
- a. rb+
 - b. wb+
 - c. ab
 - d. ab+
5. What does the following statement do?
- ```
file_handle = open("my_bin_file.bin", "rb")
```
- 1. Opens file for writing only.
  - 2. Opens file for both reading and writing.
  - 3. Opens a file for both reading and writing in binary format.
  - 4. Opens a file for only reading in the binary format.

## Answer

1. a
2. b
3. c
4. d
5. d

## Short questions and answers

1. Name the function that is used to delete files.

**Answer:** `remove()`

```
>>> import os
>>>os.remove("fileName.txt")
```

2. Name the method that is used to create directory in the current directory.

**Answer:**

```
>>> import os
>>>os.mkdir("directory_name")
```

3. What is the use of the `chdir()` method?

**Answer:** The `chdir()` method is used to change the current directory. It takes one argument which is the name of the directory that we want to make the current directory.

```
>>> import os
>>>os.chdir('directory_name')
```

4. Which method is used to check the current working directory?

**Answer:** `getcwd()` method

```
>>> import os
>>>os.getcwd()
```

5. What is the purpose of `rmdir()`?

**Answer:** The `rmdir()` method is used to remove or delete a directory. The name of the directory that has to be removed is passed as an argument.

## 6. Fill in the blanks

1. \_\_\_\_\_ are used to store huge collection of data and files permanently.
2. A file can be stored in two ways: \_\_\_\_\_ and \_\_\_\_\_.
3. The type of operations to be performed on open file are specified by the \_\_\_\_\_.
4. A \_\_\_\_\_ file stores information in ASCII or Unicode characters.
5. In text file, each line of text is terminated by a special character known as \_\_\_\_\_.
6. \_\_\_\_\_ files are files that have information stored in the same format in which it is stored in the memory.
7. In binary files, there are no \_\_\_\_\_ for a line.

**Answer:**

1. Files
2. Text, Binary
3. Access Modes
4. Text
5. End of line (EOL)
6. Binary
7. delimiter

## 7. Fill in the Blanks

1. `fileObject._____`(`sequence_of_strings`), writes the strings to the file.
2. \_\_\_\_\_ function reads the whole file and returns the text as a string.
3. The \_\_\_\_\_ method used to find current position of the file.

4. The \_\_\_\_\_ method of a file object flushes any unwritten information and closes the file object.
5. The `tell()` method returns an \_\_\_\_\_ value that provides the current position of file pointer in the file.

**Answer:**

1. ‘writelines’
  2. `read()`
  3. `tell()`
  4. `close()`
  5. integer
8. Look at the code given as follows and answer the following questions:

```
fileHandle = open('x', 'a')
/
fileHandle ()
```

1. What type of file is x?
2. Fill in the blank with a statement that would write 'Hello World' in file x.

**Answer:**

1. x is a text file
2. `fileHandle.write('Hello World')`

## Coding questions and answers

1. There is a folder by the name ‘mar2020’, which must be changed to ‘jun2020’. How would you do that?

**Answer:**

```
>>> import os
>>> os.rename("mar2020", "jun2020")
```

2. Write a statement in Python to perform the following operations:
  1. Open a file “MYPET.TXT” in write mode

2. Open a file “MYPET.TXT” in read mode

**Answer:**

1. `file_handle = open("MYPET.TXT ", "w")`
2. `file_handle = open("MYPET.TXT ", "r")`

## Descriptive questions and answers

1. What is the difference between file modes r+ and w+ with respect to Python?

**Answer:**

1. **w+** - is Read and write operation.

This opens the file in both read and write mode and if the file does not exist, then a new file will be created with the same name for reading and writing.

2. **r+** - is Read and write on existing file.

This access mode opens the file for both read and write. The file pointer is placed at the beginning of the file.

2. Differentiate between the file modes r+ and rb+ with respect to Python.

**Answer:**

1. **r+-** Read and write on existing file.

This access mode opens the file for both read and write. The file pointer is placed at the beginning of the file.

2. **rb+** - Read and Write on the existing file

Allows you to perform read and write operations on an existing binary **file.rb+** - Read and Write on the existing file. Allows you to perform read and write operations on an existing binary file.

3. How do you use the ‘with’ statement?

**Answer:** The ‘with’ statement can be used to write two statements that are related to each other as pairs:

```
'with open(filename,mode) as fileHandle:
----code to be execute----
```

This statement automatically closes the file after the nested block of code even if an exception occurs before the code ends.

# CHAPTER 4

## MySQL for Python

### Introduction

Now, that you are clear about how Python works, it is time to dive into some more advance topics. This chapter will take you on a different journey. Often a software application is associated with a database for storing information. In this chapter, you will learn how to install MySQL database, how to configure it with Python, and how to use Python to interact with MySQL.

### Structure

- Installing and configuring MySQL
  - How to install MySQL
  - Configuration of MySQL
- Creating a database in MySQL
  - Basic rules for writing SQL queries
- Connect MySQL database to Python
  - Create a database in Mysql using Python
  - Retrieve records from database using Python
- Creating a database using Python
  - Creating database directly using MySQL
  - Creating database using Python
- Working with database using Python
  - Inserting records
  - Select records
  - Fetching selected records from a table in Python using the ‘WHERE’ clause

- Using **ORDER BY** Clause
- Deleting records using the **DELETE** command

## **Objectives**

After reading this chapter, you will:

- Learn how to install and configure MySQL on your machine
- Learn basic rules for executing SQL queries
- Create a MySQL database using Python
- Execute all types of SQL queries on MySQL data base using Python.

We will start this chapter by understanding how MySQL can be installed, and brush up basic SQL concepts. Thereafter, we will see how to execute SQL queries using Python.

## **4.1 Installing and configuring MySql on your system**

In addition to Python basics, it is important to have sound knowledge of **Structured Query Language (SQL)** to work with MySQL. SQL is the standard language designed for working with databases in order to store, create, update, manage, or delete data. In this section, you will learn how to install and configure MySQL on your system.

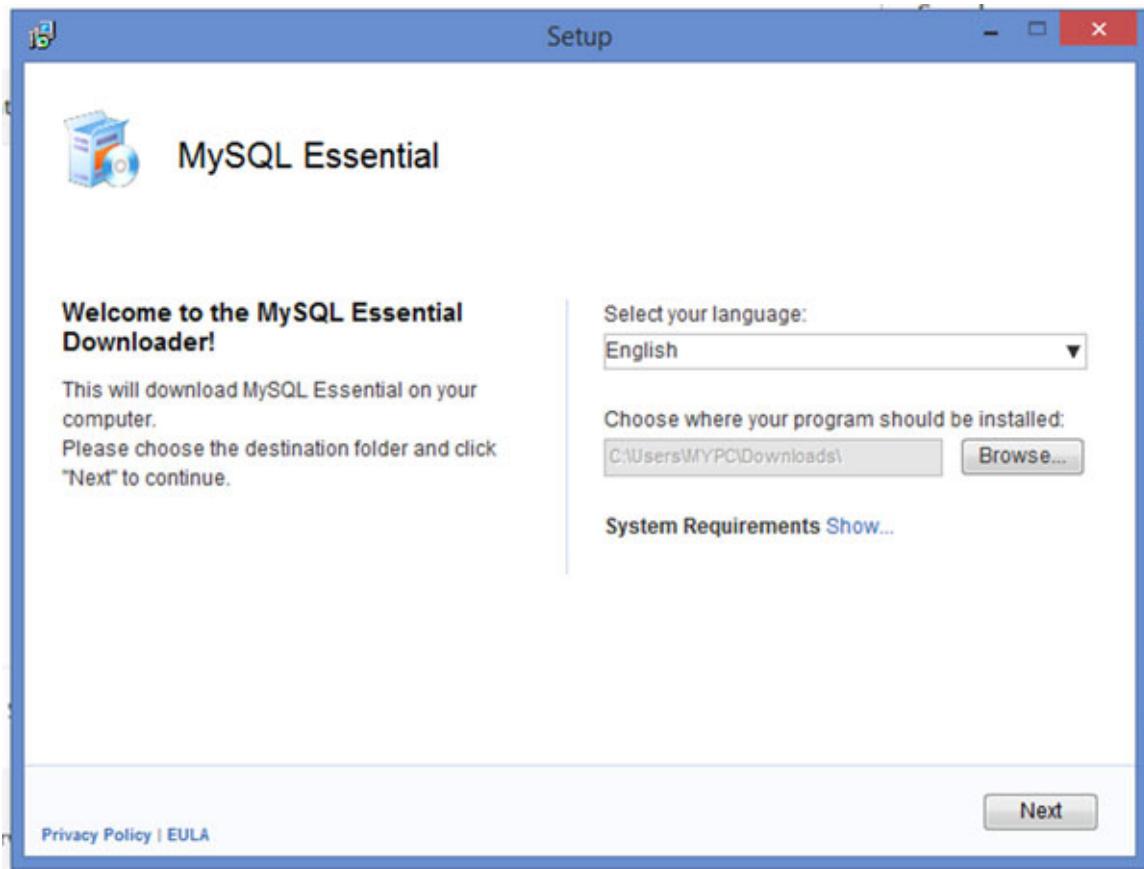
### **4.1.1 How to install MySQL**

If you already have MySQL server installed on your server, then you can skip this section. Here, this section will provide you the steps required to install MySQL 6.0 on Windows computer. MySQL essential has just what is essential, and for beginners level, this is a good choice.

If you are learning on a standalone system and you are a beginner, it is recommended that you to install MySQL Essentials from <https://mysql-essential.en.uptodown.com/windows>. Follow the steps below:

#### **Step 1:**

|          |       |                                                                                                                                    |
|----------|-------|------------------------------------------------------------------------------------------------------------------------------------|
| Download | MySQL | Essential.( <a href="https://mysql-essential.en.uptodown.com/windows"><u>https://mysql-essential.en.uptodown.com/windows</u></a> ) |
|----------|-------|------------------------------------------------------------------------------------------------------------------------------------|



*Figure 4.1: MySQL Essential set up*

## **Step 2:**

Double-click on the mysql-essential-6.0.0.msi package to start the installation process.

1. This would display MySQL Server 6.0 – Setup Wizard Window.
2. To continue with the installation process, click on the **Next** button.
3. You will now have to select the **Setup Type**. The Setup type is of three types:
  - a. Typical for general use.
  - b. Complete for installation of all program feature.
  - c. Custom to select which all programs you want to install. It is recommended to go ahead with typical installation. However, if you want to change the path of software installation, then you can opt for custom installation.
4. For this book option (a) that is **typical** was selected.

5. The next screen will prompt you to login or create a MySql account. Signing up is not mandatory.
6. You can create an account or click on **skip sign-up** radio button and click on the **Next** button.
7. Now, you have reached the last screen of the installation process.
8. Before pressing the **Finish** button, ensure that the “Configure the MySql Server now” checkbox is clicked.

### **4.1.2 Configuration of MySQL**

We will now move on to configuration of MySQL. Please follow the steps below::

1. After clicking the **Finish** button, you would be presented by “MySQL Server Instance Configuration Wizard”.
2. If the window does not pop up on its own, then go to the start button on the desktop and look for the Wizard and click on it. Click on **Next** on the first screen. Now, follow the steps below:
  - i. You have to first select whether you want to go for detailed or standard configuration. Select **Detailed Configuration**.
  - ii. Now, you have to select the server type. Your selection will influence memory, disk, and CPU usage. Go for Server if you are planning to work on a server that is hosting other applications. Click on **Next**.
  - iii. If you have selected Server in the preceding step, then you will be presented a screen where you would be asked to set a path for InnoDB data file to enable InnoDB database engine. Without making any modifications click on **Next**.
  - iv. In this screen, you will have to set the approximate number of concurrent connections to the server. Ideally, for general purpose usage, it is best to go with the first option, that is, “**Decision Support(DSS)/OLAP**”. Click on **Next** after making your selection.
  - v. In the next screen, you see two options:
    - a. Enable TCP/IP Networking
    - b. Enable Strict Mode.

- c. Check the “**Enable TCP/IP Networking**” option. The second option **Enable Strict Mode** will be checked by default. Most applications do not prefer this option, so if you do not have a good reason for using it in the Strict mode, then uncheck this option, and click on the **Next** button.
- vi. You will now be asked to set the default character set used by MySQL. Select **Standard Character Set** and click on **Next**.
- vii. In this step, you have to set window options. You will see three check boxes : **Install as Windows Service**, **Launch the MySQL Server automatically** and **Include Bin Directory in Windows PATH**. Check all three boxes and click on **Next**.
- viii. You will now have to set the root password for the account. Check **Modify Security Settings** and provide your password details. If your server is on the internet, then avoid checking **Enable root access from remote machines**. Also, it is not recommended to check the **Create An Anonymous Account** option. Please save the password at a safe place.
- ix. This window will show you how the configuration is processing. Once the processing is over **Finish** button will be enabled and you can click on it.



*Figure 4.2*

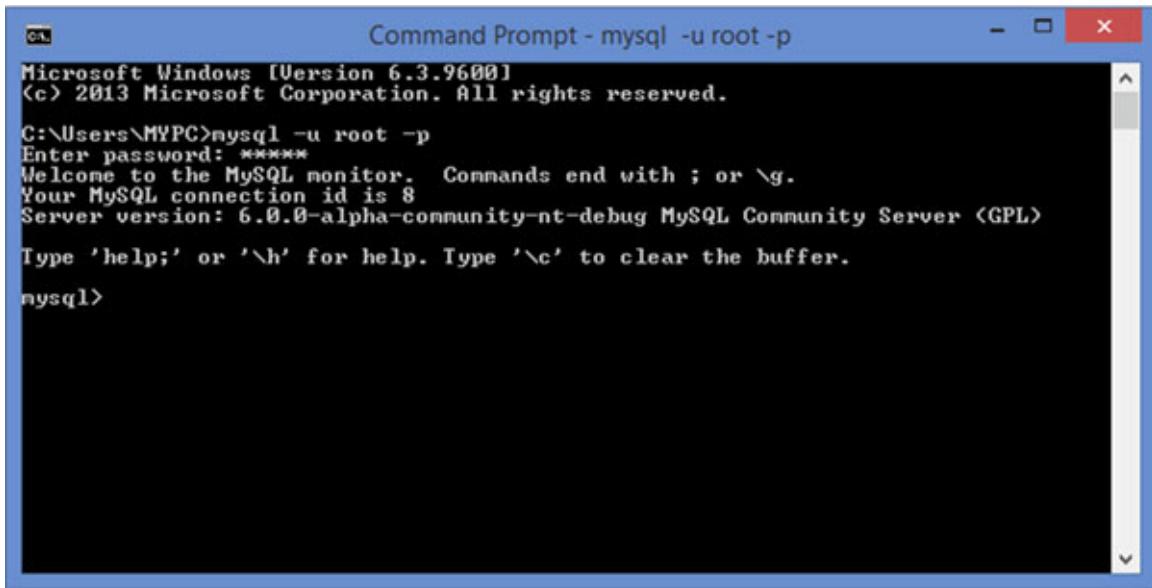
**Note:** You may come across terms like MySQL community server, MySql installer and MySql essentials while looking for information on MySql. As the name suggests Essentials provides just what is essential without any additional components and ideal to start learning. Both installer and community server have full server features, but it is only that the community server is installed online, and for installer you have to download the package and it can be installed offline.

## 4.2 Creating a database in MySQL using the command line tool

Now that you have configured MySQL, open your command prompt and type the command given in the following box:

```
mysql -u root -p
```

You will now be prompted to provide your password. Enter your password, and your command prompt should look like [figure 4.3](#):



```
61 Command Prompt - mysql -u root -p
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MVPC>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 6.0.0-alpha-community-nt-debug MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

*Figure 4.3*

To start off with SQL, we have to first create a database. For this please follow the steps given as follows:

### **Step 1: Start your command prompt window**

Go to **start > Run** and type **cmd**. Or go to **start** and just search for **cmd**. Your command prompt window will open.

### **Step 2: Log into your account**

Give the following command:

```
mysql -u root -p
```

Now, press **Enter**. The command prompt window will now show **Enter password:**

1. Type the root password that you had set at the time of configuration.
2. We will now create a database for managing our work. Let's name the database as PYMYSQL.
3. Type the following instructions:

### **Syntax:**

```
CREATE DATABASE DATABASE_NAME;
```

So, let's say we create a MySQL database for Python programs by the name PYMYSQL. So the command for creating database PYMYSQL

will be as follows:

```
CREATE DATABASE PYMYSQL;
```

#### 4.2.1 Basic rules for writing SQL queries

Here are few things that you need to know:

- SQL commands are not case sensitive. You can use upper or lower case it does not matter.
- The database name cannot have spaces.
- All MySQL commands end with a semicolon (;). If you forget to enter semicolon, then in the next line enter ‘;’ and press enter so that the command can be executed.

If the database is created, you will see the following message:

```
Query OK, 1 row affected
```

#### 4.3 Connect MySQL database to Python using MySQL connector

The standard interface for Python database access modules is defined by the Python Database API. All the Python database module adhere to this interface. The DB-API encourages similarity between modules that would be used by Python to access database. This helps in achieving consistency. It becomes easy to work with different databases as the process involved is similar. These are some basic steps that must be followed to work with any database. The API includes the following:

1. Importing the API module
2. Getting connected to the database
3. Execute SQL queries
4. Close the connection.

You can install mysql connector by giving the `pip installmysql-connector-python` command.

```

C:\Users\MYPC>cd C:\Users\MYPC\AppData\Local\Programs\Python\Python38-32\Scripts
C:\Users\MYPC\AppData\Local\Programs\Python\Python38-32\Scripts>pip install mysql-connector-python
Collecting mysql-connector-python
 Downloading mysql_connector_python-8.0.19-py2.py3-none-any.whl (355 kB)
 ! [0%] |: 355 kB 504 kB/s
Collecting protobuf==3.6.1
 Downloading protobuf-3.6.1-py2.py3-none-any.whl (390 kB)
 ! [0%] |: 390 kB 3.3 MB/s
Collecting dnspython==1.16.0
 Downloading dnspython-1.16.0-py2.py3-none-any.whl (188 kB)
 ! [0%] |: 188 kB 3.3 MB/s
Requirement already satisfied: six>=1.9 in c:\users\mypc\appdata\local\programs\python\python38-32\lib\site-packages (from protobuf==3.6.1->mysql-connector-python) (1.14.0)
Requirement already satisfied: setuptools in c:\users\mypc\appdata\local\programs\python\python38-32\lib\site-packages (from protobuf==3.6.1->mysql-connector-python) (41.2.0)
Installing collected packages: protobuf, dnspython, mysql-connector-python
Successfully installed dnspython-1.16.0 mysql-connector-python-8.0.19 protobuf-3.6.1
C:\Users\MYPC\AppData\Local\Programs\Python\Python38-32\Scripts>

```

*Figure 4.4*

### 4.3.1 Create a database in Mysql using Python

Give the following command in the command prompt:

```
Show databases;
```

```

C:\Users\MYPC>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 6.0.0-alpha-community-nt-debug MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| pymysql |
| test |
+-----+
4 rows in set (0.00 sec)

```

*Figure 4.5: Displaying Databases*

The next step is to grant privileges to your database. You can now create a user that will have complete privilege over this database.

```
GRANT ALL PRIVILEGES ON pymysql.* TO 'pymysqldadmin'@'localhost' IDENTIFIED BY
'pymysql123';
```

Please Note:

1. In this case, the name of the user that has full privileges on PYMYSQL database is `pymysqladmin`.
2. The password is `pymysql123`.

Now, to connect to the database, you will have to follow two simple steps:

### Step 1: Import `mysql.connector`

```
>>> import mysql.connector as msq
```

### Step 2: Open connection to database

It is important to connect to MySQL using `connect()` method, which returns a connection object.

#### Syntax:

```
'your_connection_name =
msql.connect(host=host_name,user=user_name,passwd=database_password,charset='utf8',
database= database_name)
```

In our case we create a connection object `pycon` using the `connect()` method.

```
>>>pycon = msql.connect(host =
'localhost',user='pymysqladmin',passwd='pymysql123',charset='utf8',database='pymysql')
```

For higher version of MySQL, *you need not provide `charset='utf8'` parameter*, this is a requirement for MySQL 6.0. [Figure 4.6](#) shows how this code will look in Python shell.

```
>>> import mysql.connector as msq
>>> pycon = msq.connect(host = 'localhost',user='pymysqladmin',passwd='pymysql1
23',charset='utf8',database='pymysql')
```

*Figure 4.6: Connecting to MySQL using mysql.connector*

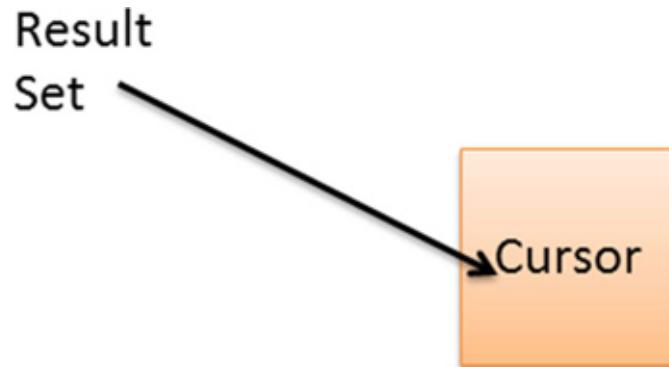
**Note: You can check whether the connection has been established with the MySQL database with the help of `is_connected()` function.**

```
>>> pycon.is_connected()
True
>>>
```

### Step 3: Creating cursor object

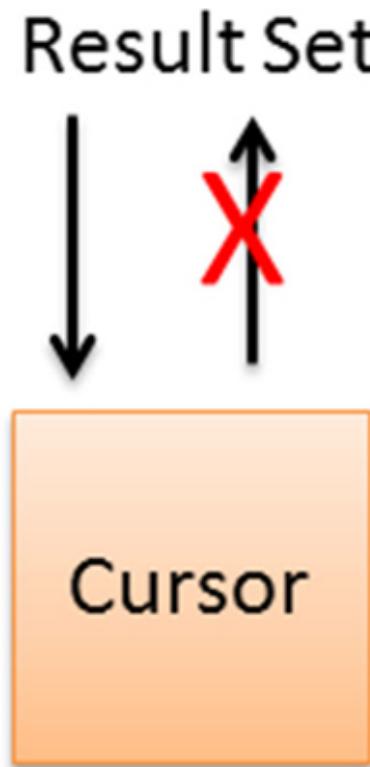
```
>>> mycursor = pycon.cursor()
```

Creating a cursor is very important because whenever you pass on a query to the database, it will return a **result set**. This result must be stored somewhere, and a **cursor** is used for this purpose.



*Figure 4.7: Storing Result set in Cursor*

However, there are some constraints imposed on this cursor. The most important one is that a cursor is **Read-only** in nature. This means that you cannot update a table in your database using cursor.



*Figure 4.8*

So, this is how one can assign the cursor to a variable.

```
mycursor = pycon.cursor()
```

mycursor

=

Result set

*Figure 4.9: Assigning cursor to a variable*

So, you now have a variable `mycursor`, which can be used to manage the cursor.

The cursor uses the `execute()` method to execute the SQL queries using Python.

It is important to close the cursor object, once you have completed the task. This can be done using `cursor.close() method()`.

#### Step 4: Create a table

We will now create a table by the name `book` with the following 3 fields:

1. `chapter_num` which is auto increment and the primary key.
2. The second field is the `chapter` (for the name of the chapter), it is a text field and is not null.
3. There is another not null field by the name `pages`.

The mysql statement for create table is:

```
CREATE TABLE book (chapter_num INT UNSIGNED PRIMARY KEY AUTO_INCREMENT, chapter TEXT
NOT NULL,pages INT NOT NULL)
```

In Python, this statement will be executed with the help of cursor as shown here:

```
mycursor.execute("CREATE TABLE book (chapter_num INT UNSIGNED PRIMARY KEY
AUTO_INCREMENT, chapter TEXT NOT NULL,pages INT NOT NULL)")
```

You can use your command prompt to check directly, if the table has been successfully created.

Login using the privileges:

```
mysql -u pymysqladmin -p
Enter Password: pymysql123
```

So, that you are able to work on the table give the following command:

```
use pymysql;
```

Now to check whether the table has been created type the following command in the command prompt:

```
show columns from book;
```

```
mysql> show columns from book;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
chapter_num	int<10> unsigned	NO	PRI	NULL	auto_increment
chapter	text	NO			
pages	int<11>	NO			
+-----+-----+-----+-----+-----+
3 rows in set (0.06 sec)
```

Figure 4.10

Now, as an exercise let's just add one record directly from command prompt using the MySQL query, and then try to retrieve that record using Python.

On your command prompt type:

```
insert into book values(1,'Introduction',12);
```

Now, check whether the record has been added to the table:

```
select * from book;
```

```
+-----+-----+-----+
| chapter_num | chapter | pages |
+-----+-----+-----+
| 1 | Introduction | 12 |
+-----+-----+
1 row in set (0.00 sec)
```

Figure 4.11

### 4.3.2 Retrieve records from database using Python

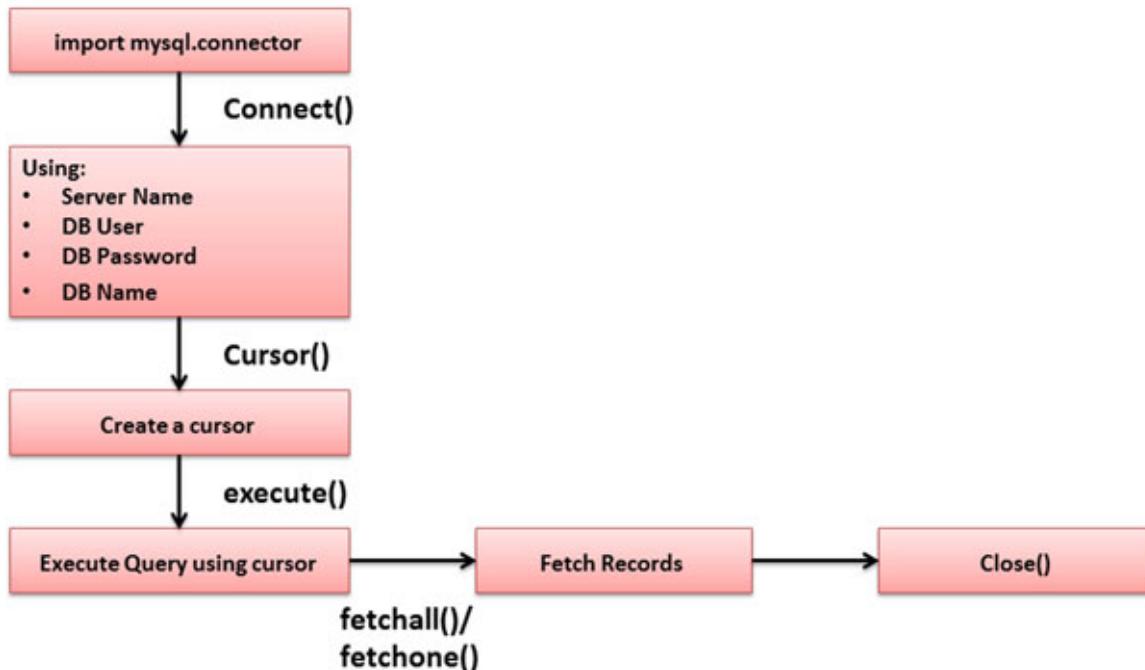
Now, we know that there is one record in the **book** table.

From your Python Idle you can check the records in the book table using the syntax as follows:

```
Syntax:
cursor_name.fetchall()
```

[Figure 4.12](#) displays how you can connect to MySQL server using Python.

## How to Connect to Remote MySQL Database Using Python



*Figure 4.12: Connecting to MySQL using Python*

Steps involved in retrieving the records are as follows:

### Step 1: Import the API Module

```
>>> import mysql.connector as msq
```

### Step 2: Acquire a connection with the database

```
>>> pycon = msq.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')
```

### **Step 3:** Use the connection object to create a cursor

```
>>> mycursor = pycon.cursor()
```

### **Step 4:** Execute the MySQL function using the SQL statements

```
>>> statement = 'select * from pymysql.book;'
>>>mycursor.execute(statement)
```

### **Step 5:** Fetch data

```
>>>mycursor.fetchall()
```

So, we can retrieve the records using **mycursor** as follows:

```
>>>import mysql.connector as msql
>>>pycon = msql.connect(host =
'localhost',user='pymysqladmin',passwd='pymysql123',charset='utf8',database='pymysql')
>>> mycursor = pycon.cursor()
>>> statement = 'select * from pymysql.book;'
>>> mycursor.execute(statement)
>>> mycursor.fetchall()
[(1, 'Introduction', 12)]
```

### **Step 6:** Closing `connection()`

Though this step is not followed in the preceding example, it is a mandatory step.

```
>>> mycursor.close()
```

We would be following it in the coming examples.

You can match the records with records obtained by MySQL directly.

Now, let's try to insert record through python shell.

Go to Python shell, and type the following commands:

```
>>> mycursor.execute("""INSERT INTO book(chapter, pages) VALUES ("Basics",
"15");""")
>>> pycon.commit()
>>> statement = 'select * from book;'
>>> mycursor.execute(statement)
>>> mycursor.fetchall()
[(1, 'Introduction', 12), (6, 'Basics', 15)]
```

So, you can see that the second record has been added. You can recheck directly using the command prompt.

```
mysql> select * from book;
+-----+-----+-----+
| chapter_num | chapter | pages |
+-----+-----+-----+
| 1 | Introduction | 12 |
| 6 | Basics | 15 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Figure 4.13

We have broadly explored how things work. Now, in the coming sections, you will learn about how basic MySQL queries are executed on the MySQL bash, and how the same queries are executed in Python.

## 4.4 Creating a database

To understand how to create a database using Python, it is important to know how database is actually created using direct MySQL queries this is explained in *Section 4.4.1. Section(Creating database directly using MySQL)and section 4.4.2(Creating database using Python)* explains how to create MySQL database using Python.

### 4.4.1 Creating database directly using MySQL

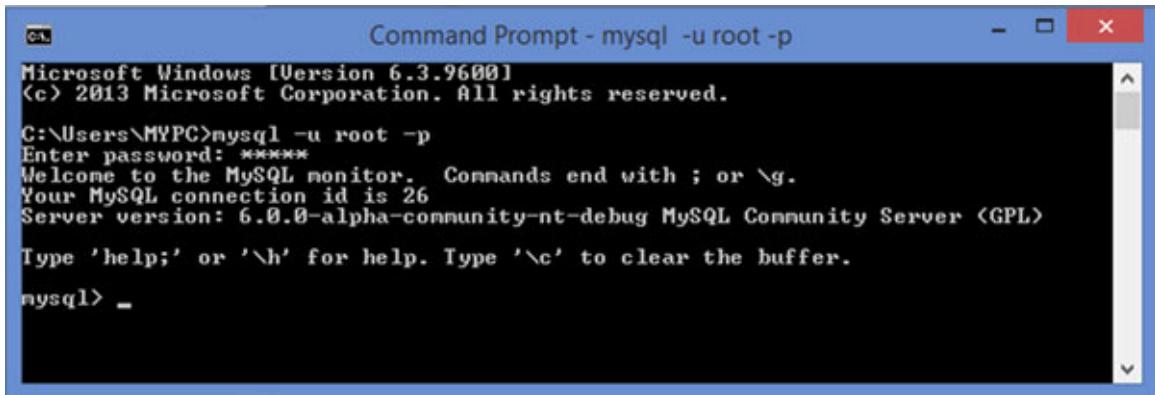
Follow the following steps to create a MySQL database using SQL:

**Step 1:** Type the following command in the command prompt

```
mysql -u root -p
```

**Step 2:** Enter Password

You will be prompted to enter the password for working with MySQL. You have set this password at the time of installation. In this case, it was “bpb12”.



```
Microsoft Windows [Version 6.3.9600]
(C) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 6.0.0-alpha-community-nt-debug MySQL Community Server <GPL>

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

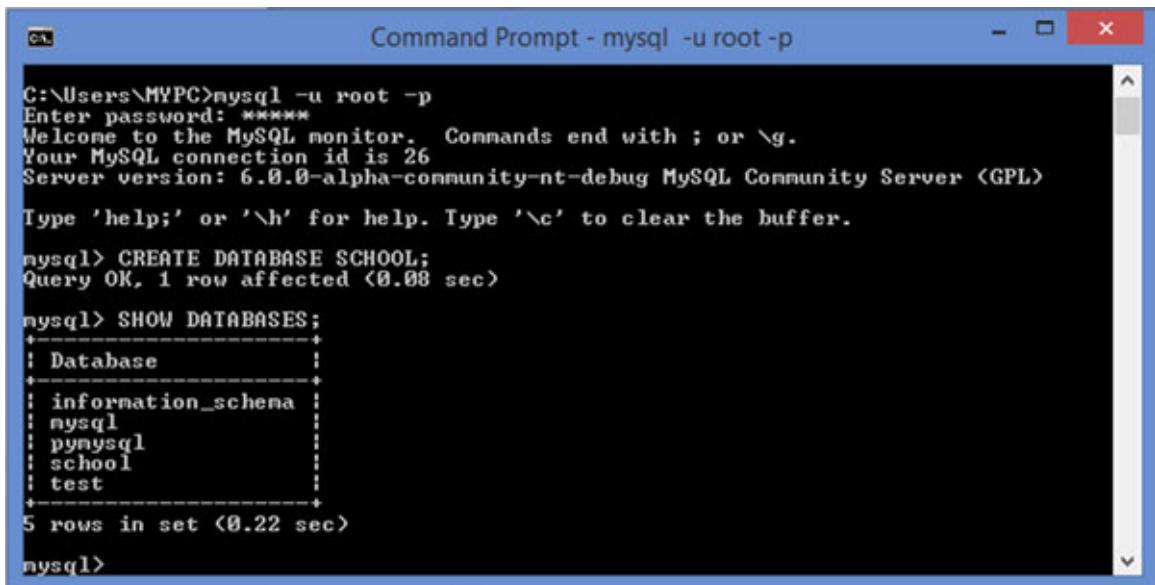
mysql> _
```

*Figure 4.14: Entering Password to connect to MySQL*

### Step 3: Provide the command to create a database

```
mysql> CREATE DATABASE SCHOOL;
```

Type the following command to see whether the database you just created is present in the list or not:



```
Microsoft Windows [Version 6.3.9600]
(C) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 6.0.0-alpha-community-nt-debug MySQL Community Server <GPL>

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE SCHOOL;
Query OK, 1 row affected (0.08 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| pynysql |
| school |
| test |
+-----+
5 rows in set (0.22 sec)

mysql>
```

*Figure 4.15: Use 'SHOW DATABASES' command to check if your database is showing*

Now, we will attempt to do the same through Python interface. In order to do that, let's first drop the database and check whether it has been deleted.

```
DROP DATABASE SCHOOL;
```

Now, type `SHOW DATABASES;` again to check whether it has been removed as shown in [figure 4.16](#).

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| pymysql |
| test |
+-----+
4 rows in set <0.00 sec>
```

*Figure 4.16: Database list after dropping database SCHOOL*

#### 4.4.2 Creating database using Python

In this section, you will learn to create the same database using Python. You are already familiar with the steps involved. You know that for creating a database by the name ‘SCHOOL’, you must give the `CREATE database SCHOOL;` command. We use `mycursor.execute()` to pass on the same command to the database.

```
import mysql.connector as msq
pycon = msq.connect(host = 'localhost',user='root',passwd='bpb12', charset='utf8')
mycursor = pycon.cursor()
statement = 'CREATE database SCHOOL;'
mycursor.execute(statement)
mycursor.execute("SHOW DATABASES;")
mycursor.fetchall()
mycursor.close()
```

[Figure 4.17](#) shows how this code will look on Python shell. Compare the output displayed in [Figure 4.17](#) with the output displayed in [Figure 4.18](#).

```
>>> import mysql.connector as msq
>>> pycon = msq.connect(host = 'localhost',user='root',passwd='bpb12',charset='utf8')
>>> mycursor = pycon.cursor()
>>> statement = 'CREATE database SCHOOL;'
>>> mycursor.execute(statement)
>>> mycursor.execute("SHOW DATABASES;")
>>> mycursor.fetchall()
[('information_schema',), ('mysql',), ('pymysql',), ('school',), ('test',)]
```

*Figure 4.17*

Now, execute the command `SHOW DATABASES;` for MySQL, and check the output. This is displayed in [Figure 4.18](#).

```

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| pymysql |
| school |
| test |
+-----+
5 rows in set <0.00 sec>

```

*Figure 4.18*

You can drop the database using Python in the same manner:

```

#drop the database
statement = 'DROP DATABASE SCHOOL;'
mycursor.execute(statement)
mycursor.execute("SHOW DATABASES;")
mycursor.fetchall()
mycursor.close()

```

*Figure 4.19* displays how you can check the information related to databases in Python shell as well as in MySQL Bash.

```

>>> statement = 'DROP DATABASE SCHOOL;'
>>> mycursor.execute(statement)
>>> mycursor.execute("SHOW DATABASES;")
>>> mycursor.fetchall()
[('information_schema',), ('mysql',), ('pymysql',), ('test',)]
>>> #Close Connection
>>> pycon.close()

```

## DROPPING A TABLE AND CHECKING THE OUTCOME USING PYTHON AND DIRECTLY AT BACKEND



```

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| pymysql |
| test |
+-----+
4 rows in set <0.00 sec>

```

*Figure 4.19: Comparison of output when checked using Python shell and MySQL Bash*

### Example 4.1:

Create a table – Book Library using Python

**Note:** We will now work on our pymysql database that we had initially created in this chapter. We created a table by the name *book* in it. Now, in the same database we will create another table by the name *book\_library*.

You may recall that after configuring MySQL we had created privileges for **pymysql** database. The username for the database was: **pymysqladmin** and the password: **pymysql123**.

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>mysql -u pymysqladmin -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 33
Server version: 6.0.0-alpha-community-nt-debug MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use pymysql;
Database changed
mysql>
```

*Figure 4.20*

Now, let's first see how we can create a table by the name **BOOK\_LIBRARY** directly on MySQL.

```
CREATE TABLE BOOK_LIBRARY(
 BOOK_ID SMALLINT NOT NULL AUTO_INCREMENT,
 BOOK_NAME CHAR(20) NOT NULL,
 AUTHOR_NAME CHAR(20) NOT NULL,
 PAGES SMALLINT NOT NULL,
 COST SMALLINT NOT NULL,
 PRIMARY KEY (BOOK_ID)
);
```

*Figure 4.21* shows how this would look at MySQL Bash.

```

mysql> CREATE TABLE BOOK_LIBRARY(
 -> BOOK_ID SMALLINT NOT NULL AUTO_INCREMENT,
 -> BOOK_NAME CHAR(20) NOT NULL,
 -> AUTHOR_NAME CHAR(20) NOT NULL,
 -> PAGES SMALLINT NOT NULL,
 -> COST SMALLINT NOT NULL,
 -> PRIMARY KEY (BOOK_ID)
 ->);
Query OK, 0 rows affected (0.27 sec)

```

*Figure 4.21*

Now, check the table. Type `SHOW COLUMNS FROM BOOK_LIBRARY`.

```

mysql> SHOW COLUMNS FROM BOOK_LIBRARY;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
BOOK_ID	smallint<6>	NO	PRI	NULL	auto_increment
BOOK_NAME	char(20)	NO			
AUTHOR_NAME	char(20)	NO			
PAGES	smallint<6>	NO			
COST	smallint<6>	NO			
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.09 sec)

```

*Figure 4.22*

We will now try to accomplish the same task using Python, but before that let's drop the table from the database ('`DROP TABLE BOOK_LIBRARY`').

```

mysql> DROP TABLE BOOK_LIBRARY;
Query OK, 0 rows affected (0.27 sec)

mysql> SHOW COLUMNS FROM BOOK_LIBRARY;
ERROR 1146 (42S02): Table 'pymysql.book_library' doesn't exist

```

*Figure 4.23*

## Creating a table using Python

The following code explains how the table can be created using Python:

```

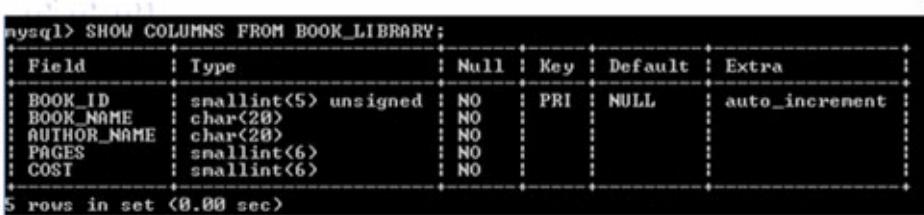
import mysql.connector as msq
pycon = msq.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """CREATE TABLE BOOK_LIBRARY(BOOK_ID SMALLINT UNSIGNED PRIMARY KEY
AUTO_INCREMENT, BOOK_NAME CHAR(20) NOT NULL,AUTHOR_NAME CHAR(20) NOT NULL, PAGES
SMALLINT NOT NULL,COST SMALLINT NOT NULL)"""
mycursor.execute(statement)
statement = 'SHOW COLUMNS FROM BOOK_LIBRARY;'
mycursor.execute(statement)
mycursor.fetchall()
pycon.close()

```

---

[Figure 4.24](#) shows the output on Python shell and the MySQL bash.

```
>>> import mysql.connector as msq
>>> pycon = msq.connect(host = 'localhost', user='pymysqladmin', passwd='pymysql23', charset='utf8', database='pymysql')
>>> mycursor = pycon.cursor()
>>> statement = """CREATE TABLE BOOK_LIBRARY(BOOK_ID SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT, BOOK_NAME CHAR(20) NOT NULL,AUTHOR_NAME CHAR(20) NOT NULL, PAGES SMALLINT NOT NULL,COST SMALLINT NOT NULL)"""
>>> mycursor.execute(statement)
>>>
>>> statement = 'SHOW COLUMNS FROM BOOK_LIBRARY;'
>>> mycursor.execute(statement)
>>> mycursor.fetchall()
[('BOOK_ID', 'smallint(5) unsigned', 'NO', 'PRI', None, 'auto_increment'), ('BOOK_NAME', 'char(20)', 'NO', '', '', ''), ('AUTHOR_NAME', 'char(20)', 'NO', '', '', ''), ('PAGES', 'smallint(6)', 'NO', '', '', ''), ('COST', 'smallint(6)', 'NO', '', '')]
>>> |
```

```
mysql> SHOW COLUMNS FROM BOOK_LIBRARY;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
BOOK_ID	smallint<5> unsigned	NO	PRI	NULL	auto_increment
BOOK_NAME	char<20>	NO			
AUTHOR_NAME	char<20>	NO			
PAGES	smallint<6>	NO			
COST	smallint<6>	NO			
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

*Figure 4.24*

We will now work on this table for inserting and updating records.

## 4.5 Working with database using Python

In this section, you will learn how to use Python to:

1. Insert records in database
2. Select records
3. Fetch selected records using **WHERE** clause
4. Delete records from database and
5. Update records in database

### 4.5.1 Inserting records

Records can be inserted into a table using the **INSERT** statement. [Figure 4.25](#) explains how **INSERT** is used to insert statements directly in the MySQL database.

#### **MySQL Insert statement:**

```
INSERT INTO BOOK_LIBRARY(BOOK_NAME,AUTHOR_NAME,PAGES,COST) VALUES ('The
```

```
Secret', 'Rhonda Byrne', 198, 3);
```

```
mysql> INSERT INTO BOOK_LIBRARY(BOOK_NAME,AUTHOR_NAME,PAGES,COST) VALUES ('The Secret','Rhonda Byrne',198,3);
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM BOOK_LIBRARY;
+-----+-----+-----+-----+
| BOOK_ID | BOOK_NAME | AUTHOR_NAME | PAGES | COST |
+-----+-----+-----+-----+
| 1 | The Secret | Rhonda Byrne | 198 | 3 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Figure 4.25

[Figure 4.26](#) displays how the records can be inserted in a database using Python.

## Insert records using Python

```
>>> import mysql.connector as msq
>>> pycon = msq.connect(host = 'localhost',user='pymysqadmin',passwd='pymysq
23',charset='utf8',database='pymysql')
>>> mycursor = pycon.cursor()
>>> statement = """INSERT INTO BOOK_LIBRARY(BOOK_NAME,AUTHOR_NAME,PAGES,COST) VA
LUES ('The Power','Rhonda Byrne',272,6);"""
>>> mycursor.execute(statement)
>>> pycon.commit()
```

Figure 4.26

**Note:** pycon.commit() reflects changes in the database.

It is time to check results [figure 4.27](#) displays how it should look in Python shell, and MySQL Bash.

```
>>> mycursor.execute(statement)
>>> mycursor.fetchall()
[(1, 'The Secret', 'Rhonda Byrne', 198, 3), (3, 'The Power', 'Rhonda Byrne', 272
, 6)]
```

```
mysql> SELECT * FROM BOOK_LIBRARY;
+-----+-----+-----+-----+
| BOOK_ID | BOOK_NAME | AUTHOR_NAME | PAGES | COST |
+-----+-----+-----+-----+
| 1 | The Secret | Rhonda Byrne | 198 | 3 |
| 3 | The Power | Rhonda Byrne | 272 | 6 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
S lone to set (0.00 sec)
```

Figure 4.27

## Example 4.2

## Inserting values dynamically in the table **BOOK\_LIBRARY**

### Code:

```
import mysql.connector as msq
data = []
pycon = msq.connect(host = 'localhost', user='pymysqldadmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """INSERT INTO BOOK_LIBRARY(BOOK_NAME, AUTHOR_NAME, PAGES, COST) VALUES (%s, %s, %s, %s);"""
#Get Name of the Book
val1 = input("Enter Book Name : ")

#Append value to list
data.append(val1)

#Get Name of the Author
val2 = input("Enter Author Name : ")

#Append value to list
data.append(val2)

#Get value of number of pages and convert to int
val3 = int(input("Enter Number of Pages: "))

#Append value to list
data.append(val3)

#Get value of cost and convert to int
val4 = int(input("Enter Cost in $: "))

#Append value to list
data.append(val4)

#Convert list to tuple
data2 = tuple(data)

print(data2)
mycursor.execute(statement, data2)
pycon.commit()
pycon.close()
```

### Output:

```
Enter Book Name : The Magic
Enter Author Name : Rhonda Byrne
Enter Number of Pages: 272
Enter Cost in $: 6
('The Magic', 'Rhonda Byrne', 272, 6)
```

You must now check whether the new record is showing in the database.

```

mysql> SELECT * FROM BOOK_LIBRARY;
+-----+-----+-----+-----+-----+
| BOOK_ID | BOOK_NAME | AUTHOR_NAME | PAGES | COST |
+-----+-----+-----+-----+-----+
1	The Secret	Rhonda Byrne	198	3
3	The Power	Rhonda Byrne	272	6
4	The Magic	Rhonda Byrne	272	6
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

*Figure 4.28*

**Note:** The preceding code can also be written using the `format()` function instead of using % formatting. You can see what suits you more.

### Code:

```

import mysql.connector as msql
pycon = msql.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """INSERT INTO BOOK_LIBRARY(BOOK_NAME,AUTHOR_NAME,PAGES,COST) VALUES
('{0}', '{1}', {2}, {3});"""
val1 = input("Enter Book Name : ")
val2 = input("Enter Author Name : ")
val3 = int(input("Enter Number of Pages: "))
val4 = int(input("Enter Cost in $: "))
mycursor.execute(statement.format(val1, val2, val3, val4))
pycon.commit()
pycon.close()

```

### Output:

```

Enter Book Name : The Hero
Enter Author Name : Rhonda Byrne
Enter Number of Pages: 240
Enter Cost in $: 6

```

```

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>mysql -u pymysqadmin -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 6.0.0-alpha-community-nt-debug MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use pymysql;
Database changed
mysql> SELECT * FROM BOOK_LIBRARY;
+---+---+---+---+---+
| BOOK_ID | BOOK_NAME | AUTHOR_NAME | PAGES | COST |
+---+---+---+---+---+
1	The Secret	Rhonda Byrne	198	3
3	The Power	Rhonda Byrne	272	6
4	The Magic	Rhonda Byrne	272	6
5	The Hero	Rhonda Byrne	240	6
+---+---+---+---+---+
4 rows in set (0.00 sec)

mysql>

```

*Figure 4.29*

Sometimes, while working with a database, you may require to add multiple records together. For that, you can use `executemany`. Let's see how this is done. Suppose there is a requirement for adding following 5 records in the `book_library` table:

| BOOK_NAME            | AUTHOR_NAME      | PAGES | COST |
|----------------------|------------------|-------|------|
| I AM Magic           | Maria Robins     | 63    | 13   |
| One Truth, One Law   | Erin Werley      | 87    | 6    |
| Mind Magic           | Merlin Starlight | 265   | 19   |
| Advanced Manifesting | Linda West       | 172   | 4    |
| The Frequency        | Linda West       | 180   | 4    |

*Table 4.1*

**Step 1:** Put each record in brackets

```

('I AM Magic','Maria Robins',63,13)
('One Truth, One Law','Erin Werley',87,6)
('Mind Magic ','Merlin Starlight',265,19)
('Advanced Manifesting','Linda West',172,4)
('The Frequency','Linda West',180,4)

```

---

## Step 2: Prepare a list of these records

```
values =[('I AM Magic','Maria Robins',63,13), ('One Truth, One Law','Erin Werley',87,6), ('Mind Magic for Beginners','Merlin Starlight',265,19), ('Advanced Manifesting','Linda West',172,4), ('The Frequency','Linda West',180,4)]
```

## Step 3: Execute the code

We will now use this list to add values as shown in following code:

```
import mysql.connector as msq
pycon = msq.connect(host =
'localhost',user='pymysqladmin',passwd='pymysql123',charset='utf8',database='pymysql')
mycursor = pycon.cursor()
statement = """INSERT INTO BOOK_LIBRARY(BOOK_NAME,AUTHOR_NAME,PAGES,COST) VALUES
(%s,%s,%s,%s);"""
values =[('I AM Magic','Maria Robins',63,13), ('One Truth, One Law','Erin Werley',87,6), ('Mind Magic','Merlin Starlight',265,19), ('Advanced Manifesting','Linda West',172,4), ('The Frequency','Linda West',180,4)]
mycursor.executemany(statement,values)
pycon.commit()
pycon.close()
print("Records Inserted!!")
```

## Output:

```
Records Inserted!!
```

```
mysql> SELECT * FROM BOOK_LIBRARY;
+----+-----+-----+-----+-----+
| BOOK_ID | BOOK_NAME | AUTHOR_NAME | PAGES | COST |
+----+-----+-----+-----+-----+
1	The Secret	Rhonda Byrne	198	3
3	The Power	Rhonda Byrne	272	6
4	The Magic	Rhonda Byrne	272	6
5	The Hero	Rhonda Byrne	240	6
8	I AM Magic	Maria Robins	63	13
9	One Truth, One Law	Erin Werley	87	6
10	Mind Magic	Merlin Starlight	265	19
11	Advanced Manifesting	Linda West	172	4
12	The Frequency	Linda West	180	4
+----+-----+-----+-----+-----+
9 rows in set (0.00 sec)
```

Figure 4.30

## 4.5.2 Select records

Let's now take a look at how to fetch records from a table with Python.

### Fetching all records from a table

Look at the illustration given above. The `SELECT * FROM BOOK_LIBRARY ;` query displayed all the records present in the `BOOK_LIBRARY TABLE`. In the next section, we will retrieve all records using Python.

#### 4.5.2.1 Fetching all records from a table in Python using `fetchall()`

The following code illustrates how to retrieve all records from a table.

##### **Code:**

```
import mysql.connector as msq
pycon = msq.connect(host =
'localhost',user='pymysqladmin',passwd='pymysql123',charset='utf8',database='pymysql')
mycursor = pycon.cursor()
statement = """SELECT * FROM BOOK_LIBRARY;"""
mycursor.execute(statement)
result_set = mycursor.fetchall()
print("The Results are as follows: ")
for result in result_set:
 print(result)
pycon.close()
print("Done!!")
```

##### **Output:**

```
The Results are as follows:
(1, 'The Secret', 'Rhonda Byrne', 198, 3)
(3, 'The Power', 'Rhonda Byrne', 272, 6)
(4, 'The Magic', 'Rhonda Byrne', 272, 6)
(5, 'The Hero', 'Rhonda Byrne', 240, 6)
(8, 'I AM Magic', 'Maria Robins', 63, 13)
(9, 'One Truth, One Law', 'Erin Werley', 87, 6)
(10, 'Mind Magic', 'Merlin Starlight', 265, 19)
(11, 'Advanced Manifesting', 'Linda West', 172, 4)
(12, 'The Frequency', 'Linda West', 180, 4)
Done!!
```

*Figure 4.31*

#### 4.5.2.2 Fetching one record from a table in Python using `fetchone()`

In python, you can use `fetchone()` function to retrieve only the first record of the table.

### Code:

```
import mysql.connector as msq
pycon = msq.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """SELECT * FROM BOOK_LIBRARY;"""
mycursor.execute(statement)
result_set = mycursor.fetchone()
print(result_set)
pycon.close()
print("Done!!")
```

### Output:

```
(1, 'The Secret', 'Rhonda Byrne', 198, 3)
Done!!
```

### Example 4.3

Retrieving selected columns from a table.

[Figure 4.32](#) displays how selected records can be retrieved directly from MySQL using SQL.

```
mysql> SELECT BOOK_NAME, AUTHOR_NAME FROM BOOK_LIBRARY;
+-----+-----+
| BOOK_NAME | AUTHOR_NAME |
+-----+-----+
The Secret	Rhonda Byrne
The Power	Rhonda Byrne
The Magic	Rhonda Byrne
The Hero	Rhonda Byrne
I AM Magic	Maria Robins
One Truth, One Law	Erin Werley
Mind Magic	Merlin Starlight
Advanced Manifesting	Linda West
The Frequency	Linda West
+-----+-----+
9 rows in set (0.00 sec)
```

*Figure 4.32*

Retrieving columns using Python

```
import mysql.connector as msq
```

```
pycon = mysql.connect(host =
'localhost',user='pymysqladmin',passwd='pymysql123',charset='utf8',database='pymysql')
mycursor = pycon.cursor()
statement = """SELECT BOOK_NAME, AUTHOR_NAME FROM BOOK_LIBRARY;"""
mycursor.execute(statement)
result_set = mycursor.fetchall()
print("The Results are as follows: ")
for result in result_set:
 print(result)
pycon.close()
print("Done!!")
```

## Output:

```
The Results are as follows:
('The Secret', 'Rhonda Byrne')
('The Power', 'Rhonda Byrne')
('The Magic', 'Rhonda Byrne')
('The Hero', 'Rhonda Byrne')
('I AM Magic', 'Maria Robins')
('One Truth, One Law', 'Erin Werley')
('Mind Magic', 'Merlin Starlight')
('Advanced Manifesting', 'Linda West')
('The Frequency', 'Linda West')
Done!!
```

*Figure 4.33*

Retrieving selected columns of first row using **fetchone()**

## Code:

```
import mysql.connector as mysql
pycon = mysql.connect(host =
'localhost',user='pymysqladmin',passwd='pymysql123',charset='utf8',database='pymysql')
mycursor = pycon.cursor()
statement = """SELECT BOOK_NAME, AUTHOR_NAME FROM BOOK_LIBRARY;"""
mycursor.execute(statement)
result_set = mycursor.fetchone()
print(result_set)
pycon.close()
print("Done!!")
```

---

## Output:

```
('The Secret', 'Rhonda Byrne')
Done!!
```

### 4.5.3 Fetching selected records from a table in Python using the ‘WHERE’ clause

The **WHERE** clause is used to filter records on the basis of a condition. Let's say we want to know all information of books having cost greater or equal to \$7. We would give the following command:

```
SELECT * FROM BOOK_LIBRARY WHERE COST >= 7;
```

```
mysql> SELECT * FROM BOOK_LIBRARY WHERE COST>=7;
+---+---+---+---+
| BOOK_ID | BOOK_NAME | AUTHOR_NAME | PAGES | COST |
+---+---+---+---+
| 8 | I AM Magic | Maria Robins | 63 | 13 |
| 10 | Mind Magic | Merlin Starlight | 265 | 19 |
+---+---+---+---+
2 rows in set (0.04 sec)
```

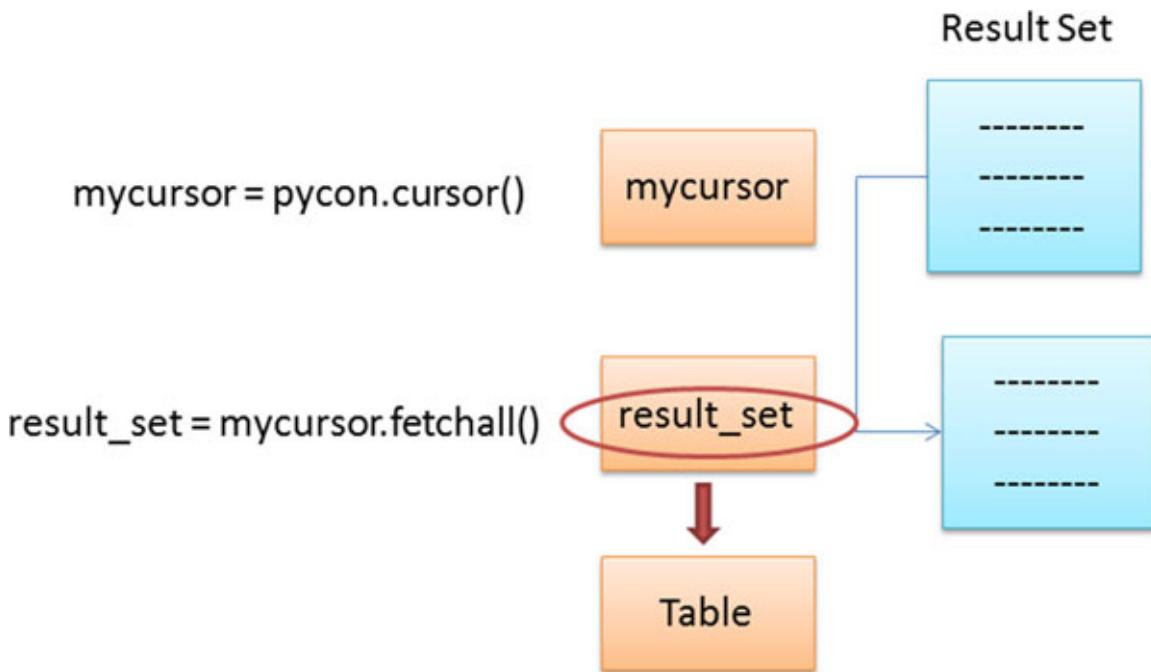
Figure 4.34

### Using ‘WHERE’ clause with Python

The following code displays how to used **WHERE** clause using Python:

```
import mysql.connector as msq
pycon = msq.connect(host =
'localhost', user='pymysqladmin',passwd='pymysql123',charset='utf8',database='pymysql')
mycursor = pycon.cursor()
statement = """SELECT * FROM BOOK_LIBRARY WHERE COST>=7;"""
mycursor.execute(statement)
result_set = mycursor.fetchall()
for result in result_set:
 print(result)
pycon.close()
print("Done!!")
```

To display the data retrieved from the database, **mycursor** is used to fetch all the records out of the cursor and these records are assigned to another variable by the name **result\_set**. You can then iterate through **result\_set** to display data or to do any other work or to update underlying data.



*Figure 4.35*

## Output:

```

(8, 'I AM Magic', 'Maria Robins', 63, 13)
(10, 'Mind Magic', 'Merlin Starlight', 265, 19)
Done!!

```

## Example 4.4

Select all records from `BOOK_LIBRARY` table where the author name has “Rhon” in it.

```

mysql> SELECT * FROM BOOK_LIBRARY WHERE AUTHOR_NAME LIKE '%Rhon%';
+-----+-----+-----+-----+-----+
| BOOK_ID | BOOK_NAME | AUTHOR_NAME | PAGES | COST |
+-----+-----+-----+-----+-----+
1	The Secret	Rhonda Byrne	198	3
3	The Power	Rhonda Byrne	272	6
4	The Magic	Rhonda Byrne	272	6
5	The Hero	Rhonda Byrne	240	6
+-----+-----+-----+-----+-----+
4 rows in set <0.00 sec>

```

*Figure 4.36*

## Code:

```

import mysql.connector as msq
pycon = msq.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')

```

```

mycursor = pycon.cursor()
statement = """SELECT * FROM BOOK_LIBRARY WHERE AUTHOR_NAME LIKE '%Rhon%'"""
mycursor.execute(statement)
result_set = mycursor.fetchall()
for result in result_set:
 print(result)
pycon.close()
print("Done!!")

```

## Output:

```

(1, 'The Secret', 'Rhonda Byrne', 198, 3)
(3, 'The Power', 'Rhonda Byrne', 272, 6)
(4, 'The Magic', 'Rhonda Byrne', 272, 6)
(5, 'The Hero', 'Rhonda Byrne', 240, 6)
Done!!

```

Escape query values by using the placeholder %s method:

## Code:

```

import mysql.connector as msq
pycon = msq.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """SELECT * FROM BOOK_LIBRARY WHERE AUTHOR_NAME = %s;"""
author = ("Rhonda Byrne",)
mycursor.execute(statement, author)
result_set = mycursor.fetchall()
for result in result_set:
 print(result)
pycon.close()
print("Done!!")

```

## Output:

```

(1, 'The Secret', 'Rhonda Byrne', 198, 3)
(3, 'The Power', 'Rhonda Byrne', 272, 6)
(4, 'The Magic', 'Rhonda Byrne', 272, 6)
(5, 'The Hero', 'Rhonda Byrne', 240, 6)
Done!!

```

## 4.5.4 Using ORDER BY clause

| mysql> SELECT * FROM BOOK_LIBRARY ORDER BY BOOK_NAME; |                      |                  |       |      |  |
|-------------------------------------------------------|----------------------|------------------|-------|------|--|
| BOOK_ID                                               | BOOK_NAME            | AUTHOR_NAME      | PAGES | COST |  |
| 11                                                    | Advanced Manifesting | Linda West       | 172   | 4    |  |
| 8                                                     | I AM Magic           | Maria Robins     | 63    | 13   |  |
| 10                                                    | Mind Magic           | Merlin Starlight | 265   | 19   |  |
| 9                                                     | One Truth, One Law   | Erin Werley      | 87    | 6    |  |
| 12                                                    | The Frequency        | Linda West       | 180   | 4    |  |
| 5                                                     | The Hero             | Rhonda Byrne     | 240   | 6    |  |
| 4                                                     | The Magic            | Rhonda Byrne     | 272   | 6    |  |
| 3                                                     | The Power            | Rhonda Byrne     | 272   | 6    |  |
| 1                                                     | The Secret           | Rhonda Byrne     | 198   | 3    |  |

9 rows in set (0.00 sec)

Figure 4.37

## Code:

```
import mysql.connector as msq
pycon = msq.connect(host =
'localhost', user='pymysqldadmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """SELECT * FROM BOOK_LIBRARY WHERE AUTHOR_NAME = %s;"""
author = ("Rhonda Byrne",)
mycursor.execute(statement, author)
result_set = mycursor.fetchall()
for result in result_set:
 print(result)
pycon.close()
print("Done!!")
```

## Output:

```
(11, 'Advanced Manifesting', 'Linda West', 172, 4)
(8, 'I AM Magic', 'Maria Robins', 63, 13)
(10, 'Mind Magic', 'Merlin Starlight', 265, 19)
(9, 'One Truth, One Law', 'Erin Werley', 87, 6)
(12, 'The Frequency', 'Linda West', 180, 4)
(5, 'The Hero', 'Rhonda Byrne', 240, 6)
(4, 'The Magic', 'Rhonda Byrne', 272, 6)
(3, 'The Power', 'Rhonda Byrne', 272, 6)
(1, 'The Secret', 'Rhonda Byrne', 198, 3)
Done!!
```

## 4.5.5 Deleting records using DELETE command

You have learnt how to insert and retrieve records from the database. In this section, you will learn how to delete records from the database.

```
import mysql.connector as msq
```

```

pycon = mysql.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """DELETE FROM BOOK_LIBRARY WHERE BOOK_ID = 8"""
mycursor.execute(statement)
pycon.commit()
pycon.close()
print("Done!!")

```

## Output:

| mysql> SELECT * FROM BOOK_LIBRARY; |                      |                  |       |      |
|------------------------------------|----------------------|------------------|-------|------|
| BOOK_ID                            | BOOK_NAME            | AUTHOR_NAME      | PAGES | COST |
| 1                                  | The Secret           | Rhonda Byrne     | 198   | 3    |
| 3                                  | The Power            | Rhonda Byrne     | 272   | 6    |
| 4                                  | The Magic            | Rhonda Byrne     | 272   | 6    |
| 5                                  | The Hero             | Rhonda Byrne     | 240   | 6    |
| 9                                  | One Truth, One Law   | Erin Werley      | 87    | 6    |
| 10                                 | Mind Magic           | Merlin Starlight | 265   | 19   |
| 11                                 | Advanced Manifesting | Linda West       | 172   | 4    |
| 12                                 | The Frequency        | Linda West       | 180   | 4    |

8 rows in set <0.00 sec>

Figure 4.38

## 4.5.6 Update clause

### Code:

```

import mysql.connector as mysql
pycon = mysql.connect(host =
'localhost', user='pymysqladmin', passwd='pymysql123', charset='utf8', database='pymysql')
mycursor = pycon.cursor()
statement = """UPDATE BOOK_LIBRARY SET BOOK_ID = 2 WHERE BOOK_ID = 12;"""
mycursor.execute(statement)
pycon.commit()
pycon.close()
print("Done!!")

```

## Output:

| mysql> SELECT * FROM BOOK_LIBRARY; |                      |                  |       |      |
|------------------------------------|----------------------|------------------|-------|------|
| BOOK_ID                            | BOOK_NAME            | AUTHOR_NAME      | PAGES | COST |
| 1                                  | The Secret           | Rhonda Byrne     | 198   | 3    |
| 2                                  | The Frequency        | Linda West       | 180   | 4    |
| 3                                  | The Power            | Rhonda Byrne     | 272   | 6    |
| 4                                  | The Magic            | Rhonda Byrne     | 272   | 6    |
| 5                                  | The Hero             | Rhonda Byrne     | 240   | 6    |
| 9                                  | One Truth, One Law   | Erin Werley      | 87    | 6    |
| 10                                 | Mind Magic           | Merlin Starlight | 265   | 19   |
| 11                                 | Advanced Manifesting | Linda West       | 172   | 4    |

8 rows in set <0.00 sec>

*Figure 4.39*

## Conclusion

In this chapter, you learnt how Python can be used to interact with the database. You can execute any kind of SQL query with the help of Python. You will go on step further in [chapter 9 MySQL and Python Graphical User Interface](#) where you will learn how to use GUI to interact with the database. Interaction with database with the help of Python is very easy as it involves standard steps. Once you understand how to implement these steps, you can easily code any type of application that interacts with the database.

## Questions and answers

1. Write the Python code create a database having the name TEXTILE.

**Answer:** The following code shows how a database name ‘TEXTILE’ can be created using Python.

```
>>> import mysql.connector as msq
>>>pycon = msq.connect(host = 'localhost',user='root',passwd=
'bpb12',charset='utf8')
>>>mycursor = pycon.cursor()
>>> statement = 'CREATE database TEXTILE;'
>>>mycursor.execute(statement)
>>>pycon.commit()
>>>pycon.close()
```

You can verify results in the database.

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| pymysql |
| test |
| textile |
+-----+
```

```
5 rows in set (0.05 sec)
```

*Figure 4.40*

2. Given a database by the name textile has user id – shopkeeper, password – shoptoday. Write the python code to connect to the textile database.

```
mysql> GRANT ALL PRIVILEGES ON TEXTILE.* TO 'shopkeeper'@'localhost' IDENTIFIED
BY 'shoptoday';
Query OK, 0 rows affected (0.34 sec)

mysql> quit
Bye

C:\Users\MYPC>mysql -u shopkeeper -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 36
Server version: 6.0.0-alpha-community-nt-debug MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use textile;
Database changed
mysql> _
```

*Figure 4.41*

**Answer:**

```
import mysql.connector as msq
pycon = msq.connect(host =
'localhost',user='shopkeeper',passwd='shoptoday',database='textile')
```

3. A company has created a new database to keep a track of its employees. The following table shows the employee records. A new employee Karan has joined the firm as a project manager. His employee number is 7980,

salary of 10 Lacs. Insert his record in the table taking system date as the hire date. Explain the steps involved.

| EmpNo | EmpName | Job                | Hiredate   | Sal     |
|-------|---------|--------------------|------------|---------|
| 7110  | Rai     | Director           | 2013-08-11 | 2500000 |
| 7123  | Rishab  | Software Developer | 2020-01-01 | 500000  |
| 7128  | Lee     | Software Developer | 2018-03-01 | 500000  |
| 7801  | Ray     | Temp               | 2020-07-01 | 60000   |

Figure 4.42

### Answer:

```
import mysql.connector as msq
from datetime import datetime
pycon = msq.connect(host = 'localhost',user='root',passwd='password',charset='utf8',database = 'database_name')
mycursor = pycon.cursor()
now = datetime.now()
formatted_date = now.strftime('%Y-%m-%d')
statement = "INSERT INTO emp VALUES(7980,'Karan','Project Manager','{}',1000000);".format(formatted_date)
mycursor.execute(statement)
pycon.commit()
pycon.close()
```

Verify results

| EmpNo | EmpName | Job                | Hiredate   | Sal     |
|-------|---------|--------------------|------------|---------|
| 7110  | Rai     | Director           | 2013-08-11 | 2500000 |
| 7123  | Rishab  | Software Developer | 2020-01-01 | 500000  |
| 7128  | Lee     | Software Developer | 2018-03-01 | 500000  |
| 7801  | Ray     | Temp               | 2020-07-01 | 60000   |
| 7980  | Karan   | Project Manager    | 2020-07-15 | 1000000 |

5 rows in set (0.00 sec)

Figure 4.43

4. In view of lockdown, the company decides to reduce the salary of all its employees by 10000. How will you update the table? Work on the `emp` table from the previous example.

### Answer:

```
import mysql.connector as msq
pycon = msq.connect(host = 'localhost',user='root',passwd='password',charset='utf8',database = 'database_name')
```

```

mycursor = pycon.cursor()
statement = 'UPDATE emp SET Sal = Sal - 10000;'
mycursor.execute(statement)
pycon.commit()
pycon.close()

```

## Output:

| mysql> Select * from emp; |         |                    |            |         |  |
|---------------------------|---------|--------------------|------------|---------|--|
| EmpNo                     | EmpName | Job                | Hiredate   | Sal     |  |
| 7110                      | Rai     | Director           | 2013-08-11 | 2490000 |  |
| 7123                      | Rishab  | Software Developer | 2020-01-01 | 490000  |  |
| 7128                      | Lee     | Software Developer | 2018-03-01 | 490000  |  |
| 7801                      | Ray     | Temp               | 2020-07-01 | 50000   |  |
| 7980                      | Karan   | Project Manager    | 2020-07-15 | 990000  |  |

Figure 4.44

5. Modern fashion sells various types of fashionable clothes for men and women on all leading online stores. In view of COVID-19 pandemic, the store decides to give 50% discount on women clothing, and 25% discount on men clothing. Explain step by step how you would reflect these changes on MySQL database using the Python interface.

| mysql> select * from modernFashions; |           |         |      |  |
|--------------------------------------|-----------|---------|------|--|
| item_num                             | item_name | for_who | cost |  |
| 101                                  | maxi      | women   | 2500 |  |
| 102                                  | jackets   | women   | 4500 |  |
| 103                                  | jackets   | men     | 6000 |  |
| 104                                  | patiala   | women   | 900  |  |
| 105                                  | pyjamas   | men     | 999  |  |
| 106                                  | scarfs    | women   | 3900 |  |
| 107                                  | skirts    | women   | 5900 |  |
| 109                                  | Jeans     | men     | 3999 |  |

8 rows in set (0.00 sec)

Figure 4.45

## Answer:

```

import mysql.connector as msq

```

```

pycon = mysql.connect(host = 'localhost', user='root', passwd='password',
 charset='utf8', database = 'database_name')
mycursor = pycon.cursor()
statement = "UPDATE modernFashions SET cost = cost - 0.5*cost where for_who = '{}';".format('women')
mycursor.execute(statement)
pycon.commit()
pycon.close()

```

### Output:

```

mysql> select * from modernFashions;
+-----+-----+-----+-----+
| item_num | item_name | for_who | cost |
+-----+-----+-----+-----+
101	maxi	women	2500
102	jackets	women	4500
103	jackets	men	6000
104	patiala	women	900
105	pyjamas	men	999
106	scarfs	women	3900
107	skirts	women	5900
109	Jeans	men	3999
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

*Figure 4.46*

6. For the `emp` table shown as follows:

```

+-----+-----+-----+-----+
| EmpNo | EmpName | Job | Hiredate | Sal |
+-----+-----+-----+-----+
7110	Rai	Director	2013-08-11	2540000
7123	Rishab	Software Developer	2020-01-01	540000
7128	Lee	Software Developer	2018-03-01	540000
7801	Ray	Temp	2020-07-01	100000
7980	Karan	Project Manager	2020-07-15	1040000
+-----+-----+-----+-----+
5 rows in set (0.02 sec)

```

*Figure 4.47*

Increment the salary of all those employees who have been working in the firm before the year 2015 started by 50000.

### Answer:

```

import mysql.connector as mysql

```

```

pycon = mysql.connect(host = 'localhost', user='root', passwd='password',
 charset='utf8', database = 'database_name')
mycursor = pycon.cursor()
statement = "UPDATE emp SET sal = sal + 50000 where hiredate<= '2015-01-01';".format('2015-01-01')
mycursor.execute(statement)
pycon.commit()
pycon.close()

```

## Output:

| EmpNo | EmpName | Job                | Hiredate   | Sal     |
|-------|---------|--------------------|------------|---------|
| 7110  | Rai     | Director           | 2013-08-11 | 2590000 |
| 7123  | Rishab  | Software Developer | 2020-01-01 | 540000  |
| 7128  | Lee     | Software Developer | 2018-03-01 | 540000  |
| 7801  | Ray     | Temp               | 2020-07-01 | 100000  |
| 7980  | Karan   | Project Manager    | 2020-07-15 | 1040000 |

Figure 4.48

7. Define the following Database Object methods:

- a. close()
- b. cursor()
- c. rollback()

## Answer:

- a. The **close()** method is used to close the cursor.
- b. The **cursor()** method is used to return a database cursor object that can be used to execute the SQL queries.
- c. The **rollback()** method is used to roll back pending transaction to the state that existed before the transaction began.

# CHAPTER 5

## Python Threads

### Introduction

**Process** is a very common term for software developers. Execution of any program on your system is a process. A **thread** is part of a process or a sub-process. It is a basic unit that is allocated processor time. It lives in a process, and shares the same memory space as the process. A process can have multiple threads. In this chapter, you will learn how to work with threads in Python.

### Structure

- Processes and threads
  - Process
  - Threads
- How to create a thread
  - Implementation of new thread using Threading Module.
- Thread Synchronization with LOCK and RLOCK
- Applying lock
- Deadlock
- Semaphore
- Thread Synchronization using an event object
- Condition class
- Daemon and Non Daemon thread

### Objectives

After reading this chapter, you will:

- Be able to differentiate between a process and a thread
- Learn to create a thread
- Learn about thread synchronization
- Learn about LOCK, RLOCK, Deadlock, Semphore
- Understand the difference between daemon and non-daemon thread

You have already learned about features of Python language that makes it so popular in the world of software development. Let's just take a moment to revise some of the most important features of Python:

1. Python has clear and readable syntax
2. It has an extensive standard library
3. Easy to learn language
4. Allows rapid development and easy debugging
5. Provided exception-based error handling
6. Very well documented
7. Has a great community

## 5.1 Processes and threads

The word **Process** is very common in computer programming. In this section, you will learn what is a process and how it is different from a thread.

### 5.1.1 Process

A **process** is created by the operating system, and can be defined as an executing instance of a program or an application. You can start a process by double-clicking on any program or application on your system, such as the Word document or a browser, and so on. Every application starts its own process that has its own data stack, space address, and other auxiliary data that allows it to track its own execution. More than one process can execute code at the same time within the same python program.

The execution of the processes is managed by the operating system. It schedules processes to allow access to the computational resources of the system.

## 5.1.2 Threads

A **thread** is a lightweight, active flow of control that can be activated in parallel to other threads, and can execute concurrently with them within the same process. Multiple threads run within a process where each thread is an independent program running alongside others.

A thread is a means by which a program can fragment itself into two or more concurrently running tasks. Every thread executes the tasks independently parallel to other threads within the same process. A thread shares space addressing, and data structures (that is, threads can read and write to the same variables) with other threads of the same process. It is a group of statements within a program that can be executed without being dependent on the rest of the code.

Threading allows the execution of multiple threads such as tasks, function calls, and so on, at the same time, threading is not parallel programming. As a matter of fact, it is used when the execution of a task comprises some sort of waiting. For example, while interacting with some other machine on a web server may require some waiting before establishing a connection. This waiting time can be utilized to execute some other code. Within the same Python program, two threads cannot execute code simultaneously.

**Note:** Threads within a process share the same memory space, which is not the case with processes. As no two processes share memory space, the sharing of information between processes is much slower than sharing information between threads.

The only challenge faced while working with threads or the only time when they are hard to manage is in a situation where the same piece of data is being used by multiple threads. In the next section, we will learn start working with threads.

## 5.2 How to create a thread

**Threading** is part of your standard library and you don't need to download anything. In order to work with threads, we need to import the '**threading**' module. Following steps are involved while working with threads:

**Step 1:** Import the threading module

This is the first statement that you write in your code while working with threads. In order to work with threads, it is important to **import** the **threading** module.

```
import threading
```

## Step 2: Define the target function

You need to define the target function that the thread would invoke.

```
>>>def function_name():


```

## Step 3: Instantiate a thread

When you instantiate a thread, you have to provide the function that you created in step 2 as the target function.

```
thread_name = threading.Thread(target = function_name, args = (i,))
```

The easiest way of using a thread is to instantiate it with the target function, and that is what we have done in this step. Python's threading module provides **Thread()** method for this purpose.

```
class threading.Thread(group = name, target = None, name = None, args = (), kwargs = {})

 • group: Is reserved for future implementations and its value is None.
 • target: Name of the function that must be executed when the thread is started.
 • name: This allows you to name your thread. By default, Python provides a new name to the thread, which is of the format Thread-N.
 • args: args is a tuple that is used to pass values to the target function.
 • kwargs: Dictionary of keyword arguments that are to be used for target functions. It is an optional keyword arguments directory.
```

## Step 4: Start the thread

To make the thread work, you need to call the **start()** method. The **start()** method allows the thread to begin its work.

```
thread_name.start()
```

This method must be called at most once per thread object. Here are few points to remember about the `start()` method:

- This method is used to create new threads in Windows and Linux.
- With this method, the thread starts and thread calls the target function with the passed list of arguments (provided for tuple `args` when you instantiated the thread).
- The tuple `args` is used to forward arguments to the target function. If your function does not require any, then make use of an empty tuple.

### Step 5: Call the `join()` method

```
thread_name.join()
```

The `join()` method is invoked to block the calling thread till the object on which it was called is blocked. The `join()` method is invoked from the main thread. Once it is invoked, the `join()` method will prevent the main thread from exiting before thread on which it has been invoked.

**Note:** Importance of `join()` method is explained in **Example 5.2**

### Example 5.1

Write a program to display the name of threads.

#### Step 1 : Import the threading module.

```
import threading
```

#### Step 2: Define the target function.

```
def thread_count(count):
 print("I am the thread number ",count,".")
```

The function `thread count` displays the number of the thread.

#### Step 3: Instantiate a thread.

```
for i in range(1,6):
 t = threading.Thread(target = thread_count,args = (i,))
```

Here, we are instantiating five threads and the `count(i)` is passed as an argument to the target function `thread_count()`.

**Step 4:** Start the thread.

```
t.start()
```

**Step 5:** Call the `join()` method.

```
t.join()
```

**The final code:**

```
import threading

def thread_count(count):
 print("I am the thread number ",count,".")

for i in range(1,6):
 t = threading.Thread(target = thread_count,args = (i,))
 t.start()
 t.join()
```

The screenshot shows a code editor window with the following Python code:

```
import threading

def thread_count(count):
 print("I am the thread number ",count,".")

for i in range(1,6):
 t = threading.Thread(target = thread_count,args = (i,))
 t.start()
 t.join()
```

Annotations with orange circles and arrows explain the code:

- An annotation points to the `import threading` statement with the text "Import threading module".
- An annotation points to the `target = thread_count` part of the `Thread` constructor with the text "target function".
- An annotation points to the `t.start()` call with the text "The thread does not start running until the start() method".
- An annotation points to the `t.join()` call with the text "The join function will make the main thread wait until the thread t has finished its task".
- A large annotation covers the `for` loop and the `Thread` instantiation with the text "The thread is instantiated using the Thread object with 'thread\_count' as target function and value of 'i' as arguments".

*Figure 5.1*

**Output:**

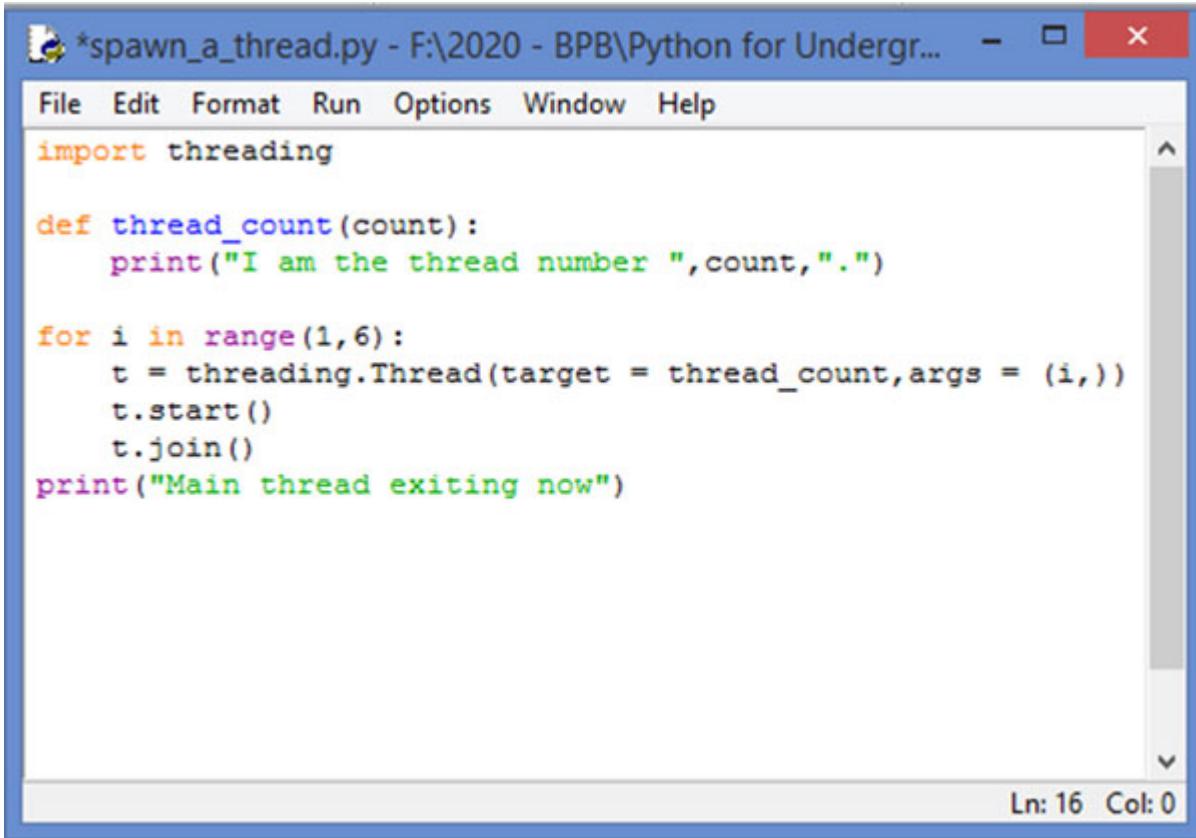
```
I am the thread number 1 .
I am the thread number 2 .
I am the thread number 3 .
I am the thread number 4 .
I am the thread number 5 .
>>>
```

**Example 5.2:**

Explain Significance of `join()` method.

**Answer:**

In this exercise, we add one line at the end of the code of *example 5.1*.



The screenshot shows a Windows-style application window titled "spawn\_a\_thread.py - F:\2020 - BPB\Python for Undergr...". The window contains a code editor with the following Python script:

```
File Edit Format Run Options Window Help
import threading

def thread_count(count):
 print("I am the thread number ",count,".")

for i in range(1,6):
 t = threading.Thread(target = thread_count,args = (i,))
 t.start()
 t.join()
print("Main thread exiting now")
```

The status bar at the bottom right indicates "Ln: 16 Col: 0".

*Figure 5.2*

Now, let's execute the code:

```
I am the thread number 1 .
I am the thread number 2 .
I am the thread number 3 .
I am the thread number 4 .
I am the thread number 5 .
Main thread exiting now
```

Figure 5.3

You can see that the main thread waits for all the threads to finish their tasks before printing its exiting statement “`Main thread exiting now`”.

Now, let’s comment the `t.join()` statement and execute the code.

The screenshot shows a Python code editor window. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is:

```
File Edit Format Run Options Window Help
import threading

def thread_count(count):
 print("I am the thread number ",count,".")

for i in range(1,6):
 t = threading.Thread(target = thread_count,args = (i,))
 t.start()
t.join()
print("Main thread exiting now")
```

The status bar at the bottom right indicates "Ln: 17 Col: 0".

Figure 5.4

## Output:

```
I am the thread number Main thread exiting nowI am the thread number I am the th
read number I am the thread number I am the thread number
 1
>>> 2345
```

Figure 5.5

The main thread exits before the other threads.

### **Example 5.3:**

Write a code to determine the current thread.

You may recall, that if you do not explicitly name a thread, then it will be given a unique name in the format: ‘Thread-N’, where N is the number of the thread. So, the names of the threads will be : Thread-1, Thread-2, Thread-3..., and so on. In this example, we will learn how to determine the current thread.

### **Answer:**

#### **Step 1:** Import the threading module

```
import threading
```

#### **Step 2:** Define the target function

```
def thread_count(count):
 print("I am the thread number ",count,".")
 print("My name is ",threading.currentThread().getName())
```

The `threading.currentThread()` returns the current thread object corresponding to the caller’s thread of control. The current thread then invokes `getName()` function to return the name of the thread.

#### **Step 3 :** Instantiate a Thread

```
for i in range(1,6):
 t = threading.Thread(target = thread_count,args = (i,))
```

#### **Step 4:** Start the thread

```
t.start()
```

#### **Step 5:** Call the `join()` method

```
t.join()
```

### **Code:**

```
import threading
def thread_count(count):
```

```

print("I am the thread number ",count,".")
print("My name is ",threading.currentThread().getName())

for i in range(1,6):
 t = threading.Thread(target = thread_count,args = (i,))
 t.start()
 t.join()

```

## Output:

```

I am the thread number 1 .
My name is Thread-1
I am the thread number 2 .
My name is Thread-2
I am the thread number 3 .
My name is Thread-3
I am the thread number 4 .
My name is Thread-4
I am the thread number 5 .
My name is Thread-5

```

## Example 5.4

Suppose you are taking some guests along with you to a club for which you have an exclusive membership. The club welcomes the guest and provides a **guest\_id** for the evening. There is a **5-second** wait for welcoming each guest and **10-second** wait for providing a **guest\_id**. Let's try to write the code for this situation:

**Step 1:** Import the time Module.

```
import time
```

**Step 2:** Create a function to take the names of the guests.

We create a function that takes the names of our guests:

```

names=[]
def getnames():
 guest_name = input("Enter a guest name : ")
 names.append(guest_name)
 confirmation = input("Any more guests? (y/n) : ")
 if(confirmation == 'y'):
 getnames()

```

**Step 3:** Create a function to welcome our guests.

```

def welcome_guests(names):
 for name in names:
 print("Welcome to our club",name)
 time.sleep(5)

```

#### Step 4: Allot a guest\_id to each guest.

```

def get_id(names):
 count=1
 for name in names:
 print("The guest_id of ",name,"is :", count)
 count = count+1
 time.sleep(10)

```

The final code will look as follows:

```

import time
names=[]

def getnames():
 guest_name = input("Enter a guest name : ")
 names.append(guest_name)
 confirmation = input("Any more guests? (y/n) : ")
 if(confirmation == 'y'):
 getnames()

def welcome_guests(names):
 for name in names:
 print("Welcome to our club",name)
 time.sleep(5)

def get_id(names):
 count=1
 for name in names:
 print("The guest_id of ",name,"is :", count)
 count = count+1
 time.sleep(10)

#EXECUTION

getnames()
welcome_guests(names)
get_id(names)

```

#### Output:

```

Enter a guest name : Raj
Any more guests? (y/n) : y
Enter a guest name : Ted

```

```

Any more guests? (y/n) : y
Enter a guest name : Reggie
Any more guests? (y/n) : n
Welcome to our club Raj
Welcome to our club Ted
Welcome to our club Reggie
The guest_idof Raj is : 1
The guest_idof Ted is : 2
The guest_idof Reggie is : 3

```

Now, every welcoming message was at a gap of 5 sec, and every id message was at a gap of 10 seconds.

### **Example 5.5:**

Accomplish the task done in *example 5.4* using threading module.

### **Code:**

```

import time
import threading
names=[]

def getnames():
 guest_name = input("Enter a guest name : ")
 names.append(guest_name)
 confirmation = input("Any more guests? (y/n) : ")
 if(confirmation == 'y'):
 getnames()

def welcome_guests(names):
 for name in names:
 print("Welcome to our club",name)
 time.sleep(5)

def get_id(names):
 count=1
 for name in names:
 print("The guest_id of ",name,"is :", count)
 count = count+1
 time.sleep(10)

getnames()
t = time.time()

#Create Threads
t1 = threading.Thread(target=welcome_guests, args = (names,))
t2 = threading.Thread(target=get_id, args = (names,))

#Start Threads
t1.start()

```

```
time.sleep(2)
t2.start()

#Join the threads
t1.join()
t2.join()
```

## Output:

```
Enter a guest name : Raj
Any more guests? (y/n) : y
Enter a guest name : Ted
Any more guests? (y/n) : y
Enter a guest name : Reggie
Any more guests? (y/n) : n
Welcome to our club Raj
The guest_idof Raj is : 1
Welcome to our club Ted
Welcome to our club Reggie
The guest_idof Ted is : 2
The guest_idof Reggie is : 3
All Done enjoy
```

### 5.2.1 Implementation of new thread using threading module

To create a new thread using the `threading` module, you will have to create a subclass of Thread class. While doing so, you can make changes in only two methods:

1. The constructor, `__init__` method
2. The `run()` method

The following example shows implementation by overriding the `run()` method only.

#### **(I) Overriding the run() method only**

**Step 1:** Import time and threading module

For this example we need to import time and threading module.

```
import time
import threading
```

## Step 2: Define a new subclass of thread class and overriding `run()` method

Defining the subclass:

```
class learning_subclassing(threading.Thread):
```

In this step. Let's try to override the original `run()` method. The original code of `run()` method in `threading.py` is given as follows:

```
try:
 if self._target:
 self._target(*self._args, **self._kwargs)
finally:
 # Avoid a refcycle if the thread is running a function with
 # an argument that has a member that points to the thread.
 del self._target, self._args, self._kwargs
```

For this example, in this step, we are just adding a print statement on top of the original code. So, the `run()` method will now look like this:

```
def run(self):
 print('\n thread {} has started'.format(self.getName()),"\n")#new code added
 try:
 if self._target:
 self._target(*self._args, **self._kwargs)

 finally:
 # Avoid a refcycle if the thread is running a function with
 # an argument that has a member that points to the thread.
 del self._target, self._args, self._kwargs
```

This `run()` method becomes the starting point for the thread.

## Step 3: Define the target function

This function takes two parameters: `sec` and `name`. Where, `name` is the name of the thread, and `sec` is the time in seconds. Basically in this thread the thread `name` sleeps for `sec` seconds.

```
def nap(sec,name):
 print("\n Hi I am a thread and my name is {}. I am here only to take a nap of {} seconds".format(name,sec),"\n")
 time.sleep(sec)
 print("{} seconds are over {} is up!!".format(sec,name),"\n")
```

## Step 4: Create new thread

**It is mandatory to call the constructor of the Thread class.**

```
t1 = learning_subclassing(target = nap, name = "King-thread",args = (10,"King-thread"))
t2 = learning_subclassing(target = nap, name = "Queen-thread",args = (6,"Queen-thread"))
t3 = learning_subclassing(target = nap, name = "Page-thread",args = (7,"Page-thread"))
```

**Step 5:** Start the execution of the threads with **start()** function.

```
t1.start()
t2.start()
t3.start()
```

**Step 6:** Call the **join()** function.

```
t1.join()
t2.join()
t3.join()
```

Now, your code should look something like this:

**Code:**

```
import time
import threading
class learning_subclassing(threading.Thread):
 def run(self):
 print('\n thread {} has started'.format(self.getName()), "\n")
 try:
 if self._target:
 self._target(*self._args, **self._kwargs)
 finally:
 del self._target, self._args, self._kwargs

 def nap(sec,name):
 print("\n Hi I am a thread and my name is {}. I am here only to take a nap of {} seconds".format(name,sec),"\n")
 time.sleep(sec)
 print("{} seconds are over {} is up!!".format(sec,name),"\n")

t1 = learning_subclassing(target = nap, name = "King-thread",args = (10,"King-thread"))
t2 = learning_subclassing(target = nap, name = "Queen-thread",args = (6,"Queen-thread"))
```

```
t3 = learning_subclassing(target = nap, name = "Page-thread",args = (7,"Page-thread"))
t1.start()
t2.start()
t3.start()
t1.join()
t2.join()
t3.join()
```

## Output:

```
thread King-thread has started
thread Queen-thread has started
thread Page-thread has started

Hi I am a thread and my name is King-thread. I am here only to take a nap of 10
seconds
Hi I am a thread and my name is Queen-thread. I am here only to take a nap of 6
seconds
Hi I am a thread and my name is Page-thread. I am here only to take a nap of 7
seconds

6 seconds are over Queen-thread is up!!
7 seconds are over Page-thread is up!!
10 seconds are over King-thread is up!!

>>>
```

## (II) Overriding the `__init__()` and `run()` method

Here, we will implement the same example by overriding the `__init__()` and `run()` methods of threading module.

### Step 1: Import statements.

```
import time
import threading
```

### Step 2: Define class `Learn_thread` that inherits the Python threading.Thread class.

```
class Learn_thread(threading.Thread):
```

### Step 3: Override constructor.

While overriding the constructor, the `__init__()` of the base class must be invoked as shown as follows:

```
class Learn_thread(threading.Thread):
 def __init__(self, name, sec):
 #invoke __init__ of base class
 threading.Thread.__init__(self)
 self.name = name
 self.sec = sec
```

#### Step 4: Define the nap function.

This is same as the previous examples:

```
def nap(sec, name):
 print("\n Hi I am a thread and my name is {}. I am here only to take a nap of {} seconds".format(name, sec), "\n")
 time.sleep(sec)
 print("{} seconds are over {} is up!!".format(sec, name), "\n")
```

#### Step 5: Override `run()` method.

Since we have created a new `__init__()` method, there is no target. The `run()` method will make a call to the `nap()` function. So, the logic must be implemented here.

```
#Overriding run()method
def run(self):
 print('\n thread {} has started'.format(self.getName()), '\n')
 nap(self.sec, self.name)
```

#### Step 6: Create an instance of threads and start the threads.

```
t1 = Learn_thread("King-thread", 10)
t2 = Learn_thread("Queen-thread", 6)
t3 = Learn_thread("Page-thread", 7)
t1.start()
t2.start()
t3.start()
t1.join()
t2.join()
t3.join()
```

The final code is as follows:

#### Code:

```

import time
import threading

class Learn_thread(threading.Thread):
 def __init__(self, name, sec):
 #invoke __init__ of base class
 threading.Thread.__init__(self)
 self.name = name
 self.sec = sec

 #Overriding run() method
 def run(self):
 print('\n thread {} has started'.format(self.getName()),"\n")
 nap(self.sec, self.name)

 def nap(sec, name):
 print("\n Hi I am a thread and my name is {}. I am here only to take a nap of {} seconds".format(name, sec), "\n")
 time.sleep(sec)
 print("{} seconds are over {} is up!!".format(sec, name), "\n")

t1 = Learn_thread("King-thread", 10)
t2 = Learn_thread("Queen-thread", 6)
t3 = Learn_thread("Page-thread", 7)
t1.start()
t2.start()
t3.start()
t1.join()
t2.join()
t3.join()

```

## Output:

```

thread King-thread has started
thread Queen-thread has started
thread Page-thread has started

Hi I am a thread and my name is King-thread. I am here only to take a nap of 10
seconds
Hi I am a thread and my name is Queen-thread. I am here only to take a nap of 6
seconds
Hi I am a thread and my name is Page-thread. I am here only to take a nap of 7
seconds

6 seconds are over Queen-thread is up!!

7 seconds are over Page-thread is up!!

10 seconds are over King-thread is up!!

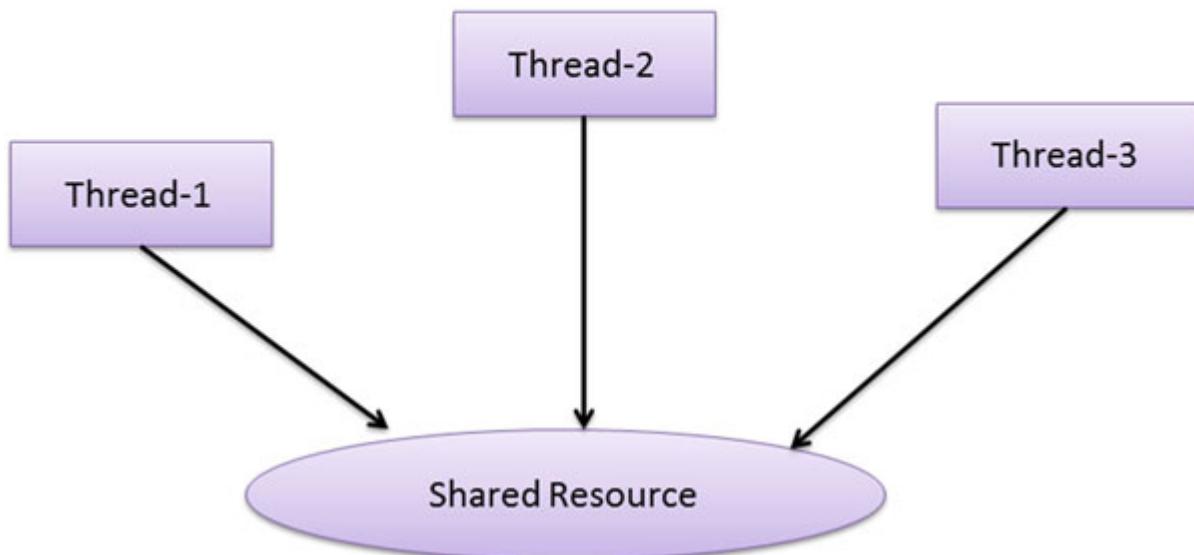
>>>

```

## 5.3 Thread synchronization with Lock and RLock

Thread synchronization is a mechanism that ensures that two or more concurrent threads that are sharing the same resource do not try to access the critical section of a program at the same time.

# Thread Synchronization



*Figure 5.6*

*Figure 5.6* shows three threads, namely, **Thread-1**, **Thread-2**, and **Thread-3** using a shared resource. Everything works well as long as all the three threads access the shared resource one at a time. The problem arises when more than one thread tries to access the shared resource simultaneously. This situation leads to **Race Condition**.

The **Race condition** is a situation where two or more threads access the shared resource simultaneously and attempt to change the data. Under this condition, the outcome of the code will be unpredictable. So, let's assume that two or more threads accessed a global variable simultaneously and changed its value. In this case, it is impossible to find out which thread changed the value of the variable.

### Example 5.6

With the help of code, explain Race Condition.

**Answer:**

Look at the following code:

**Code:**

```
import threading

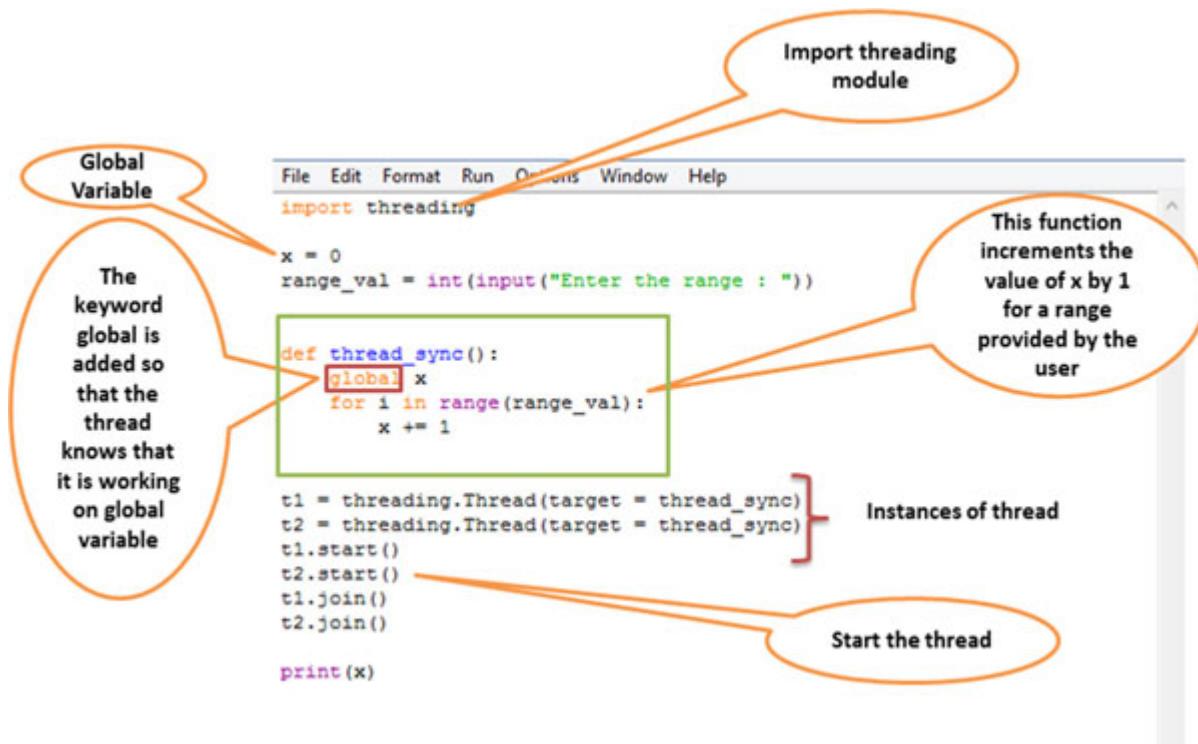
x = 0
range_val = int(input("Enter the range : "))

def thread_sync():
 global x
 for i in range(range_val):
 x += 1

t1 = threading.Thread(target = thread_sync)
t2 = threading.Thread(target = thread_sync)
t1.start()
t2.start()
t1.join()
t2.join()

print(x)
```

**Explanation of Code:**



*Figure 5.7*

The code prompts the user to enter a numeric value and assigns the value to the `range_val` variable. When the thread is invoked, it increments the value of `x` by 1 for `range = range_val`.

## Output:

### Case 1: `for range_val = 10`

The thread `t1` executes the `thread_sync()` function and increments it by 10. The thread `t2` also increments the value of `x` by 10, and so an output of 20 is displayed. This output is correct.

```

Enter the range : 10
20
>>>

```

### Case 2: `for range_val = 1000`

The thread `t1` executes the `thread_sync()` function and increments it by 1000. The thread `t2` also increments the value of `x` by 1000 and so an output of 2000 is displayed. This output is correct.

```

Enter the range : 1000

```

```
2000
">>>>
```

### Case 3: Output for range = 1,00,000

The thread **t1** executes the **thread\_sync()** function and increments it by 1,00,000. The thread **t2** also increments the value of **x** by 1,00,000 and we expect an output of 2,00,000. However, an output of 137540 is displayed. This output is incorrect.

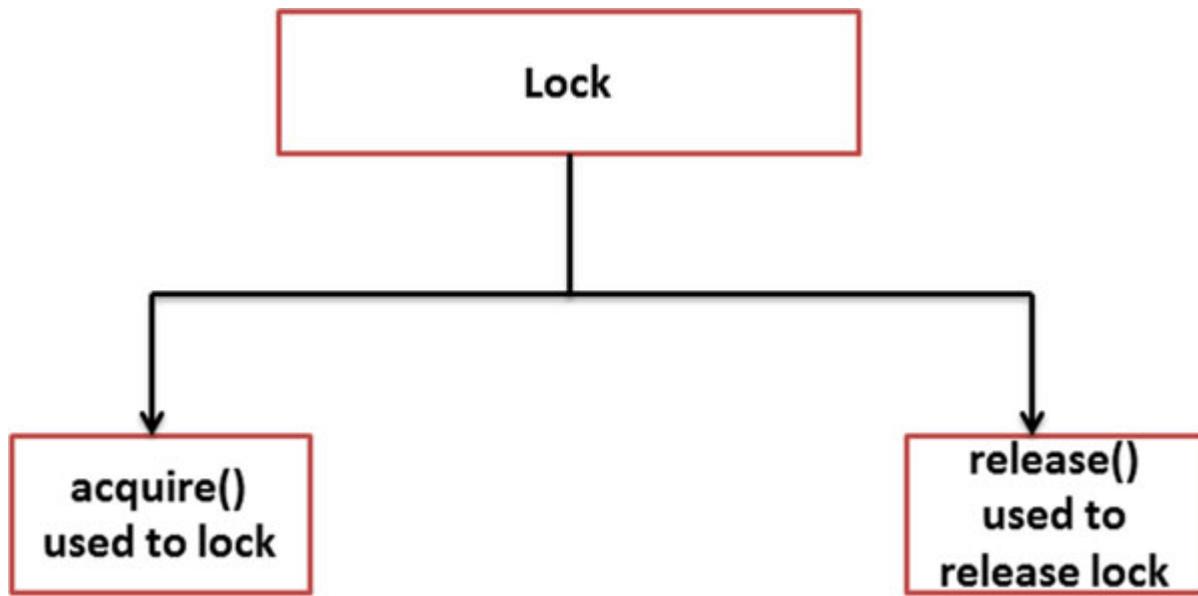
```
Enter the range : 100000
137540
>>>
```

In the first two cases, the results were as expected probably because the value of range was very small. However, when you provided the range of 1,00,000, the large range of value led to **thread synchronization**. Clearly, the two threads have tried to change the value of **x** at the same time which is why the result is not as expected.

## 5.4 Applying lock

In the last section, you read about **thread synchronization** and how it can yield an unpredictable output. It is important to have a synchronization tool to handle the situation leading to a Race condition. The **Lock** class of Python's **threading** module has a solution to this problem.

The lock object can be used to acquire and release the lock with the help of **acquire ()** and **release ()** methods, respectively.



*Figure 5.8*

The lock that is acquired by the thread can be **blocking** or **non-blocking** in nature. The `acquire()` method can take **True** or **False** as an argument. The default value is **True** which means if nothing is specified, call to `acquire()` function will be blocking in nature.

This means that the nature of the lock depends on whether the `acquire()` method is invoked with a **True** or **False** value. When invoked with value **True** (which is also the default value), the thread execution is blocked till the lock is unlocked. If the lock is acquired with value **False**, then the thread execution will not be blocked until it is set to **True**.

The acquired lock can be released by the `release()` method. This method allows only one thread to proceed if multiple threads are waiting for the lock to unlock. `RuntimeError` will be raised if the `release()` method is called when the lock is already unlocked.

### Example 5.7:

Work on the same code as **example 5.6** and show how it can be improved by applying a lock.

**Step 1:** Import statements.

```
import threading
```

## Step 2: Declare global variables.

```
x = 0
range_val = int(input("Enter the range : "))
```

## Step 3: Create an instance of the Lock class.

```
lock = threading.Lock()
```

## Step 4: Define the target function.

The target function `thread_sync` takes in one parameter, which is the `lock` object.

Notice that `lock.acquire()` is called before incrementing the value of `x`. The `acquire()` method is invoked with no values so the default value of `true` implies. Once the thread that has acquired the lock completes execution of the function, the lock is released by invoking the `release()` method, allowing the next thread to work on the function. The function code is a little different than that in *example 5.6* the difference between the two code is that:

1. The `acquire()` method is called before the value of `x` is changed so that only one thread at a time has access to global variable `x`.
2. A `release()` method is called after the value of `x` is changed to unlock the lock.

```
def thread_sync(lock):
 global x
 for i in range(range_val):
 lock.acquire()
 x += 1
 lock.release()
```

## Step 5: Create Threads.

Create threads with `target_sync` as the target function, and `lock` instance as the argument to be passed on to the target function.

```
t1 = threading.Thread(target = thread_sync, args = (lock,))
t2 = threading.Thread(target = thread_sync, args = (lock,))
```

## Step 6: Call the `start()` method.

```
t1.start()
t2.start()
```

### Step 7: Call the `join()` method

```
t1.join()
t2.join()
```

#### Code:

```
import threading
x = 0
range_val = int(input("Enter the range : "))

def thread_sync(lock):
 global x
 for i in range(range_val):
 lock.acquire()
 x += 1
 lock.release()

lock = threading.Lock()
t1 = threading.Thread(target = thread_sync, args = (lock,))
t2 = threading.Thread(target = thread_sync, args = (lock,))
t1.start()
t2.start()
t1.join()
t2.join()
print(x)
```

Output for  $x = 100, 00, 00$

```
Enter the range : 1000000
2000000
>>>
```

To check what happens if you try to release an unlocked lock, just comment the statement `lock.acquire()`. Execute the code, and see what happens.

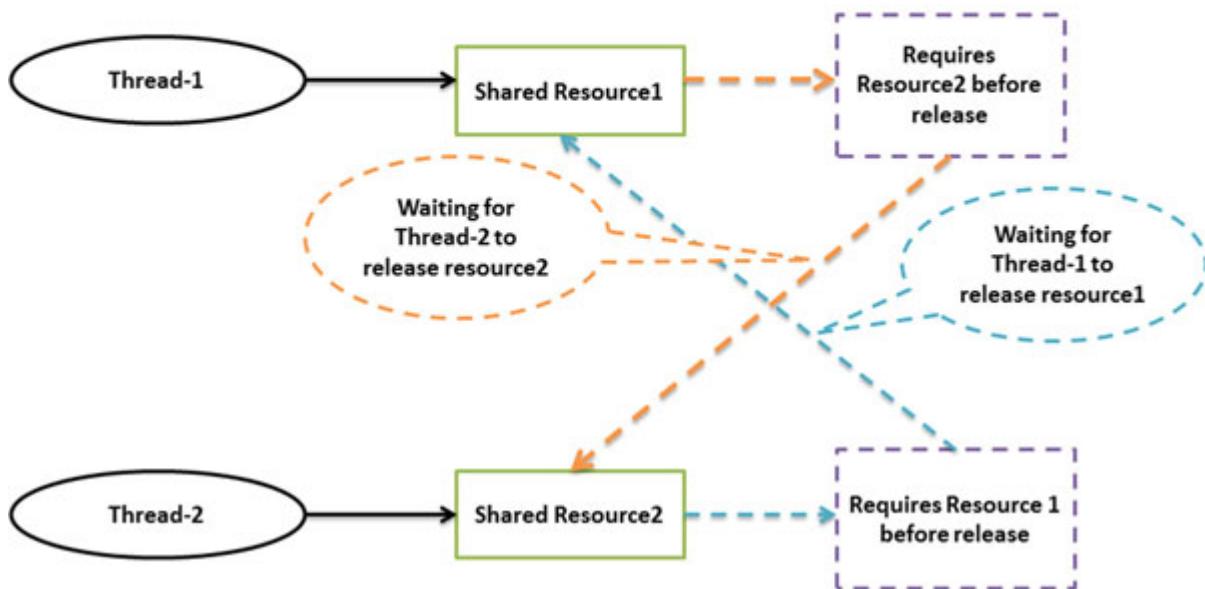
## 5.5 Deadlock

So far, you have come a long way. You have learned how to create threads and how to make them work. You realized how working with threads can sometimes lead to the problem of the **Race condition**. Finally, you learned how to use locking to avoid a Race condition. In this section, you will learn

about a major problem that can be caused by the locking mechanism. This problem is a case of **Deadlock**.

Let's assume that there are two threads ([Figure 5.9](#)) Thread-1 and Thread-2 having two shared resources at their disposal - Shared Resource1 and Shared Resource2. Assume that Thread-1 requires access to Shared Resource1 and Thread-2 requires access to Shared Resource2. Both the threads can acquire a lock and access the resource that they require without any problem. Now, imagine a state where Thread-1 requires access to Shared Resource2 and Thread-2 requires access to Shared Resource1 before releasing the lock that they have acquired. Since, the resources are locked, both threads are blocked from accessing the resource that they require and they keep waiting. This condition is known as Deadlock.

## Dead Lock



*Figure 5.9*

The following code displays a scenario of **dead lock**. It has two functions `thread_sync1()` and `thread_sync2()` that carry out different calculations by using two global variables `x` and `y`. Thread `t1` calls `thread_sync1()`, acquires `lock 1` and starts working with `x` and thread `t2` calls `thread_sync2()` and acquires `lock2`. Till this time things work fine, but now `t1` requires access to `y` and `t2` requires access to `x` but both the

variables are locked, and so it is a **deadlock** condition. This is also explained in [figure 5.9](#).

```
import threading
x = 0
y = 0

def thread_sync1(number):
 global x
 global y
 print("thread_sync1 acquiring lock1\n")
 lock1.acquire()
 x = number*2
 print("thread_sync1 acquiring lock2\n")
 lock2.acquire()
 z = x+y
 print("z =", z)
 print("thread_sync1 releasing lock2\n")
 lock2.release()
 print("thread_sync1 releasing lock1\n")
 lock1.release()
 print("thread_sync1 done\n")

def thread_sync2(number):
 global x
 global y
 print("thread_sync2 acquiring lock2\n")

 lock2.acquire()
 y = number*4
 print("thread_sync2 acquiring lock1\n")
 lock1.acquire()
 v = y - x
 print("v = ", v)
 print("thread_sync2 releasing lock1\n")
 lock1.release()
 print("thread_sync2 releasing lock2\n")
 lock2.release()
 print("thread_sync2 done\n")

lock1 = threading.Lock()
lock2 = threading.Lock()
t1 = threading.Thread(target = thread_sync1, args =(2,))
t2 = threading.Thread(target = thread_sync2, args =(3,))
t1.start()
t2.start()
t1.join()
t2.join()
print("Final x = ", x)
print("Final y = ", y)
```

## Explanation:

The diagram illustrates the execution flow of two threads, t1 and t2, using locks lock1 and lock2. Thread t1 starts at step 2, while thread t2 starts at step 3. Both threads print their respective acquisition and release messages. Arrows indicate the sequence of events: t1 acquires lock1, then lock2, and releases them in reverse order. t2 acquires lock2, then lock1, and releases them in reverse order. A double-headed arrow connects the two threads, indicating they are running simultaneously.

```
import threading
x = 0
y = 0

def thread_sync1(number):
 global x
 global y
 print("thread_sync1 acquiring lock1\n")
 lock1.acquire()
 x = number*2 t1 acquires access to x
 print("thread_sync1 acquiring lock2\n")
 lock2.acquire() t1 requires access to y but
 t2 is working on it
 z = x*y
 print("z =", z)
 print("thread_sync1 releasing lock2\n")
 lock2.release()
 print("thread_sync1 releasing lock1\n")
 lock1.release()
 print("thread_sync1 done\n")

lock1 = threading.Lock()
t1 = threading.Thread(target = thread_sync1, args =(2,))

def thread_sync2(number):
 global x
 global y
 print("thread_sync2 acquiring lock2\n")
 lock2.acquire()
 y = number*4 t2 acquires access to y
 print("thread_sync2 acquiring lock1\n")
 lock1.acquire() t2 requires access to x but
 t1 is working on it
 v = y - x
 print("v =", v)
 print("thread_sync2 releasing lock1\n")
 lock1.release()
 print("thread_sync2 releasing lock2\n")
 lock2.release()
 print("thread_sync2 done\n")

lock2 = threading.Lock()
t2 = threading.Thread(target = thread_sync2, args =(3,))
```

Figure 5.10

## Output:

```
thread_sync1 acquiring lock1
thread_sync2 acquiring lock2

thread_sync1 acquiring lock2
thread_sync2 acquiring lock1
```

### 5.5.1 Using `locked()` function to check if a resource is locked

You can make use of the `locked()` method to check whether the lock is held. The lines of code using this method have been marked in bold for you to identify.

```
import threading
x = 0
y = 0

def thread_sync1(number):
 global x
 global y
 print("thread_sync1 acquiring lock1\n")
 lock1.acquire()
 x = number*2
```

```

print("thread_sync1 acquiring lock2\n")
if lock2.locked():
 print("in thread_sync1, the next resource is locked waiting\n")
else:
 print("We have got the next resource")
lock2.acquire()
z = x+y
print("z =", z)
print("thread_sync1 releasing lock2\n")
lock2.release()
print("thread_sync1 releasing lock1\n")
lock1.release()
print("thread_sync1 done\n")

def thread_sync2(number):
 global x
 global y
 print("thread_sync2 acquiring lock2\n")
 lock2.acquire()
 y = number*4
 print("thread_sync2 acquiring lock1\n")
 if lock1.locked():
 print("in thread_sync2, the next resource is locked waiting\n")
 else:
 print("We have got the next resource")
 lock1.acquire()
 v = y - x
 print("v = ", v)
 print("thread_sync2 releasing lock1\n")
 lock1.release()
 print("thread_sync2 releasing lock2\n")
 lock2.release()
 print("thread_sync2 done\n")

lock1 = threading.Lock()
lock2 = threading.Lock()
t1 = threading.Thread(target = thread_sync1, args =(2,))
t2 = threading.Thread(target = thread_sync2, args =(3,))

t1.start()
t2.start()
t1.join()
t2.join()

```

## Output:

```

thread_sync1 acquiring lock1
thread_sync2 acquiring lock2

thread_sync1 acquiring lock2
thread_sync2 acquiring lock1

```

```
in thread_sync1, the next resource is locked waiting
in thread_sync2, the next resource is locked waiting
```

Although locking provides a solution for some issues in programming, it comes along with its share of problems. **Deadlock** is one such example. **Locking** introduces unnecessary overhead and affects the scalability and readability of the code. A code containing the locking mechanism is quite difficult to debug. This is why it is important to have alternative methods to ensure synchronized access to shared memory.

## 5.5.2 Resolving with RLOCK

In this section, you will learn about **re-entrant lock**. As the name suggests, a **re-entrant** or **RLock** can be acquired several times by the same thread. **RLock** too has an **acquire()** and a **release()** method, but its approach is a little different. Unlike the **lock**, **RLock** can be acquired multiple times and it must be released the same number of times to be unlocked. Another major difference between **RLock** and a lock is that **RLock** can only be released by the thread that has acquired it which is not the case with **lock**. It does not recognize which thread currently holds the **lock**.

The **RLock** can prevent undesired blocking of threads from accessing the shared resource. A shared resource in **RLock** can be accessed repeatedly by various threads. It is a little advanced version of simple locking mechanism. In case of simple lock, the same thread cannot acquire the same lock twice whereas that is not the case with **RLock** object can be re-acquired by the same threads.

The following code resolves the deadlock condition . with the help of RLock.

```
import threading
x = 0
y = 0

def thread_sync1(number):
 global x
 global y
 print("thread_sync1 acquiring lock1\n")
 lock.acquire()
 x = number*2
 print("thread_sync1 acquiring lock2\n")
 lock.acquire()
```

```

z = x+y
print("z =",z)
print("thread_sync1 releasing lock2\n")
lock.release()
print("thread_sync1 releasing lock1\n")
lock.release()
print("thread_sync1 done\n")

def thread_sync2(number):
 global x
 global y
 print("thread_sync2 acquiring lock2\n")
 lock.acquire()
 y = number*4
 print("thread_sync2 acquiring lock1\n")
 lock.acquire()
 v = y - x
 print("v = ",v)
 print("thread_sync2 releasing lock1\n")
 lock.release()
 print("thread_sync2 releasing lock2\n")
 lock.release()
 print("thread_sync2 done\n")

lock = threading.RLock()
t1 = threading.Thread(target = thread_sync1, args =(2,))
t2 = threading.Thread(target = thread_sync2, args =(3,))

t1.start()
t2.start()
t1.join()
t2.join()

print("Final x = ",x)
print("Final y = ",y)

```

## Output:

```

thread_sync1 acquiring lock
thread_sync2 acquiring lock

thread_sync1 acquiring lock

z = 4
thread_sync1 releasing lock

thread_sync1 releasing lock

thread_sync1 done
thread_sync2 acquiring lock

```

```
v = 8
thread_sync2 releasing lock
thread_sync2 releasing lock
thread_sync2 done

Final x = 4
Final y = 12
>>>
```

## 5.6 Semaphore

A **semaphore** is an abstract data type managed by the operating system that provides threads with synchronized access to a restricted number of resources.

A semaphore is nothing but an internal variable that reflects number of concurrent access to the resource it is associated with. The value of semaphore is always greater than 0 and less than or equal to the total number of available resources. It also has an `acquire()` and a `release()` method.

Every time a thread acquires a resource synchronized by a semaphore, the value of semaphore is decremented, and similarly when a thread releases a resource synchronized by a semaphore, the value of semaphore is incremented. This concept of **Semaphore** was invented by the Dutch computer scientist Edsger Dijkstra.

Multiple threads can access the same code simultaneously depending on how the semaphore is initialized.

Let's try to write a simple code using semaphore:

**Step 1:** Import statement.

```
import threading
```

**Step 2:** Initialize a semaphore.

```
t = threading.Semaphore(3)
```

In this step, the access to resource is limited to 3 threads.

**Step 3:** Define the function.

Here we have defined a simple function `count()` that prints numbers 1 to 5.

```
def count():
 t.acquire()
 print("Start")
 for i in range(1, 6):
 print(i)
 t.release()
```

#### Step 4: Create threads.

In this step, let's create 6 threads all have the target function as `count` which we created in the last step.

```
thread1 = threading.Thread(target=count)
thread2 = threading.Thread(target=count)
thread3 = threading.Thread(target=count)
thread4 = threading.Thread(target=count)
thread5 = threading.Thread(target=count)
thread6 = threading.Thread(target=count)
```

#### Step 5: `start()` and `join()`

```
thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()

thread1.join()
thread2.join()
thread3.join()
thread4.join()
thread5.join()
thread6.join()
```

#### Code:

```
import threading

t = threading.Semaphore(3)
def count():
 t.acquire()
 print("Start")
 for i in range(1, 6):
 print(i)
```

```
t.release()

thread1 = threading.Thread(target=count)
thread2 = threading.Thread(target=count)
thread3 = threading.Thread(target=count)
thread4 = threading.Thread(target=count)
thread5 = threading.Thread(target=count)
thread6 = threading.Thread(target=count)

thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()
thread1.join()
thread2.join()
thread3.join()
thread4.join()
thread5.join()
thread6.join()
```

Since the semaphore has limited the maximum number of concurrent threads to 3, the output is in two sets. The first three threads execute the function together, and the next three execute later.

## Output:

```
StartStartStart
>>>
111
222
333
444
555
StartStartStart
111
222
333
444
555
```

You can check which threads executed together.

```
import threading

t = threading.Semaphore(3)
def count():
 t.acquire()
```

```

print("Starting ",threading.currentThread().getName())
for i in range(1, 6):
 print(i)
t.release()

thread1 = threading.Thread(target=count)
thread2 = threading.Thread(target=count)
thread3 = threading.Thread(target=count)
thread4 = threading.Thread(target=count)
thread5 = threading.Thread(target=count)
thread6 = threading.Thread(target=count)

thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()
thread1.join()
thread2.join()
thread3.join()
thread4.join()
thread5.join()
thread6.join()

```

## Output:

```

Starting
>>> Starting Starting Thread-1Thread-2Thread-3

111
222
333
444
555
Starting StartingStarting Thread-4Thread-5Thread-6
111
222
333
444
555

```

Now, let's assume that the Semaphore allowed 7 concurrent threads instead of three.

```
t = threading.Semaphore(7)
```

In this case, all six threads will be able to access the code simultaneously.

```
StartStartStart
StartStartStart
111111
222222
333333
444444
555555
```

## 5.7 Thread synchronization using an event object

In this section, you will learn about how **Event** class objects can be used for communication within threads. This involves a very simple mechanism, where one thread signals an event, and the other waits for it.

The event object uses an internal flag known as the **event** flag. It can be set to **True** using **set()** method, and similarly, it can be set to **False** using the **clear()** method. The **wait()** method of the **Event** class blocks the thread until the event flag is set to **True**.

The event object can be initialized as follows:

```
e = threading.Event()
```

By default, the **event** flag is set to **False** at the time of initialization. The status of the flag can be checked with the help of **isSet()** method.

```
>>> import threading
>>>evt = threading.Event()
>>>evt.isSet()
False
>>>
```

The internal event flag can be set to **True** using the **set()** method.

```
>>> import threading
>>>evt = threading.Event()
>>>evt.isSet()
False
>>>evt.set()
>>>evt.isSet()
True
>>>
```

You can reset the **event** flag to **False** using the **clear()** method.

```
>>> import threading
>>>evt = threading.Event()
>>>evt.isSet()
False
>>>evt.set()
>>>evt.isSet()
True
>>>evt.clear()
>>>evt.isSet()
False
>>>
```

The `wait()` method can be used to make a thread wait for an event. The `wait()` method can be implied only on that event, which has its event flag set to `False`. A thread can wait as long as the event flag is `False`. If the state of the internal flag is `True`, then the thread cannot be blocked. You can specify the timeout for the `wait()` method as follows:

```
wait([Timeout])
```

### Code for event object with internal flag set to False.

The following code displays how event object works when internal flag is set to `False`:

```
import threading
import time

#The function task(), takes event object and time as parameter
def task(event, sec):
 print("Started thread but waiting for event...")
 # make the thread wait for event with timeout set
 internal_set = event.wait(sec)
 if internal_set:
 print("Event set")
 else:
 print("Time out,event not set")

initializing the event object
e = threading.Event()

starting the thread, target function is task, name = Event-Blocking-Thread,
#passes an event object and time of 5 seconds

t1 = threading.Thread(name='Event-Blocking-Thread', target=task, args=(e,5))
t1.start()

Letting the main thread sleep for 3 seconds
time.sleep(3)
```

## Output:

```
Started thread but waiting for event...
>>> Time out,event not set
```

## Code for event object with internal flag set to true.

The following code displays how event object works when internal flag is set to **True** using the **set()** method.

```
import threading
import time

def task(event, sec):
 print("Started thread but waiting for event...")
 # make the thread wait for event with timeout set
 internal_set = event.wait(sec)
 print("waiting")
 if internal_set:
 print("Event set")
 else:
 print("Time out,event not set")
initializing the event object
e = threading.Event()
print("Event is set.")
starting the thread
t1 = threading.Thread(name='Event-Blocking-Thread', target=task, args=(e,10))
t1.start()
e.set()
```

## Output:

```
Event is set.
Started thread but waiting for event...
>>>
waiting
Event set
```

So, now that you have understood how an event object works, can you guess what the output of the code will be if the internal flag is set to **True** after calling the **join()** method?

```
import threading
import time

def task(event, sec):
 print("Started thread but waiting for event...")
```

```

make the thread wait for event with timeout set
internal_set = event.wait(sec)
print("waiting")
if internal_set:
 print("Event set")
else:
 print("Time out,event not set")

initializing the event object
e = threading.Event()
print('Event Object Created')
starting the thread
t1 = threading.Thread(name='Event-Blocking-Thread', target=task, args=(e,10))
t1.start()
t1.join()
e.set()

```

The `join()` statement will halt everything till `task()` function has finished executing and the internal flag of the event is set to `True` after the `join()` method. So, while the `task()` function is executing the 'e' is not set to true. Therefore, the output will be as follows:

```

Event is set.
Started thread but waiting for event...
waiting
Time out,event not set

```

So, it is important to set the internal flag of the event to True before calling the `join()` method.

## 5.8 Condition Class

The `Condition` class defines methods that can significantly improve the speed of communication between two threads. The objects of the `Condition` class are known as **Condition variables**:

- These objects make a thread wait till a condition occurs.
- A Condition variable is like a more advanced version of an `event` object.
- Condition variables cannot be shared across processes.
- They are always related to some kind of lock
- A lock is part of the Condition variable which eliminates the need to track it separately.

## Important Methods:

- **notify(n-1)**: This method will wake up one thread waiting on condition. You can also provide the number of the thread that you want to wake up.
- **notify\_all()**: Wakes up all threads waiting on the condition.
- **wait([timeout =none])**: Waits until notified or until a timeout occurs. This method terminates if **notify()** or **notify\_all()** method is called. A Runtime Error will be generated if the thread has not acquired a lock when this method is called. This method returns False if timeout occurs else it will return True.

## Code:

```
import threading
import time
lst = []
def buy():
 c.acquire()
 for i in range (1,6):
 item = input("Enter the item bought : ")
 lst.append(item)
 print("Notifying t2")
 c.notify()
 c.release()
def billing():
 c.acquire()
 c.wait(timeout = 0)
 c.release()
 print("you are billed for:")
 for item in lst:
 print(item)
c = threading.Condition()
t1 = threading.Thread(target = buy)
t2 = threading.Thread(target = billing)

t1.start()
time.sleep(5)
t2.start()
t1.join()
t2.join()
```

## Code:

```
Enter the item bought : Rice
```

```
Enter the item bought : Wheat
Enter the item bought : Grain
Enter the item bought : Pen
Enter the item bought : Pencil
Notifying t2
you are billed for:
Rice
Wheat
Grain
Pen
Pencil
```

## 5.9 Daemon and Non-Daemon Thread

Threads are of two types:

1. Daemon
2. Non-Daemon

A **daemon** thread does not block the main thread from exiting and continue to run in the background. A **non-daemon** thread on the contrary block the main thread from exiting till all the threads are killed.

**Daemon** threads provide support to **non-daemon** threads. Till now we have only worked with **non-daemon** threads. All **daemon** threads get terminated after the termination of a **non-daemon** thread. They need be terminated them explicitly.

Status of a thread can be set only after it has started.

There are three ways to create a daemon thread:

(I)

```
t1 = threading.Thread(target = method_name)
t1.setDaemon(True)
```

(II)

```
t = threading.Thread(target = method_name)
t.daemon = True
```

(III)

```
t = threading.Thread(target = method_name, daemon = True)
```

## Setting a Daemon thread and checking whether it is True

In the following example, the `setDaemon()` method is used to set a thread as daemon thread. The status must be set to `True` before the thread starts.

- `setDaemon(True)`: Sets thread to `daemon` thread.
- `setDaemon(False)`: Sets the thread to `non-daemon` thread.
- `setDaemon()` must be called before the `start()` method. If this method is called after calling the `start()` method, `Runtime Error` will be generated.
- You can check whether a thread is daemon or not with the help of the `isDaemon()` method. It returns `True` if a thread is daemonic, else it will return `False`.

## Check the status of the daemon thread whether it is True or Not:

```
import threading
import time

def print_if_daemon():
 print("inside target function")
 if t1.isDaemon():
 print(threading.currentThread().name, " is a daemon!!")

t1 = threading.Thread(target = print_if_daemon)
#Check if t1 is daemon thread
print("Present Status of thread t1 is: ",t1.isDaemon())

#Set t1 as Deamon thread
t1.setDaemon(True)

#start()
t1.start()

time.sleep(5)
print("Main Exiting")
t1.join()
```

## Output:

```
Present Status of thread t1 is: False
inside target function
Thread-1 is a daemon!!
Main Exiting
```

Last **daemon** thread terminates automatically after all non-daemon threads terminate:

### Behaviour of thread with daemon not set to true

```
import threading
import time

def work_for_t1():

 print('Starting of thread :', threading.current_thread().name)
 for i in range(0,11):
 print("Printing Number : ",i," \n")
 time.sleep(2)
 print('Finishing of thread :', threading.current_thread().name, " \n")

t1 = threading.Thread(target=work_for_t1, name='Thread1')
t1.start()
print("thread is Daemon : ", t1.daemon)
t1.join()
time.sleep(5)
print("Finishing thread: ", threading.current_thread().name, " \n")
```

### Output:

```
thread is Daemon : Starting of thread : FalseThread1

Printing Number : 0

Printing Number : 1

Printing Number : 2

Printing Number : 3

Printing Number : 4

Printing Number : 5

Printing Number : 6

Printing Number : 7

Printing Number : 8

Printing Number : 9

Printing Number : 10

Finishing of thread : Thread1

Finishing thread: MainThread
```

Since `t1` is a `non-daemon` thread, it blocks main thread from exiting till its task is finished.

### Behaviour of thread with daemon set to True

As you can see, the main thread got over after 2 was printed but `Thread1` continued.

By setting daemon `True`, the daemon thread will continue working even after the main thread terminates.

```
import threading
import time

def work_for_t1():

 print('Starting of thread :', threading.current_thread().name)
 for i in range(0,11):
 print("Printing Number : ",i," \n")
 time.sleep(2)
 print('Finishing of thread :', threading.current_thread().name, " \n")

t1 = threading.Thread(target=work_for_t1, name='Thread1')
t1.setDaemon(True)
t1.start()
time.sleep(5)
print("Finishing thread: ", threading.current_thread().name, " \n")
```

You can see that in the output the main thread terminates much before `t1` that is a daemon thread.

### Output:

```
Starting of thread : Thread1
Printing Number : 0

Printing Number : 1

Printing Number : 2

Finishing thread: MainThread

Printing Number : 3

Printing Number : 4

Printing Number : 5

Printing Number : 6

Printing Number : 7
```

```
Printing Number : 8
Printing Number : 9
Printing Number : 10
Finishing of thread : Thread1
```

## Methods of the threading module:

Some of the important methods in threading module are as follows:

- **threading.activeCount() / threading.active\_count()**: Returns the number of active threads.
- **threading.currentThread() / threading.current\_thread()**: Returns the current thread.
- **threading.enumerate()**: The `enumerate()` method is used to return a list of all currently alive threads. This includes daemonic threads, dummy thread objects that are created by the current thread, and the main thread. Threads that have terminated not in the list.
- **threading.main\_thread()**: This method returns the main thread that starts the Python interpreter.

The implementation of these methods is shown in the code as follows:

```
import threading

def thread_count(count):
 print("I am the thread number ",count,".")
 print("My name is ",threading.currentThread().getName())
 print("active count",threading.activeCount())
 l1 = threading.enumerate()
 for i in l1:
 print(i)

 for i in range(1,3):
 t = threading.Thread(target = thread_count,args = (i,))
 t.start()
 t.join()
```

## Output:

```
I am the thread number 1 .
My name is Thread-1
active count 3
<_MainThread(MainThread, started 7188)>
```

```
<Thread(SocketThread, started daemon 7884)>
<Thread(Thread-1, started 10208)>
I am the thread number 2 .
My name is Thread-2
active count 3
<_MainThread(MainThread, started 7188)>
<Thread(SocketThread, started daemon 7884)>
<Thread(Thread-2, started 5648)>
>>>
```

## The nature of main thread

The main thread in Python is a non-daemon thread. You can check by typing the following statements in the Python shell.

```
>>>import threading
>>>threading.current_thread().getName()
'MainThread'
>>>threading.current_thread().isDaemon()
False
>>>
```

## Conclusion

In this chapter, you learnt about how to use the threading module to work with Python threads. Python threading can be implemented to simplify a design and speed up the program by allowing certain sections of the code to run concurrently. In the next chapter, you will learn how to work with Graphical User Interface in Python.

## Questions and answers

1. Fill in the Blanks:

1. The operating system creates a \_\_\_\_\_ to run programs.
2. A process can have multiple \_\_\_\_\_.
3. Threads are \_\_\_\_\_ -within a process.
4. Threads share \_\_\_\_\_ space.
5. Thread is the most fundamental synchronization mechanism provided by \_\_\_\_\_ module.

6. At any given time, a lock can be held by a \_\_\_\_\_ -thread or no thread at all.
7. Once the lock is \_\_\_\_\_, other threads are blocked from accessing the shared resource and they will have to wait till the lock is \_\_\_\_\_.
8. A lock can be acquired by two steps: (1) \_\_\_\_\_ and (2) \_\_\_\_\_.
9. To unlock the lock, call the \_\_\_\_\_ method.
10. The \_\_\_\_\_ method wakes up all thread waiting for the condition.
11. To create a new thread using the threading module, you will have to create a subclass of the \_\_\_\_\_ class.

**Answers:**

1. Process
  2. Threads
  3. mini-processes
  4. memory
  5. threading
  6. single
  7. acquired, released
  8. lock = Lock(),lock.acquire()
  9. release()
  10. notifyAll()
  11. Thread
2. Why is a thread called a lightweight process?
- Answer:** A thread is called a lightweight process because:
1. It is a part of a process and hence shares many of its characteristics.
  2. Its implementation is less onerous as compared to the implementation of the process to which it belongs.
3. What is GIL? Explain.

**Answer:** **GIL** stands for **Global Interpreter Lock**. Only one thread is allowed control over Python interpreter at a time. This means that only one thread execute at a time. It is a lock or a mutex that protects Python objects by preventing multiple threads from executing at the same time.

**GIL** is required because the Python interpreter is not completely thread safe. For supporting multi-threaded programming, it is important for the current thread to hold the global lock before accessing a Python object. For multi-threading, the interpreter regularly releases and reacquires the lock (by default after every ten byte codes of instructions, but this value can be changed).

#### 4. Compare a process and a thread.

**Answer:**

#### Process vs Thread

| Process                                                               | Thread                                                                            |
|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Independent sequence of execution.                                    | Independent sequence of execution.                                                |
| Run in separate memory space.                                         | Run in shared memory space.                                                       |
| The CPU feature.                                                      | Operating environment feature.                                                    |
| Process is an executing program.                                      | Thread is the basic unit to which the operating system allocates processing time. |
| Processes are heavy because they take more time in context switching. | Threads are light weight and hence take less time in switching.                   |

*Table 5.1*

#### 5. True or False:

1. Threads are separate programs running alongside each other.
2. Threads run within one process.
3. Two processes can share data between one another more easily than two threads.
4. Processes require much more memory overhead than threads.
5. Threads will make your program faster even if it is utilizing 100% of your CPU.

**Answer:**

1. True
  2. True
  3. False
  4. True
  5. Fast
6. Why do we need threads in programming?

**Answer:** Threads are required to speed up a process. They are used for executing quick tasks in the background of a process.

7. What is the purpose of the threading module?

**Answer:** The threading module is required to create, control, and manage threads.

8. It is said that the threading module has the thread class that implements threading. Name the important methods provided by the thread class and briefly explain their function.

**Answer:** Some of the important methods provided by the Thread class are as follows:

1. **run()**: Entry point for thread.
  2. **start()**: This method is called to start the thread. The **start()** method calls the **run()** method, which is the entry point of the thread.
  3. **join()**: This method blocks the calling thread or the main thread till the time the threads whose **join()** method are called are terminated.
  4. **isAlive()**: This method is used to check whether a thread has finished executing or not.
  5. **getName()**: This method returns the name of the thread.
  6. **setName()**: This method sets the name of the thread.
9. What are the various types of objects in the Python threading module?
- Answer:** The various types of objects in Python threading module are:
1. **Thread**: that represents the single thread of execution.

2. **Lock**: A primitive lock object. The lock object can be used to acquire and release the lock with the help of `acquire ()` and `release ()` methods respectively.
3. **RLock**: Re-entrant lock object. As the name suggests, a re-entrant or RLock can be acquired several times by the same thread. RLock too has an `acquire()` and a `release()` method, but its approach is a little different. Unlike the lock, RLock can be acquired multiple times, and it must be released the same number of times to be unlocked. Another major difference between RLock and a lock is that RLock can only be released by the thread that has acquired it, which is not the case with lock. It does not recognize which thread currently holds the lock.
4. **Condition**: The `Condition` class defines methods that can significantly improve the speed of communication between two threads. The objects of the Condition class are known as **Condition variables**.
  - These objects make a thread wait till a condition occurs.
  - A condition variable is like a more advanced version of an `Event` object.
  - Condition variables cannot be shared across processes.
  - They are always related to some kind of lock.
  - A lock is part of the Condition variable which eliminates the need to track it separately.
5. **Event**: Event class objects can be used for communication within threads. This involves a very simple mechanism, where one thread signals an event, and the other waits for it. The event object uses an internal flag known as the `event flag`. It can be set to `True` using `set()` method and similarly, it can be set to `False` using the `clear()` method. The `wait()` method of the Event class blocks the thread until the event flag is set to `True`.
6. **Semaphore**: A semaphore is nothing but an internal variable that reflects number of concurrent access to the resource it is associated with. The value of semaphore is always greater than 0 and less than or equal to the total number of available resources. It also has an `acquire()` and a `release()` method. Every time a

thread acquires a resource synchronized by a semaphore, the value of semaphore is decremented and similarly when a thread releases a resource synchronized by a semaphore, the value of semaphore is incremented. This concept of Semaphore was invented by the Dutch computer scientist *Edsger Dijkstra*.

10. Name the thread method that is used to wait until it terminates?

**Answer:** `join()`

11. How can you terminate a blocking thread?

**Answer:** You can terminate a blocking thread using the `thread.block()` or `thread.wait()` method.

12. What is the difference between a semaphore and bounded semaphore?

**Answer:** A bounded semaphore makes sure its current value does not exceed its initial value, which is not the case with semaphore.

13. What type of lock is RLock?

**Answer:** Re-entrant

14. Name the synchronization method that can be used to guard the resources with limited capacity such as a database server?

**Answer:** Semaphore

15. Which method is used to detect the status of a python thread?

**Answer:** `isAlive()`

16. How is global value mutation implemented for thread-safety?

**Answer:** With the help of Global Interpreter Lock.

17. Which module in Python supports threads?

**Answer:** Threading

18. Which of the following statement is true for RLock?

- a. If a thread already owns the lock, `acquire()` will increment the recursion level by one, and return immediately.
- b. If another thread owns the lock, `acquire()` will block until the lock is unlocked.

**Answer:** Both statements are true.

19. How would multi-threading impact a uniprocessor?

**Answer:** Multi-threading would degrade the performance of a uniprocessor.

20. Which are the two methods of Thread class that you can change while implementation of a new thread using threading module.

**Answer:** You can make changes in only two methods:

1. The constructor, `__init__` method
2. The `run()` method

21. Main thread is always non-daemon thread. Rest of the threads inherit daemon nature from their parents. Prove that:

- a. If parent thread is daemon, then child thread will also be non-daemon.
- b. If parent thread is daemon, then child thread will also be daemon thread.
- c. When last non-daemon thread terminates, automatically all daemon threads will be terminated. We are not required to terminated daemon thread explicitly.

**Answer:**

a.

```
from threading import Thread, current_thread
def disp():
 print('Disp Function')
#Proves that main thread is not Daemon
mt = current_thread()
print(mt.getName())
print(mt.isDaemon())
#Since main thread is executing t1.
#therefore, t1 is the child of the main thread.
t1 = Thread(target = disp)
print(t1.isDaemon())
t1.start()
```

**Output:**

```
MainThread
False
False
Disp Function
```

**b.**

```
from threading import Thread, current_thread
def disp():
 print('Disp Function')
If t1 is daemon then t2 is also daemon
#If t1 is non daemon t2 is also non daemon
 t2 = Thread(target= show)
 print("Is t2 Daemon? ",t2.isDaemon())
 t2.start()
def show():
 print('show function')

#Proves that main thread is not Daemon
mt = current_thread()
print("Is main thread Daemon? ",mt.isDaemon())

#Since main thread is executing t1.
#therefore, t1 is the child of the main thread.
t1 = Thread(target = disp)
print("Is t1 thread Daemon? ",t1.isDaemon())
t1.start()
```

**Output:**

```
Is main thread Daemon? False
Is t1 thread Daemon? False
Disp Function
>>>
Is t2 Daemon? False
show function
```

**c.**

```
import threading
import time

def work_for_t1():

 print('Starting of thread :', threading.current_thread().name)
 for i in range(0,11):
 print("Printing Number : ",i," \n")
 time.sleep(2)
 print('Finishing of thread :', threading.current_thread().name, " \n")

t1 = threading.Thread(target=work_for_t1, name='Thread1')
t1.setDaemon(True)
t1.start()
time.sleep(5)
print("Finishing thread: ", threading.current_thread().name, " \n")
```

## **Output:**

```
Starting of thread : Thread1
Printing Number : 0
Printing Number : 1
Printing Number : 2
Finishing thread: MainThread
```

# CHAPTER 6

## Errors, Exceptions, Testing, and Debugging

### Introduction

It is very important for a developer to have knowledge about various types of errors and exceptions that he or she can face while coding. A developer must also know how to fix the issues when they arise. In this chapter, you will learn about different types of errors, errors and exceptions faced by developers in Python, and how a developer can carry out testing and debugging using Python.

### Structure

- What is an Error?
  - Syntax errors
  - Runtime error
  - Logical error
- Exception
  - Try and Catch
  - Catching Exception in General
  - Try ... Except...else Statement
  - Try...Except...Finally.....
  - Try and Finally
  - Raise an Exception
- How to debug a program
- The ‘pdb’ debugger
- Debugger at the command line

- Unit testing and Test-driven development in Python
- Levels of testing
- What is pytest?
- The unittest module
- Defining multiple test cases with unittest and pytest
- List of Assert methods available in the ‘unittest’ module

## **Objectives**

After reading this chapter, you will:

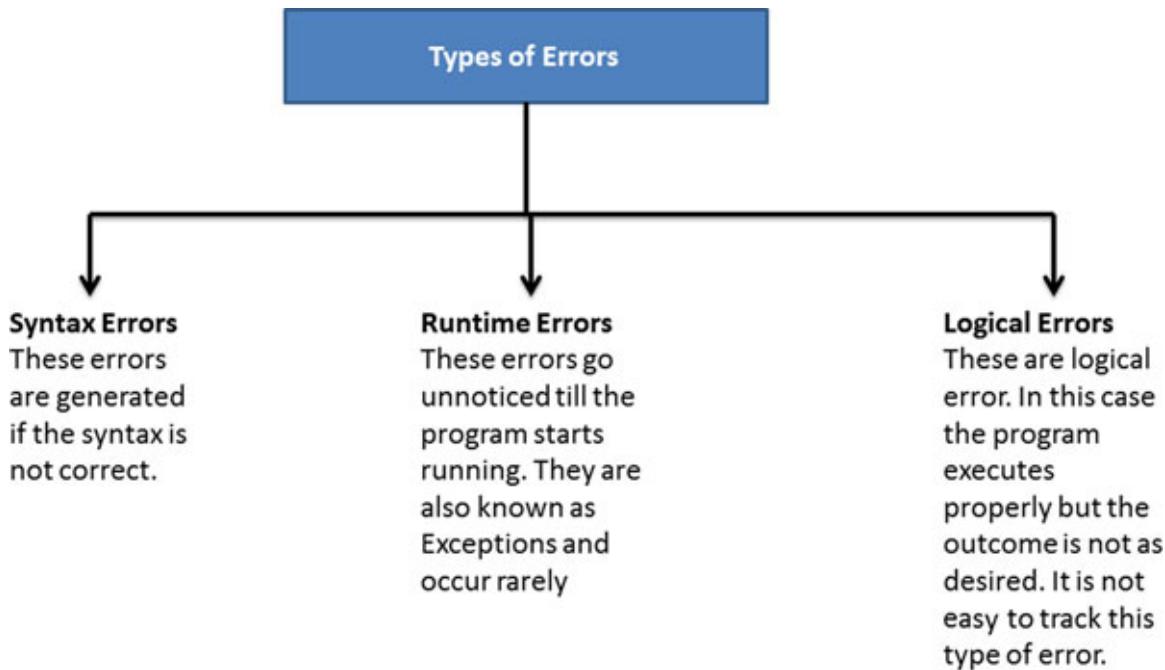
- Learn about various types of errors and exceptions
- How to catch an exception
- Debugging a program
- Creating unit test using unittest module

### **6.1 What is an error?**

At the time of software development, it is a common practice to run your code, again and again, to check if it is working fine. At times, the programmers notice small issues and fix them immediately, and sometimes they encounter problems that do not allow the program to recover and cause it to exit. It cannot be caught or handled. Such issues in the code that do not let the program to recover and forces it to exit are known as **errors**.

Errors can be classified as:

- Syntax errors
- Runtime errors
- Logical errors



*Figure 6.1*

### 6.1.1 Syntax Errors

Programming languages comprises of set of rules that directs how commands, symbols and syntax can be put together to write a piece of code that a computer can execute to carry out a particular task. When there is a problem with the syntax of the code, the developer encounters a syntax error.

- Syntax errors are very common
- A syntax error stops the program completely
- Syntax error can be spotted and fixed easily

#### **Example 6.1**

Give an example of syntax error.

#### **Answer:**

```

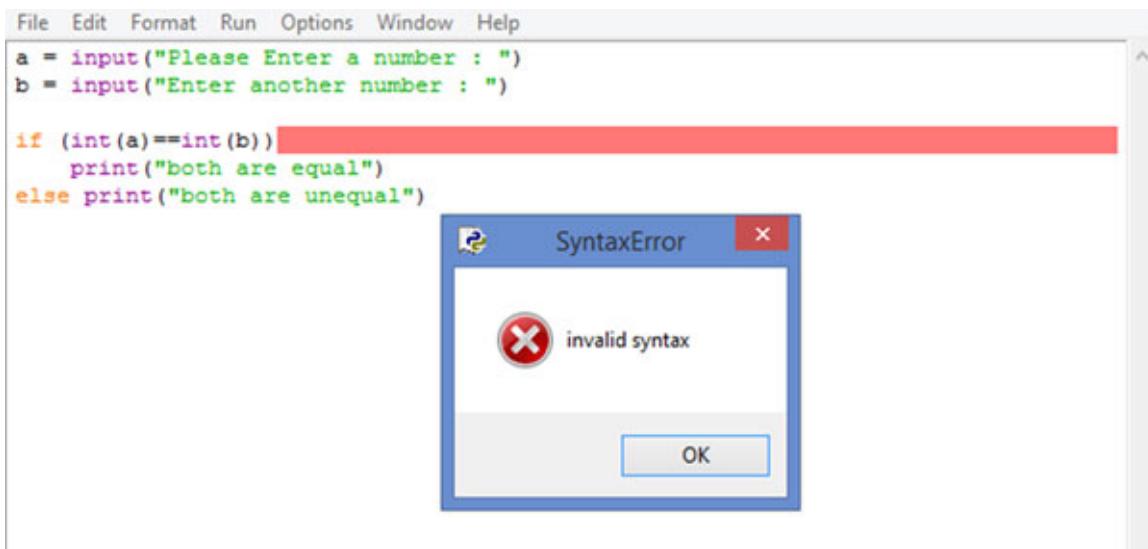
a = input("Please Enter a number : ")
b = input("Enter another number : ")

if (int(a)==int(b))
 print("both are equal")
else
 print("both are unequal")

```

As you can see the colon ":" is missing after the `if` statement.

On execution of the code syntax, the error message box shows up as shown in following screenshot:



The screenshot shows a Python code editor window with the following code:

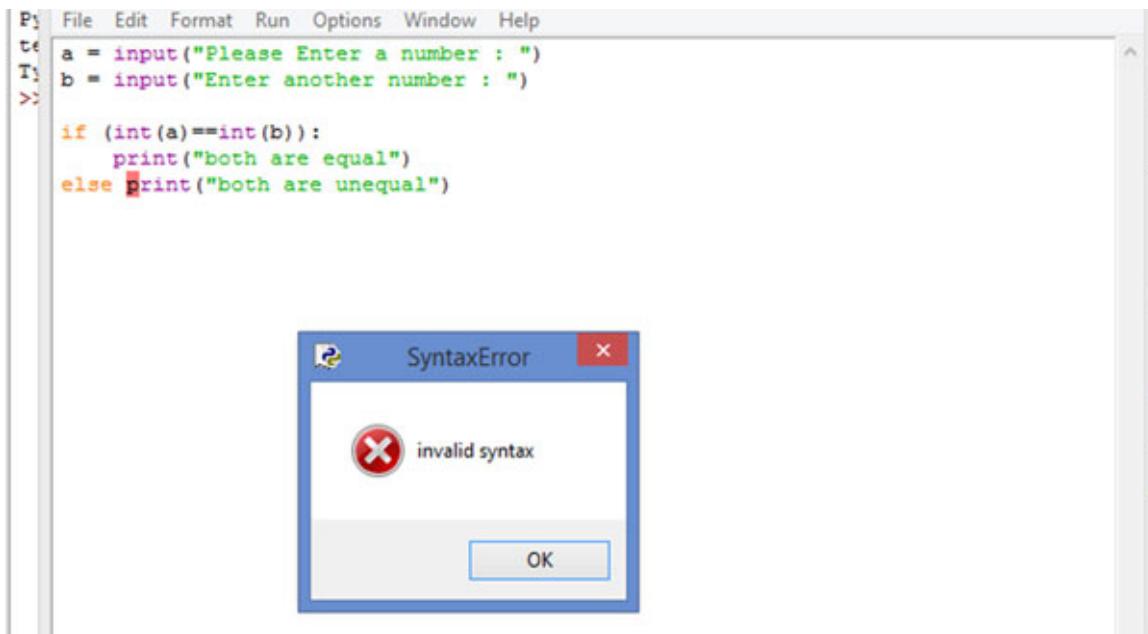
```
a = input("Please Enter a number : ")
b = input("Enter another number : ")

if (int(a)==int(b))
 print("both are equal")
else print("both are unequal")
```

A red rectangular highlight is placed over the line `if (int(a)==int(b))`. A modal error dialog box titled "SyntaxError" is displayed in the foreground, containing a red "X" icon and the text "invalid syntax". An "OK" button is at the bottom of the dialog.

*Figure 6.2: Syntax error*

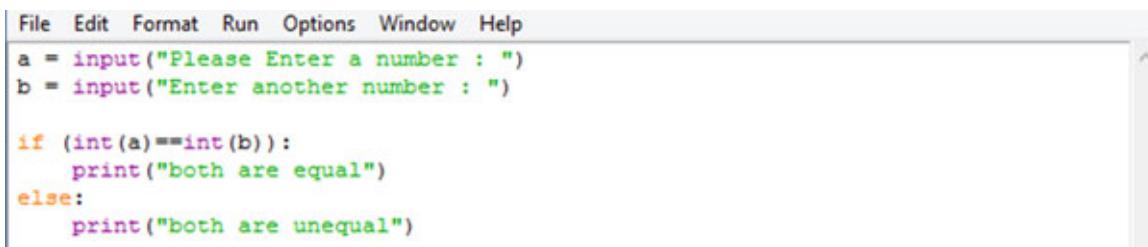
However, there seems to be more issues with the syntax. There is another syntax error – the ":" is missing again after the `else` statement. On executing the code again we get the following output:



The screenshot shows a Python code editor window with the same code as Figure 6.2, but now the line `else print("both are unequal")` is highlighted with a red rectangle. A modal error dialog box titled "SyntaxError" is displayed in the foreground, containing a red "X" icon and the text "invalid syntax". An "OK" button is at the bottom of the dialog.

*Figure 6.3*

So, after fixing all the issues we finally have an error-free code and on execution it works fine:



```
File Edit Format Run Options Window Help
a = input("Please Enter a number : ")
b = input("Enter another number : ")

if (int(a)==int(b)):
 print("both are equal")
else:
 print("both are unequal")
```

Figure 6.4

**Output1:**

```
Please Enter a number : 3
Enter another number : 3
both are equal
...
```

Figure 6.5

**Output2:**

```
Please Enter a number : 6
Enter another number : 5
both are unequal
```

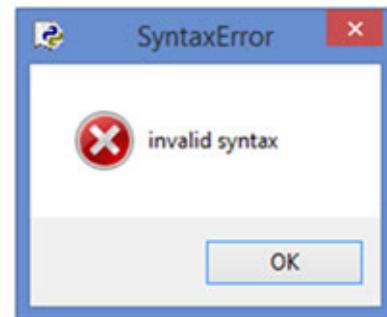
Figure 6.6

**Example 6.2**

```

import array as arr
a = arr.array('i', [9,3,2,90,65,23,45,77,76,57,0])
print("Array before Removing values") ← Parenthesis not closed
for element in a:
 print(element, end = ' ')
print("\n removing 77")
a.remove(77)
print("removing 76")
a.remove(76)
print("removing 57")
a.remove(57)
print("removing 0")
a.remove(0)
print("Array after removing the elements")
for element in a:
 print(element, end = ' ')

```



*Figure 6.7*

## 6.1.2 Runtime error

As the name suggests, the **runtime errors** show up during runtime or when the program is executing. Runtime errors are also called **Exceptions**, but there is a difference. There is a whole new section dedicated to Exceptions, where we will take that up. Runtime errors occur rarely. Trying to divide a number by zero is a perfect example of runtime error:

- Runtime errors go undetected until run time
- Some examples of runtime errors are:
  - Unable to find data or data does not exist
  - Invalid data type

### **Example 6.3**

Now, look at the following piece of code:

```

a = input("Please Enter father's age : ")
b = input("Enter Child's age : ")
print("age difference = ", (a-b))

```

Nothing seems wrong with the code. However, when we execute it, we receive a runtime error.

```
Please Enter father's age : 34
Enter Child's age : 4
Traceback (most recent call last):
 File "F:\2020 - BPB\Computer Science with Python - XI\Errors and Exceptions\code\errors.py", line 4, in <module>
 print("age difference = "+(a-b))
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>> |
```

*Figure 6.8*

What is the problem with the code?

**Answer:**

The problem with the code is that the input is received is in string format and before subtracting, we must convert the two values to integer.

We now fix the code and it looks like this:

```
a = input("Please Enter father's age : ")
b = input("Enter Child's age : ")

print("age difference = ",(int(a)-int(b)))
```

**Output:**

```
Please Enter father's age : 34
Enter Child's age : 4
age difference = 30
```

*Figure 6.9*

**Example 6.4**

The following program prompts the user to enter two numbers, and then it divides the first number by the second one and displays the output.

```
x = int(input("x = "))
y = int(input("y = "))
print(x/y)
```

**Output1:**

```
x = 8
y = 4
2.0
```

What would be the output if the user provides values x = 8 and y = 0?

**Output2:**

```
x = 8
y = 0
Traceback (most recent call last):
, line 3, in <module>
 print(x/y)
ZeroDivisionError: division by zero
```

### 6.1.3 Logical error

**Logical errors** do not create any problem at the time of code execution. It is only after the code has been executed that you realize that the output is not as expected. Logical errors generally show up at the time of testing the logical functioning of the code.

Look at the following code:

```
x = int(input("x = "))
y = int(input("y = "))
print(" The sum of x and y = ",x*y)
```

**Output 1:**

For the first output, let's run the code for x = 2 and y = 2.

```
x = 2
y = 2
The sum of x and y = 4
```

All seems fine. Let's again execute the code for x = 15 and y = 2.

**Output 2:**

```
x = 15
y = 2
The sum of x and y = 30
```

We expect the output to be 17, but it turns out to be 30. There is some problem in the logic. The message displays sum of x and y whereas the output is the

product of x and y. So, either the message is wrong or there is a problem with the output.

#### Semantic error

**Semantic errors** are errors that provide meaning less data. There is a difference of opinion on whether it is a **logical error** or a **syntax error**. **Logical errors** are caused due to wrong logic whereas semantic errors produce an output that has no meaning at all. For example, you have to find the percentage of number of students absent in the class and by mistake you multiply (number of students absent/total number of students) by 10 instead of 100. **Semantic errors** are not detected during the compile time. For this reason, many programmers consider **semantic error** to be same as **logical error**.

#### Design errors

**Design errors** are attributable to the lack of understanding of requirements forwarded by the client. Imagine a scenario where your code is correct, absolutely bug-free, exactly as per your expectations. However, the software that you created is not exactly what your client wanted. Many times, due to communication problems, either the client does not express the requirement in detail, or the service providers do not take down the details of the requirements properly. This is known as a **design error**.

## 6.2 Exception

An **Exception** is an event encountered while a program is executing that interrupts the working of the program. **Exceptions** too occur during runtime like runtime errors. It is for this reason that there is often confusion between the two, and sometimes exceptions are considered to be the same as **runtime errors**.

Exceptions are not the same as runtime errors because they can be handled and are not unconditionally fatal. As a matter of fact, an error can be considered to be an unchecked exception.

## 6.3 Errors with respect to Python

In this chapter, till now you have learnt about errors and exceptions in general. Now, it's time to learn about Errors Python's point of view. While coding in Python, you will encounter two types of Errors:

1. Syntax errors
2. Exception

You may recall that syntax errors are identified at the time of compilation, and it occurs when coding syntax rules are not followed properly. Exceptions occur

during runtime. They can be handled so that the execution of the script does not come to halt.

### **6.3.1 Try and Catch**

Let's once again discuss the logic of example 6.4.

```
x = int(input("x = "))
y = int(input("y = "))
print(x/y)
```

The code works fine till the user provides a value of 0 for y. This generates an error:

#### **ZeroDivisionError: division by zero**

`ZeroDivisionError` is **nonfatal**, but it can affect the working of a program. The good news is that we can handle it so that the execution of the program does not halt. This can be handled using the **try-except** block. The statement `divide = x/y` can raise an exception when `y = 0`. This is how you can implement **try-except**.

**Step 1:** The code capable of producing an error must be placed within the try block.

```
x = int(input("Enter first number : "))
y = int(input("enter second number : "))
try:
 divide = x/y
 print("try block over")
 print("divide = ",divide)
```

**Step 2.** Define the except block.

Inside the except block, we define what statements to execute if an error occurs.

```
except ZeroDivisionError:
 print("OOPS exception occured due to Zero division error")
 print("*****THE END*****")
```

**Code:**

```
x = int(input("Enter first number : "))
```

```

y = int(input("enter second number : "))
try:
 divide = x/y
 print("try block over")
 print("divide = ",divide)
except ZeroDivisionError:
 print("OOPS exception occured due to Zero division error")
 print("*****THE END*****")

```

1. The statements of the `try` block are executed first.
2. If no error occurs, then the exception block is skipped but if an error occurs and its type matches the `exception name` (which in this case is `ZeroDivisionError`) that is named after the `except` keyword, then the `except` clause is executed.

## **Output:**

### **Case 1:**

All ok; the try block is executed, except the block is skipped.

```

Enter first number : 1
enter second number : 2
try blockover
divide = 0.5
*****THE END*****

```

### **Case 2:**

Exception occurs; instructions in `try` block are skipped. The error matches the `except` error and executes it.

```

Enter first number : 1
enter second number : 0
OOPS exception occured due to Zero division error
*****THE END*****

```

## **Example 6.5**

What if the error does not match the exception name in front of the `except` clause?

### **Answer:**

If the exception that occurs does not match the exception name in the `except` clause, then it goes as an unhandled exception, and the execution will stop with a message.

```

a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("try block over")
 print("divide = ",divide)
except ZeroDivisionError:
 print("OOPS exception occurred due to Zero division error")
 print("*****THE END*****")

```

## Output:

```

Enter first number : 1
enter second number : 2
Traceback (most recent call last):
 File "F://Errors and Exceptions/code/trycatch.py", line 4, in <module>
 divide = a/b
TypeError: unsupported operand type(s) for /: 'int' and 'str'

```

In *example 6.5* we encountered **TypeError** whereas the **except** block was defined only to handle **ZeroDivisionError**. Since there was no handler found for **TypeError**, it went unhandled and the execution of the code stopped.

In the following code, the **TypeError** is also added as exception name in front of **except** keyword.

```

a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("try block over")
 print("divide = ",divide)
except (ZeroDivisionError,TypeError) :
 print("OOPS exception occurred due to Exception")
 print("*****THE END*****")

```

## Output:

```

Enter first number : 1
enter second number : 2
OOPS exception occurred due to Exception
*****THE END*****

```

So, you see the **except** clause handled the **TypeError** with same grace as **ZeroDivisionError**.

A **try** statement can have more than one **except** clause to specify handlers for different exceptions. At most only one handler is executed. A handler will only

handle the exception that has occurred in the `try` block associated to it, and not the one that exist in other handlers of the same `try` block.

### 6.3.2 Catching exception in general

There may be times when you are not sure about what kind of exception can arise, and you may want to catch exception in general and retrieve more details about it. Following is a simple way of getting the information:

```
a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("try block over")
 print("divide = ",divide)
except TypeError as e:
 print(str(e))
```

#### **Output:**

```
Enter first number : 1
enter second number : 2
unsupported operand type(s) for /: 'int' and 'str'
```

This is a very simple explanation of the error; however, it may not be useful when you are working with thousands of lines of code. You may want to know more information about the error so that it can be detected easily. For this, first `import sys`. It is a very important module of Python, and provides access to some very important variables, and functions that are used by the interpreter.

```
import sys
```

Next, we use `sys.exc_info()` to get details about the exception.

The `sys.exc_info()` function returns a tuple of three values.

These three values provide very important information about the exception that is being handled. The three values returned are:

- `type`: It is the type of exception being handled.
- `value`: Value gets the exception parameter
- `traceback`: It gets the traceback object, which encapsulates the call stack at the point where the exception originally occurred.

## **Example 6.6**

Write a program to retrieve the **type** for the exception that occurred.

### **Answer:**

```
import sys
a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("try block over")
 print("divide = ",divide)
#Access more information with sysexc_info
except:
 print("OOPS exception occurred due to ",sys.exc_info()[0])
 print("****THE END****")
```

### **Output:**

```
Enter first number : 1
enter second number : 2
OOPS exception occurred due to <class 'TypeError'>
****THE END****
```

## **Example 6.7:**

Write a program to retrieve the value for the exception that occurred.

### **Answer:**

```
import sys
a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("try block over")
 print("divide = ",divide)
#Access more information with sysexc_info
except:
 print("OOPS exception occurred due to ",sys.exc_info()[1])
 print("****THE END****")
```

### **Output:**

```
Enter first number : 1
enter second number : 6
OOPS exception occurred due to unsupported operand type(s) for /: 'int' and 'str'
```

```
*****THE END*****
```

## Example 6.7

Write a program to retrieve the **Traceback** for the exception that occurred.

### Answer:

```
import sys
a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("try block over")
 print("divide = ",divide)
#Access more information with sysexc_info
except:
 print("OOPS exception occurred due to ",sys.exc_info()[2])
 print("*****THE END*****")
```

### Output:

```
Enter first number : 1
enter second number : 2
OOPS exception occurred due to <traceback object at 0x010BF288>
*****THE END*****
```

## Example 6.8

Write a program to retrieve the Lineinfo for the exception that occurred:

```
import sys
a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("try block over")
 print("divide = ",divide)
#Access more information with sysexc_info
except:
 print("OOPS exception occurred due to line no.",sys.exc_info()[2].tb_lineno)
 print("*****THE END*****")
```

### Output:

```
Enter first number : 2
enter second number : 2
OOPS exception occurred due to line no. 5
```

```
*****THE END*****
```

### 6.3.3 Try ... Except...else Statement

The **try-except** statement comes with an optional **else** clause. When this clause is used, it must follow all except clauses. The **else** clause is used for a situation where we want that some code must be executed if the **try** clause does not raise an exception.

#### **Case 1 with exception, else block not executed.**

```
import sys
a = int(input("Enter first number : "))
b = input("enter second number : ")
try:
 divide = a/b
 print("divide = ",divide)
except:
 print("OOPS exception occurred due to line no.",sys.exc_info()[2].tb_lineno)
else:
 print("try over successfully")
 print("*****THE END*****")
```

#### **Output:**

```
Enter first number : 1
enter second number : 3
OOPS exception occurred due to line no. 5
*****THE END*****
```

#### **Case 2 With no exception, else block executed.**

```
a = int(input("Enter first number : "))
b = int(input("enter second number : "))
try:
 divide = a/b
 print("divide = ",divide)
except:
 print("OOPS exception occurred due to line no.",sys.exc_info()[2].tb_lineno)
else:
 print("try over successfully")
 print("*****THE END*****")
```

#### **Output:**

```
Enter first number : 2
```

```
enter second number : 6
divide = 0.3333333333333333
try over successfully
*****THE END*****
```

Instead of additional code to the **try** clause, one can use the **else** clause. This avoids accidentally catching an exception that wasn't raised by the code being protected by the **try-except** statement.

### 6.3.4 Try...Except...Finally...

Look at the following code, it displays how to use **try...except** and **Finally**:

```
def just_having_fun():
 try:
 print(4)
 except TypeError as e:
 print(str(e))
 finally:
 print("Ok Bye")

just_having_fun()
```

#### **Output:**

```
4
Ok Bye
>>>
```

### 6.3.5 Try and Finally

You can use the **try** block with **finally**. In the **finally** block, you can place the code which must be executed irrespective of whether the **try** block has thrown an exception or not.

#### **Syntax:**

```
try:
 # To do statements

finally:
 # Always to do statements


```

### 6.3.6 Raise an exception

An exception can be raised by using the **raise** clause. The syntax is as shown as follows:

```
Raise Exception_class
```

- The raise statement is used to raise an exception.
- The raise statement is followed by the name of the **Exception** class.

#### **Code:**

```
try:
 voter_age = int(input("Enter the age of the voter : "))
 if voter_age< 18:
 raise ValueError;
 else:
 print("You can vote")
except ValueError:
 print("You are not allowed to vote")
```

#### **Output1:**

```
Enter the age of the voter : 6
You are not allowed to vote
```

#### **Output2:**

```
Enter the age of the voter : 19
You can vote
```

Figure 6.10 displays some standard Exceptions you are likely to encounter while coding.

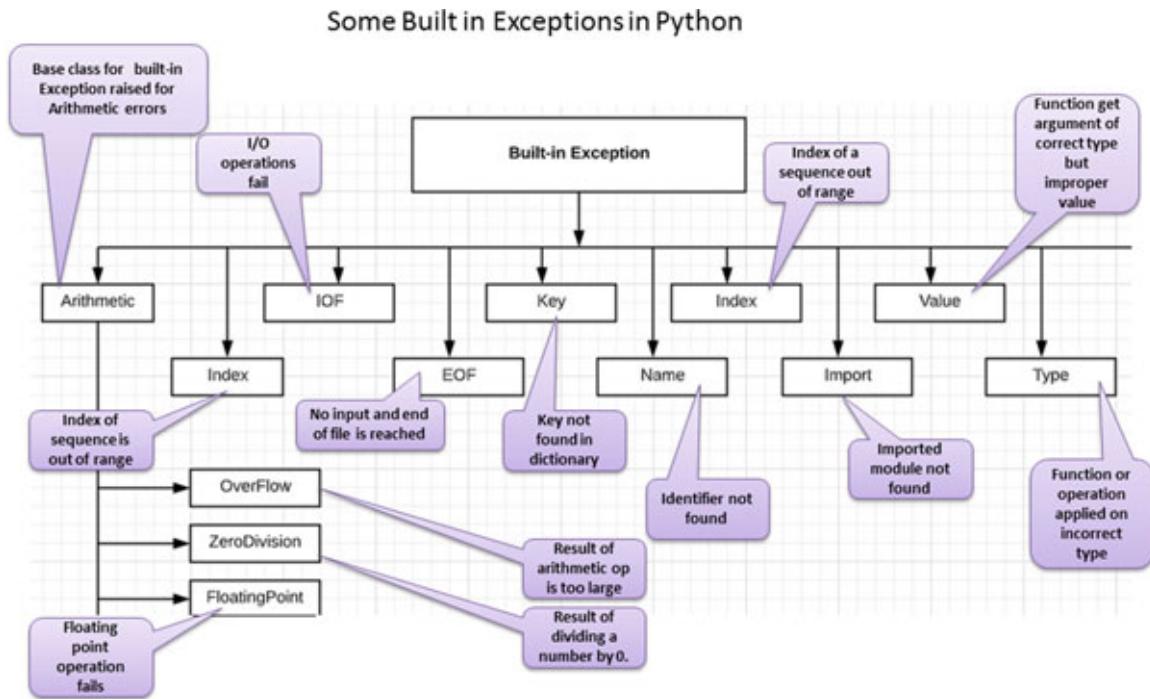


Figure 6.10: Standard exceptions

## 6.4 How to debug a program?

In order to be a good developer, it is very important to learn how to debug a program. If you pay attention to the error generated, line at which it is generated, and the error type you can easily locate the error. When you are debugging manually how well you will be able to resolve a problem is completely dependent on you and your understanding of debugging. We have already seen how to use exceptions to handle suspicious code and how to get more information about what went wrong. Now, in this section we will see some basic techniques to figure out where your code went wrong.

Small errors can be easily spotted. For example, if you look at the following code, everything may seem alright in the first go:

```

How to debug a program?
class Student:
 def __init__(self, name, email, department, age):
 self.name = name
 self.email = email
 self.department = department
 self.age = age

 def print_student(self):
 print("Name : ",self.name)

```

```
print("Email : ",self.email)
print("Department : ",self.department)
print("Age : ",self.age)

s1 = Student("Alex","alex@company_name.com","Electronics",18)
s1.print_stdent()
```

There is a class `student` and what this program does is create an instance of the class and print various values. An object `s1` is created and the `print_stdent()` message is called. But when you execute the program, it displays the following error:

```
Traceback (most recent call last):
 File "identifyerrors.py", line 16, in <module>
 s1.print_stdent()
AttributeError: 'Student' object has no attribute 'print_stdent'
```

Now, look at the error carefully. It says line 16.

```
Traceback (most recent call last):
 File "identifyerrors.py", line 16, in <module>
 s1.print_stdent()
AttributeError: 'Student' object has no attribute 'print_stdent'
```

*Figure 6.11*

Although, this is a small code, and you can easily count the lines, but if the code is huge then there will be more number of line. Go to your Python file, go to **Options > Show Line Numbers**.

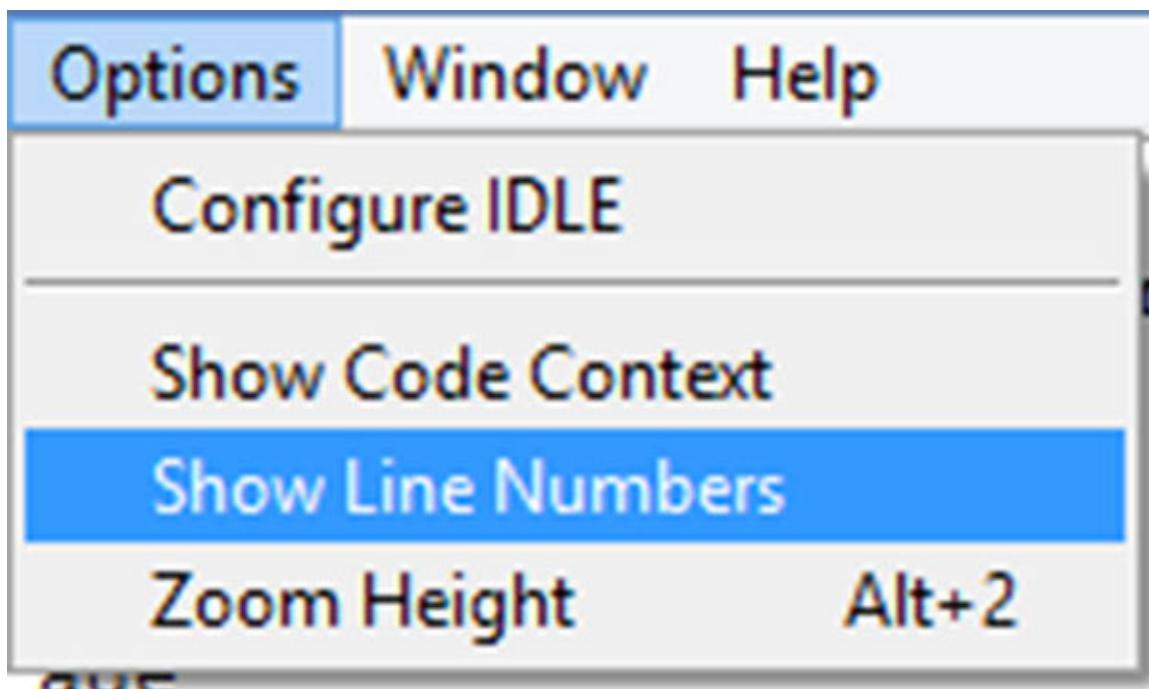


Figure 6.12

This would display the line numbers on the left-hand side of the code:

A screenshot of a Python code editor window. The menu bar includes 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code itself is as follows:

```
1 class Student:
2 def __init__(self, name, email, department, age):
3 self.name = name
4 self.email = email
5 self.department = department
6 self.age = age
7
8 def print_student(self):
9 print("Name : ",self.name)
10 print("Email : ",self.email)
11 print("Department : ",self.department)
12 print("Age : ",self.age)
13
14
15 s1 = Student("Alex","alex@company_name.com","Electronics",18)
16 s1.print_stdent()
17
```

The code uses standard Python syntax with color-coded keywords and comments.

Figure 6.13

So, you now have the line that is actually creating the problem([Figure 6.11](#)).

The error message further says: **AttributeError**: ‘Student’ object has no attribute ‘print\_stdent’

It is pointing at the word `print_stdent`.

If you look properly there is a spelling mistake. The name of the function is `print_student()` where as the object is calling function `print_stdent()` which does not exist. So, one small mistake in spelling can halt the entire program.

After making the changes, the code looks like this:

```
class Student:
 def __init__(self, name, email, department, age):
 self.name = name
 self.email = email
 self.department = department
 self.age = age

 def print_student(self):
 print("Name : ",self.name)
 print("Email : ",self.email)
 print("Department : ",self.department)
 print("Age : ",self.age)

s1 = Student("Alex","alex@company_name.com","Electronics",18)
s1.print_student()
```

And the output is as follows:

```
Name : Alex
Email : alex@company_name.com
Department : Electronics
Age : 18
>>>
```

Another way of debugging is by printing variable's intermediate values. Look at the following example. There is no syntax error and so no error is detected at the time of compilation; however, the output does not match the actual calculation. The code is supposed to take two numbers from the user, and multiply the first number with two and the second number with three and then add the two results together. So, if we take two numbers : 2 and 4, then the final answer should be  $4+12=16$ . However, that is not the case.

```
a = input("enter a number : ")
b = input("enter one more number : ")
c = a*2
d = b*3
print(c+d)
```

The output of this code is as follows:

```
enter a number : 2
enter one more number : 4
22444
```

While we are expecting an answer of **16**, what we get in return is **22444**, which indicates that there is definitely a **logical error**. In such cases, we can print values of variables, and see where things are going wrong.

```
a = input("enter a number : ")
print("value of first number a is ",a)
b = input("enter one more number : ")
print("value of second number b is ",b)
c = a*2
print("c is a multiplied by 2 i.e. ",c)
d = b*3
print("d is b multiplied by 3 i.e. ",d)
print("c + d = ",c+d)
```

We now see the output to check what is going wrong:

```
enter a number : 1
value of first number a is 1
enter one more number : 2
value of second number b is 2
c is a multiplied by 2 i.e. 11
d is b multiplied by 3 i.e. 222
c + d = 11222
>>>
```

The value of  $a = 1$ , therefore  $c = a*2 = 2$

The value of  $b = 2$ , therefore  $d = b *3 = 6$

Here, looking at the output, we can make out that the Python is treating these numbers as strings and is carrying out **string concatenation** operation. At a closer look, we realize that the input values were not converted to integer at the first place.

### Corrected code

```
a = int(input("enter a number : "))
print("value of first number a is ",a)
b = int(input("enter one more number : "))
print("value of second number b is ",b)
c = a*2
print("c is a multiplied by 2 i.e. ",c)
d = b*3
print("d is b multiplied by 3 i.e. ",d)
print("c + d = ",c+d)
```

## Output1:

```
enter a number : 1
value of first number a is 1
enter one more number : 2
value of second number b is 2
c is a multiplied by 2 i.e. 2
d is b multiplied by 3 i.e. 6
c + d = 8
```

## Output2:

```
enter a number : 2
value of first number a is 2
enter one more number : 4
value of second number b is 4
c is a multiplied by 2 i.e. 4
d is b multiplied by 3 i.e. 12
c + d = 16
```

Another technique for debugging is that of **code tracing**. In this case, you execute one line at a time, and see how the code is actually carrying out various functions. For code tracing, we often use debugger.

## 6.5 ‘pdb’ debugger

‘**pdb**’(Python debugger) is a command-line tool.

- It is a Python module that gets installed on your operating system when you install Python.
- It is used to debug Python programs.
- It is important to learn about **pdb** because in real-time scenario, it is required to run the script on server where you will not have luxury of running IDE. **pdb** will help you debug the Python code.
- It allows you to set break points so that you can:
  - Examine variables
  - Step forward line by line
  - Show the code as it executes
- The two most important statements that you would use with Python debugger are:
  - The import statement: **import pdb**

- o **`pdb.set_trace()`**: Adds a breakpoint wherever you want to examine the code.

```
import pdb
a = int(input("enter a number : "))
b = int(input("enter one more number : "))
pdb.set_trace()
c = a*2
d = b*3
pdb.set_trace()
print("c + d = ", c+d)
```

## 6.6 Debugger at command line

*Table 6.1* lists down some commands that can be used while debugging. *Figure 6.15* illustrates how these commands can be implemented.

|            |                                                                                       |
|------------|---------------------------------------------------------------------------------------|
| 'n'        | Execute the next line of code                                                         |
| 'c'        | Continue executing the code/complete execution                                        |
| 'l'        | Will list down 3 lines of code before after the line that is being executed right now |
| 's'        | Step in function and execute line by line                                             |
| 'b'        | Display the list of all the break points                                              |
| 'b[int] '  | You can set multiple break points with line number                                    |
| 'b[func] ' | Break at function name                                                                |
| 'cl'       | Clear all break points                                                                |
| 'cl[int] ' | Clear break points at a given line number                                             |
| 'p'        | Print                                                                                 |

*Table 6.1*

```

enter a number : 4
enter one more number : 2
> f:\2020 - bp\computer science with python - xi\errors and exceptions\code\identifyerrors.py(5)<module>()
-> c = a*2
(Pdb)
(Pdb) l
 1 import pdb
 2 a = int(input("enter a number : "))
 3 b = int(input("enter one more number : "))
 4 pdb.set_trace()
 5 -> c = a*2
 6 d = b*3
 7 pdb.set_trace()
 8 print("c + d = ",c+d)

[EOF]
(Pdb) p(a)
4
(Pdb) p(b)
2
(Pdb) n
> f:\2020 - bp\computer science with python - xi\errors and exceptions\code\identifyerrors.py(6)<module>()
-> d = b*3
(Pdb) p(d)
*** NameError: name 'd' is not defined
(Pdb) n
> f:\2020 - bp\computer science with python - xi\errors and exceptions\code\identifyerrors.py(7)<module>()
-> pdb.set_trace()
(Pdb) p(d)
6
(Pdb) c|l
Clear all breaks? y

```

*Figure 6.14*

## 6.7 Unit testing and test-driven development in Python

In this section, you will understand what is the need to carry out **unit testing**, and what is **test-driven development**.

### 6.7.1 Why do we unit test?

**unit testing** is the first of the several levels of software testing that a program undergoes. Always attempt to catch bugs as early as possible to prevent bugs from reaching the final product. Some of the reasons why unit testing is necessary are listed as follows:

- It reduces bugs in the existing features to a great extent and simplifies the process of adding in new features.
- Unit testing documentation provides a detailed report on how the process of testing was carried out, and if it is behaving as per expectations.
- Unit tests significantly reduce the cost of change in production because the bugs are caught, and fixed very early in the software development life cycle. It saves a lot of time and effort. It is difficult to fix bugs at the later stages of software development. Bugs in Software can affect the

business. It spoils the reputation of the company, and customers would prefer other products. Unit testing addresses problems very early and catches the bugs before they get to the field.

- Unit testing allows faster debugging.
- Unit tests play a significant role in designing a better program.

## **6.7.2 Objectives of unit testing**

The objectives of unit testing are as follows:

1. Tests the individual functions in the code.
2. Ensures that the basic individual units of software are error-free by identifying errors.
3. All types of positive and negative tests must be written for all functions of each unit.
4. All tests must be executed in the development environment rather than the production environment, so that you can run them easily any time, and any number of time.

## **6.8 Levels of testing**

In this section, we will discuss various levels of testing:

1. Unit testing
2. Integration testing
3. System-level testing
4. Performance testing

You will also learn about unit testing in Python, and frequent errors encountered in Python programming.

### **6.8.1 Unit testing**

**Software testing** is carried out at different levels. These levels ensure that errors are caught in time, and are prevented from moving on to the final product. **Unit testing** is the first layer of software testing, and is carried out at basic individual units of source code. Units are tested to determine whether they ready for integration with each other. Unit Testing is thus, the lowest level

testing (functions, subroutines, and classes are tested to check whether they are giving desired results or not). These are the most comprehensive tests that tests all the positives and negative test cases for a function.

## **6.8.2 Integration testing**

The units are integrated together, and integration testing is carried out on the components formed.

## **6.8.3 System-level testing**

System-level testing is carried out on the external interfaces of the system which is collection of subsystems.

## **6.8.4 Performance testing**

After unit, integration and system-level testing software is subjected to performance testing at subsystem, and system levels. During performance testing, the software is tested to verify that the timing and resource usages such as memory, CPU, disk usage) are acceptable.

## **6.8.5 Unit testing in Python**

The ideal approach for software development is to first understand the problem, create test cases, and then get going with the code. This is not mandatory, but undoubtedly, it is the best practise for software development.

## **6.8.6 Frequent errors encountered in Python programming**

Some of the frequent errors encountered in Python programming are as follows:

- Spelling mistakes in a variable name.
- Failure in initializing or reinitializing of variables.
- Sometimes, a coder gets confused between value equality and object equality. There is a difference between `a == b` and `id(a) == id(b)`.
- Logical errors.

The two most popular testing frameworks in Python are – `unittest` and `pytest`, and we are going to discuss both the frameworks in the coming sections.

### Assertions

While unit testing with `pytest` and `unittest` frameworks, you will be using assertions.

An assertion is a test that has an expression. If the expression evaluates to `True`, it means that the test has passed. If the expression evaluates to `False`, it means that the test has failed.

Assertions are carried out with the help of `Assert` statements. Python evaluates the expression associated with the assert statement. If the expression is `False`, Python generates an `AssertionError`.

Syntax for Assertion statement is as follows:

```
assert Expression (Arguments)
```

## 6.9 What is pytest?

The `pytest` is actually a Python framework for unit testing. It allows you to create tests, modules and fixtures. Working with `pytest` is much easier, and convenient than working on any other unit testing framework. In this section, you will learn how to:

- Install `pytest`.
- Use the in-built Python's assert statement, which simplifies the implementation of unit testing.
- Use the command line parameters to help filter the order in, which tests will be executed.

### Installation of pytest using pip

Give the following command on the command line:

```
pip install pytest
```

To check whether it has been installed, go to Python shell and write:

```
import pytest
```

Check whether it has been installed.

```
>>> import pytest
>>> |
```

**Figure 6.15**

Since no error is generated, it is successfully installed.

Let's start with our first program.

### **Step 1:** Write your code

Create a file that defines functions that have to be tested. In this example, we will work on a code that has functions defined for addition, subtraction, multiplication, and division.

```
def add(x,y):
 return x+y

def subtract(x,y):
 return x-y

def multiply(x,y):
 return x*y

def divide(x,y):
 if(y==0):
 str1 = "invalid denominator"
 return str1
 else:
 return x/y
```

Save your file. In this case, the file was saved by the name **arith\_funcs.py** in **F:\testing directory**.

### **Step 2:** Write a file to test code

Now, let's create another file to test these functions.

*The name of the file should start with “test\_”.*

For this example, the name of the file is **test\_arith\_funcs.py**.

#### **Step 2.1:** Import the file that it has to test.

```
import arith_funcs
```

#### **Step 2.2:** Define tests

Python functions written for testing start with ‘**test**’ at the beginning of the function name. Every test function calls the function that it is testing and validates the result of the action. For this purpose, it makes use of **assert** keyword. For example, the **add()** function in the **arith\_funcs.py** file must

provide addition of two numbers. So, when you write assert `arith_funcs.add(10,20) == 30`, you are defining that when you call the `add()` function of `arith_funcs.py` file and pass arguments 10 & 20, it must produce an output of 30, else an error must be generated. The assert statements have been defined for the rest of the functions in the same manner.

```
import arith_funcs

def test_add():
 assert arith_funcs.add(10,20) == 30

def test_subtract():
 assert arith_funcs.subtract(10,20) == -10

def test_multiply():
 assert arith_funcs.multiply(10,20) == 200

def test_divide():
 assert arith_funcs.divide(10,0) == "invalid denominator"
 assert arith_funcs.divide(10,5) == 2
```

**Step 2.3 : Save the file.**

Save the file by the name `test_arith_funcs.py` in `F:\testing` directory.

#### **Step 2.4:** Execute the test file.

#### **2.4.1:** Open the command prompt window.

#### 2.4.2: Give the `pytest` command.

The syntax of the executing the file is as follows:

```
pytest test_pythonfile.py
```

In this case, the test pythonfile.py = F:\testing\test\_arith\_funcs.py.

Give the following command on the command prompt:

```
pytest F:\testing\test_arith_funcs.py
```

```
----- 4 passed in 0.03s -----
C:\Users\MYPC>pytest F:\testing\test_arith_funcs.py
===== test session starts =====
platform win32 -- Python 3.8.2, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: C:\Users\MYPC
collected 4 items
...
[100%]
```

*Figure 6.16*

All test executed properly.

Now, let's see how it looks if there is an error in the code. For that, let's make a small change in the code. In the source file, we have defined that in the divide function if the denominator is zero, a string “*invalid denominator*” should be returned. Now suppose, in the test file, we put an assertion that, if the denominator is zero the value of the string returned should be “*zero not allowed as denominator*”. Since both strings don't match this test would fail. Let's try this out.

F:\testing\test\_arith\_funcs.py

```
import arith_funcs

def test_add():
 assert arith_funcs.add(10,20) == 30

def test_subtract():
 assert arith_funcs.subtract(10,20) == -10

def test_multiply():
 assert arith_funcs.multiply(10,20) == 200

def test_divide():
 assert arith_funcs.divide(10,0) == "zero not allowed as denominator"
 assert arith_funcs.divide(10,5) == 2
```

## Output:

```
===== FAILURES =====
test_divide
=====
> def test_divide():
> assert arith_funcs.divide(10,0) == "zero not allowed as denominator"
E AssertionError: assert 'invalid denominator' == 'zero not allowed as denominator'
E - zero not allowed as denominator
E + invalid denominator
R:\testing\test_arith_funcs.py:13: AssertionError
===== short test summary info =====
FAILED ::test_divide - AssertionError: assert 'invalid denominator' == 'zero ...
=====
1 failed, 3 passed in 0.30s =====
```

Figure 6.17

## 6.10 The unittest module

You have seen how to use **pytest** for unit testing. Now, get ready to work with the standard Python library containing module named **unittest**. The **unnittest** consists of:

- Core framework classes that form basis of test cases.
- Utility class for running the tests and reporting the outcome.
- **Class Testcase:** Sub classes of this class are used to perform unit testing.

## Rules for writing test methods

- If the name of the test method does not start with the word ‘**test**’, it would be ignored. Therefore, rule no. 1 is to start the name of the test with the word **test**.
- All the test methods will have only one parameter – **self**.
- ‘**self**’ will be used to call all the built-in assertion methods.

Let's now see how to work with the **unittest** module. We will work will be working with the same example of **arith\_funcs.py** file shown as follows:

```
def add(x,y):
 return x+y

def subtract(x,y):
 return x-y

def multiply(x,y):
 return x*y

def divide(x,y):
 if(y==0):
 str1 = "invalid denominator"
 return str1
 else:
 return x/y
```

To work with **unittest** module, please follow the steps as below:

### Step 1: Import the **unittest** module.

Import the **unittest** module in the **test\_arith\_funcs.py** file. The code for testing will be something like this:

```
import arith_funcs
import unittest

class TestArithFuncs(unittest.TestCase):
 #Define Testing Methods

 def test_add(self):
 self.assertEqual(30,arith_funcs.add(10,20))

 def test_subtract(self):
```

```

 self.assertEqual(-10,arith_funcs.subtract(10,20))

def test_multiply(self):
 self.assertEqual(200,arith_funcs.multiply(10,20))

def test_divide(self):
 self.assertEqual("zero not allowed as denominator",arith_funcs.divide(10,0))
 self.assertEqual(2,arith_funcs.divide(10,5))

unittest.main()

```

## Output:

```

= RESTART: F:\2020 - BPB\Python for Undergraduates\Testing and Debugging\Code\te
st_arith_funcs.py
.F..
=====
FAIL: test_divide (__main__.TestArithFuncs)

Traceback (most recent call last):
 File "F:\2020 - BPB\Python for Undergraduates\Testing and Debugging\Code\test_
arith_funcs.py", line 17, in test_divide
 self.assertEqual("zero not allowed as denominator",arith_funcs.divide(10,0))
AssertionError: 'zero not allowed as denominator' != 'invalid denominator'
- zero not allowed as denominator
+ invalid denominator

Ran 4 tests in 0.010s
FAILED (failures=1)
>>> |

```

*Figure 6.18*

So, you can see that 4 tests were executed and there was one failure because in the code we have mentioned that if you attempt to divide a number by zero, a string “*invalid denominator*” should be returned. However, the test case asserts that the value of the returned string should be “*zero not allowed as denominator*”. Since the two values are not the same, the test case has failed.

So, now let’s make a change in the `arith_funcs.py` file, and execute the tests again.

```

def add(x,y):
 return x+y

def subtract(x,y):
 return x-y

def multiply(x,y):
 return x*y

```

```

def divide(x,y):
 if(y==0):
 #Change of code:
 str1 = "zero not allowed as denominator"
 return str1
 else:
 return x/y

```

Now, if we execute the tests again, we get the following output:

```

Ran 4 tests in 0.074s
OK

```

*Figure 6.19*

## **6.11 Defining multiple test cases with unittest and pytest:**

You can define multiple tests and execute it with both `unittest` as well as `pytest`:

### **(I) Defining multiple cases with unittest**

```

import arith_funcs
import unittest

class TestArithFuncs(unittest.TestCase):
 #Define Testing Methods

 def test_add(self):
 vals = [(10,20),(20,78),(60,54),(13,25)]
 ans = [30,98,114,38]
 for i in range(len(vals)):
 self.assertEqual(ans[i],arith_funcs.add(vals[i][0],vals[i][1]))

 def test_subtract(self):
 self.assertEqual(-10,arith_funcs.subtract(10,20))

 def test_multiply(self):
 self.assertEqual(200,arith_funcs.multiply(10,20))

 def test_divide(self):
 self.assertEqual("zero not allowed as denominator",arith_funcs.divide(10,0))
 self.assertEqual(2,arith_funcs.divide(10,5))

unittest.main()

```

So, you can see that in the `test_add()` function, we have used lists to create multiple cases. The output is as follows:

```
....

Ran 4 tests in 0.023s

OK
```

*Figure 6.20*

## (II) Defining multiple test cases with pytest

`arith_func.py`

```
def add(x,y):
 return x+y

def subtract(x,y):
 return x-y

def multiply(x,y):
 return x*y

def divide(x,y):
 if(y==0):
 str1 = "invalid denominator"
 return str1
 else:
 return x/y
```

`test_arith_func.py`

```
import arith_funcs

def test_add():
 vals = [(10,20),(20,78),(60,54),(13,25)]
 ans = [30,98,114,38]
 for i in range(len(vals)):
 assert arith_funcs.add(vals[i][0],vals[i][1]) == ans[i]

def test_subtract():
 assert arith_funcs.subtract(10,20) == -10

def test_multiply():
 assert arith_funcs.multiply(10,20) == 200

def test_divide():
 assert arith_funcs.divide(10,0) == "invalid denominator"
 assert arith_funcs.divide(10,5) == 2
```

```

Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>cd C:\Users\MYPC\AppData\Local\Programs\Python\Python38-32
C:\Users\MYPC>pytest F:\testing\test_arith_funcs.py
=====
platform win32 -- Python 3.8.2, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: C:\Users\MYPC\AppData\Local\Programs\Python\Python38-32
collected 4 items

.
=====
4 passed in 0.08s
C:\Users\MYPC\AppData\Local\Programs\Python\Python38-32>

```

*Figure 6.21*

## 6.12 List of Assert methods available in the ‘unittest’ module.

*Table 6.2* displays various assertion methods, and explanation of how they work.

| Assertion Methods                                                                   | Explanation                                                                                                      |
|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| <code>self.assertEqual(expected_Result,<br/>actual_Result,[Message])</code>         | The test passes only if the expected result is equal to the actual result.                                       |
| <code>self.assertNotEqual(expected_Result,<br/>actual_Result,[Message])</code>      | The test passes only if the expected result is not equal to the actual result.                                   |
| <code>self.assertAlmostEqual(expected_Result,<br/>actual_Result,[Message])</code>   | Checks if expected, and actual results are almost equal after rounding off to the certain number decimal places. |
| <code>self.assertTrue(Condition,[Message])</code>                                   | The test passes if the condition is <b>True</b> .                                                                |
| <code>self.assertFalse(Condition,[Message])</code>                                  | The test passes if the condition is <b>False</b> .                                                               |
| <code>self.assertRaises(exception, functionName,<br/>parameter,...parameter)</code> | Tests that the function, functionName, being called with the given parameters raise the given exception.         |

*Table 6.2*

## Conclusion

Testing is a very important part of software development. Developers must test their code from time to time during the process of development so that they are able to develop a bug free application. Knowledge of types of errors and exceptions is important so that you are able to identify where and what type of problem exists in the program. Once a problem is identified, it can be easily fixed.

## Questions and answers

1. Explain the difference between Syntax, Runtime and Logical error.

**Answer:** The difference between syntax, Runtime and Loical errors is summarised in [table 6.3](#).

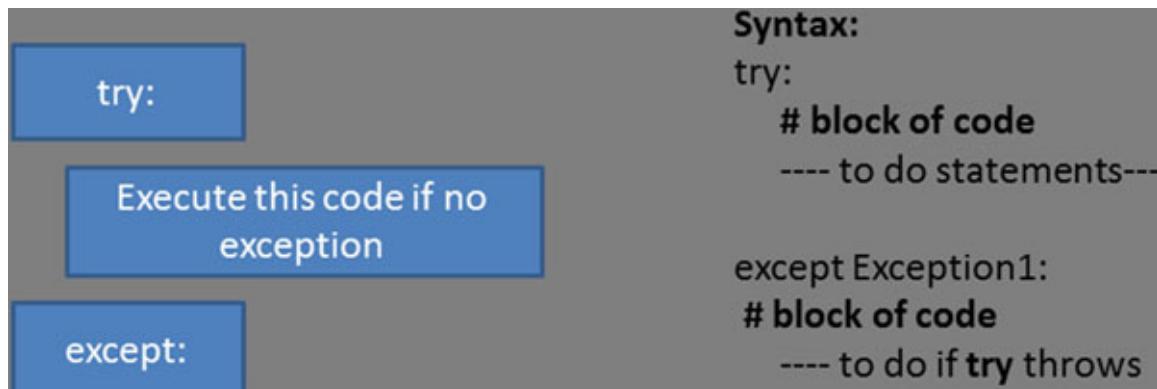
| Summary              | Syntax                                         | Runtime              | Logical                          |
|----------------------|------------------------------------------------|----------------------|----------------------------------|
| <b>Cause</b>         | Wrong syntax, sequence of characters or tokens | Incorrect statements | Implementation of wrong logic    |
| <b>Output</b>        | Unable to compile                              | Unable to execute    | Delivers wrong result            |
| <b>Error Message</b> | Displayed                                      | Displayed            | Not displayed                    |
| <b>Detected</b>      | While parsing code                             | During runtime       | While testing for logical errors |

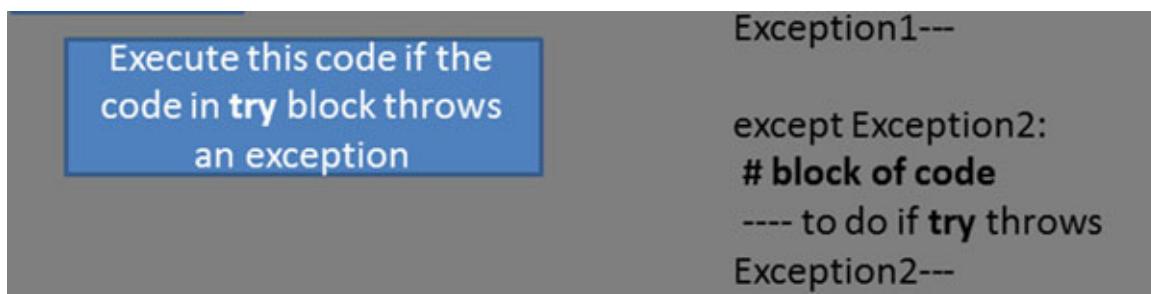
*Table 6.3*

2. Explain with the help of diagram when to use the `try` block.

**Answer:** Whenever you are suspicious about certain lines of your code that they are capable of throwing exception, place it in the try block. The try block must be followed by an except block that has lines of code that must be executed if the code in the try block throw an exception:

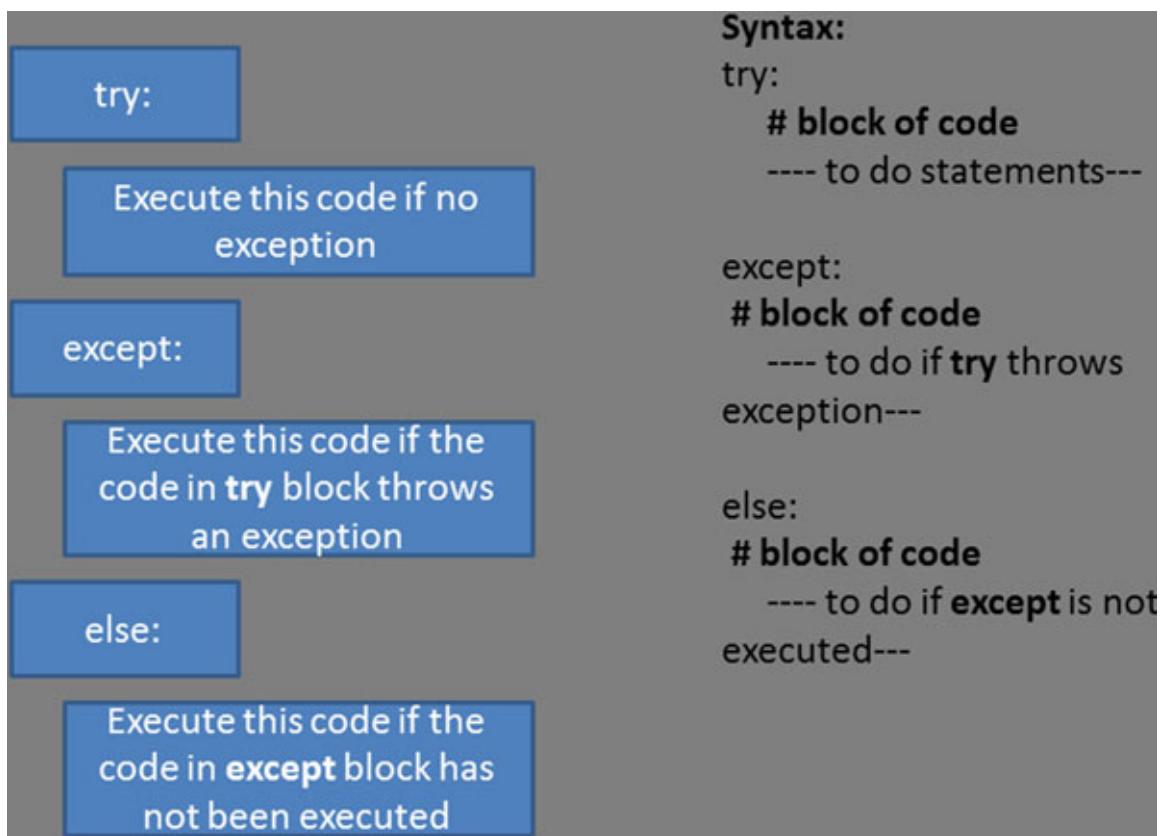
`simple try-except`





*Figure 6.22*

### try...except...else code



*Figure 6.23*

- You can have an `except` block where the name of the exception is not mentioned. You have the option of not specifying the exception name with the `except` statement.
- You can also specify an `else` block along with `try-except` statement, which will be executed if no exception is raised in the `try` block.
- The statements that don't throw the exception must be placed inside the `else` block.

3. Explain how multiple exceptions can be declared.

**Answer:**

**Syntax:**

```
try:
 #block of code
 except (<Exception1>,<Exception2>,<Exception3>,...<Exceptionn>)
 #block of code
 else:
 #block of code

The finally block:
#We can use the finally block with the try block
```

4. Explain how exception can be raised?

**Answer:** The `raise` clause can be used to raise an exception. The syntax is as follows:

```
Syntax:
raise Exception_class<value>
```

- The `raise` statement is used to raise an exception.
- The `raise` statement is followed by the name of the Exception class.

5. Identify the error types:

1. Missing colons
2. Denominator equates to zero
3. Adding instead of subtracting
4. Incorrect format in selection and loop statements
5. Trying to open a file that does not exist
6. Missing Parenthesis, square brackets and curly braces

7. Displaying wrong message
8. Taking data from wrong file

**Answer:**

1. Syntax error
2. Runtime error
3. Logical error
4. Syntax error
5. Runtime error
6. Syntax error
7. Logical error
8. Logical error
  
6. Differentiate between syntax error and runtime error.

**Answer:**

| Syntax Error                                                                                                                                                                                    | Runtime Error                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Caused by grammatically incorrect statement.                                                                                                                                                    | Have grammatically correct statement.                                                                                                                                                 |
| Error occurs while parsing of code.                                                                                                                                                             | Error occurs at the time of execution of code after it has been parsed as grammatically correct.                                                                                      |
| <b>Examples:</b> <ul style="list-style-type: none"> <li>• Missing operator.</li> <li>• Two operators in a line without any intervening semicolon.</li> <li>• Unbalanced parenthesis.</li> </ul> | <b>Examples:</b> <ul style="list-style-type: none"> <li>• Incorrect variable types or sizes.</li> <li>• Non-existent variables.</li> <li>• Out of range errors, and so on.</li> </ul> |

*Table 6.4*

7. Fill in the blanks

1. \_\_\_\_\_ errors occur when syntax is alright but there is a problem while executing the code.
2. \_\_\_\_\_ halts the interpreter, reports the error, but does not execute the program.
3. For \_\_\_\_\_ error, the interpreter executes the program but stops when it encounters an error and reports it as an “Exception”.

4. For \_\_\_\_\_ error, the interpreter runs the program and does not report an error.

**Answer:**

1. RuntimeErrors
  2. Syntax error
  3. Runtime
  4. Logical/ Semantic
8. What error will the following code generate?

```
>>> list1 = ['a', 'b', 'c', 'd']
>>> list1[5]
```

- a. ZeroDivisionError
- b. ValueError
- c. IndexError
- d. KeyError

**Answer:** (c)

9. What error will the following code generate?

```
>>> dict1 = {1:100,2:200,3:300}
>>>dict1[4]
```

**Answer:** KeyError

**Question:** What error will the following code generate?

```
from math import add
```

**Answer:** ImportError

**Question:** What error will the following code generate?

```
>>> a= 1
>>> b = 'a'
>>>a+b
```

**Test your knowledge.**

**Question:** Look at the following code:

```
def what_day(day):
 if day in range(1,32):
```

```
 print("Valid day")
else:
 raise ValueError("Invalid day value")
```

What would be the output for:

- i. what\_day(0)
- ii. what\_day(12)
- iii. what\_day(32)

**Answer:**

- i. **ValueError**: Invalid day value
- ii. Valid day
- iii. **ValueError**: Invalid day value

**Question:** What would be the output of the following code:

```
def just_having_fun():
 try:
 return 45
 except TypeError as e:
 print(str(e))
 finally:
 return("Ok Bye")

jhf = just_having_fun()
print(jhf)
```

**Answer:** Ok Bye

# CHAPTER 7

## Data Visualization and Data Analysis

### Introduction

Data visualization makes it easier for the human brain to understand the patterns, and trends present in the data irrespective of its size. A lot of crucial information is hidden in these patterns, which can be of great benefit to the company because analysis of these hidden patterns bring into light those areas of business that require attention and must be improved. In this chapter, you will learn how to carry out data visualization, and data analysis using Python.

### Structure

- Introduction to Data Visualization
  - Introduction to Data Visualization
  - Why data visualization
- Matplotlib
  - Working with PyPlot
  - Plotting a point
  - Plotting multiple points
  - Plotting a line
  - Labelling the x and y axis
- Numpy
  - Installing numpy
  - Shape of numpy arrays
  - How to get value of elements from Array
  - Creating numpy Arrays

- Pandas
- Operations on Dataframe

## Objectives

After reading this chapter, you will be able to work with:

- Matplotlib
- Numpy
- Pandas
- Dataframe

### 7.1 Introduction to Data Visualization

No matter how complex the data sets are, good visualization helps in giving it a meaning. However, you must know that the concept of visualization is not new. Not long back, it was a common trend to take information from the Excel sheet, and use the information to create graphs, pie charts, and so on. However, the concepts of data visualization are much more advanced. More elaborate visualization techniques such as bubble cloud, and radial trees have come into existence.

We live in the world of competitive environment where companies are competing with each other. In order to remain on top of the game, it is important to know the connections between operations and overall business performance. This is where data visualization is required.

**Note:** Presentation of data in pictorial or graphical formats is known as **data visualization**. Data visualization helps in faster action as the graphs and charts are able to summarize complex data very efficiently. One need not scroll through thousands of lines of data instead with just one look at the graphs the human brain is able to process the information and take the right decisions.

#### 7.1.1 Why Data Visualization?

The key reasons why data visualization is required are as follows:

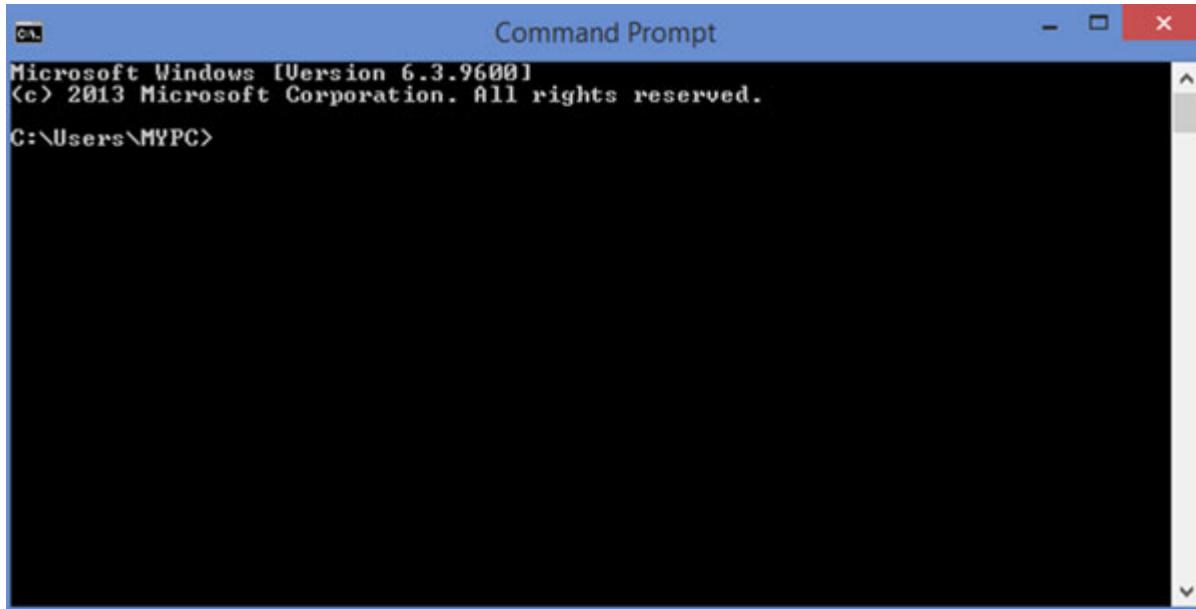
- When information is displayed in graphical or pictorial form, the human brain is able to process well. Data visualization helps in understanding the trends well, and helps you take better decisions.
- Data visualization helps in identifying the market shifts, and trends in the large data sets.
- With data visualization, not only can you understand the trends well but you can also experiment, and see how the outcome will be affected if changes are made to some variables.
- You can also identify which all variables contribute to improving the processes, and which all variables are completely redundant and not required. So, you can let go of all the extra information that you have with yourself.
- One is able to detect the loop holes or the areas that require improvement and bring about those changes in the system accordingly. Data visualization helps in identifying which opportunities are valuable and which ones are risky.
- Data visualization helps in viewing how data trends over time.
- It gives information about how the important events take place over time.
- Correlations between various variables of a data set are best understood when the information is produced in pictorial or graphical form.

## 7.2 Matplotlib

Matplotlib is a Python's library, equivalent of MATLAB. It is a Python package that is more of plotting library used for generating high-quality 2D graphics such as graphs (histogram, bar plot), charts and figures. Matplotlib consists of several interfaces, out of which `PyPlot` is the one that we will work with in this chapter.

Since `PyPlot` exists in the Matplotlib library, it is important to install `matplotlib` in order to work with it. Follow the steps below to install the `matplotlib` library:

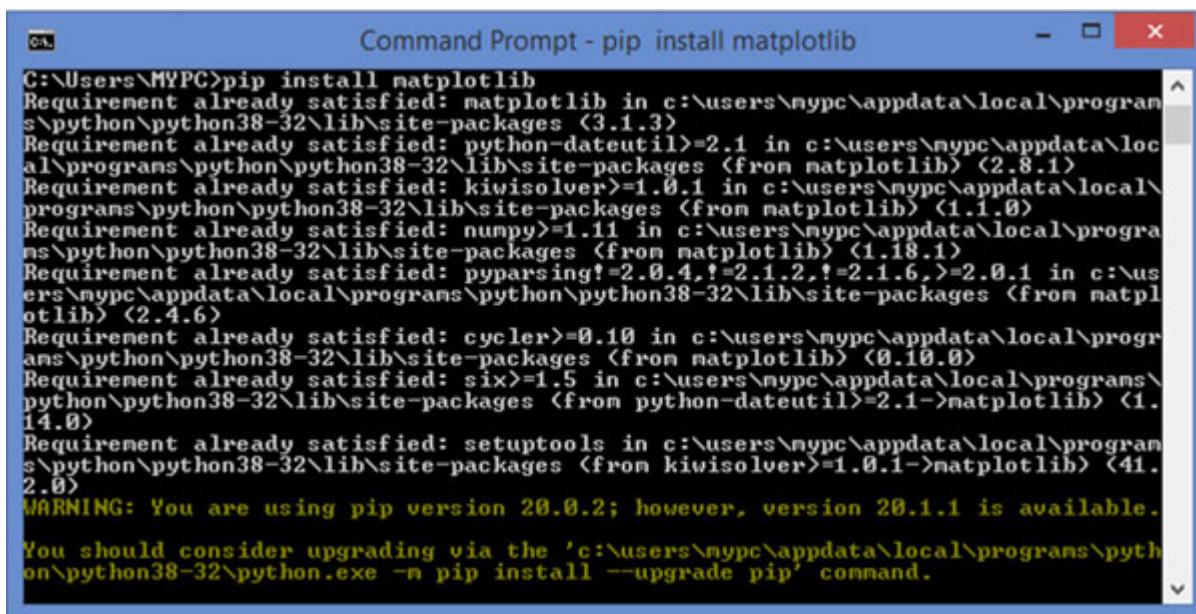
Go to Command Prompt.



*Figure 7.1*

Now, give the following command:

```
pip install Matplotlib
```



*Figure 7.2: Installing matplotlib*

Wait till the installation is completed. You are now ready to work with **Matplotlib**.

## 7.2.1 Working with pyplot

**PyPlot** is a module in **Matplotlib**. It is a framework used to generate 2D graphics and is often utilized in Python scripts, shell, web application and other graphical user interface toolkits.

In order to use PyPlot in Python program, it is important to first import it in our program using:

```
import matplotlib.pyplot as plt
```

In the preceding statement, the name “**plt**” can be anything –x, y, z, p1, and so on, but in this book, we are using the name “**plt**” and for the ease of understanding in all the examples that follow, we will keep the same name. However, please feel free to experiment; use a different name, it will work the same way.

Look at the following statement:

```
import matplotlib.pyplot as plt
```

When we say `import matplotlib.pyplot as plt` we are actually importing module `pyplot` from the library `Matplotlib` and then binding the name “**plt**” to it.

## 7.2.2 Plotting a point

Let's start with a simple program of plotting a point say (5,8) on the graph. This requires four simple steps:

1. Importing the `pyplot` module from the `matplotlib` library.

```
import matplotlib.pyplot as plt
```

2. Define the values for x(=5) and y(=8).

```
x = 5
y = 8
```

3. Call the `plot(x,y)` function. This function takes two arguments. ‘**x**’ stands for value of x-axis and ‘**y**’ stands for value on y-axis.

```
plt.plot(x,y)
```

#### 4. Display the plot.

```
plt.show()
```

The final code looks something like this:

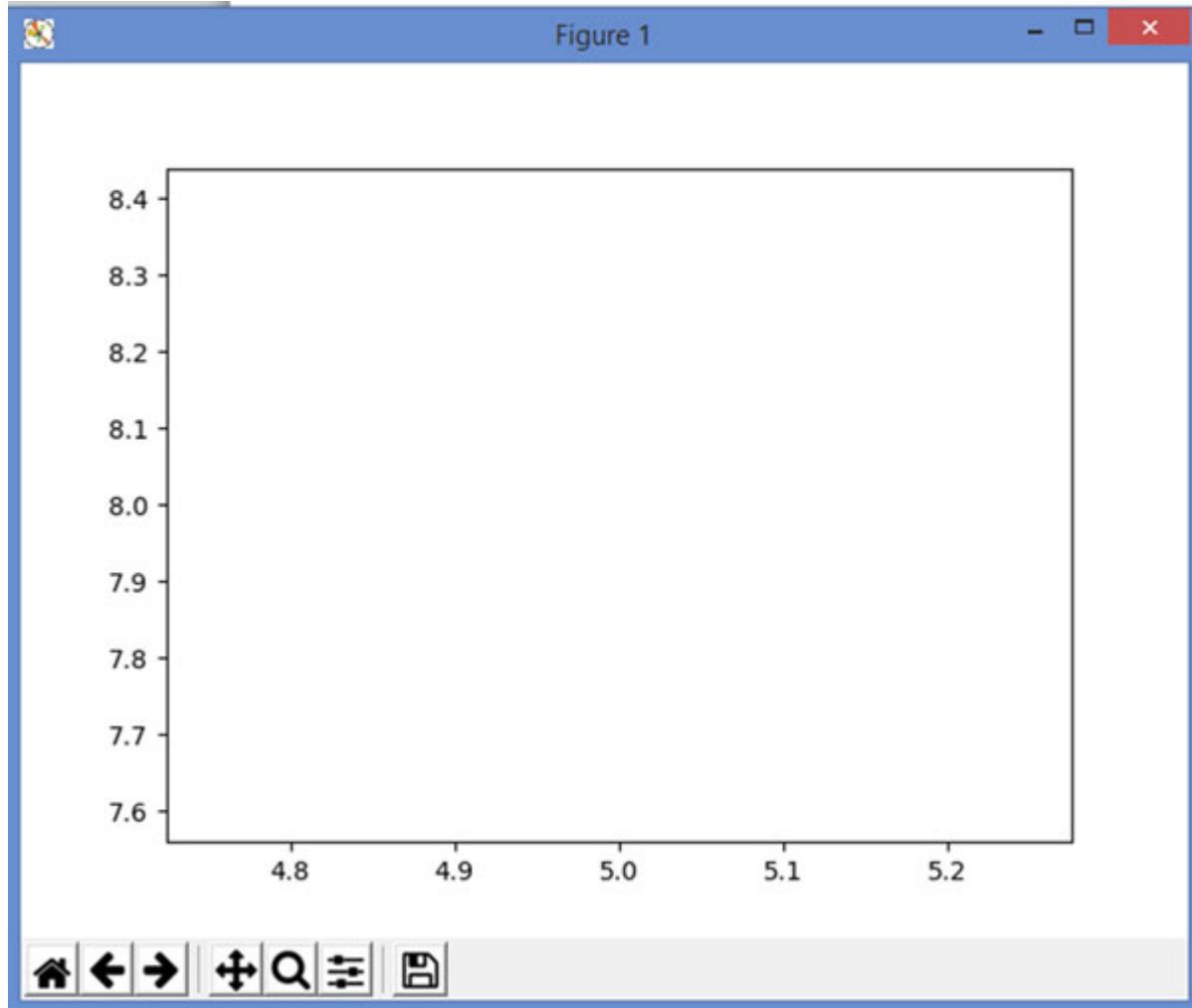
```
#import pyplot
import matplotlib.pyplot as plt

#define the values of x and y
x = 5
y = 8

#plot the point with the help of plot(x,y) function
plt.plot(x,y)

#display the plot
plt.show()
```

The output will be as follows:



*Figure 7.3*

You are likely to be disappointed as the point will not be visible. This can be fixed by defining the color and style for the plot. [Table 7.1](#) displays the color that can be used with the plot.

| Character | Color               |
|-----------|---------------------|
| 'b'       | Blue(default color) |
| 'g'       | Green               |
| 'r'       | Red                 |
| 'c'       | Cyan                |
| 'm'       | Magenta             |
| 'y'       | Yellow              |
| 'k'       | Black               |
|           |                     |

|     |       |
|-----|-------|
| 'w' | White |
|-----|-------|

**Table 7.1**

The plot styles are defined in [table 7.2](#):

| Character | Style                 |
|-----------|-----------------------|
| '-        | solid line style      |
| '--'      | dashed line style     |
| '-.'      | dash-dot line style   |
| '..'      | dotted line style     |
| '.'       | point marker          |
| ','       | pixel marker          |
| 'o'       | circle marker         |
| 'v'       | triangle_down marker  |
| '^'       | triangle_up marker    |
| '<'       | triangle_left marker  |
| '>'       | triangle_right marker |
| '1'       | tri_down marker       |
| '2'       | tri_up marker         |
| '3'       | tri_left marker       |
| '4'       | tri_right marker      |
| 's'       | square marker         |
| O         | Circle                |
| 'p'       | pentagon marker       |
| '*'       | star marker           |
| 'h'       | hexagon1 marker       |
| 'H'       | hexagon2 marker       |
| '+'       | plus marker           |
| 'x'       | x marker              |
| 'D'       | diamond marker        |
| 'd'       | thin_diamond marker   |
| ' '       | vline marker          |
| '_'       | Hline marker          |
| V         | Down triangle         |

|   |                |
|---|----------------|
| ^ | Up triangle    |
| < | Left triangle  |
| > | Right triangle |
| S | Square         |
| P | Pentagon       |
| H | Hexagon        |

**Table 7.2**

Now, keeping the information provided in [table 7.1](#) and [table 7.2](#), let's try to plot the point as a red coloured cross. For this while plotting, we will call the plot function as:

```
plot(x,y,'rx')
```

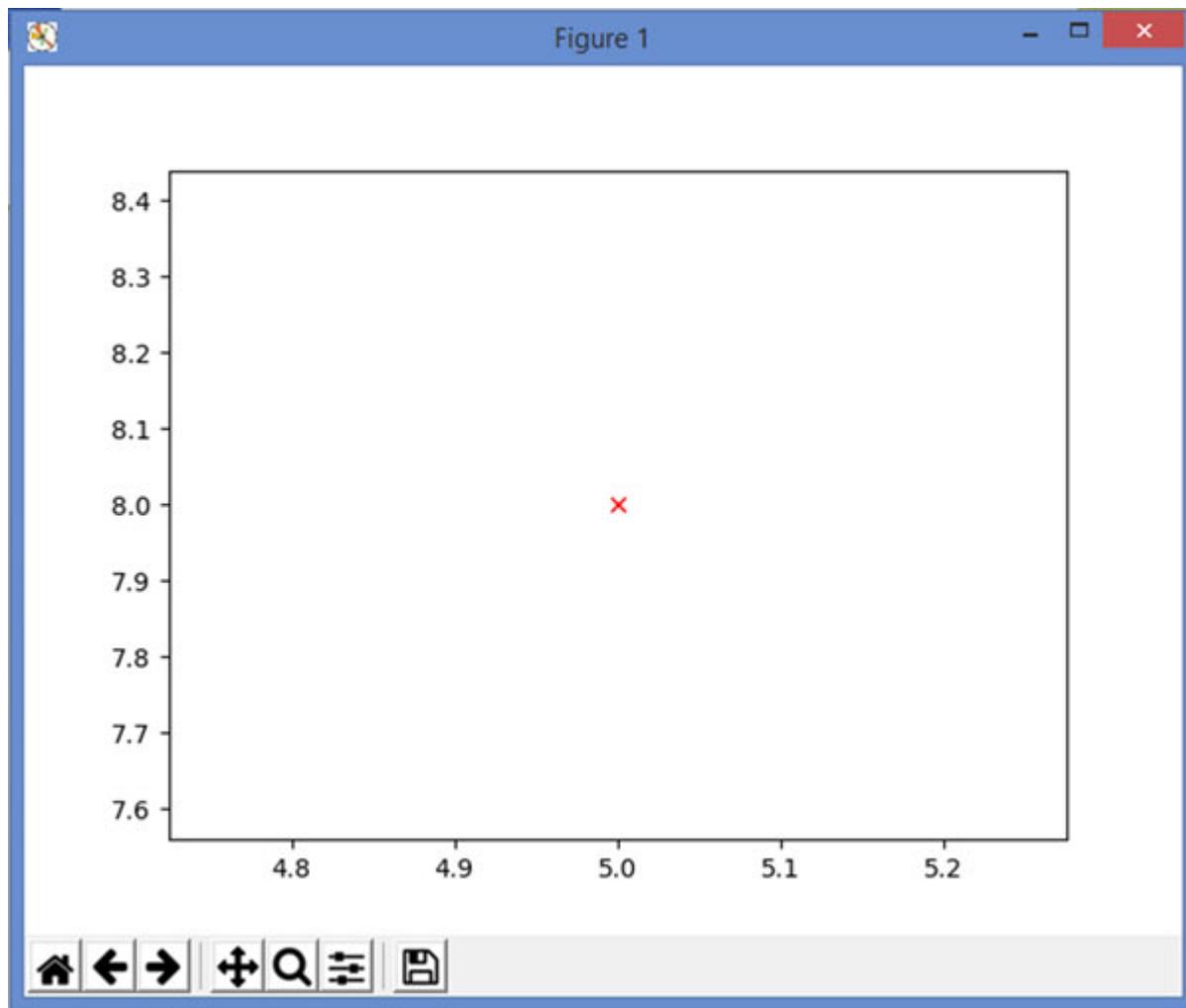
The code will now look as follows:

```
#import pyplot
import matplotlib.pyplot as plt

#define the values of x and y
x = 5
y = 8

#plot the point with the help of plot(x,y) function, red color, marker style x
plt.plot(x,y,'rx')

#display the plot
plt.show()
```



*Figure 7.4*

You can now see the point clearly.

While plotting a point, step2 can be skipped. Since only one value of **x** and **y** axis is used, the values can be directly placed in the **plot** function, and the result will be the same.

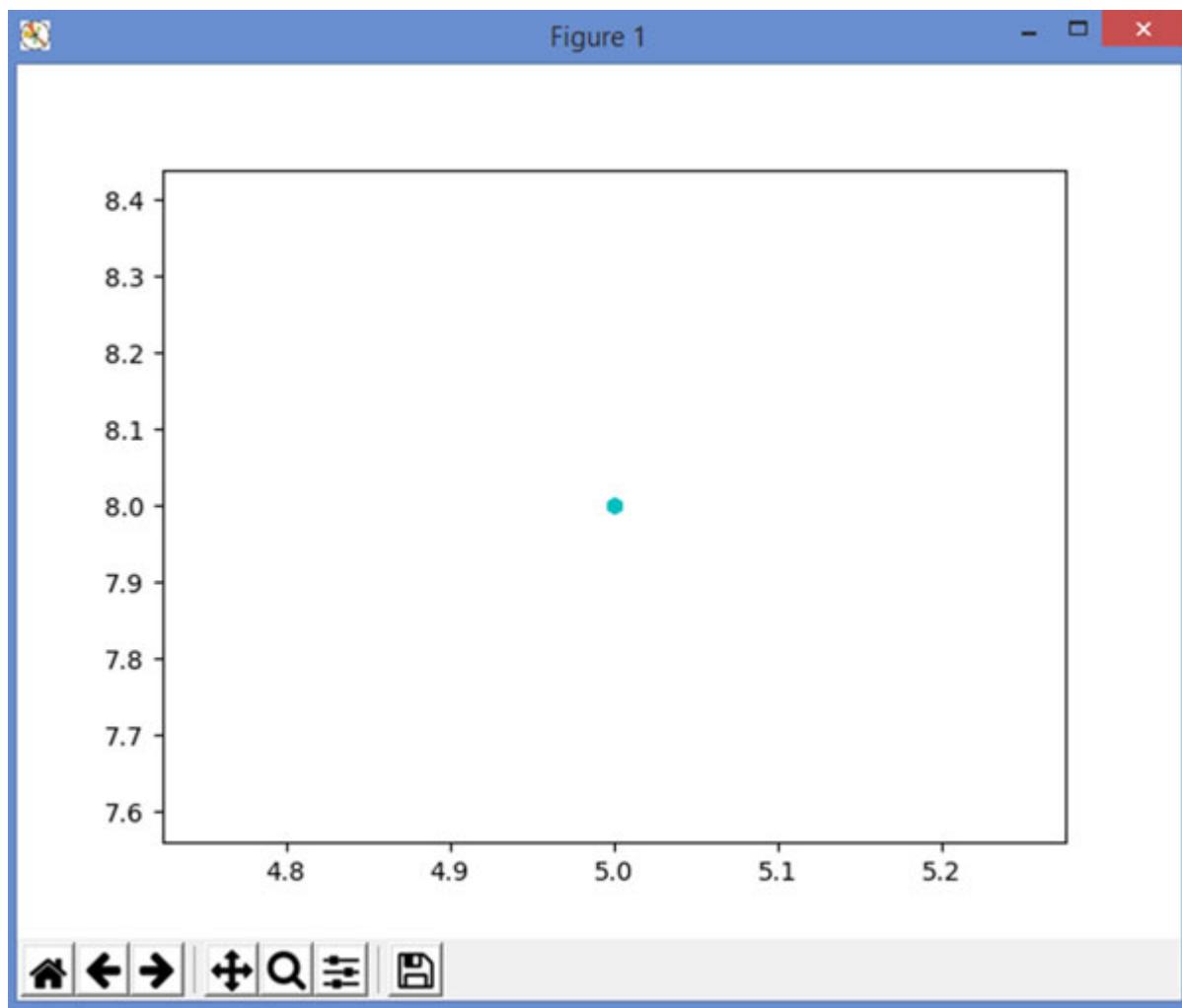
In the following code, we have skipped step 2, and directly provided the values to the **plot(x,y)** function and it works fine.

```
#import pyplot
import matplotlib.pyplot as plt

#plot the point with the help of plot(5,8) function, cyan color, marker #style
hexagon
plt.plot(5,8,'ch')

#display the plot
```

```
plt.show()
```



*Figure 7.5*

### 7.2.3 Plotting multiple points

Look at the following dataset:

|   |    |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|----|
| X | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
| Y | 7  | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |

*Table 7.3*

Suppose you have to plot all points for the data set given above. How will you do it? The following steps are involved in plotting multiple points:

1. Importing the `pyp1ot` module from the matplotlib library.

```
import matplotlib.pyplot as plt
```

2. Define two lists for the values of **x** and **y** as shown as follows:

```
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]
```

3. Call the **plot(x,y)** function. Also, provide the color and style for the point. Here we are going for green square point.

```
plt.plot(x,y, 'gs')
```

4. Display the plot.

```
plt.show()
```

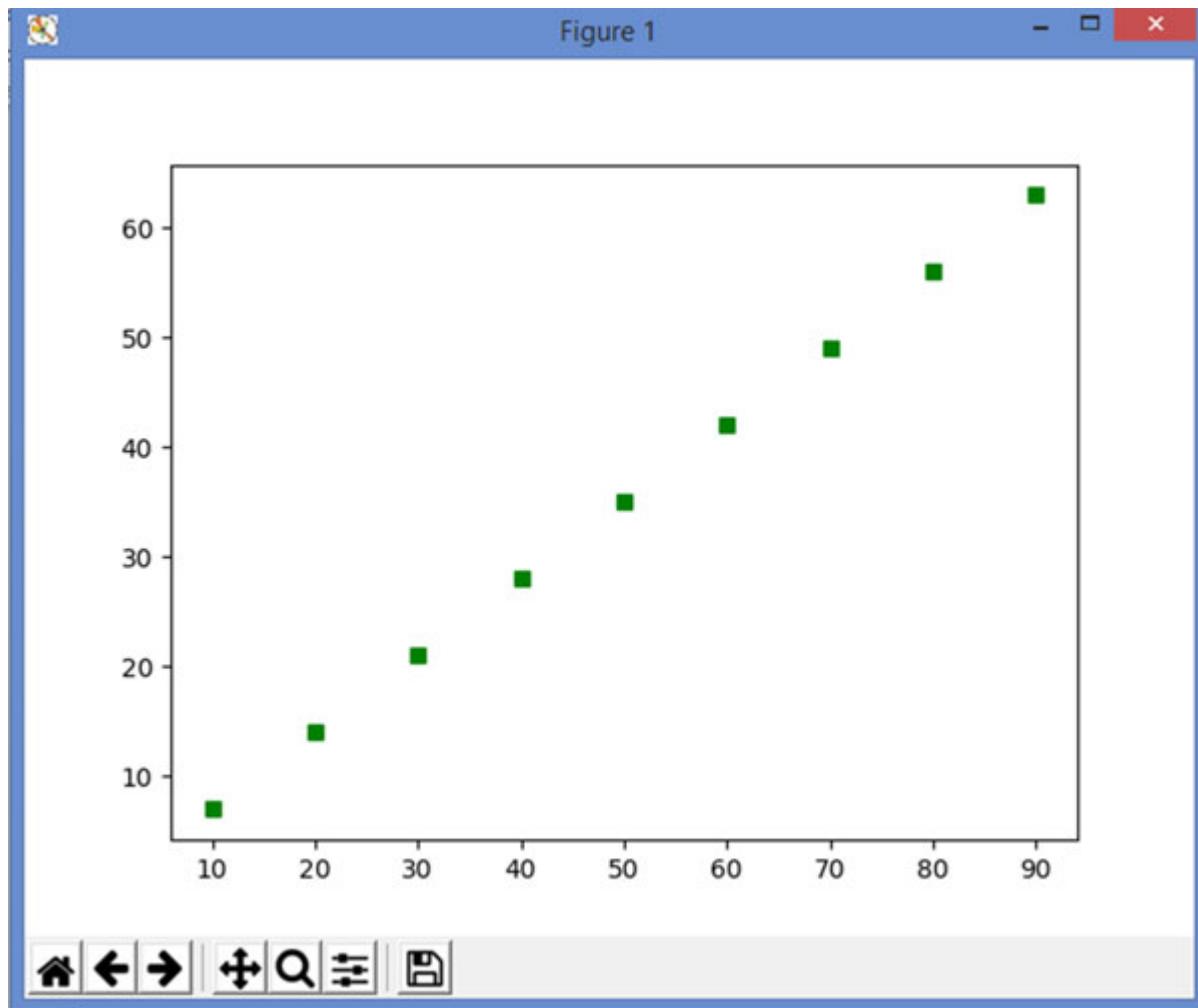
## Code:

```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt

#define two lists for the values of x and y
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function. Also provide the color and style for the #point.
#Here we are going for green square point.
plt.plot(x,y, 'gs')

#Display the plot
plt.show()
```



*Figure 7.6*

#### 7.2.4 Plotting a line

Plotting a line is similar to plotting multiple points as you will see shortly. Here we just skip the part of defining the color, and the marker style for the point. We take the same dataset that we used for the previous example:

| X | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|----|----|----|----|----|----|----|----|----|
| Y | 7  | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |

*Table 7.4*

To plot a line for the dataset given above, follow the steps given as follows:

1. Importing the `pypplot` module from the `matplotlib` library.

```
import matplotlib.pyplot as plt
```

2. Define two lists for the values of x and y as shown here:

```
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]
```

3. Call the `plot(x,y)` function.(Do not define the color and marker style.)

```
plt.plot(x,y)
```

4. Display the plot.

```
plt.show()
```

## Code:

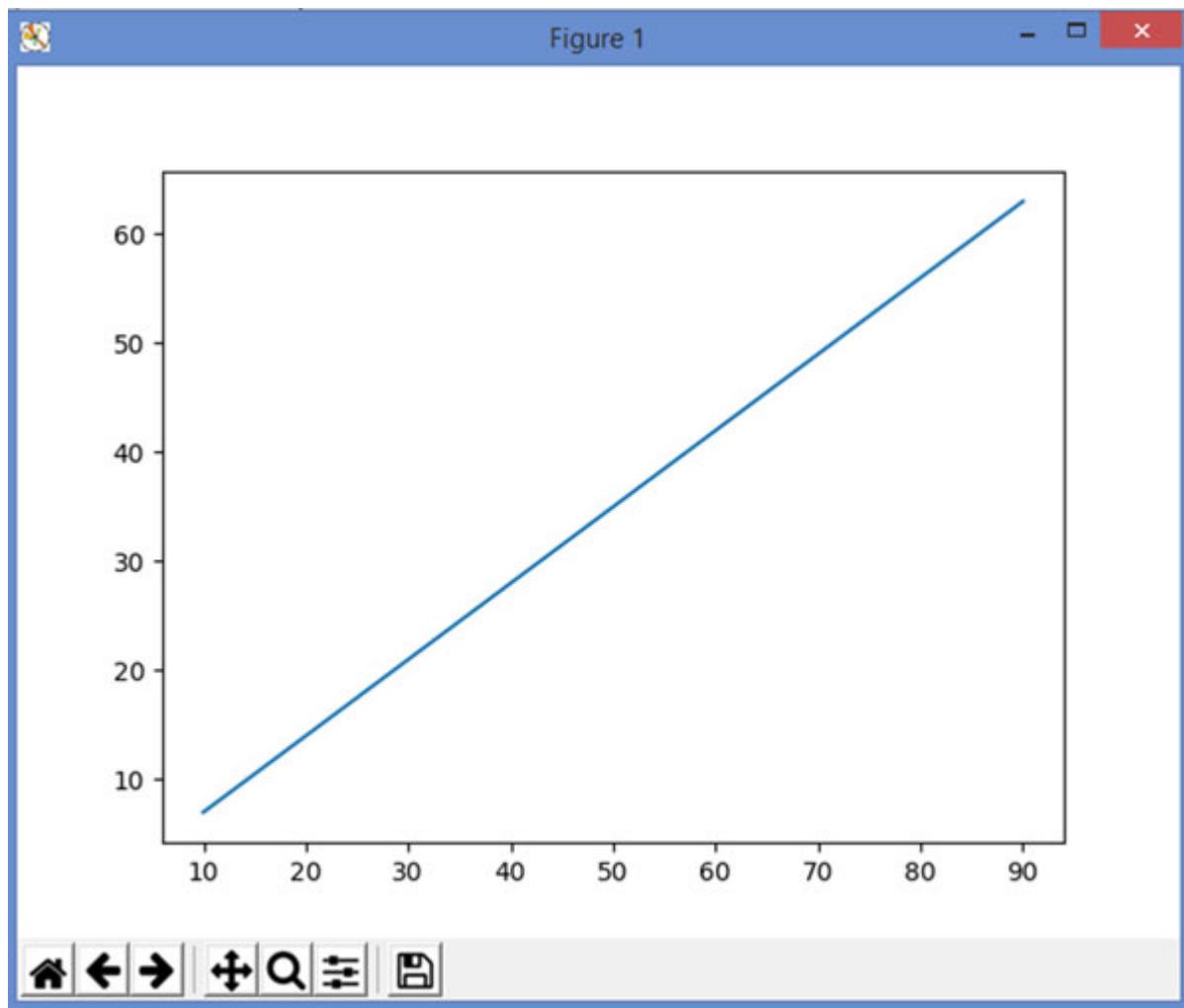
```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt

#define two lists for the values of x and y
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function.
plt.plot(x,y)

#Display the plot
plt.show()
```

## Output:



*Figure 7.7*

Specifiers for lines are different from specifiers of point. Some of the specifiers are as follows:

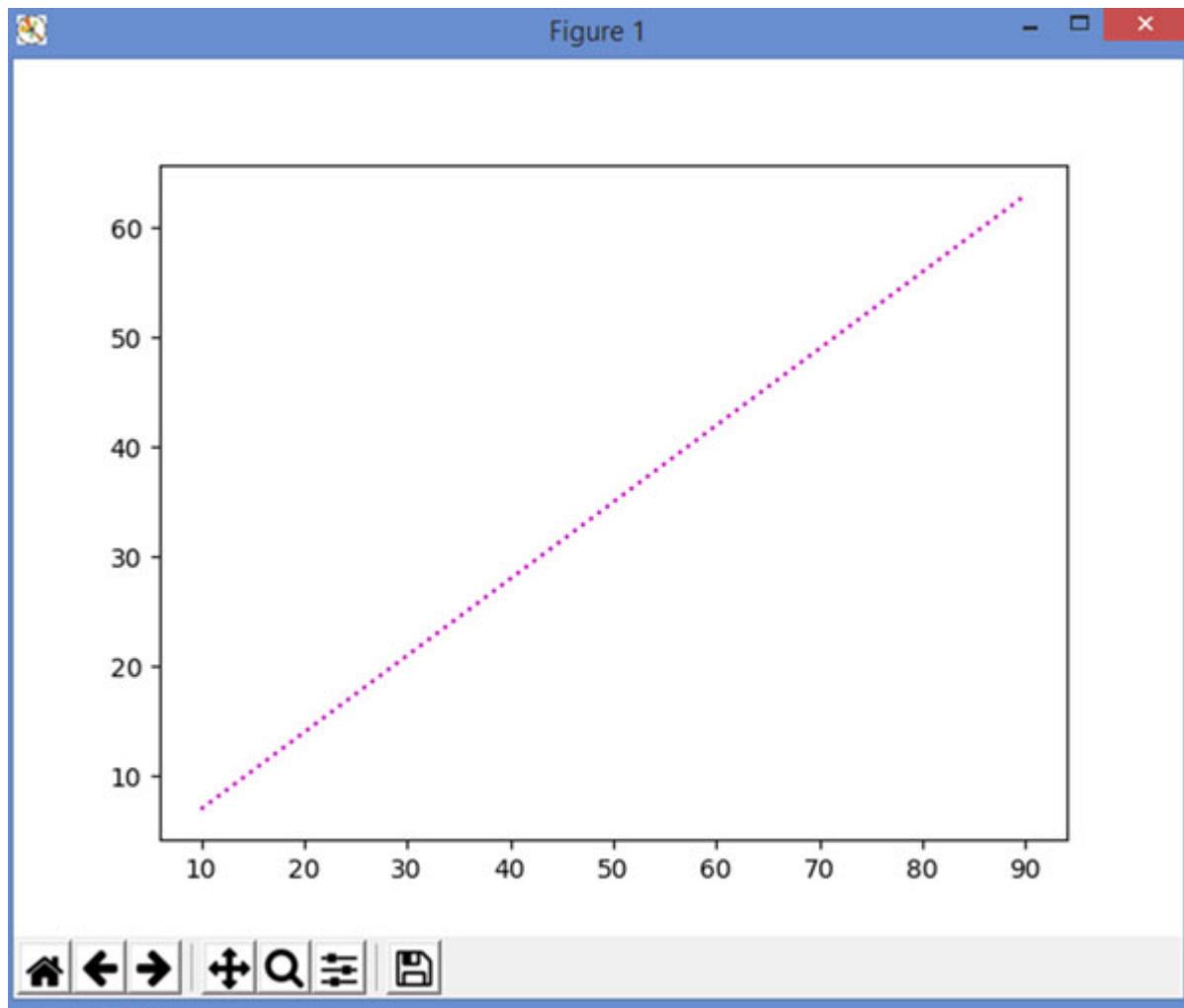
‘-’ : solid line

‘--’ : dashed line

‘..’ : dotted line

‘-.’ : dash-dot line

We now make slight changes to the preceding example, and try to plot a magenta-dotted line.



*Figure 7.8*

If you want to show both point as well as the line then you can specify color, markerstyle, and the line style.

### Code:

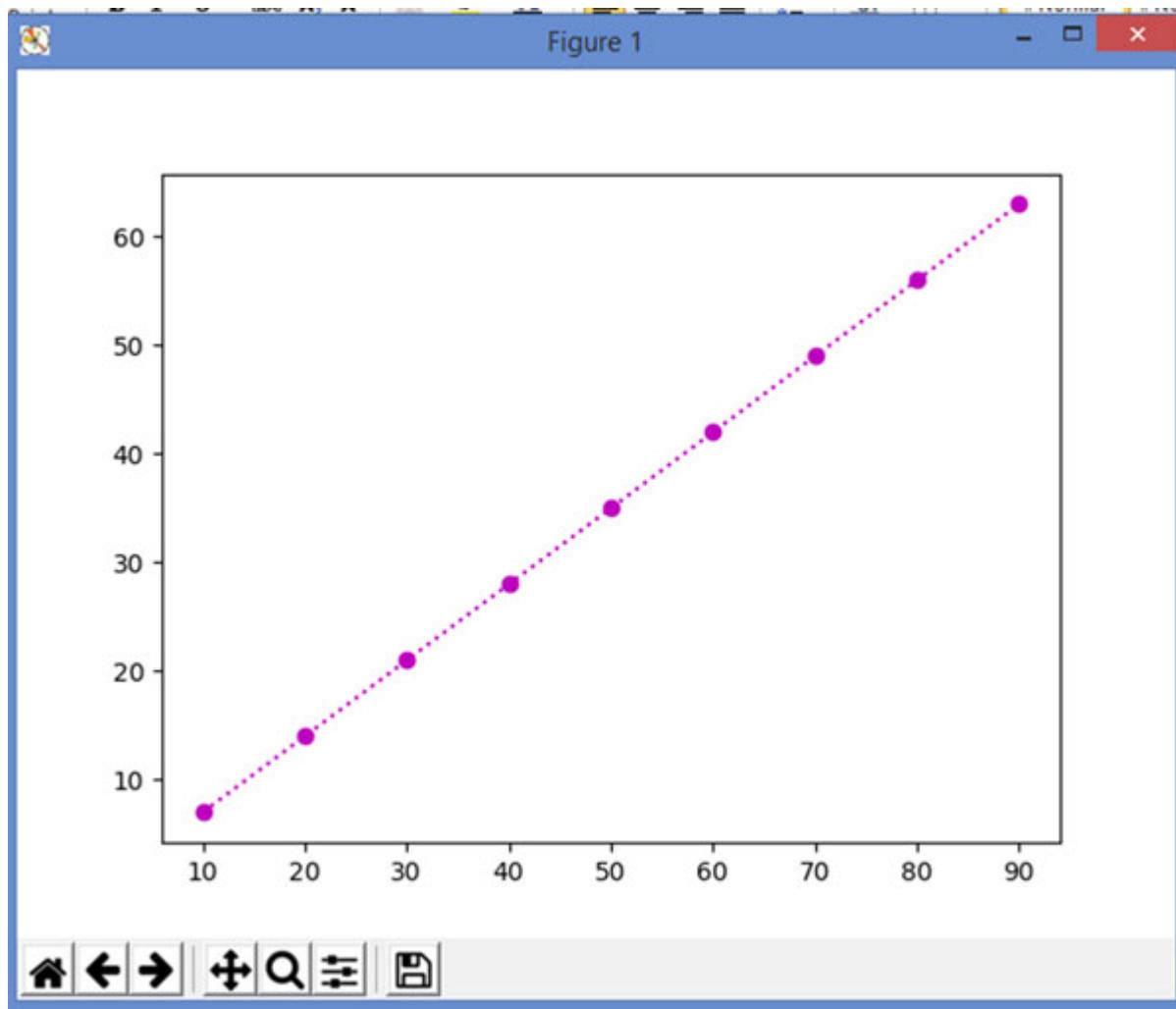
```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt

#define two lists for the values of x and y
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function. Magenta color,circle points on dotted line
plt.plot(x,y,"mo:")

#Display the plot
plt.show()
```

## Output:



*Figure 7.9*

You can also define the size of the point(markersize) and width of the line(linewidth).

## Code:

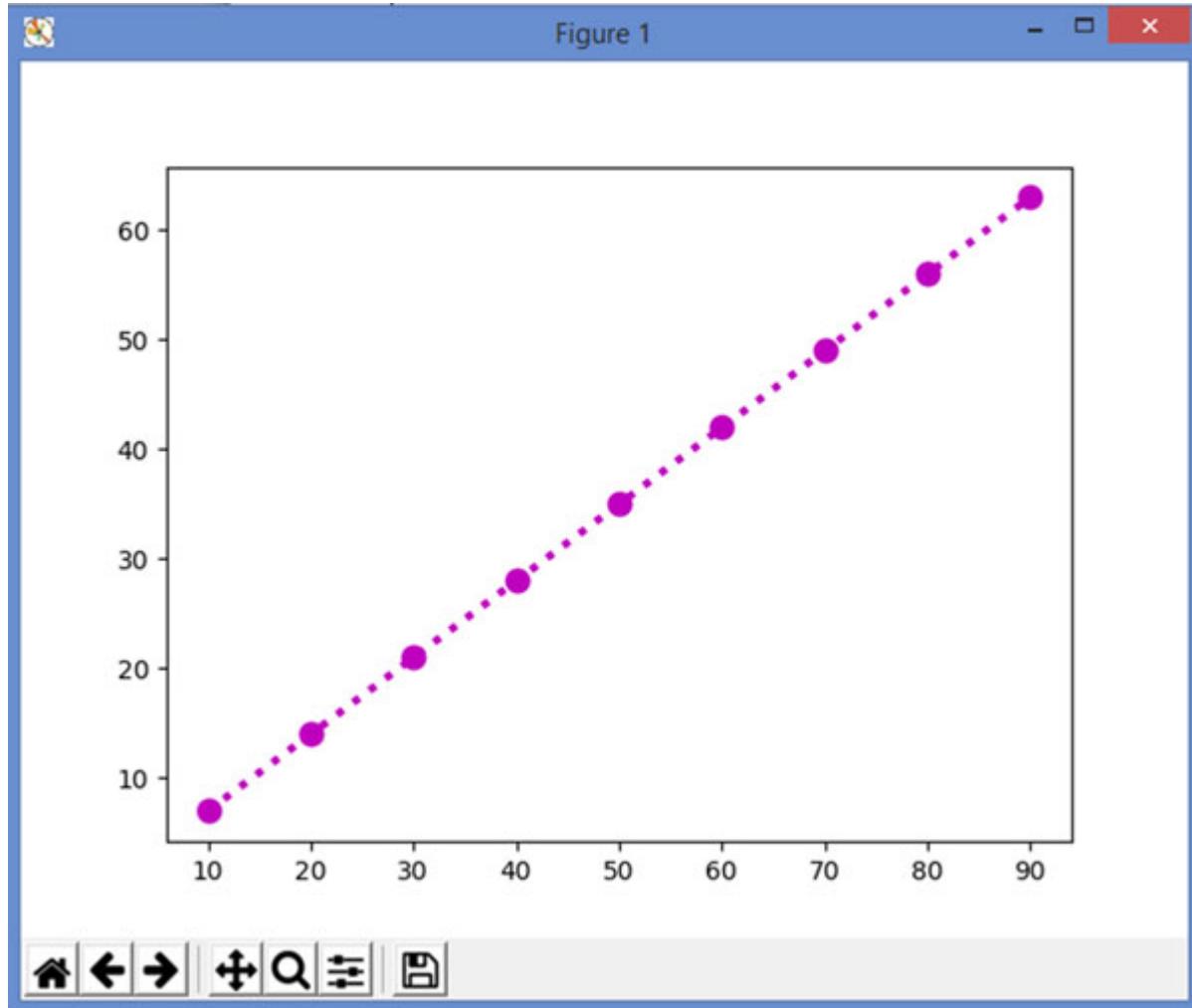
```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt

#define two lists for the values of x and y
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function. Magenta color,circle points on dotted line
plt.plot(x,y,"mo:", markersize = 9, linewidth = 3)
```

```
#Display the plot
plt.show()
```

## Output:



*Figure 7.10*

If you want to display a grid in the background, use the `grid()` function as shown in the following code:

```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt

#define two lists for the values of x and y
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function. Magenta color,circle points on dotted line
plt.plot(x,y,"mo:", markersize = 9, linewidth = 3)
```

```
#Display the grid
plt.grid()

#Display plot
plt.show()
```

## Output:

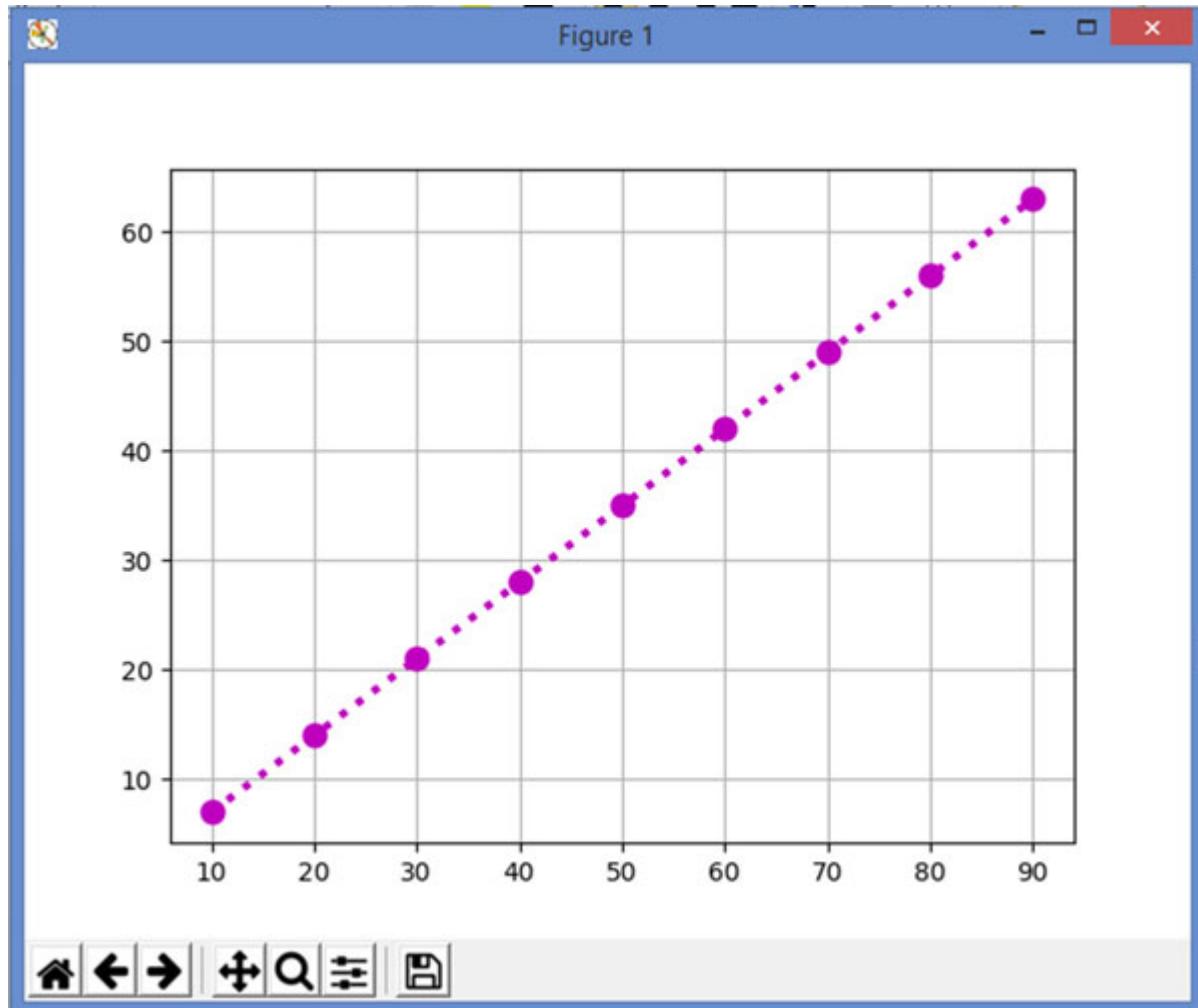


Figure 7.11

### 7.2.5 Labelling the x and y axis

Functions `xlabel`, `ylabel` are used for labelling the `x` and `y` axis and function title is used to give a title to the plot.

The following example shows how this can be done:

## Code:

```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt

#define two lists for the values of x and y
x = [10,20,30,40,50,60,70,80,90]
y = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function
plt.plot(x,y)

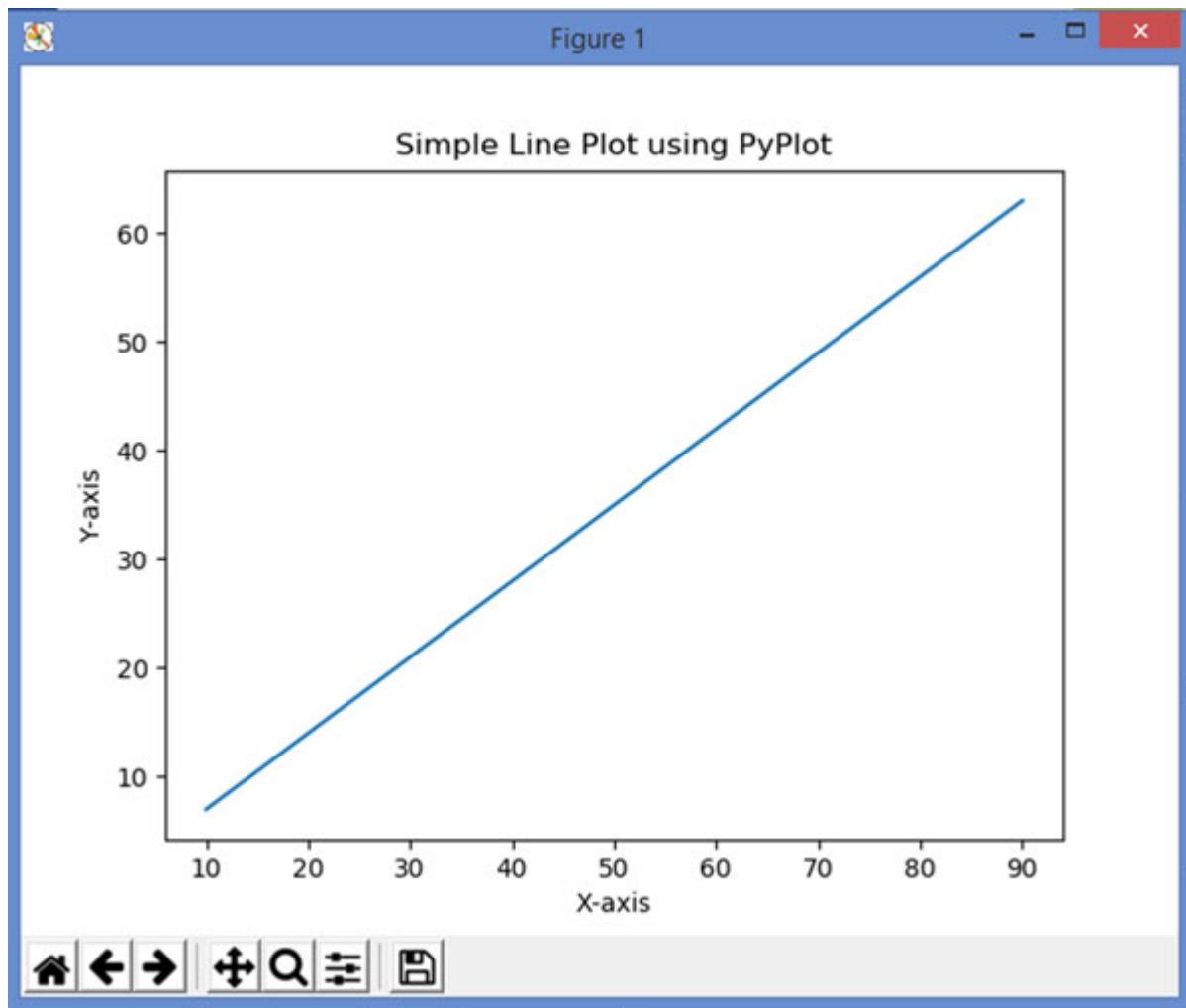
#Label the x-axis
plt.xlabel('X-axis')

#Label the y-axis
plt.ylabel('Y-axis')

#Give a title to the graph
plt.title('Simple Line Plot using PyPlot')

#Display the plot
plt.show()
```

## Output:



*Figure 7.12*

In the next program, you will learn how to plot two lines on the same plot. Let's say we have two datasets.

### **Dataset (I)**

| X1 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|----|----|----|----|----|----|----|----|----|----|
| Y1 | 7  | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |

*Table 7.5*

### **Dataset (II)**

| X | 7  | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |
|---|----|----|----|----|----|----|----|----|----|
| Y | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |

**Table 7.6**

The steps involved are as follows:

**Step 1:** Import the `pypplot` module from the `matplotlib` library.

```
import matplotlib.pyplot as plt
```

**Step 2:** Define two lists for the values of `x1` and `y1`.

```
x1 = [10,20,30,40,50,60,70,80,90]
y1 = [7,14,21,28,35,42,49,56,63]
```

**Step 3:** Call the `plot(x,y)` function for values of `x1` and `y1`.

```
plt.plot(x1,y1)
```

**Step 4:** Define two lists for the values of `x2` and `y2`.

```
x2 = [7,14,21,28,35,42,49,56,63]
y2 = [10,20,30,40,50,60,70,80,90]
```

**Step 5:** Call the `plot(x,y)` function for values of `x2` and `y2`.

```
plt.plot(x2,y2)
```

**Step 6:** Label the x-axis.

```
plt.xlabel('X-axis')
```

**Step 7:** Label the y-axis.

```
plt.ylabel('Y-axis')
```

**Step 8:** Give a title to the graph.

```
plt.title('Plotting two lines on the same graph using PyPlot')
```

**Step 9:** Display the plot.

```
plt.show()
```

## Code:

```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt

#define two lists for the values of x1 and y1
x1 = [10,20,30,40,50,60,70,80,90]
y1 = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function for values of x1 and y1
plt.plot(x1,y1)

#define two lists for the values of x2 and y2
x2 = [7,14,21,28,35,42,49,56,63]
y2 = [10,20,30,40,50,60,70,80,90]

#Call the plot(x,y) function for values of x2 and y2
plt.plot(x2,y2)

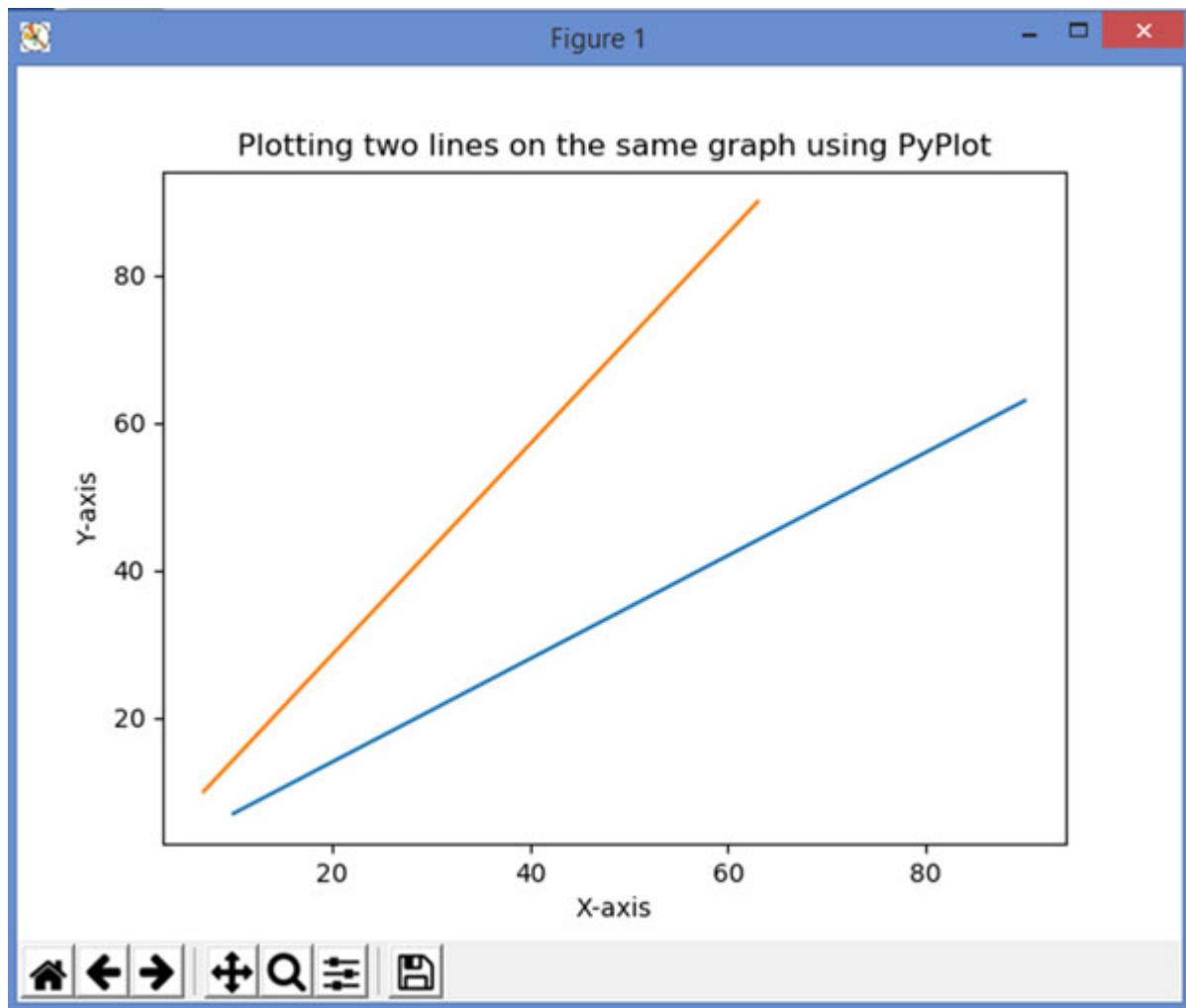
#Label the x-axis
plt.xlabel('X-axis')

#Label the y-axis
plt.ylabel('Y-axis')

#Give a title to the graph
plt.title('Plotting two lines on the same graph using PyPlot')

#Display the plot
plt.show()
```

## Output:



*Figure 7.13*

If you look at this plot, you will realize that although the plot is well labelled, it is difficult to make out which line represents which dataset. We can make use of the `legend()` function to clarify which plot is for which function.

A `legend()` function creates a small rectangular box that provides information about which line represents which data set. For this, while calling the plot function for each dataset we will define a label for that plot. When we invoke the legend function, the type of line will be displayed along with the label of the related data set. For better understanding, have a look at the code shown as follows:

```
#import the pyplot module from matplotlib library
import matplotlib.pyplot as plt
```

```
#define two lists for the values of x1 and y1
x1 = [10,20,30,40,50,60,70,80,90]
y1 = [7,14,21,28,35,42,49,56,63]

#Call the plot(x,y) function for values of x1 and y1 and assign it a label
plt.plot(x1,y1, label='Line for dataset x1, y1')

#define two lists for the values of x2 and y2
x2 = [7,14,21,28,35,42,49,56,63]
y2 = [10,20,30,40,50,60,70,80,90]

#Call the plot(x,y) function for values of x2 and y2 and assign it a label
plt.plot(x2,y2, label='Line for dataset x2, y2')

#Label the x-axis
plt.xlabel('X-axis')

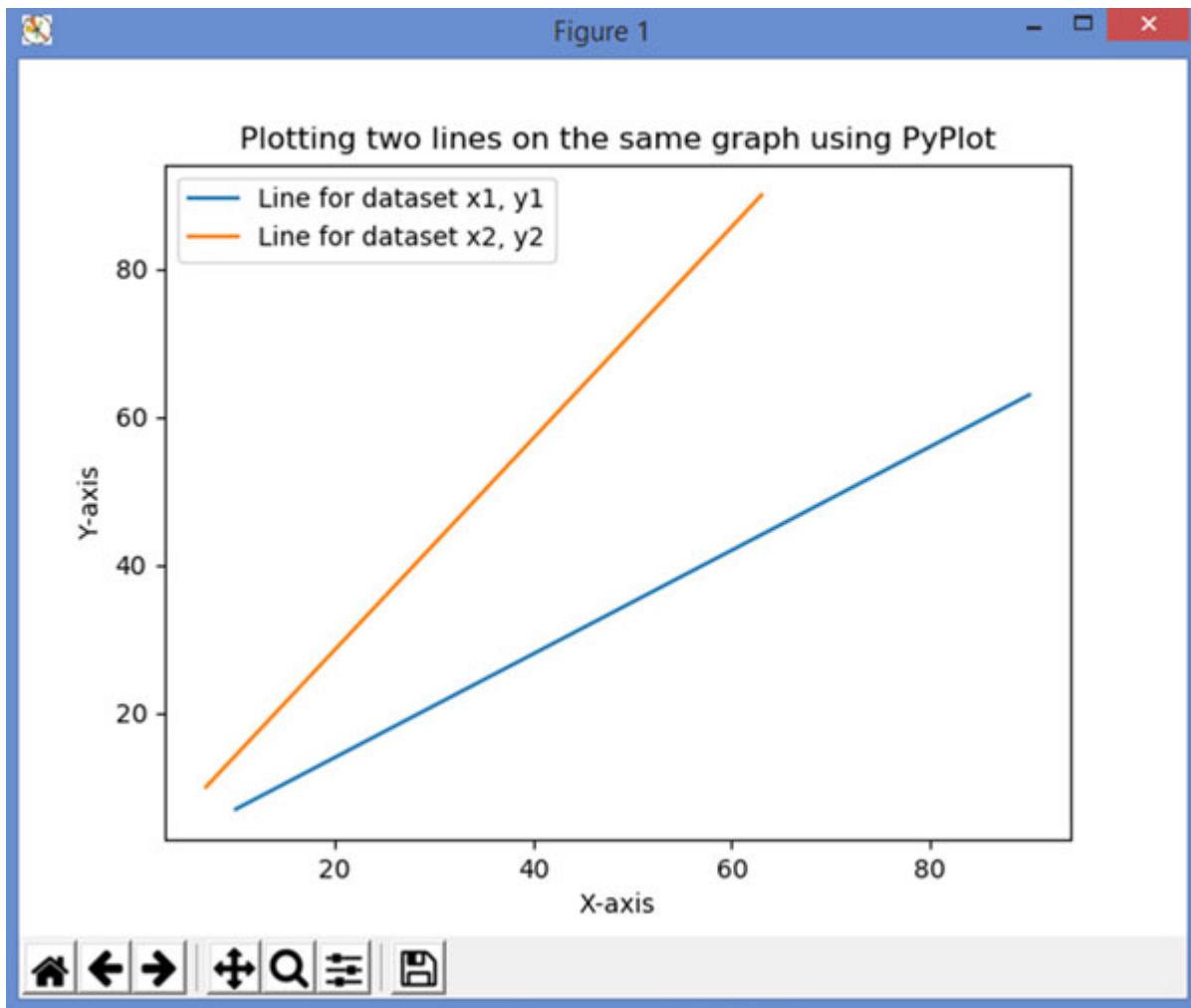
#Label the y-axis
plt.ylabel('Y-axis')

#Give a title to the graph
plt.title('Plotting two lines on the same graph using PyPlot')

#invoke the legend function
plt.legend()

#Display the plot
plt.show()
```

## Output:



*Figure 7.14*

As you can see now, the blue line displays the plot for values of  $x_1$  and  $y_1$ , and the red line displays the plot for values of  $x_2$  and  $y_2$ .

## **7.3 numpy**

In this section, you will learn to work with `numpy` arrays. Let's start with installing `numpy` module.

### **7.3.1 Installing numpy**

The following steps explain how to install Numpy.

1. Open command prompt.

2. Change the directory to the directory where Python is installed (as we did while installing matplotlib).
3. Give the command `pip install numpy`.

### 7.3.2 Shape of numpy Arrays

The ‘`numpy`’ arrays have dimensions, which can be found out with the help keyword `shape`. The shape of an array is a tuple of number of elements present in each dimension.

Look at the following lines of code:

```
>>> import numpy as np
>>> arr = np.array([[1,2,3,4],
[3,5,2,1],
[6,4,1,4]])
```

If you print the array arr, the ouput will be as follows:

```
[[1 2 3 4]
[3 5 2 1]
[6 4 1 4]]
```

You can observe that the output has 3 rows and 4 columns:

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 3 | 5 | 2 | 1 |
| 6 | 4 | 1 | 4 |

So, the shape of this array is `3 x 4`.

To print the dimension, you can give the `shape` command:

```
print(arr.shape)
```

The output is as follows:

```
(3, 4)
```

### 7.3.3 How to get value of elements from Array

To get values of elements from an array, you can use the index.

Suppose we give the following command:

```
print(arr[2][0])
```

The output is 6.

|   |   |   |   |   |
|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 |
| 0 | 1 | 2 | 3 | 4 |
| 1 | 3 | 5 | 2 | 1 |
| 2 | 6 | 4 | 1 | 4 |

The element at the index 2 of arr is [6,4,1,4].

The element at index 0 of the element at index 2 of arr is 6.

Similarly,

```
print(arr[0][0])
```

Will give an output of 1.

|   |   |   |   |   |
|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 |
| 0 | 1 | 2 | 3 | 4 |
| 1 | 3 | 5 | 2 | 1 |
| 2 | 6 | 4 | 1 | 4 |

### 7.3.4 Creating numpy arrays

Numpy arrays can be created using the following method:

```
numpy.linspace()
```

This method is used to return evenly spaced numbers over a specified interval.

**Syntax:**

```
numpy.linspace(start, stop, num)
```

Where,

**start:** is a mandatory value that tells the starting point of the sequence.

**stop**: is a mandatory value that tells the end point of the sequence.

**num**: optional value that describes the number of values to be generated between start and stop values.

Look at the following code:

```
import numpy as np
arr = np.linspace(3,5)
print(arr)
```

The output for this is as follows:

```
[3. 3.04081633 3.08163265 3.12244898 3.16326531 3.20408163
 3.24489796 3.28571429 3.32653061 3.36734694 3.40816327 3.44897959
 3.48979592 3.53061224 3.57142857 3.6122449 3.65306122 3.69387755
 3.73469388 3.7755102 3.81632653 3.85714286 3.89795918 3.93877551
 3.97959184 4.02040816 4.06122449 4.10204082 4.14285714 4.18367347
 4.2244898 4.26530612 4.30612245 4.34693878 4.3877551 4.42857143
 4.46938776 4.51020408 4.55102041 4.59183673 4.63265306 4.67346939
 4.71428571 4.75510204 4.79591837 4.83673469 4.87755102 4.91836735
 4.95918367 5.]
```

This is because we have not specified the number of values that we want to generate between the **start** and the **stop** values.

Now, let's suppose that we want to generate 10 values between the number 2 and 3. The code will be as follows:

```
import numpy as np
arr = np.linspace(2,3,10)
print(arr)
```

## Output:

```
[2. 2.11111111 2.22222222 2.33333333 2.44444444 2.55555556
 2.66666667 2.77777778 2.88888889 3.]
```

## **numpy.arange() function**

The syntax for **numpy.arange()** function is as follows:

```
numpy.arange(start, stop, step)
```

**start**: starting point of the sequence.

**stop**: ending point of the sequence.

**Step** : interval between two consecutive values, default is set to 1.

### Example 7.1

What will be the output of the following code:

```
>>>arr = np.arange(2,3,10)
>>> print(arr)
```

**Answer:**

[2]

### Example 7.2

What will be the output of the following code?

```
arr = np.arange(1,9,2)
print(arr)
```

**Answer:**

[1 3 5 7]

### Example 7.3

What will be the output of the following code?

```
arr = np.arange(2,5)
print(arr)
```

**Answer:**

[2 3 4]

Till now you have seen examples where you have the values of **x** and **y** available. How would you plot a graph if the value of one variable is dependent on the other? Suppose we have line defined in the form of  $y = mx + c$ . So, while you have the values for **x**, the values of **y** needs to be computed. This can be achieved with the help of numpy.

Plot a graph for following values of **x**:

*Table 7.7*

|   |   |   |   |   |    |    |
|---|---|---|---|---|----|----|
| X | 1 | 4 | 7 | 9 | 10 | 15 |
|---|---|---|---|---|----|----|

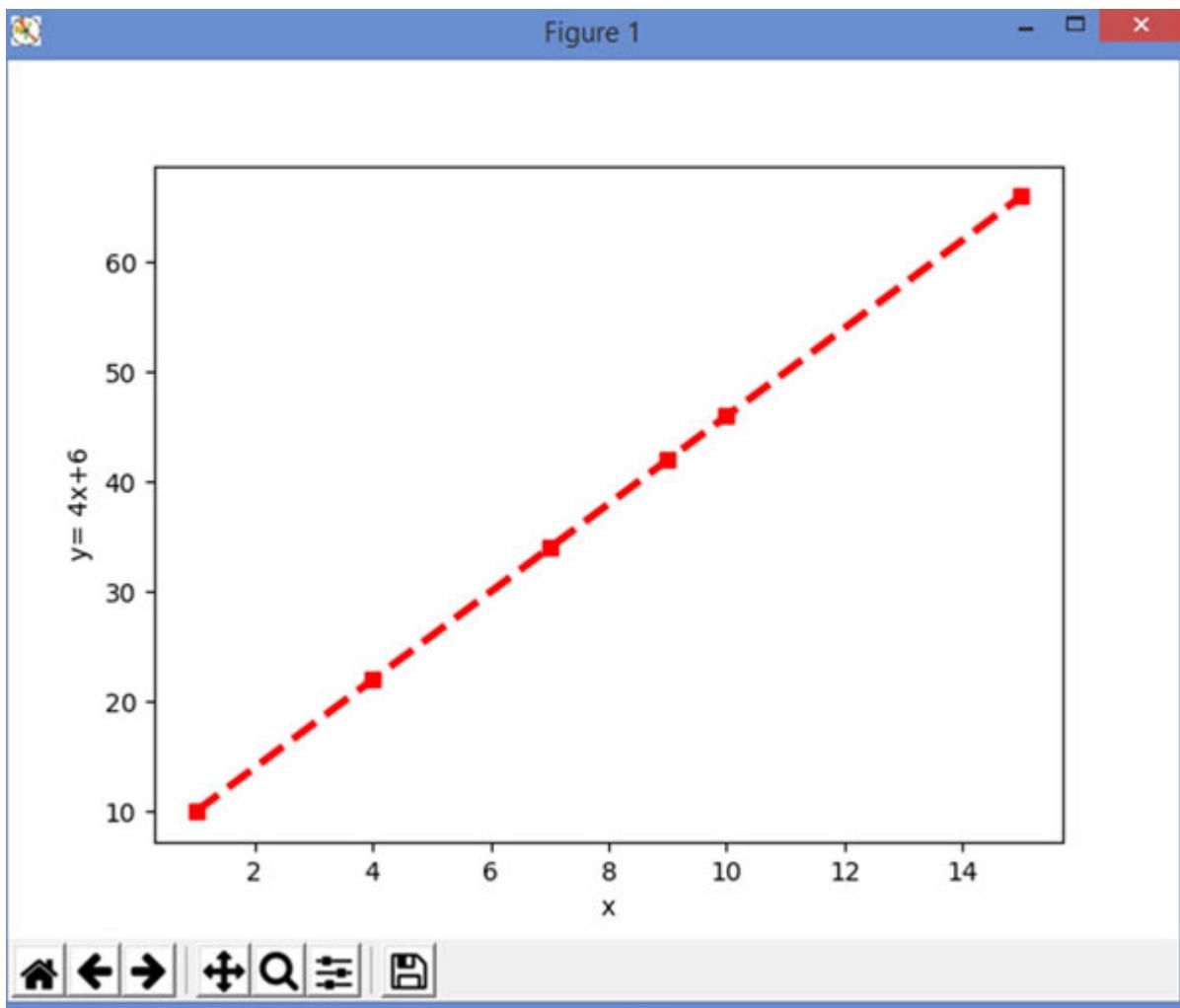
$$y = 4x + 6$$

**Code:**

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1,4,7,9,10,15])
y = 4*x+6
plt.plot(x,y,"rs--", markersize = 6, linewidth = 3)
plt.xlabel("x")
plt.ylabel("y= 4x+6")
plt.show()
```

**Output:**



*Figure 7.15*

## Example 7.4

Plot a graph for  $f(x)$  vs.  $x$ , where  $x$  is in the range of 20 to 40, with 5 values apart and

$$f(x) = \log(x)$$

**Answer:**

**Step 1:** Import numpy.

```
import numpy as np
```

**Step 2:** Import PyPlot.

```
import matplotlib.pyplot as plt
```

**Step 3:** Define array have values from 20 to 40, 5 values apart.

```
x = np.arange(20, 40, 5)
```

**Step 4:** Define  $f(x) = \log(x)$ .

```
y = np.log(x)
```

**Step 5:** Call the `plot()` function.

```
plt.plot(x, y)
```

**Step 6:** Label the x-axis.

```
plt.xlabel("x values from 20-40, 5 steps apart")
```

**Step 7:** Label the y – axis.

```
plt.ylabel("f(x) = log(x)")
```

**Step 8:** Give a title to the plot.

```
plt.title("f(x) vs. x")
```

**Step 9:** Display the plot.

```
plt.show()
```

## Code:

```
#import numpy
import numpy as np

#import PyPlot
import matplotlib.pyplot as plt

#define array have values from 20 to 40, 5 values apart
x = np.arange(20,40,5)

define f(x) = log(x)
y = np.log(x)

#Call the plot() function
plt.plot(x,y)

#Label the x-axis
plt.xlabel("x values from 20-40, 5 steps apart")

#label the y - axis
plt.ylabel("f(x) = log(x)")

#Give a title to the plot
plt.title("f(x) vs. x")

#Display the plot
plt.show()
```

## Output:

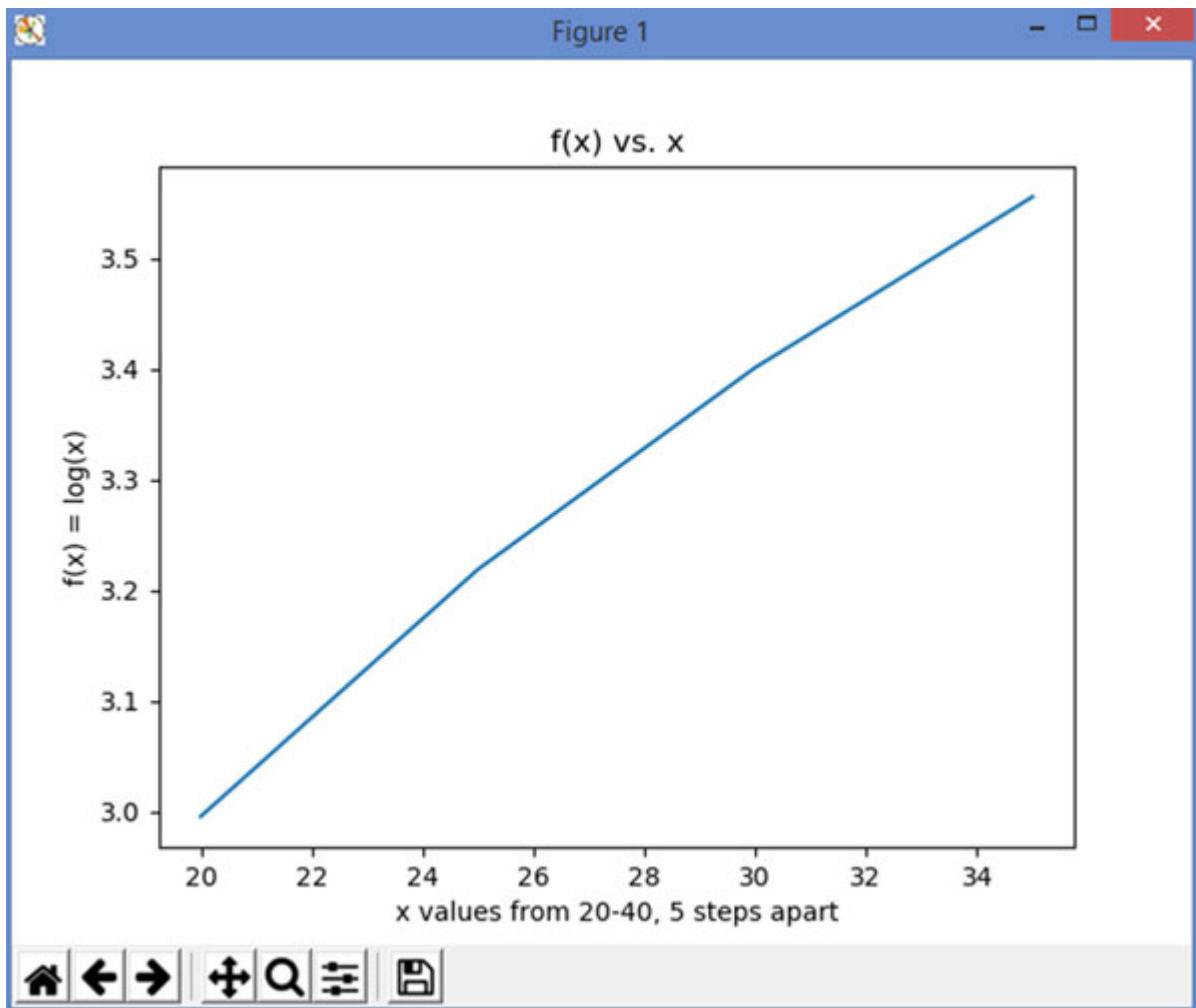


Figure 7.16

The main purpose of numpy is to use multi-dimensional arrays.

**Example 7.5:** Take a suitable list as input from user and plot a graph for  $y$ .

**Answer:**

**Code:**

```
#import PyPlot
import matplotlib.pyplot as plt

#import NumPy
import numpy as np

#Define start, end and step values of the range
start = int(input("Enter the starting point of the range : "))
end = int(input("Enter the end point of the range : "))
```

```
step = int(input("Enter the interval between two consecutive values : "))

#define array for x-axis
x = np.arange(start,end,step)

define array for y-axis
y = (x**2)+(3*x)

#Call the plot() function
plt.plot(x,y)

#Label the x-axis
plt.xlabel("X-axis")

#label the y - axis
plt.ylabel("Y-axis")

#Give a title to the plot
plt.title("plotting of function")

Display the plot
plt.show()
```

## Output:

```
Enter the starting point of the range : 100
Enter the end point of the range : 600
Enter the interval between two consecutive values : 30
```

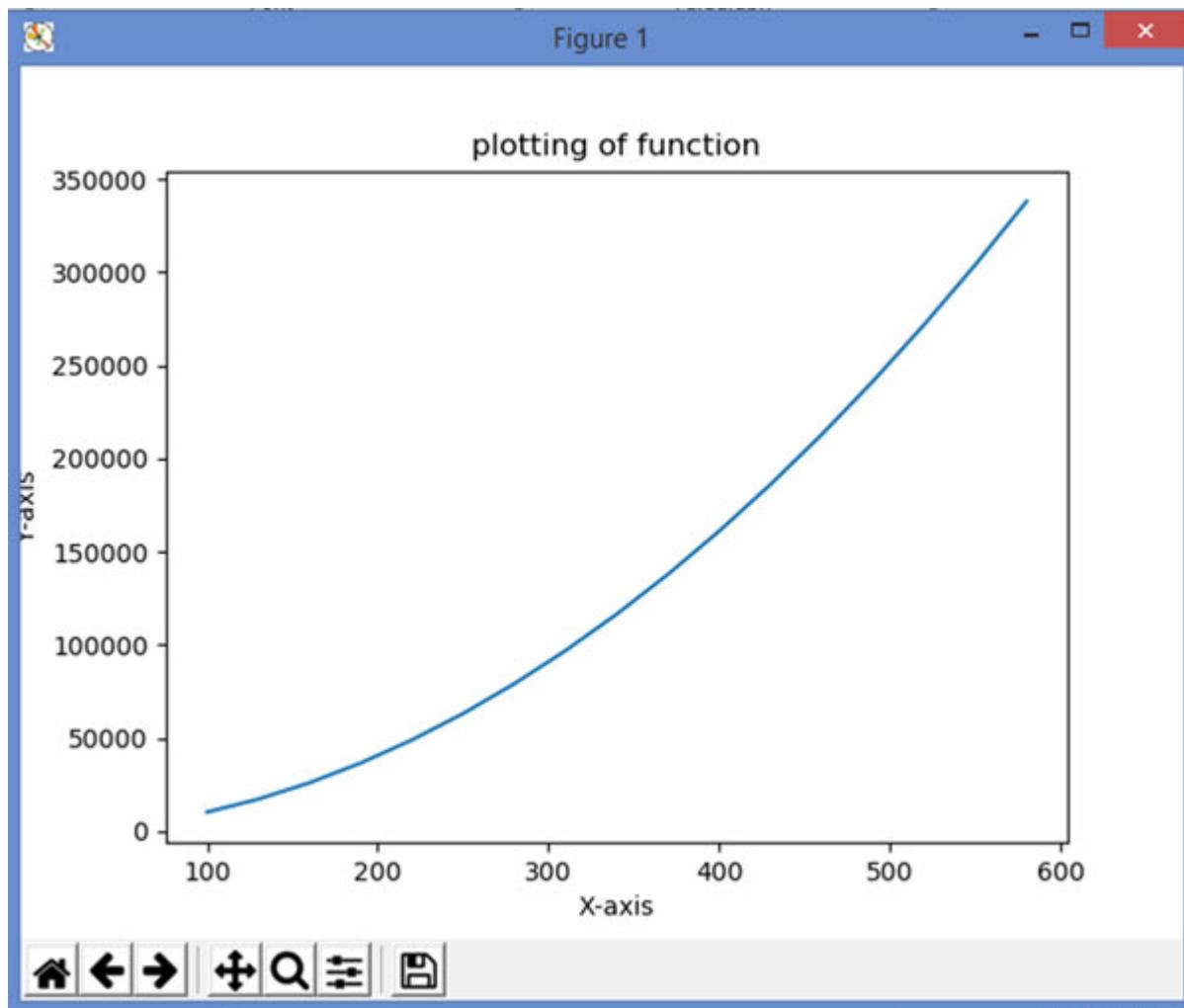


Figure 7.17

## Bar graphs

### Example 7.6:

The following table provides the value of countries and their population:  
Depict in a bar graph.

| Country       | Population |
|---------------|------------|
| China         | 1384688986 |
| India         | 1296834042 |
| United States | 329256465  |
| Indonesia     | 262787403  |

|            |           |
|------------|-----------|
| Brazil     | 208846892 |
| Nigeria    | 195300343 |
| Bangladesh | 159453001 |

**Table 7.8**

### Step 1. importPyPlot

```
import matplotlib.pyplot as plt
```

### Step 2. Define array of country names

```
x = ['China','India','USA','Indonesia','Brazil','Nigeria','Bangladesh']
```

### Step 3. Define array of population

```
y = [1384688986,1296834042,329256465,262787403,208846892,195300343,159453001]
```

### Step 4. Call the `bar()` function

```
plt.bar(x,y)
```

### Step 5. Label the x-axis

```
plt.xlabel("Country")
```

### Step 6. Label the y – axis

```
plt.ylabel("Population")
```

### Step 7. Give a title to the plot

```
plt.title("Country vs. Population")
```

### Step 8. Display the plot

```
plt.show()
```

### Code:

```
#import PyPlot
import matplotlib.pyplot as plt

#define array of country names
x = ['China', 'India', 'USA', 'Indonesia', 'Brazil', 'Nigeria', 'Bangladesh']

define array of population
y = [1384688986, 1296834042, 329256465, 262787403, 208846892, 195300343, 159453001]

#Call the bar() function
plt.bar(x,y)

#Label the x-axis
plt.xlabel("Country")

#label the y - axis
plt.ylabel("Population")

#Give a title to the plot
plt.title("Country vs. Population")

Display the plot
plt.show()
```

## Output:

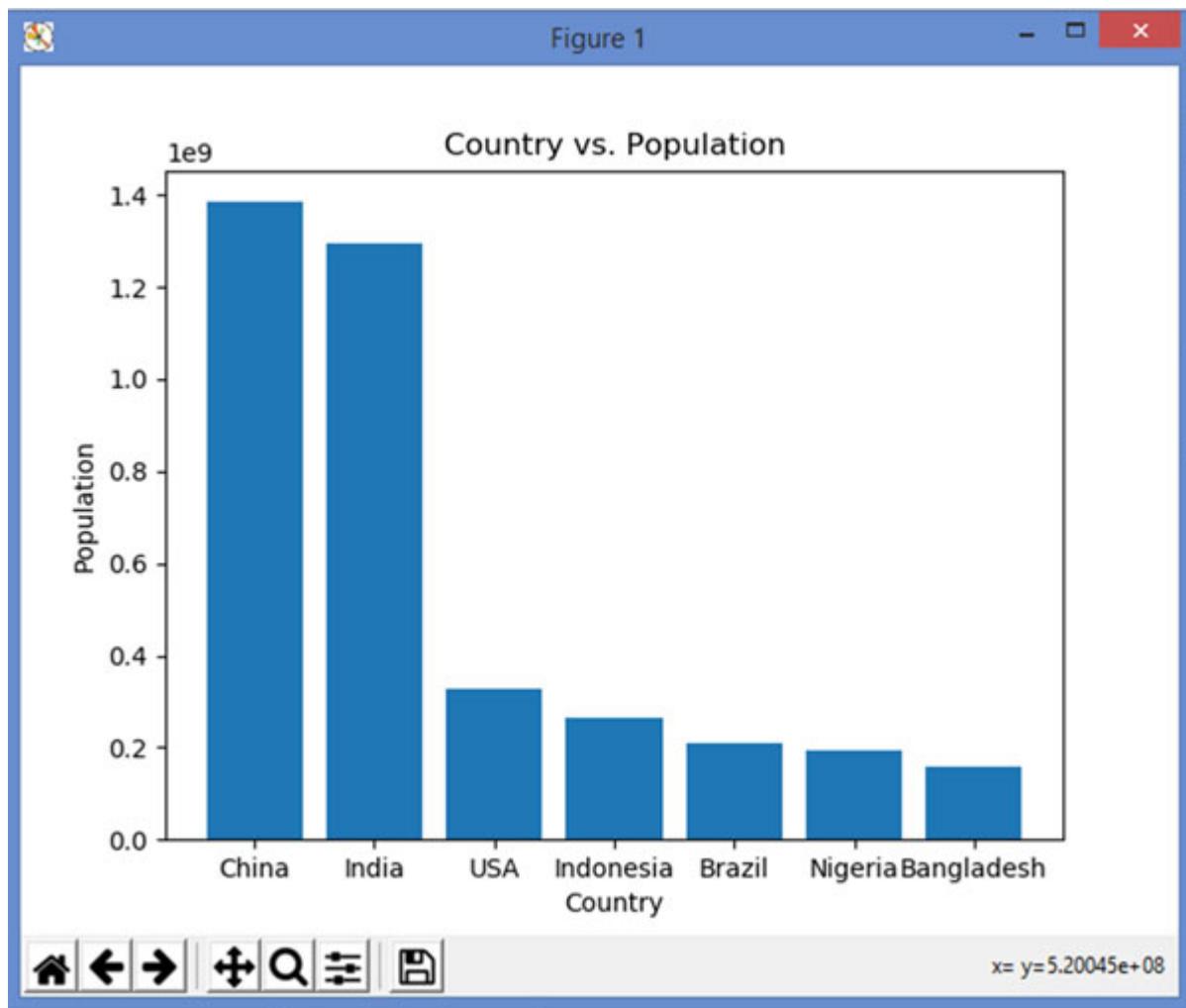


Figure 7.18

### Example 7.7:

Write a program in Python to display student name on x-axis, and percentage score on y-axis in the form of a bar graph for two streams (science and commerce).

| Science    |          | Commerce   |           |
|------------|----------|------------|-----------|
| Student id | Score(%) | Student id | Score (%) |
| 101        | 78       | 102        | 89        |
| 104        | 57       | 103        | 100       |
| 108        | 89       | 105        | 98        |
| 109        | 99       | 106        | 67        |

|     |    |     |     |
|-----|----|-----|-----|
| 110 | 86 | 107 | 100 |
| 112 | 72 | 111 | 20  |

**Table 7.9**

**Answer:**

**Code:**

```
#import PyPlot
import matplotlib.pyplot as plt

#define arrays of Science Student id and Score
x1 = [101,104,108,109,110,112]
y1 = [78,57,89,99,86,72]

define arrays of commerce Students id and Score
x2 = [102,103,105,106,107,111]
y2 = [89,100,98,67,100,20]

#Call the plot() function
plt.bar(x1,y1,label='Science department')
plt.bar(x2,y2,label='Commerce department')

#Label the x-axis
plt.xlabel("Student id")

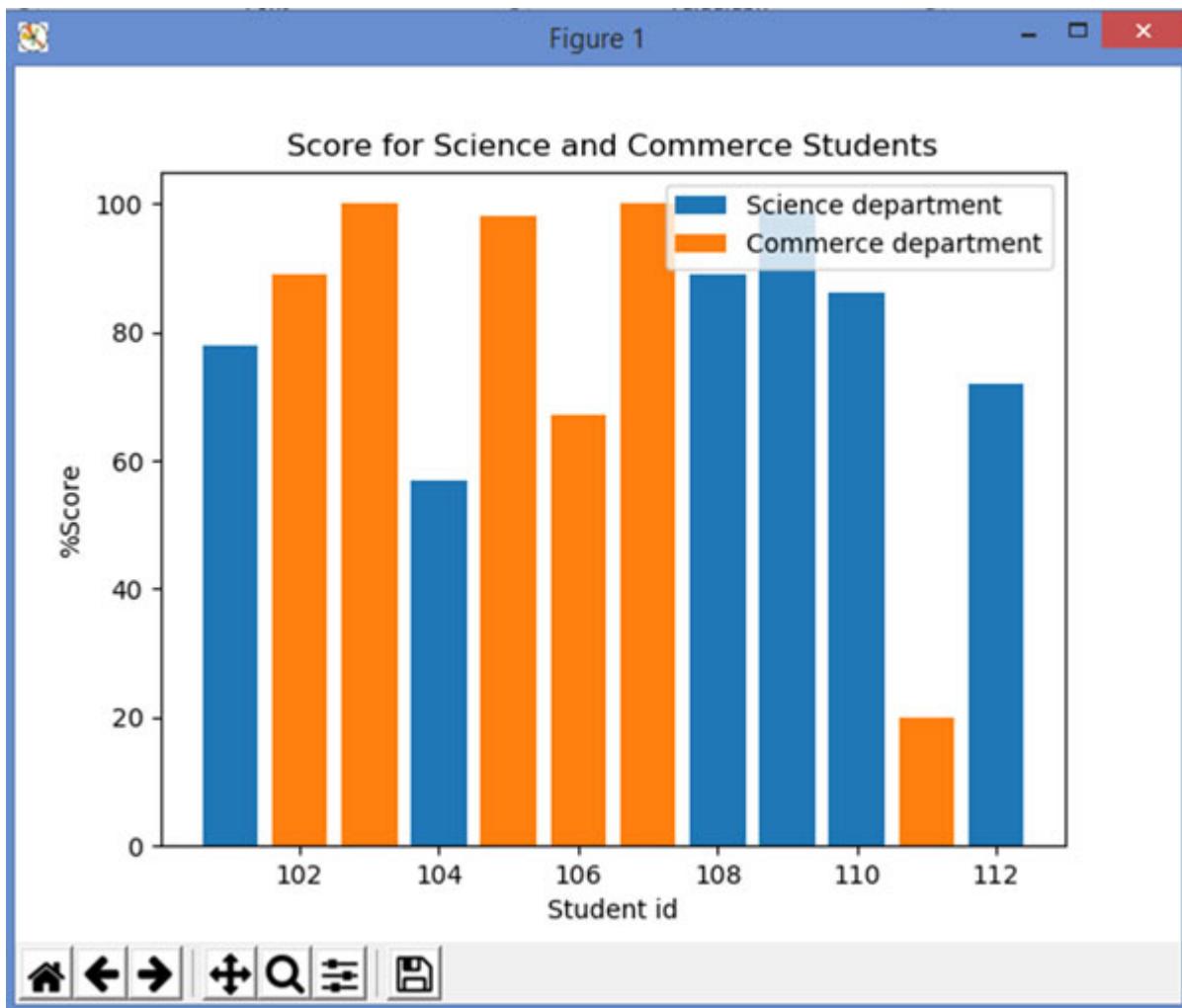
#label the y - axis
plt.ylabel("%Score")

#Give a title to the plot
plt.title("Score for Science and Commerce Students")

plt.legend()

Display the plot
plt.show()
```

**Output:**



*Figure 7.19*

### Example 7.8:

The following table displays the overall score of students out of 500. Convert the score into percentage, and display as bar graph.

| Student | Score(out of 500) |
|---------|-------------------|
| John    | 376               |
| Ted     | 455               |
| Lizy    | 489               |
| Elena   | 300               |
| Alex    | 250               |
| Albert  | 340               |
| Meg     | 400               |

**Table 7.10**

**Answer:**

**Code:**

```
#import PyPlot
import matplotlib.pyplot as plt

#import NumPy
import numpy as np

#define array of Student Name
x = ['John','Ted','Lizy','Elena','Alex','Albert','Meg']

define array of Student Score
arr = np.array([376,455,489,300,250,340,400])
y = arr*100/500

#Call the plot() function
plt.bar(x,y)

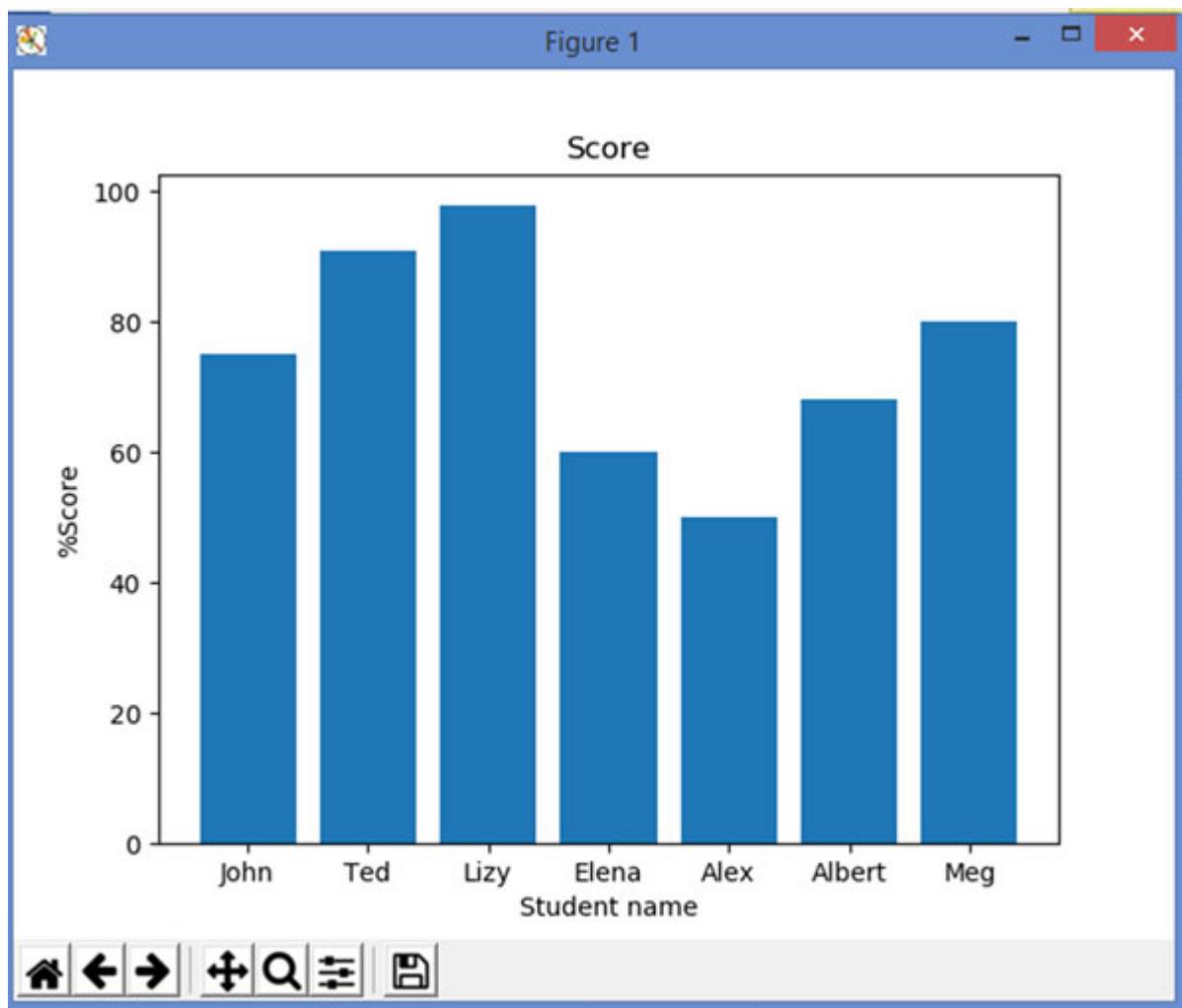
#Label the x-axis
plt.xlabel("Student name")

#label the y - axis
plt.ylabel("%Score")

#Give a title to the plot
plt.title("Score")

Display the plot
plt.show()
```

**Output:**



*Figure 7.20*

## Pie charts

### Example 7.9:

The following table expresses the components of air. Convert this data into pie chart.

**Answer:**

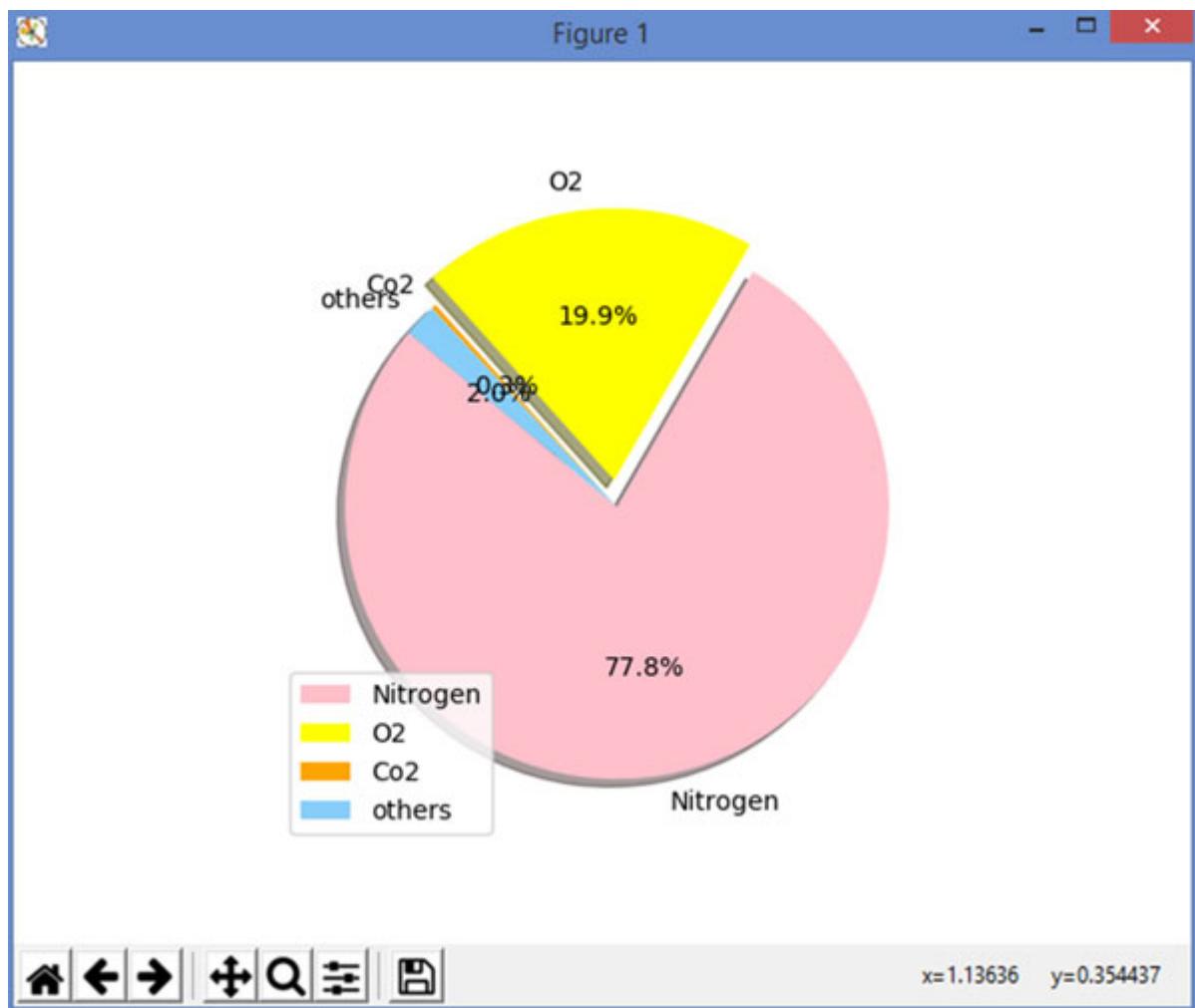
| Gas             | %    |
|-----------------|------|
| Nitrogen        | 78   |
| Oxygen          | 20   |
| Carbon di oxide | 0.3  |
| Others          | 1.97 |

**Table 7.11**

**Code:**

```
import matplotlib.pyplot as plt
Data to plot
gases = ['Nitrogen', 'O2', 'Co2', 'others']
sizes = [78,20,0.3,1.97]
colors = ['pink', 'yellow','orange', 'lightskyblue']

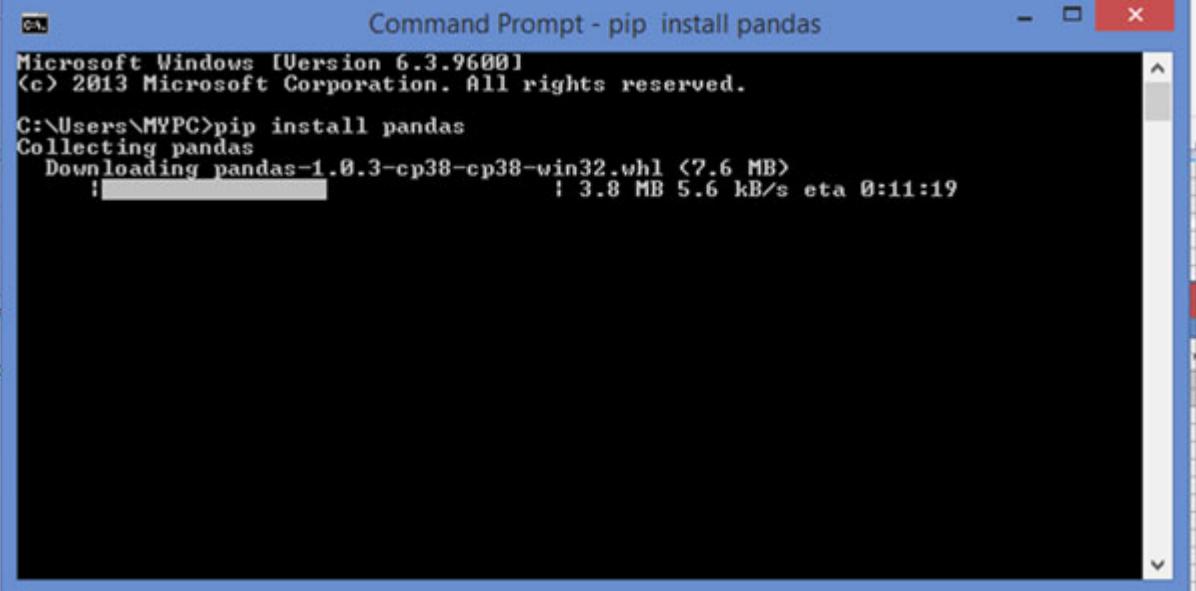
Plot
plt.pie(sizes,explode=(0, 0.1, 0, 0),labels =gases,colors=colors,
autopct='%.1f%%', shadow=True, startangle=140)
plt.legend()
plt.show()
```



**Figure 7.21**

## 7.4 pandas

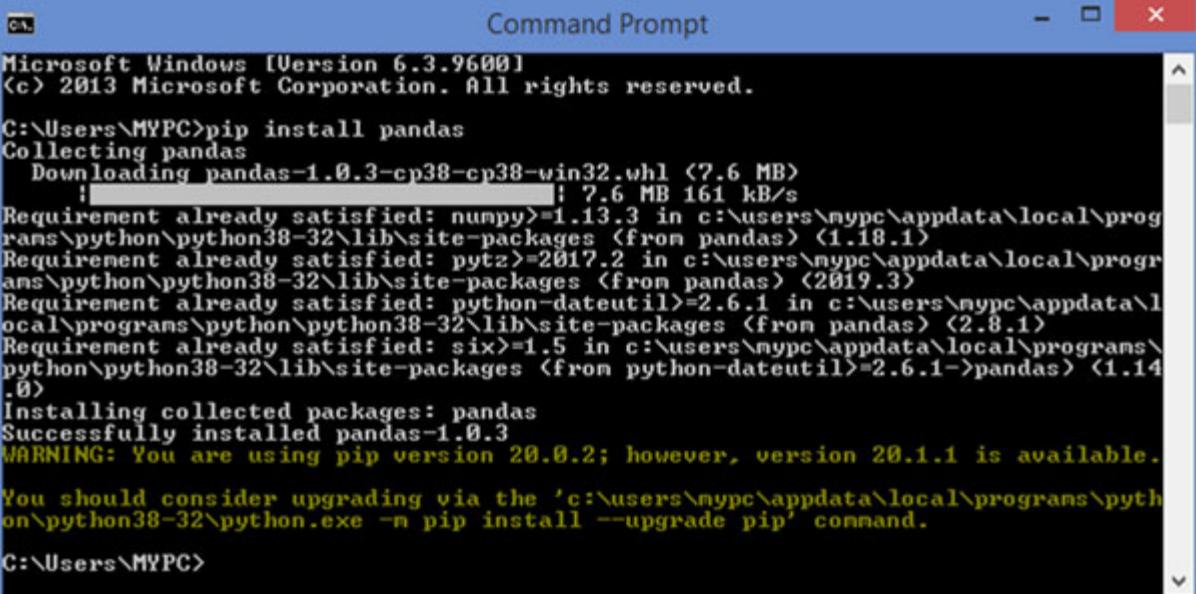
In this section, we will learn about **pandas**. ‘**pandas**’ is a fast, easy-to-use, and powerful data analysis/manipulation tool that is known for its flexibility. It is a software library that the Python programmers can install and use in order to work with data.



The screenshot shows a Microsoft Windows Command Prompt window titled "Command Prompt - pip install pandas". The window displays the command "C:\Users\MYPC>pip install pandas" and the output of the pip install process. The output shows "Collecting pandas", "Downloading pandas-1.0.3-cp38-cp38-win32.whl <7.6 MB>", and a progress bar indicating the download is at 3.8 MB (5.6 kB/s) with an estimated time of 0:11:19 remaining.

Figure 7.22

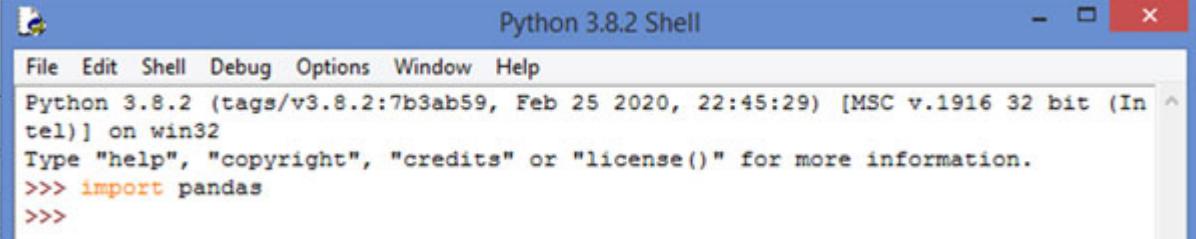
If it is successfully installed, it will be displayed on the command prompt.



The screenshot shows a Microsoft Windows Command Prompt window titled "Command Prompt". The window displays the command "C:\Users\MYPC>pip install pandas" and the output of the pip install process. The output shows "Collecting pandas", "Downloading pandas-1.0.3-cp38-cp38-win32.whl <7.6 MB>", and a progress bar indicating the download is at 7.6 MB (161 kB/s). The output also lists requirements already satisfied for numpy, pytz, python-dateutil, and six. It then shows "Installing collected packages: pandas", "Successfully installed pandas-1.0.3", and a warning message about pip version 20.0.2 being used instead of 20.1.1. It also suggests upgrading pip via the command "c:\users\mypc\appdata\local\programs\python\python38-32\python.exe -m pip install --upgrade pip".

Figure 7.23

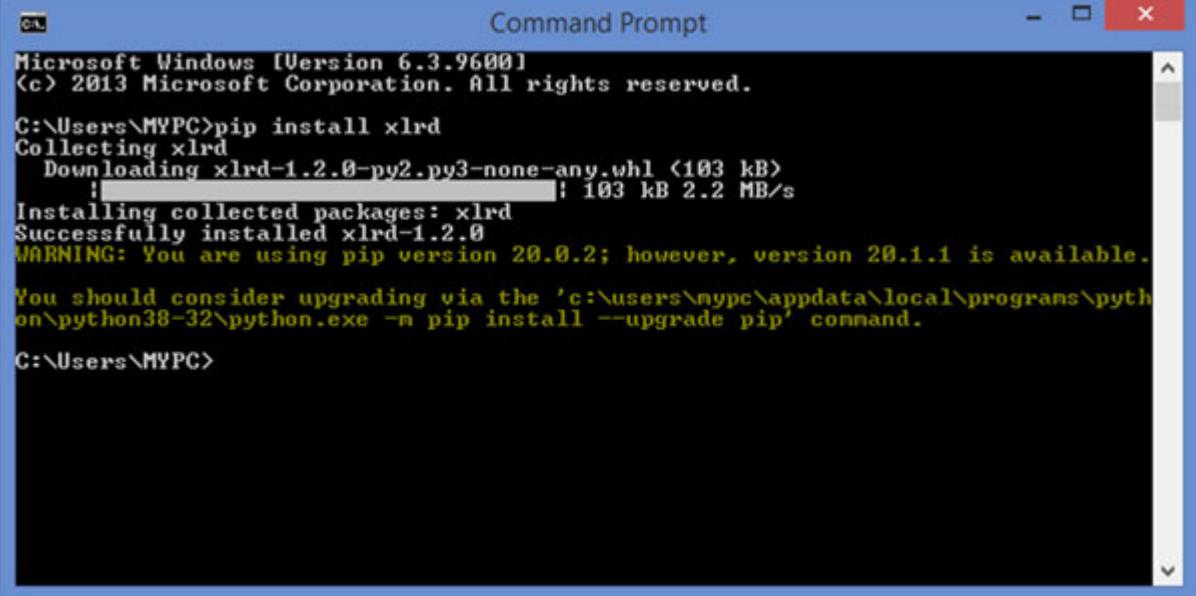
You can check whether **pandas** has been installed by trying to import it in python shell. If it is properly installed, no error will be generated.



```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import pandas
>>>
```

Figure 7.24

To work with **pandas**, we also need to install **xlrd** package using pip.



```
Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>pip install xlrd
Collecting xlrd
 Downloading xlrd-1.2.0-py2.py3-none-any.whl (103 kB)
 :███ 103 kB 2.2 MB/s
Installing collected packages: xlrd
Successfully installed xlrd-1.2.0
WARNING: You are using pip version 20.0.2; however, version 20.1.1 is available.
You should consider upgrading via the 'c:\users\mypc\appdata\local\programs\python\python38-32\python.exe -m pip install --upgrade pip' command.

C:\Users\MYPC>
```

Figure 7.25

We will now work with an excel sheet. For this example we have an excel file by the name of **shares.xlsx** in **F:\data** directory.



```
>>> import xlrd
>>> |
```

Figure 7.26

## 7.4.1 Creating dataframe from an Excel sheet

In the first step, we will learn how to access the excel file using pandas. For this example we have an excel file by the name **shares.xlsx** in the **F:\data**

directory. The data is in the sheet 1 of the Excel sheet.

|    | A                   | B        | C       | D                                              | E               | F       | G |
|----|---------------------|----------|---------|------------------------------------------------|-----------------|---------|---|
| 1  | share               | quantity | price   | description                                    | amount invested | profit% |   |
| 2  | Coal india          | 12       | 311.7   | POWER                                          | 3740.4          | 15%     |   |
| 3  | Powergrid           | 16       | 97.88   |                                                | 1566.08         | 7%      |   |
| 4  | Polyplex Corporatic | 10       | 484     | 4th largest manufacturer of thin polyester Fil | 4840            |         |   |
| 5  | noida toll          | 40       | 33.5    |                                                | 1340            |         |   |
| 6  | Kotak bank          | 5        | 479.85  |                                                | 2399.25         | 2.00%   |   |
| 7  | Sail                | 10       | 187.75  |                                                | 1877.5          | 4%      |   |
| 8  | SBI                 | 3        | 2756.6  |                                                | 8269.8          |         |   |
| 9  | SBI                 | 2        | 2191.5  |                                                | 4383            |         |   |
| 10 | SBI                 | 2        | 1911.5  |                                                | 3,823           |         |   |
| 11 | Reliance            | 5        | 773.5   |                                                | 3867.5          | 11%     |   |
| 12 | JP Associate        | 13       | 59.5    |                                                | 773.5           | 19%     |   |
| 13 | suzlon              | 13       | 38.78   |                                                | 504.14          |         |   |
| 14 | tata steel          | 10       | 415.87  |                                                | 4158.7          | 14%     |   |
| 15 | SAIL                | 10       | 109.7   |                                                | 1097            |         |   |
| 16 | ICICI               | 3        | 857.3   |                                                | 2571.9          | 8%      |   |
| 17 | IDFC                | 10       | 110.45  |                                                | 1104.5          | 12%     |   |
| 18 |                     |          |         |                                                |                 | 15%     |   |
| 19 | SAIL                | 9        | 104.1   |                                                | 936.9           |         |   |
| 20 | IDFC                | 9        | 108     |                                                | 972             | 15%     |   |
| 21 | SUZLON              | 13       | 37.2    |                                                | 483.6           |         |   |
| 22 | tatamotors          | 5        | 154.85  |                                                | 774.25          | 21%     |   |
| 23 | ICICI               | 3        | 780.7   |                                                | 2342.1          | 18%     |   |
| 24 | icici               | 6        | 786.25  |                                                |                 |         |   |
| 25 | sbi                 | 2        | 1727.55 |                                                |                 |         |   |
| 26 | icici               | 1        | 722.1   |                                                |                 |         |   |
| 27 | sbi                 | 1        | 1636.35 |                                                |                 |         |   |
| 28 | icici               | 1        | 713.4   |                                                |                 |         |   |
| 29 | Total               |          |         |                                                | 51825.12        |         |   |
| 30 |                     |          |         |                                                |                 |         |   |

Figure 7.27

**Step 1:** Import pandas.

```
>>>import pandas as pd
```

**Step 2:** Open the excel file and read the contents.

```
>>> df = pd.read_excel("F:\\data\\shares.xlsx",'Sheet1')
>>> df
```

Here, `df` is a **dataframe** object. A dataframe is a two-dimensional data structure. It is size mutable, and data in a table is stored in a tabular format that has rows and columns.

So, this is how your code looks like:

```
>>> import pandas as pd
>>> df = pd.read_excel("F:\\data\\shares.xlsx",'Sheet1')
>>> df
```

## Output:

```
>>> df = pd.read_excel("F:\\data\\shares.xlsx",'Sheet1')
>>> df
 share quantity ... amount invested profit%
0 Coal india 12.0 ...
1 Powergrid 16.0 ...
2 Polyplex Corporation 10.0 ...
3 noida toll 40.0 ...
4 Kotak bank 5.0 ...
5 Sail 10.0 ...
6 SBI 3.0 ...
7 SBI 2.0 ...
8 SBI 2.0 ...
9 Reliance 5.0 ...
10 JP Associate 13.0 ...
11 suzlon 13.0 ...
12 tata steel 10.0 ...
13 SAIL 10.0 ...
14 ICICI 3.0 ...
15 IDFC 10.0 ...
16 NaN NaN ...
17 SAIL 9.0 ...
18 IDFC 9.0 ...
19 SUZLON 13.0 ...
20 tatamotors 5.0 ...
21 ICICI 3.0 ...
22 icici 6.0 ...
23 sbi 2.0 ...
24 icici 1.0 ...
25 sbi 1.0 ...
26 icici 1.0 ...
27 Total NaN ...
[28 rows x 6 columns]
```

Figure 7.28

Notice that a data frame object displays an additional column on the left-hand side.

### 7.4.2 Creating dataframe from .csv file

For this example we have the same file stored in the `.csv` format. Steps involved in creating a dataframe from a `.csv` file are as follows:

**Step 1:** Import pandas.

```
import pandas as pd
```

**Step 2:** Read the csv file and store data in a data frame object.

```
>>> df = pd.read_csv("F:\\\\data\\\\shares.csv")
>>> df
```

## Output:

|    | share                | quantity | ... | amount invested | profit% |
|----|----------------------|----------|-----|-----------------|---------|
| 0  | Coal india           | 12.0     | ... | 3740.4          | 15%     |
| 1  | Powergrid            | 16.0     | ... | 1566.08         | 7%      |
| 2  | Polyplex Corporation | 10.0     | ... | 4840            | NaN     |
| 3  | noida toll           | 40.0     | ... | 1340            | NaN     |
| 4  | Kotak bank           | 5.0      | ... | 2399.25         | 2.00%   |
| 5  | Sail                 | 10.0     | ... | 1877.5          | 4%      |
| 6  | SBI                  | 3.0      | ... | 8269.8          | NaN     |
| 7  | SBI                  | 2.0      | ... | 4383            | NaN     |
| 8  | SBI                  | 2.0      | ... | 3,823           | NaN     |
| 9  | Reliance             | 5.0      | ... | 3867.5          | 11%     |
| 10 | JP Associate         | 13.0     | ... | 773.5           | 19%     |
| 11 | suzlon               | 13.0     | ... | 504.14          | NaN     |
| 12 | tata steel           | 10.0     | ... | 4158.7          | 14%     |
| 13 | SAIL                 | 10.0     | ... | 1097            | NaN     |
| 14 | ICICI                | 3.0      | ... | 2571.9          | 8%      |
| 15 | IDFC                 | 10.0     | ... | 1104.5          | 12%     |
| 16 | NaN                  | NaN      | ... | NaN             | 15%     |
| 17 | SAIL                 | 9.0      | ... | 936.9           | NaN     |
| 18 | IDFC                 | 9.0      | ... | 972             | 15%     |
| 19 | SUZLON               | 13.0     | ... | 483.6           | NaN     |
| 20 | tatamotors           | 5.0      | ... | 774.25          | 21%     |
| 21 | ICICI                | 3.0      | ... | 2342.1          | 18%     |
| 22 | icici                | 6.0      | ... | NaN             | NaN     |
| 23 | sbi                  | 2.0      | ... | NaN             | NaN     |
| 24 | icici                | 1.0      | ... | NaN             | NaN     |
| 25 | sbi                  | 1.0      | ... | NaN             | NaN     |
| 26 | icici                | 1.0      | ... | NaN             | NaN     |
| 27 | Total                | NaN      | ... | 51825.12        | NaN     |

Figure 7.29

### 7.4.3 Creating Dataframe from Dictionary

You are familiar with dictionary and would know that the data in dictionary is stored in a key-value pair.

**Step 1:** Import Pandas.

```
import pandas as pd
```

## Step 2: Define the dictionary object.

```
shares = {"share": ["Coal india", "Powergrid", "Polyplex Corporation", "noida toll", "kotakbank", "Sail"], "quantity": [12.0, 16.0, 10.0, 40.0, 5.0, 10.0], "amount invested": [3740.4, 1566.08, 4840.0, 1340.0, 2399.25, 1877.5], "profit": [15, 7, 0, 0, 2, 4]}
```

## Step 3: Create data frame object

```
df = pd.DataFrame(shares)
```

### Code:

```
>>> import pandas as pd
>>> shares = {"share": ["Coal india", "Powergrid", "Polyplex Corporation", "noida toll", "kotakbank", "Sail"], "quantity": [12.0, 16.0, 10.0, 40.0, 5.0, 10.0], "amount invested": [3740.4, 1566.08, 4840.0, 1340.0, 2399.25, 1877.5], "profit": [15, 7, 0, 0, 2, 4]}
>>> df = pd.DataFrame(shares)
>>> df
```

### Output:

```
 share quantity amount invested profit
0 Coal india 12.0 3740.40 15
1 Powergrid 16.0 1566.08 7
2 Polyplex Corporation 10.0 4840.00 0
3 noida toll 40.0 1340.00 0
4 kotak bank 5.0 2399.25 2
5 Sail 10.0 1877.50 4
>>>
```

Figure 7.30

## 7.4.4 Creating Data Frame from list of tuples

In the last section, you created a DataFrame from a dictionary. In this section, you will learn how to create a DataFrame from a list of Tuples.

### Step 1: Import pandas.

```
import pandas as pd
```

### Step 2: Define list of tuples.

```
Shares = [("Coal india",12.0,3740.4,15), ("Powergrid",16.0, 1566.08,7), ("Polyplex Corporation",10.0,4840.0,0), ("noida toll",40.0, 1340.0,0), ("kotak bank",5.0,2399.25,2), ("Sail",10.0,1877.5,4)]
```

### Step 3: Create a DataFrame

Pass on the list of tuples, and define the column names.

```
df = pd.DataFrame(shares,columns =["share","quantity","amountinvested", "profit"])
```

#### Code:

```
import pandas as pd
shares = [("Coal india",12.0,3740.4,15), ("Powergrid",16.0, 1566.08,7), ("Polyplex Corporation",10.0,4840.0,0), ("noida toll",40.0, 1340.0,0), ("kotak bank",5.0,2399.25,2), ("Sail",10.0,1877.5,4)]
df = pd.DataFrame(shares,columns =["share","quantity","amountinvested", "profit"])
print(df)
```

#### Output:

```
 share quantity amount invested profit
0 Coal india 12.0 3740.40 15
1 Powergrid 16.0 1566.08 7
2 Polyplex Corporation 10.0 4840.00 0
3 noida toll 40.0 1340.00 0
4 kotak bank 5.0 2399.25 2
5 Sail 10.0 1877.50 4
>>>
```

Figure 7.31

## 7.5 Operations on DataFrame

In this section, you will learn how to find out the number of rows and columns in a DataFrame.

Let's go back to creating Data Frame with Excel sheet. This is how your code looks like.

```
>>> import pandas as pd
>>> df = pd.read_excel("F:\\data\\shares.xlsx","Sheet1")
```

To know how many rows and columns are present in the DataFrame object, we use shape attribute as shown here:

```
>>>df.shape
(28, 6)
```

For clarity, we can write:

```
>>>row,col = df.shape
>>>print("The dataframe has {} rows.".format(row))
The dataframe has 28 rows.
>>>print("The dataframe has {} columns.".format(col))
The dataframe has 6 columns.
```

### 7.5.1 Retrieving rows and columns

You can view the first five rows of the dataframe using the `head()` function.

```
>>> import pandas as pd
>>> df = pd.read_excel("F:\\data\\shares.xlsx","Sheet1")
>>> df.head()
```

#### Output:

```
 share quantity ... amount invested profit%
0 Coal india 12.0 ... 3740.40 0.15
1 Powergrid 16.0 ... 1566.08 0.07
2 Polyplex Corporation 10.0 ... 4840.00 NaN
3 noida toll 40.0 ... 1340.00 NaN
4 Kotak bank 5.0 ... 2399.25 0.02
```

*Figure 7.32*

Similarly, to display last 5 rows, you can use the `tail()` function.

```
>>>df.tail()
```

#### Output:

```
>>> df.tail()
 share quantity price description amount invested profit%
23 sbi 2.0 1727.55 NaN NaN NaN
24 icici 1.0 722.10 NaN NaN NaN
25 sbi 1.0 1636.35 NaN NaN NaN
26 icici 1.0 713.40 NaN NaN NaN
27 Total NaN NaN NaN 51825.12 NaN
```

*Figure 7.33*

If you want to retrieve less than first five rows or columns, then you can mention the number in the **head()** or **tail()** function.

```
>>> df.head(3)
 share quantity ... amount invested profit%
0 Coal india 12.0 ... 3740.40 0.15
1 Powergrid 16.0 ... 1566.08 0.07
2 Polyplex Corporation 10.0 ... 4840.00 NaN

[3 rows x 6 columns]
>>> df.tail(1)
 share quantity price description amount invested profit%
27 Total NaN NaN NaN 51825.12 NaN
>>>
```

*Figure 7.34*

A range of rows can be retrieved using **slicing** just the way you use it for lists.

```
>>> df[4:16]
 share quantity price description amount invested profit%
4 Kotak bank 5.0 479.85 NaN 2399.25 0.02
5 Sail 10.0 187.75 NaN 1877.50 0.04
6 SBI 3.0 2756.60 NaN 8269.80 NaN
7 SBI 2.0 2191.50 NaN 4383.00 NaN
8 SBI 2.0 1911.50 NaN 3823.00 NaN
9 Reliance 5.0 773.50 NaN 3867.50 0.11
10 JP Associate 13.0 59.50 NaN 773.50 0.19
11 suzlon 13.0 38.78 NaN 504.14 NaN
12 tata steel 10.0 415.87 NaN 4158.70 0.14
13 SAIL 10.0 109.70 NaN 1097.00 NaN
14 ICICI 3.0 857.30 NaN 2571.90 0.08
15 IDFC 10.0 110.45 NaN 1104.50 0.12
```

*Figure 7.35*

**Rows** can also be displayed in **reverse order** using string operator similar to lists or strings.

```
>>>df[::-1]
```

```

>>> df[::-1]
 share quantity ... amount invested profit%
27 Total NaN ... 51825.12 NaN
26 icici 1.0 ... NaN NaN
25 sbi 1.0 ... NaN NaN
24 icici 1.0 ... NaN NaN
23 sbi 2.0 ... NaN NaN
22 icici 6.0 ... NaN NaN
21 ICICI 3.0 ... 2342.10 0.18
20 tatamotors 5.0 ... 774.25 0.21
19 SUZLON 13.0 ... 483.60 NaN
18 IDFC 9.0 ... 972.00 0.15
17 SAIL 9.0 ... 936.90 NaN
16 NaN NaN ... NaN 0.15
15 IDFC 10.0 ... 1104.50 0.12
14 ICICI 3.0 ... 2571.90 0.08
13 SAIL 10.0 ... 1097.00 NaN
12 tata steel 10.0 ... 4158.70 0.14
11 suzlon 13.0 ... 504.14 NaN
10 JP Associate 13.0 ... 773.50 0.19
9 Reliance 5.0 ... 3867.50 0.11
8 SBI 2.0 ... 3823.00 NaN
7 SBI 2.0 ... 4383.00 NaN
6 SBI 3.0 ... 8269.80 NaN
5 Sail 10.0 ... 1877.50 0.04
4 Kotak bank 5.0 ... 2399.25 0.02
3 noida toll 40.0 ... 1340.00 NaN
2 Polyplex Corporation 10.0 ... 4840.00 NaN
1 Powergrid 16.0 ... 1566.08 0.07
0 Coal india 12.0 ... 3740.40 0.15

[28 rows x 6 columns]

```

*Figure 7.36*

## 7.5.2 Working with columns

You can retrieve the column names in a dataframe using the `column` attribute.

```

>>> import pandas as pd
>>> df = pd.read_excel("F:\\data\\shares.xlsx","Sheet1")
>>> df.columns
Index(['share', 'quantity', 'price', 'description', 'amount invested',
 'profit%'],
 dtype='object')

```

Suppose you only want to retrieve the `description` column.

In the following image, you can see that to retrieve only the first five entries of description column, we can use:

```
>>>df.description.head()
```

And to retrieve all the entries of description column, you can simply provide the following instruction:

```
>>>df.description
```

```
>>> df.description.head()
0 POWER
1 NaN
2 4th largest manufacturer of thin polyester Film
3 NaN
4 NaN
Name: description, dtype: object
>>> df.description
0 POWER
1 NaN
2 4th largest manufacturer of thin polyester Film
3 NaN
4 NaN
5 NaN
6 NaN
7 NaN
8 NaN
9 NaN
10 NaN
11 NaN
12 NaN
13 NaN
14 NaN
15 NaN
16 NaN
17 NaN
18 NaN
19 NaN
20 NaN
21 NaN
22 NaN
23 NaN
24 NaN
25 NaN
26 NaN
27 NaN
```

Figure 7.37

There is one more way to access column content. Suppose you want to have a look at the column “*amount invested*”. As there is space between the two words, you can find the result as follows:

```
>>>df['amount invested']
```

The first five entries in the column can be retrieved using the `head()` function as shown here:

```
>>>df['amount invested'].head()
```

```
>>> df['amount invested']
0 3740.40
1 1566.08
2 4840.00
3 1340.00
4 2399.25
5 1877.50
6 8269.80
7 4383.00
8 3823.00
9 3867.50
10 773.50
11 504.14
12 4158.70
13 1097.00
14 2571.90
15 1104.50
16 NaN
17 936.90
18 972.00
19 483.60
20 774.25
21 2342.10
22 NaN
23 NaN
24 NaN
25 NaN
26 NaN
27 51825.12
Name: amount invested, dtype: float64
>>> df['amount invested'].head()
0 3740.40
1 1566.08
2 4840.00
3 1340.00
4 2399.25
Name: amount invested, dtype: float64
>>>
```

*Figure 7.38*

Similarly, the last 5 entries in the “*amount invested column*” can be retrieved using:

```
df['amount invested'].tail()

>>> df['amount invested'].tail()
23 NaN
24 NaN
25 NaN
26 NaN
27 51825.12
Name: amount invested, dtype: float64
>>>
```

*Figure 7.39*

### 7.5.3 Retrieving data from multiple columns

Data from multiple columns can be retrieved using:

```
DataFrame_name. [[col_1,col_2,...]]
```

```
>>> df[['price','amount invested']]
 price amount invested
0 311.70 3740.40
1 97.88 1566.08
2 484.00 4840.00
3 33.50 1340.00
4 479.85 2399.25
5 187.75 1877.50
6 2756.60 8269.80
7 2191.50 4383.00
8 1911.50 3823.00
9 773.50 3867.50
10 59.50 773.50
11 38.78 504.14
12 415.87 4158.70
13 109.70 1097.00
14 857.30 2571.90
15 110.45 1104.50
16 NaN NaN
17 104.10 936.90
18 108.00 972.00
19 37.20 483.60
20 154.85 774.25
21 780.70 2342.10
22 786.25 NaN
23 1727.55 NaN
24 722.10 NaN
25 1636.35 NaN
26 713.40 NaN
27 NaN 51825.12
>>>
```

*Figure 7.40*

The **head()** and **tail()** functions can be used the same way.

```
>>> df[['price','amount invested']].tail(2)
 price amount invested
26 713.4 NaN
27 NaN 51825.12
```

*Figure 7.41*

To calculate the maximum and minimum values, you can make use of the **max()** and **min()** functions.

```
>>>df.price.max()
2756.6
>>>df['amount invested'].min()
483.6
>>>df['profit%'].max()
0.21
```

The `describe()` function is used for displaying information.

```
>>> df.describe()
 quantity price amount invested profit%
count 26.000000 26.000000 22.000000 13.000000
mean 8.230769 676.533846 4711.374545 0.123846
std 7.900925 755.414971 10692.573093 0.057813
min 1.000000 33.500000 483.600000 0.020000
25% 3.000000 108.425000 1003.250000 0.080000
50% 7.500000 447.860000 2109.800000 0.140000
75% 10.000000 784.862500 3856.375000 0.150000
max 40.000000 2756.600000 51825.120000 0.210000
>>>
```

*Figure 7.42*

As you can see the `describe()` function provides information on:

1. Total count of values
2. Average
3. Standard deviation
4. Minimum value
5. Maximum value
6. 25%, 50% and 75% of the total value

## 7.5.4 Retrieving data based on conditions

### **Example 7.10:**

Retrieve share, decription, price values where profit % value is maximum.

### **Answer:**

```
>>> import pandas as pd
>>> import xlrd
>>> df = pd.read_excel("F:\\data\\shares.xlsx","Sheet1")
>>> # retrieve column names
>>> df.columns
Index(['share', 'quantity', 'price', 'description', 'amount invested', 'profit%'],
 dtype='object')
>>> #Retrieve share, decription, price values where profit% was maximum
>>> df[['share','description','price']][df['profit%']==df['profit%'].max()]
 share description price
20 tatamotorsNaN 154.85
```

### **Example 7.11:**

Retrieve share, decription, price values where profit % value is minimum.

```
>>> #Retrieve share, amount invested profit% where profit was minimum
>>> df[['share','amount invested','profit%']][df['profit%']==df['profit%'].max()]
 share amount invested profit%
19 tatamotors 774.25 0.21
>>>
```

### **Example 7.12:**

Retrieve share where amount invested is more than Rs 500.

```
>>> [['share']][df['amount invested']>=500]
 share
0 Coal india
1 Powergrid
2 Polyplex Corporation
3 noida toll
4 Kotak bank
5 Sail
6 SBI
7 SBI
8 SBI
9 Reliance
10 JP Associate
11 suzlon
12 tata steel
13 SAIL
14 ICICI
15 IDFC
16 SAIL
17 IDFC
19 tatamotors
20 ICICI
26 Total
```

## **7.5.5 Index range**

We can retrieve information regarding the index using:

```
>>> import pandas as pd
>>> df = pd.read_excel("F:\\data\\shares.xlsx","Sheet1")
>>> df.index
RangeIndex(start=0, stop=28, step=1)
```

In the beginning of this chapter, we learnt that index column is generated automatically. Any column with unique values can be set as index. Let's add a column by the name '**share\_id**'.

|    | A        | B                    | C        | D      | E                                               | F               | G       |
|----|----------|----------------------|----------|--------|-------------------------------------------------|-----------------|---------|
| 1  | share_id | share                | quantity | price  | description                                     | amount invested | profit% |
| 2  | sh001    | Coal India           | 12       | 311.7  | POWER                                           | 3740.4          | 15%     |
| 3  | sh002    | Powergrid            | 16       | 97.88  |                                                 | 1566.08         | 7%      |
| 4  | sh003    | Polyplex Corporation | 10       | 484    | 4th largest manufacturer of thin polyester Film | 4840            |         |
| 5  | sh004    | noida toll           | 40       | 33.5   |                                                 | 1340            |         |
| 6  | sh005    | Kotak bank           | 5        | 479.85 |                                                 | 2399.25         | 2.00%   |
| 7  | sh006    | Sail                 | 10       | 187.75 |                                                 | 1877.5          | 4%      |
| 8  | sh007    | SBI                  | 3        | 2756.6 |                                                 | 8269.8          |         |
| 9  | sh008    | SBI                  | 2        | 2191.5 |                                                 | 4383            |         |
| 10 | sh009    | SBI                  | 2        | 1911.5 |                                                 | 3,823           |         |
| 11 | sh010    | Reliance             | 5        | 773.5  |                                                 | 3867.5          | 11%     |
| 12 | sh011    | JP Associate         | 13       | 59.5   |                                                 | 773.5           | 19%     |
| 13 | sh012    | suzlon               | 13       | 38.78  |                                                 | 504.14          |         |
| 14 | sh013    | tata steel           | 10       | 415.87 |                                                 | 4158.7          | 14%     |
| 15 | sh014    | SAIL                 | 10       | 109.7  |                                                 | 1097            |         |
| 16 | sh015    | ICICI                | 3        | 857.3  |                                                 | 2571.9          | 8%      |
| 17 | sh016    | IDFC                 | 10       | 110.45 |                                                 | 1104.5          | 12%     |
| 18 |          |                      |          |        |                                                 |                 | 15%     |

*Figure 7.43*

Now, since **share\_id** has unique values, we can set it up as index.

```
>>> import pandas as pd
>>> df = pd.read_excel("F:\\data\\shares.xlsx","Sheet1")
>>> df.index
RangeIndex(start=0, stop=28, step=1)
>>> df1 = df.set_index('share_id')
>>> df1
```

The contents of df1 are as follows:

```

>>> df1
 share quantity ... amount invested profit%
share_id
sh001 Coal india 12.0 ... 3740.40 0.15
sh002 Powergrid 16.0 ... 1566.08 0.07
sh003 Polyplex Corporation 10.0 ... 4840.00 NaN
sh004 noida toll 40.0 ... 1340.00 NaN
sh005 Kotak bank 5.0 ... 2399.25 0.02
sh006 Sail 10.0 ... 1877.50 0.04
sh007 SBI 3.0 ... 8269.80 NaN
sh008 SBI 2.0 ... 4383.00 NaN
sh009 SBI 2.0 ... 3823.00 NaN
sh010 Reliance 5.0 ... 3867.50 0.11
sh011 JP Associate 13.0 ... 773.50 0.19
sh012 suzlon 13.0 ... 504.14 NaN
sh013 tata steel 10.0 ... 4158.70 0.14
sh014 SAIL 10.0 ... 1097.00 NaN
sh015 ICICI 3.0 ... 2571.90 0.08
sh016 IDFC 10.0 ... 1104.50 0.12
sh017 SAIL 9.0 ... 936.90 NaN
sh018 IDFC 9.0 ... 972.00 0.15
sh019 SUZLON 13.0 ... 483.60 NaN
sh020 tatamotors 5.0 ... 774.25 0.21
sh021 ICICI 3.0 ... 2342.10 0.18
sh022 icici 6.0 ... NaN NaN
sh023 sbi 2.0 ... NaN NaN
sh024 icici 1.0 ... NaN NaN
sh025 sbi 1.0 ... NaN NaN
sh026 icici 1.0 ... NaN NaN
sh027 Total NaN ... 51825.12 NaN

[27 rows x 6 columns]

```

*Figure 7.44*

You can see that in this case the content of dataframe df remains the same.  
 We can find content of a share using the `loc` attribute.  
 Suppose you want to find information of share at index 2.

```

>>>df.loc[2]
share_idsh003
share Polyplex Corporation
quantity 10
price 484
description 4th largest manufacturer of thin polyester Film
amount invested 4840
profit% NaN
Name: 2, dtype: object

```

Same information you can retrieve using df1 as shown here:

```

>>>df1.loc['sh003']

```

```
share Polyplex Corporation
quantity 10
price 484
description 4th largest manufacturer of thin polyester Film
amount invested 4840
profit% NaN
Name: sh003, dtype: object
```

If you want to make `share_id` the index for `df`, then you can use the following instruction:

```
>>>import pandas as pd
>>>df = pd.read_excel("F:\\data\\shares.xlsx","Sheet1")
>>>df.set_index('share_id',inplace = True)
>>>df
```

**Output:**

| share_id |          | share        | quantity | ... | amount invested | profit% |
|----------|----------|--------------|----------|-----|-----------------|---------|
| sh001    |          | Coal india   | 12.0     | ... | 3740.40         | 0.15    |
| sh002    |          | Powergrid    | 16.0     | ... | 1566.08         | 0.07    |
| sh003    | Polyplex | Corporation  | 10.0     | ... | 4840.00         | NaN     |
| sh004    |          | noida toll   | 40.0     | ... | 1340.00         | NaN     |
| sh005    |          | Kotak bank   | 5.0      | ... | 2399.25         | 0.02    |
| sh006    |          | Sail         | 10.0     | ... | 1877.50         | 0.04    |
| sh007    |          | SBI          | 3.0      | ... | 8269.80         | NaN     |
| sh008    |          | SBI          | 2.0      | ... | 4383.00         | NaN     |
| sh009    |          | SBI          | 2.0      | ... | 3823.00         | NaN     |
| sh010    |          | Reliance     | 5.0      | ... | 3867.50         | 0.11    |
| sh011    |          | JP Associate | 13.0     | ... | 773.50          | 0.19    |
| sh012    |          | suzlon       | 13.0     | ... | 504.14          | NaN     |
| sh013    |          | tata steel   | 10.0     | ... | 4158.70         | 0.14    |
| sh014    |          | SAIL         | 10.0     | ... | 1097.00         | NaN     |
| sh015    |          | ICICI        | 3.0      | ... | 2571.90         | 0.08    |
| sh016    |          | IDFC         | 10.0     | ... | 1104.50         | 0.12    |
| sh017    |          | SAIL         | 9.0      | ... | 936.90          | NaN     |
| sh018    |          | IDFC         | 9.0      | ... | 972.00          | 0.15    |
| sh019    |          | SUZLON       | 13.0     | ... | 483.60          | NaN     |
| sh020    |          | tatamotors   | 5.0      | ... | 774.25          | 0.21    |
| sh021    |          | ICICI        | 3.0      | ... | 2342.10         | 0.18    |
| sh022    |          | icici        | 6.0      | ... | NaN             | NaN     |
| sh023    |          | sbi          | 2.0      | ... | NaN             | NaN     |
| sh024    |          | icici        | 1.0      | ... | NaN             | NaN     |
| sh025    |          | sbi          | 1.0      | ... | NaN             | NaN     |
| sh026    |          | icici        | 1.0      | ... | NaN             | NaN     |
| sh027    |          | Total        | NaN      | ... | 51825.12        | NaN     |

[27 rows x 6 columns]

*Figure 7.45*

## 7.5.6 Reset index

We can reset the `index` for `df` as shown here:

```
>>>df.reset_index(inplace = True)
>>>df
```

**Output:**

```

>>> df.reset_index(inplace = True)
>>> df
 share_id share ... amount invested profit%
0 sh001 Coal india ... 3740.40 0.15
1 sh002 Powergrid ... 1566.08 0.07
2 sh003 Polyplex Corporation ... 4840.00 NaN
3 sh004 noida toll ... 1340.00 NaN
4 sh005 Kotak bank ... 2399.25 0.02
5 sh006 Sail ... 1877.50 0.04
6 sh007 SBI ... 8269.80 NaN
7 sh008 SBI ... 4383.00 NaN
8 sh009 SBI ... 3823.00 NaN
9 sh010 Reliance ... 3867.50 0.11
10 sh011 JP Associate ... 773.50 0.19
11 sh012 suzlon ... 504.14 NaN
12 sh013 tata steel ... 4158.70 0.14
13 sh014 SAIL ... 1097.00 NaN
14 sh015 ICICI ... 2571.90 0.08
15 sh016 IDFC ... 1104.50 0.12
16 sh017 SAIL ... 936.90 NaN
17 sh018 IDFC ... 972.00 0.15
18 sh019 SUZLON ... 483.60 NaN
19 sh020 tatamotors ... 774.25 0.21
20 sh021 ICICI ... 2342.10 0.18
21 sh022 icici ... NaN NaN
22 sh023 sbi ... NaN NaN
23 sh024 icici ... NaN NaN
24 sh025 sbi ... NaN NaN
25 sh026 icici ... NaN NaN
26 sh027 Total ... 51825.12 NaN

[27 rows x 7 columns]
>>>

```

*Figure 7.46*

## 7.5.7 Sorting data

Data can be sorted using the `sort_values()` function as shown here:

```
>>>df.sort_values('profit%')
```

**Output:**

```

>>> df.sort_values('profit%')
 share_id share ... amount invested profit%
4 sh005 Kotak bank ... 2399.25 0.02
5 sh006 Sail ... 1877.50 0.04
1 sh002 Powergrid ... 1566.08 0.07
14 sh015 ICICI ... 2571.90 0.08
9 sh010 Reliance ... 3867.50 0.11
15 sh016 IDFC ... 1104.50 0.12
12 sh013 tata steel ... 4158.70 0.14
0 sh001 Coal india ... 3740.40 0.15
17 sh018 IDFC ... 972.00 0.15
20 sh021 ICICI ... 2342.10 0.18
10 sh011 JP Associate ... 773.50 0.19
19 sh020 tatamotors ... 774.25 0.21
2 sh003 Polyplex Corporation ... 4840.00 NaN
3 sh004 noida toll ... 1340.00 NaN
6 sh007 SBI ... 8269.80 NaN
7 sh008 SBI ... 4383.00 NaN
8 sh009 SBI ... 3823.00 NaN
11 sh012 suzlon ... 504.14 NaN
13 sh014 SAIL ... 1097.00 NaN
16 sh017 SAIL ... 936.90 NaN
18 sh019 SUZLON ... 483.60 NaN
21 sh022 icici ... NaN NaN
22 sh023 sbi ... NaN NaN
23 sh024 icici ... NaN NaN
24 sh025 shi ... NaN NaN

```

*Figure 7.47*

In case you want the share with highest profit on top, then input the following command:

```
df.sort_values('profit%', ascending = False)
```

## Output:

```

>>> df.sort_values('profit%', ascending = False)
 share_id share ... amount invested profit%
19 sh020 tatamotors ... 774.25 0.21
10 sh011 JP Associate ... 773.50 0.19
20 sh021 ICICI ... 2342.10 0.18
 0 sh001 Coal india ... 3740.40 0.15
17 sh018 IDFC ... 972.00 0.15
12 sh013 tata steel ... 4158.70 0.14
15 sh016 IDFC ... 1104.50 0.12
 9 sh010 Reliance ... 3867.50 0.11
14 sh015 ICICI ... 2571.90 0.08
 1 sh002 Powergrid ... 1566.08 0.07
 5 sh006 Sail ... 1877.50 0.04
 4 sh005 Kotak bank ... 2399.25 0.02
 2 sh003 Polyplex Corporation ... 4840.00 NaN
 3 sh004 noida toll ... 1340.00 NaN
 6 sh007 SBI ... 8269.80 NaN
 7 sh008 SBI ... 4383.00 NaN
 8 sh009 SBI ... 3823.00 NaN
11 sh012 suzlon ... 504.14 NaN
13 sh014 satt ... 1007.00 NaN

```

*Figure 7.48*

## Conclusion

In any field, we advance only if we constantly keep updating ourselves. We can update ourselves only if we are aware of our strength and weaknesses. Same is the case with businesses. The strength and weakness of a business is identified with the help of information that is generally available in the form of data. The data that we deal with in a real world scenario to understand a performance is generally huge, and by converting this data into graphs or maps, we are able to understand the scenario easily. In this chapter, you learnt about various tools that can be used with help of Python for Data Visualization and Analysis.

# CHAPTER 8

## Creating GUI Form and Adding Widgets

### Introduction

In this chapter, you will learn to work with the `tkinter` module. The `tkinter` module is offered by Python to create GUI. It provides the right tools required to work with graphics.

### Structure

- Getting started
- Introduction to widgets
- Layout management
  - Pack
  - Grid
  - Place
- Working with buttons and messagebox
- Canvas
- Frame
- Working with Labels
- Mini Project
- Listbox widgets
- Menubutton and menu
- Radiobuttons
- Scrollbars and sliders
- Spinbox

### Objectives

After reading this chapter, you will:

- Understand how to use the `tkinter` module and work with various types of widgets

- Be able to create a basic functional GUI application

*'tkinter'* allows you to work with the classes of the TK module of **Tool Command Language (TCL)**, which is known to be apt for web and desktop applications along with networking, administration, and so on. TK stands for tool kit, which is used by TCL to create graphics. TK provides standard GUI, which can be used by dynamic programming languages such as Python. Python developers can access TK with the help of the **tkinter** module. Python offers multiple ways for developing GUI. However, the easiest and the fastest way to create GUI is using Python with tkinter. Hence, **tkinter** is the most commonly used module for GUI.

## **8. 1 Getting started**

Let's start working with GUI. The first thing that you need to know is that in order to work with GUI, you need to import tkinter.

### **Importing tkinter**

The first and the most important step involve in working with GUI applications is to import **tkinter**. This would be the first step for all the code on **Graphical User Interface (GUI)**.

You can use any one of the following statements:

```
from tkinter import*
```

or

```
import tkinter as t
```

### **Example 8.1:**

Write a code to create a main window object.

#### **Answer:**

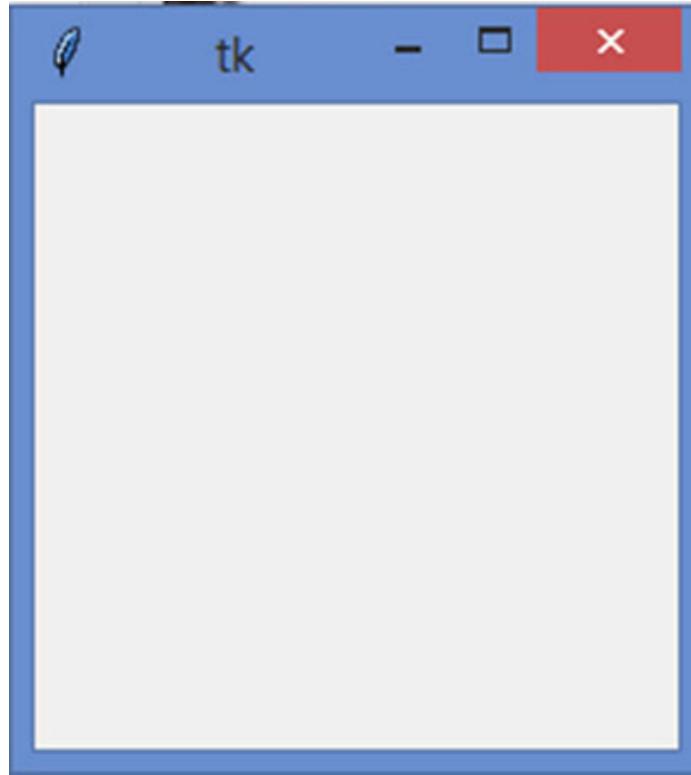
To create the main window, you can input the following command:

```
>>> import tkinter
>>> mw =Tk()
```

or

```
>>> import tkinter as t
>>> # create main window by the name mw(it can actually be anything)
>>> mw = t.Tk()
```

This would open the main window as shown here:



*Figure 8.1*

Now, it's important to introduce you to a very important method called `mainloop()`, which must be called when you are prepared to run your application. The `mainloop()` method is an infinite loop that is used to run the application. It waits for an event to occur and processes the event as long as the window is not closed.

So, the code for starting a simple window is as follows:

```
>>> import tkinter
>>> mw = tkinter.Tk()
>>> mw.mainloop()
```

or

```
>>> import tkinter as t
>>> mw = t.Tk()
>>> mw.mainloop()
```

### **Example 8.2:**

Write a code to give a title to root Window.

#### **Answer:**

You can set the title of the window using the `title()` method.

The following code shows how to set the title for the root window:

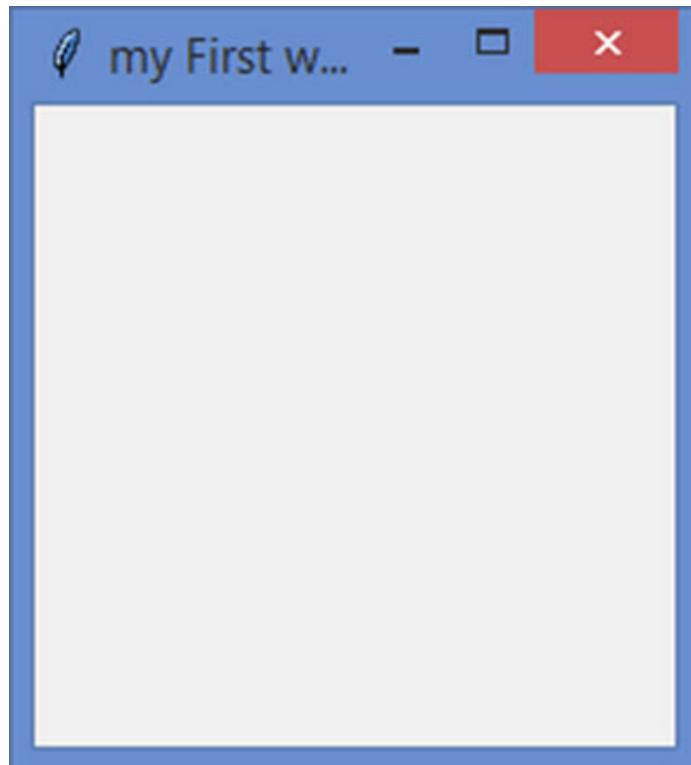
```
#import tkinter
from tkinter import*

#create root window
mw = Tk()

#Set the title for your root window
mw.title("my First window")

mw.mainloop()
```

The output will be as follows:



*Figure 8.2*

You may find the size of the window to be very small. Do not worry. You can change the size with the help of the `geometry()` function as shown in the next example.

### **Example 8.3:**

Write a code to define the size of the window.

You can set the size of the window using the `geometry()` function as follows:

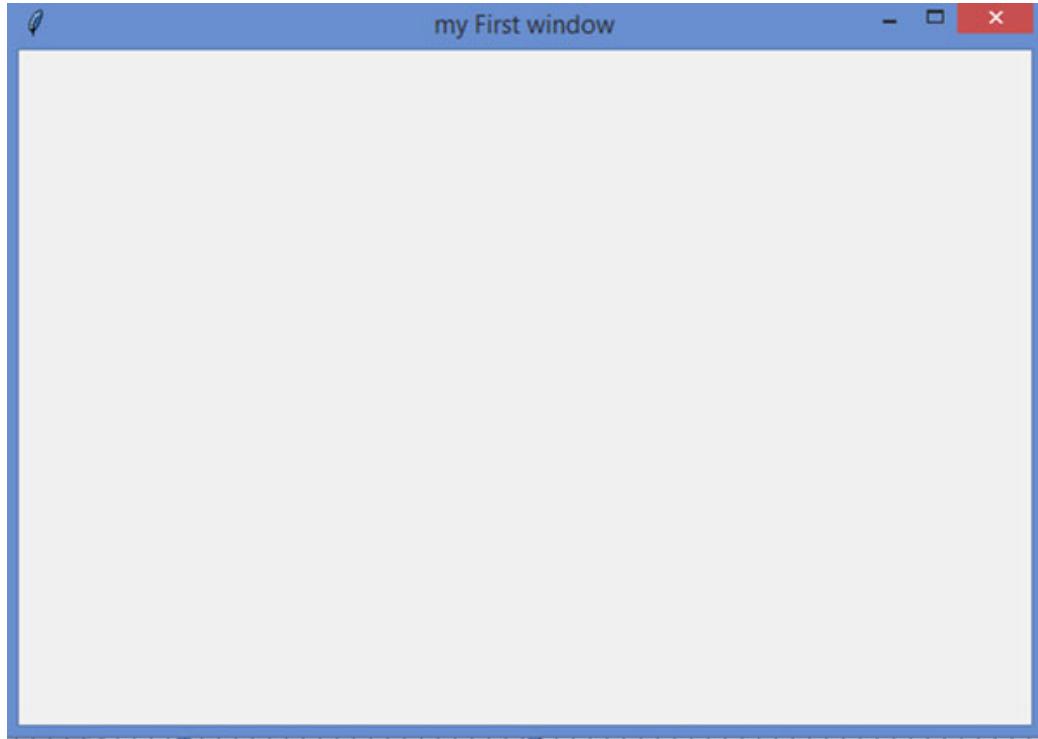
```
#import tkinter
from tkinter import*

#create root window
mw = Tk()
```

```
#Set the title for your root window
mw.title("my First window")

#Set the size of the window
mw.geometry("600x400")

mw.mainloop()
```



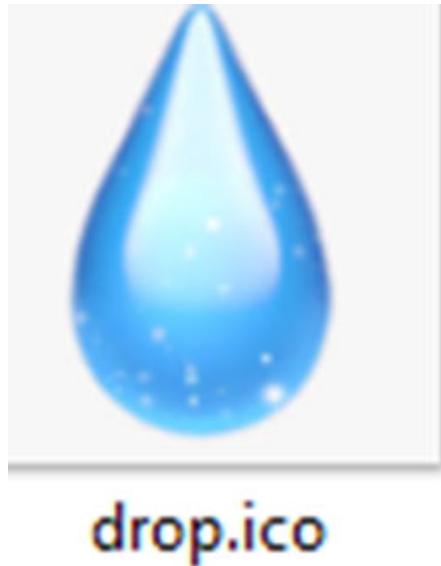
*Figure 8.3*

#### **Example 8.4:**

Write a program to change the image of the root window.

#### **Answer:**

Now, let's try to change the root window's leaf image. For this example we will use the following image of a water drop, which is stored in a directory. For, this book it is stored at **F:/my\_images** folder.



*Figure 8.4*

*To replace the image of the leaf with a new image, it is important that the image file is of .ico type only. The method used for this purpose is `wm_iconbitmap()`.*

```
#import tkinter
from tkinter import*

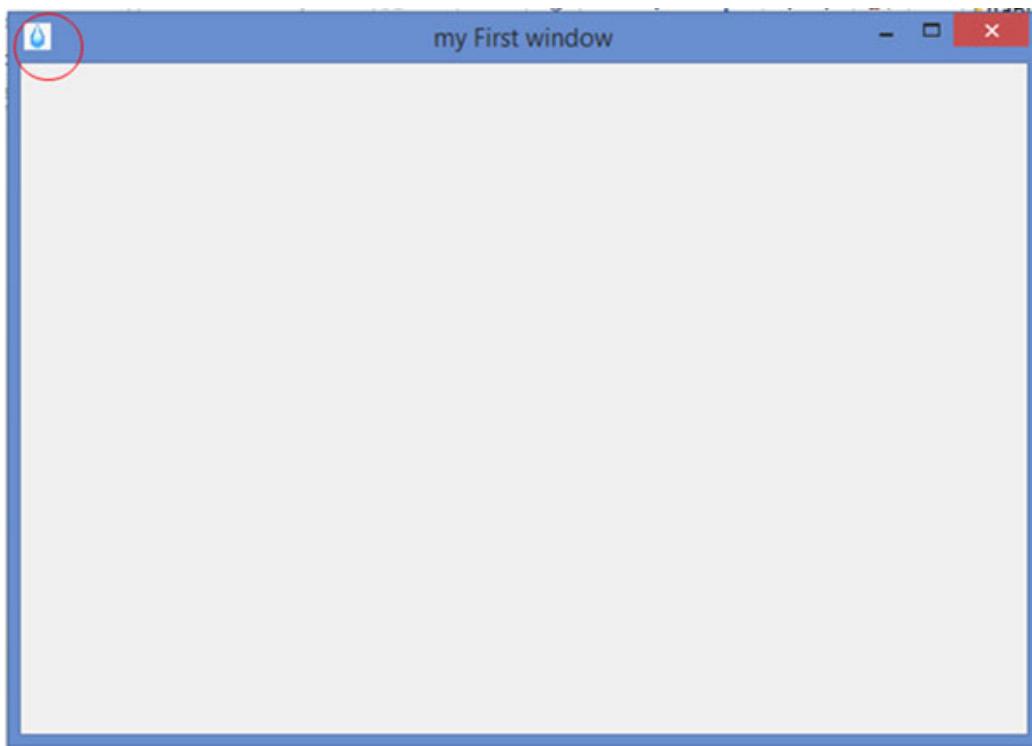
#create root window
mw = Tk()

#Set the title for your root window
mw.title("my First window")

#Set the size of the window
mw.geometry("600x400")

#set the image icon for the window (For this example the image is at F:/my_images/drop.ico)
#Provide the location of the image.
mw.wm_iconbitmap("F:/my_images/drop.ico")

mw.mainloop()
```



*Figure 8.5*

## **8.2 Introduction to widgets**

In this section, we will work with the widgets that are nothing but standard GUI elements such as buttons, labels, and so on. Before proceeding further, you must know that in order to work with the widgets, you must perform the following tasks:

**Step 1:** Import the `tinker` module.

**Step 2:** Create a root/main window for GUI.

**Step 3:** Set the title for the root window.

**Step 4:** Write the code for working with widgets.

**Step 5:** Call `mainloop()` in order to take action against each event triggered by the user.

So, the basic structure of your code would be something like this:

```
#step 1: import tkinter
from tkinter import*

step 2: create root window
mw = Tk()

step 3: Set the title for the root window(optional)
mw.title("my First window")

#step 4: write the code for working with widgets
----- Your code comes here
```

```
#step 5:Call mainloop()
mw.mainloop()
```

### Some Important Widgets

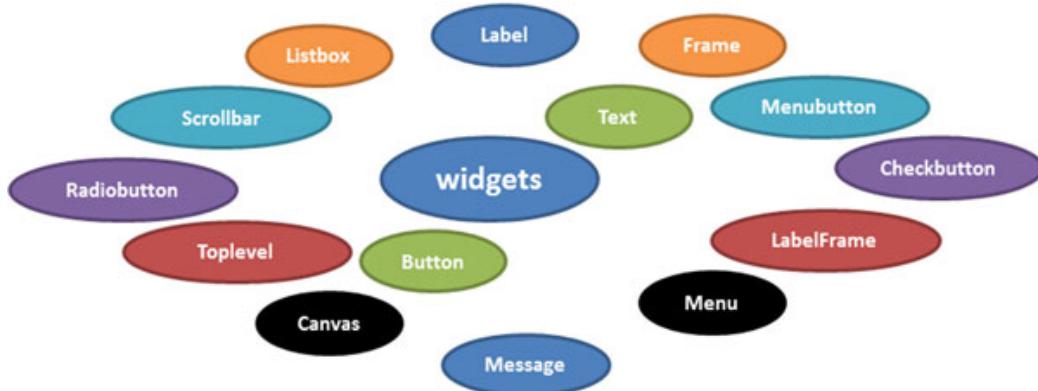


Figure 8.6

## 8.2.1 Layout management

Layout managers (also known as `geometry managers`) play a very important role in designing of GUI applications. The `tkinter` module has 3 layout managers to offer. These layout managers should not be used together with each other in the same parent window. The layout managers are as follows:

1. `pack`
2. `grid`
3. `place`

The three layout managers mentioned above are used to place widgets on to the parent window. Their task is to arrange the position of the widget on the screen. While adding a widget, you decide the size of the widget, but their position on the parent window depends on the `layout manager` settings.

### 8.2.1.1 pack

The `pack()` is the easiest to use and most of the examples in this book uses the `pack()` geometry manager. In case of `pack()`, you need not specifically define where the widget must be displayed. The position is declared with respect to other widgets and `pack()` does the needful. Although `pack()` is the easiest layout manager to use, the `grid()` and `place()` methods offer more flexibility to the developer.

#### Syntax:

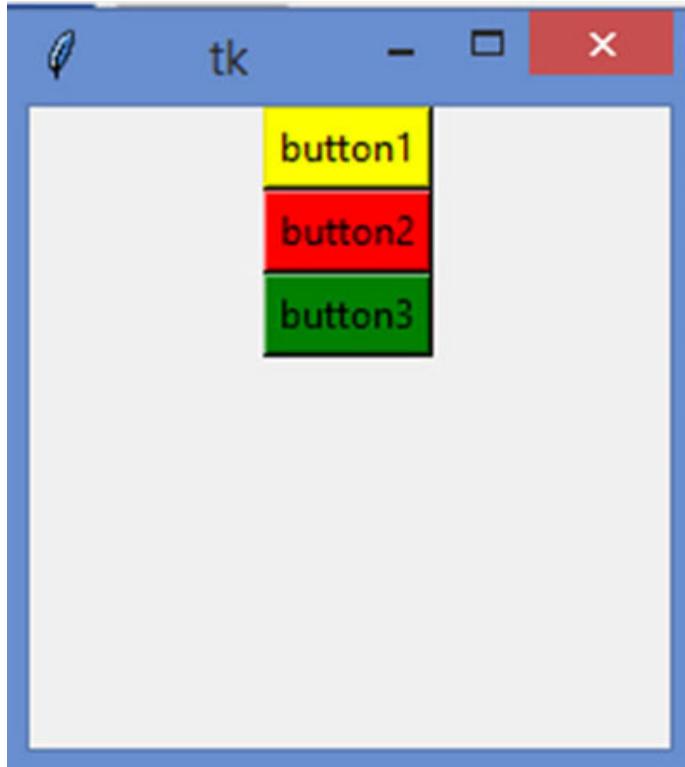
```
widget_name.pack(options)
```

## pack methods

Let's say we have three buttons - `my_first_button`, `my_second_button`, and `my_third_button`. Now, if you simply use `pack()` with them:

```
my_first_button.pack()
my_second_button.pack()
my_third_button.pack()
```

The buttons appear as follows:



*Figure 8.7*

`expand` can have two values: `True` or `False`

```
my_first_button.pack(expand = True)
my_second_button.pack(expand = True)
my_third_button.pack(expand = True)
```

This would display the buttons as shown in following figure ([Figure 8.8](#)):

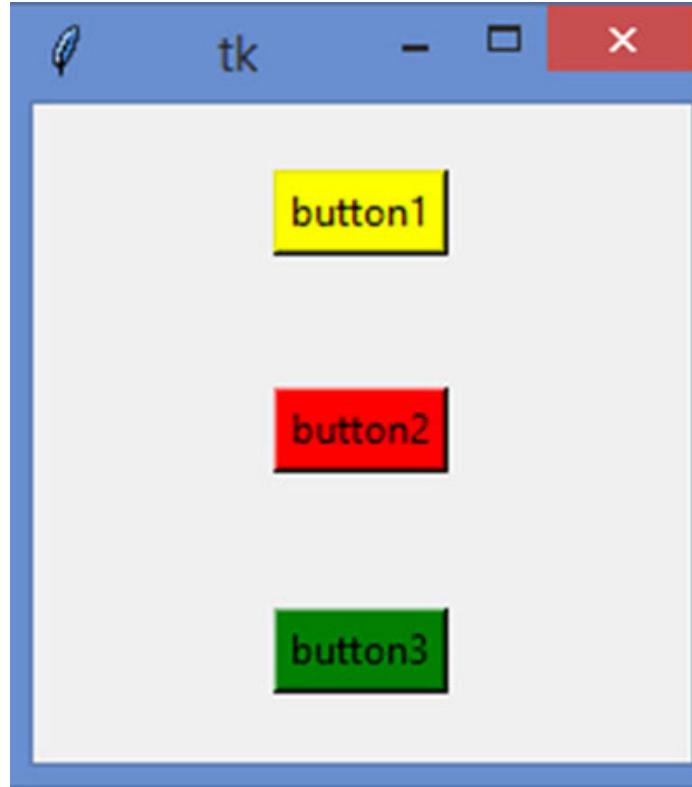
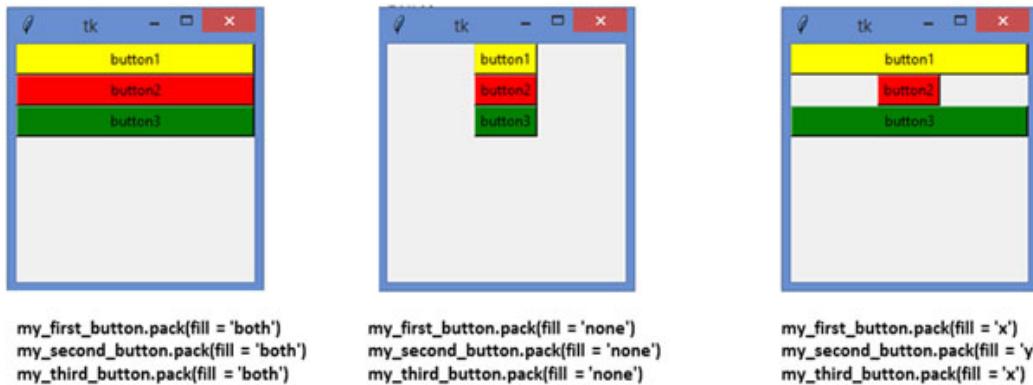


Figure 8.8

'fill' can have the values : 'x', 'y', 'both', 'none,' as shown in following [figure 8.9](#):



#### fill option for pack()

Figure 8.9

**side** : Decides which side of the parent window will the widget appear. The four values are - top, bottom, left, right.

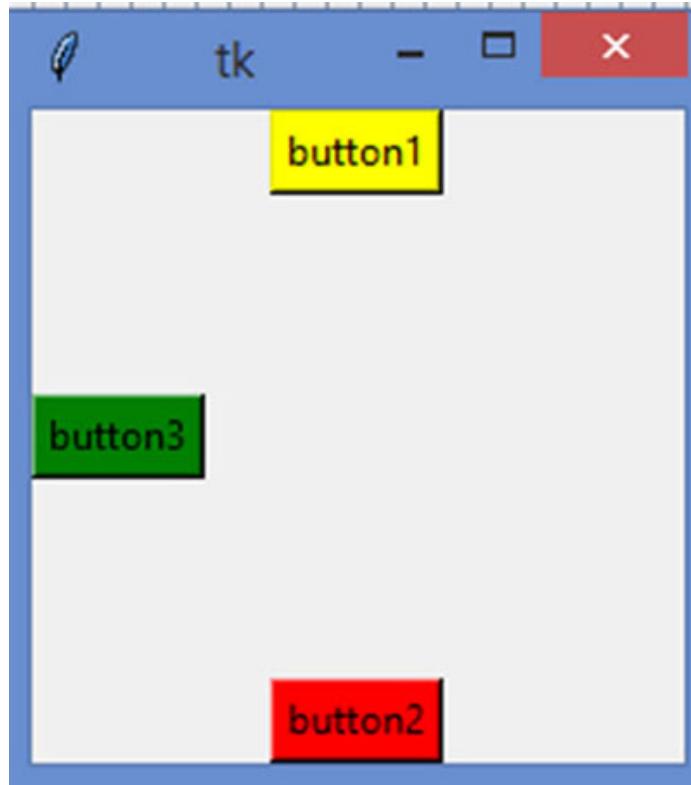


Figure 8.10

```
my_first_button.pack(side = 'top')
my_second_button.pack(side = 'bottom')
my_third_button.pack(side = 'left')
```

### 8.2.1.2 Grid

**grid** is another method that allows you to organize the widgets in a table-like structure. It is one of the most preferred methods of layout geometry. Often used with dialog boxes, the **grid** method allows you to position the widgets based on the position coordinates of the grid. Methods used in grid layout are as follows:

| Grid Method    | Explanation                                                                                  |
|----------------|----------------------------------------------------------------------------------------------|
| column and row | The coordinates that determine the position of the widget.                                   |
| ipadx, ipady   | Tells how much horizontal and vertical padding must be provided inside the widget's border.  |
| padx, pady     | Tells how much horizontal and vertical padding must be provided outside the widget's border. |
| Columnspan     | Tells how many columns are occupied by the widget. The default value is 1.                   |
| Rowspan        | Tells how many rows are occupied by the widget. The default value is 1.                      |
| sticky         | It defines how a widget must stick to the cell when it is smaller than the cell:             |

- W – stick to left
- E – stick to right
- N – stick to top
- S – stick to bottom

**Table 8.1:** Methods used in a Grid Layout

### Code:

```
import tkinter as tk

mw = tk.Tk()

labelusern = tk.Label(mw, text = "User Name")
labelusern.grid(column=0, row=0, ipadx=5, pady=5, sticky=tk.W+tk.N)

labelpwd = tk.Label(mw, text = "Password")
labelpwd.grid(column=0, row=1, ipadx=5, pady=5, sticky=tk.W+tk.S)

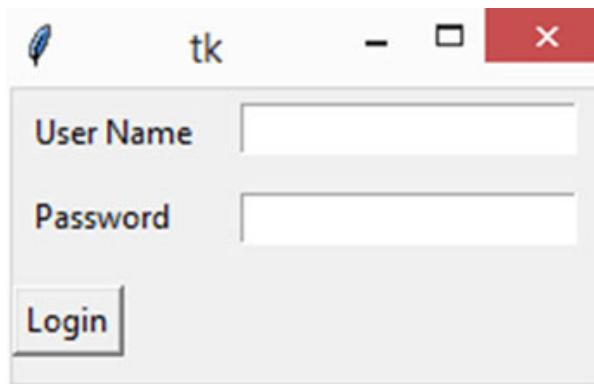
entryusern = tk.Entry(mw, width=20)
entrypwd = tk.Entry(mw, width=20)

entryusern.grid(column=1, row=0, padx=10, pady=5, sticky=tk.N)
entrypwd.grid(column=1, row=1, padx=10, pady=5, sticky=tk.S)

loginButton = tk.Button(mw, text = 'Login')
loginButton.grid(column=0, row=2, pady=10, sticky=tk.W)

mw.mainloop()
```

### Output:



**Figure 8.11**

### 8.2.1.3 Place

This method allows you to place widgets in a window in an **absolute** or **relative** position. The following code is an example of absolute position:

```
import tkinter as tk
```

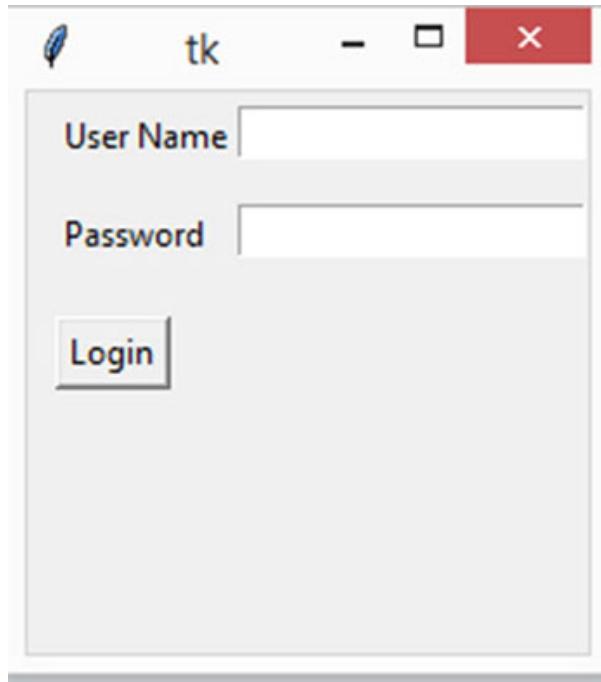
```

mw = tk.Tk()
#define widgets

labelusern = tk.Label(mw, text = "User Name")
labelpwd = tk.Label(mw, text = "Password")
entryusern = tk.Entry(mw, width=20)
entrypwd = tk.Entry(mw, width=20)
loginButton = tk.Button(mw, text = 'Login')

#position widgets
labelusern.place(x = 10, y = 5)
labelpwd.place(x = 10, y = 40)
entryusern.place(x = 75, y = 5)
entrypwd.place(x = 75, y = 40)
loginButton.place(x = 10, y = 80)
mw.mainloop()

```



*Figure 8.12*

## Place with relative position

In the following code, notice the values of `relx` and `rely` are the relative percentage of the widget position to the window size. For example, `relx=0.01, rely=0.15` means the widget is placed in the 1% of the window width and 15% of the window height.

```

import tkinter as tk

mw = tk.Tk()
#define widgets

labelusern = tk.Label(mw, text = "User Name")
labelpwd = tk.Label(mw, text = "Password")

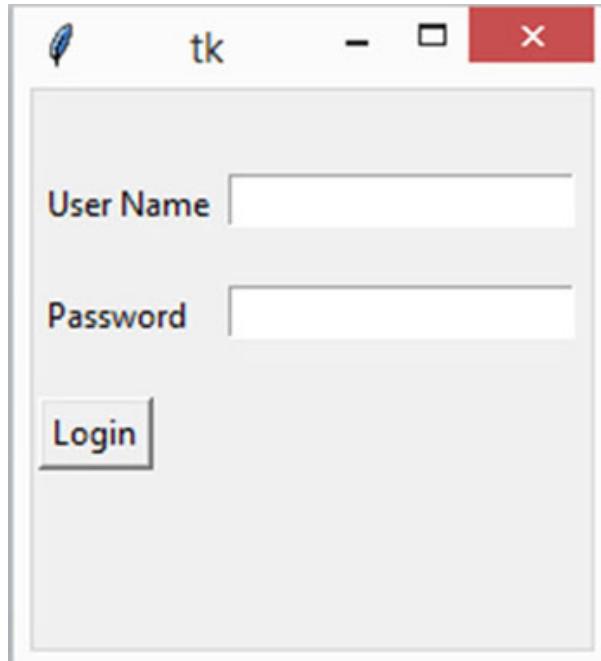
```

```

entryusern = tk.Entry(mw, width=20)
entrypwd = tk.Entry(mw, width=20)
loginButton = tk.Button(mw, text = 'Login')

#position widgets
labelusern.place(relx = 0.01, rely = 0.15)
labelpwd.place(relx = 0.01, rely = 0.35)
entryusern.place(relx = 0.35, rely = 0.15)
entrypwd.place(relx = 0.35, rely = 0.35)
loginButton.place(relx = 0.01, rely = 0.55)
mw.mainloop()

```



*Figure 8.13*

### 8.3 Working with buttons and Messagebox

**Button** is one of the most commonly used widget for user interaction. On the click of the button, some event takes place. This event is generally defined in a function that is called when we click on the button. You can place a meaningful text or image on top of the button to describe its purpose.

**Syntax for button:**

```

button_name = Button(mw, option = value)
where,

```

**mw** = Parent window, which in this case, is the root window.

**option = value**: These are key value-pairs, separated by comma. Such as background color, font, image, border, and so on.

### 8.3.1 Code for displaying a button

#### **Step 1: Import tkinter**

Import tkinter to get all the functionality of Tk.

```
from tkinter import*
```

#### **Step 2: Create an instance of the window**

```
mw = Tk()
```

#### **Step 3: Set the size of the window**

```
mw.geometry("150x150")
```

#### **Step 4: Create an instance of a button**

The first argument `mw` is the instance of the window with which the button will be associated. The second argument is the `text` that would be displayed on the button. However, this would not display the button on the window yet.

```
my_first_button = Button(mw, text="Click Here")
```

#### **Step 5: Use pack() method to pack the button on to the window**

```
my_first_button.pack()
```

#### **Step 6: Call the mainloop()**

```
mw.mainloop()
```

#### **Code:**

```
#import tkinter to get all the functionality of Tk
from tkinter import*

#Create instance of window
mw = Tk()

#Set the size of the window
mw.geometry("150x150")

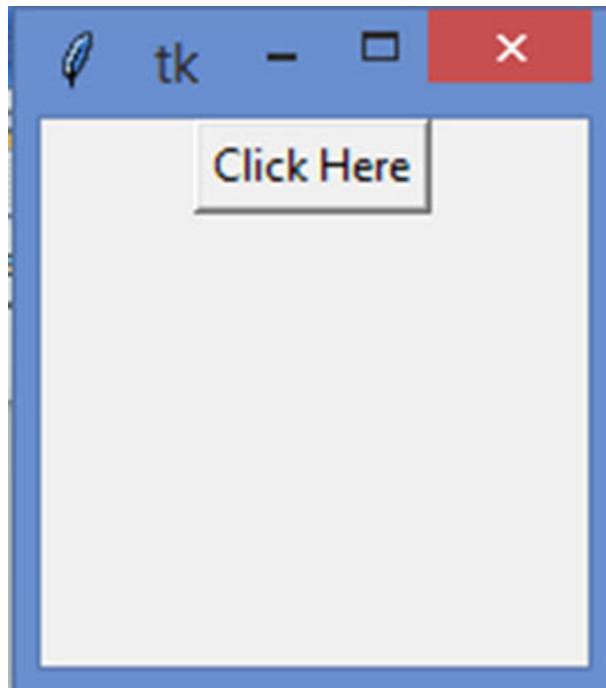
#Create instance of a button.
my_first_button = Button(mw, text="Click Here")

#For the button to display pack the button on to the window
my_first_button.pack()

#Call the mainloop()
mw.mainloop()
```

---

## Output:



*Figure 8.14*

In the preceding example, all that we have done is a simple display of button. However, we have not defined the purpose of this button or the code associated with it.

### Example 8.5

Write a code that at the click of the button we want to display a message box with a message “Welcome to the world of Widgets.”

The steps involved are as follows:

#### Step 1: import statements

We want to display a message box on click of button. So, for that, we have to import the `tkinter.message` box, which is a module and not a class.

```
from tkinter import*
```

#### Step 2: Define the function that would be called on click of the button

The `showinfo()` function takes two parameters. The first parameter is a string that would be the title of the window and the second parameter is also a string that is the message to be displayed.

```
def message_display():
 tkinter.messagebox.showinfo("Welcome Note","welcome to the world of Widgets")
```

### **Step 3: Create instance of window**

```
mw = Tk()
```

### **Step 4: Set the size of the window**

```
mw.geometry("150x150")
```

### **Step 5: Create instance of a button**

The first argument `mw` is the instance of the window with which the button will be associated. The second argument is the text that would be displayed on the button. The third argument is the `command` that defines the action that we want to take on click of the button. In this case, we want to call the function defined in `step 2`. Please note `command` can be a function, bound method, or any other callable Python object. If this option is not used, nothing will happen when the user presses the button.

```
my_first_button = Button(mw, text="Click Here", command = message_display)
```

### **Step 6: For the button to display, pack the button on to the window**

```
my_first_button.pack()
```

### **Step 7: call the mainloop()**

```
mw.mainloop()
```

### **Code:**

```
#import statements
from tkinter import*
import tkinter.messagebox

Create a function for button
def message_display():
 tkinter.messagebox.showinfo("Welcome Note","welcome to the world of Widgets")

#Create instance of window
mw = Tk()

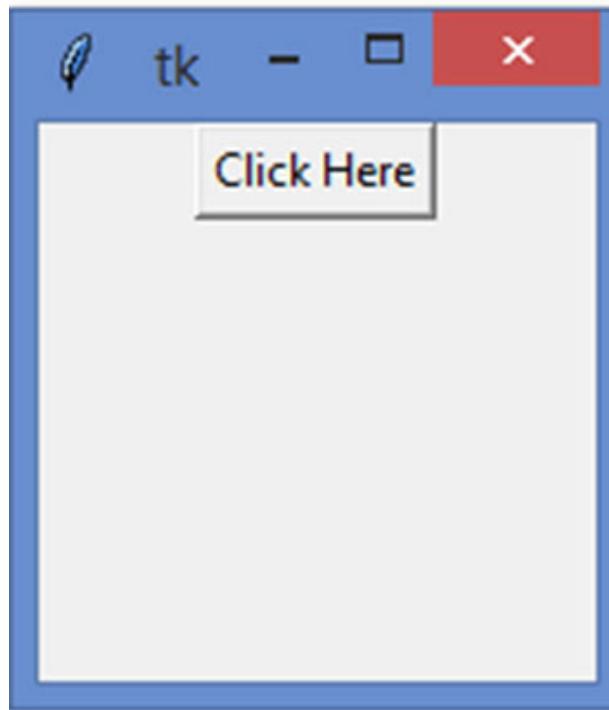
#set the size of the window
mw.geometry("150x150")

#Create instance of a button.
my_first_button = Button(mw, text="Click Here", command = message_display)

#for the button to display pack the button on to the window
my_first_button.pack()

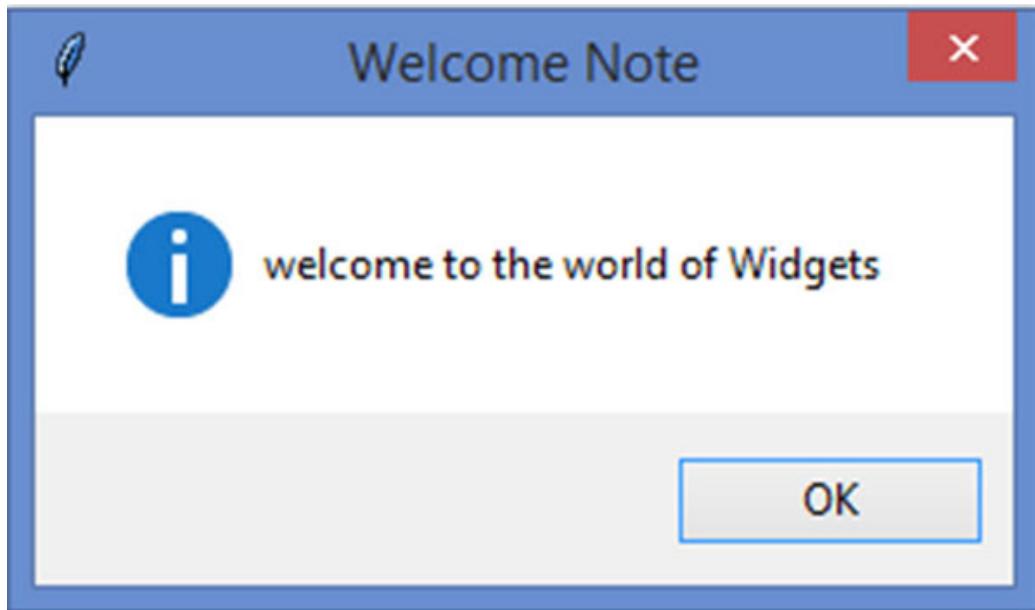
#call the mainloop()
mw.mainloop()
```

**Output:**



*Figure 8.15*

On clicking the button, the following message would appear:



*Figure 8.16*

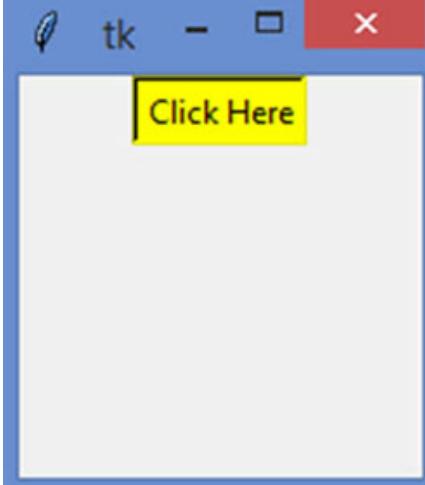
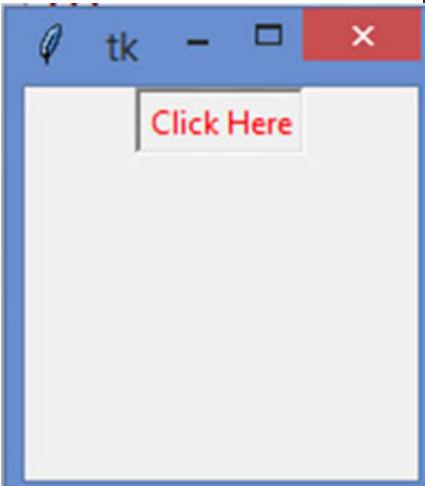
So, the syntax for creating a button is as follows:

```
[REDACTED]
```

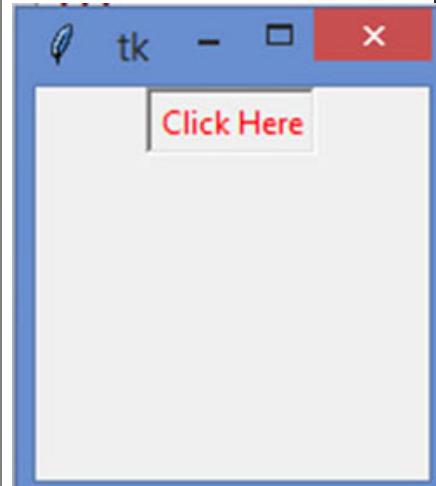
**Syntax:**

```
button_name = Button(master, option = value)
```

Here `master` stands for the parent window, and the options are key-value pair separated by comma.

| Option           | Description                                                  | Code                                                                                                                         | Output                                                                                |
|------------------|--------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| activebackground | Color of the button when the cursor presses the button       | <pre><code>my_first_button = Button(mw, activebackground = "yellow", text="Click Here")</code></pre>                         |    |
| activeforeground | Color of the foreground of the button when under the cursor. | <pre><code>my_first_button = Button(mw, activeforeground = "red", text="Click Here", command = message_display)</code></pre> |  |
| bd               | Border width. The default value for border is 2.             | <pre><code>my_first_button = Button(mw, bd = "10", text="Click Here")</code></pre>                                           |                                                                                       |

**Figure 8.17:** `activebackground='yellow'`



**Figure 8.18:** `activeforeground = 'red'`

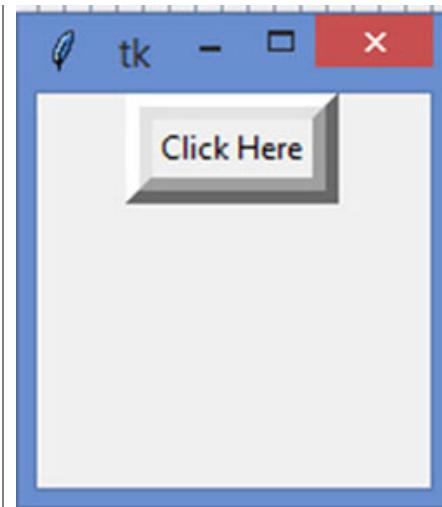


Figure 8.19: `bd='10'`

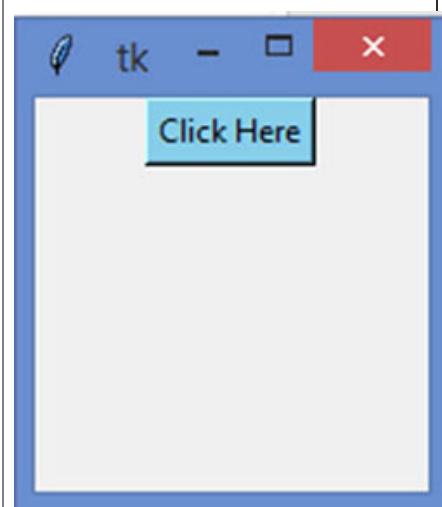


Figure 8.20: `bg='skyblue'`

|         |                                                                                    |                                                                                       |  |
|---------|------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|--|
| bg      | The background color                                                               | <pre>my_first_button = Button(mw,bg ="sky blue", text="Click Here")</pre>             |  |
| command | Action to be taken on the click of a button. Already seen in the previous example. | <pre>my_first_button = Button(mw, text="Click Here", command = message_display)</pre> |  |
| fg      | Normal foreground/text color.                                                      | <pre>my_first_button = Button(mw,fg ="purple", text="Click Here")</pre>               |  |

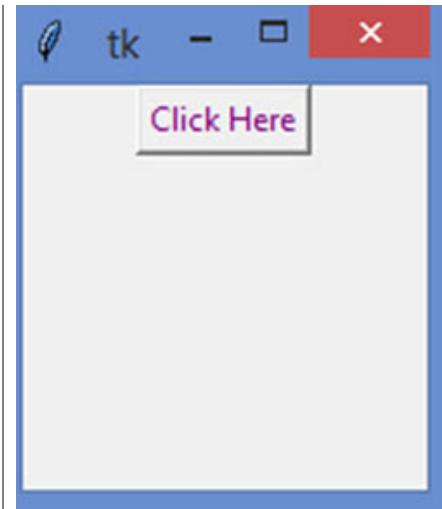


Figure 8.21: fg='purple'

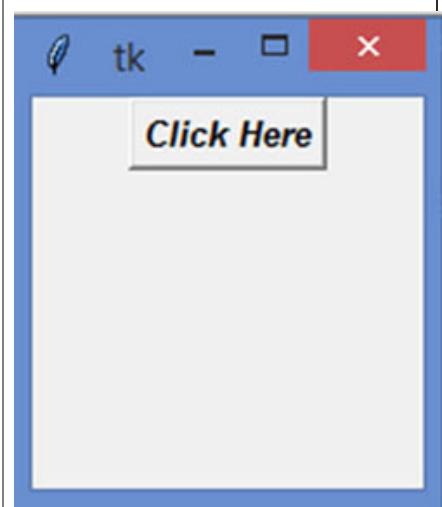
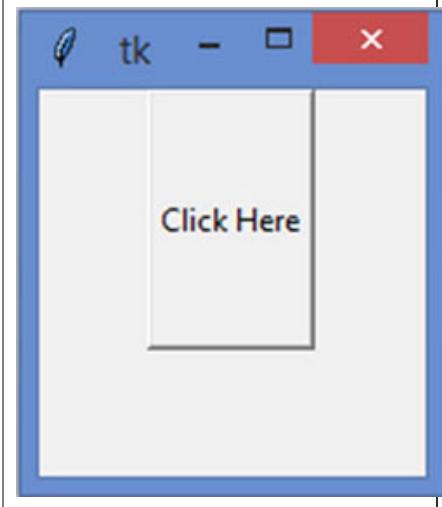
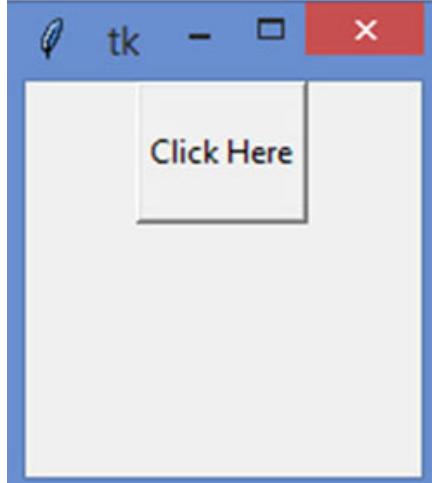
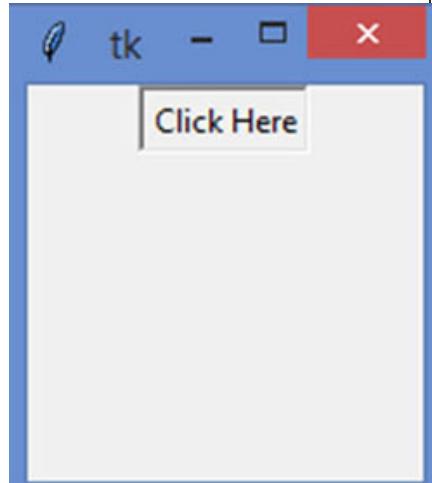
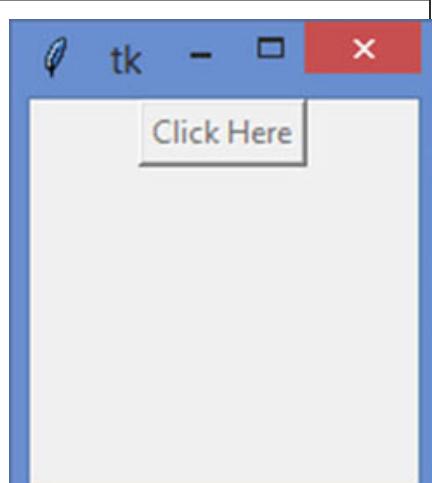
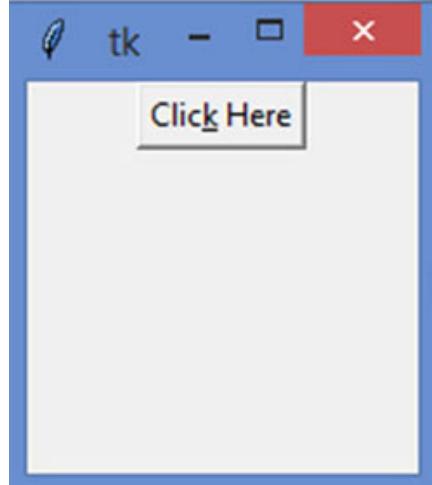


Figure 8.22: font = "Helvetica 10 bold italic"



|        |                                                                         |                                                                      |                                                                                      |
|--------|-------------------------------------------------------------------------|----------------------------------------------------------------------|--------------------------------------------------------------------------------------|
|        |                                                                         |                                                                      | <b>Figure 8.23: height = "6"</b>                                                     |
| pady   | Additional padding above and below text.                                | my_first_button<br>Button(mw, text="Click Here", pady = "15")        |    |
| relief | Defines type of border : flat, groove, raised, ridge, solid, or sunken. | my_first_button<br>Button(mw, text="Click Here", relief = "sunken")  |   |
| state  | State of the button: active, disabled, normal.                          | my_first_button<br>Button(mw, text="Click Here", state = "disabled") |  |

|           |                                                                                                          |                                                                  | <b>Figure 8.26: state = "disabled"</b>                                                                                    |
|-----------|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| underline | If -1, then no character is underlined. If non-negative, then the corresponding character is underlined. | my_first_button = Button(mw, text="Click Here", underline = "4") | <br><b>Figure 8.27: underline = "4"</b> |

**Table 8.2**

### Example 8.6

Write a program for two buttons: one on the left and the other on the right. Try it out, and match your results with the output given in the book.

**Answer:**

**Code:**

```
#import Statements
from tkinter import*
import tkinter.messagebox

Create a function for right button
def message_display_right():
 tkinter.messagebox.showinfo("Next Topic","Welcome to Canvas")

Create a function for right button
def message_display_left():
 tkinter.messagebox.showinfo("Previous Topic","Welcome to Widgets")

#Create instance of window
mw = Tk()
mw.title("Select Topic")
#Create instance of a button

my_first_button = Button(mw, text="Next", fg="Green", command = message_display_right)
my_second_button = Button(mw, text="Previous", fg="Red", command = message_display_left)

#Adjust the position of buttons
my_first_button.pack(side = tkinter.RIGHT)
my_second_button.pack(side = tkinter.LEFT)
```

```
#call the mainloop()
mw.mainloop()
```

### Output:

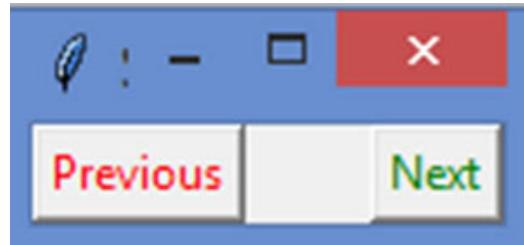


Figure 8.28

On clicking **Next**:

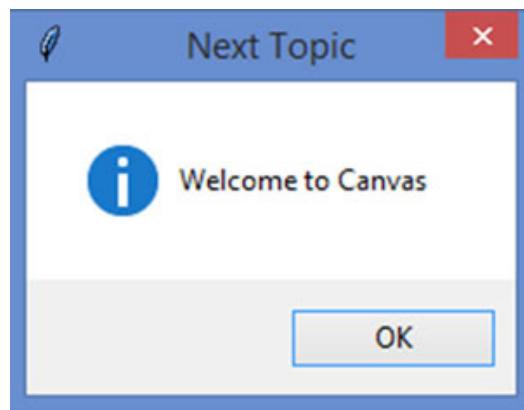


Figure 8.29

On clicking **Previous**:

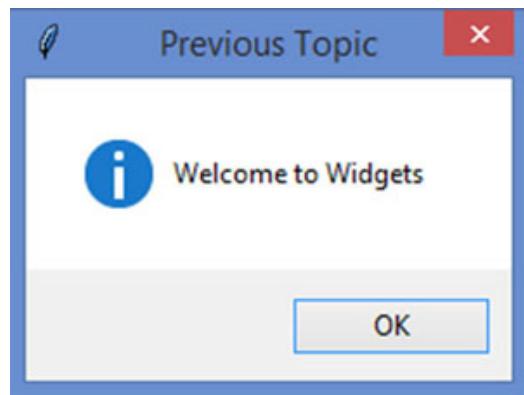


Figure 8.30

### Example 8.7:

Write a code to add an image (available at `f:\\my_images\\thumb2.gif`) to the button.

## **Answer:**

### **Code:**

```
#import Statements
from tkinter import*
import tkinter.messagebox
from tkinter import ttk

Create a function for right button
def message_display():
 tkinter.messagebox.showinfo("Experimenting with GIFs", "Welcome to GIFs")

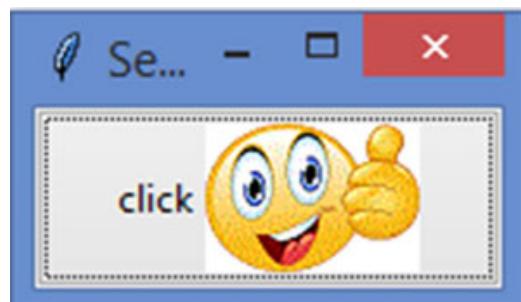
#Create instance of window
mw = Tk()
mw.title("Thumbs UP")

#Create instance of a button
my_first_button = ttk.Button(mw, text="click", command = message_display)
my_first_button.pack()

#Add image to button
img = PhotoImage(file = "f:\\my_images\\thumb2.gif")
my_first_button.config(image = img, compound = RIGHT)

#call the mainloop()
mw.mainloop()
```

### **Output:**



*Figure 8.31*

On click:

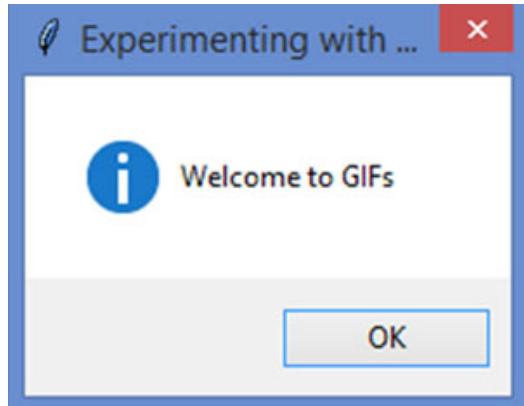


Figure 8.32

## 8.4 Canvas

In this section, we will learn about the Canvas which is a rectangular shaped area that can be used for drawing images.

### Syntax:

```
c = Canvas(master, option = value...)
```

Here, ‘c’ is the `Canvas` class object, ‘`master`’ is the root window, and option refers to the commonly used options with the Canvas object. All option=value pairs are separated from each other with a comma.

Before we start exploring canvas, let’s just have a look at the basic steps involved in associating a canvas to the root window.

### Step 1: Import tkinter

```
from tkinter import*
```

### Step 2: Create instance of window

```
mw = Tk()
```

### Step 3: Create instance of Canvas class

In this case, the dimension of 200 x 200 is taken.

You can give any value as per your desire.

```
mc = Canvas(mw, width = 200, height = 200)
```

### Step 4: Add canvas object mc on to the window object mw

```
mc.pack()
```

### Step 5: Call mainloop()

```
mw.mainloop()
```

Your code should now look like this:

#### Code:

```
#Import tkinter
from tkinter import*

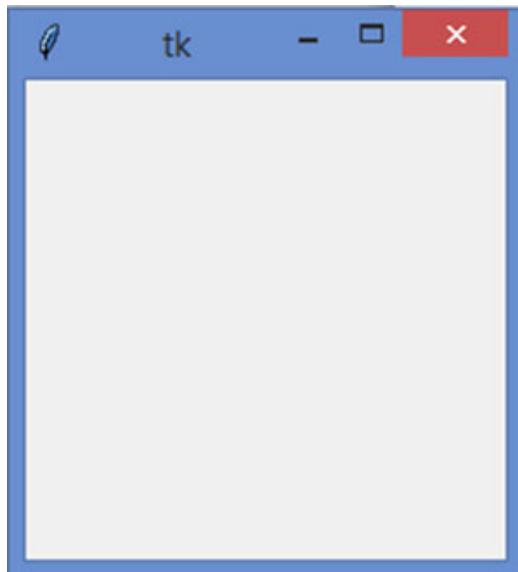
#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#Add canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

#### Output:



*Figure 8.33*

Now, let's do the analysis of the output window.

Look at the following illustration ([Figure 8.34](#)). The area marked in orange is the canvas on which one can draw. The positive x axis is towards the right and the positive y axis is downwards. Since the canvas dimensions are  $200 \times 200$ . The top left point has

$(x = 0, y = 0)$ , the top right point of canvas is at  $x = 200, y = 0$ . Similarly, the bottom left point of the canvas is at  $x = 0$  and  $y = 200$  and the bottom right corner is at  $x = 200$  and  $y = 200$ .

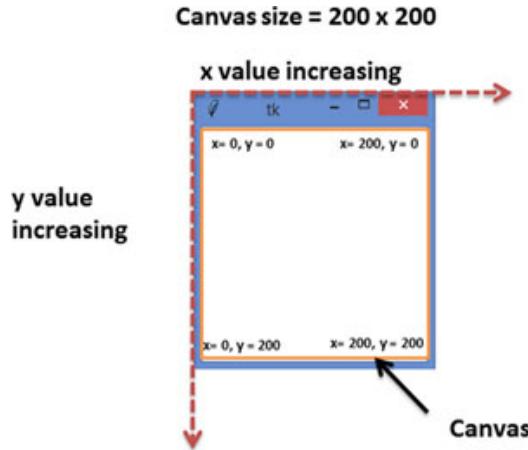


Figure 8.34

So, now let's try to draw a line that joins the top-left corner with the bottom-right corner of the canvas.

The syntax for creating a line is as follows:

**Syntax:**

```
line = Canvas.create_line(x0, y0, x1, y1,...xn, yn, other options)
```

We will draw a blue line on the canvas which is defined by the option **fill**. So, we are adding just one line of code to the preceding example.

```
#Import tkinter
from tkinter import *

#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create line
line = mc.create_line(0,0,200,200,fill = "Blue")

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

**Output:**

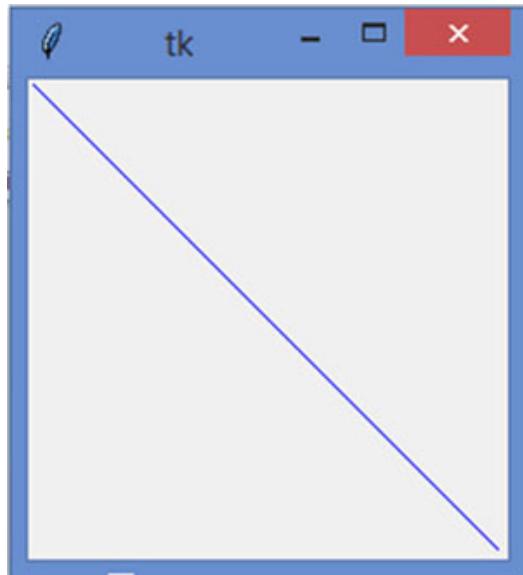


Figure 8.35

**Example 8.8:**

Join points 2, 3 and 4 given in [Figure 8.36](#):

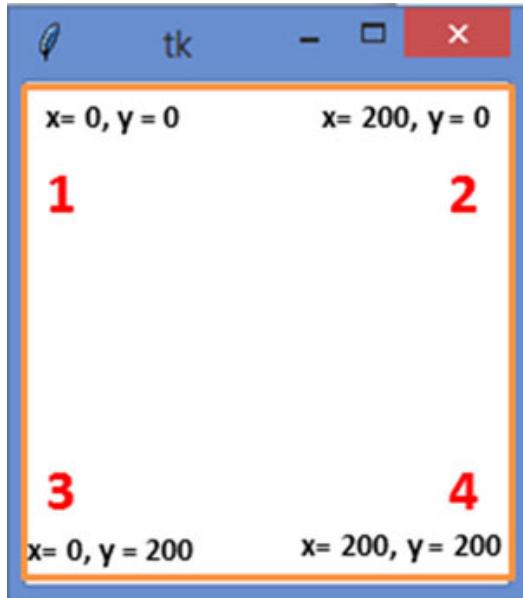


Figure 8.36

So, what we are looking at is :

1. Point 2 (200,0) connects to point 3(0,200)
2. Point 3 (0,200) connects to point 4 (200,200)
3. Point 4 (200,200) connects to point 2(200,0)

Therefore,

- x0, y0 = 200,0
- x1, y1 = 0,200
- x2, y2 = 200,200
- x3, y3 = 200,0

x0,y0      x1,y1      x2,y2      x3,y3  
**line = mc.create\_line(200,0,0,200,200,200,200,0,fill = "orange")**

Figure 8.37

Code:

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()

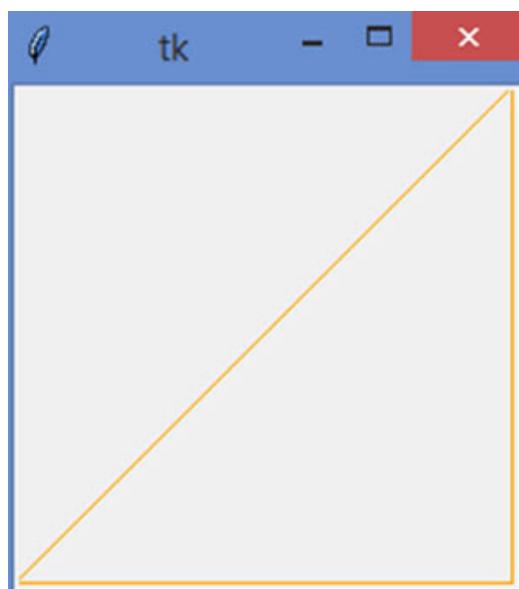
#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create lines to join point 2, 3 and 4 together
line = mc.create_line(200,0,0,200,200,200,200,0,fill = "orange")

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

Output:



*Figure 8.38*

Here's the code to draw a cross on the canvas, with the line width of 10.

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()

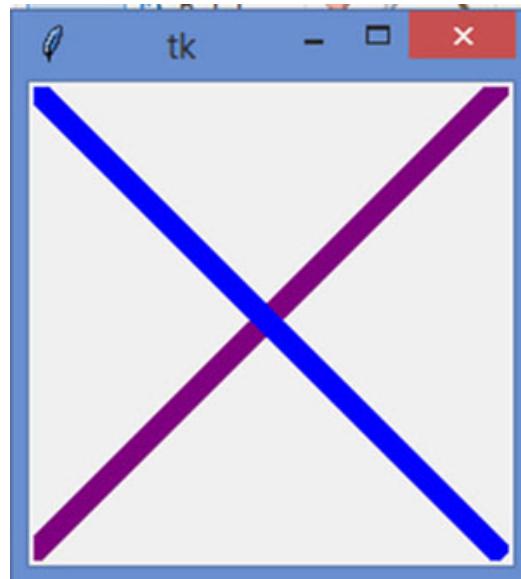
#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create line
line = mc.create_line(200,0,0,200,fill = "Purple", width = 10)
line2 = mc.create_line(0,0,200,200,fill = "Blue", width = 10)

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

## Output:



*Figure 8.39*

Let's now quickly have a look at other shapes that we can draw on the canvas.

## Rectangle

The following example shows how to define a rectangle.

### Example 8.9

Write the code to define a rectangle, which has been highlighted in the red box.

```
#Import tkinter
from tkinter import *

#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create rectangle
rect = mc.create_rectangle(5,10,100,50,fill = "Purple", width = 10)

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

### Output:

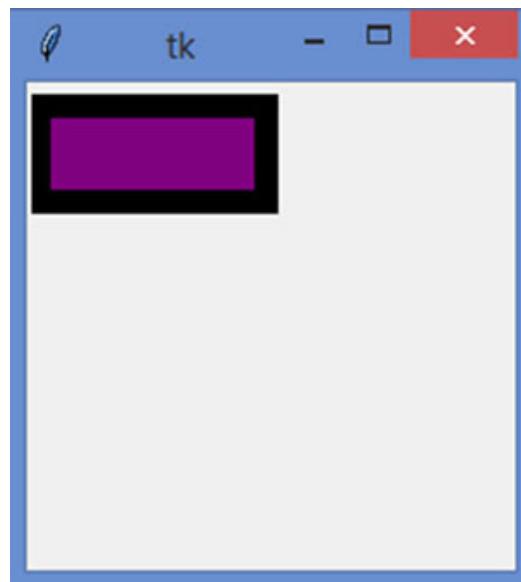
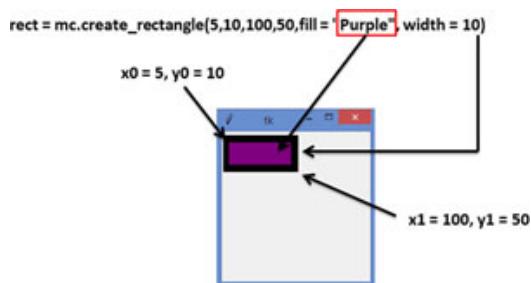


Figure 8.40

### Analysis



*Figure 8.41*

A rectangle is formed with top-left coordinate of 5, 10 and bottom-right coordinate of 100, 50.

We can change the color of the outline of the rectangle, and we can also change the color of the rectangle when it comes under the cursor.

**Code:**

```
#Import tkinter
from tkinter import *

#Create instance of window
mw = Tk()

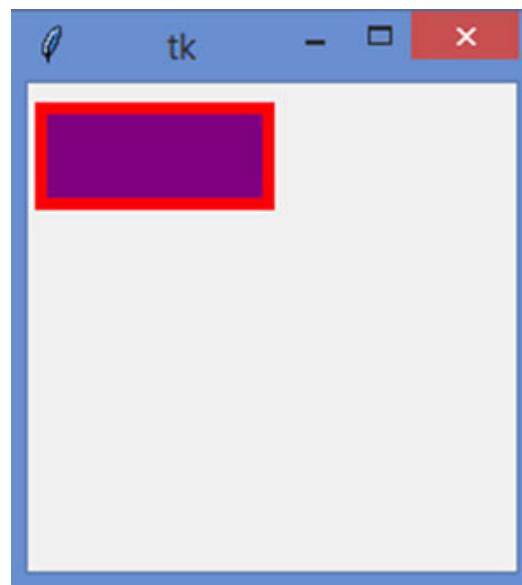
#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create rectangle
rect = mc.create_rectangle(5,10,100,50,fill = "Purple", width = 5,outline = "Red",activefill =
"yellow")

#Pack canvas object mc on to window object mw
mc.pack()

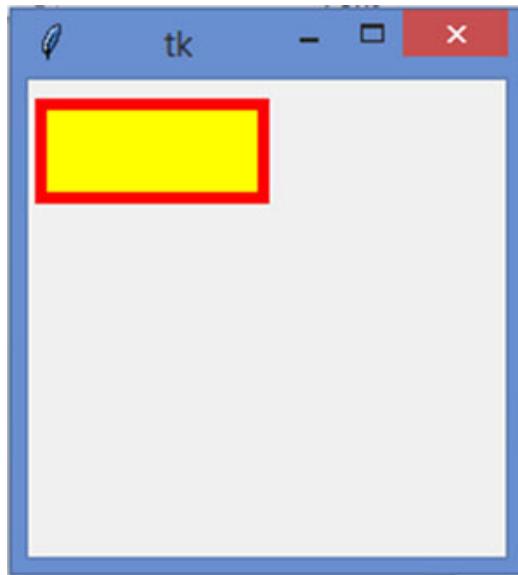
#Call mainloop()
mw.mainloop()
```

**Output:**



*Figure 8.42*

You will notice the change of color when the cursor is placed on the rectangle:



*Figure 8.43*

## Arc

You have just learnt how to work with rectangle on canvas. In this section, you will learn how to work with arc.

### Example 8.10

Write code to draw an arc.

#### Answer:

The arc is created in a rectangular space that is defined by given coordinates. The function used to create an arc is called `create_arc()`. Let's start by looking at a simple piece of code:

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()

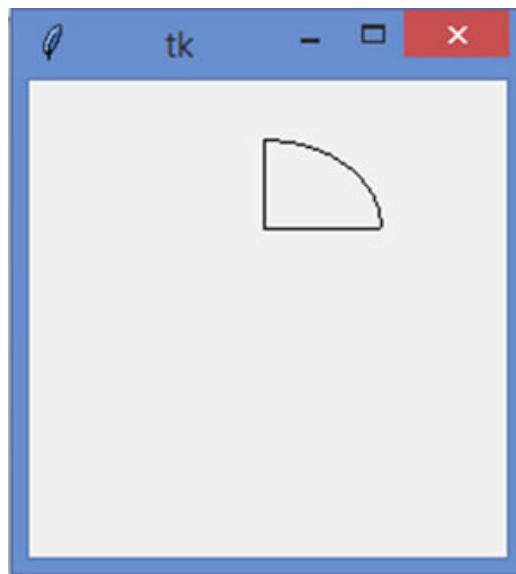
#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create an Arc
arc = mc.create_arc(50,25,150,100)

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

This would create an arc as shown in [figure 8.44](#).



*Figure 8.44*

In order to see how this arc is created, let's first draw the outline of the rectangle and then see how this arc is placed.

```
#Import tkinter
from tkinter import *

#Create instance of window
mw = Tk()

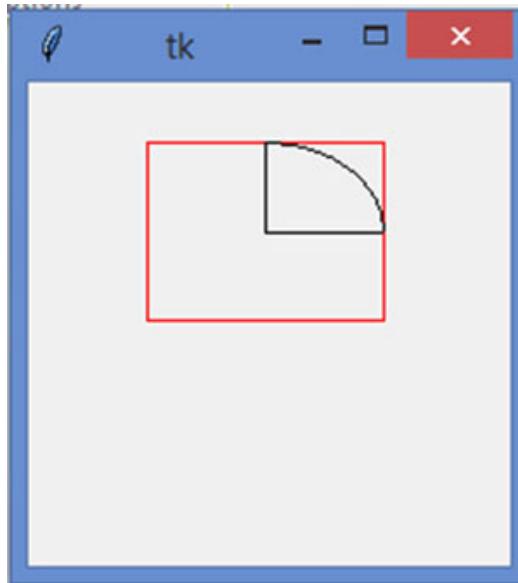
#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#Create outline of rectangle
rect = mc.create_rectangle(50,25,150,100,outline = "Red")

#Create an Arc
arc = mc.create_arc(50,25,150,100)

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```



*Figure 8.45*

The arc is created within a rectangle that is defined by the coordinates (50, 25) and (150, 100). Since no angle is defined, there is a default angle of 90°.

You can also define the start and end angles for the arc as follows:

```
#Import tkinter
from tkinter import *

#Create instance of window
mw = Tk()

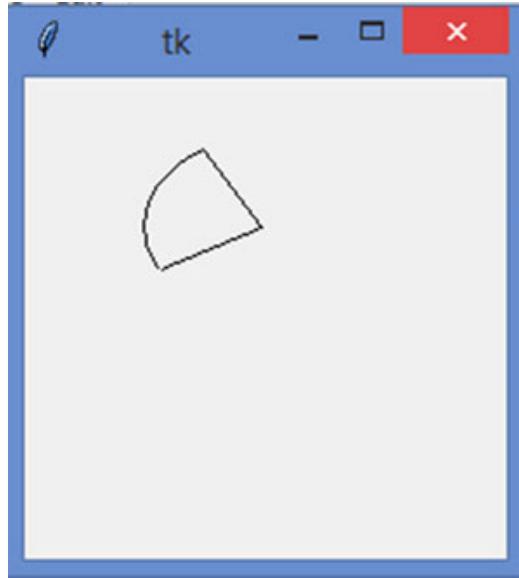
#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#Create an Arc
arc = mc.create_arc(50,25,150,100, start = 120, extent = 90)

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

The arc looks something like this:



*Figure 8.46*

To understand this, let's again have a look at how this arc is placed within a rectangle. The following code traces the rectangle in which the arc has been created.

### **Code:**

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#Create outline of rectangle
rect = mc.create_rectangle(50,25,150,100,outline = "Red")

#create an Arc
arc = mc.create_arc(50,25,150,100, start = 120, extent = 90)

#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

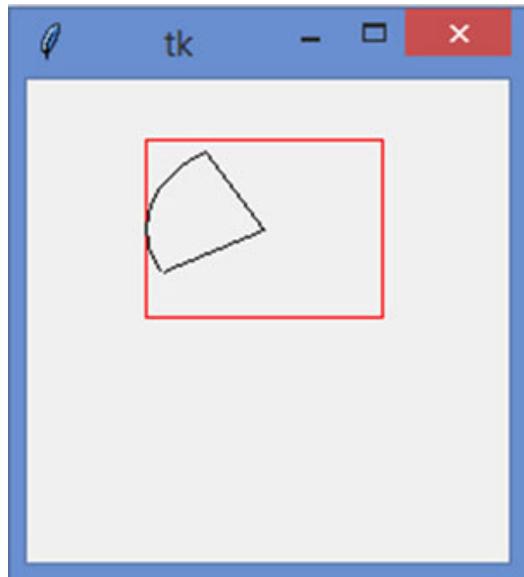


Figure 8.47

The following figure ([Figure 8.48](#)) illustrates how the angles work:

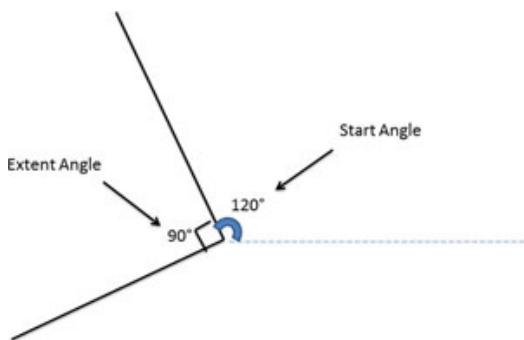


Figure 8.48

Similarly, to draw a complete oval, start can be 0 and extent can be 360, as shown in [Figure 8.49](#).

## Creating Oval and polygon on Canvas

```
#Import tkinter
from tkinter import *

#Create instance of window
mw = Tk()

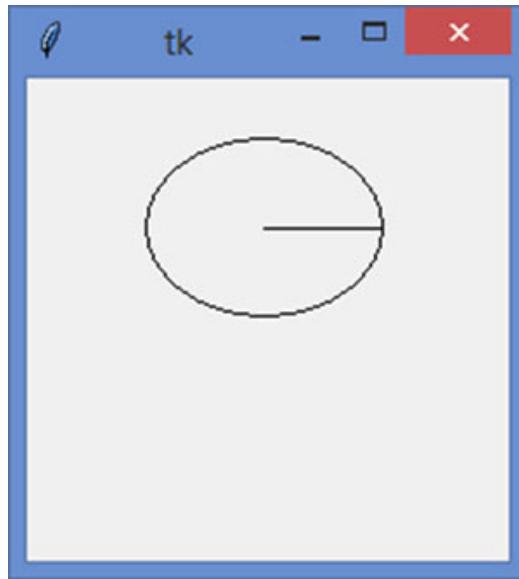
#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create an Arc
arc = mc.create_arc(50,25,150,100,start=0, extent = 359)

#Pack canvas object mc on to window object mw
```

```
mc.pack()

#Call mainloop()
mw.mainloop()
```



*Figure 8.49*

Like a line and rectangle, you can define several features for the arc:

```
#Import tkinter
from tkinter import*

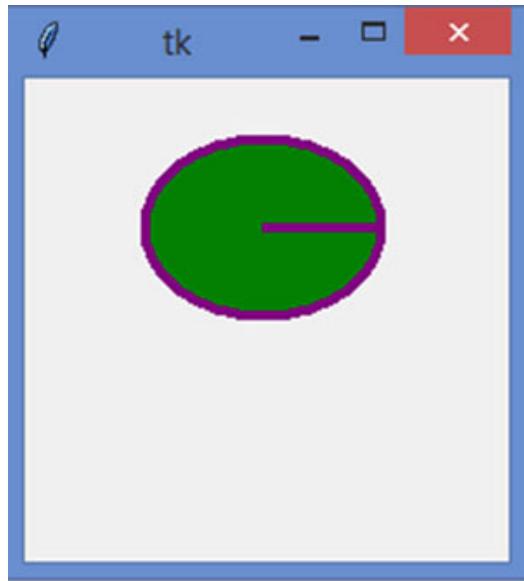
#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#create an Arc
arc = mc.create_arc(50,25,150,100,start=0, extent = 359,width = 4, fill = "green", outline =
"Purple", activefill = "White")

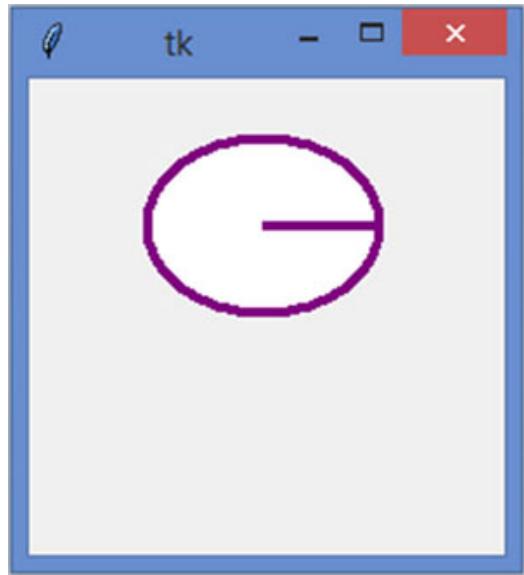
#Pack canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```



*Figure 8.50*

As you can see the outline has a width of 4, color purple, the arc is filled with color green and when the cursor is placed on top of the arc the color inside turns to white as shown in following figure ([Figure 8.51](#)):



*Figure 8.51*

### **Example 8.11**

Create a button on the window by the name **create Arc**, and on the click of that button, the user should be prompted to input the parameters for the arc and the arc should be displayed accordingly.

**Answer:**

```

#import Statements
from tkinter import*

Create a function for button
def arc_display():

#Create instance of Canvas class
mc = Canvas(mw, width = 350, height = 350)

#Take inputs for all values
x0 = int(input("please enter the value for x0 : "))
y0 = int(input("please enter the value for y0 : "))

x1 = int(input("please enter the value for x1 : "))
y1 = int(input("please enter the value for y1 : "))

s_angle = int(input("please enter the value for start angle : "))
e_angle = int(input("please enter the value for extent angle : "))

#Create an arc with the given value
mc.create_arc(x0,y0,x1,y1,start = s_angle,extent = e_angle)

#Make Canvas visible on the root window
mc.pack()

#Create instance of window
mw = Tk()

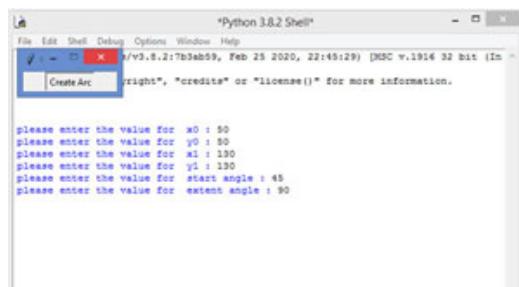
#Create instance of a button.
my_first_button = Button(mw, text="Create Arc", command = arc_display)

#For the button to display pack the button on to the window
my_first_button.pack()

#call the mainloop()
mw.mainloop()

```

## Output:



*Figure 8.52(a)*

This would give the following result:

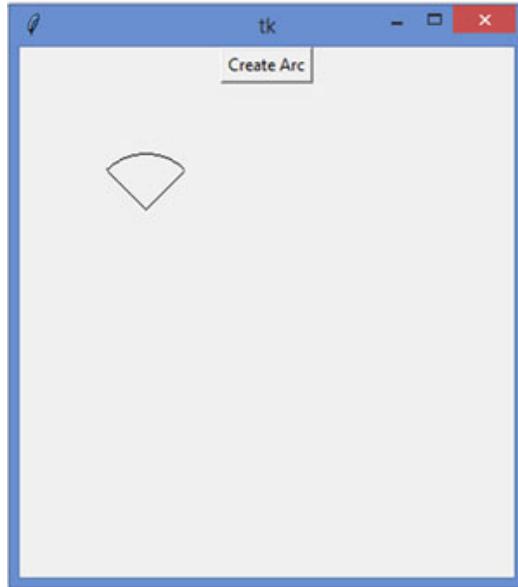


Figure 8.52(b)

### Example 8.12

Create an image of one fourth eaten pizza.

**Answer:**

```
#import Statements
from tkinter import*

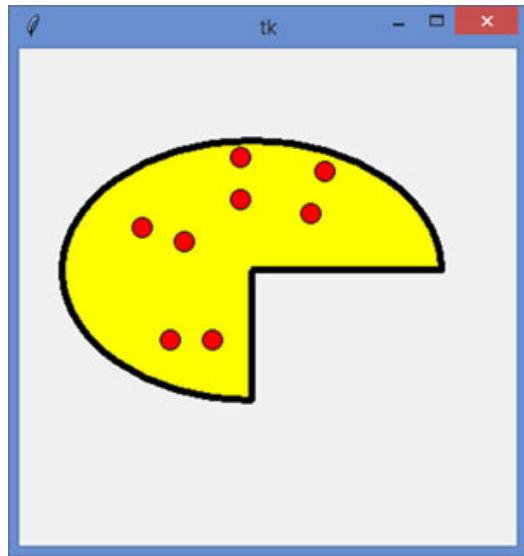
#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 350, height = 350)

#Draw Polygon
mc.create_arc(30,65,300,250,start = 0, extent = 270, fill="yellow",width = 5,outline="black")

#plot all points for inner dressing of pizza
mc.create_oval(80,120,94,134,fill="red",outline = "black")
mc.create_oval(210,80,224,94,fill="red",outline = "black")
mc.create_oval(110,130,124,144,fill="red",outline = "black")
mc.create_oval(200,110,214,124,fill="red",outline = "black")
mc.create_oval(150,70,164,84,fill="red",outline = "black")
mc.create_oval(150,100,164,114,fill="red",outline = "black")
mc.create_oval(130,200,144,214,fill="red",outline = "black")
mc.create_oval(100,200,114,214,fill="red",outline = "black")
#Make Canvas visible on the root window
mc.pack()

#call the mainloop()
mw.mainloop()
```



*Figure 8.53*

### **Example 8.13**

Write a program to create an oval.

**Answer:** The code for creating an oval is as follows:

#### **Code:**

```
#import Statements
from tkinter import*

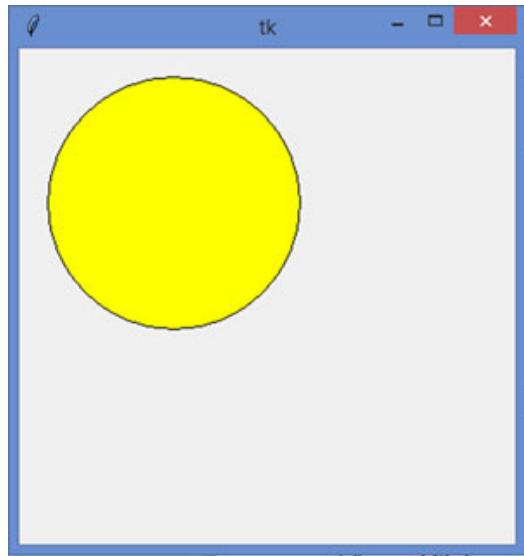
#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 350, height = 350)

#Draw an oval
mc.create_oval(20,20,200,200,fill = "yellow")

#Make Canvas visible on the root window
mc.pack()

#call the mainloop()
mw.mainloop()
```



*Figure 8.54*

### **Example 8.14**

Write a program for drawing a Polygon.

**Answer:** The following code is an example for drawing a polygon:

```
#import Statements
from tkinter import*

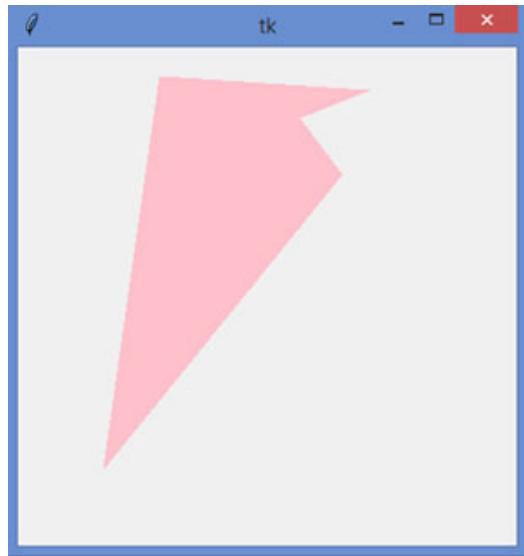
#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 350, height = 350)

#Draw Polygon
mc.create_polygon(250,30,200,50,230,90,60,300,100,20, fill = "pink")

#Make Canvas visible on the root window
mc.pack()

#call the mainloop()
mw.mainloop()
```



*Figure 8.55*

**Explanation:**

You may be wondering how this polygon has been created. Well, you can plot the points by drawing small circles at the point.

In this figure, we have created a polygon with following sets of point:

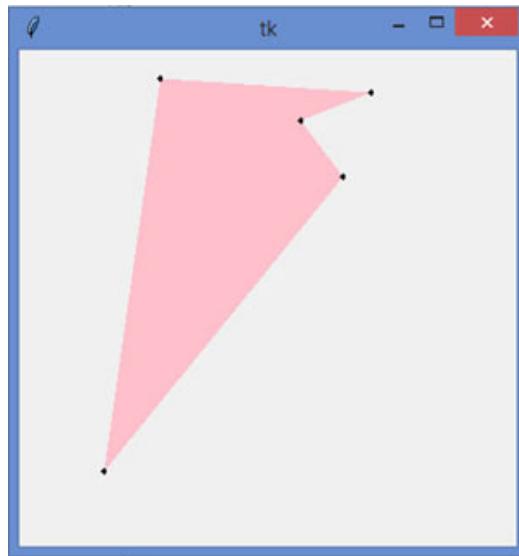
- 250,30
- 200,50
- 230,90
- 60,300
- 100,20

Around each point, we can create a small circle using the following table:

| Point  | X0(x-1) | Y0(y-1) | X1(x+1) | Y1(y+1) |
|--------|---------|---------|---------|---------|
| 250,30 | 249     | 29      | 251     | 31      |
| 200,50 | 199     | 49      | 201     | 51      |
| 230,90 | 229     | 89      | 231     | 91      |
| 60,300 | 59      | 299     | 61      | 301     |
| 100,20 | 99      | 19      | 101     | 21      |

*Table 8.3*

**Output:**



*Figure 8.56*

At the end of this topic on canvas, let's have a look at some of the options available for canvas. The basic code for creating a canvas is as follows:

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()

#Create instance of Canvas class
mc = Canvas(mw, width = 200, height = 200)

#Add canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```

Following [Table 8.4](#) describes the options available for the Canvas widget:

| Option                          | Description                                                                                   |
|---------------------------------|-----------------------------------------------------------------------------------------------|
| <b>bd(Default is 2)</b>         | Border width in pixels.                                                                       |
| <b>bg</b>                       | Normal background color.                                                                      |
| <b>confine(Default is True)</b> | The canvas cannot be scrolled outside of the scroll region.                                   |
| <b>Cursor</b>                   | You can define what type (arrow, circle, dot, and so on) of cursor you want to use in canvas. |
| <b>Height</b>                   | Size of the canvas in the Y dimension.                                                        |
| <b>Highlightcolor</b>           | Color when highlight when in focus.                                                           |
| <b>relief(SUNKEN, RAISED,</b>   | Type of the border.                                                                           |

|                           |                                                                                                                  |
|---------------------------|------------------------------------------------------------------------------------------------------------------|
| <b>GROOVE, and RIDGE)</b> |                                                                                                                  |
| <b>Scrollregion</b>       | How large an area the canvas can be scrolled defined by A tuple defining (left, top, right and bottom) position. |
| <b>Width</b>              | Size of the canvas in the X dimension.                                                                           |
| <b>xscrollcommand</b>     | This attribute should be the <code>.set()</code> method of the horizontal scrollbar.                             |
| <b>Yscrollcommand</b>     | This attribute should be the <code>.set()</code> method of the vertical scrollbar.                               |

*Table 8.4*

## 8.4.1 Writing text on Canvas

The following code shows how to write text on canvas.

**Code:**

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()

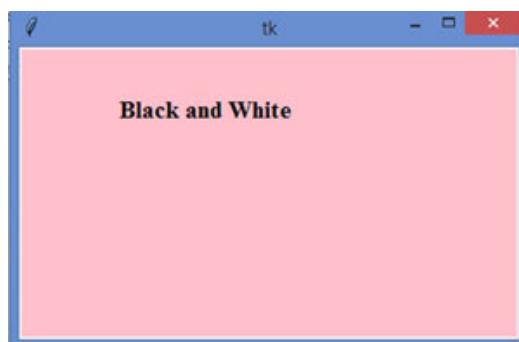
#Create instance of Canvas class
mc = Canvas(mw,bg="pink",width = 400, height = 400)

#define Font for text on canvas
fnt = ('Times',15,"bold")

#define the text
txt = mc.create_text(150,50, text = "Black and White", font = fnt,fill ="black",activefill =
"white")

#Add canvas object mc on to window object mw
mc.pack()

#Call mainloop()
mw.mainloop()
```



*Figure 8.57*

The text changes color to white when you place the cursor on top.



*Figure 8.58*

## 8.5 Frame

You just learnt about canvas which is used for drawing images. Now, let's work on Frame, which is a container used to display widgets such as buttons, check buttons, and so on. So, basically a frame helps in organizing the GUI.

### Working with Frames

Look at the following code:

```
#import statements
from tkinter import*
mw = Tk()

#create a frame object
mf = Frame(mw)

#attach frame to root window
mf.pack()

#First button
button1 = Button(mf, text = "Left",bg = "red", bd = 10)

#Attach button1 to the frame on the left side
button1.pack(side = "left")

#Second Button
button2 = Button(mf, text = "Right",bg = "Green",bd = 10)

#Attach button2 to the frame on the right side
button2.pack(side = "right")

#Main Loop
mw.mainloop()
```



*Figure 8.59*

## Options for Frame

```
#import statements
from tkinter import*

mw = Tk()
mw.geometry("200x200")

#Create a frame Object
mf = Frame(mw,width =150, height = 150,bg= "green",bd = 20, relief = "sunken")

#attach frame to root window
mf.pack()

#Main Loop
mw.mainloop()
```

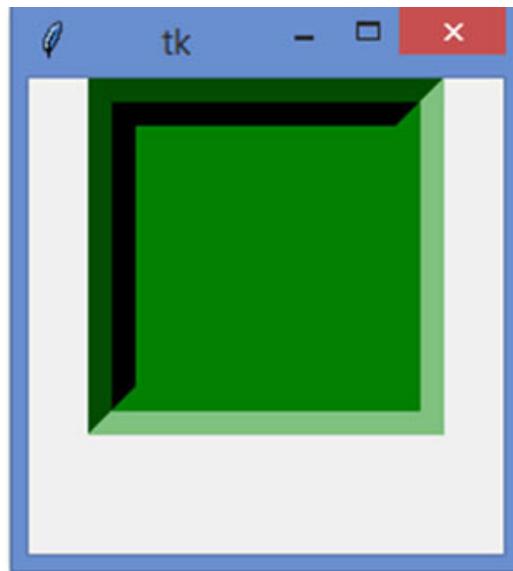


Figure 8.60

## 8.6 Working with Labels

Just like canvas, a **label** is also used to display text or images. The image or text on the label can be changed any time. It is easier to use than the canvas widget.

Just like other widgets, label too takes the master window and options separated by comma as parameters to create an instance.

```
Label(master, option1,option2....)
```

Look at the following code:

```
#Import Statements
from tkinter import*

#Create a root window
```

```

mw = Tk()

#Define dimensions of root window
mw.geometry("200x200")

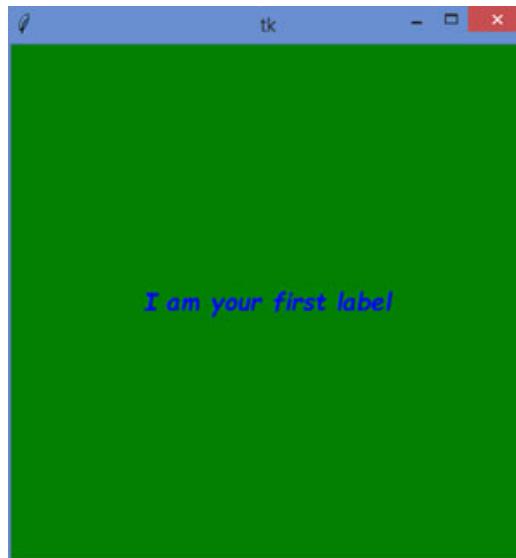
#Create Label and define options such as font, width, height, bg,fg etc
lbl = Label(mw,text="I am your first label",fg = "blue",bg = "green", font=("Comic Sans MS",14,"bold italic"),width = 200, height = 200)

#Associate the label to the root window
lbl.pack()

#Call the mainloop()
mw.mainloop()

```

### Output:



*Figure 8.61*

## 8.7 Mini Project - Stop Watch

The following code displays two buttons. When you click on the start button a timer starts, and when you click on stop, the time elapsed is displayed. Feel free to make changes in the code, and refine it more with the knowledge that you have obtained so far.

```

#import statements
from tkinter import*
import time

start_time = 0
end_time = 0
total_time = 0

def time_display(seconds):
 #get the floor value of minutes by dividing value of seconds by 60

```

```

minutes = seconds//60
#get the floor value of hours by dividing value of minutes by 60
hours = minutes//60
minutes = minutes%60
seconds = seconds%60
msg = "Time Lapsed = {0}:{1}:{2}".format(int(hours),int(minutes), int(seconds))
lbl = Label(mw,text = msg,fg = "blue",bg = "white",font=("Comic Sans MS",14,"bold italic"),width = 100, height = 100)
lbl.pack(side = "top")

def timer_start():
 global start_time
 print("in start time")
 start_time = time.time()
 print(start_time)

def timer_end():
 end_time = time.time()
 total_time = end_time - start_time
 time_display(int(total_time))

#Create root window
mw = Tk()
mw.geometry("300x300")
#Create a frame Object
mf = Frame(mw)

#attach frame to root window
mf.pack(side = "bottom")

#First button
button1 = Button(mf, text = "Start",bg = "green", bd = 10,command = timer_start)

#Attach button1 to the frame on the left side
button1.pack(side = "left")

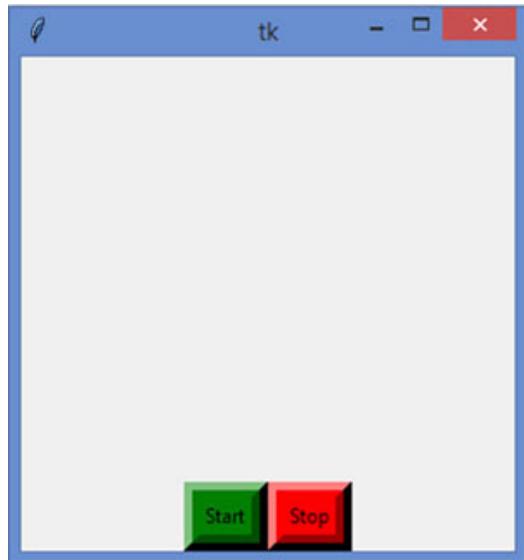
#Second Button
button2 = Button(mf, text = "Stop",bg = "red",bd = 10,command = timer_end)

#Attach button2 to the frame on the right side
button2.pack(side = "right")

#Main Loop
mw.mainloop()

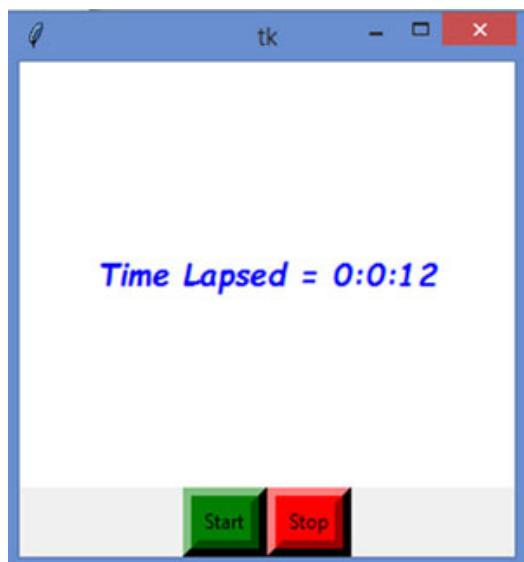
```

## Output:



*Figure 8.62*

Click on start and after sometime click on the stop button.



*Figure 8.63*

## 8.8 List Box Widget

A Listbox widget is used for displaying a list of items. The syntax for creating a list box is:

```
Listbox(master,options.....)
```

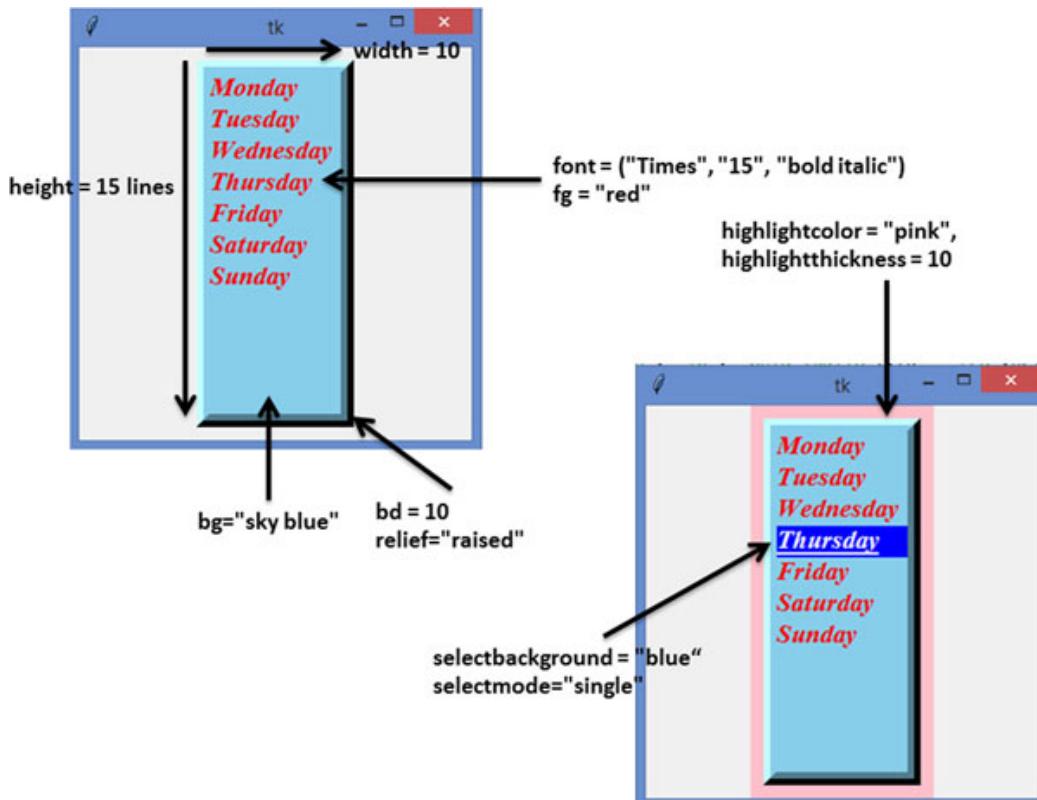
**Code:**

```

from tkinter import*
mw = Tk()
mw.geometry("300x300")
lstbx = Listbox(mw,bg="sky blue",bd = 10,relief="raised", font = ("Times", "15", "bold italic"),fg = "red",height = 15,width = 10, highlightcolor = "pink", highlightthickness = 10,selectbackground = "blue",selectmode="single")
lstbx.insert(1,"Monday")
lstbx.insert(2,"Tuesday")
lstbx.insert(3,"Wednesday")
lstbx.insert(4,"Thursday")
lstbx.insert(5,"Friday")
lstbx.insert(6,"Saturday")
lstbx.insert(7,"Sunday")
lstbx.pack()
mw.mainloop()

```

## Output:



*Figure 8.64: Options for List Box*

Some of the important methods of a listbox are given as follows:

| Method                        | Description                                                                                                                                                   |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>activate (index)</code> | Based on the value passed as argument, it selects the line specified.                                                                                         |
| <code>curselection()</code>   | This method returns line numbers of the selected element or elements packed in a tuple, (counting from 0). An empty tuple is returned if nothing is selected. |

|                                        |                                                                                                                                                              |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>delete (first, last=None)</code> | Deletes the range of lines passed as argument. In case no value is passed as second argument, a single line with the value of first argument is deleted.     |
| <code>get(first, last=None)</code>     | Retrieves the text of the lines within the range mentioned in the argument. If there is only one argument is given, then only text of that line is returned. |
| <code>index (i)</code>                 | Used to position the visible part of the listbox such that that the line containing index is at the top.                                                     |
| <code>insert (index, *elements)</code> | This method inserts one or more lines into the listbox before the line specified by index.                                                                   |
| <code>nearest (y)</code>               | This method will return the index of line that is nearest to the y-coordinate with respect to the listbox widget.                                            |
| <code>see (index)</code>               | This method adjusts the position of the list box in such a manner that the line mentioned in the method is visible.                                          |
| <code>size()</code>                    | This method returns the size or the number of lines in the listbox.                                                                                          |
| <code>xview()</code>                   | Makes the listbox horizontally scrollable.                                                                                                                   |
| <code>yview()</code>                   | Makes the listbox vertically scrollable, sets the command option of the associated vertical scrollbar to this method.                                        |

*Table 8.5*

## 8.9 The Menu button and Menu

In this section, we will explore how to work with Menu buttons.

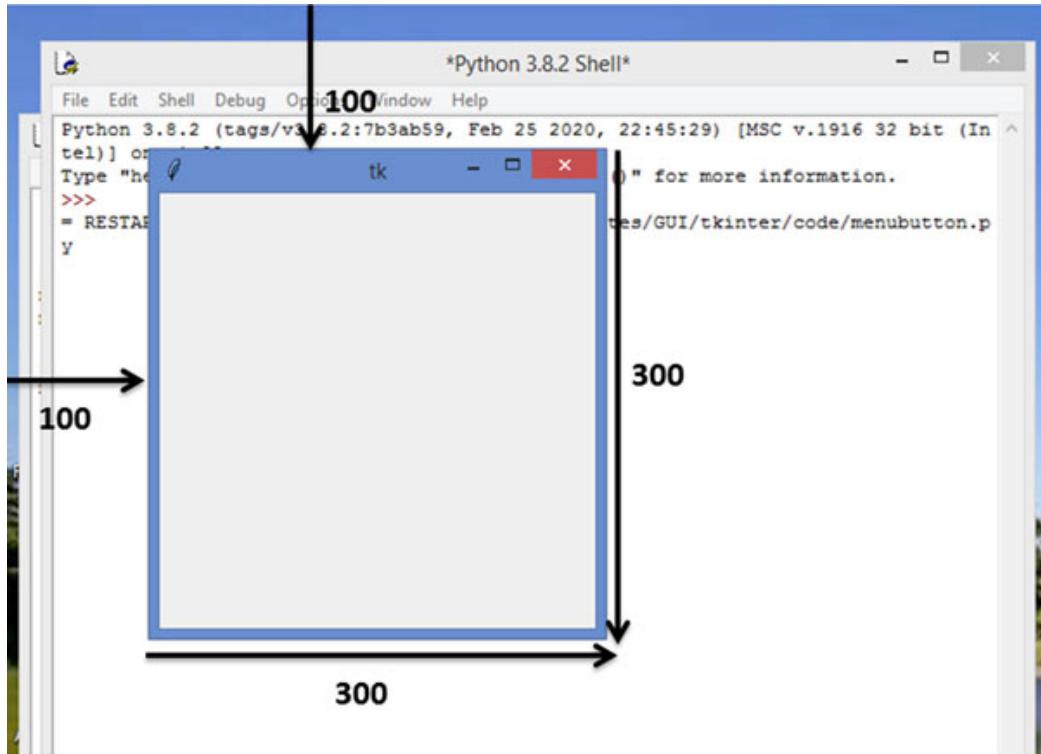
### **Step 1: Create the basics for the root window**

```
#Import statements
from tkinter import*

#Create root window(+120 +120 is the distance from xy)
mw = Tk()
mw.geometry("300x300+100+100") #explained below

#Call Main Loop Method
mw.mainloop()
```

This would create a window as shown here:



*Figure 8.65*

If you look at the code, you would find the following statement a bit strange:

```
mw.geometry("300x300+100+100")
```

In the preceding image, 300x300 is the dimension of the window, where +100+100 indicates the position on the screen where the window will be displayed. So, it is the window from the x and y of the screen.

### Step 2: Create a Menu Button object.

The code for creating a **Menubutton** is given as follows:

```
#Create Menu Button
mb = Menubutton(mw, text = "File")
mb.grid()
```

Just in case of other widgets, the first parameter to be passed while creating a Menubutton is the root window, which in our case is `mw`. The second parameter is the text for the menu button.

Place this code just before we call `mainloop()`. So, the code so far should look like this:

```
#Import statements
from tkinter import*

#Create root window
mw = Tk()
```

```

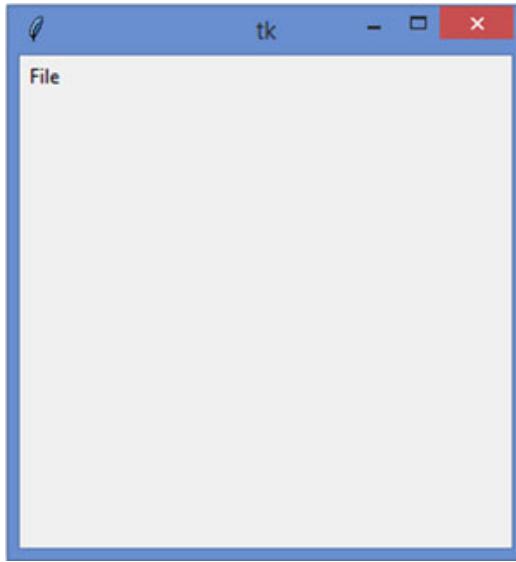
mw.geometry("300x300+100+100")

#Create Menu Button
mb = Menubutton(mw, text = "File")
mb.grid()

#Call Main Loop Method
mw.mainloop()

```

The output of this would be as follows:



*Figure 8.66*

### Step 3: Create a pull down Menu for the Menu button.

We will now add the following lines to our code. The first parameter is the menubutton and the second parameter that we have given is tearoff value =0. Tkinter menus support tearoff by default which means you can create floating menus by detaching it from the main menu. Hence, to disable this feature, we set it to 0. For those who have not understood this feature, just go with this setting and then read the explanation provided at end of the example.

```

#create a pull down menu
mb.menu = Menu(mb, tearoff=0)
mb["menu"] = mb.menu

```

So, now the code developed so far should look like this:

```

#import statements
from tkinter import *

#create root window(+120 +120 is the distance from xy)
mw = Tk()
mw.geometry("300x300+100+100")

```

```

#Create Menu Button
mb = Menubutton(mw, text = "File")
mb.grid()

#create a pull down menu
mb.menu = Menu(mb, tearoff=0)
mb["menu"] = mb.menu

#Call Main Loop Method
mw.mainloop()

```

## Step 4: Add commands to pull down menu

```

Add commands to pull down menu
mb.menu.add_command(label = "New File")
mb.menu.add_command(label = "Open")
mb.menu.add_command(label = "Recent Files")
mb.menu.add_command(label = "Save")
mb.menu.add_command(label = "Save As")
mb.menu.add_command(label = "Print")
mb.menu.add_command(label = "Close")
mb.menu.add_command(label = "Exit")

```

With this our code for menu button will be as follows:

```

#import statements
from tkinter import *

#Create root window(+120 +120 is the distance from xy)
mw = Tk()
mw.geometry("300x300+100+100")

#Create Menu Button
mb = Menubutton(mw, text = "File")
mb.grid()

#create a pull down menu
mb.menu = Menu(mb, tearoff=0)
mb["menu"] = mb.menu

Add commands to pull down menu
mb.menu.add_command(label = "New File")
mb.menu.add_command(label = "Open")
mb.menu.add_command(label = "Recent Files")
mb.menu.add_command(label = "Save")
mb.menu.add_command(label = "Save As")
mb.menu.add_command(label = "Print")
mb.menu.add_command(label = "Close")
mb.menu.add_command(label = "Exit")

#Call Main Loop Method
mw.mainloop()

```

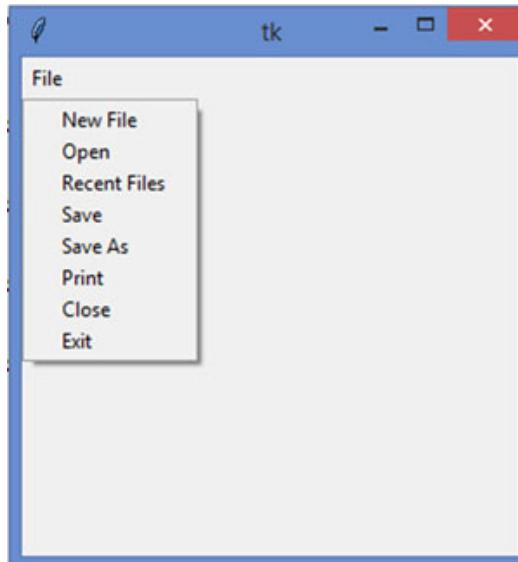
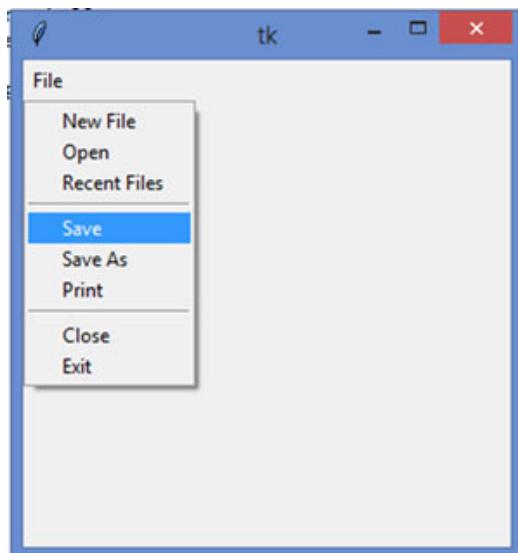


Figure 8.67

You can also add separators between two commands, so the code will now look as follows:

```
mb.menu.add_command(label = "New File")
mb.menu.add_command(label = "Open")
mb.menu.add_command(label = "Recent Files")
mb.menu.add_separator()
mb.menu.add_command(label = "Save")
mb.menu.add_command(label = "Save As")
mb.menu.add_command(label = "Print")
mb.menu.add_separator()
mb.menu.add_command(label = "Close")
mb.menu.add_command(label = "Exit")
```

The menu would be as follows:



*Figure 8.68*

Now, bring your attention to the **tearoff** feature mentioned in step 3.

*“Tkinter menus support tearoff by default, which means you can create floating menus by detaching it from the main menu. Hence, to disable this feature, we set it to 0.”*

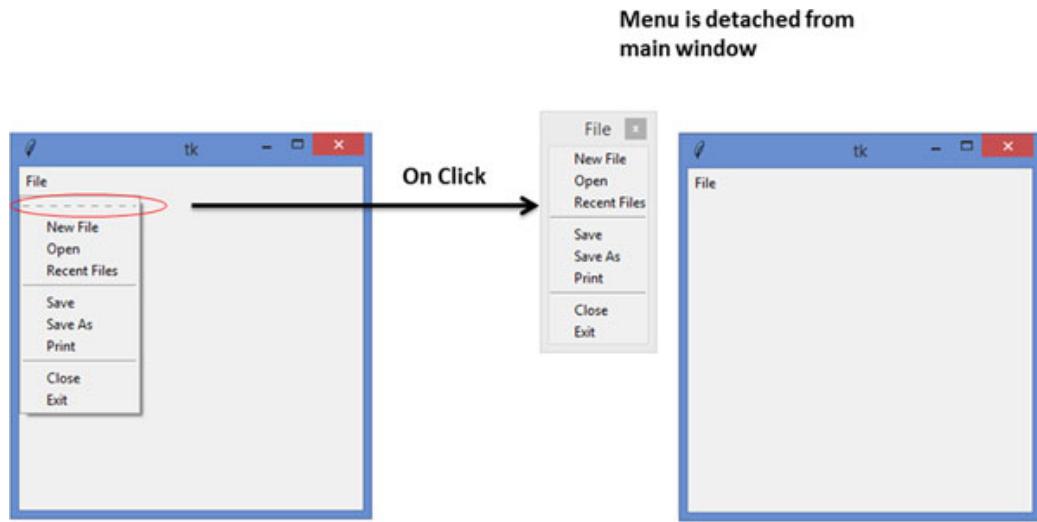
Go to the code written for creating a pull-down menu.

```
#create a pull down menu
mb.menu = Menu(mb, tearoff=0)
mb["menu"] = mb.menu
```

Now, remove the tearoff setting.

```
#create a pull down menu
mb.menu = Menu(mb)
mb["menu"] = mb.menu
```

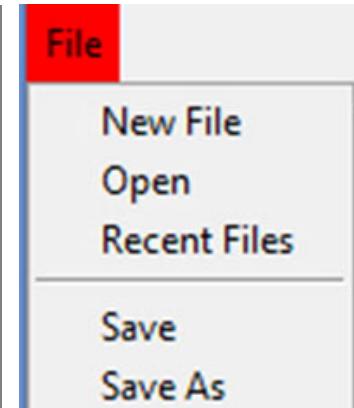
Now, execute the code. You would notice a dotted line on top of the menu options. If you click on it, the menu will detach itself from the main menu as shown in the following figure ([Figure 8.69](#)):



*Figure 8.69*

[Table 8.6](#) displays some important options for **Menubutton**

| Option           | Description                                              | Code                                                    | Output |
|------------------|----------------------------------------------------------|---------------------------------------------------------|--------|
| activebackground | Background color when the mouse is placed over the menu. | Menubutton(mw, text = "File", activebackground = "Red") |        |



**Figure 8.70:** `aactivebackground = "Red"`

|                  |                                                          |                                                                    |  |
|------------------|----------------------------------------------------------|--------------------------------------------------------------------|--|
|                  |                                                          |                                                                    |  |
| activeforeground | Foreground color when the mouse is placed over the menu. | <pre>Menubutton(mw, text = "File", activeforeground = "Red")</pre> |  |

**Figure 8.71:** `activeforeground = "Red"`

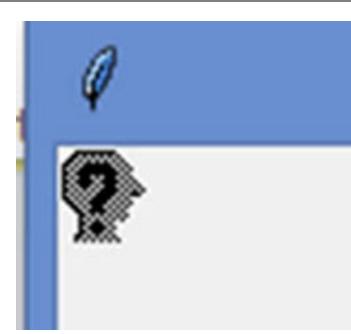
|        |                                                                                                     |                                                                                |  |
|--------|-----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|--|
|        |                                                                                                     | <pre>Menubutton(mw, text = "File", height = 5, width = 5, anchor = "se")</pre> |  |
| anchor | Position of text: options are - n, ne, e, se, s, sw, w, nw, or center. Center is the default value. |                                                                                |  |

**Figure 8.72:** `anchor = "se"`

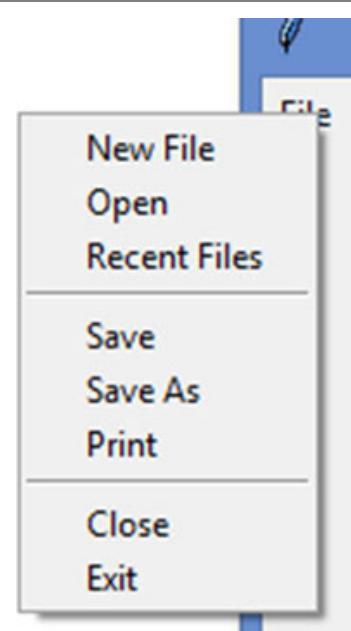
|    |                                              |                                                      |  |
|----|----------------------------------------------|------------------------------------------------------|--|
| bg | Background color displayed behind the label. | <pre>Menubutton(mw, text = "File", bg = "red")</pre> |  |
|----|----------------------------------------------|------------------------------------------------------|--|



**Figure 8.73:** *bg = "red"*

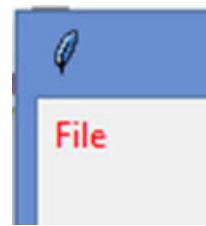
|        |                                                                                                                                                                                 |                                                                  |                                                                                     |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| bitmap | Standard bitmaps available are : 'error', 'gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info', 'questhead', 'question', and 'warning'. You can add your own bitmap too. | <pre>Menubutton(mw,text<br/>"File",bitmap<br/>"questhead")</pre> |  |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|-------------------------------------------------------------------------------------|

**Figure 8.74:** *bitmap = "questhead"*

|           |                                                                                    |                                                                 |                                                                                      |
|-----------|------------------------------------------------------------------------------------|-----------------------------------------------------------------|--------------------------------------------------------------------------------------|
| direction | Direction for displaying the menu. It can have the value of left, right, or above. | <pre>mb = Menubutton(mw,text<br/>"File",direction="left")</pre> |  |
|-----------|------------------------------------------------------------------------------------|-----------------------------------------------------------------|--------------------------------------------------------------------------------------|

**Figure 8.75:** *direction="left"*

|    |                  |                                                    |  |
|----|------------------|----------------------------------------------------|--|
| fg | Foreground color | <pre>Menubutton(mw,text<br/>"File",fg="red")</pre> |  |
|----|------------------|----------------------------------------------------|--|



*Figure 8.76: fg= "red"*

|  |  |                                                                                                                    |  |
|--|--|--------------------------------------------------------------------------------------------------------------------|--|
|  |  | <pre>img = PhotoImage(file = "file:\\my_images\\thumb2.gif") mb = Menubutton(mw, text = "File", image = img)</pre> |  |
|--|--|--------------------------------------------------------------------------------------------------------------------|--|

*Figure 8.77: image*

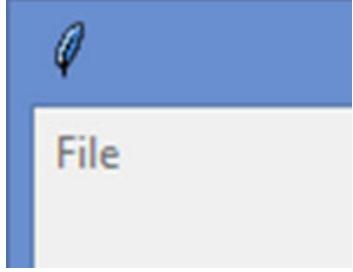
|             |                                                                                                                   |                                                               |  |
|-------------|-------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|--|
| padx & pady | Space to be left to the left and right of the text is padx and space to be left above and below the text is pady. | <pre>Menubutton(mw, text = "File", padx = 10, pady= 10)</pre> |  |
|-------------|-------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|--|

*Figure 8.78: padx = 10, pady= 10*

|        |                |                                                             |  |
|--------|----------------|-------------------------------------------------------------|--|
| Relief | Border shading | <pre>Menubutton(mw, text = "File", relief = "raised")</pre> |  |
|--------|----------------|-------------------------------------------------------------|--|

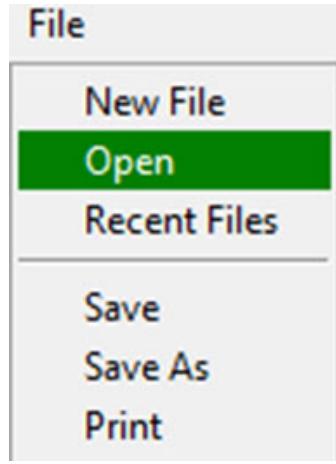
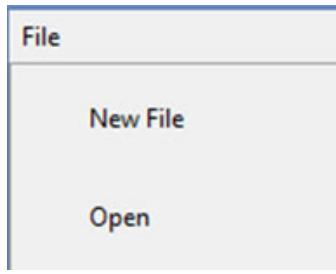
*Figure 8.79: relief = "raised"*

|       |             |                                  |  |
|-------|-------------|----------------------------------|--|
| State | When set to | <pre>Menubutton(mw, text =</pre> |  |
|-------|-------------|----------------------------------|--|

|                                                   |                                       |                                        |                                                                                     |
|---------------------------------------------------|---------------------------------------|----------------------------------------|-------------------------------------------------------------------------------------|
|                                                   | disabled, it will disable the button. | <code>"File", state="disabled")</code> |  |
| <b>Figure 8.80:</b> <code>state="disabled"</code> |                                       |                                        |                                                                                     |

*Table 8.6*

*Table 8.7* displays some important options for Menu.

| Option            | Description                                                         | Code                                                                    | Output                                                                                |
|-------------------|---------------------------------------------------------------------|-------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| activebackground  | Color that appears when under the mouse.                            | <code>Menu(mb, tearoff =0, activebackground = "green")</code>           |   |
| activeborderwidth | Width of a border drawn around a choice when it is under the mouse. | <code>Menu(mb, tearoff =0, activeborderwidth = 15)</code>               |  |
| activeforeground  | Color of text when it comes under mouse.                            | <code>mb.menu<br/>Menu(mb, tearoff =0, activeforeground = "red")</code> |                                                                                       |

*Figure 8.81:*  
`activebackground = "green"`

*File*

New File

Open

Recent Files

Save

Save As

Print

*Figure 8.82:*  
`activeborderwidth = 15`

*File*

New File

Open

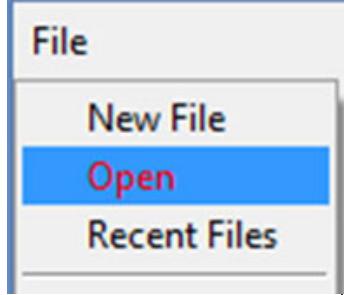
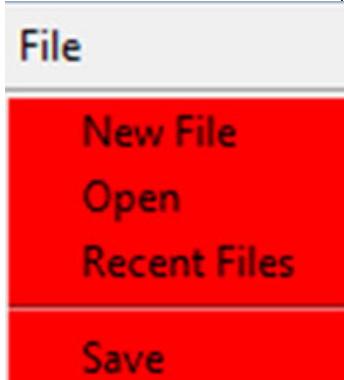
|                    |                                                                  |                                                      |                                                                                     |
|--------------------|------------------------------------------------------------------|------------------------------------------------------|-------------------------------------------------------------------------------------|
|                    |                                                                  |                                                      |  |
| bg                 | Background color                                                 | <pre>mb.menu = Menu(mb, tearoff =0,bg = "red")</pre> |  |
| disabledforeground | Color of text of disabled items.                                 |                                                      |                                                                                     |
| Postcommand        | Procedure will be called every time someone brings up this menu. |                                                      |                                                                                     |

Table 8.7

## 8.10 Radiobutton

The **Radio button** widget is an implementation of a multiple-choice button. A radio button is used to present multiple objects to user, and lets user choose only one of them.

**Syntax:**

```
Radiobutton(root, options)
```

Here is how you can create a Radio button:

**Code:**

```
#Import statements
from tkinter import*
#Create root window
```

```

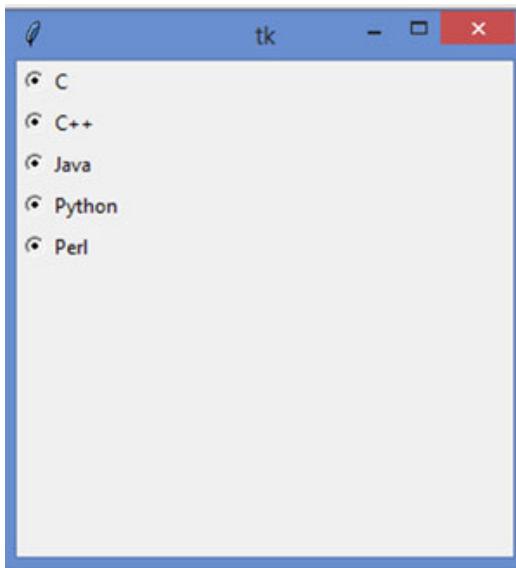
mw = Tk()
mw.geometry("300x300")

#Create Radio Button
rb1 = Radiobutton(mw,text = "C")
rb2 = Radiobutton(mw,text = "C++")
rb3 = Radiobutton(mw,text = "Java")
rb4 = Radiobutton(mw,text = "Python")
rb5 = Radiobutton(mw,text = "Perl")

rb1.grid(row = 0,column = 0,sticky=W)
rb2.grid(row = 1,column = 0,sticky=W)
rb3.grid(row = 2,column = 0,sticky=W)
rb4.grid(row = 3,column = 0,sticky=W)
rb5.grid(row = 4,column = 0,sticky=W)

#Call Main Loop Method
mw.mainloop()

```



*Figure 8.85*

Or

```

#Import statements
from tkinter import*

#Create root window
mw = Tk()
mw.geometry("300x300")

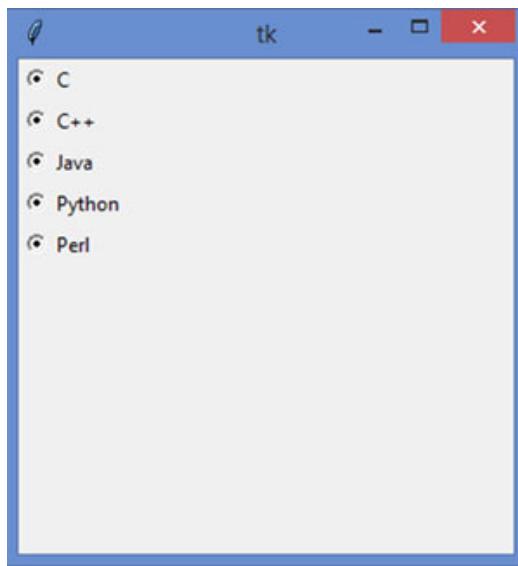
#Create Radio Button
rb1 = Radiobutton(mw,text = "C")
rb2 = Radiobutton(mw,text = "C++")
rb3 = Radiobutton(mw,text = "Java")
rb4 = Radiobutton(mw,text = "Python")
rb5 = Radiobutton(mw,text = "Perl")

rb1.pack(anchor = W)

```

```
rb2.pack(anchor = W)
rb3.pack(anchor = W)
rb4.pack(anchor = W)
rb5.pack(anchor = W)

#Call Main Loop Method
mw.mainloop()
```



*Figure 8.86*

## 8.11 Scrollbar and Sliders

A scrollbar widget can be used to scroll text in another widget. To scroll from left to right, we require horizontal scroll bar, and to scroll from top to bottom, we require vertical scroll bar.

Here are step-by-step instructions:

### **Step 1: Create the main structure**

```
import tkinter as tk
mw = tk.Tk()
mw.geometry("100x100")
mw.mainloop()
```

This code creates a window like the one shown as follows, which is ready to be populated:

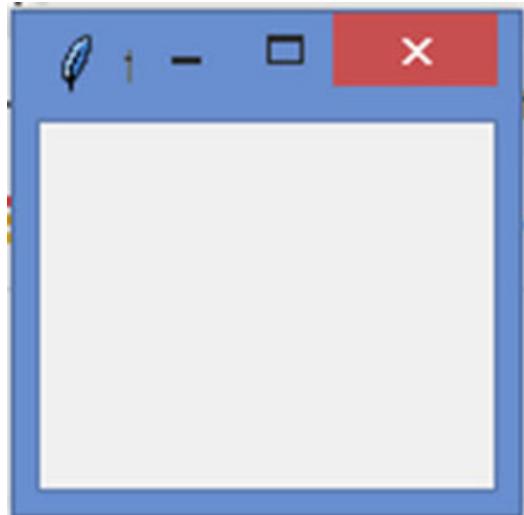


Figure 8.87

**Step 2: Now, create a scroll bar and link it to the main window**

```
#Vertical Scrollbar
sb = tk.Scrollbar(mw)
sb.pack(side = tk.RIGHT, fill="y")
```

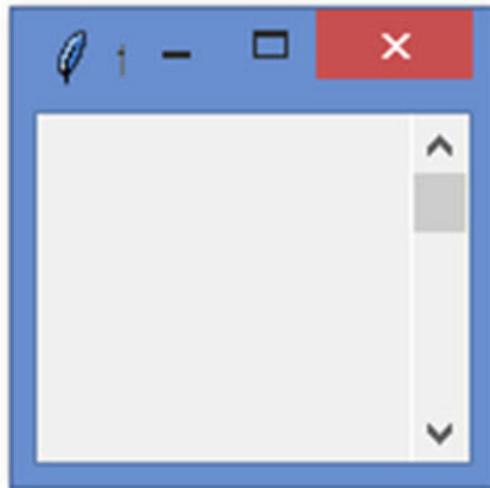


Figure 8.88

**Step 3: Create a text box and pack it to the main window**

You will learn more about textbox in the next section.

```
#Create a Textbox
tb = tk.Text(mw, height = 500, width = 500, yscrollcommand = sb.set, bg="sky blue")
tb.pack()
```

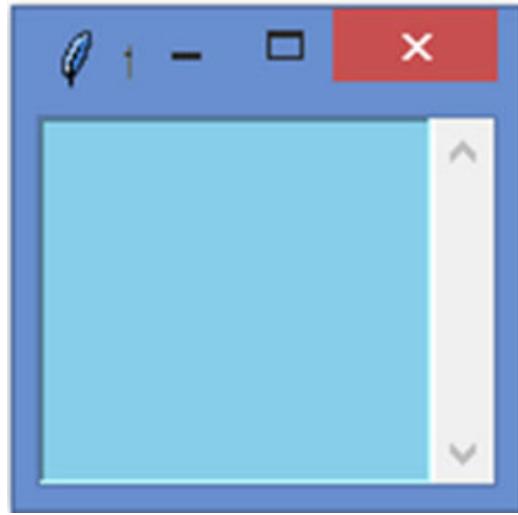


Figure 8.89

#### Step 4: Configure the scrollbar to the textbox using config() method

The `config()` method is used to access the object's attribute after its initialization so that it can be set during runtime.

```
Configure scroll bar to the textbox
sb.config(command=tb.yview)
```

#### Step 5: Create content for textbox

```
list_of_month =
"January\nFebruary\nMarch\nApril\nMay\nJune\nJuly\nAugust\nSeptember\nOctober\nNovember\nDecember"
```

Insert content in the textbox.

```
#content for textbox
list_of_month =
"January\nFebruary\nMarch\nApril\nMay\nJune\nJul\nAugust\nSeptember\nOctober\nNovember\nDecember"
"
```

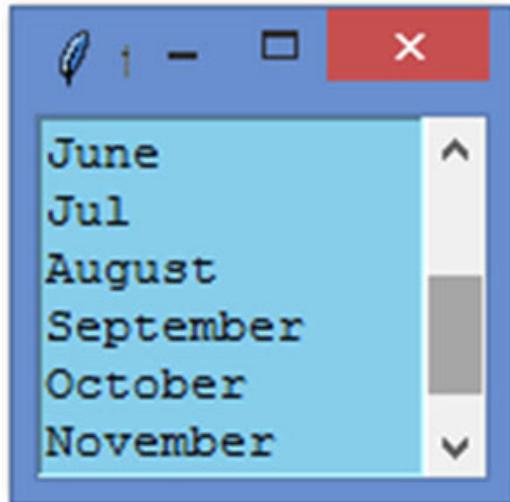


Figure 8.90

By now your code would look like this:

```
import tkinter as tk

#content for textbox
list_of_month = "January\nFebruary\nMarch\nApril\nMay\nJune\nJuly\nAugust\nSeptember\nOctober\nNovember"

mw = tk.Tk()
mw.geometry("100x100")

#Vertical Scrollbar
sb = tk.Scrollbar(mw)
sb.pack(side = tk.RIGHT,fill="y")

#create a Textbox
tb = tk.Text(mw,height = 500, width = 500, yscrollcommand = sb.set, bg="sky blue")
tb.pack()

sb.config(command = tb.yview)
tb.insert(tk.END,list_of_month)

mw.mainloop()
```

Figure 8.91

### Step 6: Now, add the horizontal scroll bar

```
#Horizontal Scroll bar
sb2 = tk.Scrollbar(mw,orient = tk.HORIZONTAL)
sb2.pack(side = tk.BOTTOM, fill="x")
```

### Step 7: Add content horizontally

```
list_of_days = "Monday Tuesday Wednesday Thursday Friday Saturday Sunday"
tb.insert(tk.END,list_of_days)
```

### Step 8: Define xscrollcommand

In step 3 we had created a textbox where we had set the yscrollcommand.

```
tb = tk.Text(mw,height = 500, width = 500, yscrollcommand = sb.set, bg="sky blue")
```

Now, we will add two more things to this statement:

- a. xscrollcommand = sb2.set
- b. wrap = "none"

So, the statement would now look like the following:

```
tb = tk.Text(mw,height = 500, width = 500, yscrollcommand = sb.set, xscrollcommand = sb2.set,
wrap = "none", bg="sky blue")
```

Configure the x scrollbar to the textbox.

```
sb2.config(command = tb.xview)
```

### Final code:

```
import tkinter as tk

#content for textbox
list_of_month =
"January\nFebruary\nMarch\nApril\nMay\nJune\nJul\nAugust\nSeptember\nOctober\nNovember\nDecember\n"
list_of_days = "Monday Tuesday Wednesday Thursday Friday Saturday Sunday"

mw = tk.Tk()
mw.geometry("100x100")

#Vertical Scrollbar
sb = tk.Scrollbar(mw)
sb.pack(side = tk.RIGHT, fill="y")

#Horizontal Scroll bar
sb2 = tk.Scrollbar(mw,orient = tk.HORIZONTAL)
sb2.pack(side = tk.BOTTOM, fill="x")

#create a Textbox
tb = tk.Text(mw,height = 500, width = 500, yscrollcommand = sb.set, xscrollcommand = sb2.set,
wrap = "none", bg="sky blue")
tb.pack()

sb.config(command = tb.yview)
sb2.config(command = tb.xview)

tb.insert(tk.END,list_of_month)
tb.insert(tk.END,list_of_days)

mw.mainloop()
```

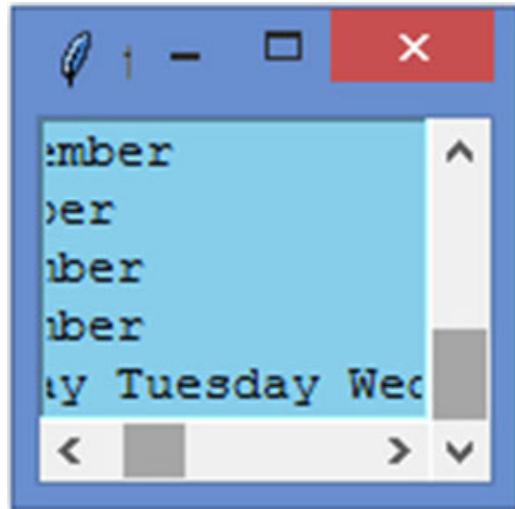


Figure 8.92

## 8.12 Text

You have already worked with a text in the last exercise. In this, you will learn more about this widget. This widget provides you a multiline text area. It is quite flexible, and can be used for accomplishing a lot of tasks. Mostly, developers use it whenever there is a requirement for multiline areas, but in addition to that, text areas can also be used to display links images and HTML, and so on. In the last example, we saw the basics of how to create a text widget. Just a small recap:

### **Step 1: Write a basic structure**

#### **Basic code:**

```
from tkinter import*
mw = Tk()
mw.geometry("150x150")
ta = Text(mw)
ta.insert(INSERT,"Sky is the limit!!")
ta.pack()
mw.mainloop()
```

#### **Output:**

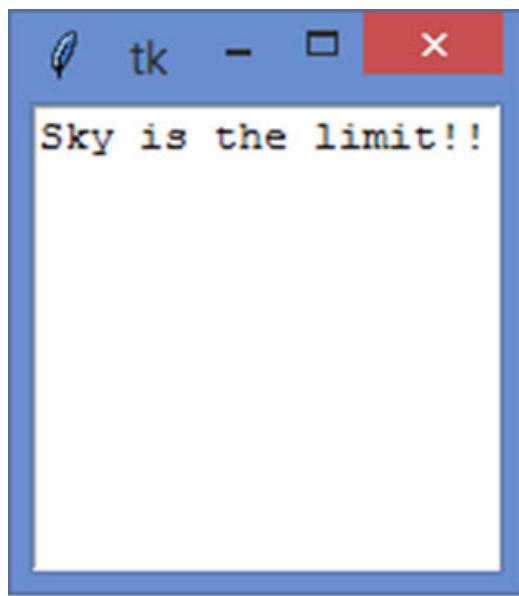


Figure 8.93

### Step 2: Insert an image in the text box

```
img = PhotoImage(file = "f:\\my_images\\sky.gif")
ta.image_create(END,image=img)
```

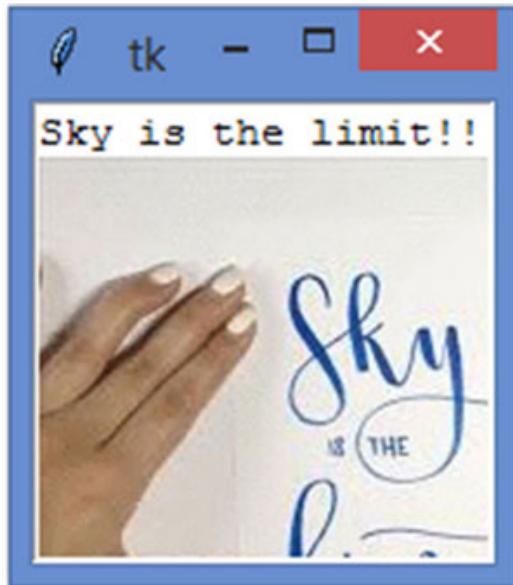


Figure 8.94

### Step 3: This requires the need to have scroll bars

```
#Vertical Scrollbar
sb = Scrollbar(mw)
sb.pack(side = RIGHT,fill="y")
```

```

#Horizontal Scroll bar
sb2 = Scrollbar(mw,orient = HORIZONTAL)
sb2.pack(side = BOTTOM, fill="x")

sb.config(command = ta.yview)
sb2.config(command = ta.xview)

```

Also, change the statement `ta = Text(mw)` to:

```
ta = Text(mw,yscrollcommand = sb.set,xscrollcommand = sb2.set)
```

So, your final code should look like this:

```

From tkinter import*
mw = Tk()
mw.geometry("150x150")
img = PhotoImage(file = "f:\\my_images\\sky.gif")

#Vertical Scrollbar
sb = Scrollbar(mw)
sb.pack(side = RIGHT,fill="y")

#Horizontal Scroll bar
sb2 = Scrollbar(mw,orient = HORIZONTAL)
sb2.pack(side = BOTTOM, fill="x")

#create text
ta = Text(mw,yscrollcommand = sb.set,xscrollcommand = sb2.set)

#Configure scrollbars to text widget
sb.config(command = ta.yview)
sb2.config(command = ta.xview)

#insert content
ta.insert(INSERT,"Sky is the limit!!")
ta.image_create(END,image=img)

ta.pack()
mw.mainloop()

```

## 8.13 Spinbox

A spinbox is a standard tkinter entry widget.

```

from tkinter import*
mw = Tk()
mw = Spinbox(mw)
mw.pack()
mainloop()

```

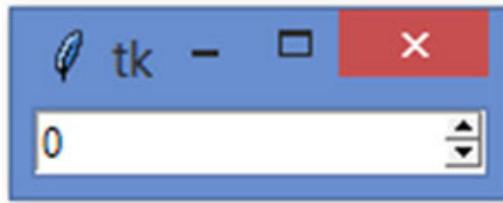


Figure 8.95

## Points to remember

- The `tkinter` module is offered by Python to create GUI.
- '`tkinter`' allows you to work with the classes of the TK module of **Tool Command Language (TCL)**.
- **TK** stands for tool kit, which is used by TCL to create graphics.
- TK provides standard GUI, which can be used by dynamic programming languages such as Python.
- Python developers can access TK with the help of the `tkinter` module.
- Python offers multiple ways for developing GUI.
- The easiest and the fastest way to create GUI is using Python with `tkinter`.
- The `tkinter` is the most commonly used module for GUI.
- A very important method called `mainloop()`, which must be called when you are prepared to run your application.
- The `mainloop()` method is an infinite loop that is used to run the application.
- It waits for an event to occur and processes the event as long as the window is not closed.
- You can set the title of the window using the `title()` method.
- You can change the size with the help of the `geometry()` function .
- To replace the image of the leaf with a new image, it is important that the image file is of `.ico` type only. The method used for this purpose is `wm_iconbitmap()`.
- The `tkinter` module provides different types of controls for GUI applications such as labels, buttons, radiobuttons, and so on. These controls are known as **widgets**.
- Steps involved in working with widgets:
  - **Step 1:** Import `tkinter`
  - **Step 2:** Create root window
  - **Step 3:** Write the code for working with widgets
  - **Step 4:** Call `mainloop()`

## **Questions and answers**

1. Which is the first and the most important step involved in working with GUI applications?

**Answer:** import tkinter.

2. Is tkinter the only way to develop GUI with Python?

**Answer:** No.

3. Does tkinter offer the easiest and the fastest way to create GUIs using Python?

- a. True
- b. False

**Answer:** True

4. The `mainloop()` in Python tkinter is used for:

**Answer:** Holding the window screen

**Question:** The \_\_\_\_\_ method must be called when you are prepared to run your application.

**Answer:** mainloop()

**Question:** Match the following:

|                                                             |                              |
|-------------------------------------------------------------|------------------------------|
| Create a root window                                        | <code>title()</code>         |
| Give window a title using this window                       | <code>.ico</code>            |
| Method used to assign new image to root window.             | <code>geometry()</code>      |
| Image file                                                  | <code>Tk()</code>            |
| Changing size of window                                     | <code>mainloop()</code>      |
| An infinite loop that is used to run the GUI application(). | <code>wm_iconbitmap()</code> |

**Answer:**

|                                                             |                              |
|-------------------------------------------------------------|------------------------------|
| Create root window.                                         | <code>Tk()</code>            |
| Give window a title using this window.                      | <code>title()</code>         |
| Method used to assign new image to root window.             | <code>wm_iconbitmap()</code> |
| The image file.                                             | <code>.ico</code>            |
| Changing size of window.                                    | <code>geometry()</code>      |
| An infinite loop that is used to run the GUI application(). | <code>mainloop()</code>      |

## CHAPTER 9

# MySQL and Python Graphical User Interface

### Introduction

You have learnt how to access MySQL database using Python programming. In this chapter, you will learn how to create a fully functional application where a user is presented with a graphical user interface that can be used to work with the database directly.

### Structure

- MySQLdb database
- Creating a table using GUI
- Insert data using GUI
- Create GUI to retrieve results

### Objectives

After reading this chapter, you will be able to create a functional Python code that allows users to update a database from a graphical user interface.

### 9.1 MySQLdb Database

In order to interact with a database using a Python program, it is important to first connect to the database server (which in this case is MySQL). In order to connect to the MySQL server, we require the following information:

1. **Host**: The host is the IP address of the MySQL server, which in our case, is the ‘localhost’.
2. **User**: Which is the username for connecting to the database.
3. **Password**: Which stands for the password for connecting to the database.
4. **Database**: The name of the database to which the Python program wants to connect.

#### **Syntax:**

```
Your_connection = mysql.connect(host=host_name,user=user_name,passwd=
```

```
database_password, charset='utf8', database= database_name)
```

**Note:** For higher versions of MySQL, you need not provide the charset='utf8' parameter. This is a requirement for MySQL 6.0.

The code required to work with the sql server is as follows. Please note that in the following case, the name of the database is '**textile**', user name is '**shopkeeper**', and the password is '**shoptoday**':

```
import mysql.connector as msq
pycon = msq.connect(host = 'localhost', user = 'shopkeeper', passwd = 'shoptoday', database =
'textile', charset = 'utf8')
---- your code comes here----
pycon.close()
```

A faster way to connect to a MySQL server is to store all the information required to connect to the server in a configuration file and then use it to connect to a database.

Create a dictionary that would store all the information required for connection:

```
dbConfig =
{'user':'shopkeeper','password':'shoptoday','host':'localhost','database':'textile','charset':'utf8'
'}
```

Now, while creating a database connection, simply unpack the values stored in the dictionary.

```
pycon = msq.connect(**dbConfig)
```

So, basically you can use the following code to connect:

```
import mysql.connector as msq
dbConfig =
{'user':'shopkeeper','password':'shoptoday','host':'localhost','database':'textile','charset':'utf8
'}
pycon = msq.connect(**dbConfig)
---- your code comes here---
pycon.close()
```

Both the code give you the same result. However, what you need to understand here is that simply hardcoding the login credentials is not a safe thing, especially when you are working on a web-based application. It is definitely the shortest and the fastest method, but definitely not secure. Ideally, you should put all the login credentials in a dictionary and save it in a Python file.

So, just create a python file, say **credential.py** and save it in the current working directory

Paste the following code in it:

```
dbConfig =
{'user':'shopkeeper','password':'shoptoday','host':'localhost','database':'textile','charset':'utf8'
```

```
'}
```

Now, try the following code:

```
import mysql.connector as msq
import credentials as c
pycon = msq.connect(**c.dbConfig)
print(pycon)
pycon.close()
```

### Output:

```
<mysql.connector.connection.MySQLConnection object at 0x037035F8>
```

**Note:** In [Chapter 4: MySQL for Python](#), you learnt that it is important to connect to MySQL using the `connect()` method, which returns a connection object. So, if the connection is successful and you give a command `print(connection_object)`, the information of the connection object will be displayed as follows.

So, basically you are keeping the credentials in a separate file in a dictionary object and unpacking it in your code. So, this provides secured connection.

Now, let's create a cursor object and execute the (`SHOW DATABASES`) command.

```
import mysql.connector as msq
import credentials as c
pycon = msq.connect(**c.dbConfig)
mycursor = pycon.cursor()
mycursor.execute('SHOW DATABASES;')
result_set = mycursor.fetchall()
for result in result_set:
 print(result)
pycon.close()
print('Done!!!')
```

### Output:

```
('information_schema',)
('textile',)
Done!!
```

Making database work with GUI.

Let's try to make GUI work with database.

**Note:** For more information on cursor object, please refer [Chapter 4: MySQL for Python](#).

## [9.2 Creating a table using GUI](#)

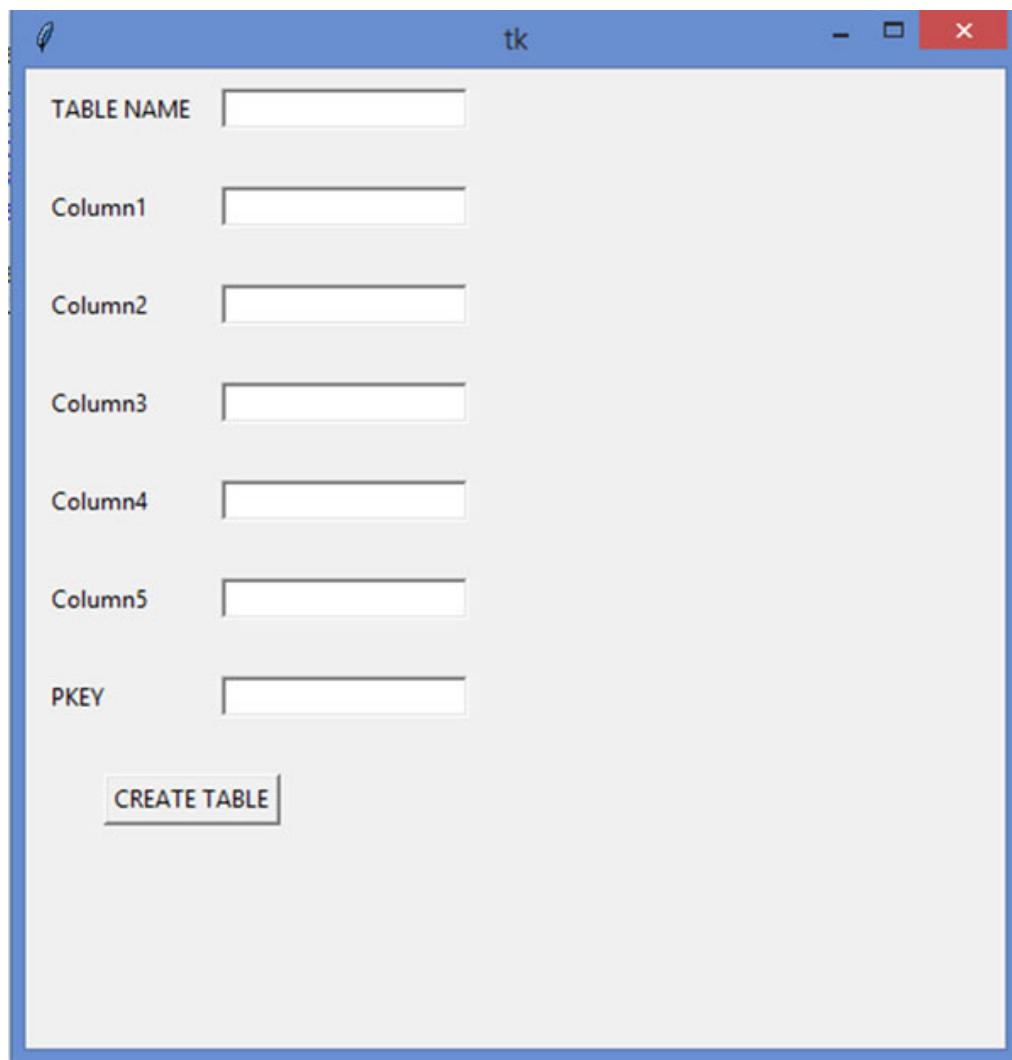
We will now try to create a table with a GUI. The example here is very simple and there is lot of scope for improvement in this design. But ideally, it is better to go with something

simple in order to understand how things work.

So basically, we want to create a table **Fabric** in **textile** database as follows:

```
CREATE TABLE FABRIC(
 FABRIC_ID SMALLINT NOT NULL AUTO_INCREMENT,
 FABRIC_NAME CHAR(20) NOT NULL,
 IMPORT_EXPORT CHAR(20) NOT NULL,
 COST_PER_METER SMALLINT NOT NULL,
 COLORS_AVAILABLE SMALLINT NOT NULL,
 PRIMARY KEY (FABRIC_ID)
);
```

The GUI that we want to create and use is as follows. It takes the table name, details of each column, and primary key. It will join this information and pass the statement to the database.



*Figure 9.1*

### Step 1: Prepare GUI

The first step is to prepare a GUI to create a table.

### Code:

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()
mw.geometry('500x500')
table_name = Label(mw, text = 'TABLE NAME')
table_name.place(x = 10, y = 10)

table_name_E = Entry(mw, bd = 2)
table_name_E.place(x = 100, y = 10)

column1 = Label(mw, text = 'Column1')
column1.place(x = 10, y = 60)

column1_E = Entry(mw, bd = 2)
column1_E.place(x = 100, y = 60)

column2 = Label(mw, text = 'Column2')
column2.place(x = 10, y = 110)

column2_E = Entry(mw, bd = 2)
column2_E.place(x = 100, y = 110)

column3 = Label(mw, text = 'Column3')
column3.place(x = 10, y = 160)

column3_E = Entry(mw, bd = 2)
column3_E.place(x = 100, y = 160)
column4 = Label(mw, text = 'Column4')
column4.place(x = 10, y = 210)

column4_E = Entry(mw, bd = 2)
column4_E.place(x = 100, y = 210)

column5 = Label(mw, text = 'Column5')
column5.place(x = 10, y = 260)

column5_E = Entry(mw, bd = 2)
column5_E.place(x = 100, y = 260)

columnpk = Label(mw, text = 'PKEY')
columnpk.place(x = 10, y = 310)

columnpk_E = Entry(mw, bd = 2)
columnpk_E.place(x = 100, y = 310)

create_button = Button(mw, text = 'CREATE TABLE')
create_button.place(x = 40, y = 360)

mw.mainloop()
```

Next, we want that on the click of create table button, the query should be passed on to the database.

## Step 2: Import statements to connect to database.

Before you create a function, import the following:

```
#import mysql.connector
import mysql.connector as msq1

#import credentials.py which has login details
import credentials as c
```

## Step 3: Define the function to create a table.

```
#define a function to create table
def create_tab():
 pycon = msq1.connect(**c.dbConfig)
 mycursor = pycon.cursor()
 statement = "CREATE TABLE "+str(table_name_E.get())+"("+str(column1_E.get())+", "+str(column2_E.get())+", "+str(column3_E.get())+", "+str(column4_E.get())+", "+str(column5_E.get())+", "+"PRIMARY KEY("+str(columnpk_E.get())+"));"
 print("Passing following information to MySQL textile database:"+statement)
 mycursor.execute(statement)
 pycon.close()
 print('Done!!!')
```

## Step 4: Call the `create_tab` function on the click of a `create table` button.

```
create_button = Button(mw, text = 'CREATE TABLE', command = create_tab)
```

So, the final code should look like this:

```
#import tkinter
from tkinter import *

#import mysql.connector
import mysql.connector as msq1

#import credentials.py which has login details
import credentials as c

#define a function to create table
def create_tab():
 pycon = msq1.connect(**c.dbConfig)
 mycursor = pycon.cursor()
 statement = "CREATE TABLE "+str(table_name_E.get())+"("+str(column1_E.get())+", "+str(column2_E.get())+", "+str(column3_E.get())+", "+str(column4_E.get())+", "+str(column5_E.get())+", "+"PRIMARY KEY("+str(columnpk_E.get())+"));"
 print("Passing following information to MySQL textile database:"+statement)
 mycursor.execute(statement)
 pycon.close()

#Create instance of window
mw = Tk()
mw.geometry('500x500')

table_name = Label(mw, text = 'TABLE NAME')
table_name.place(x =10, y =10)
```

```
table_name_E = Entry(mw,bd = 2)
table_name_E.place(x = 100, y =10)

column1 = Label(mw, text = 'Column1')
column1.place(x = 10, y = 60)

column1_E = Entry(mw,bd = 2)
column1_E.place(x = 100, y = 60)

column2 = Label(mw, text = 'Column2')
column2.place(x = 10, y = 110)

column2_E = Entry(mw,bd = 2)
column2_E.place(x = 100, y = 110)

column3 = Label(mw, text = 'Column3')
column3.place(x = 10, y = 160)

column3_E = Entry(mw,bd = 2)
column3_E.place(x = 100, y = 160)

column4 = Label(mw, text = 'Column4')
column4.place(x = 10, y = 210)

column4_E = Entry(mw,bd = 2)
column4_E.place(x = 100, y = 210)

column5 = Label(mw, text = 'Column5')
column5.place(x = 10, y = 260)

column5_E = Entry(mw,bd = 2)
column5_E.place(x = 100, y = 260)

columnpk = Label(mw, text = 'PKEY')

columnpk.place(x = 10, y = 310)

columnpk_E = Entry(mw,bd = 2)
columnpk_E.place(x = 100, y = 310)

create_button = Button(mw,text = 'CREATE TABLE', command = create_tab)
create_button.place(x = 40, y = 360)

mw.mainloop()
```

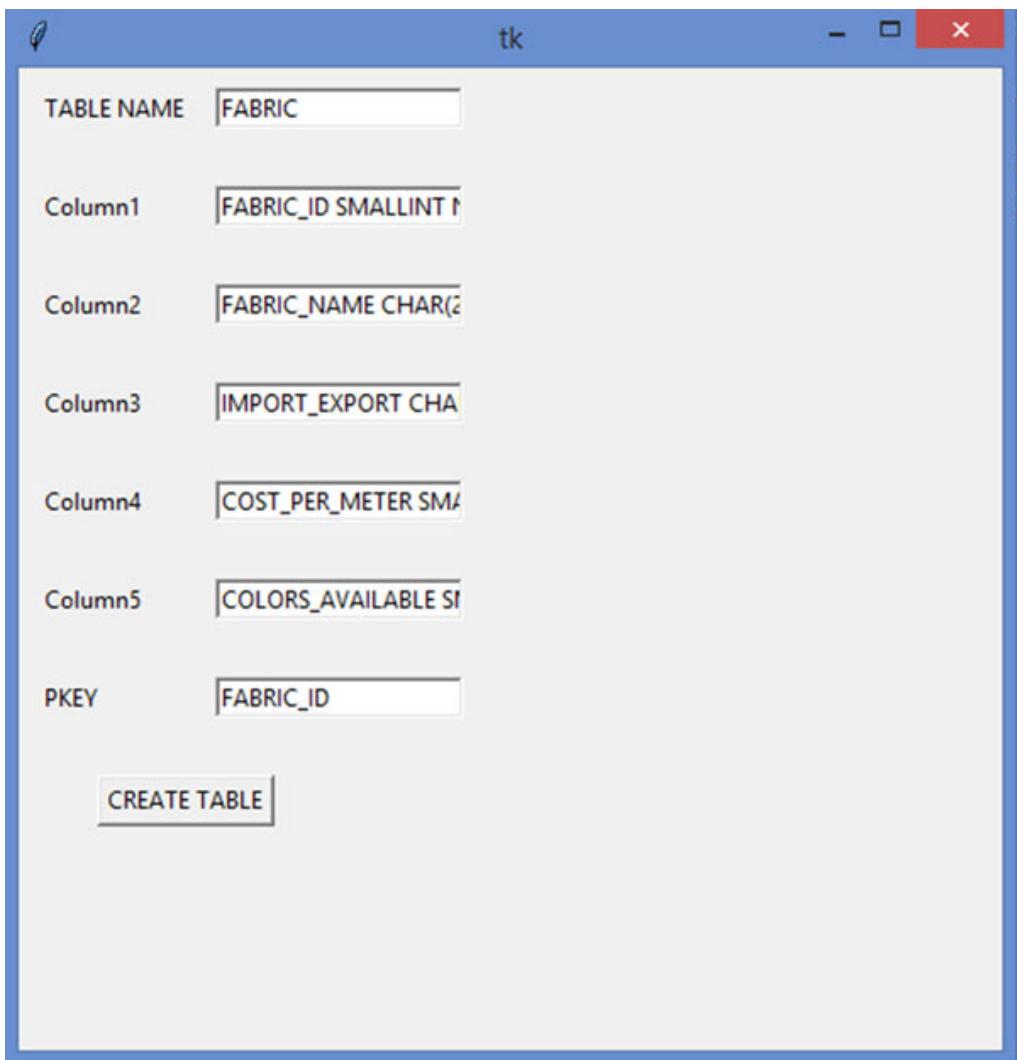


Figure 9.2

## Output:

```
Passing following information to MySQL textile database:CREATE TABLE FABRIC(FABRIC_ID SMALLINT NOT
NULL AUTO_INCREMENT, FABRIC_NAME CHAR(20) NOT NULL, IMPORT_EXPORT CHAR(20) NOT NULL,
COST_PER_METER SMALLINT NOT NULL, COLORS_AVAILABLE SMALLINT NOT NULL, PRIMARY KEY(FABRIC_ID));
Done!!
```

```
mysql> SHOW COLUMNS FROM FABRIC;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default |
+-----+-----+-----+-----+-----+
FABRIC_ID	smallint<6>	NO	PRI	NULL
FABRIC_NAME	char<20>	NO		
IMPORT_EXPORT	char<20>	NO		
COST_PER_METER	smallint<6>	NO		
COLORS_AVAILABLE	smallint<6>	NO		
+-----+-----+-----+-----+-----+
5 rows in set (0.08 sec)
```

Figure 9.3

## 9.3 Insert data using GUI

In this section, you will learn how to insert data using GUI.

### **Step 1.** Code for GUI.

We will make minor changes to the GUI code of *section 9.2*.

```
#Import tkinter
from tkinter import*

#Create instance of window
mw = Tk()
mw.geometry('400x400')
table_name = Label(mw, text = 'TABLE NAME')
table_name.place(x =10, y =10)

table_name_E = Entry(mw,bd = 2)
table_name_E.place(x = 150, y =10)

column1 = Label(mw, text = 'FABRIC_NAME')
column1.place(x = 10, y = 60)

column1_E = Entry(mw,bd = 2)
column1_E.place(x = 150, y = 60)

column2 = Label(mw, text = 'IMPORT_EXPORT')
column2.place(x = 10, y = 110)

column2_E = Entry(mw,bd = 2)
column2_E.place(x = 150, y = 110)

column3 = Label(mw, text = 'COST_PER_METER')
column3.place(x = 10, y = 160)

column3_E = Entry(mw,bd = 2)
column3_E.place(x = 150, y = 160)

column4 = Label(mw, text = 'COLORS_AVAILABLE')
column4.place(x = 10, y = 210)

column4_E = Entry(mw,bd = 2)
column4_E.place(x = 150, y = 210)

insert_button = Button(mw,text = 'INSERT VALUES')
insert_button.place(x = 40, y = 260)

mw.mainloop()
```

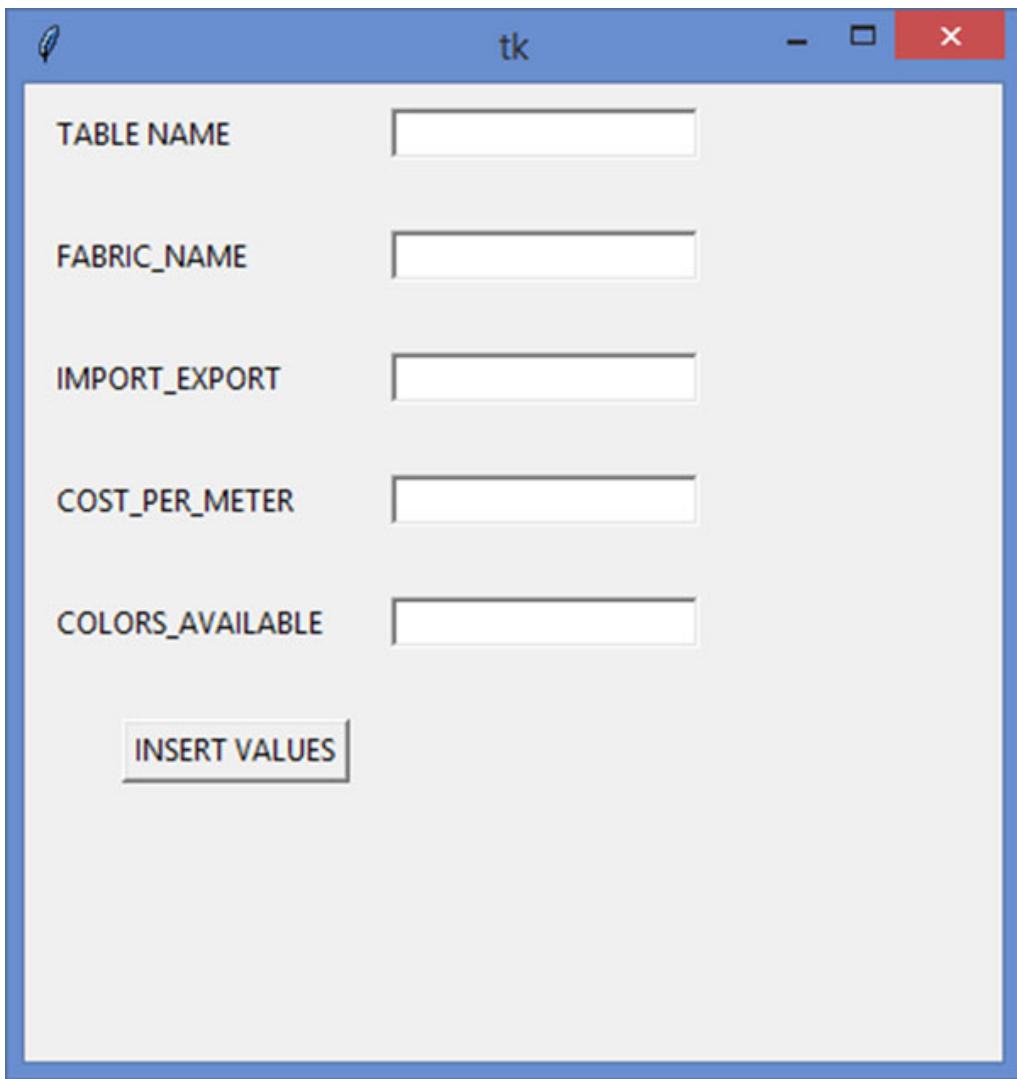


Figure 9.4

## Step 2: Define Function

```
#import mysql.connector
import mysql.connector as msq1

#import credentials.py which has login details
import credentials as c

#define a function to create table
def insert_val():
 pycon = msq1.connect(**c.dbConfig)
 mycursor = pycon.cursor()
 statement = "INSERT INTO "+str(table_name_E.get())+
 (FABRIC_NAME, IMPORT_EXPORT, COST_PER_METER, COLORS_AVAILABLE) VALUES
 ("+"!"+str(column1_E.get())+", "+"!"+str(column2_E.get())+", "+'{0},"
 {1});".format(int(column3_E.get()),int(column4_E.get()))
 print("Passing following information to MySQL textile database: "+statement)
 mycursor.execute(statement)
 pycon.commit()
```

```
 pycon.close()
print('Done!!!')
```

### Step 3: Map the function with the insert button

```
insert_button = Button(mw, text = 'INSERT VALUES', command = insert_val)
insert_button.place(x = 40, y = 260)
```

The final code is given as follows:

#### Code:

```
#Import tkinter
from tkinter import*
#import mysql.connector
import mysql.connector as msq1
#import credentials.py which has login details
import credentials as c

#define a function to create table
def insert_val():
 pycon = msq1.connect(**c.dbConfig)
 mycursor = pycon.cursor()
 statement = "INSERT INTO "+str(table_name_E.get())+""
 (FABRIC_NAME,IMPORT_EXPORT,COST_PER_METER,COLORS_AVAILABLE) VALUES
 ("+"!"+str(column1_E.get())+", "+"!"+str(column2_E.get())+", "+{0},
 {1});".format(int(column3_E.get()),int(column4_E.get()))
 print("Passing following information to MySQL textile database:"+statement)
 mycursor.execute(statement)
 pycon.commit()
 pycon.close()
 print('Done!!!')

#Create instance of window
mw = Tk()
mw.geometry('400x400')
table_name = Label(mw, text = 'TABLE NAME')
table_name.place(x = 10, y = 10)

table_name_E = Entry(mw,bd = 2)
table_name_E.place(x = 150, y = 10)

column1 = Label(mw, text = 'FABRIC_NAME')
column1.place(x = 10, y = 60)

column1_E = Entry(mw,bd = 2)
column1_E.place(x = 150, y = 60)

column2 = Label(mw, text = 'IMPORT_EXPORT')
column2.place(x = 10, y = 110)

column2_E = Entry(mw,bd = 2)
column2_E.place(x = 150, y = 110)

column3 = Label(mw, text = 'COST_PER_METER')
column3.place(x = 10, y = 160)

column3_E = Entry(mw,bd = 2)
```

```

column3_E.place(x = 150, y = 160)

column4 = Label(mw, text = 'COLORS_AVAILABLE')
column4.place(x = 10, y = 210)

column4_E = Entry(mw, bd = 2)
column4_E.place(x = 150, y = 210)

insert_button = Button(mw, text = 'INSERT VALUES', command = insert_val)
insert_button.place(x = 40, y = 260)

mw.mainloop()

```

## Output:

Passing the following information to MySQL textile database:

```

INSERT INTO FABRIC(FABRIC_NAME,IMPORT_EXPORT,COST_PER_METER,COLORS_AVAILABLE)
VALUES('COTTON','EXPORT',80,12);

```

Done!!

| FABRIC_ID | FABRIC_NAME | IMPORT_EXPORT | COST_PER_METER | COLORS_AVAILABLE |
|-----------|-------------|---------------|----------------|------------------|
| 2         | COTTON      | EXPORT        | 80             | 12               |

1 row in set <0.00 sec>

*Figure 9.5*

## 9.4 Create a GUI to retrieve results

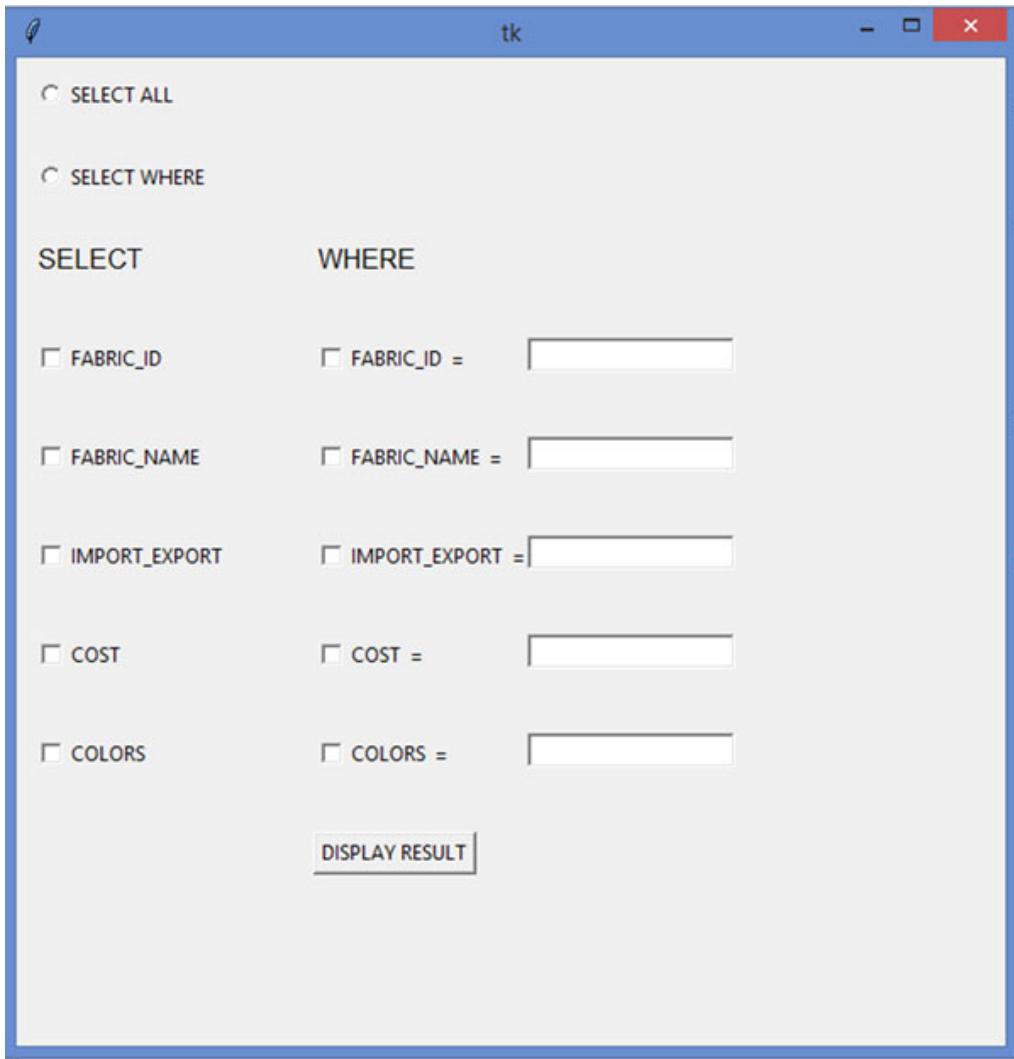


Figure 9.6

If you select the `select all` button then all the records will be displayed. Else you can select the columns that you require and the filter conditions as well.

### Step 1: Write the code for GUI

```
#Import tkinter
from tkinter import*
from tkinter import messagebox

#Create instance of window
mw = Tk()
mw.geometry('600x600')

#Variable for Radio
var = IntVar()

#Variable for Checkbutton
cvar1 = IntVar()
cvar2 = IntVar()
```

```

cvar3 = IntVar()
cvar4 = IntVar()
cvar5 = IntVar()

svar1 = IntVar()
svar2 = IntVar()
svar3 = IntVar()
svar4 = IntVar()
svar5 = IntVar()

#Radio button SELECT ALL
selectRadio = Radiobutton(mw, text = 'SELECT ALL',variable=var, value=1)
selectRadio.place(x =10, y =10)

#SELECT FILTER
selectFilterRadio = Radiobutton(mw, text = 'SELECT WHERE',variable=var, value=2)
selectFilterRadio.place(x = 10, y =60)

#LABEL SELECT
selectLabel = Label(mw, text = 'SELECT',font='Bold')
selectLabel.place(x = 10, y = 110)

#CHECKS FOR SELECT

#FABRIC ID
fabricIdCheck = Checkbutton(mw,text='FABRIC_ID',variable = svar1, onvalue = 1,offvalue = 0)
fabricIdCheck.place(x = 10, y = 170)

#FABRIC NAME
fabricNameCheck = Checkbutton(mw,text='FABRIC_NAME',variable = svar2, onvalue = 1,offvalue = 0)
fabricNameCheck.place(x = 10, y = 230)

#IMPORT_EXPORT
ieCheck = Checkbutton(mw,text='IMPORT_EXPORT',variable = svar3, onvalue = 1,offvalue = 0)
ieCheck.place(x = 10, y = 290)

#COST
costCheck = Checkbutton(mw,text='COST',variable = svar4, onvalue = 1,offvalue = 0)
costCheck.place(x = 10, y = 350)

#COLORS
colorsCheck = Checkbutton(mw, text = 'COLORS',variable = svar5, onvalue = 1,offvalue = 0)
colorsCheck.place(x = 10, y = 410)

#LABEL WHERE
whereLabel = Label(mw, text = 'WHERE', font='Bold')
whereLabel.place(x = 180, y = 110)

#CHECKS FOR WHERE
#FABRIC ID
fabricIdCheck2 = Checkbutton(mw,text='FABRIC_ID = ',variable = cvar1, onvalue = 1,offvalue = 0)
fabricIdCheck2.place(x = 180, y = 170)
fabricIdCheck2entry = Entry(mw, bd = 2)
fabricIdCheck2entry.place(x = 310, y = 170)

#FABRIC NAME
fabricNameCheck2 = Checkbutton(mw,text='FABRIC_NAME = ',variable = cvar2, onvalue = 1,offvalue = 0)
fabricNameCheck2.place(x = 180, y = 230)
fabricNameCheck2entry = Entry(mw, bd = 2)
fabricNameCheck2entry.place(x = 310, y = 230)

```

```

#IMPORT_EXPORT
ieCheck2 = Checkbutton(mw,text='IMPORT_EXPORT = ',variable = cvar3, onvalue = 1,offvalue = 0)
ieCheck2.place(x = 180, y = 290)
ieCheck2entry = Entry(mw, bd = 2)
ieCheck2entry.place(x = 310, y = 290)

#COST
costCheck2 = Checkbutton(mw,text='COST = ',variable = cvar4, onvalue = 1,offvalue = 0)
costCheck2.place(x = 180, y = 350)
costCheck2entry = Entry(mw, bd = 2)
costCheck2entry.place(x = 310, y = 350)

#COLORS
colorsCheck2 = Checkbutton(mw, text = 'COLORS = ',variable = cvar5, onvalue = 1,offvalue = 0)
colorsCheck2.place(x = 180, y = 410)
colorsCheck2entry = Entry(mw, bd = 2)
colorsCheck2entry.place(x = 310, y = 410)

#DISPLAY RESULTS
display_button = Button(mw,text = 'DISPLAY RESULT')
display_button.place(x = 180, y = 470)

mw.mainloop()

```

## Step 2: Write down the import statements for database connection

```

#import mysql.connector
import mysql.connector as msq1
#import credentials.py which has login details
import credentials as c

```

## Step 3: Define the steps for displaying data

First create an outline:

```

def display_result():
 print("var = ",var.get())
 pycon = msq1.connect(**c.dbConfig)
 mycursor = pycon.cursor()
 statement = ""

 if (var.get() == 1):
#your code here
 elif (var.get() == 2):
#your code here

 mycursor.execute(statement)
 result_set = mycursor.fetchall()
 for result in result_set:
 print(result)
 messagebox.showinfo("FABRIC from TEXTILE",result_set)
 pycon.close()
 print('Done!!')

```

Let's first write the code for the **SELECT ALL** radio button. If this radio button is selected and you click on the **DISPLAY RESULT** button, then all the record of the table will be

displayed. In the definition of the **DISPLAY RESULT** button, type `command = display_result` and check whether all the results are displayed properly.

```
def display_result():
 print("var = ", var.get())
 pycon = msql.connect(**c.dbConfig)
 mycursor = pycon.cursor()
 statement = ""

 if (var.get() == 1):
 statement = "SELECT * FROM FABRIC;"

 elif (var.get() == 2):
 #your code here

 mycursor.execute(statement)
 result_set = mycursor.fetchall()
 for result in result_set:
 print(result)
 messagebox.showinfo("FABRIC from TEXTILE", result_set)
 pycon.close()
 print('Done!!!')
```

Changes in button code:

```
#DISPLAY RESULTS
display_button = Button(mw, text = 'DISPLAY RESULT', command = display_result)
display_button.place(x = 180, y = 470)
```

SELECT ALL  
 SELECT WHERE

SELECT                    WHERE

FABRIC\_ID             FABRIC\_ID =

FABRIC\_NAME         FABRIC\_NAME =

IMPORT\_EXPORT     IMPORT\_EXPORT =

COST                 COST =

COLORS             COLORS =

Figure 9.7

**Output:**



Figure 9.8

Now check the database by displaying all records.

| FABRIC_ID | FABRIC_NAME | IMPORT_EXPORT | COST_PER_METER | COLORS_AVAILABLE |
|-----------|-------------|---------------|----------------|------------------|
| 2         | COTTON      | EXPORT        | 80             | 12               |
| 3         | SILK        | EXPORT        | 300            | 15               |
| 4         | CHIFFON     | IMPORT        | 250            | 8                |
| 5         | RAYON       | EXPORT        | 50             | 8                |
| 6         | FLEECE      | IMPORT        | 190            | 25               |

5 rows in set <0.00 sec>

Figure 9.9

All five records are displayed.

The tricky part now is to form the query to display certain columns based on filter conditions.

When the second radio button is selected, you have to check which all values you want to see based on what conditions. So, in the first half, we work on how to form the **SELECT** part of the query, and in the second half, we work on how to form the **WHERE** part of the query.

```
def display_result():
 print("var = ", var.get())
 pycon = msql.connect(**c.dbConfig)
 mycursor = pycon.cursor()
 statement = ""

 if (var.get() == 1):
 statement = "SELECT * FROM FABRIC;"
 elif (var.get() == 2):

 statement+= "SELECT "

 if(svar1.get() == 1):
 statement+= "FABRIC_ID "

 if(svar1.get() == 1 and svar2.get() == 1):
 statement+= ", FABRIC_NAME "
 elif(svar1.get() == 0 and svar2.get() == 1):
 statement+= "FABRIC_NAME "

 if((svar1.get() == 1 or svar2.get() == 1) and svar3.get() == 1):
 statement+= ", IMPORT_EXPORT "

 elif((svar1.get() == 0 and svar2.get() == 0) and svar3.get() == 1):
 statement+= "IMPORT_EXPORT "
 if((svar1.get() == 1 or svar2.get() == 1 or svar3.get() == 1) and svar4.get() == 1):
 statement+= ", COST_PER_METER "

 elif((svar1.get() == 0 and svar2.get() == 0 and svar3.get() == 0) and svar4.get() == 1):
 statement+= "COST_PER_METER "

 if((svar1.get() == 1 or svar2.get() == 1 or svar3.get() == 1 or svar4.get() == 1) and svar5.get() == 1):
 statement+= ", COLORS_AVAILABLE "
 elif((svar1.get() == 0 and svar2.get() == 0 and svar3.get() == 0 and svar4.get() == 0) and svar5.get() == 1):
 statement+= "COLORS_AVAILABLE "
```

```

statement+=" COLORS_AVAILABLE "

statement+=" FROM FABRIC WHERE "
if(cvar1.get() == 1):
 statement+=" FABRIC_ID = '{}' ".format(str(fabricIdCheck2entry.get()))

if(cvar1.get() == 1 and cvar2.get() == 1):
 statement+=" AND FABRIC_NAME = '{}' ".format(str(fabricNameCheck2entry.get()))

elif(cvar1.get() == 0 and cvar2.get() == 1):
 statement+=" FABRIC_NAME = '{}' ".format(str(fabricNameCheck2entry.get()))

if((cvar1.get() == 0 and cvar2.get() == 0) and cvar3.get() == 1):
 statement+=" IMPORT_EXPORT = '{}' ".format(str(ieCheck2entry.get()))

elif((cvar1.get() == 1 or cvar2.get() == 1) and cvar3.get() == 1):
 statement+=" AND IMPORT_EXPORT = '{}' ".format(str(ieCheck2entry.get()))

if((cvar1.get() == 0 and cvar2.get() == 0 and cvar3.get() == 0) and cvar4.get() == 1):
 statement+=" COST_PER_METER = {} ".format(int(costCheck2entry.get()))

elif((cvar1.get() == 1 or cvar2.get() == 1 or cvar3.get() == 1) and cvar4.get() == 1):
 statement+=" AND COST_PER_METER = {} ".format(int(costCheck2entry.get()))

if((cvar1.get() == 0 and cvar2.get() == 0 and cvar3.get() == 0 and cvar4.get() == 0) and cvar5.get() == 1):
 statement+=" COST_PER_METER = {} ".format(int(colorsCheck2entry.get()))

elif((cvar1.get() == 1 or cvar2.get() == 1 or cvar3.get() == 1 or cvar4.get() == 1) and cvar5.get() == 1):
 statement+=" AND COST_PER_METER = {} ".format(int(colorsCheck2entry.get()))

statement += ";"
mycursor.execute(statement)
result_set = mycursor.fetchall()
for result in result_set:
 print(result)
messagebox.showinfo("FABRIC from TEXTILE",result_set)
pycon.close()
print('Done!!')

```

Syntax for the select statement is **select col1 col2... where col3 ='value' and col6 ='value 2'**

So, we check which all check boxes are selected under the select section and create select part of the the query and then we check based on what all values we have to retrieve resut and get those values.

### Final Code:

```

#import tkinter
from tkinter import*
from tkinter import messagebox
#import mysql.connector
import mysql.connector as msq
#import credentials.py which has login details
import credentials as c

def display_result():
 print("var = ",var.get())

```

```

pycon = msq1.connect(**c.dbConfig)
mycursor = pycon.cursor()
statement = ""

if (var.get() == 1):
 statement = "SELECT * FROM FABRIC;"
elif (var.get() == 2):
 statement+= "SELECT "

 if(svar1.get() == 1):
 statement+= "FABRIC_ID "

 if(svar1.get() == 1 and svar2.get() == 1):
 statement+= ", FABRIC_NAME "
 elif(svar1.get() == 0 and svar2.get() == 1):
 statement+= "FABRIC_NAME "

 if((svar1.get() == 1 or svar2.get() == 1) and svar3.get() == 1):
 statement+= ", IMPORT_EXPORT "
 elif((svar1.get() == 0 and svar2.get() == 0) and svar3.get() == 1):
 statement+= "IMPORT_EXPORT "

 if((svar1.get() == 1 or svar2.get() == 1 or svar3.get() == 1) and svar4.get() == 1):
 statement+= ", COST_PER_METER "
 elif((svar1.get() == 0 and svar2.get() == 0 and svar3.get() == 0) and svar4.get() == 1):
 statement+= "COST_PER_METER "

 if((svar1.get() == 1 or svar2.get() == 1 or svar3.get() == 1 or svar4.get() == 1)and
svar5.get() == 1):
 statement+= ", COLORS_AVAILABLE "
 elif((svar1.get() == 0 and svar2.get() == 0 and svar3.get() == 0 and svar4.get() == 0)and
svar5.get() == 1):
 statement+= " COLORS_AVAILABLE "

statement+= "FROM FABRIC WHERE "

if(cvar1.get() == 1):
 statement+= "FABRIC_ID = '{}' ".format(str (fabricIdCheck2entry.get()))

if(cvar1.get() == 1 and cvar2.get() == 1):
 statement+= " AND FABRIC_NAME = '{}' ".format(str (fabricNameCheck2entry.get()))

elif(cvar1.get() == 0 and cvar2.get() == 1):
 statement+= " FABRIC_NAME = '{}' ".format(str (fabricNameCheck2entry.get()))

if((cvar1.get() == 0 and cvar2.get() == 0) and cvar3.get() == 1):
 statement+= " IMPORT_EXPORT = '{}' ".format(str (ieCheck2entry.get()))

elif((cvar1.get() == 1 or cvar2.get() == 1) and cvar3.get() == 1):
 statement+= " AND IMPORT_EXPORT = '{}' ".format(str (ieCheck2entry.get()))

if((cvar1.get() == 0 and cvar2.get() == 0 and cvar3.get() == 0) and cvar4.get() == 1):
 statement+= " COST_PER_METER = {} ".format(int (costCheck2entry.get()))

elif((cvar1.get() == 1 or cvar2.get() == 1 or cvar3.get() == 1) and cvar4.get() == 1):
 statement+= " AND COST_PER_METER = {} ".format(int (costCheck2entry.get()))

if((cvar1.get() == 0 and cvar2.get() == 0 and cvar3.get() == 0 and cvar4.get() == 0)and
cvar5.get() == 1):
 statement+= " COST_PER_METER = {} ".format(int (colorsCheck2entry.get()))
elif((cvar1.get() == 1 or cvar2.get() == 1 or cvar3.get() == 1 or cvar4.get() == 1)and
cvar5.get() == 1):

```

```

 statement+=" AND COST_PER_METER = {} ".format(int (colorsCheck2entry.get()))

statement += ";"
mycursor.execute(statement)
result_set = mycursor.fetchall()
for result in result_set:
 print(result)
messagebox.showinfo("FABRIC from TEXTILE",result_set)
pycon.close()
print('Done!!')

#Create instance of window
mw = Tk()
mw.geometry('600x600')
#Variable for Radio
var = IntVar()

#Variable for Checkbutton-WHERE SECTION
cvar1 = IntVar()
cvar2 = IntVar()
cvar3 = IntVar()
cvar4 = IntVar()
cvar5 = IntVar()

#Variable for Checkbutton - SELECT SECTION
svar1 = IntVar()
svar2 = IntVar()
svar3 = IntVar()
svar4 = IntVar()
svar5 = IntVar()

#Radio button SELECT ALL
selectRadio = Radiobutton(mw, text = 'SELECT ALL',variable=var, value=1)
selectRadio.place(x =10, y =10)

#SELECT FILTER
selectFilterRadio = Radiobutton(mw, text = 'SELECT WHERE',variable=var, value=2)
selectFilterRadio.place(x = 10, y =60)

#LABEL SELECT
selectLabel = Label(mw, text = 'SELECT',font='Bold')
selectLabel.place(x = 10, y = 110)

#CHECKS FOR SELECT

#FABRIC ID

fabricIdCheck = Checkbutton(mw,text='FABRIC_ID',variable = svar1, onvalue = 1,offvalue = 0)
fabricIdCheck.place(x = 10, y = 170)

#FABRIC NAME
fabricNameCheck = Checkbutton(mw,text='FABRIC_NAME',variable = svar2, onvalue = 1,offvalue = 0)
fabricNameCheck.place(x = 10, y = 230)

#IMPORT_EXPORT
ieCheck = Checkbutton(mw,text='IMPORT_EXPORT',variable = svar3, onvalue = 1,offvalue = 0)
ieCheck.place(x = 10, y = 290)

#COST
costCheck = Checkbutton(mw,text='COST',variable = svar4, onvalue = 1,offvalue = 0)
costCheck.place(x = 10, y = 350)

```

```

#COLORS
colorsCheck = Checkbutton(mw, text = 'COLORS',variable = svar5, onvalue = 1,offvalue = 0)
colorsCheck.place(x = 10, y = 410)

#LABEL WHERE
whereLabel = Label(mw, text = 'WHERE', font='Bold')
whereLabel.place(x = 180, y = 110)

#CHECKS FOR WHERE
#FABRIC ID
fabricIdCheck2 = Checkbutton(mw,text='FABRIC_ID = ',variable = cvar1, onvalue = 1,offvalue = 0)
fabricIdCheck2.place(x = 180, y = 170)
fabricIdCheck2entry = Entry(mw, bd = 2)
fabricIdCheck2entry.place(x = 310, y = 170)

#FABRIC NAME
fabricNameCheck2 = Checkbutton(mw, text='FABRIC_NAME = ',variable = cvar2, onvalue = 1,offvalue = 0)
fabricNameCheck2.place(x = 180, y = 230)
fabricNameCheck2entry = Entry(mw, bd = 2)
fabricNameCheck2entry.place(x = 310, y = 230)

#IMPORT_EXPORT
ieCheck2 = Checkbutton(mw, text='IMPORT_EXPORT = ',variable = cvar3, onvalue = 1,offvalue = 0)
ieCheck2.place(x = 180, y = 290)
ieCheck2entry = Entry(mw, bd = 2)
ieCheck2entry.place(x = 310, y = 290)

#COST
costCheck2 = Checkbutton(mw, text='COST = ',variable = cvar4, onvalue = 1,offvalue = 0)
costCheck2.place(x = 180, y = 350)
costCheck2entry = Entry(mw, bd = 2)
costCheck2entry.place(x = 310, y = 350)

#COLORS
colorsCheck2 = Checkbutton(mw, text = 'COLORS = ',variable = cvar5, onvalue = 1,offvalue = 0)
colorsCheck2.place(x = 180, y = 410)
colorsCheck2entry = Entry(mw, bd = 2)
colorsCheck2entry.place(x = 310, y = 410)

#DISPLAY RESULTS
display_button = Button(mw, text = 'DISPLAY RESULT',command = display_result)
display_button.place(x = 180, y = 470)

mw.mainloop()

```

## Output:

tk

SELECT ALL  
 SELECT WHERE

SELECT WHERE

FABRIC\_ID       FABRIC\_ID =

FABRIC\_NAME       FABRIC\_NAME =

IMPORT\_EXPORT       IMPORT\_EXPORT =  EXPORT

COST       COST =

COLORS       COLORS =

Figure 9.10

**Output:**

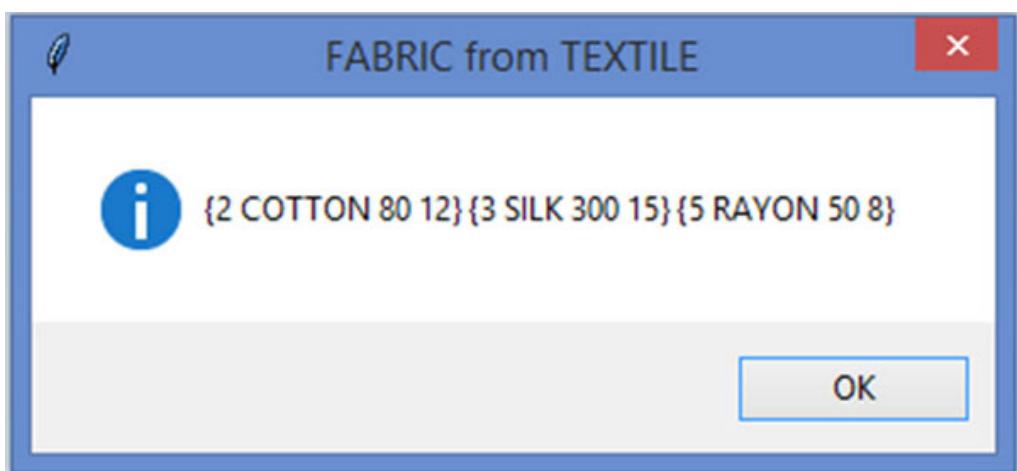


Figure 9.11

The result is right because there are only three entries for `IMPORT_EXPORT = 'EXPORT'`.

| FABRIC_ID | FABRIC_NAME | IMPORT_EXPORT | COST_PER_METER | COLORS_AVAILABLE |
|-----------|-------------|---------------|----------------|------------------|
| 2         | COTTON      | EXPORT        | 80             | 12               |
| 3         | SILK        | EXPORT        | 300            | 15               |
| 4         | CHIFFON     | IMPORT        | 250            | 8                |
| 5         | RAYON       | EXPORT        | 50             | 8                |
| 6         | FLEECE      | IMPORT        | 190            | 25               |

5 rows in set (0.00 sec)

Figure 9.12

## Activity

Try to make changes to the preceding GUI and code to create a GUI that can update records in the database.

## Conclusion

In [chapter 4, MySQL for Python](#), you learnt how Python programs can be used to connect to the database, and in [chapter 8, Creating GUI forms and adding widgets](#) you learnt how to create GUI using the `tkinter` module. In this chapter, you learnt how to use GUI to connect to database. You are now equipped with all the knowledge required to create a functional-application that has a user interface and can connect to a database. Use this knowledge to create a real-time project.

# CHAPTER 10

## Stack, Queue, and Deque

### Introduction

In this chapter, you will learn how to implement stack, queue, and Deque using Python.

### Structure

- Stack
  - Implementation of stack in Python
  - `push()` and `pop()` functions
- Queue
  - Basic Queue Functions
  - Implementation of Queue
  - Implementation of stack as single queue
  - Implementation of queue using two stacks
- Deque

### Objectives

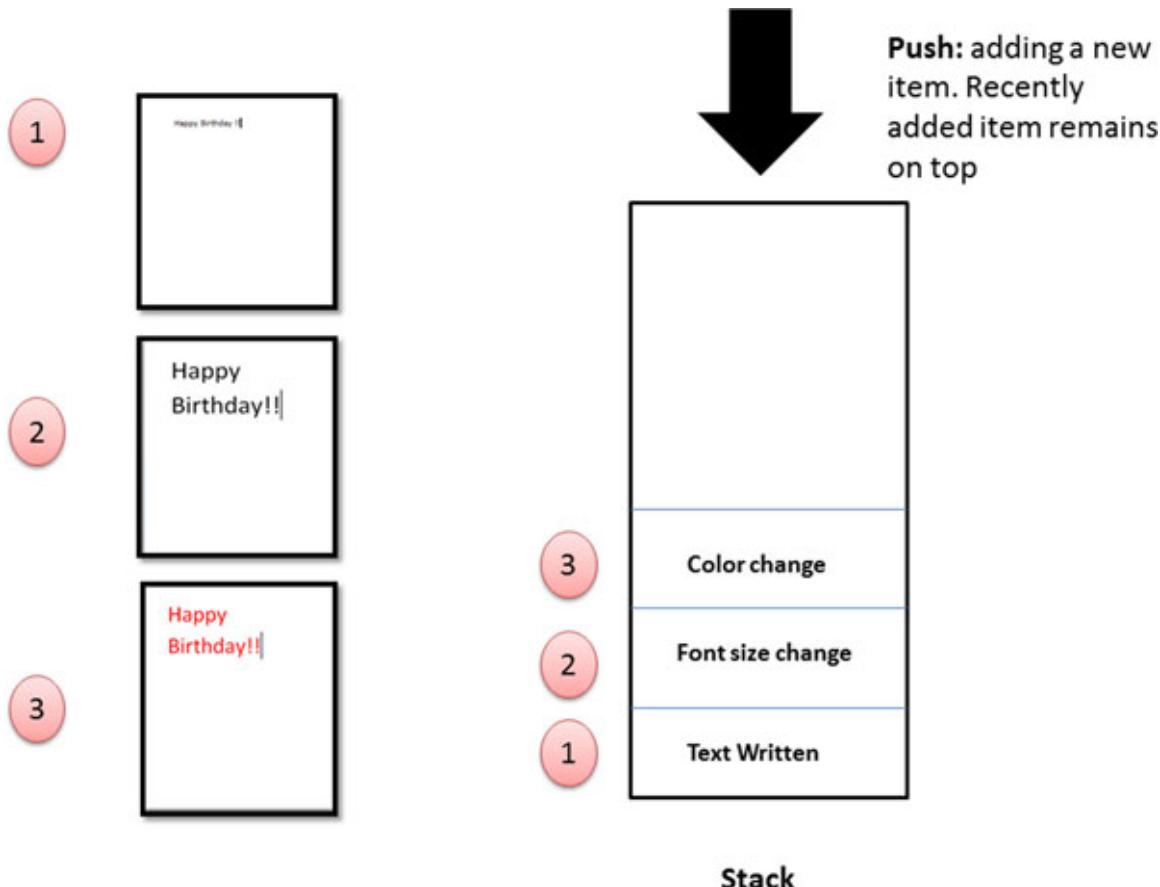
After reading this chapter, you will know what is stack, queue and deque, and how they can be implemented.

### 10.1 Stack

Stack is an ordered collection of items where the addition and removal of items take place at the same end, which is also known as the `TOP`. The other end of the stack is known as the `BASE`. The base of the stack is significant because the items that are closer to the base represent those that have been in the stack for the longest time.

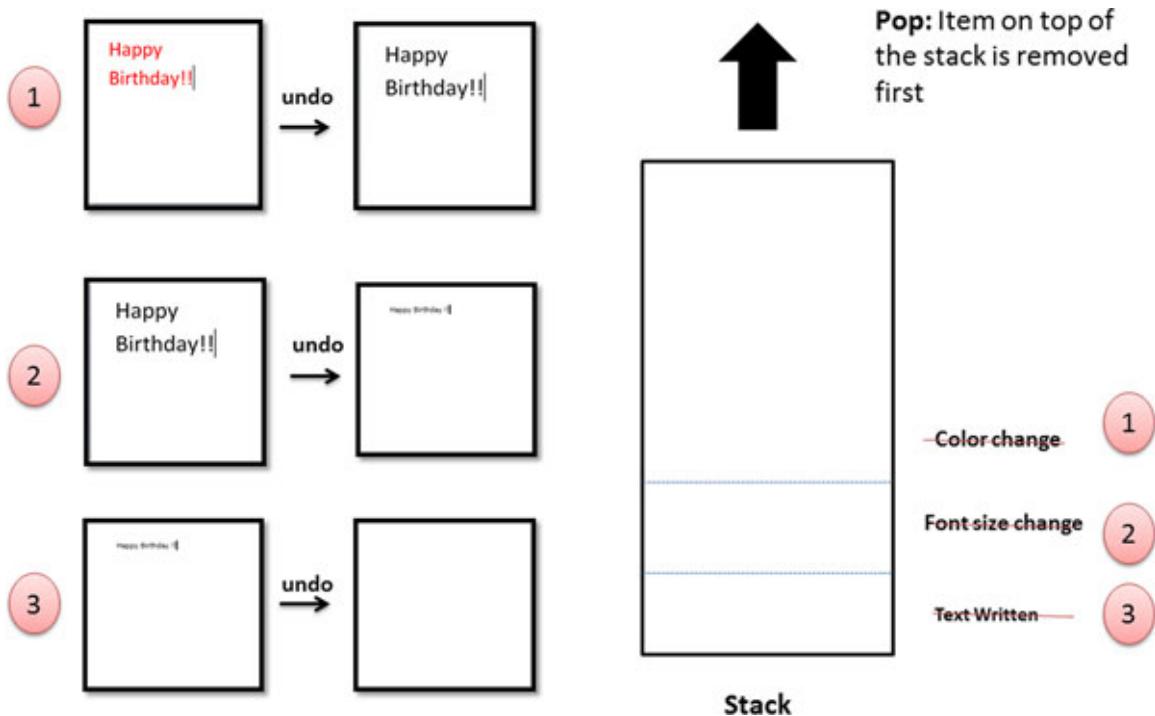
The most recently added item is the one that is in the top position so that it can be removed first. The items are added on to the stack using **push** operation and removed using **pop** operation.

To understand **push** and **pop**, let's take a look at a familiar situation. Suppose, you start working on a word document; you type in a title and then you change its Font size followed by change of color.



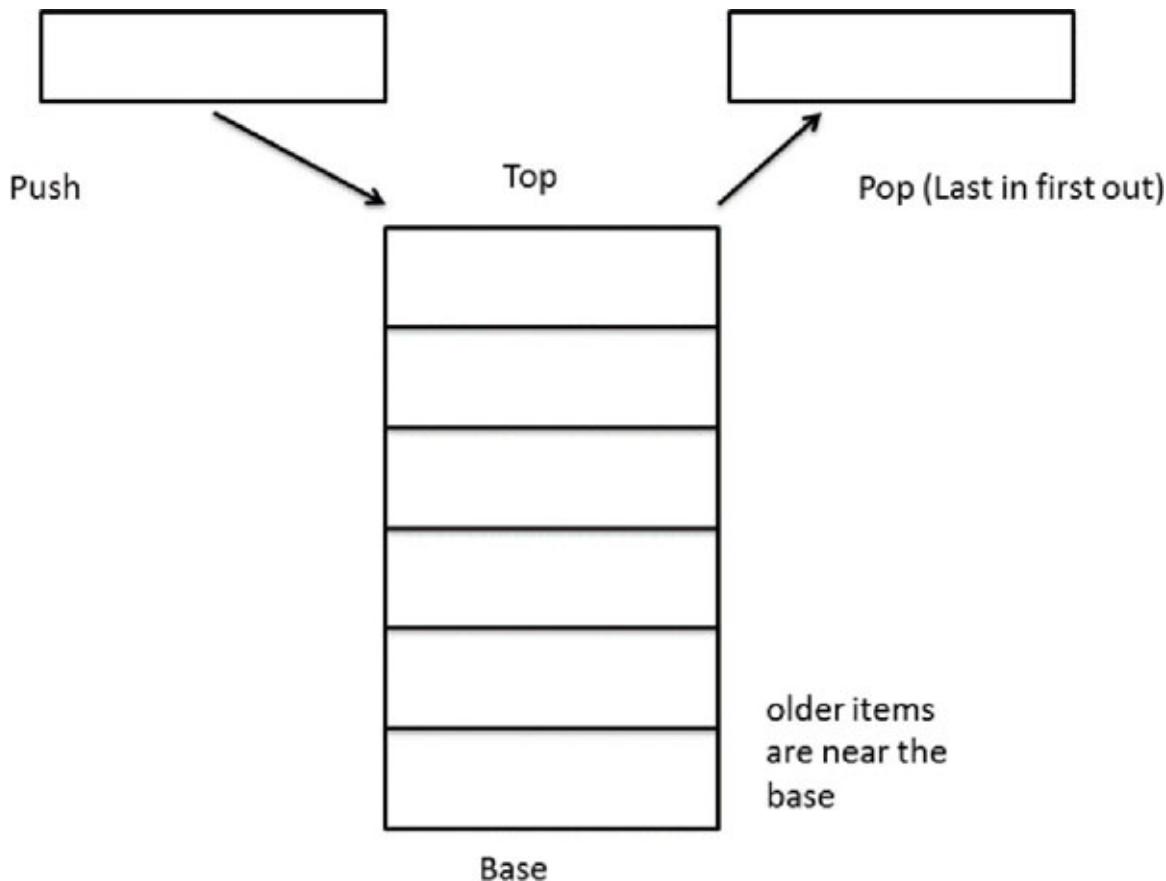
*Figure 10.1(a): Push operation*

So, you can do whatever you want to do on the word file but when you hit the undo button, only the last task done is undone.



*Figure 10.1(b): PopOperation*

Similarly, in case of stack you can add as many items you want using **push** but you can **pop** or remove only the object which is on the top which means last in is first to be removed. This principle of ordering is known as **LIFO**-that stands for **Last -in-first-out**, where the newer items are near the top, while older items are near the base.



*Figure 10.2*

**stacks** are important because they are required whenever there is a need to reverse the order of items as the order of removal is reverse of the order of insertion.

### Example:

- Pushing back button on browser while surfing on internet.
- *Ctrl+Z* (undo) in Microsoft applications.
- Remove recently used object from cache.

Stacks are commonly used by software programmers; it may not be quite noticeable while using a program as these operations generally take place at the background. However, many times you may come across **stack overflow error**. This happens when a stack actually runs out of memory.

Stacks seem to be very simple. Yet they are one of the most important data structures as you will see very soon. Stacks serve as a very important element of several data structures and algorithms.

## 10.1.1 Implementation of a stack in Python

Here we will implement Stack using lists.

### **Step 1:** Define the Stack Class

```
#Define Stack Class
class Stack:
```

### **Step 2:** Create constructor

Create a constructor that takes self and size(n) of the stack. In the method we declare that self.stack is an empty list ([]) and self.size is equal to n that is, the size provided.

```
#Define Stack Class
class Stack:
 Constructor
 def __init__(self, n):
 self.stack = []
 self.size = n
```

### **Step 3:** Define Push Function

A push function will have two arguments self and the element that we want to push in the list. In this function, we first check if the length of the stack is equal to the size provided as input(n). If yes, then it means that the stack is full and prints the message that *no more elements can be appended as the stack is full*. However, if that is not the case then we can call the append method to push the element to the stack.

```
#push method
def push(self,element):
 if len(self.stack)== self.size:
 print("no more elements can be appended as the stack is full")
 else:
 self.stack.append(element)
```

### **Step 4:** Define POP Function

Check the stack. If it is empty, then print: Stack is empty. Nothing to POP!!.. If it is not empty pop the last item from the stack.

```
#pop method
def pop(self):
 if(self.stack == []):
 print("Stack is empty. Nothing to POP!!")
```

```
 else:
 self.stack.pop()
```

Step 5: Write down the steps for executing the code

### Execution:

```
s = Stack(3)
s.push(6)
s.push(2)
print(s.stack)
s.pop()
print(s.stack)
```

The complete code will be as follows:

### Code:

```
#Define Stack Class
class Stack:

 #Constructor
 def __init__(self, n):
 self.stack = []
 self.size = n

 #push method
 def push(self, element):
 if len(self.stack) == self.size:
 print("no more elements can be appended as the stack is full")
 else:
 self.stack.append(element)

 #pop method
 def pop(self):
 if self.stack == []:
 print("Stack is empty. Nothing to POP!!")
 else:
 self.stack.pop()

s = Stack(3)
s.push(6)
s.push(2)
print(s.stack)
s.pop()
print(s.stack)
```

### Output:

```
[6, 2]
```

[6]  
">>>>

## Example 10.1

Write a program to check if a given string has balanced set of parenthesis or not.

Balanced parenthesis: (), {}, [], {{()}}, [], and so on.

### Answer:

Here we have to check if pairs of brackets exist in right format in a string. Expressions such as “[]{}()” are correct. However, if the opening bracket does not find the corresponding closing bracket then the parenthesis is a mismatch. For example: “[)” or “{)[]}”. To solve this problem we will follow following steps:

#### Step 1: Define class `paranthesis_match`

```
class parenthesis_match:
```

#### Step 2: Defining lists for opening and closing brackets

We now define two lists such that the index of an opening bracket matches with the index of the corresponding closing bracket:

1. List `opening_brackets` will have all types of opening brackets as elements as elements – [“(, ”{, ”[”]
2. List `closing_brackets` will have all types of closing brackets as elements – [”)”, ”}”, ”]”]

Here is how we define the lists:

```
class parenthesis_match:
 opening_brackets = ["(", "{", "["]
 closing_brackets = [")", "}", "]"]
```

#### Step 3: Defining Constructor, Push, and Pop functions

### Constructor:

The constructor takes expression as parameter which is the string provided for validation of parameters.

We also initialize list for stacking purposes. The `push()` and `pop()` functions will be applied on this list.

```
#declare constructor
def __init__(self, expression):
 self.expression = expression
 self.stack = []
```

Since we are implementing this using a stack it is quite obvious that we would require `push()` and `pop` functions.

The `push()` function when called, will add element to the stack.

```
#push operation
def push(self,element):
 self.stack.append(element)
```

The `pop()` element on the other hand will pop the last element from the stack.

```
#pop operation
def pop(self):
 if self.stack == []:
 print("Unbalanced Parenthesis")
 else:
 self.stack.pop()
```

#### Step 4: Defining the function to do the analysis

We will now write the code to analyse the string.

In this function, we will perform the following steps:

- First we check the length of the expression. A string of balanced parenthesis will always have even number of characters. If the length of the expression is divisible by two only then we would move ahead with the analysis. So, an `if...else` loop forms the outer structure of this function.

```
if len(self.expression)%2 == 0:
 ---- we analyse.....
else:
 print("Unbalanced Parenthesis")
```

- Now, considering that we have received a length of even number. We can move ahead with analysis and we can write our code in the “if” block. We will now traverse through the list element by element. If we encounter an opening bracket we will push it on to the stack, if it is not an opening bracket then we check if the element is in the closing bracket list. If yes then we pop the last element from the stack and see if the

index of the elements in `opening_brackets` and `closing_brackets` list is of same bracket. If yes, then there is a match else the list is unbalanced.

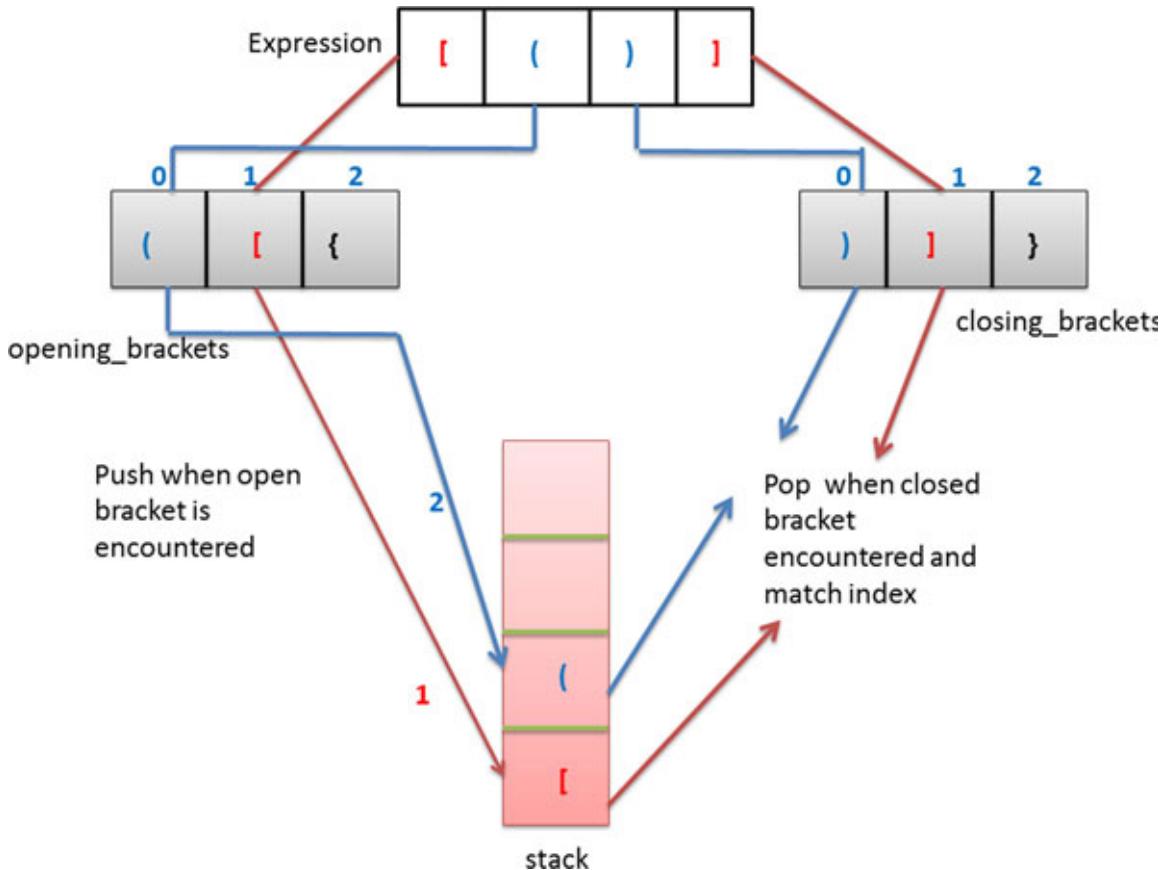


Figure 10.3

```
def is_match(self):

 print("expression is = ", self.expression)
 if len(self.expression)%2 == 0:
 for element in self.expression:
 print("evaluating ", element)
 if element in self.opening_brackets:
 print("it is an opening bracket - ", element, "pushing to stack")
 self.push(element)
 print("pushed", element, "on to stack the stack is ", self.stack)
 elif element in self.closing_brackets:
 x = self.stack.pop()
 print("time to pop element is ", x)
 if self.opening_brackets.index(x) == self.closing_brackets.index(element):
 print("Match Found")
 else:
 print("Match not found - check pranthesis")
 return;
 else:
 print("Unbalanced Paranthesis")
```

---

## Step 5: Write down the steps for executing the code

### Execution:

```
pm = paranthesis_match("([{}])")
pm.is_match()
```

So, the final code will be as follows, to make things easier for the end user, print commands have been added:

```
class paranthesis_match:
 opening_brackets = ["(", "{", "["]
 closing_brackets = [")", "}", "]"]

 #declare constructor
 def __init__(self, expression):
 self.expression = expression
 self.stack = []

 #push operation
 def push(self, element):
 self.stack.append(element)

 #pop operation
 def pop(self):
 if self.stack == []:
 print("Unbalanced Parenthesis")
 else:
 self.stack.pop()

 def is_match(self):

 print("expression is =", self.expression)
 if len(self.expression)%2 == 0:
 for element in self.expression:
 print("evaluating ", element)
 if element in self.opening_brackets:
 print("it is an opening bracket - ", element, "pushing to stack")
 self.push(element)
 print("pushed", element, " on to stack the stack is ", self.stack)
 elif element in self.closing_brackets:
 x = self.stack.pop()
 print("time to pop element is ", x)
 if self.opening_brackets.index(x) == self.closing_brackets.index(element):
 print("Match Found")
 else:
 print("Match not found - check parenthesis")
 return;
 else:
 print("Unbalanced Parenthesis")
```

```
pm = parenthesis_match("([{}])")
pm.is_match()
```

## Output:

```
expression is = ([{}])
evaluating (
it is an opening bracket - (pushing to stack
pushed (on to stack the stack is ['(']
evaluating [
it is an opening bracket - [pushing to stack
pushed [on to stack the stack is ['(', '[']
evaluating {
it is an opening bracket - {pushing to stack
pushed {on to stack the stack is ['(', '[', '{']
evaluating}
time to pop element is {
Match Found
evaluating]
time to pop element is [
Match Found
evaluating)
time to pop element is (
Match Found
```

## 10.2 Queue

A **queue** is a sequence of objects where elements are added from one end and removed from the other. The queues follow the principle of **first in first out**. The removal of items is done from one end called the **front** and the items are removed from another end that's referred to as **rear**. So, just as in case of any queue in real life, the items enter the queue from the rear and start moving towards the front as the items are removed one by one.

So, in **queue** the item at the front is the one that has been in the sequence for the longest and the most recently added item must wait at the end. The **insert** and **delete** operations are also called **enqueue** and **dequeue**.

### 10.2.1 Basic queue functions

Basic Queue functions are as follows:

- **enqueue(i)**: Add element “i” to the queue.
- **dequeue()**: Removes the first element from the queue and returns its value.

- **isEmpty()**: Boolean function that returns “true” if the queue is empty else it will return false.
- **size()**: Returns length of the queue.



*Figure 10.4*

## 10.2.2 Implementation of Queue.

Let's write a program for implementation of queue.

**Step 1:** Define the class

```
class Queue:
```

**Step 2:** Define the constructor

Here, we initialize an empty list queue:

```
def __init__(self):
 self.queue = []
```

**Step 3:** Define **isEmpty()** function

As the name suggests, this method is called to check if the queue is empty or not. **The isEmpty()** function checks the queue. If it is empty, it prints a message - *Queue is Empty* or else it will print a message - *Queue is not Empty*

```
def isEmpty(self):
 if self.queue == []:
 print("Queue is Empty")
 else:
 print("Queue is not Empty")
```

**Step 4:** Define the **enqueue()** function

This function takes an element as parameter and inserts it at index “0”. All elements in the queue shift by one position.

```
def enqueue(self,element):
 self.queue.insert(0,element)
```

## **Step 5:** Define the `enqueue()` function

This function pops the oldest element from the queue.

```
def dequeue(self):
 self.queue.pop()
```

## **Step 6:** Define the `size()` function

This function returns the length of the queue.

```
def size(self):
 print("size of queue is",len(self.queue))
```

## **Step 7:** Write down the commands to execute the code

### **Code Execution**

```
#Code Execution
q = Queue()
q.isEmpty()
inserting elements
print("inserting element no.1")
q.enqueue("apple")
print("inserting element no.2")
q.enqueue("banana")
print("inserting element no.3")
q.enqueue("orange")
print("The queue elements are as follows:")
print(q.queue)
print("check if queue is empty?")
q.isEmpty()
removing elements
print("remove first element")
q.dequeue()
print("what is the size of the queue?")
q.size()
print("print contents of the queue")
print(q.queue)
```

### **Code:**

```
class Queue:
 def __init__(self):
 self.queue = []

 def isEmpty(self):
 if self.queue == []:
 print("Queue is Empty")

 else:
```

```

 print("Queue is not empty")

def enqueue(self,element):
 self.queue.insert(0,element)

def dequeue(self):
 self.queue.pop()

def size(self):
 print("size of queue is",len(self.queue))
#Code Execution
q = Queue()
q.isEmpty()
inserting elements
print("inserting element no.1")
q.enqueue("apple")
print("inserting element no.2")
q.enqueue("banana")
print("inserting element no.3")
q.enqueue("orange")
print("The queue elements are as follows:")
print(q.queue)
print("check if queue is empty?")
q.isEmpty()
removing elements
print("remove first element")
q.dequeue()
print("what is the size of the queue?")
q.size()
print("print contents of the queue")
print(q.queue)

```

## Output:

```

Queue is Empty
inserting element no.1
inserting element no.2
inserting element no.3
The queue elements are as follows:
['orange', 'banana', 'apple']
check if queue is empty?
Queue is not empty
remove first element
what is the size of the queue?
size of queue is 2
print contents of the queue
['orange', 'banana']

```

## 10.2.3 Implementation of a stack using single queue

Before implementing the code, it is important to understand the logic behind it. The question demands that you make a queue work like a stack. A queue works on the principle of First-In-First-Out whereas a stack works on the principle of Last In, First Out.

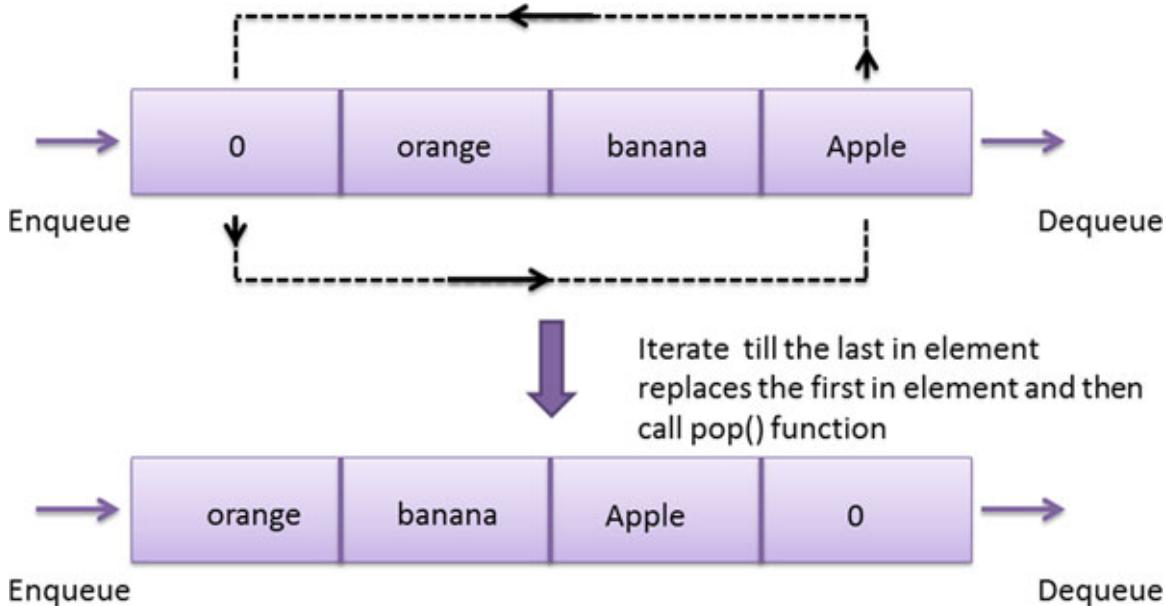


Figure 10.5

## Code:

```
class Stack_from_Queue:
 def __init__(self):
 self.queue = []
 def isEmpty(self):
 if self.queue == []:
 print("Queue is Empty")
 else:
 print("Queue is not empty")
 def enqueue(self,element):
 self.queue.insert(0,element)
 def dequeue(self):
 return self.queue.pop()
 def size(self):
 print("size of queue is",len(self.queue))
 def pop(self):
 for i in range(len(self.queue)-1):
 x = self.dequeue()
 print(x)
```

```
 self.enqueue(x)
 print("element removed is",self.dequeue())
```

## Execution – I

```
sq = Stack_from_Queue()
sq.isEmpty()
print("inserting element apple")
sq.enqueue("apple")
print("inserting element banana")
sq.enqueue("banana")
print("inserting element orange")
sq.enqueue("orange")
print("inserting element 0")
sq.enqueue("0")
print("The queue elements are as follows:")
print(sq.queue)
print("check if queue is empty?")
sq.isEmpty()
print("remove the last in element")
sq.pop()
sq.pop()
sq.pop()
sq.pop()
sq.isEmpty()
```

## Output – I

```
Queue is Empty
inserting element apple
inserting element banana
inserting element orange
inserting element 0
The queue elements are as follows:
['0', 'orange', 'banana', 'apple']
check if queue is empty?
Queue is not empty
remove the last in element
apple
banana
orange
element removed is 0
apple
banana
element removed is orange
apple
element removed is banana
element removed is apple
Queue is Empty
>>>
```

## Execution – II

```
sq = Stack_from_Queue()
sq.isEmpty()
print("inserting element apple")
sq.enqueue("apple")
print("inserting element banana")
sq.enqueue("banana")
print("inserting element orange")
sq.enqueue("orange")
print("inserting element 0")
sq.enqueue("0")
for i in range(len(sq.queue)):
 print("The queue elements are as follows:")
 print(sq.queue)
 sq.pop()
 print("check if queue is empty?")
 sq.isEmpty()
 print("remove the last in element")
 print(sq.queue)
```

## Output –II

```
inserting element apple
inserting element banana
inserting element orange
inserting element 0
The queue elements are as follows:
['0', 'orange', 'banana', 'apple']
apple
banana
orange
element removed is 0
check if queue is empty?
Queue is not empty
remove the last in element
['orange', 'banana', 'apple']
The queue elements are as follows:
['orange', 'banana', 'apple']
apple
banana
element removed is orange
check if queue is empty?
Queue is not empty
remove the last in element
['banana', 'apple']
The queue elements are as follows:
['banana', 'apple']
apple
element removed is banana
check if queue is empty?
```

```

Queue is not empty
remove the last in element
['apple']
The queue elements are as follows:
['apple']
element removed is apple
check if queue is empty?
Queue is Empty
remove the last in element
[]

```

## 10.2.4 Implementation of a queue using two stacks

Let's see how to implement a queue using two stacks.

### Step 1:

Create a basic `Stack()` class with `push()`, `pop()`, and `isEmpty()` functions.

```

class Stack:
 def __init__(self):
 self.stack = []

 def push(self,element):
 self.stack.append(element)

 def pop(self):
 return self.stack.pop()

 def isEmpty(self):
 return self.stack == []

```

### Step 2: Define the Queue class

```
class Queue:
```

### Step 3: Define the constructor

Since the requirement here is of two stacks, we initialize two stack objects.

```

def __init__(self):
 self.inputStack = Stack()
 self.outputStack = Stack()

```

### Step 4: Define the enqueue function

This function will push the elements into the first stack.

```

def enqueue(self,element):
 self.inputStack.push(element)

```

---

### **Step 5:** Define the `dequeue()` function

This function checks whether the output stack is empty. If it is empty, then elements will be popped out from `inputStack` one by one and pushed into the `outputStack`, so that the last in element is the first one to be out. However, if the output stack is not empty, then the elements can be popped directly from it.

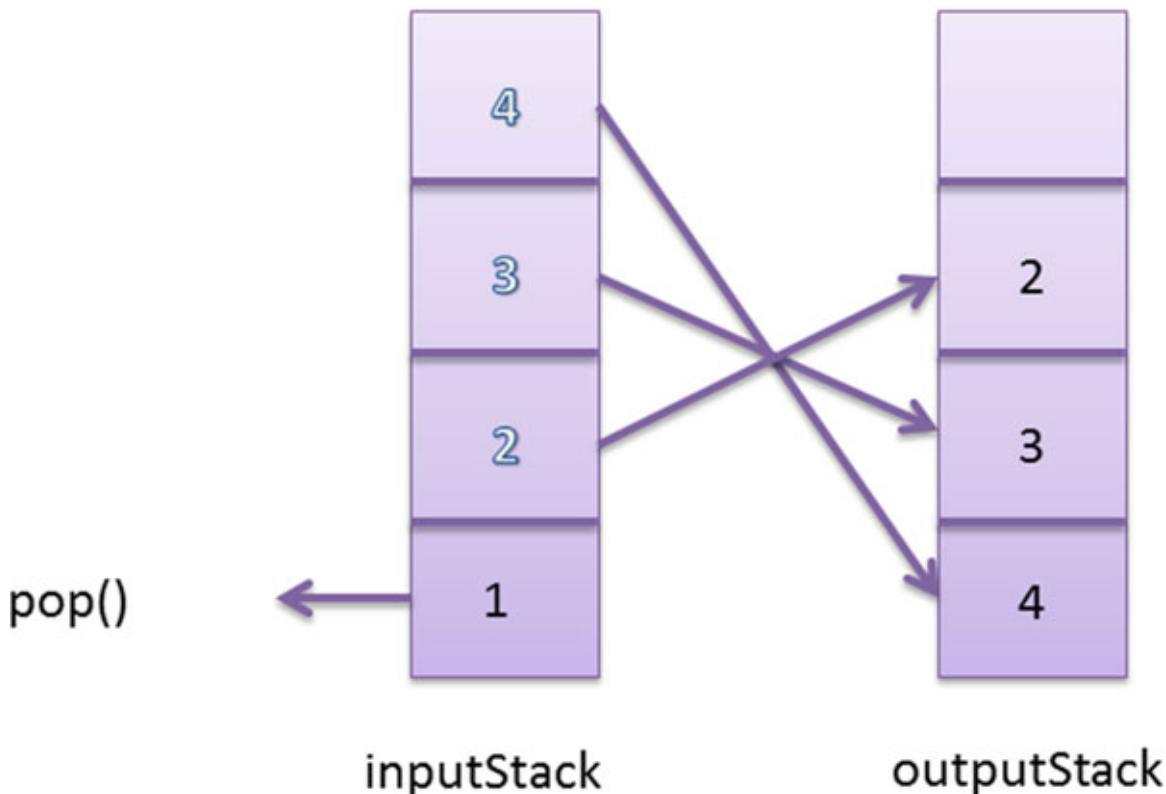
Now, suppose we insert 4 values: 1,2,3,4 calling the enqueue function. Then, the input stack would be like this:



inputStack

*Figure 10.6*

When we call a `dequeue` function, the elements from `inputStack` are popped and pushed one by one on to the output stack till we reach the last element and that last element is popped from the `inputStack` and returned. If the output stack is not empty then it means that it already has elements in the right order and they can be popped in that order.



*Figure 10.7*

```
def dequeue(self):
 #if not self.inputStack.isEmpty():
 if self.outputStack.isEmpty():
 for i in range(len(self.inputStack.stack)-1):
 x = self.inputStack.pop()
 self.outputStack.push(x)
 print("popping out value =", self.inputStack.pop())
 else:
 print("popping out value =", self.outputStack.pop())
```

### Code:

```
class Queue:
 def __init__(self):
 self.inputStack = Stack()
```

```

 self.outputStack = Stack()

 def enqueue(self,element):
 self.inputStack.push(element)

 def dequeue(self):
 if self.outputStack.isEmpty():
 for i in range(len(self.inputStack.stack)-1):
 x = self.inputStack.pop()
 self.outputStack.push(x)
 print("popping out value =", self.inputStack.pop())
 else:
 print("popping out value =", self.outputStack.pop())
#Define Stack Class
class Stack:
 def __init__(self):
 self.stack = []

 def push(self,element):
 self.stack.append(element)

 def pop(self):
 return self.stack.pop()

 def isEmpty(self):
 return self.stack == []

```

## Execution:

```

Q = Queue()
print("insert value 1")
Q.enqueue(1)
print("insert value 2")
Q.enqueue(2)
print("insert value 3")
Q.enqueue(3)
print("insert value 4")
Q.enqueue(4)
print("dequeue operation")
Q.dequeue()
Q.dequeue()
print("insert value 7")
Q.enqueue(7)
Q.enqueue(8)
Q.dequeue()
Q.dequeue()
Q.dequeue()
Q.dequeue()

```

## Output

```
insert value 1
```

```

insert value 2
insert value 3
insert value 4
dequeue operation
popping out value = 1
popping out value = 2
insert value 7
popping out value = 3
popping out value = 4
popping out value = 7
popping out value = 8

```

## 10.3 Deque

A **deque** is more like a **queue** only that it is double ended. It has items positioned in ordered collection and has two ends, the front and the rear. A deque is more flexible in nature, in the sense that the elements can be added or removed from front or rear. So, you get the qualities of both stack and queue in this one linear data structure.

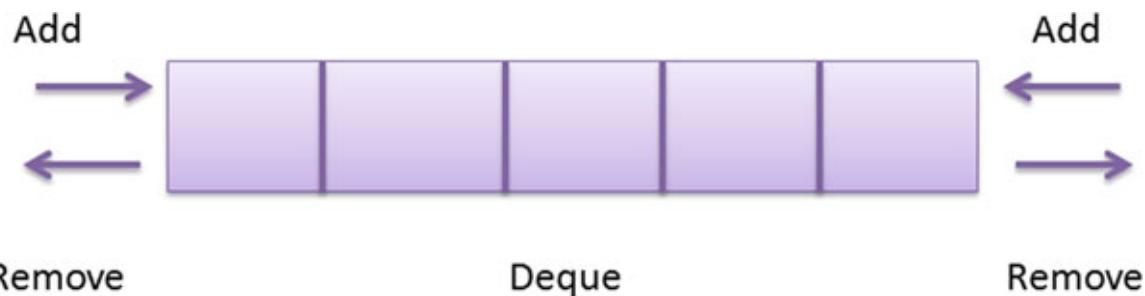


Figure 10.8

### 10.3.1 Write a code to implement a deque

Implementation of deque is easy. If you have to add an element from rear, you will have to add it at `index 0` same way. If you have to add it from the front, call the `append()` function. Similarly, if you have to remove from front, call the `pop()` function and if you want to pop it from the rear, call `pop(0)`.

**Code:**

```

class Deque:
 def __init__(self):
 self.deque = []

 def addFront(self,element):
 self.deque.append(element)

```

```

 print("After adding from front the deque value is : ", self.deque)

 def addRear(self,element):
 self.deque.insert(0,element)
 print("After adding from end the deque value is : ", self.deque)

 def removeFront(self):
 self.deque.pop()
 print("After removing from the front the deque value is : ", self.deque)

 def removeRear(self):
 self.deque.pop(0)
 print("After removing from the end the deque value is : ", self.deque)

```

## Execution:

```

d = Deque()
print("Adding from front")
d.addFront(1)
print("Adding from front")
d.addFront(2)
print("Adding from Rear")
d.addRear(3)
print("Adding from Rear")
d.addRear(4)
print("Removing from Front")
d.removeFront()
print("Removing from Rear")
d.removeRear()

```

## Output:

```

After adding from front the deque value is : [1]
After adding from front the deque value is : [1, 2]
After adding from end the deque value is : [3, 1, 2]
After adding from end the deque value is : [4, 3, 1, 2]
After removing from the front the deque value is : [4, 3, 1]
After removing from the end the deque value is : [3, 1]

```

# CHAPTER 11

## Linked List

### Introduction

In this chapter, you will learn how to work with linear data structure - Linked Lists using Python.

### Structure

- Introduction to Linked List
  - What is a Linked list
  - Structure of a linked list
- Implementation of Node Class
  - Traversing through linked list
  - How to add a node at the beginning of a linked list
  - How to add a node at the end of a linked list
  - Inserting a node between two nodes in a linked list
  - Removing a node from a linked list
  - Implementation of doubly linked list
  - Reversing a linked list

### Objective

After reading this chapter, you will have thorough knowledge about Linked list, and you will know how to work with it.

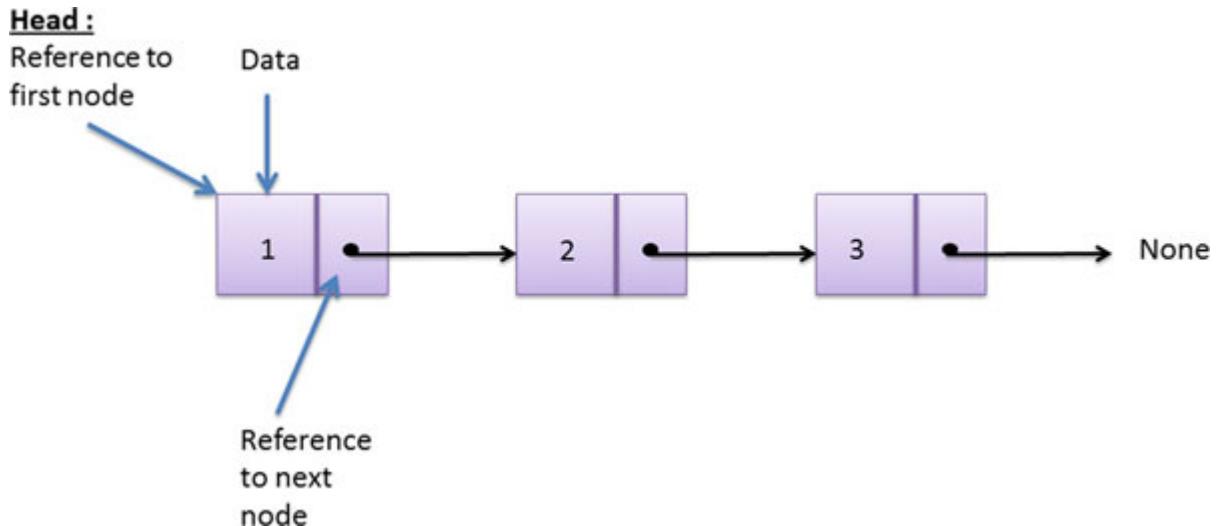
### 11.1 Introduction to Linked Lists

The linked list is a linear structure that consists of elements such that each element is an individual object and contains information regarding:

1. Data

2. Reference to the next element

In linked list, each element is called a **node**.

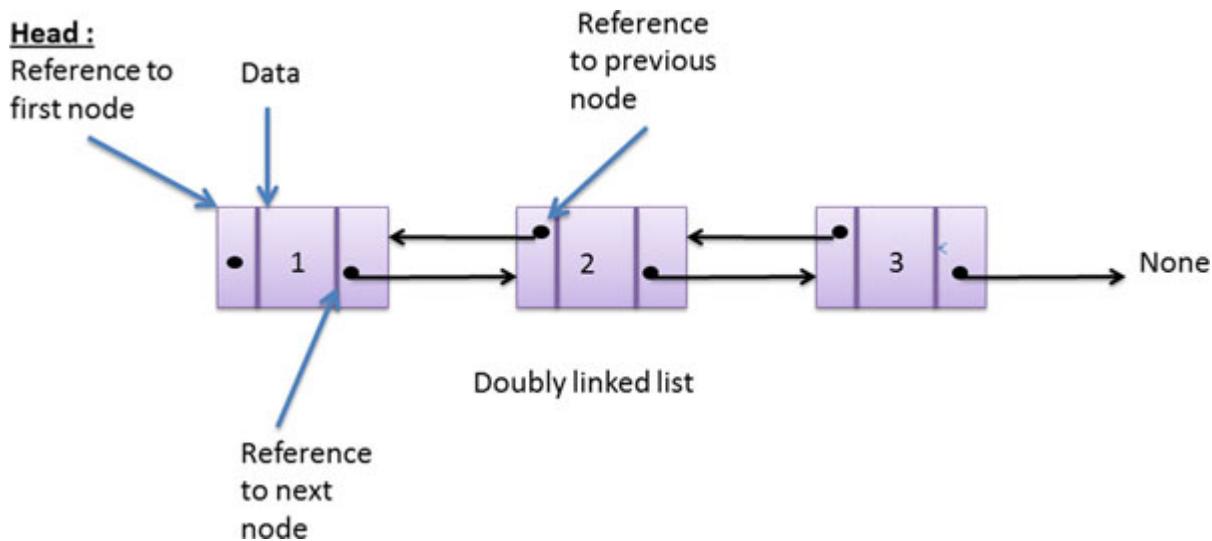


*Figure 11.1*

You can see in the diagram, the reference to the first node is called *Head*. It is the entry point of the linked list. If the list is empty, then the head points to null. The last node of the linked list refers to null.

As the number of nodes can be increased or decreased as per requirement, linked lists are considered to be dynamic data structure. However, in a linked list, direct access to data is not possible. Search for any item starts from the head, and you will have to go through each reference to get that item. A linked list occupies more memory.

The linked list described above is called a *singly linked list*. There is one more type of linked list known as *doubly linked list*. A double linked list has the reference to the next node and the previous node.



*Figure 11.2*

## 11.2 Implementation of Node class

In this section, we will learn how to implement a `Node` class. A `Node` contains:

1. Data
2. Reference to the next node

**Process involved:** To create a node object, we pass data value to the constructor. The constructor assigns the data value to data, and sets node's reference to `None`. Once we create all the objects, we assign memory address of the second object as the reference to the first node object, memory address of third object is assigned as reference to second object, and so on. The last object, thus have no (or none as) reference.

The code for the `Node` class will be as follows:

### **Code:**

```
class Node:
 def __init__(self, data = None):
 self.data = data
 self.reference = None
```

### **Execution:**

```
objNode1 = Node(1)
```

```

objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
objNode1.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
objNode4.reference = None

print("DATA VALUE = ",objNode1.data,"REFERENCE = ",objNode1.reference)
print("DATA VALUE = ",objNode2.data,"REFERENCE = ",objNode2.reference)
print("DATA VALUE = ",objNode3.data,"REFERENCE = ",objNode3.reference)
print("DATA VALUE = ",objNode4.data,"REFERENCE = ",objNode4.reference)

```

## Final Code:

```

class Node:
 def __init__(self,data = None):
 self.data = data
 self.reference = None
#Execution
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
objNode1.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
objNode4.reference = None

print("DATA VALUE = ",objNode1.data,"REFERENCE = ",objNode1.reference)
print("DATA VALUE = ",objNode2.data,"REFERENCE = ",objNode2.reference)
print("DATA VALUE = ",objNode3.data,"REFERENCE = ",objNode3.reference)
print("DATA VALUE = ",objNode4.data,"REFERENCE = ",objNode4.reference)

```

## Output:

```

DATA VALUE = 1 REFERENCE = <__main__.Node object at 0x0284B490>
DATA VALUE = 2 REFERENCE = <__main__.Node object at 0x0284B448>
DATA VALUE = 3 REFERENCE = <__main__.Node object at 0x0284B3A0>
DATA VALUE = 4 REFERENCE = None
>>>

```

## 11.2.1 Traversing through a linked list

Let's see how we can traverse through a linked list.

### Method I

We have already written the code for the `Node` class:

```
class Node:
 def __init__(self,data = None):
 self.data = data
 self.reference = None

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
```

We will now see how to traverse through the linked list:

## Step 1

We create a variable `presentNode` and assign the first object to it.

```
presentNode = objNode1
```

On doing this, `presentNode` gets the data, and reference values of `objNode1`.

## Step 2

The reference value points to `objNode2`.

So, we can write a while loop:

```
while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.reference
```

Once the `presentNode` is assigned, the reference value contained in `objNode4`. It will come out of the while loop because the value of reference is **None**.

## Code:

```
class Node:
 def __init__(self,data = None):
 self.data = data
 self.reference = None
```

## Execution:

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
objNode1.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
objNode4.reference = None
presentNode = objNode1
while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.reference

```

## Final Code

```

class Node:
 def __init__(self,data = None):
 self.data = data
 self.reference = None
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
objNode1.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
objNode4.reference = None
presentNode = objNode1
while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.reference

```

## Output:

```

DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4

```

## Method II

Another method to do this by creating two classes: Node and Linked list  
Code

```

class Node:
 def __init__(self,data = None):
 self.data = data

```

```

 self.reference = None
class Linked_list:
 def __init__(self):
 self.head = None
 def traverse(self):
 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.reference

```

## Execution:

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.traverse()

```

## Output:

```

DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4

```

## 11.2.2 How to add a node at the beginning of a linked list

In order to add a node at the beginning of a Linked list, we will just add a new method to insert the node in the same code that is mentioned in the last example.

In the last example, we pointed the head of the linked list object to the first node object.

```
linkObj.head = objNode1
```

When we add the node at the beginning, we just have to make the `linkObj.head = new_node` and `new node.reference = obj_Node1`.

For this, we write a code where, the value of the `linkObj.head` is first passed on to `new node.reference` and then `linkObj.head` is set to the new node object.

```
def insert_at_Beginning(self,data):
 new_data = Node(data)
 new_data.reference = self.head
 self.head = new_data
```

So, the full code would be as follows:

### Code:

```
class Node:

 def __init__(self,data = None):
 self.data = data
 self.reference = None

class Linked_list:
 def __init__(self):
 self.head = None
 def traverse(self):
 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.reference
 def insert_at_Beginning(self,data):
 new_data = Node(data)
 new_data.reference = self.head
 self.head = new_data
```

### Execution:

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()

#head of the linked list to first object
linkObj.head = objNode1
reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
```

```
linkObj.insert_at_Beginning(5)
linkObj.traverse()
```

## Output:

```
DATA VALUE = 5
DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4
```

### 11.2.3 How to add a Node at the end of a linked list

In order to add a node at the end of the linked list, it is important that you point the reference of the last node to the new node that you create.

#### Step 1: Define the function

```
def insert_at_end(self,data):
```

#### Step 2: Create a new `node` object

```
new_data = Node(data)
```

#### Step 3: Traverse through the linked list to reach the last node

Remember that you cannot directly access the last node in the linked list. You will have to traverse through all the nodes and reach the last node in order to take the next step.

```
presentNode = self.head
while presentNode.reference != None:
 presentNode = presentNode.reference
```

#### Step 4: Add the new node at the end

After traversing through the linked list, you know that you have reached the last node when `presentNode.reference = None`. Since this won't remain, the last node anymore, you need to do the following:

```
presentNode.reference = new_data
```

With this we have added a new node at the end of a linked list.

### Code:

```
class Node:

 def __init__(self,data = None):
 self.data = data
 self.reference = None

class Linked_list:

 def __init__(self):
 self.head = None

 def traverse(self):
 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)

 presentNode = presentNode.reference

 def insert_at_end(self,data):
 new_data = Node(data)
 presentNode = self.head
 while presentNode.reference != None:
 presentNode = presentNode.reference
 presentNode.reference = new_data
```

### Execution:

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()

#head of the linked list to first object
linkObj.head = objNode1

reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4

linkObj.insert_at_end(5)
linkObj.insert_at_end(6)
linkObj.insert_at_end(7)
linkObj.traverse()
```

### Output:

```
DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4
DATA VALUE = 5
DATA VALUE = 6
DATA VALUE = 7
```

## 11.2.4 Inserting a node between two nodes in a linked list

The solution for this problem is very similar to adding a node to the beginning. The only difference is that when we add a node at the beginning we point the head value to the new node, whereas in this case, the function will take two parameters. First will be the node object, after which the new object will be inserted, and second would be the data for the new object. Once the new node is created, we pass on the reference value stored in the existing node object to it and the existing node is then made to point at the new node object.

### **Step 1:** Define the function

This function will take two parameters:

1. The node object after which the data is to be inserted.
2. Data for the new node object.

```
def insert_in_middle(self,insert_data,new_data):
```

### **Step 2:** Assign references

```
new_node = Node(new_data)
new_node.reference = insert_data.reference
insert_data.reference = new_node
```

### **Code:**

```
class Node:
 def __init__(self,data = None):
 self.data = data
 self.reference = None
```

```

class Linked_list:
 def __init__(self):
 self.head = None

 def traverse(self):
 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.reference

 def insert_in_middle(self,insert_data,new_data):
 new_node = Node(new_data)
 new_node.reference = insert_data.reference
 insert_data.reference = new_node

```

## Execution:

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.insert_in_middle(objNode3,8)
linkObj.traverse()

```

## Output:

```

DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 8
DATA VALUE = 4
>>>

```

## 11.2.5 Removing a node from a linked list

Suppose, we have a linked list as follows:

A -> B -> C

A. reference = B

B. reference = C

C. reference = A.

To remove B, we traverse through the linked list. When we reach node A that has reference pointing to B, we replace that value with the reference value stored in B (that points to C). So that will make A point to C and B is removed from the chain.

The function code will be as follows:

```
def remove(self,removeObj):
 presentNode = self.head
 while presentNode:
 if(presentNode.reference == removeObj):
 presentNode.reference = removeObj.reference
 presentNode = presentNode.reference
```

The function takes the `Node` object as parameter and traverses through the linked list, till it reaches the object that needs to be removed. Once we reach the node that has reference to the node that has to be removed, we simply change the reference value to reference value stored in object `removeObj`. Thus, the node now points directly to the node after the `removeObj`.

## Code:

```
class Node:
 def __init__(self,data = None):
 self.data = data
 self.reference = None

class Linked_list:
 def __init__(self):
 self.head = None
 def traverse(self):
 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.reference

 def remove(self,removeObj):
 presentNode = self.head
 while presentNode:
 if(presentNode.reference == removeObj):
 presentNode.reference = removeObj.reference
 presentNode = presentNode.reference
```

## Execution:

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.remove(objNode2)
linkObj.traverse()
```

## Output:

```
DATA VALUE = 1
DATA VALUE = 3
DATA VALUE = 4
>>>
```

## 11.2.6 Printing the values of the node in the centre of a linked list

Printing the values of the node in the centre of a linked list involves counting the number of nodes in the object. If the length is even then the data of two nodes in the middle should be printed else only the data of node in the centre should be printed.

### Step 1: Define the function

```
def find_middle(self,llist):
```

### Step 2: Find the length of the counter

Here, we set a variable **counter = 0**. As we traverse through the linked list we increment the counter. At the end of the while loop we get the count of number of nodes in the linked list. This is also the length of the linked list.

```
counter = 0
presentNode = self.head
```

```

while presentNode:
 presentNode = presentNode.reference
 counter = counter + 1

print("size of linked list = ",counter)

```

### Step 3: Reach the middle of the linked list

The reference to the node in the middle is stored in the node before that. So, in the for loop instead of iterating ( $counter/2$ ) times, we iterate  $(counter-1)/2$  times. This brings us to the node which is placed just before the centre value.

```

presentNode = self.head
for i in range((counter-1)//2):
 presentNode = presentNode.reference

```

### Step 4: Display the result depending on whether the number of nodes in the linked list

If the linked list has even number of nodes then print the value of reference stored in the present node and the next node.

```

if (counter%2 == 0):
 nextNode = presentNode.reference
 print("Since the length of linked list is an even number the two middle
elements are:")
 print(presentNode.data,nextNode.data)

```

Else, print the value of the present node.

```

else:
 print("Since the length of the linked list is an odd number, the middle
element is: ")
 print(presentNode.data)

```

## Code:

```

class Node:
 def __init__(self,data = None):
 self.data = data
 self.reference = None
class Linked_list:

 def __init__(self):
 self.head = None

```

```

def find_middle(self,llist):
 counter = 0
 presentNode = self.head
 while presentNode:
 presentNode = presentNode.reference
 counter = counter + 1
 print("size of linked list = ",counter)
 presentNode = self.head

 for i in range((counter-1)//2):
 presentNode = presentNode.reference
 if (counter%2 == 0):
 nextNode = presentNode.reference
 print("Since the length of linked list is an even number the two middle
elements are:")
 print(presentNode.data,nextNode.data)
 else:
 print("Since the length of the linked list is an odd number, the middle
element is: ")
 print(presentNode.data)

```

## Execution (odd number of nodes)

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
objNode5 = Node(5)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
objNode4.reference = objNode5
linkObj.find_middle(linkObj)

```

## Output:

```

size of linked list = 5

Since the length of the linked list is an odd number, the middle element is:

3

```

## Execution (even numbers)

```

objNode1 = Node(1)

```

```

objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
reference of the first node object to second object
linkObj.head.reference = objNode2
objNode2.reference = objNode3
objNode3.reference = objNode4
linkObj.find_middle(linkObj)

```

## Output:

```

size of linked list = 4
Since the length of linked list is an even number the two middle elements are:
2 3

```

## 11.2.7 Implementation of doubly linked list

A doubly linked list has three parts:

1. Pointer to the previous node
2. Data
3. Pointer to the next node

Implementation of doubly linked list is easy. We just need to take care of one thing that each node is connected to the next and the previous data.

### Step 1: Create a Node class

The node class will have a constructor that initializes three parameters: data, reference to next node – **refNext** and reference to previous node – **refPrev**.

```

class Node:
 def __init__(self,data = None):
 self.data = data
 self.refNext = None
 self.refPrev = None

```

### Step 2: Create functions to traverse through the double linked list

#### I. Traverse Forward

To traverse forward with the help of `refNext` that points to next value of the linked list. We start with the head and move on to the next node using `refNext`.

```
def traverse(self):
 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.refNext
```

## II. Traverse Reverse

Traverse reverse is opposite of traverse forward. We are able to traverse backwards with the help of value of `refPrev` because it points to previous node. We start from the tail and move on to the previous node using `refPrev`.

```
def traverseReverse(self):
 presentNode = self.tail
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.refPrev
```

### Step 3: Write a function to add a node at the end

Appending a node at the end of the doubly linked list is same as appending in linked list the only difference is that we have to ensure that the node that is appended has its `refPrev` pointing to the node after which it has been added.

```
def append(self,data):
 new_data = Node(data)
 presentNode = self.head
 while presentNode.refNext != None:
 presentNode = presentNode.refNext
 presentNode.refNext = new_data
 new_data.refPrev = presentNode
```

### Step 4: Write function to remove a node

This function takes the node object that needs to be removed as parameter. In order to remove a node we iterate through the doubly linked list twice. We first start with the head and move forward using `refNext` and when we

encounter the object that needs to be removed we change the `refNext` value of the present node (which is presently pointing to the object that needs to be removed) to the node that comes after the object that needs to be removed. We then traverse through the linked list backwards starting from tail and when we encounter the object to be removed again we change the `refPrev` value of the present node to the node that is placed before it.

```
def remove(self,removeObj):
 presentNode = self.head
 presentNodeTail = self.tail
 while presentNode.refNext != None:
 if(presentNode.refNext == removeObj):
 presentNode.refNext = removeObj.refNext
 presentNode = presentNode.refNext
 while presentNodeTail.refPrev != None:
 if(presentNodeTail.refPrev == removeObj):
 presentNodeTail.refPrev = removeObj.refPrev
 presentNodeTail = presentNodeTail.refPrev
```

## Code

```
class Node:
 def __init__(self,data = None):
 self.data = data
 self.refNext = None
 self.refPrev = None
class dLinked_list:
 def __init__(self):
 self.head = None
 self.tail = None

 def append(self,data):
 new_data = Node(data)
 presentNode = self.head
 while presentNode.refNext != None:
 presentNode = presentNode.refNext
 presentNode.refNext = new_data
 new_data.refPrev = presentNode
 self.tail = new_data

 def traverse(self):
 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.refNext

 def traverseReverse(self):
 presentNode = self.tail
```

```

while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.refPrev

def remove(self,removeObj):
 presentNode = self.head
 presentNodeTail = self.tail
 while presentNode.refNext != None:
 if(presentNode.refNext == removeObj):
 presentNode.refNext = removeObj.refNext
 presentNode = presentNode.refNext
 while presentNodeTail.refPrev != None:
 if(presentNodeTail.refPrev == removeObj):
 presentNodeTail.refPrev = removeObj.refPrev
 presentNodeTail = presentNodeTail.refPrev

```

## Execution:

```

objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
dlinkObj = dLinked_list()
#head of the linked list to first object
dlinkObj.head = objNode1
dlinkObj.tail = objNode4
reference of the first node object to second object
dlinkObj.head.refNext = objNode2
dlinkObj.tail.refPrev = objNode3
objNode2.refNext = objNode3
objNode3.refNext = objNode4
objNode4.refPrev = objNode3
objNode3.refPrev = objNode2
objNode2.refPrev = objNode1
print("Appending Values")
dlinkObj.append(8)
dlinkObj.append(9)
print("traversing forward after Append")
dlinkObj.traverse()
print("traversing reverse after Append")
dlinkObj.traverseReverse()
print("Removing Values")
dlinkObj.remove(objNode2)
print("traversing forward after Remove")
dlinkObj.traverse()
print("traversing reverse after Remove")
dlinkObj.traverseReverse()

```

## Output:

```
Appending Values
traversing forward after Append
DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4
DATA VALUE = 8
DATA VALUE = 9
traversing reverse after Append
DATA VALUE = 9
DATA VALUE = 8
DATA VALUE = 4
DATA VALUE = 3
DATA VALUE = 2
DATA VALUE = 1
Removing Values
traversing forward after Remove
DATA VALUE = 1
DATA VALUE = 3
DATA VALUE = 4
DATA VALUE = 8
DATA VALUE = 9
traversing reverse after Remove
DATA VALUE = 9
DATA VALUE = 8
DATA VALUE = 4
DATA VALUE = 3
DATA VALUE = 1
>>>
```

## 11.2.8 Reversing a linked list

To reverse a linked list we have to reverse the pointers. Look at the following figure shown. The first table shows how information is stored in the linked list. The second table shows how the parameters are initialized in the `reverse()` function before beginning to traverse through the list and reversing the elements.

| Node 1         |              | Node 2              |              | Node 3 |              | Node 4 |              |
|----------------|--------------|---------------------|--------------|--------|--------------|--------|--------------|
| data           | reference to | data                | reference to | data   | reference to | data   | reference to |
| 1              | node2        | 2                   | node3        | 3      | node4        | 4      | none         |
| Initialization |              |                     |              |        |              |        |              |
| Parameters     |              | set to value of     |              |        | Final Value  |        |              |
| previous       |              | None                |              |        | None         |        |              |
| presentNode    |              | self.head           |              |        | node1        |        |              |
| nextval        |              | presentNode.refNext |              |        | node2        |        |              |

Figure 11.3

We then use the following while loop:

```

while nextval != None:
 presentNode.refNext = previous
 previous = presentNode
 presentNode = nextval
 nextval = nextval.refNext
presentNode.refNext = previous
self.head = presentNode

```

This is how the while loop works:

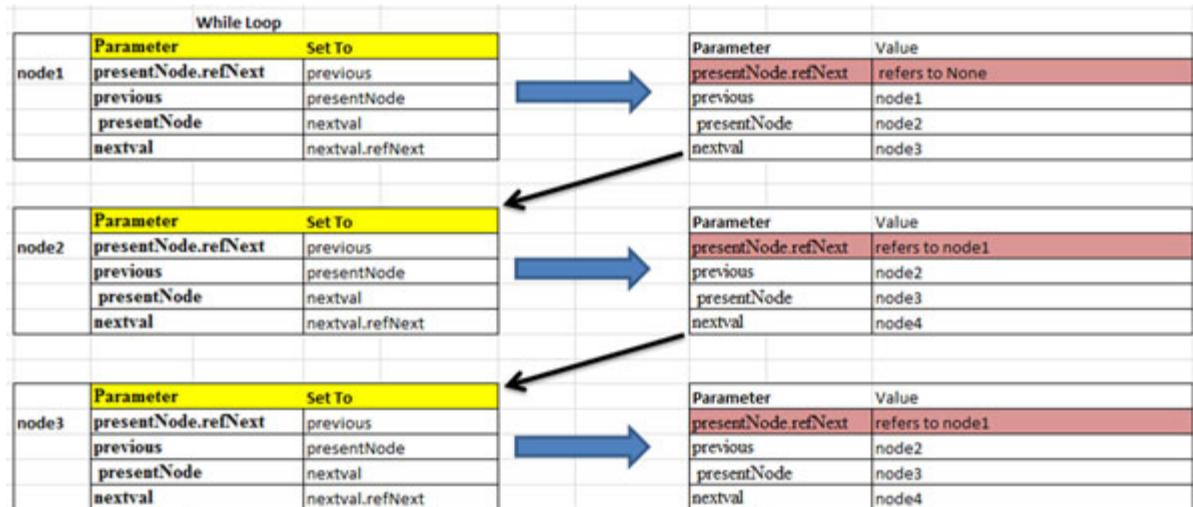


Figure 11.4

You can see as we iterate through the while loop how the values of presentnode.refNext change. Node1 that was earlier pointing to node2 changes its pointer to none. Same way node2 changes its pointer value to node1, and so on.

## Code:

```
class Node:

 def __init__(self,data = None):
 self.data = data
 self.refNext = None

class Linked_list:
 def __init__(self):
 self.head = None

 def reverse(self):
 previous = None
 presentNode = self.head
 nextval = presentNode.refNext
 while nextval != None:
 presentNode.refNext = previous
 previous = presentNode
 presentNode = nextval
 nextval = nextval.refNext

 presentNode.refNext = previous
 self.head = presentNode

 def traverse(self):

 presentNode = self.head
 while presentNode:
 print("DATA VALUE = ",presentNode.data)
 presentNode = presentNode.refNext
```

## Execution:

```
objNode1 = Node(1)
objNode2 = Node(2)
objNode3 = Node(3)
objNode4 = Node(4)
linkObj = Linked_list()
#head of the linked list to first object
linkObj.head = objNode1
reference of the first node object to second object
linkObj.head.refNext = objNode2
objNode2.refNext = objNode3
objNode3.refNext = objNode4
print("traverse before reversing")
linkObj.traverse()
linkObj.reverse()
print("traverse after reversing")
linkObj.traverse()
```

## **Output:**

```
traverse before reversing
DATA VALUE = 1
DATA VALUE = 2
DATA VALUE = 3
DATA VALUE = 4
traverse after reversing
DATA VALUE = 4
DATA VALUE = 3
DATA VALUE = 2
DATA VALUE = 1
```

## **Conclusion**

Python does not offer linked lists as part of its standard library. Linked lists contains data elements in a sequence and these data elements are connected to each other via links. The list element can be easily removed or inserted in a linked list without affecting its basic structure. Also it is a dynamic data structure that can shrink or grow during runtime. However linked lists occupy more memory and traversal though a linked list can be difficult.

# CHAPTER 12

## Trees

### Introduction

In this chapter, you will learn how to implement hierachial data structures – Trees.

### Structure

- Introduction
- Simple tree representation
- Representing a tree as list of lists
- Implement trees with Lists
- Binary Heap

### Objective

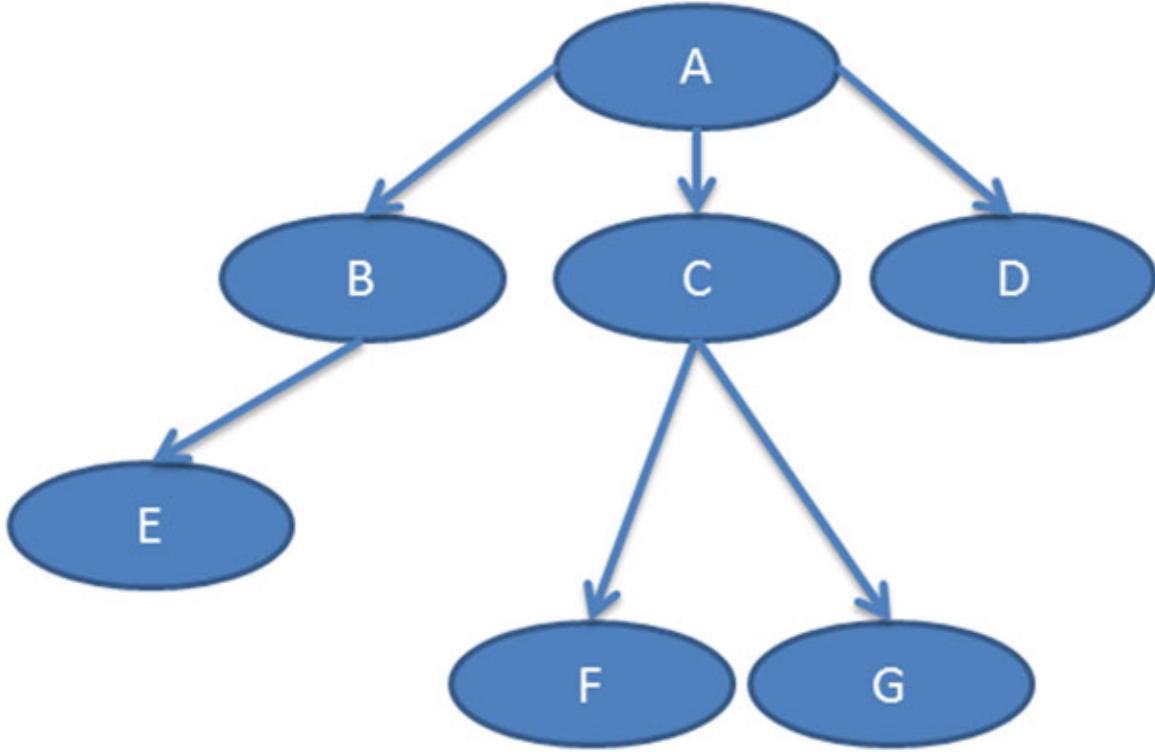
After reading this chapter you will have thorough knowledge of working with trees.

### 12.1 Introduction

There are several types of data structures that can be used to tackle application problems. We have seen how linked lists work in a sequential manner, we have also seen how stacks and queues can be used in programming applications but they are very restricted data structures. The biggest problem while dealing with linear data structure is that if we have to conduct a search, the time taken increases linearly with the size of data. Whereas there are some cases where linear structures can be really helpful but the fact remains that they may not be a good choice for situations that require high speed.

So, now let's move on from the concept of linear data structures to nonlinear data structures called **trees**. Every tree has a distinguished node called the **root**. Unlike the trees that we know in real life the tree data structure branches

downwards from parent to child and every node except the root is connected by a directly edge from exactly one other node.



*Figure 12.1*

Look at [Figure 12.1](#)

A is the root and it is parent to three nodes – B, C, and D.

Same way B is parent to E and C is parent to F and G.

Nodes like D, E, F, and G that have no children are called **leaves** or **external nodes**.

Nodes that have at least child such as B and C are called **internal nodes**.

The number of edges from root to node is called the **depth** or **level of a node**.  
Depth of B is 1 whereas depth of G is 2.

Height of a node is the number of edges from the node to the deepest leaf.

B, C, and D are siblings because they have same parent A. Similarly, F and G are also siblings because they have same parent C.

Children of one node are independent of children of another node.

**Every leaf node is unique**

- The file system that we use on our computer machines is an example of tree structure.
- Additional information about the node is known as **payload**. Payload is not given much of importance in algorithms but it plays a very important role in modern day computer applications.
- An **edge** connects two nodes to show that there is a relationship between them.
- There is only one incoming edge to every node(except the root). However, a node may have several outgoing edges.
- Root of the tree is the only node of the tree that has no incoming edges as it marks the starting point for a tree.
- Set of nodes having incoming edges from the same node are the children of that node.
- A node is a parent of all the nodes that connect to it by the outgoing edges.
- A set of nodes and edges that comprises of a parent along with all its descendants are called **subtrees**.
- A unique path traverses from the root to each node.
- A tree having maximum of two children is called a **binary tree**.

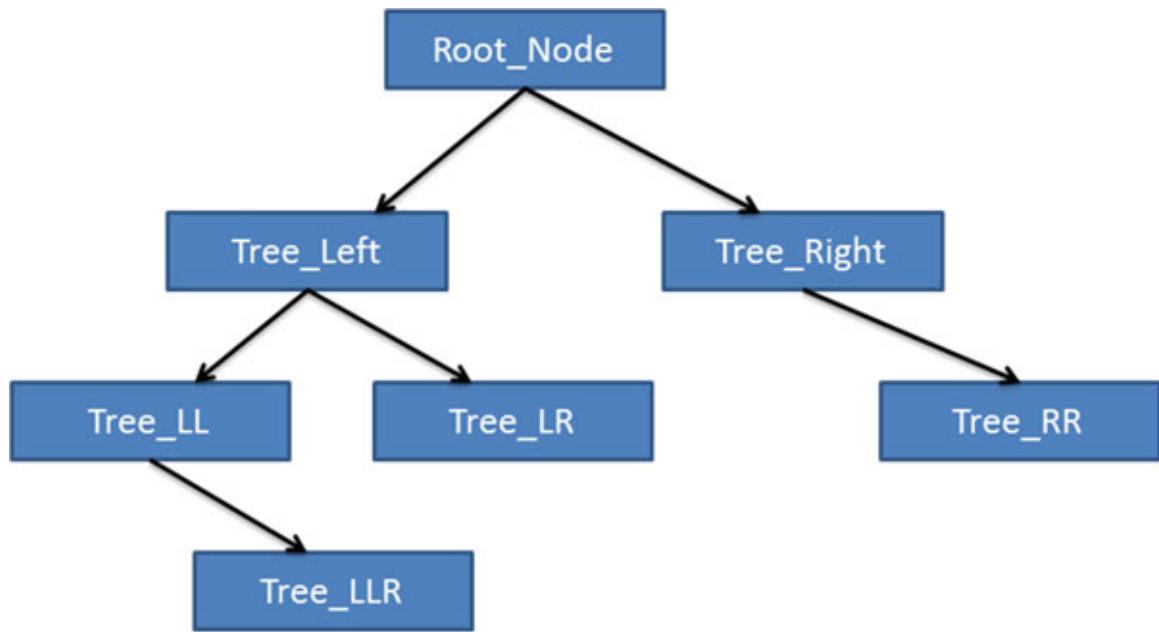
## **Recursive definition of a tree**

A tree can be empty, or it may have a root with zero or more subtree.

Root of every subtree is connected to the root of a parent tree by an edge.

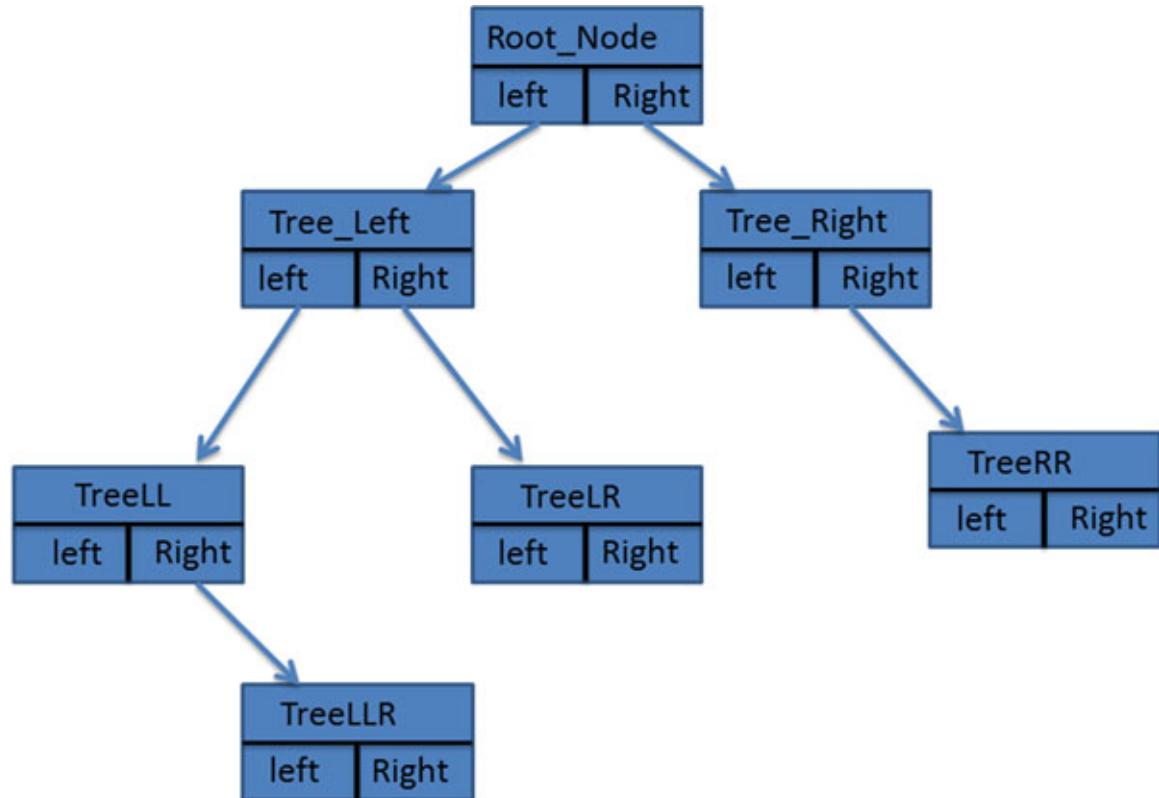
## **12.2 Simple tree representation**

Have a look at the following tree. Here, we are considering case of binary tree.



*Figure 12.2(a)*

In case of a binary tree, a node cannot have more than two children. So, for ease of understanding, we can say that preceding scenario is similar to something like this:

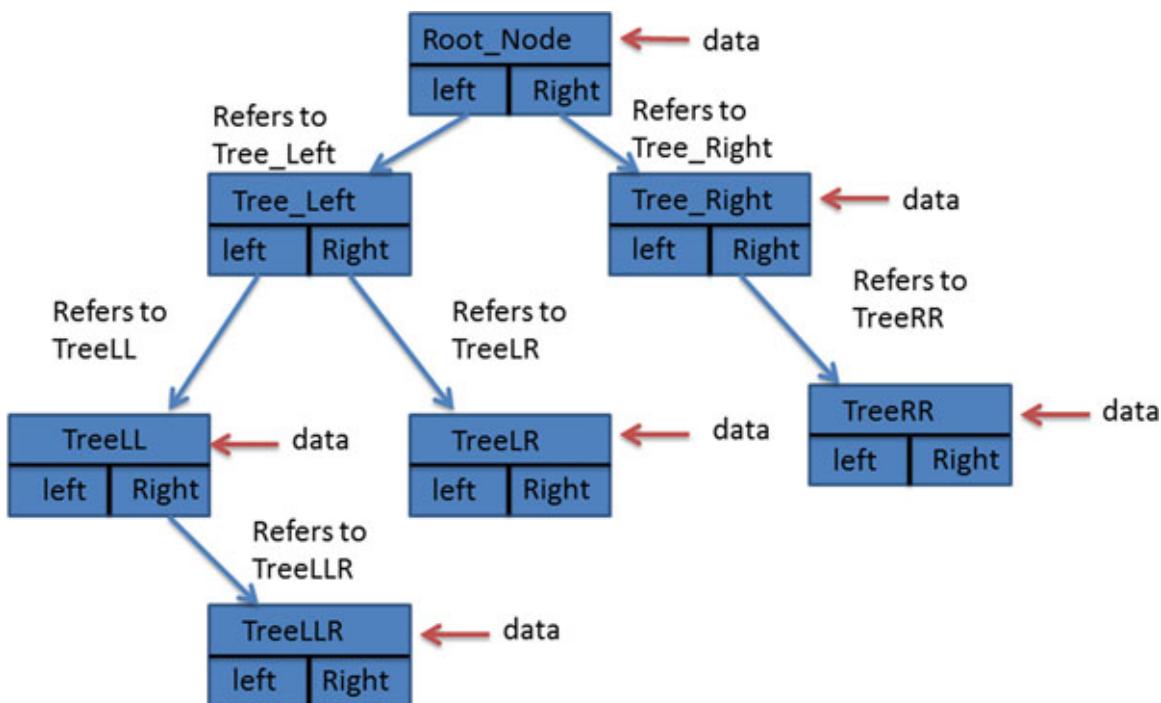


*Figure 12.2(b)*

In this diagram, left and right are references to the instances of node that are located on the left and right of this node respectively. As you can see, every node has three values:

1. Data
2. Reference to child on left
3. Reference to child on right

So, the constructor for the node class to create a Node object will be as follows in [figure 12.3](#):



*Figure 12.3*

Now let's create the **Node** class

```
Class Node(object):
 def __init__(self, data_value):
 self.data_value = data_value
 self.left = None
 self.right = None
```

So, this is how a root node is created:

```
Root_Node
```

```
print("Create Root Node")
root = Node("Root_Node")
print("Value of Root = ",root.data_value," left =",root.left, " right = ",root.right)
```

When we execute this block of code, the output is as follows:

Create Root Node

```
Value of Root = Root_Node left = None right = None
Value of Node = Tree_Left left = None right = None
```

Now, we write code for inserting values to the left or right.

When a node is created, initially its left and right reference point to None.

So, to add a child to left we just need to say:

```
self.left = child_node
```

And a child can be added to right in the similar way:

```
self.right = child_node
```

However, if the root node is already pointing to some existing child, and we try to insert a child node then the existing child should be pushed down one level and the new object must take its place. So, the reference of the existing child stored in `self.left` is passed on to `child.left` and then `self.left` is assigned the reference to child. This can be achieved in the following manner:

```
def insert_left(self, child):
 if self.left is None:
 self.left = child
 else:
 child.left = self.left
 self.left = child

def insert_right(self, child):
 if self.right is None:
 self.right = child
 else:
 child.right = self.right
 self.right = child
```

**Code:**

```
class Node(object):
 def __init__(self, data_value):
 self.data_value = data_value
```

```

 self.left = None
 self.right = None

 def insert_left(self, child):
 if self.left is None:
 self.left = child
 else:
 child.left = self.left
 self.left = child

 def insert_right(self, child):
 if self.right is None:
 self.right = child
 else:
 child.right = self.right
 self.right = child

```

## Execution:

```

Root_Node
print("Create Root Node")
root = Node("Root_Node")
print("Value of Root = ",root.data_value," left =",root.left, " right = ",root.right)

#Tree_Left
print("Create Tree_Left")
tree_left = Node("Tree_Left")
root.insert_left(tree_left)
print("Value of Node = ",tree_left.data_value," left =",tree_left.left, " right =
",tree_left.right)
print("Value of Root = ",root.data_value," left =",root.left, " right = ",root.right)

#Tree_Right
print("Create Tree_Right")
tree_right = Node("Tree_Right")
root.insert_right(tree_right)
print("Value of Node = ",tree_right.data_value," left =",tree_right.left, " right =
",tree_right.right)
print("Value of Root = ",root.data_value," left =",root.left, " right = ",root.right)

#TreeLL
print("Create TreeLL")
treell = Node("TreeLL")
tree_left.insert_left(treell)
print("Value of Node = ",treell.data_value," left =",treell.left, " right =
",treell.right)
print("Value of Node = ",tree_left.data_value," left =",tree_left.left, " right =
",tree_left.right)
print("Value of Root = ",root.data_value," left =",root.left, " right = ",root.right)

```

## Output:

```

Create Root Node
Value of Root = Root_Node left = None right = None
Create Tree_Left
Value of Node = Tree_Left left = None right = None
Value of Root = Root_Node left = <__main__.Node object at 0x000000479EC84F60> right =
None
Create Tree_Right
Value of Node = Tree_Right left = None right = None
Value of Root = Root_Node left = <__main__.Node object at 0x000000479EC84F60> right =
<__main__.Node object at 0x000000479ED05E80>
Create TreeLL
Value of Node = TreeLL left = None right = None
Value of Node = Tree_Left left = <__main__.Node object at 0x000000479ED0F160> right =
None
Value of Root = Root_Node left = <__main__.Node object at 0x000000479EC84F60> right =
<__main__.Node object at 0x000000479ED05E80>

```

## What is the definition of a tree?

A tree is a set of nodes that store elements. The nodes have parent-child relationship such that:

- If the tree is not empty, it has a special node called the root of tree. The root has no parents.
- Every node of the tree different from the root has a unique parent node.

## 12.3 Representing a tree as list of lists

In list of lists, we shall store the value of the node as the first element. The second element will be the list that represents the left subtree and the third element represents the right subtree. The following figure shows a tree with just the root node.

```
['Root_Node',[],[]]
```

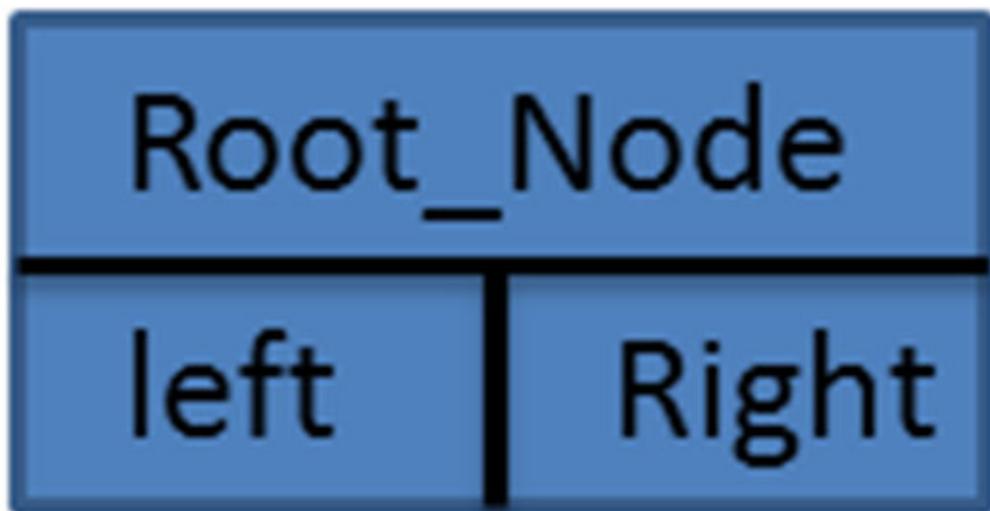
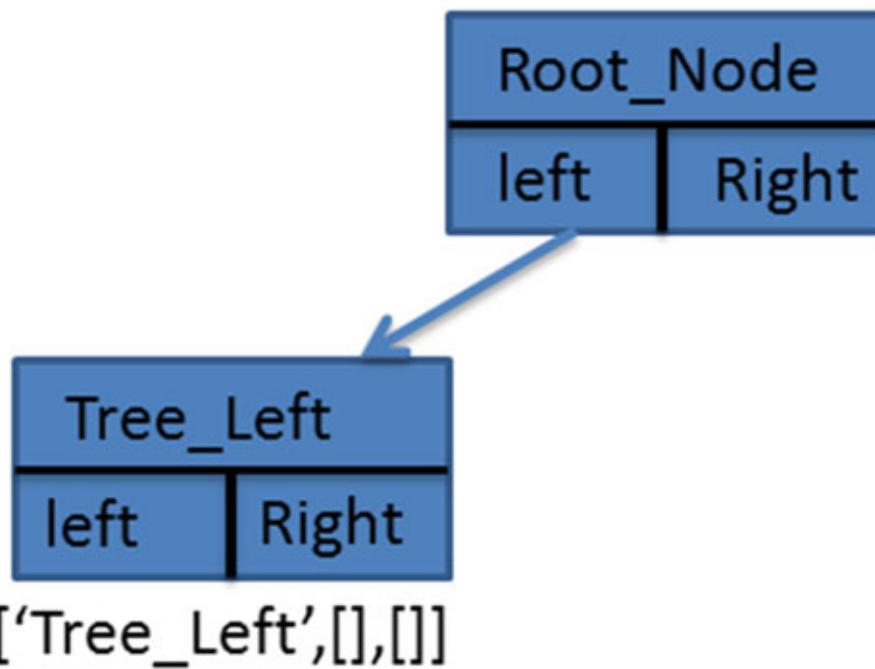


Figure 12.4

Now, suppose we add a node to the left.

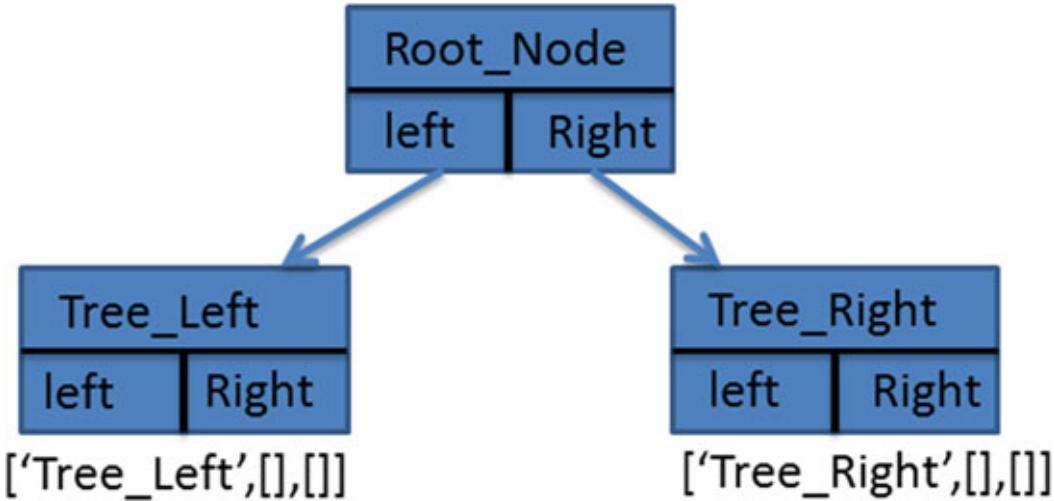
```
['Root_Node', ['Tree_Left',[],[]],[]]
```



*Figure 12.5*

Now, adding another subtree to the right would be equal to:

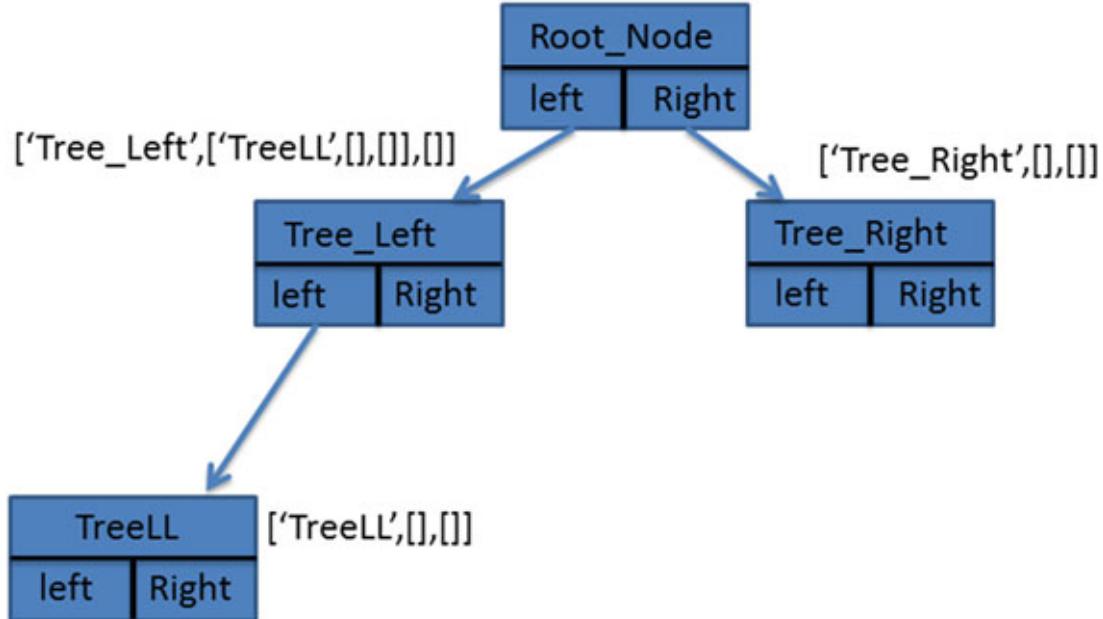
[‘Root\_Node’, [‘Tree\_Left’,[],[]], [‘Tree\_Right’,[],[]]]



*Figure 12.6*

Same way adding a node to the left of `Tree_Left` can be done as shown in [Figure 12.7](#).

[‘Root\_Node’, [‘Tree\_Left’, [‘TreeLL’,[],[]],[]], [‘Tree\_Right’,[],[]]]



*Figure 12.7*

Here you can see that the tree can be defined as follows:

```
binary_tree = ['Root_Node', ['Tree_Left', ['TreeLL', [], []], []], ['Tree_Right', [], []]]
```

Here, the root is `Root_Node` which is located at `binary_tree[0]`.

Left subtree is at `binary_tree[1]`.

Right subtree is at `binary_tree[2]`.

Now let's write the code for this.

### Step 1: Define Class

```
class Tree:
```

### Step 2: Create constructor

Now let's write the code for constructor. Here, when we create an object we pass a value. The constructor creates a list where the value is placed at index 0 and at index 1 and 2 we have two empty list. If we have to add subtree at the left side we will do so at index 1 and for right subtree we will insert values in the list at index 2.

```
def __init__(self,data):
 self.tree = [data, [], []]
```

### Step 3: Define function to insert left and right subtree

If you have to insert a value in left sub tree then pop the element at index 1 and insert the new list at that place. Similarly, if you have to insert a child at right hand side, pop the value at index 2 and insert the new list.

```
def left_subtree(self,branch):
 left_list = self.tree.pop(1)
 self.tree.insert(1,branch.tree)

def right_subtree(self,branch):
 right_list = self.tree.pop(2)
 self.tree.insert(2,branch.tree)
```

Now, let's execute the code:

### Code:

```
class Tree:
 def __init__(self,data):
 self.tree = [data, [], []]
```

```

def left_subtree(self,branch):
 left_list = self.tree.pop(1)
 self.tree.insert(1,branch.tree)

def right_subtree(self,branch):
 right_list = self.tree.pop(2)
 self.tree.insert(2,branch.tree)

```

## Execution:

```

print("Create Root Node")
root = Tree("Root_node")
print("Value of Root = ",root.tree)
print("Create Left Tree")
tree_left = Tree("Tree_Left")
root.left_subtree(tree_left)
print("Value of Tree_Left = ",root.tree)
print("Create Right Tree")
tree_right = Tree("Tree_Right")
root.right_subtree(tree_right)
print("Value of Tree_Right = ",root.tree)

```

## Output:

```

Create Root Node
Value of Root = ['Root_node', [], []]
Create Left Tree
Value of Tree_Left = ['Root_node', ['Tree_Left', [], [], []], []]
Create Right Tree
Value of Tree_Right = ['Root_node', ['Tree_Left', [], []], ['Tree_Right', [], []]]

```

There is however one thing ignored in this code. What if we want to insert a child somewhere in between? Here, in this case the child will be inserted at the given location and the subtree existing at that place will be pushed down.

For this we make changes in the insert functions.

```

def left_subtree(self,branch):
 left_list = self.tree.pop(1)
 if len(left_list) > 1:
 branch.tree[1]=left_list
 self.tree.insert(1,branch.tree)
 else:
 self.tree.insert(1,branch.tree)

```

If we have to insert a child in the left then first we pop the element at index 1. If the length of element at index 1 is 0 then, we simply insert the list. However,

if the length is not zero then we push the element to the left of the new child. The same happens in case of right subtree.

```
def right_subtree(self,branch):
 right_list = self.tree.pop(2)
 if len(right_list) > 1:
 branch.tree[2]=right_list
 self.tree.insert(2,branch.tree)
 else:
 self.tree.insert(2,branch.tree)

print("Create TreeLL")
treell = Tree("TreeLL")
tree_left.left_subtree(treell)
print("Value of Tree_Left = ",root.tree)
```

## Code:

```
class Tree:
 def __init__(self,data):
 self.tree = [data, [], []]

 def left_subtree(self,branch):
 left_list = self.tree.pop(1)
 if len(left_list) > 1:
 branch.tree[1]=left_list
 self.tree.insert(1,branch.tree)
 else:
 self.tree.insert(1,branch.tree)
 def right_subtree(self,branch):
 right_list = self.tree.pop(2)
 if len(right_list) > 1:
 branch.tree[2]=right_list
 self.tree.insert(2,branch.tree)
 else:
 self.tree.insert(2,branch.tree)
```

## Execution:

```
print("Create Root Node")
root = Tree("Root_node")
print("Value of Root = ",root.tree)
print("Create Left Tree")
tree_left = Tree("Tree_Left")
root.left_subtree(tree_left)
print("Value of Tree_Left = ",root.tree)
print("Create Right Tree")
tree_right = Tree("Tree_Right")
root.right_subtree(tree_right)
print("Value of Tree_Right = ",root.tree)
```

```

print("Create Left Inbetween")
tree_inbtw = Tree("Tree left in between")
root.left_subtree(tree_inbtw)
print("Value of Tree_Left = ",root.tree)
print("Create TreeLL")
treell = Tree("TreeLL")
tree_left.left_subtree(treell)
print("Value of TREE = ",root.tree)

```

## Output:

```

Create Root Node
Value of Root = ['Root_node', [], []]
Create Left Tree
Value of Tree_Left = ['Root_node', ['Tree_Left', [], []], []]
Create Right Tree
Value of Tree_Right = ['Root_node', ['Tree_Left', [], []], ['Tree_Right', [], []]]
Create Left Inbetween
Value of Tree_Left = ['Root_node', ['Tree left in between', ['Tree_Left', [], []],
[], ['Tree_Right', [], []]]]
Create TreeLL
Value of TREE = ['Root_node', ['Tree left in between', ['Tree_Left', ['TreeLL', [], []], []], ['Tree_Right', [], []]]]

```

### 12.3.1 Tree traversal methods

In this section, you will look at three types of Tree Traversal Methods:

- Preorder Traversal
- In Order Traversal
- Post Order Traversal

#### 12.3.1.1 Preorder Traversal

In **Preorder Traversal** we will first visit the root node, then visit all nodes on the left followed by all nodes on the right.

## Code:

```

class Node(object):
 def __init__(self, data_value):
 self.data_value = data_value
 self.left = None
 self.right = None

 def insert_left(self, child):

```

```

 if self.left is None:
 self.left = child
 else:
 child.left = self.left
 self.left = child

 def insert_right(self, child):
 if self.right is None:
 self.right = child
 else:
 child.right = self.right
 self.right = child

 def preorder(self, node):
 res=[]
 if node:
 res.append(node.data_value)
 res = res + self.preorder(node.left)
 res = res + self.preorder(node.right)
 return res

```

## Execution:

```

Root_Node
print("Create Root Node")
root = Node("Root_Node")

#Tree_Left
print("Create Tree_Left")
tree_left = Node("Tree_Left")
root.insert_left(tree_left)
#Tree_Right
print("Create Tree_Right")
tree_right = Node("Tree_Right")
root.insert_right(tree_right)

#TreeLL
print("Create TreeLL")
treell = Node("TreeLL")
tree_left.insert_left(treell)
print("*****Preorder Traversal*****")
print(root.preorder(root))

```

## Output:

```

Create Root Node
Create Tree_Left
Create Tree_Right
Create TreeLL
*****Preorder Traversal*****
['Root_Node', 'Tree_Left', 'TreeLL', 'Tree_Right']

```

```
>>>
```

### 12.3.1.2 In Order Traversal

In this case, we will first visit all nodes on the left then the root node and then all nodes on the right.

```
class Node(object):
 def __init__(self, data_value):
 self.data_value = data_value
 self.left = None
 self.right = None

 def insert_left(self, child):
 if self.left is None:
 self.left = child
 else:
 child.left = self.left
 self.left = child

 def insert_right(self, child):
 if self.right is None:
 self.right = child
 else:
 child.right = self.right
 self.right = child

 def inorder(self, node):
 res=[]
 if node:
 res = self.inorder(node.left)
 res.append(node.data_value)
 res = res + self.inorder(node.right)
 return res
```

### **Execution:**

```
Root_Node
print("Create Root Node")
root = Node("Root_Node")

#Tree_Left
print("Create Tree_Left")
tree_left = Node("Tree_Left")
root.insert_left(tree_left)

#Tree_Right
print("Create Tree_Right")
tree_right = Node("Tree_Right")
root.insert_right(tree_right)
```

```
#TreeLL
print("Create TreeLL")
treell = Node("TreeLL")
tree_left.insert_left(treell)
print("*****Inorder Traversal*****")
print(root.inorder(root))
```

## Output:

```
Create Root Node
Create Tree_Left
Create Tree_Right
Create TreeLL
*****Inorder Traversal*****
['TreeLL', 'Tree_Left', 'Root_Node', 'Tree_Right']
>>>
```

### 12.3.1.3 Post Order Traversal

Visit all nodes on the left, then visit all nodes on the right, then visit the root node.

## Code:

```
class Node(object):

 def __init__(self, data_value):
 self.data_value = data_value
 self.left = None
 self.right = None

 def insert_left(self, child):
 if self.left is None:
 self.left = child
 else:
 child.left = self.left
 self.left = child

 def insert_right(self, child):
 if self.right is None:
 self.right = child
 else:
 child.right = self.right
 self.right = child

 def postorder(self, node):
 res=[]
 if node:
 res = self.postorder(node.left)
 res = res + self.postorder(node.right)
```

```
 res.append(node.data_value)
return res
```

## Execution:

```
Root_Node
print("Create Root Node")
root = Node("Root_Node")

#Tree_Left
print("Create Tree_Left")
tree_left = Node("Tree_Left")
root.insert_left(tree_left)

#Tree_Right
print("Create Tree_Right")
tree_right = Node("Tree_Right")
root.insert_right(tree_right)

#TreeLL
print("Create TreeLL")
treell = Node("TreeLL")
tree_left.insert_left(treell)
print("*****Postorder Traversal*****")
print(root.postorder(root))
```

## Output:

```
Create Root Node
Create Tree_Left
Create Tree_Right
Create TreeLL
*****Postorder Traversal*****
['TreeLL', 'Tree_Left', 'Tree_Right', 'Root_Node']
```

## 12.4 Binary Heap

A binary heap is a binary tree. It is a complete tree, which means that all levels are completely filled except possibly the last level. It also means that the tree is balanced.

As every new item is inserted next to the available space. A Binary heap can be stored in an array.

A binary heap can be of two types: Min Heap or Max Heap. In case of Min Heap binary heap, the root node is the minimum of all the nodes in the binary heap, all parent nodes are smaller than their children. On the other hand in case

of Max Heap the root is the maximum among all the keys present in the heap, and all nodes are bigger than their children.

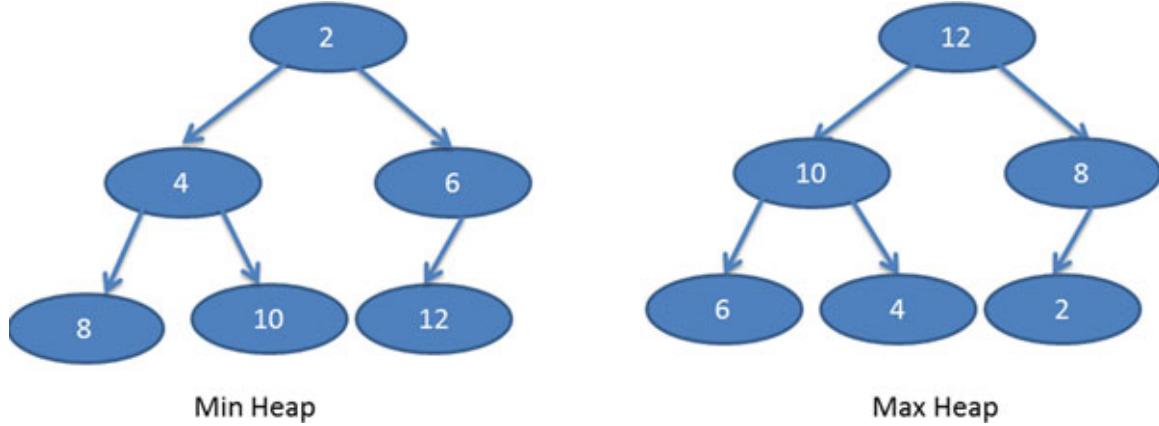
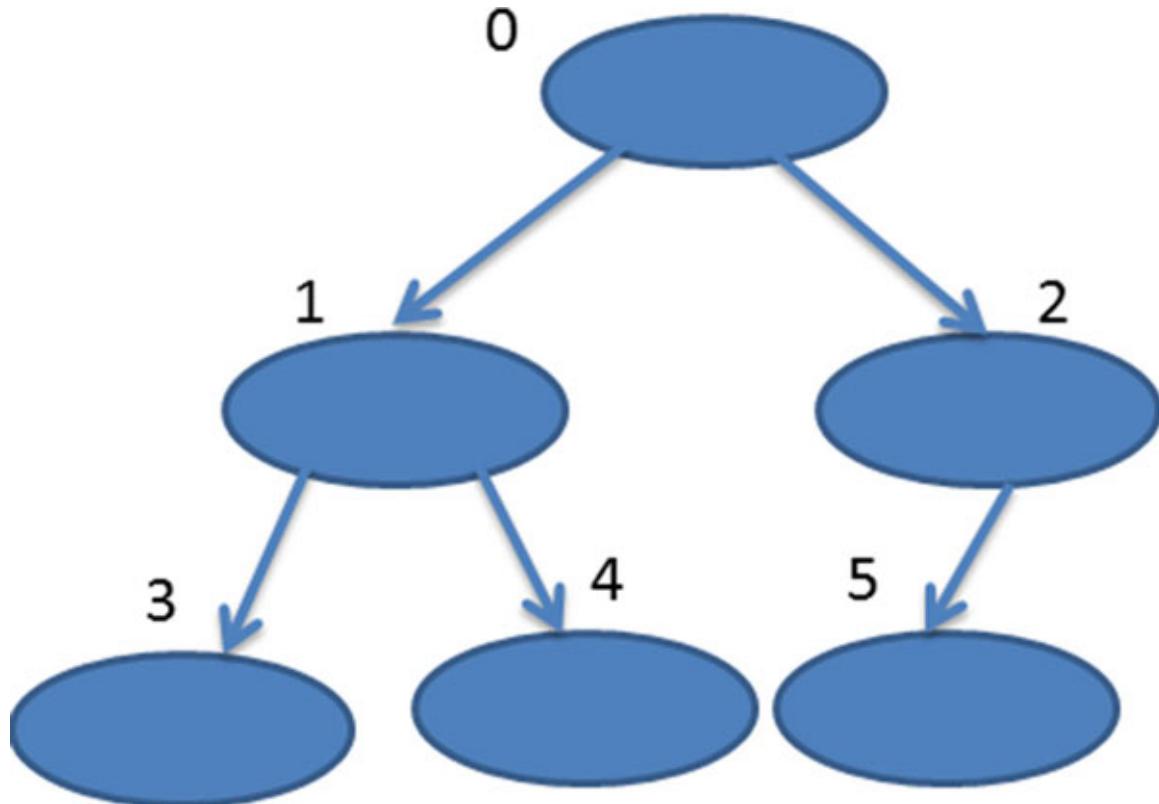


Figure 12.8

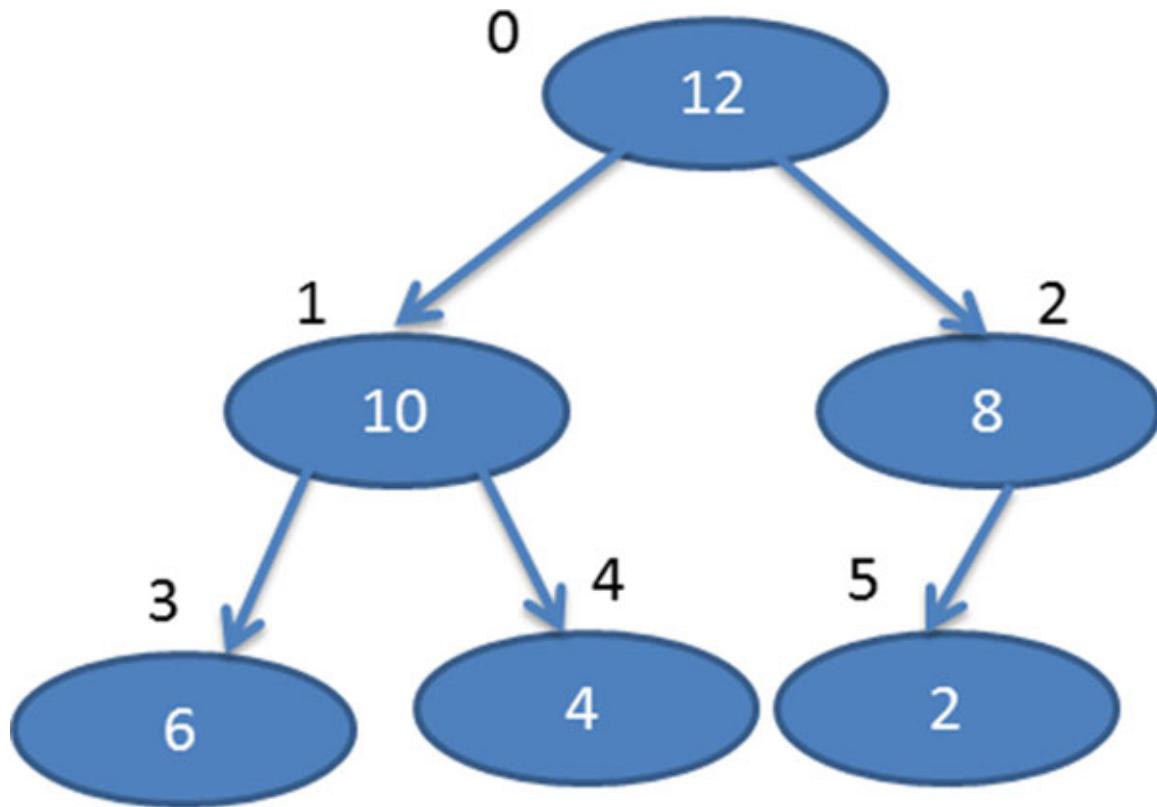
Heap has two important properties:

1. It is complete and is constructed from left to right across each row and the last row may not be completely full. Each row should be filled up sequentially. So, the order of inserting values should be from left to right, row by row sequentially as shown in the following figure:



*Figure 12.9*

2. The parent must be larger in case of Max Heap and smaller in case of Min Heap than the children.



*Figure 12.10*

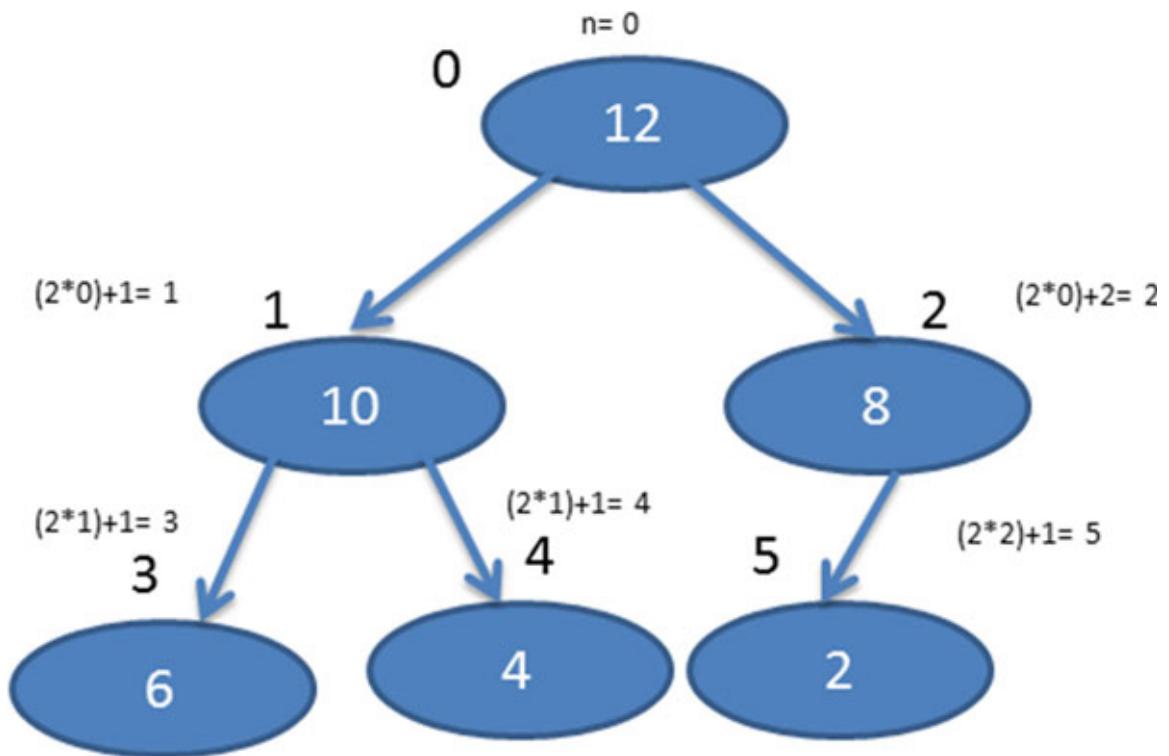
Look at the preceding figure, it is a case of Maximum heap because all parents are greater than their children.

The binary heap can be represented in an array as follows:



*Figure 12.11*

If you look at the array carefully you will realize that if a parent exists at location n then the left child is at  $2n+1$  and the right child is at  $2n + 2$ .



*Figure 12.12*

So, if we know the location of the parent child we can easily find out the location of the left and right child.

Now suppose we have to build up a Max Heap using following values:  
20, 4, 90, 1, 125

### Step 1

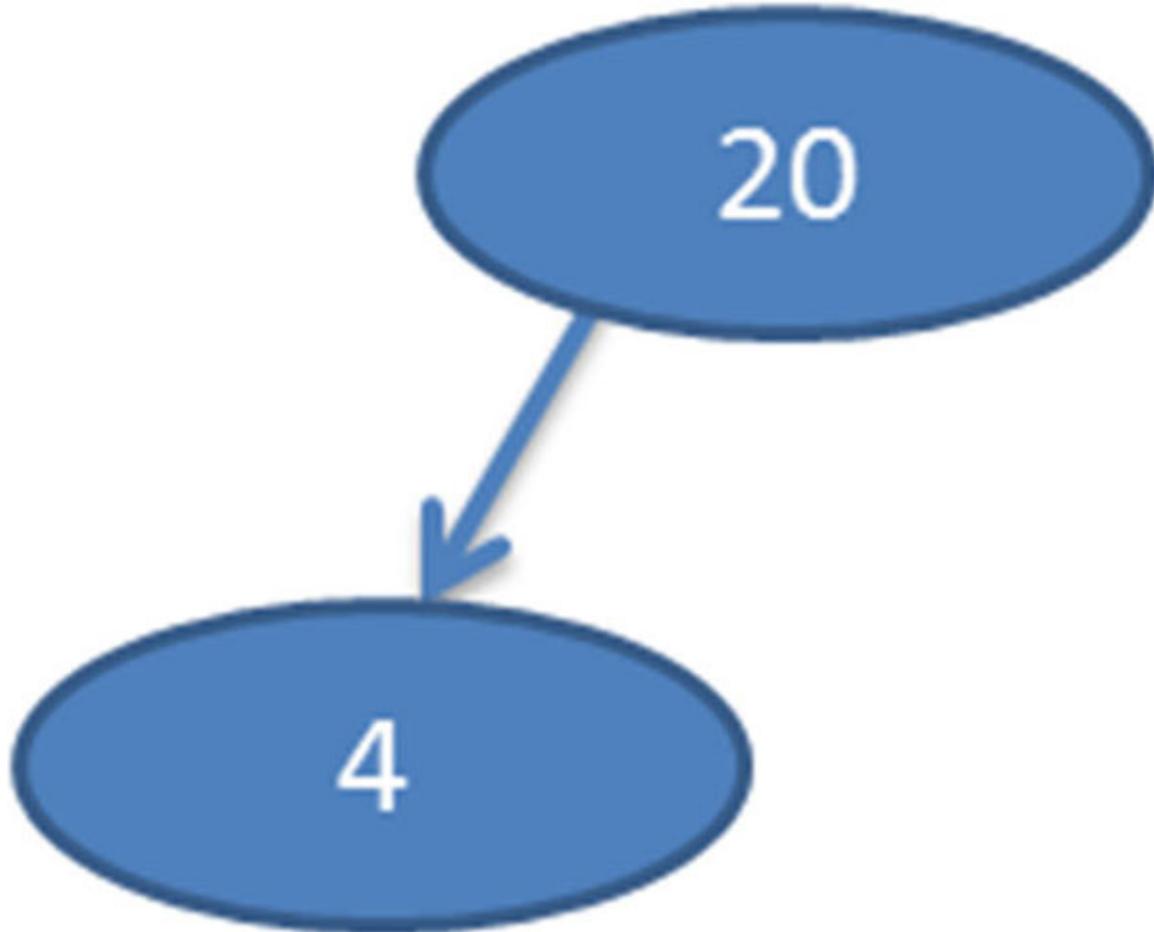
Insert 20



*Figure 12.13*

### Step 2

Insert 4 -> Left to Right-> First element in second row



*Figure 12.14*

Since the parent is greater than the child, this is fine.

### Step 3

Insert 90 -> Left to Right -> Second element in second row

However, when we insert 90 as the right child, it violates the rule of the max heap because the parent has a key value of 20. So, in order to resolve this problem it is better to swap places as shown in the following figure:



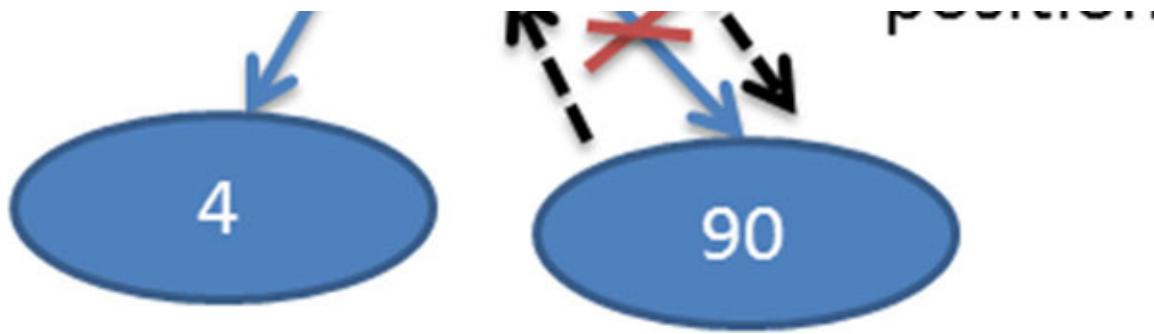


Figure 12.15

After swapping, the heap would look something like this:

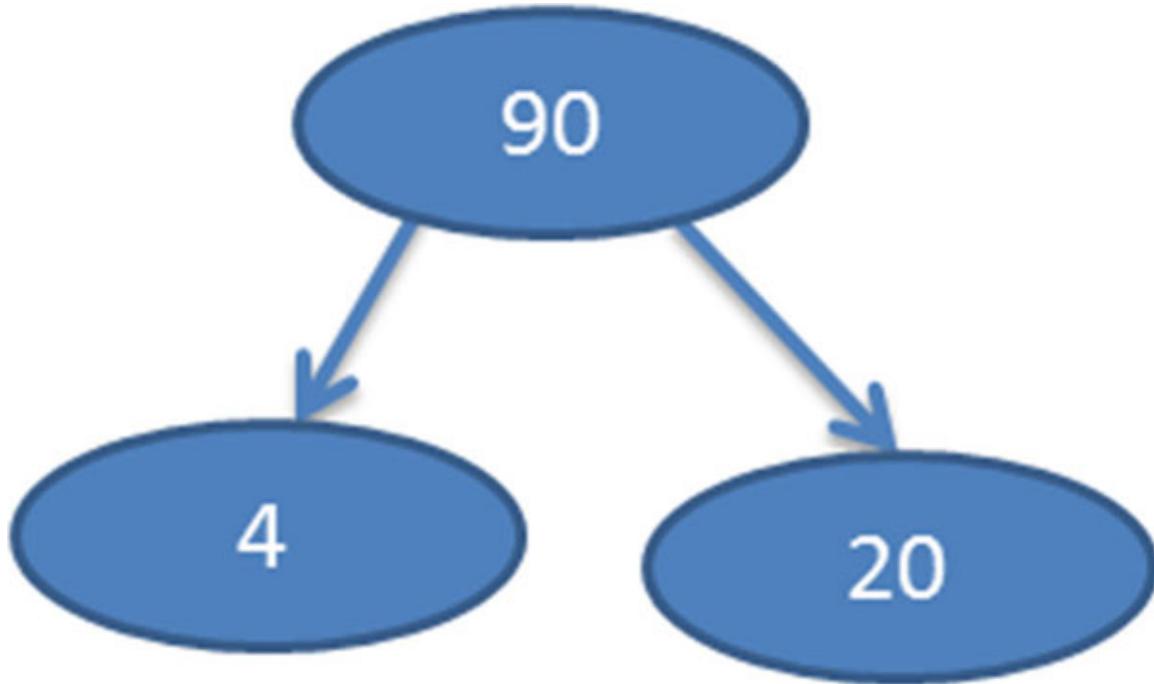
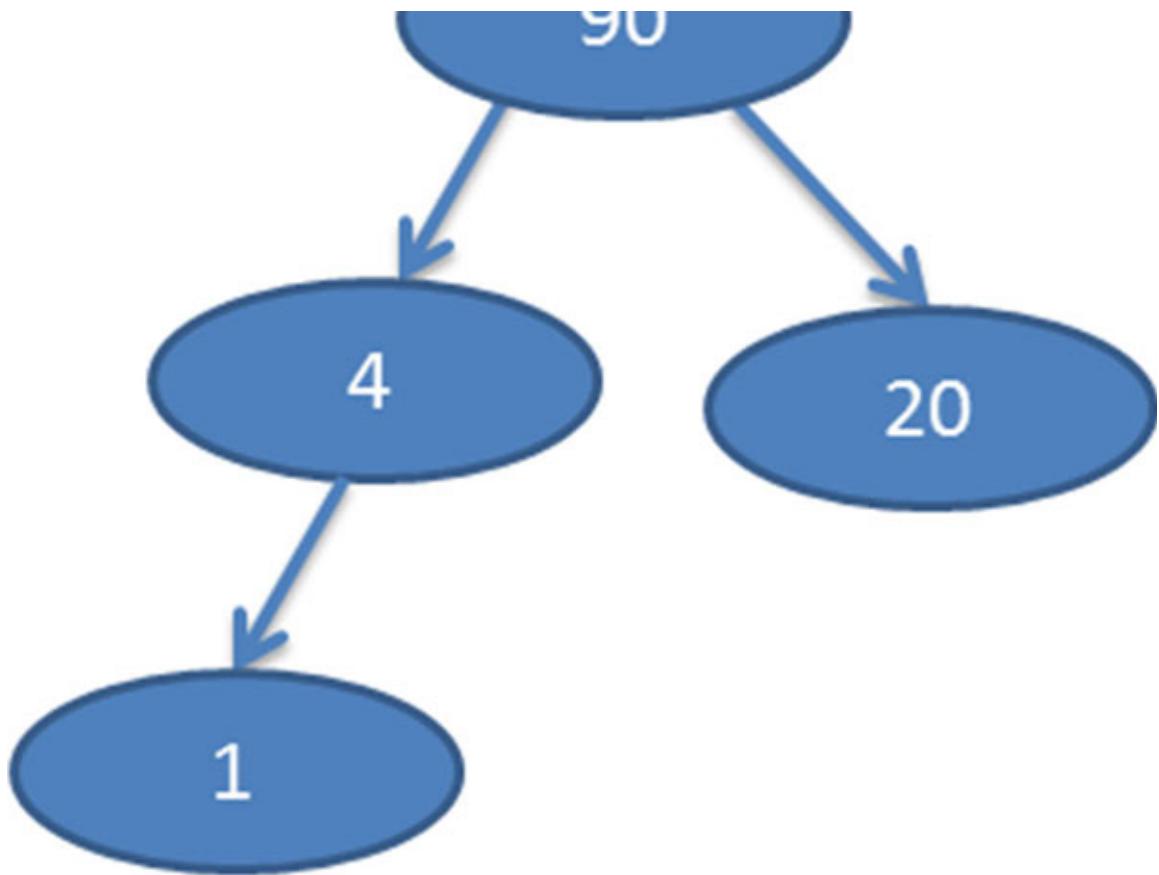


Figure 12.16

#### Step 4:

Insert 1 -> left to right -> first element third row



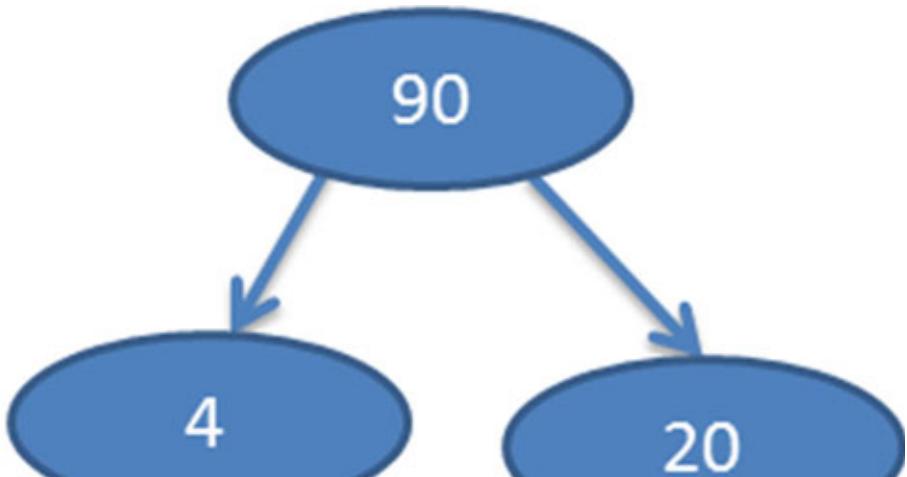


*Figure 12.17*

Since 1 is smaller than the parent node, this is fine.

### Step 5

Insert 125



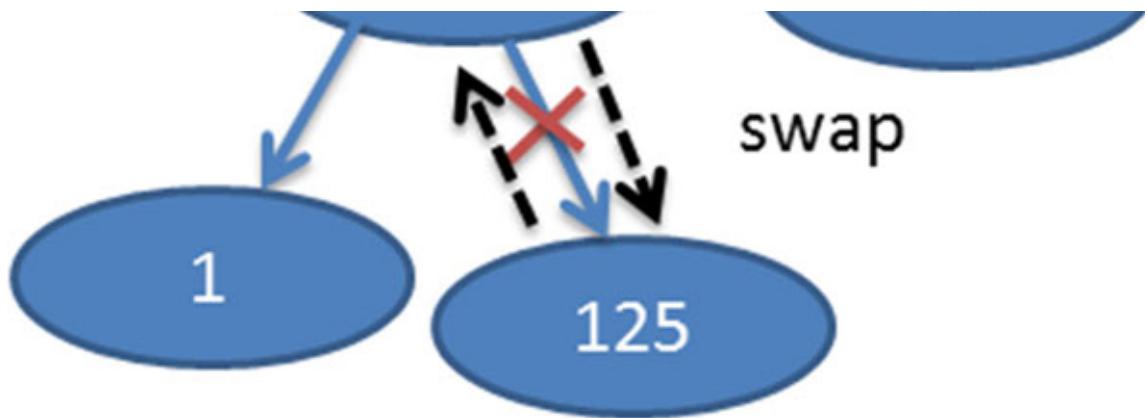
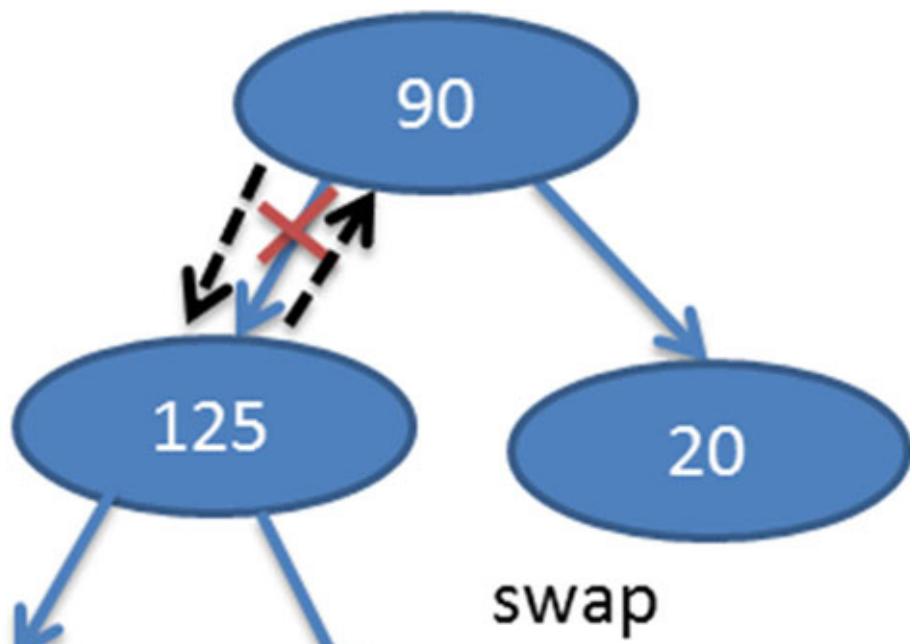


Figure 12.18

This violated the rule of Max Heap

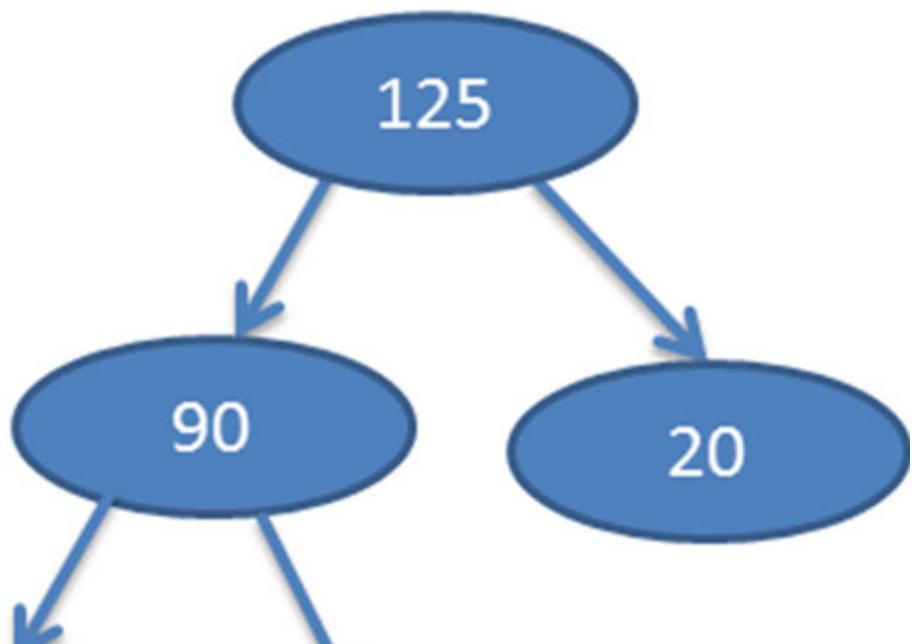
Swap 4 and 125

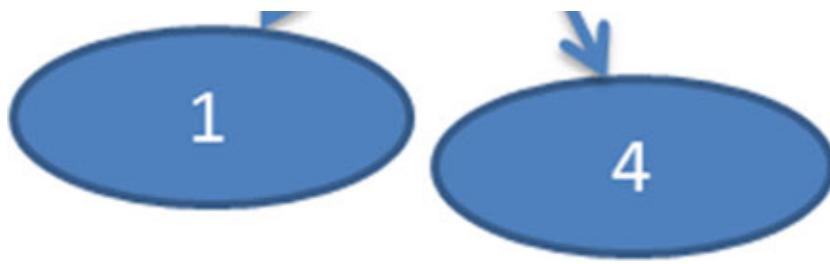




*Figure 12.19*

Not a heap still  
Swap 125 and 90





*Figure 12.20*

We now have a Max Heap.

### **Example 12.1**

Write python code to implement **Max Heap**. How would you insert a value so that the root node has the maximum value and all parents are greater than their children.

**Answer:** Here we will write the code to implement `MaxHeap` class. The class will have two functions:

`Push()` : To insert value.

`Float_up()`: To place the value where it belongs.

### **Step 1**

Define the class

```
class MaxHeap:
```

## Step 2

Define the Constructor

```
def __init__(self):
 self.heap = []
```

## Step 3

Define the `push()` function.

The `push` function does two jobs:

1. Appends the value at the end of the list. (We have seen earlier that the value has to be inserted first at the available location)
2. After appending the value, the `push()` function will call the `float_up(index)` function. The `push()` function passes the index of the last element to the `float_up()` function so that the `float_up()` function can analyse the values and do the needful as described in the next step.

```
def push(self,value):
 self.heap.append(value)
 self.float_up(len(self.heap)-1)
```

## Step 4

Define the `float_up` function

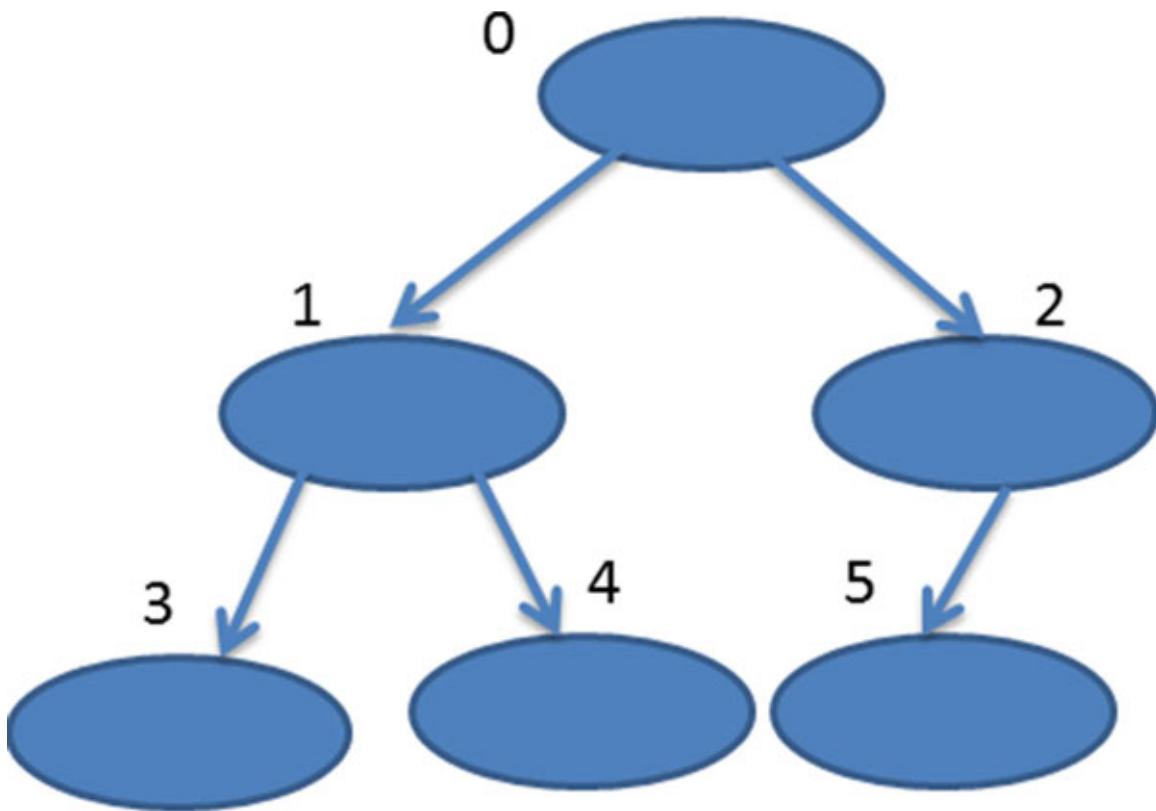
1. The `float_up()` function takes `index` value as parameter.

```
def float_up(self,index):
```

2. The `push()` function passes the index value of the last element in the heap. The first thing that the `float_up` function does is that it checks if the element is at index 0 or not. If yes, then it is the root node. Root node has no parents and since it's the last element of the heap it has no children either. So, we can return the value.

```
if index==0:
 return
```

3. If the index is greater than 0 then we can proceed further. Look the following figure once again:



*Figure 12.21*

For programming, you need to understand few things:

The index 0 has two children at position 1 and 2. If there is an element at 1 then you can find the parent by calculating value of  $(1//2)$ . Similarly for element at index 2, the parent's index value can be found out by calculating value of  $(2//2 -1)$ . So, we can say that if an element has an odd index value then its parent's index value is defined as **parent\_of\_index= index//2**. However, if its index value is even then **parent\_of\_index** will be equal **(index//2)-1**. This becomes the outer frame for the **float\_up** function. The right child has an even number as index and the left child has an odd number as index.

```

if index==0:
 return
else:
 if index%2==0:
 parent_of_index = (index//2)-1
 -----write code-----
 else:
 parent_of_index = index//2
 -----write code-----

```

4. Now, we compare the values of the child and the parent. If the child is greater than the parent, swap the elements.

```
def float_up(self, index):
 if index==0:
 return
 else:
 if index%2==0:
 parent_of_index = (index//2)-1
 if self.heap[index]> self.heap[parent_of_index]:
 self.swap(index, parent_of_index)
 else:
 parent_of_index = index//2
 if self.heap[index]> self.heap[parent_of_index]:
 self.swap(index, parent_of_index)
 self.float_up(parent_of_index)
```

## Step 5

Define the swap function

The `swap()` function helps in swapping the values of the parent and the child. The code for this is given as follows:

```
def swap(self, index1, index2):
 temp = self.heap[index1]
 self.heap[index1] = self.heap[index2]
 self.heap[index2] = temp
```

## Code:

```
class MaxHeap:

 def __init__(self):
 self.heap = []

 def push(self,value):
 self.heap.append(value)
 self.float_up(len(self.heap)-1)

 def float_up(self, index):
 if index==0:
 return
 else:
 if index%2==0:
 parent_of_index = (index//2)-1
 if self.heap[index]> self.heap[parent_of_index]:
 self.swap(index, parent_of_index)
 else:
 parent_of_index = index//2
 if self.heap[index]> self.heap[parent_of_index]:
```

```

 self.swap(index, parent_of_index)
 self.float_up(parent_of_index)

def peek(self):
 print(self.heap[0])

def pop(self):
 if len(self.heap)>=2:
 temp = self.heap[0]
 self.heap[0]=self.heap[len(self.heap)-1]
 self.heap[len(self.heap)-1]
 self.heap.pop()
 self.down_adj()
 elif len(self.heap)==1:
 self.heap.pop()
 else:
 print("Nothing to pop")

def swap(self,index1, index2):
 temp = self.heap[index1]
 self.heap[index1] = self.heap[index2]
 self.heap[index2] = temp

```

## Execution:

```

H = MaxHeap()
print("*****pushing values*****")
print("pushing 165")

H.push(165)
print(H.heap)
print("pushing 60")

H.push(60)
print(H.heap)
print("pushing 179")

H.push(179)
print(H.heap)
print("pushing 400")

H.push(400)
print(H.heap)
print("pushing 6")

H.push(6)
print(H.heap)
print("pushing 275")

H.push(275)
print(H.heap)

```

## Output:

```
*****pushing values*****
pushing 165
[165]

pushing 60
[165, 60]

pushing 179
[179, 60, 165]

pushing 400
[400, 179, 165, 60]

pushing 6
[400, 179, 165, 60, 6]

pushing 275
[400, 179, 275, 60, 6, 165]
>>>
```

## Example 12.2

Write code to find out the maximum value in the Max heap.

### Answer

It is easy to find the max value in the max heap as the maximum value is available at the root node which is index 0 of the heap. If you call the function `peek()` in *example 12.1* it will display the maximum value of the heap.

```
def peek(self):
 print(self.heap[0])
```

## Example 12.3

Write the code to pop the maximum value from Max Heap.

**Answer:** There are two steps involved:

1. Swap the root with the last element in the array and pop the value.
2. The root node now does have the maximum value. So, now we need to move downwards, comparing the parent with their left and right child to ensure that the children are smaller than the parent. If not we will have to swap places again.

### Step 1

Define `pop()` function

The `pop()` function which swaps the values of the root node with the last element in the list and pops the value. The function then calls the `down_adj()` function which moves downwards and adjusts the values. The `pop()` function first checks the size of the heap. If the length of the heap is 1 then that means that it only contains one element that's the root and there is no need to swap further.

```
def pop(self):
 if len(self.heap)>2:
 temp = self.heap[0]
 self.heap[0]=self.heap[len(self.heap)-1]
 self.heap[len(self.heap)-1]
 self.heap.pop()
 print("heap after popping largest value =", self.heap)
 self.down_adj()
 elif len(self.heap)==1:
 self.heap.pop()
 else:
 print("Nothing to pop")
```

## Step 2

Define `down_adj()` function

Set index value to 0.

Index of left child = `left_child = index*2+1`

Index of right child = `right_child = index*2+2`

Then we go through loop where we check the value of the parent with both left and right child. If the parent is smaller than the left child then we swap the value. We then compare the parent value with the right child, if it is less then we swap again.

This can be done as follows:

- Check if parent is less than left child:
  - If yes, check if left child is less than the right child
  - If yes, then swap the parent with the right child
  - Change the value of index to value of `right_child` for further assessment
- If no the just swap the parent with the left child
- Set the value of index to value of `left_child` for further assessment

- If the parent is not less than left child but only less than the right child then swap values with the right child
- Change the value of index to `right_child`

```
def down_adj(self):
 index = 0
 for i in range(len(self.heap)//2):
 left_child = index*2+1
 if left_child > len(self.heap):
 return
 print("left child = ", left_child)
 right_child = index*2+2
 if right_child > len(self.heap):
 return
 print("right child = ", right_child)
 if self.heap[index]<self.heap[left_child]:
 temp = self.heap[index]
 self.heap[index] = self.heap[left_child]
 self.heap[left_child] = temp
 index = left_child
 if self.heap[index]<self.heap[right_child]:
 temp = self.heap[index]
 self.heap[index] = self.heap[right_child]
 self.heap[right_child] = temp
 index = right_child
```

## Code:

```
class MaxHeap:

 def __init__(self):
 self.heap = []

 def push(self,value):
 self.heap.append(value)
 self.float_up(len(self.heap)-1)

 def float_up(self,index):
 if index==0:
 return
 else:
 if index%2==0:
 parent_of_index = (index//2)-1
 if self.heap[index]> self.heap[parent_of_index]:
 temp = self.heap[parent_of_index]
 self.heap[parent_of_index] = self.heap[index]
 self.heap[index] = temp
 else:
 parent_of_index = index//2
 if self.heap[index]> self.heap[parent_of_index]:
 temp = self.heap[parent_of_index]
```

```

 self.heap[parent_of_index] = self.heap[index]
 self.heap[index] = temp
 self.float_up(parent_of_index)

 def peek(self):
 print(self.heap[0])

 def pop(self):
 if len(self.heap)>=2:
 temp = self.heap[0]
 self.heap[0]=self.heap[len(self.heap)-1]
 self.heap[len(self.heap)-1]
 self.heap.pop()
 self.down_adj()
 elif len(self.heap)==1:
 self.heap.pop()
 else:
 print("Nothing to pop")
 def swap(self,index1, index2):
 temp = self.heap[index1]
 self.heap[index1] = self.heap[index2]
 self.heap[index2] = temp

 def down_adj(self):
 index = 0
 for i in range(len(self.heap)//2):
 left_child = index*2+1
 if left_child > len(self.heap)-1:
 print(self.heap)
 print("End Point")
 print("Heap value after pop() = ",self.heap)
 return
 right_child = index*2+2
 if right_child > len(self.heap)-1:
 print("right child does not exist")
 if self.heap[index]<self.heap[left_child]:
 self.swap(index,left_child)
 index = left_child
 print("Heap value after pop() = ",self.heap)
 return
 if self.heap[index]<self.heap[left_child]:
 if self.heap[left_child]<self.heap[right_child]:
 self.swap(index,right_child)
 index = right_child
 else:
 self.swap(index,left_child)
 index = left_child
 elif self.heap[index]<self.heap[right_child]:
 self.swap(index,right_child)
 index = right_child
 else:
 print("No change required")
 print("Heap value after pop() = ",self.heap)

```

## Execution:

```
H = MaxHeap()
print("*****pushing values*****")

H.push(165)
print(H.heap)

H.push(60)
print(H.heap)

H.push(179)
print(H.heap)

H.push(400)
print(H.heap)

H.push(6)
print(H.heap)

H.push(275)
print(H.heap)
print("*****popping values*****")

H.pop()
H.pop()
H.pop()
H.pop()
H.pop()
H.pop()
H.pop()
```

## Output:

```
pushing values
[165]
[165, 60]
[179, 60, 165]
[400, 179, 165, 60]
[400, 179, 165, 60, 6]
[400, 179, 275, 60, 6, 165]

*****popping values*****
[275, 179, 165, 60, 6]

End Point
Heap value after pop() = [275, 179, 165, 60, 6]
right child does not exist
Heap value after pop() = [179, 60, 165, 6]
Heap value after pop() = [165, 60, 6]
right child does not exist
Heap value after pop() = [60, 6]
Heap value after pop() = [6]
```

```
| Nothing to pop
>>>
```

**Example 12.4:** What are the applications of binary heap?

**Answer:**

- Dijkstra algorithm
- Prims algorithm
- Priority queue
- Can be used to solve problems such as:
  - K'th largest element in an array
  - Sort an almost sorted array
  - Merge K sorted array

**Example 12.5**

What is a **priority queue** and how can it be implemented?

**Answer:**

**Priority queue** is like a queue but it is more advanced. It has methods same as a queue. However, the main difference is that it keeps the value of higher priority at front, and the value of lowest priority at the back. Items are added from the rear and removed from the front. In priority queue elements are added in order. So, basically there is a priority associated with every element. Element of highest priority is removed first. Elements of equal priority are treated as per their order in queue.

## **Conclusion**

In last chapter you learnt about linked list which is a linear data structure. Trees on the other hand are hierachial data structures. Searching information on trees is faster than linked lists.

# CHAPTER 13

## Searching and Sorting

### Introduction

It is important to sort data in order before working on it because this makes searching information easy. In this chapter you will be learning about various algorithms that can be used to search and sort data and how these algorithms can be implemented using Python.

### Structure

- Sequential search
- Binary search
- Hash tables
- Bubble sort
- Insertion Sort
- Shell sort
- Quick sort

### Objective

In this chapter, you will learn about implementation of various algorithms for searching and sorting.

#### 13.1 Sequential search

In the chapter, based on Python operators we have seen that we can make use of membership operator ‘in’ to check if a value exists in a list or not.

```
>>> list1 = [1,2,3,4,5,6,7]
>>> 3 in list1
True
>>> 8 in list1
```

```
False
>>>
```

This is nothing but searching for an element in a list. We are going to look at how elements can be searched and how process efficiency can be improved. We start by learning about **sequential search**.

The simplest way of searching for an element in a sequence is to check every element one by one. If the element is found then the search ends, and the element is returned or else the search continues till the end of the sequence. This method of searching is known as **linear search** or **sequential search**. It follows a simple approach but it is quite inefficient way of searching for an element because we move from one element to the other right from beginning to end and if the element is not present in the sequence we would not know about it till we reach the last element.

Look For  
Number 2

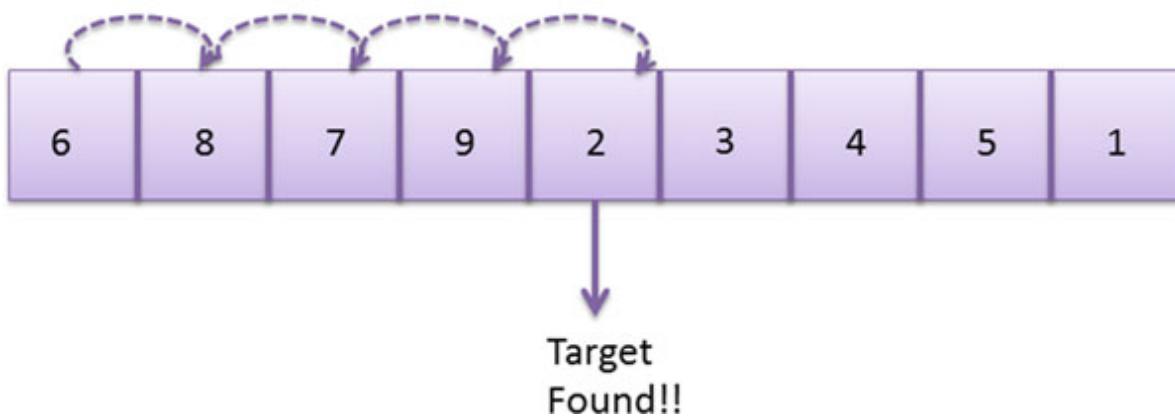


Figure 13.1

## Implementation of sequential search

The sequential search can be implemented as follows:

- The function takes two values: `seq_list` which is the list and `target_num` which is the number to search in the list.
- We set `search_flag = 0` if the target number is found in the list we will set the `search_flag` to 1 else we let it be 0.

- We iterate through the list comparing each element in the list with the `target_num`.
- If a match is found we print a message and update the `search_flag` to 1.
- After the “for” loop if the `search_flag` is still 0 then it means that the number was not found.

### **Code:**

```
def sequential_search(seq_list, target_num):
 search_flag = 0
 for i in range(len(seq_list)):
 if seq_list[i] == target_num:
 print("Found the target number ", target_num, " at index", i,".")
 search_flag = 1;

 if search_flag == 0:
 print("Target Number Does Not Exist. Search Unsuccessful.")
```

### **Execution:**

```
seq_list = [1,2,3,4,5,6,7,8,2,9,10,11,12,13,14,15,16]
target_num = input("Please enter the target number : ")
sequential_search(seq_list, int(target_num))
```

### **Output 1**

```
Please enter the target number : 5
Found the target number 5 at index 4 .
```

### **Output 2**

```
Please enter the target number : 2
Found the target number 2 at index 1 .
Found the target number 2 at index 8 .
```

### **Output 3**

```
Please enter the target number : 87
Target Number Does Not Exist. Search Unsuccessful.
```

## **Implementation of sequential search for an ordered list**

When elements in a list are sorted, then many times there may not be the need to scan the entire list. The moment we reach an element that has a value greater than the target number, we know that we need not go any further.

## Step 1

Define a function `sequential_search()` that takes two arguments – a list (`seq_list`) and the number that we are looking for (`target_num`).

```
def sequential_search(seq_list, target_num):
```

## Step 2

The first thing we do is set define a flag (`search_flag`) and set it to “`False`” or “`0`” value. The flag is set to “`True`” or “`1`” if the element is found. So, after traversing though the list if the `search_flag` value is still “`False`” or “`0`”, we would know that the number that we are looking for does not exist in the list.

```
def sequential_search(seq_list, target_num):
 search_flag = 0
```

## Step 3

Now, it's time to check the elements one by one so, we define the for loop:

```
def sequential_search(seq_list, target_num):
 search_flag = 0
 for i in range(len(seq_list)):
```

## Step 4

We now define how the elements are compared. Since it is an ordered list for every “`i`” in `seq_list` we have to check if `i > target_num`. If yes, then it means that there is no point moving further as it is an ordered list, and we have reached an element that is greater than the number that we are looking for. However if `seq_list[i] == target_num` then, the search is successful and we can set the `search_flag` to `1`.

```
def sequential_search(seq_list, target_num):
 search_flag = 0
 for i in range(len(seq_list)):
 if seq_list[i] > target_num:
 print("search no further.")
```

```
 break;
 elif seq_list[i] == target_num:
 print("Found the target number ", target_num, " at index", i,".")
 search_flag = 1
```

## Step 5

After the for loop has been executed if the value of `search_flag` is still 0 then a message stating that the target number was not found must be displayed.

### Code:

```
def sequential_search(seq_list, target_num):
 search_flag = 0
 for i in range(len(seq_list)):
 if seq_list[i] > target_num:
 print("search no further.")
 break;
 elif seq_list[i] == target_num:
 print("Found the target number ", target_num, " at index", i,".")
 search_flag = 1

 if search_flag == 0:
 print("Target Number Does Not Exist. Search Unsuccessful.")
```

### Execution:

```
seq_list = [1,2,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
target_num = input("Please enter the target number : ")
sequential_search(seq_list, int(target_num))
```

### Output 1

```
Please enter the target number : 2
Found the target number 2 at index 1 .
Found the target number 2 at index 2 .
search no further.
>>>
```

### Output 2

```
Please enter the target number : 8
Found the target number 8 at index 8 .
search no further.
>>>
```

## Output 3

```
Please enter the target number : 89
Target Number Does Not Exist. Search Unsuccessful.
>>>
```

## 13.2 Binary search

**Binary search** is used to locate a target value from a sorted list. The search begins from the centre of the sequence. The element present at the centre is not equal to the target number it is compared with the target number. If the target number is greater than the element at the centre then, it means that we need to search for the number in the right half of the list and the left half need not be touched. Similarly, if the target number is less than the element present in the centre then we will have to direct our search efforts towards the left. This process is repeated till the search is completed. The beauty of binary search is that in every search operation the sequence is cut into half and focus is shifted to only that half that has chances of having the value.

|             |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------------|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Sorted list | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------------|---|---|---|---|---|---|---|---|---|----|----|----|----|

Target number = 5

Middle value = 7

Is 7 == 5 ? No

Is 7 > 5? Yes, right side of the list is not required any more

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|

Middle value = 3

Is 3 == 5 ? No

Is 3 > 5? No, left side of the list is not required any more

|   |   |   |
|---|---|---|
| 4 | 5 | 6 |
|---|---|---|

Middle value = 5

Is 5 == 5 ? Yes

we have found the target Number

*Figure 13.2*

Implement binary search function

The binary search can be implemented in the following manner:

**Step 1:** Define the `binary_search` function. It takes four parameters:

- `sorted_list`: The input list that is in sorted form.
- `target_num`: The number that we are looking for.
- `starting_point`: The place from where we want to start searching, default value = 0.
- `end_point`: The end point for search, default value = None.

Note that the list will be split into half in every step so the starting and ending point may change in every search operation.

```
def binary_search(sorted_list, target_num, start_point=0, end_point= None):
```

**Step 2:** Do the following:

- Set the `search_flag` to “False”
- If the `end_point` is not provided, it would have the default value of “None”, set it to the length of the input list.

```
def binary_search(sorted_list, target_num, start_point=0, end_point= None):
 search_flag = False
 if end_point == None:
 end_point = len(sorted_list)-1
```

**Step 3:** Check, the `start_point` should be less than the end point. If that is true, do the following:

- Get midpoint index value: `mid_point = (end_point+start_point)//2`
- Check the value of the element at `mid_point`. Is it equal to the `target_num`?
  - If `sorted_list[mid_point] == target_num`?
  - Set `search_flag` to True
- If not check if value at `mid_point` is greater than `target_num`:
  - `sorted_list[mid_point] > target_num`

- If yes, then we can discard the right side of the list now we can repeat search from beginning to `mid_point-1` value. Set end point to `mid_point - 1`. The starting point can remain the same (0).
- The function should now call itself with:
  - `sorted_list`: Same as before
  - `target_num`: Same as before
  - `starting_point`: Same as before
  - `end_point`: `mid_point - 1`
- If not check if value at `mid_point` is lesser than `target_num`:
  - `sorted_list[mid_point] < target_num`
  - If yes, then left side of the list is not required. We can repeat search from `mid_point+1` to end of the list. Set starting point to `mid_point+1`. The `ending_point` can remain the same.
  - The function should now call itself with:
    - `sorted_list`: Same as before
    - `target_num`: Same as before
    - `starting_point`: `mid_point+1`
    - `end_point`: Same as before
- If at the end of this procedure the `search_flag` is still set to “`False`”, then it means that the value does not exist in the list.

## Code:

```
def binary_search(sorted_list, target_num, start_point=0, end_point= None):
 search_flag = False
 if end_point == None:
 end_point = len(sorted_list)-1
 if start_point < end_point:
 mid_point = (end_point+start_point)//2
 if sorted_list[mid_point] == target_num:
 search_flag = True
 print(target_num," Exists in the list at ",sorted_list.index(target_num))
 elif sorted_list[mid_point] > target_num:
 end_point = mid_point-1
 binary_search(sorted_list, target_num,start_point, end_point)
 elif sorted_list[mid_point] < target_num:
 start_point = mid_point+1
```

```
binary_search(sorted_list, target_num, start_point, end_point)
elif not search_flag:
 print(target_num, " Value does not exist")
```

## Execution:

```
sorted_list=[1,2,3,4,5,6,7,8,9,10,11,12,13]
binary_search(sorted_list,14)
binary_search(sorted_list,0)
binary_search(sorted_list,5)
```

## Output:

```
14 Value does not exist
0 Value does not exist
5 Exists in the list at 4
```

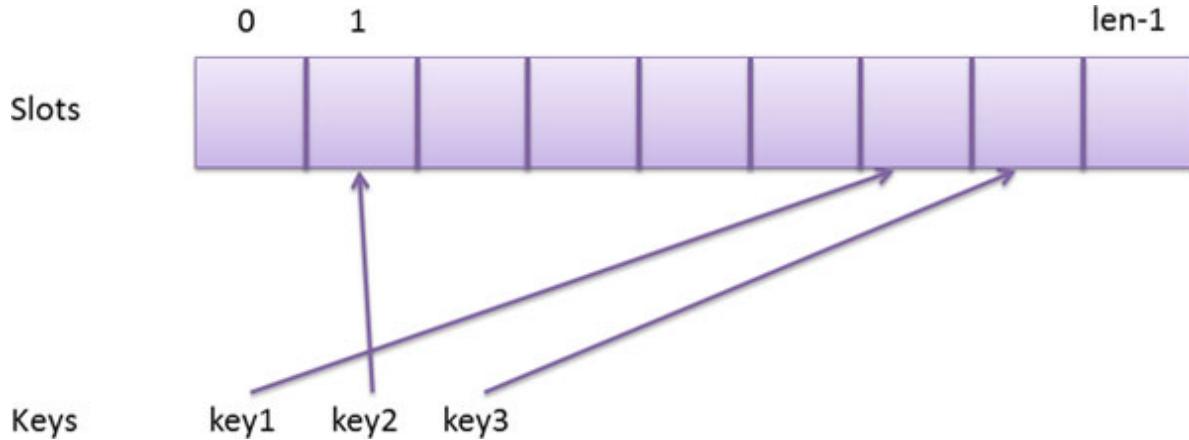
**Note:** Some of the sections below refer to time complexity. If you do not have knowledge of B-O notation then please refer the [Appendix](#) given at the end of the book.

## 13.3 Hash Tables

Hash tables are data structures where a hash function is used to generate the index or address value for a data element. It is used to implement an associative array which can map keys to values. The benefit of this is that it allows us to access data faster as the index value behaves as a key for data value. Hash tables store data in key-value pairs but the data is generated using the hash function. In Python the Hash Tables are nothing but dictionary data type. Keys in the dictionary are generated using a hash function and the order of data elements in Dictionary is not fixed. We have already learnt about various functions that can be used to access a dictionary object but what we actually aim at learning here is how hash tables are actually implemented.

We know that by binary search trees we can achieve time complexity of  $O(\log n)$  for various operations. The question that arises here is that can search operations be made faster? Is it possible to reach time complexity of  $O(1)$ ? This is precisely why hash tables came into existence. Like in a list or an array if the index is known, the time complexity for search operation can become  $O(1)$ . Similarly, if data is stored in key value pairs, the result can

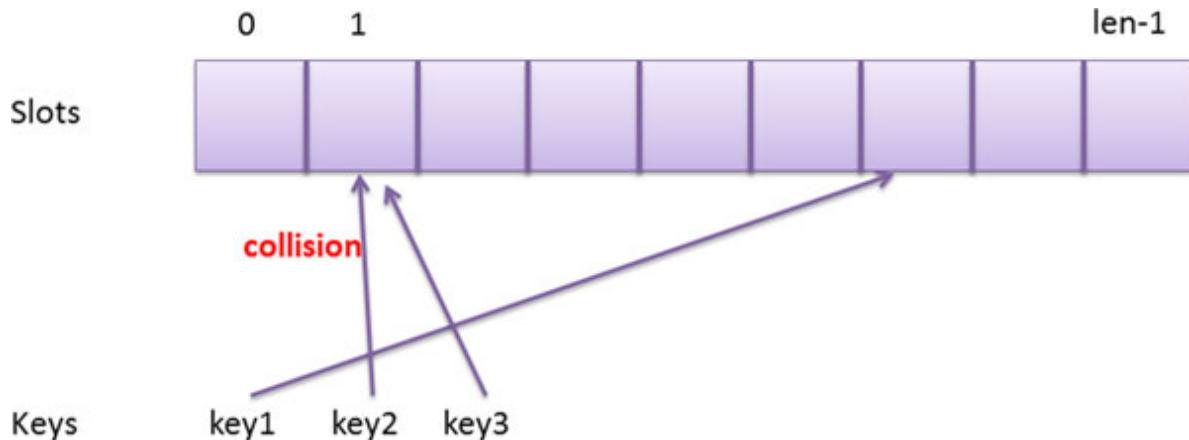
be retrieved faster. So, we have keys and we have slots available where the values can be placed. If we are able to establish a relationship between the slots and the key it would be easier for to retrieve the value at a fast rate. Look at the following figure:



*Figure 13.3*

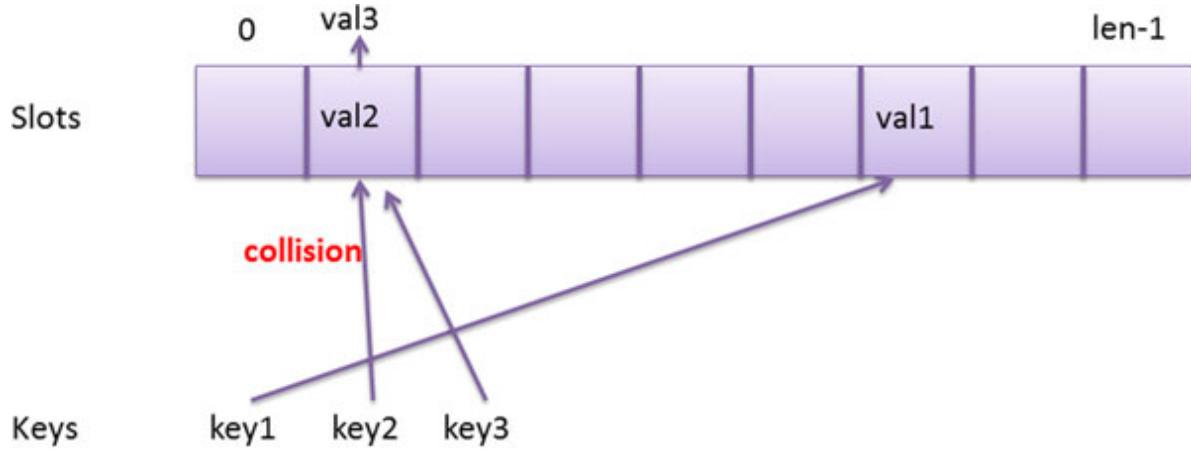
The key value is not always a non negative integer, it can be a string also, where as the array has index starting from `0` to `length_of_array -1`. So there is a need to do prehashing in order to match the string keys to indexes. So, for every key there is a need to find an index in array where the corresponding value can be placed. In order to do this we will have to create a `hash()` function that can map a key of any type to a random array index.

During this process there are chances of collision. Collision is when we map two keys to the same index as shown in the following figure:



*Figure 13.4*

To resolve collision we can use chaining. Chaining is when values are stored in the same slot with the help of a linked list as shown in the following figure:

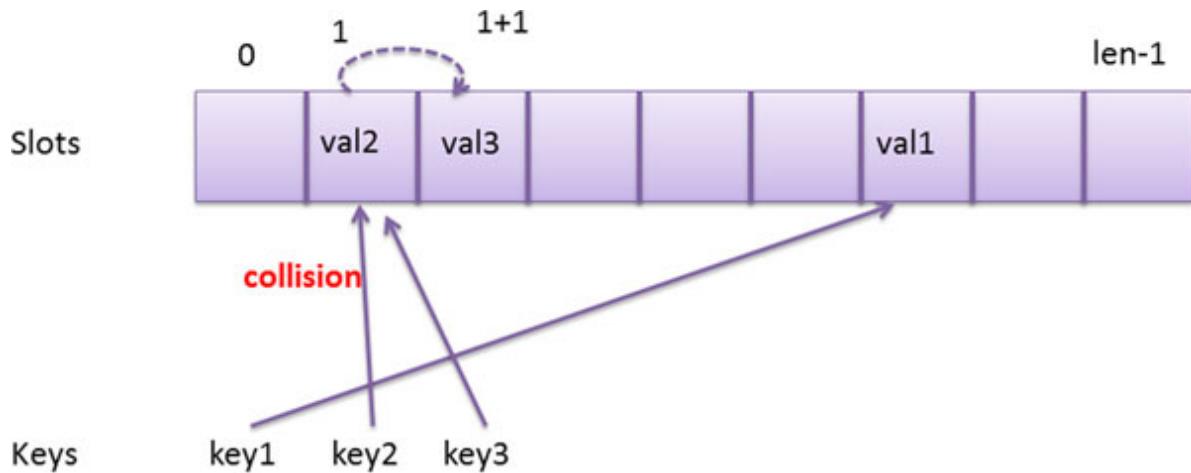


### CHAINING

*Figure 13.5*

However, there can be cases of more than one collision for the same spot and considering the worst case scenario where there is a need to insert all values as elements of a linked list it can be a tough situation that will have a severe impact on the time complexity. Worst case scenario will be if we land up placing all values as linked list element at the same index.

To avoid this scenario we can consider the process of **open addressing**. **Open addressing** is a process of creating a new address. Consider a case where if there is a collision we increment the index by 1 and place the value there as follows, there is collision while placing **val3**, as **val2** already exists at index 1. So, the index value is incremented by 1 ( $1+1 = 2$ ) and **val3** is placed at index 2.



### OPEN ADDRESSING

*Figure 13.6*

Had there been any other value at index 2 then the index would have incremented again and `val3` could be placed at index 3. Which means that this process of incrementing index is continued till an empty slot is spotted. This is called **Linear probing**. **Quadratic probing** on the other hand increments by two times the index value. So, the search for empty slot is done at a distance of 1,2,4,8, and so on. Rehashing is the process of hashing the result obtained again to find an empty slot.

The purpose of the hash function is to calculate an index from which the right value can be found therefore its job would be:

1. To distribute the keys uniformly in the array.
2. If  $n$  is the number of keys and  $m$  is the size of an array, the `hash() = n%m (modulo operator)` in case we use integers as keys.
  - a. Prefer to use prime numbers both for array and the hash function for uniform distribution
  - b. For string keys you can calculate the ascii value of each character, and add them up and make modulo operator on them

In many scenarios, hash tables prove to be more efficient than the search trees, and are often used in caches, databases, and sets.

Important points:

1. You can avoid clustering by using prime numbers.

2. Number of entries divided by the size of the array is called load factor.
3. If the load factor increases the number of collisions will increase. This will reduce the performance of the hash table.
4. Resize the table when load factor exceeds the given threshold. However, this would be an expensive option as the hashes of the values entered will change whenever resizing is done and this can take  $O(n)$  to complete. Hence, dynamic size array may be inappropriate for real time scenarios.

The purpose of a hash function is to map the values or entries into the slots that are available in a hash table. So, for every entry the hash function will compute an integer value that would be in the range of 0 to  $m-1$  where  $m$  is the length of the array.

## Implementation of Remainder Hash Function

The remainder hash function calculates the index value by taking one item at a time from collection. It is then divided by the size of the array, and the remainder is returned.

$h(\text{item}) = \text{item} \% m$ , where  $m = \text{size of the array}$

Let's consider the following array:[18, 12, 45, 34, 89, 4]

The above array is of size 8.

| Item | Calculation= item%m | Result |
|------|---------------------|--------|
| 18   | 18%8                | 2      |
| 12   | 12%8                | 4      |
| 45   | 45%8                | 5      |
| 34   | 34%8                | 2      |
| 89   | 89%8                | 1      |
| 4    | 4%8                 | 4      |

**Drawback:** You can see here that 18 and 34 have same hash value of 2 and 12 and 4 have the same hash value of 4. This is a case of **collision** as a result when you execute the program, values 18 and 12 are replaced by 34 and 4, and you will not find these values in the hash table.

Let's have a look at the implementation:

**Step 1:** Define the hash function that takes a list and size of array as input.

```
def hash(list_items, size):
```

```
 def hash(list_items, size):
```

**Step 2:** Do the following:

- Create an empty list.
- Now populate this key from the numbers 0 to size mention. This example takes a list of 8 elements so we are creating a list [0, 1, 2, 3, 4, 5, 6, 7].
- Convert this list to dict using `fromkeys()`. We should get a dictionary object of form `{0: None, 1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None}`. This value is assigned to `hash_table`.

```
def hash(list_items, size):
 temp_list =[]
 for i in range(size):
 temp_list.append(i)
 hash_table = dict.fromkeys(temp_list)
```

**Step 3**

- Now iterate through the list.
- Calculate the index for every item by calculating `item%size`.
- For the key value in the `hash_table = index`, insert the item.

**Code:**

```
def hash(list_items, size):
 temp_list =[]
 for i in range(size):
 temp_list.append(i)
 print(temp_list)
 hash_table = dict.fromkeys(temp_list)
 print(hash_table)
 for item in list_items:
 i = item%size
 hash_table[i] = item
 print("value of hash table is : ",hash_table)
```

## Execution:

```
list_items = [18,12,45,34,89,4]
hash(list_items, 8)
```

## Output:

```
[0, 1, 2, 3, 4, 5, 6, 7]
{0: None, 1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None}
value of hash table is : {0: None, 1: 89, 2: 34, 3: None, 4: 4, 5: 45, 6: None, 7:
None}
>>>
```

## Folding hash function

Folding hash function is a technique used to avoid collisions while hashing. The items are divided into equal-size pieces, they are added together and then the slot value is calculated using the same hash function ( $\text{item \% size}$ ).

Suppose, we have a phone list as follows:

```
phone_list = [4567774321, 4567775514, 9851742433, 4368884732]
```

We convert every number to string, then each string is converted to a list and then each list is appended to another list and we get the following result:

```
[['4', '5', '6', '7', '7', '4', '3', '2', '1'], ['4', '5', '6', '7', '7', '7', '5',
'5', '1', '4'], ['9', '8', '5', '1', '7', '4', '2', '4', '3', '3'], ['4', '3', '6', '8',
'8', '8', '4', '7', '3', '2']]
```

Now from this new list, we take one list item one by one, for every item we concatenate two characters convert them to integer, and then concatenate next to characters convert them to integer and add the two values and continue this till we have added all elements. The calculation will be something like this:

```
ssss['4', '5', '6', '7', '7', '4', '3', '2', '1']
```

1. items = 4 5

string val = 45

integer value = 45

hash value= 45

2. items = 6 7

string val = 67  
integer value = 67  
hash value= 45+67 =112

3. items = 7 7  
string val = 77  
integer value = 77  
hash value= 112+77 = 189

4. items = 4 3  
string val = 43  
integer value = 43  
hash value= 189+43 = 232

5. items = 2 1  
string val = 21  
integer value = 21  
hash value= 232+21 = 253

Similarly,

[‘4’, ‘5’, ‘6’, ‘7’, ‘7’, ‘5’, ‘5’, ‘1’, ‘4’] will have hash value of 511.  
[‘9’, ‘8’, ‘5’, ‘1’, ‘7’, ‘4’, ‘2’, ‘4’, ‘3’, ‘3’] will have hash value of 791.  
[‘4’, ‘3’, ‘6’, ‘8’, ‘8’, ‘8’, ‘4’, ‘7’, ‘3’, ‘2’] will have a hash value of 1069.

We now call the hash function for [253,511,791,1069] for size 11.

| Item | Calculation= item%m | Result |
|------|---------------------|--------|
| 253  | 253%11              | 0      |
| 511  | 511%11              | 5      |
| 791  | 791%11              | 10     |
| 1069 | 1069%11             | 2      |

So, the result we get is:

{0: 253, 1: None, 2: 1069, 3: None, 4: None, 5: 511, 6: None, 7: None, 8: None, 9: None, 10: 791}

## Implementation

Let's look at the execution statements for this program:

```
phone_list = [4567774321, 4567775514, 9851742433, 4368884732]
str_phone_values = convert_to_string(phone_list)
folded_value = folding_hash(str_phone_values)
folding_hash_table = hash(folded_value,11)
print(folding_hash_table)
```

1. A list of phone numbers is defined: `phone_list = [4567774321, 4567775514, 9851742433, 4368884732]`
2. The next statement "`str_phone_values = convert_to_string(phone_list)`" calls a function `convert_to_string()` and passes the `phone_list` as argument. The function in turn returns a list of lists. The function takes one phone number at a time converts it to a list and adds to new list. So, we get the output as: `[[‘4’, ‘5’, ‘6’, ‘7’, ‘7’, ‘7’, ‘4’, ‘3’, ‘2’, ‘1’], [‘4’, ‘5’, ‘6’, ‘7’, ‘7’, ‘5’, ‘5’, ‘1’, ‘4’], [‘9’, ‘8’, ‘5’, ‘1’, ‘7’, ‘4’, ‘2’, ‘4’, ‘3’, ‘3’], [‘4’, ‘3’, ‘6’, ‘8’, ‘8’, ‘4’, ‘7’, ‘3’, ‘2’]]`. The following steps are involved in this function:
  - a. Define two lists a `phone_list[]`
  - b. For elements in `phone_list`, take every element that is, the phone number one by one and:
    - i. Convert the phone number to string: `temp_string = str(i)`.
    - ii. Convert each string to list: `temp_list = list(temp_string)`.
    - iii. Append the list obtained to the `phone_list` defined in previous step.
    - iv. Return the `phone_list` and assign values to `str_phone_values`.

```
def convert_to_string(input_list):
 phone_list=[]
 for i in input_list:
 temp_string = str(i)
 temp_list = list(temp_string)
 phone_list.append(temp_list)
 return phone_list
```

3. The list `str_phone_values` is passed on to `folding_hash()`. This method takes a list as input.

- a. It will take each `phone_list` element which is also a list.
- b. Take one list item one by one.
- c. For every item concatenate first two characters convert them to integer and then concatenate next to characters convert them to integer and add the two values.
- d. Pop the first two elements from the list.
- e. Repeat c and d till we have added all elements.
- f. The function returns a list of hash values.

```
def folding_hash(input_list):
 hash_final = []
 while len(input_list) > 0:
 hash_val = 0
 for element in input_list:
 while len(element) > 1:
 string1 = element[0]
 string2 = element[1]
 str_combine = string1 + string2
 int_combine = int(str_combine)
 hash_val += int_combine
 element.pop(0)
 element.pop(0)
 if len(element) > 0:
 hash_val += element[0]
 else:
 pass
 hash_final.append(hash_val)
 return hash_final
```

4. Call hash function for size 11. The code for the hash function is same.

```
def hash(list_items, size):
 temp_list = []
 for i in range(size):
 temp_list.append(i)
 hash_table = dict.fromkeys(temp_list)
 for item in list_items:
 i = item%size
 hash_table[i] = item
 return hash_table
```

**Code:**

```

def hash(list_items, size):
 temp_list = []
 for i in range(size):
 temp_list.append(i)
 hash_table = dict.fromkeys(temp_list)
 for item in list_items:
 i = item%size
 hash_table[i] = item
 return hash_table
def convert_to_string(input_list):
 phone_list = []
 for i in input_list:
 temp_string = str(i)
 temp_list = list(temp_string)
 phone_list.append(temp_list)
 return phone_list
def folding_hash(input_list):
 hash_final = []
 while len(input_list) > 0:
 hash_val = 0
 for element in input_list:
 while len(element) > 1:
 string1 = element[0]
 string2 = element[1]
 str_combine = string1 + string2
 int_combine = int(str_combine)
 hash_val += int_combine
 element.pop(0)
 element.pop(0)
 if len(element) > 0:
 hash_val += element[0]
 else:
 pass
 hash_final.append(hash_val)
 return hash_final

```

## Execution:

```

phone_list = [4567774321, 4567775514, 9851742433, 4368884732]
str_phone_values = convert_to_string(phone_list)
folded_value = folding_hash(str_phone_values)
folding_hash_table = hash(folded_value, 11)
print(folding_hash_table)

```

## Output:

```
{0: 253, 1: None, 2: 1069, 3: None, 4: None, 5: 511, 6: None, 7: None, 8: None, 9: None, 10: 791}
```

In order to store phone numbers at the index, we slightly change the `hash()` function;

1. The `hash()` function will take one more parameter : `phone_list`
2. After calculating the index the corresponding element from the `phone_list` is saved instead of the `folded_value`.

```
def hash(list_items, phone_list, size):
 temp_list = []
 for i in range(size):
 temp_list.append(i)
 hash_table = dict.fromkeys(temp_list)
 for i in range(len(list_items)):
 hash_index = list_items[i] % size
 hash_table[hash_index] = phone_list[i]
 return hash_table
```

## Final Code

```
def hash(list_items, phone_list, size):
 temp_list = []
 for i in range(size):
 temp_list.append(i)
 hash_table = dict.fromkeys(temp_list)
 for i in range(len(list_items)):
 hash_index = list_items[i] % size
 hash_table[hash_index] = phone_list[i]
 return hash_table

def convert_to_string(input_list):
 phone_list = []
 for i in input_list:
 temp_string = str(i)
 temp_list = list(temp_string)
 phone_list.append(temp_list)
 return phone_list

def folding_hash(input_list):
 hash_final = []
 while len(input_list) > 0:
 hash_val = 0
 for element in input_list:
 while len(element) > 1:
 string1 = element[0]
 string2 = element[1]
 str_combine = string1 + string2
 int_combine = int(str_combine)
 hash_val += int_combine
 element.pop(0)
```

```

element.pop(0)
if len(element) > 0:
 hash_val += int(element[0])
else:
 pass
hash_final.append(hash_val)
return hash_final

```

## Execution:

```

phone_list = [4567774321, 4567775514, 9851742433, 4368884732]
str_phone_values = convert_to_string(phone_list)
folded_value = folding_hash(str_phone_values)
folding_hash_table = hash(folded_value, phone_list, 11)
print(folding_hash_table)

```

## Output:

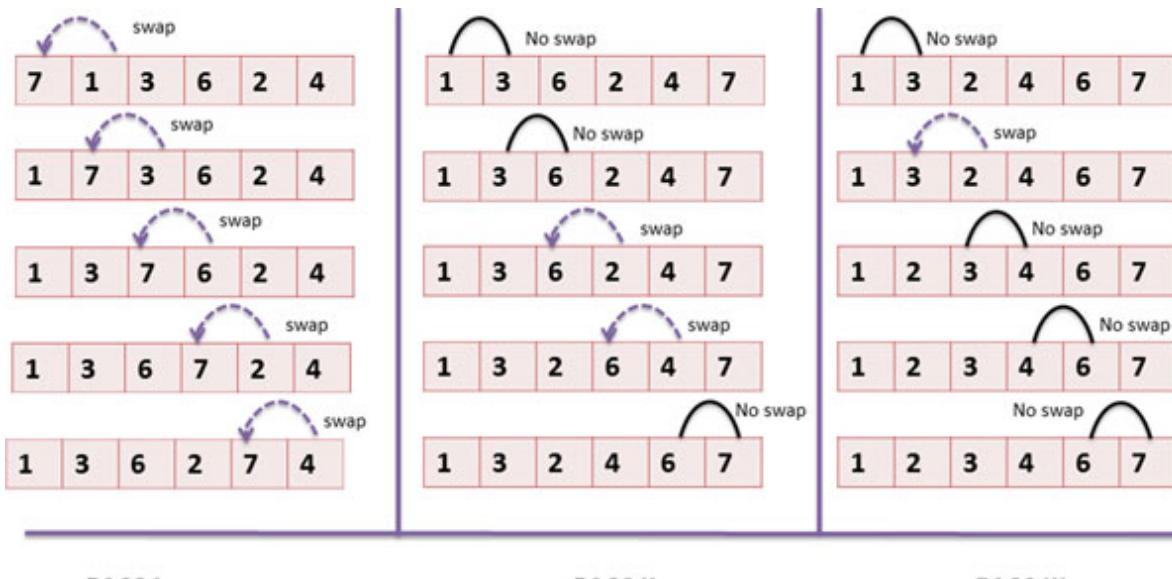
```

{0: 4567774321, 1: None, 2: 4368884732, 3: None, 4: None, 5: 4567775514, 6: None,
7: None, 8: None, 9: None, 10: 9851742433}

```

## 13.4 Bubble sort

**Bubble sort** is also known as **sinking sort** or **comparison sort**. In bubble sort each element is compared with the adjacent element, and the elements are swapped if they are found in wrong order. However this is a time consuming algorithm. It is simple but quite inefficient.



*Figure 13.7*

## Implementation of Bubble Sort

The code for a bubble sort algorithm is very simple.

### Step 1

Define the function for bubble sort. It would take the list that needs to be sorted as input.

```
def bubble_sort(input_list):
```

### Step 2

1. Set a loop **for i in range len(input\_list)**
  - a. Inside this for loop set another loop **for j in range len(input\_list)-i-1).**
  - b. For every i, in the nested loop value at index j is compared with value at index **j+1**. If the value at index **j+1** is smaller than the value at index **j** then the values are swapped.
  - c. After the for loop is over print the sorted list.

### Code:

```
def bubble_sort(input_list):
 for i in range(len(input_list)):
 for j in range(len(input_list)-i-1):
 if input_list[j]>input_list[j+1]:
 temp = input_list[j]
 input_list[j]=input_list[j+1]
 input_list[j+1]= temp
 print(input_list)
```

### Execution:

```
x = [7,1,3,6,2,4]
print("Executing Bubble sort for ",x)
bubble_sort(x)
y = [23,67,12,3,45,87,98,34]
print("Executing Bubble sort for ",y)
bubble_sort(y)
```

## **Output:**

```
Executing Bubble sort for [7, 1, 3, 6, 2, 4]
[1, 2, 3, 4, 6, 7]
Executing Bubble sort for [23, 67, 12, 3, 45, 87, 98, 34]
[3, 12, 23, 34, 45, 67, 87, 98]
```

## **13.5 Implementation of selection sort**

### **Answer:**

**Step 1:** Define the function for `selection_sort`. It would take the list that needs to be sorted as input.

```
def selection_sort(input_list):
```

### **Step 2:**

1. Set a loop `for i in range len(input_list)`
  - a. Inside this for loop set another loop `for j in range (i+1, len(input_list)-i-1)`
  - b. For every i, in the nested loop value at index j is compared with value at index i. If the value at index j is smaller than the value at index i then the values are swapped.
  - c. After the for loop is over print the sorted list.

### **Code:**

```
def selection_sort(input_list):
 for i in range(len(input_list)-1):
 for j in range(i+1, len(input_list)):
 if input_list[j] < input_list[i]:
 temp = input_list[j]
 input_list[j] = input_list[i]
 input_list[i] = temp
 print(input_list)
```

### **Execution:**

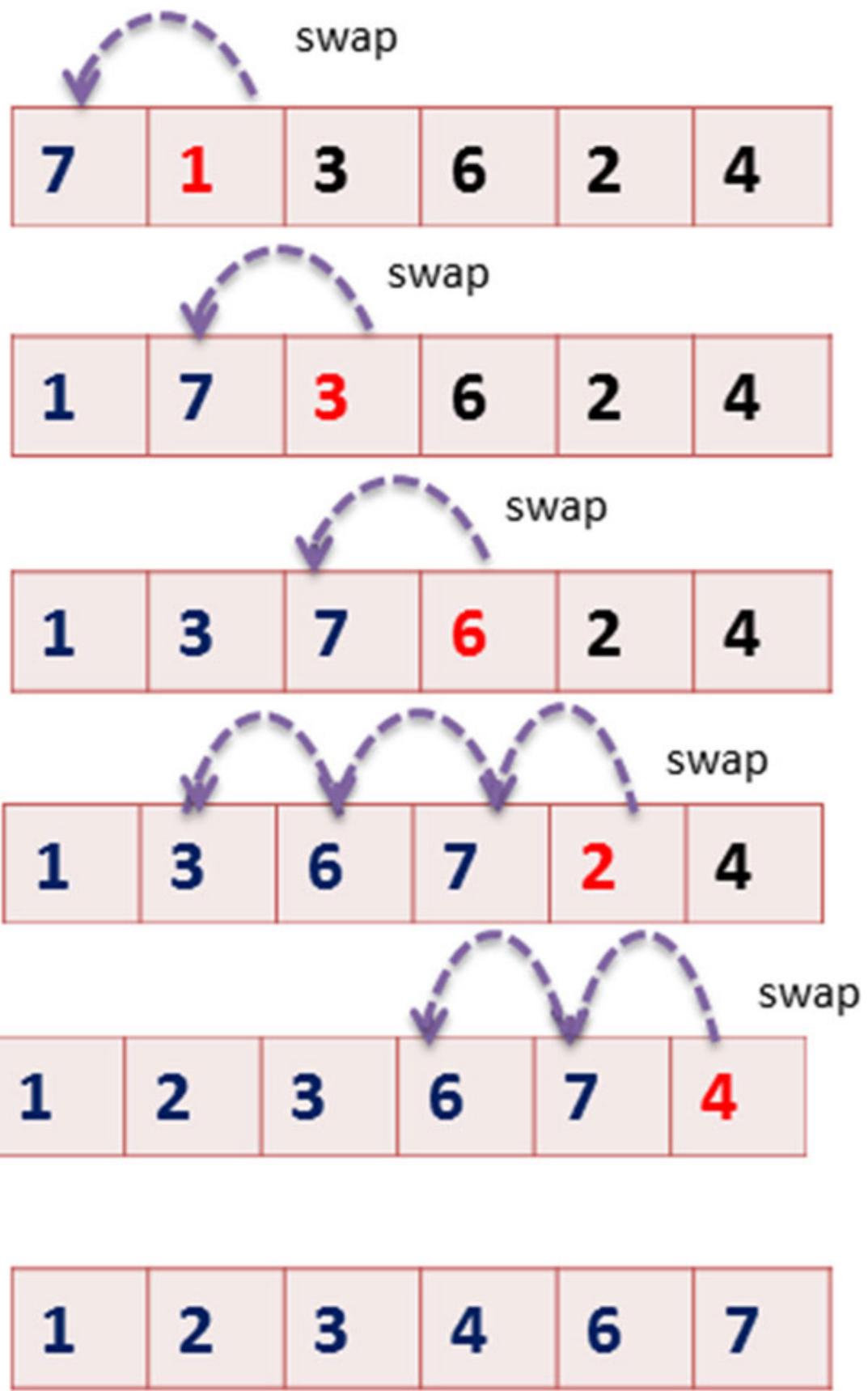
```
selection_sort([15,10,3,19,80,75])
selection_sort([5,9,80,65,71,24,15,10,3,19,85,75])
```

## **Output:**

```
[3, 10, 15, 19, 75, 80]
[3, 5, 9, 10, 15, 19, 24, 65, 71, 75, 80, 85]
>>>
```

## **13.6 Insertion sort**

In insertion sort, each element at position x is compared with elements located at position x-1 to position 0. If the element is found to be less than any of the values that it is compared to then the values are swapped. This process is repeated till the last element has been compared.



*Figure 13.8*

## Implementation of Insertion Sort

It is very easy to implement insertion sort:

Consider a list [7,1,3,6,2,4]

Let `indexi = i`

`indexj = indexi + 1`

| Indexi             | indexj                   | val[i] < val[j] | Swap | Change in index value                                                  |
|--------------------|--------------------------|-----------------|------|------------------------------------------------------------------------|
| 0                  | 1                        | 1<7 yes         | yes  | <code>indexi= indexi-1 = -1</code><br><code>indexj = indexj-1=0</code> |
| <b>Iteration 2</b> | <i>List: 1,7,3,6,2,4</i> |                 |      |                                                                        |
| 1                  | 2                        | 3<7 yes         | yes  | <code>indexi= indexi-1= 0</code><br><code>indexj = indexj-1 =1</code>  |
|                    | <i>List: 1,3,7,6,2,4</i> |                 |      |                                                                        |
| 0                  | 1                        | 3<1 no          | no   | <code>indexi= indexi-1= -1</code>                                      |
| <b>Iteration 3</b> | <i>List: 1,3,7,6,2,4</i> |                 |      |                                                                        |
| 2                  | 3                        | 6<7 yes         | yes  | <code>indexi= indexi-1= 1</code><br><code>indexj = indexj-1 =0</code>  |
|                    | <i>List: 1,3,6,7,2,4</i> |                 |      |                                                                        |
| 1                  | 2                        | 7<3             | no   | <code>indexi= indexi-1= 0</code>                                       |
| 0                  | 2                        | 7<1             | no   | <code>indexi= indexi-1= -1</code>                                      |
| <b>Iteration 4</b> | <i>List: 1,3,6,7,2,4</i> |                 |      |                                                                        |
| 3                  | 4                        | 2<7 yes         | yes  | <code>indexi= indexi-1= 2</code><br><code>indexj = indexj-1 =1</code>  |
|                    | <i>List: 1,3,6,2,7,4</i> |                 |      |                                                                        |
| 2                  | 3                        | 2<6 yes         | yes  | <code>indexi= indexi-1= 1</code><br><code>indexj = indexj-1 =2</code>  |

|                    |                          |          |     |                                                               |
|--------------------|--------------------------|----------|-----|---------------------------------------------------------------|
|                    | <i>List: 1,3,2,6,7,4</i> |          |     |                                                               |
| 1                  | 2                        | 2<3 yes  | yes | <code>indexi= indexi-1= 0<br/>indexj = indexj-1<br/>=1</code> |
|                    | <i>List: 1,2,3,6,7,4</i> |          |     |                                                               |
| 0                  | 1                        | 2<1 no   | no  | <code>indexi= indexi-1=-1</code>                              |
| <b>Iteration 5</b> | <i>List: 1,2,3,6,7,4</i> |          |     |                                                               |
| 4                  | 5                        | 4<7 yes  | yes | <code>indexi= indexi-1= 3<br/>indexj = indexj-1<br/>=4</code> |
|                    | <i>List: 1,2,3,6,4,7</i> |          |     |                                                               |
| 3                  | 4                        | 4<6 yes  | yes | <code>indexi= indexi-1= 2<br/>indexj = indexj-1<br/>=1</code> |
|                    | <i>List: 1,2,3,4,6,7</i> |          |     |                                                               |
| 2                  | 3                        | 4<3 no   | No  | <code>indexi= indexi-1=1</code>                               |
| 1                  | 3                        | 4<2 no   | No  | <code>indexi= indexi-1=0</code>                               |
| 0                  | 3                        | 4 < 1 no | No  | <code>indexi= indexi-1=-1</code>                              |

**Table 11.1**

## Step 1

Define the `insert_sort()` function. It will take `input_list` as input.

```
def insertion_sort(input_list):
```

## Step 2

for i in range(`input_list`-1), set `indexi = i`, `indexj = i+1`

```
for i in range(len(input_list)-1):
 indexi = i
 indexj = i+1
```

## Step 3

set while loop, condition `indexi>=0`

- if `input_list[indexi]>input_list[indexj]`
  - swap values of `input_list[indexi]` and `input_list[indexj]`
  - set `indexi = indexi - 1`
  - set `indexj = indexj - 1`
- else
  - set `indexi = indexi - 1`

```

while indexi >= 0:
 if input_list[indexi]>input_list[indexj]:
 print("swapping")
 temp = input_list[indexi]
 input_list[indexi] = input_list[indexj]
 input_list[indexj] = temp
 indexi = indexi - 1
 indexj = indexj - 1
 else:
 indexi = indexi - 1

```

## Step 4

Print updated list

**Code:**

```

def insertion_sort(input_list):

 for i in range(len(input_list)-1):
 indexi = i
 indexj = i+1
 print("indexi = ", indexi)
 print("indexj = ", indexj)
 while indexi >= 0:
 if input_list[indexi]>input_list[indexj]:
 print("swapping")
 temp = input_list[indexi]
 input_list[indexi] = input_list[indexj]
 input_list[indexj] = temp
 indexi = indexi - 1
 indexj = indexj - 1

 else:
 indexi = indexi - 1
 print("list update:",input_list)
 print("final list = ", input_list)

```

## **Execution:**

```
insertion_sort([9,5,4,6,7,8,2])
```

## **Output:**

```
indexi = 0
indexj = 1
swapping
list update: [5, 9, 4, 6, 7, 8, 2]
indexi = 1
indexj = 2
swapping
swapping
list update: [4, 5, 9, 6, 7, 8, 2]
indexi = 2
indexj = 3
swapping
list update: [4, 5, 6, 9, 7, 8, 2]
indexi = 3
indexj = 4
swapping
list update: [4, 5, 6, 7, 9, 8, 2]
indexi = 4
indexj = 5
swapping
list update: [4, 5, 6, 7, 8, 9, 2]
indexi = 5
indexj = 6
swapping
swapping
swapping
swapping
swapping
list update: [2, 4, 5, 6, 7, 8, 9]
final list = [2, 4, 5, 6, 7, 8, 9]
```

## **13.7 Shell sort**

- Shell sort is a very efficient sorting algorithm.
- Based on insertion sort.
- Implements insertion sort on widely spread elements at first and then in each step the space or interval is narrowed down.
- Great for medium size data set.

- Worst case time complexity:  $O(n)$
- Worst case space complexity :  $O(n)$

I. Consider a list:  $[10, 30, 11, 4, 36, 31, 15, 1]$

Size of the list,  $n = 8$

Divide  $n/2 = 4$ , let this value be named k

Consider every kth (in this case 4th) element and sort them in right order.

|        |                                     |           |    |   |    |    |    |   |
|--------|-------------------------------------|-----------|----|---|----|----|----|---|
| List   | 10                                  | 30        | 11 | 4 | 36 | 31 | 15 | 1 |
| n      | 8                                   |           |    |   |    |    |    |   |
| pass 1 | k                                   | $n/2 = 4$ |    |   |    |    |    |   |
|        | 10                                  | 30        | 11 | 4 | 36 | 31 | 15 | 1 |
| step 1 | compare 10 and 36, no swap required |           |    |   |    |    |    |   |
|        | 10                                  | 30        | 11 | 4 | 36 | 31 | 15 | 1 |
| step 2 | compare 30 and 31, no swap required |           |    |   |    |    |    |   |
|        | 10                                  | 30        | 11 | 4 | 36 | 31 | 15 | 1 |
| step 3 | compare 11 and 15, no swap required |           |    |   |    |    |    |   |
|        | 10                                  | 30        | 11 | 4 | 36 | 31 | 15 | 1 |
| step 4 | compare 4 and 1, swap values        |           |    |   |    |    |    |   |
|        | 10                                  | 30        | 11 | 1 | 36 | 31 | 15 | 4 |

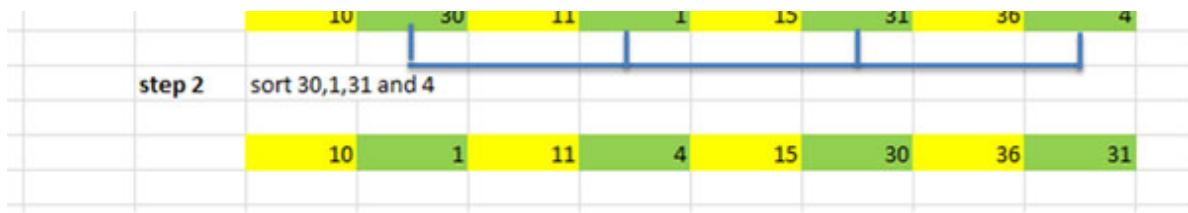
Figure 13.9

II. do the following:

$$K = k/2 = 4/2 = 2$$

Consider every kth element and sort the order.

|              |                      |           |    |    |    |    |    |    |   |
|--------------|----------------------|-----------|----|----|----|----|----|----|---|
| <b>k = 4</b> | List                 | 10        | 30 | 11 | 1  | 36 | 31 | 15 | 4 |
|              |                      |           |    |    |    |    |    |    |   |
| pass 2       | k                    | $k/2 = 2$ |    |    |    |    |    |    |   |
|              | 10                   | 30        | 11 | 1  | 36 | 31 | 15 | 4  |   |
| step 1       | sort 10,11,36 and 15 |           |    |    |    |    |    |    |   |
|              | 10                   | 30        | 11 | 1  | 36 | 31 | 15 | 4  |   |

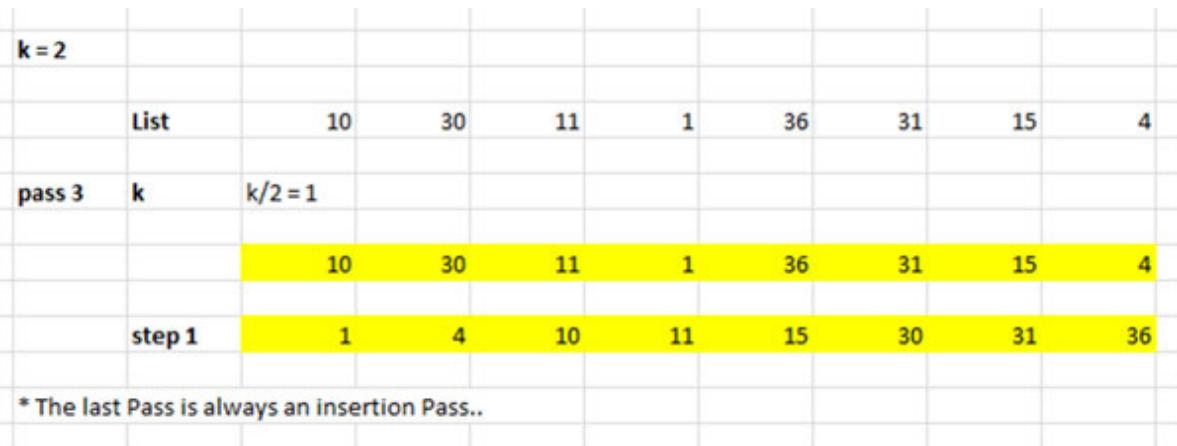


*Figure 13.10*

III. Do the following:

$$k = k/2 = 2/2 = 1$$

This is the last pass and will always be an insertion pass.



*Figure 13.11*

## Implementation of shell sort algorithm

The shell sort algorithm can be implemented as follows:

### Step 1:

Define the `shell_sort()` function to sort the list. It will take the `list(input_list)` that needs to be sorted as input value.

```
def shell_sort(input_list):
```

---

## Step 2:

Calculate size,  $n = \text{len}(\text{input\_list})$

Number of steps for the while loop,  $k = n/2$

```
def shell_sort(input_list):
 n = len(input_list)
 k = n//2
```

## Step 3:

- While  $k > 0$ :
  - for  $j$  in range 0, size of input list
    - for  $i$  in range  $(k, n)$
    - if the value of element at  $i$  is less than the element located at index  $i-k$ , then swap the two values
  - set  $k = k//2$

```
while k > 0:
 for j in range(n):
 for i in range(k, n):
 temp = input_list[i]
 if input_list[i] < input_list[i-k]:
 input_list[i] = input_list[i-k]
 input_list[i-k] = temp
 k = k//2
```

## Step 4:

Print the value of the sorted list.

## Code:

```
def shell_sort(input_list):
 # step 1 Calculate the size n = len(input_list)
 n = len(input_list)
 # step 2 Calculate the gap k = n/2
 k = n//2

 #step 3 create a loop for sorting

 while k > 0:
 for j in range(n):
```

```

for i in range(k,n):
 temp = input_list[i]
 if input_list[i] < input_list[i-k]:
 input_list[i] = input_list[i-k]
 input_list[i-k] = temp
 k = k//2
print(input_list)

```

### Execution:

```
shell_sort([10,9,89,30,11,1,36,31,15,4])
```

### Output:

```
[[1, 4, 9, 10, 11, 15, 30, 31, 36, 89]]
```

## 13.8 Quick Sort

- In quicksort we make use of a pivot to compare numbers.
- All items smaller than the pivot are moved to its left side and all items larger than the pivot are moved to the right side. This would provide left partition that has all values less than the pivot and right partition having all values greater than the pivot.
- Let's take a list of 9 numbers:  $[15, 39, 4, 20, 50, 6, 28, 2, 13]$ .
- The last element '13' is taken as the pivot.
- We take the first element '15' as the left mark and the second last element '2' as the right mark.
- If leftmark > pivot and ightmark < pivot then swap left mark and right mark and increment leftmark by 1 and decrement rightmak by 1.
- If leftmark > pivot and rightmark > pivot then only decrement the rightmark.
- Same way if leftmark < pivot and rightmark < pivot then only increment the leftmark.
- If leftmark < pivot and rightmark > pivot, increment leftmark by 1 and decrement right mark by 1.
- When left mark and rightmark meet at one element, swap that element with the pivot.

I.

| quicksort | 15 | 39 | 4 | 20 | 50 | 6  | 28 | 2  | 13    | 15>13      | yes  | swap    |
|-----------|----|----|---|----|----|----|----|----|-------|------------|------|---------|
|           | *  |    |   |    |    |    |    | *  | pivot | 2<13       | yes  |         |
|           | 2  | 39 | 4 | 20 | 50 | 6  | 28 | 15 | 13    | 39>13      | yes  | no swap |
|           | *  |    |   |    |    | *  | *  |    |       | 28<13      | no   |         |
|           | 2  | 39 | 4 | 20 | 50 | 6  | 28 | 15 | 13    | 39>13      | yes  | swap    |
|           | *  |    |   |    | *  | *  |    |    |       | 6<13       | yes  |         |
|           | 2  | 6  | 4 | 20 | 50 | *  | 39 | 28 | 15    | 13         | 4>13 | no      |
|           | *  |    | * |    | *  |    |    |    |       | 50<13      | no   |         |
|           | 2  | 6  | 4 | 20 | 50 | 39 | 28 | 15 | 13    | index meet |      | swap    |
|           | *  | *  |   |    |    |    |    |    |       |            |      |         |
|           | 2  | 6  | 4 | 13 | 50 | 39 | 28 | 15 | 20    |            |      |         |

Figure 13.12

The updated list is now [2,6,4,13,50,39,28,15,20]:

- Take the elements to the left of 13, takin 4 as pivot and sort them in the same manner.
- Once the left partition is sorted take the elements on the right and sort them taking 20 as pivot.

II.

| quicksort | 2 | 6 | 4 | 13 | 50 | 39 | 28 | 15 | 20 | 2>4   | yes   | No swap                             |  |
|-----------|---|---|---|----|----|----|----|----|----|-------|-------|-------------------------------------|--|
|           | * | * | * |    |    |    |    |    |    | 6<4   | no    | pointers meet at 6 swap 6 and 4     |  |
|           | 2 | 4 | 6 | 13 | 50 | 39 | 28 | 15 | 20 |       |       |                                     |  |
|           | * | * |   |    |    |    |    | *  |    | 50>20 | yes   | swap                                |  |
|           | 2 | 4 | 6 | 13 | 50 | *  | 39 | 28 | 15 | 20    | 15<20 | no                                  |  |
|           | * | * |   |    |    |    |    | *  |    | 39>20 | yes   |                                     |  |
|           | 2 | 4 | 6 | 13 | 15 | 39 | 28 | 50 | 20 |       |       | pointers meet at 39 swap 28 and 20  |  |
|           | * | * |   |    | *  |    |    | *  |    |       |       |                                     |  |
|           | 2 | 4 | 6 | 13 | 15 | 20 | 28 | 50 | 39 | 28>39 | NO    |                                     |  |
|           | * | * |   |    | *  |    | *  | *  |    |       |       | pointers meet at 50 swap with pivot |  |
|           | 2 | 4 | 6 | 13 | 15 | 20 | 28 | 39 | 50 |       |       |                                     |  |

Figure 13.13

## Implementation quick sort algorithm

**Step 1:** Decide upon the Pivot

- This function takes three parameters, the `list(input_list)`, starting (fast) and ending (last) index for the list that has to be sorted.

- Take the `input_list.pivot = input_list[last]`. We set the pivot to last value of the list.
- Set the `left_pointer` to first.
- Set `right_pointer` to last -1, because the last element is the pivot.
- Set `pivot_flag` to True.
- While `pivot_flag` is true:
  - If `leftmark > pivot` and `rightmark < pivot` then swap left mark and right mark and increment `leftmark` by 1 and decrement `rightmark` by 1.
  - If `leftmark > pivot` and `rightmark > pivot` then only decrement the `rightmark`.
  - Same way if `leftmark < pivot` and `rightmark < pivot` then only increment the `leftmark`.
  - If `leftmark < pivot` and `rightmark > pivot`, increment `leftmark` by 1 and decrement right mark by 1.
  - When left mark and rightmark meet at one element, swap that element with the pivot.
  - When, `leftmark >= rightmark`, swap the value of the pivot with the element at left pointer, set the `pivot_flag` to false.

```

def find_pivot(input_list, first, last):
 pivot = input_list[last]
 print("pivot =", pivot)
 left_pointer = first
 print("left pointer = ", left_pointer, " ", input_list[left_pointer])
 right_pointer = last-1
 print("right pointer = ", right_pointer, " ", input_list[right_pointer])
 pivot_flag = True
 while pivot_flag:
 if input_list[left_pointer]>pivot:
 if input_list[right_pointer]<pivot:
 temp = input_list[right_pointer]
 input_list[right_pointer]=input_list[left_pointer]
 input_list[left_pointer]= temp
 right_pointer = right_pointer-1
 left_pointer = left_pointer+1
 else:
 right_pointer = right_pointer-1
 else:
 left_pointer = left_pointer+1

```

```

 right_pointer = right_pointer-1
 if left_pointer >= right_pointer:
 temp = input_list[last]
 input_list[last] = input_list[left_pointer]
 input_list[left_pointer] = temp
 pivot_flag = False
 print(left_pointer)
 return left_pointer

```

## Step 2: Define `quicksort(input_list)` function.

- This function will take a list as input.
- Decides the starting point(0) and end point(length\_of\_the\_list-1) for sorting.
- Call the `qsHelper()` function.

```

def quickSort(input_list):
 first = 0
 last = len(input_list)-1
 qsHelper(input_list,first,last)

```

## Step 3: Define the `qsHelper()` function

This function checks the first index and last index value, it is a recursive function, it calls the `find_pivot` method where leftmark is incremented by 1 and rightmark is decremented by 1. So, as long as the leftmark (which is parameter first in this case) is less than the rightmark (which is parameter last in this case) the while loop will be executed where `qsHelper` finds new value of pivot, creates ;eft and right partition and calls itslef.

```

def qsHelper(input_list,first,last):
 if first<last:
 partition = find_pivot(input_list,first,last)
 qsHelper(input_list,first,partition-1)
 qsHelper(input_list,partition+1,last)

```

## Code:

```

def find_pivot(input_list, first,last):
 pivot = input_list[last]
 left_pointer = first
 right_pointer = last-1
 pivot_flag = True

 while pivot_flag:

```

```

if input_list[left_pointer]>pivot:
 if input_list[right_pointer]<pivot:
 temp = input_list[right_pointer]
 input_list[right_pointer]=input_list[left_pointer]
 input_list[left_pointer]= temp
 right_pointer = right_pointer-1
 left_pointer = left_pointer+1
 else:
 right_pointer = right_pointer-1

else:
 if input_list[right_pointer]<pivot:
 left_pointer = left_pointer+1
 else:
 left_pointer = left_pointer+1
 right_pointer = right_pointer-1

if left_pointer >= right_pointer:
 temp = input_list[last]
 input_list[last] = input_list[left_pointer]
 input_list[left_pointer] = temp
 pivot_flag = False
return left_pointer

def quickSort(input_list):
 first = 0
 last = len(input_list)-1
 qsHelper(input_list,first,last)

def qsHelper(input_list,first,last):
 if first<last:
 partition = find_pivot(input_list,first,last)
 qsHelper(input_list,first,partition-1)
 qsHelper(input_list,partition+1,last)

```

## Execution:

```

input_list=[15,39,4,20,50,6,28,2,13]
quickSort(input_list)
print(input_list)

```

## Output:

```
[2, 4, 6, 13, 15, 20, 28, 39, 50]
```

## Conclusion

In this chapter, you learnt about some very important searching and sorting algorithms and you also learnt how to implement them using Python. Now get ready to work with Flask in the next chapter.

# CHAPTER 14

## Getting Started with Flask

### Introduction

This chapter offers a brief introduction to Flask which is a very popular web framework that is being used extensively for creating web applications with Python.

### Structure

- Introduction
- Installation of virtual environment
- “Hello world” Application with Flask
- Debugging a flask application

### Objective

After reading this chapter, you will have an understanding of Flask web Framework, and you will know how to create a simple webpage using Flask.

### 14.1 Introduction

Flask is a very popular web framework that is being used extensively for creating web applications with Python. Due to its small framework flask is often referred to as micro framework. Here are few features about flask.

1. Flask was developed by *Armin Ronacher*, and it is written in Python.
2. Light weight **web server gateway interface (WSGI)** web application framework.
3. Free for commercial use.

4. Flask was created with the intention of making it easier for the beginners to get started with web development. However, it can also be used for creating complex applications.
5. Knowledge of flask can be of great benefit for a Python developer.
6. Provides libraries, modules, and tools to create web applications.
7. Flask is easy to use.
8. It has built in development server and debugger.
9. Flask has integrated unit testing support.
10. Uses Jinja2 templating.
11. Support for secure cookies (client side sessions).
12. 100% WSGI 1.0 compliant.
13. Unicode based.
14. Extensively documented.

## **Advantages**

- Scalable
- Simpler development
- Flexibility
- Performance
- Modularity

Jinja2 is the template language that is used by flask. Jinja is a web template engine for Python used to create HTML, HML or other markup formats that are returned to the user via HTTP response. Python code can also be used for creating web applications.

## **14.2 Installation of virtual environment**

### **Install virtual environment using pip**

```
pip install virtualenv
```

The idea behind virtual environment is to have an environment where we can install packages specific to a particular project. Suppose you create

some projects in python version 3.6, and then you want to create a new project in updated version of python if you work directly on the mail installation, it would break the certain modules of the old project. So, it is better if each project has its own package to work on.

It is very important to have a virtual environment installed for developing flask application. It is not advisable to deploy a flask application especially a web app that has been created using main Python installation as that can lead to lot of complications. You may have experimented a lot with your main Python installation, which makes it unsuitable for web application development. Ideally, it is better to work on a clean environment that provides a clean environment that gives access to isolated Python environment. You will use that environment only for creating application with flask and nothing else. This is the reason why creation of virtual environment is done before you get started with the coding processes. The virtual environment files will be created inside a folder that is at the same directory level with your project folder.

In order to use virtual environment, you must use anything above python *version 3.3* or higher (and also *version 2.7* but we are not working with Python 2 in this book). We are following Python *version 3.8.2*.

So, before we get started, type the following command to check which all packages are already installed in your system:

```
pip list
```

```
Command Prompt
dnspython 1.16.0
filelock 3.0.12
kiwisolver 1.1.0
matplotlib 3.1.3
more-itertools 8.2.0
mysql-connector-python 8.0.19
numpy 1.18.1
packaging 20.3
pandas 1.0.3
pip 20.0.2
pluggy 0.13.1
protobuf 3.6.1
pscript 0.7.3
py 1.8.1
pyparsing 2.4.6
pytest 5.4.1
python-dateutil 2.8.1
pytz 2019.3
setuptools 41.2.0
six 1.14.0
sqlparse 0.3.1
tornado 6.0.4
virtualenv 20.0.7
wcwidth 0.1.9
webruntime 0.5.6
```

*Figure 14.1*

Go to the directory where you want to create a project. For this example, it is desired to create the project in C: drive. So, let's go to C: drive.

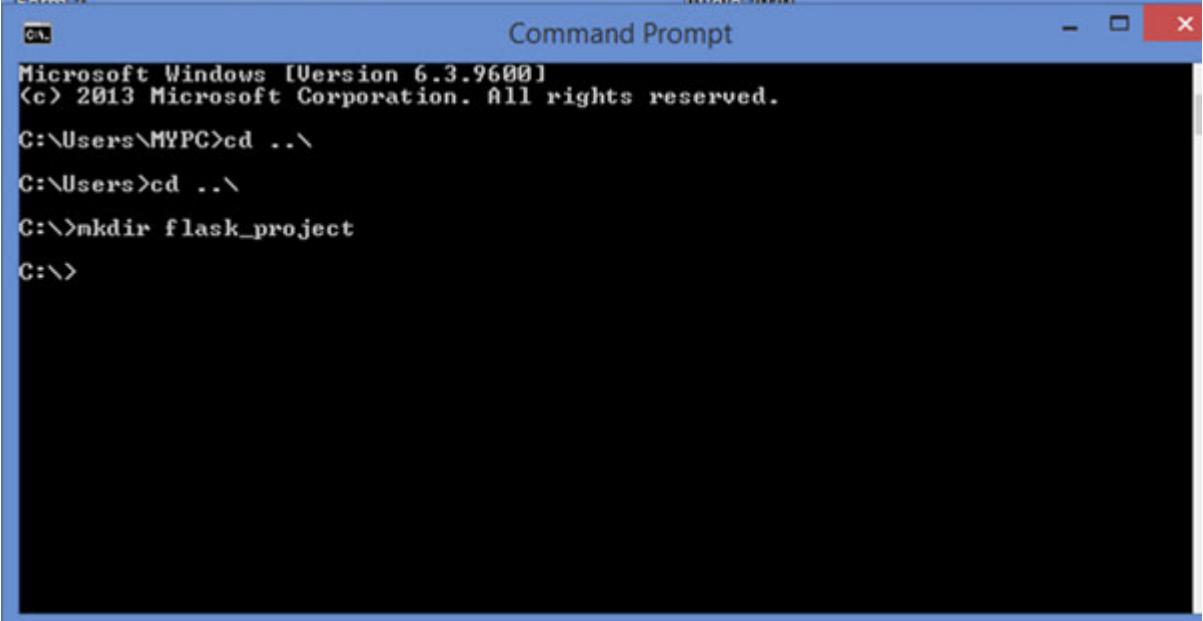
```
Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>cd ..
C:\Users>cd ..
C:\>
```

*Figure 14.2*

The next step is to create a folder for your project.

```
mkdir flask_project
```

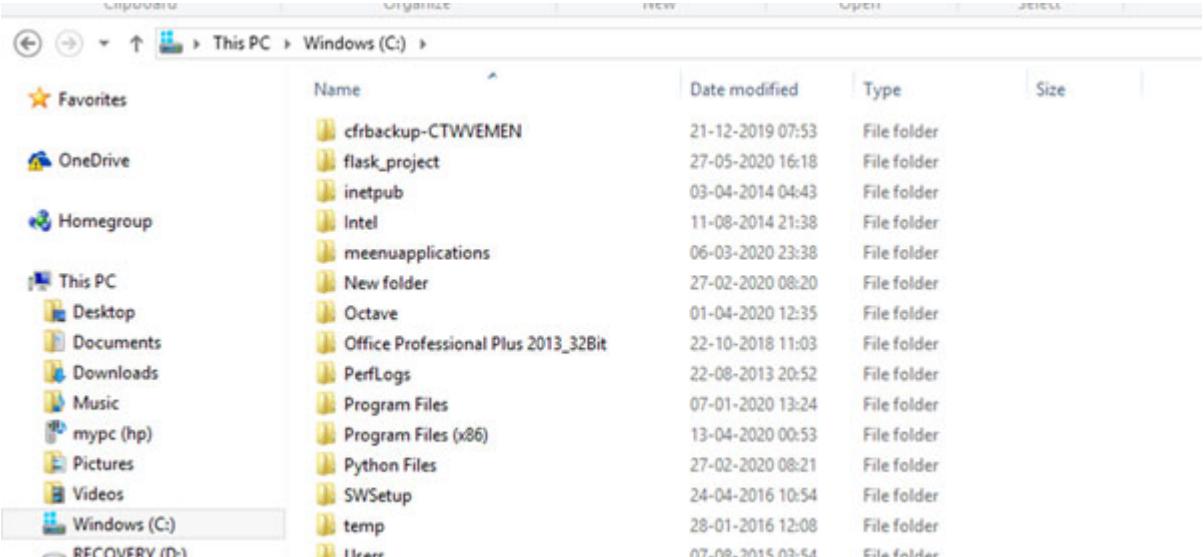


```
Windows PowerShell
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MyPC>cd ..
C:\Users>cd ..
C:\>mkdir flask_project
C:\>
```

*Figure 14.3*

This will create a new folder **flask\_project** in c: drive as follows:



| Name                                | Date modified    | Type        | Size |
|-------------------------------------|------------------|-------------|------|
| cfrbackup-CTWVEMEN                  | 21-12-2019 07:53 | File folder |      |
| flask_project                       | 27-05-2020 16:18 | File folder |      |
| inetpub                             | 03-04-2014 04:43 | File folder |      |
| Intel                               | 11-08-2014 21:38 | File folder |      |
| meenuapplications                   | 06-03-2020 23:38 | File folder |      |
| New folder                          | 27-02-2020 08:20 | File folder |      |
| Octave                              | 01-04-2020 12:35 | File folder |      |
| Office Professional Plus 2013_32Bit | 22-10-2018 11:03 | File folder |      |
| PerfLogs                            | 22-08-2013 20:52 | File folder |      |
| Program Files                       | 07-01-2020 13:24 | File folder |      |
| Program Files (x86)                 | 13-04-2020 00:53 | File folder |      |
| Python Files                        | 27-02-2020 08:21 | File folder |      |
| SWSetup                             | 24-04-2016 10:54 | File folder |      |
| temp                                | 28-01-2016 12:08 | File folder |      |
| Users                               | 07-08-2015 03:54 | File folder |      |

*Figure 14.4*

So, this is where we will work:

```
cd flask_project
```

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\MYPC>cd ..
C:\Users>cd ..
C:\>mkdir flask_project
C:\>cd flask_project
C:\flask_project>_
```

*Figure 14.5*

Now, let's create a new virtual environment.

To create a new virtual environment, give the following command:

```
python -m venv virtual_environment_name
```

So, now if you just type `dir` you can see that the new environment `flvirenv` has been created in `c:\flask_project`.

```
C:\Users\MYPC>cd ..
C:\Users>cd ..
C:\>mkdir flask_project
C:\>cd flask_project
C:\flask_project>python -m venv flvirenv
C:\flask_project>dir
Volume in drive C is Windows
Volume Serial Number is 224C-DE55

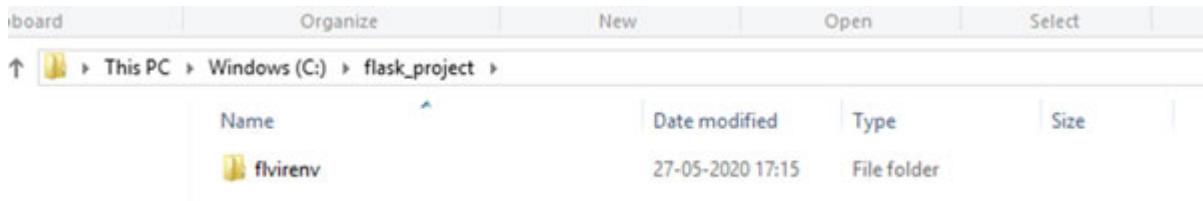
Directory of C:\flask_project

27-05-2020 17:15 <DIR> .
27-05-2020 17:15 <DIR> flvirenv
0 File(s) 0 bytes
3 Dir(s) 414,013,313,024 bytes free

C:\flask_project>
C:\flask_project>_
```

*Figure 14.6*

You can also check in the window explorer:



*Figure 14.7*

So, if you go to flvirenv/Scripts folder, you will see activate.bat file. That must be activated to create the virtual environment.

```
C:\>mkdir flask_project
C:\>cd flask_project
C:\flask_project>python -m venv flvirenv
C:\flask_project>dir
 Volume in drive C is Windows
 Volume Serial Number is 224C-DE55
 Directory of C:\flask_project
27-05-2020 17:15 <DIR>
27-05-2020 17:15 <DIR>
27-05-2020 17:15 <DIR> flvirenv
 0 File(s) 0 bytes
 3 Dir(s) 414,013,313,024 bytes free
C:\flask_project>
C:\flask_project>flvirenv\Scripts\activate.bat
(flvirenv) C:\flask_project>
```

*Figure 14.8*

Once the environment is activated, you will see its name in parenthesis on the left side of the prompt. The version of python used in the environment is same as the version used to create it. Now if you type `pip list` you will know what all is installed in the virtual environment. In this environment we can install all that is required to work on our project.

Our requirement right now is to install flask, which can be done by giving the following command:

```
pip install flask
```

```

Command Prompt

<flvirenv> C:\flask_project>pip list
Package Version

pip 19.2.3
setuptools 41.2.0
WARNING: You are using pip version 19.2.3, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

<flvirenv> C:\flask_project>
<flvirenv> C:\flask_project>pip install flask
Collecting flask
 Downloading https://files.pythonhosted.org/packages/f2/28/2a03252dfb9ebf377f40fba6a7841b47083260bf8hd8e737b0c6952df83f/Flask-1.1.2-py2.py3-none-any.whl (94kB)
 : [=====] | 102kB 3.3MB/s
Collecting Jinja2>=2.10.1 (from flask)
 Downloading https://files.pythonhosted.org/packages/30/9e/f663a2aa66a09d838042ae1a2c5659828bb9b41ea3abefa20a20fd92b121/Jinja2-2.11.2-py2.py3-none-any.whl (125kB)
 : [=====] | 133kB 3.3MB/s
Collecting itsdangerous>=0.24 (from flask)
 Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl

```

*Figure 14.9*

So if you again run `pip list`, you will know what all has been installed in `flvirenv`.

```

Command Prompt

cf65f2b07a5ae73eeacf66d2010c0e934737d1d9/MarkupSafe-1.1.1-cp38-cp38-win32.whl
Installing collected packages: MarkupSafe, Jinja2, itsdangerous, Werkzeug, click, flask
Successfully installed Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2
flask-1.1.2 itsdangerous-1.1.0
WARNING: You are using pip version 19.2.3, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

<flvirenv> C:\flask_project>pip list
Package Version

click 7.1.2
Flask 1.1.2
itsdangerous 1.1.0
Jinja2 2.11.2
MarkupSafe 1.1.1
pip 19.2.3
setuptools 41.2.0
Werkzeug 1.0.1
WARNING: You are using pip version 19.2.3, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

<flvirenv> C:\flask_project>

```

*Figure 14.10*

## 14.3 “Hello world” Application with Flask

We will now create our first app in Flask framework.

**Step 1.** Import the Flask class

```
from flask import Flask
```

## Explanation:

The instance of Flask class will be the WSGI application, and is always the central object in any application built in the flask framework. WSGI (pronounced as whiskey) stands for **Web Server gateway Interface** and is the Python standard web server interface defined in **Python Enhancement Proposal (PEP) 3333**. WSGI is a simple calling convention that allows python web apps to interact with the webservers. As a developer, you only have to focus on developing the app, Flask handles implementation of all the necessary procedures. Importing Flask is mandatory.

**Step 2:** Create an instance of the **Flask** class.

The flask instance is generally created in the main module or in the **\_\_init\_\_.py** file of the package.

```
app = Flask(__name__)
```

The parameter **\_\_name\_\_** is used to find the resources on the filesystem. It is a correct value to provide if you are planning to create a single module such as this application that we are working on. However, if you are using a package, then you will have to provide your package name as parameter.

**Step 3:** Use the **route()** decorator

```
@app.route('/hello')
def helloIndex():
 return 'Hello World!'
```

With the help of **route()** function, the application comes to know which URL is to be associated with the function. So, a function is triggered by the URL. So, basically all requests for the root URL('/hello') will be directed to the function **helloIndex()** that is defined in the next line.

The **@app.route** decorator will wrap the function **helloIndex()** to route requests for URL('/hello') to a particular view. The name of the function could be anything. The **helloIndex()** function will not take any argument and returns a simple string.

**Step 4:** Give command to run the application

```
if __name__ == '__main__':
 app.run(port=5000, debug=True)
```

This is how the code looks:

```
File Edit Format Run Options Window Help
from flask import Flask
app = Flask(__name__)

@app.route('/hello')
def helloIndex():
 return 'Hello World!'

if __name__ == '__main__':
 app.run(port=5000, debug=True)
```

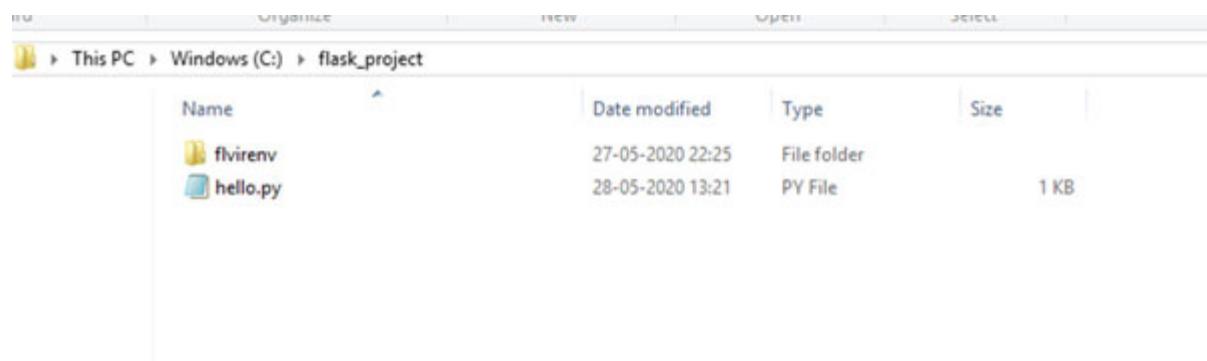
*Figure 14.11*

Now, follow the following steps to run the application:

### Step 1

Save the file in your project folder.

We have saved the file by the name `hello.py` in the `flask_project` folder.



*Figure 14.12*

### Step 2. Activate the virtual environment

```
C:\flask_project>flvirenv\Scripts\activate.bat
'"C:\Windows\System32\chcp.com"' is not recognized as an internal or external co
mmand,
operable program or batch file.

<flvirenv> C:\flask_project>
```

*Figure 14.13*

**Step 3.** Give the following command:

```
python hello.py
```

```
C:\flask_project>flvirenv\Scripts\activate.bat
'"C:\Windows\System32\chcp.com"' is not recognized as an internal or external co
mmand,
operable program or batch file.

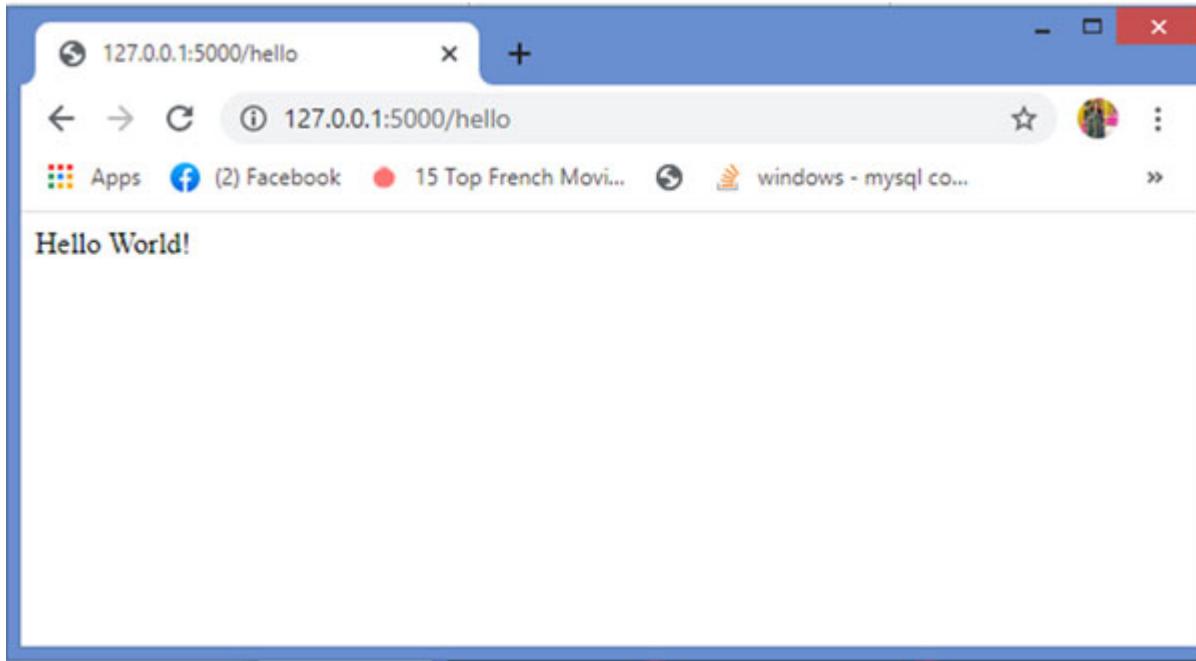
<flvirenv> C:\flask_project>python hello.py
* Serving Flask app "hello" (lazy loading)
* Environment: production
 WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 332-562-534
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

*Figure 14.14*

**Step 4:** Open the application in browser

In the command prompt, you can see that the app is running at <http://127.0.0.1:5000>, and in your code, you have set app.route to '/hello'.

So, to view your app, you will have to type **http://127.0.0.1:5000/hello**.



*Figure 14.15*

Let's now try working with dynamic routes. This is designed to pick up the name of the person from dynamic URL and welcome him/her. So, we add another function in the same **hello.py** file.

```
from flask import Flask
app = Flask(__name__)

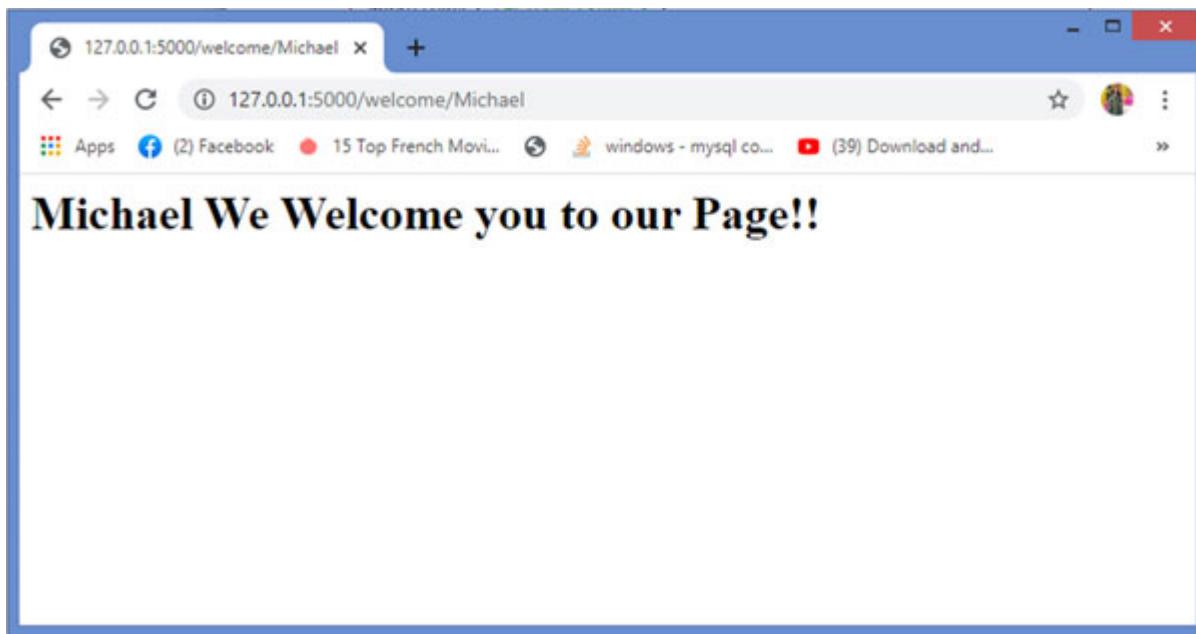
@app.route('/hello')
def helloIndex():
 return 'Hello World!'

@app.route('/welcome/<name>')
def welcomeUser(name):
 return '<h1> {} We Welcome you to our Page!!'.format(name)

if __name__ == '__main__':
 app.run(port=5000,debug=True)
```

*Figure 14.16*

**Output:**



*Figure 14.17*

Now, let's have a look at what we have done in this example:

```
@app.route('/welcome/<name>')
def welcomeUser(name):
 return '<h1> {} We Welcome you to our Page!!!'.format(name)
```

The app.route decorator has the route as '/welcome/<name>'. Here, name is a parameter that will change dynamically.

If you now look at the definition of `welcomeUser()`, it too has the same parameter in the parenthesis.

So, when you type in `http://127.0.0.1/welcome/Michael`, flask will automatically map, the `<name>` which in this case is *Michael* to the name parameter passed on to the function `welcomeUser()`.

The function adds the name to the welcome message and displays it on the browser. Instead of browser, we have used the html tag `<h1>` to display the message as a heading.

## 14.4 Debugging a flask application

Right now, it may seem like it's all so simple and why is coding required in the first place. However, as you get more and more ambitious with your project, you will realize that it is impossible to code without introducing

bugs in your application. So, to make your life easy, you can set the debug mode to True or False. When you run the app, you can set the debugging mode on by typing `app.run(debug =True)`. You can see that the code shown in [figure 14.18](#) has an error in the second line, but it can be identified by setting debug = True.

```
from flask import Flask
app = Flask(__name__)

@app.route('/hello')
def helloIndex():
 return 'Hello World!'

@app.route('/welcome/<name>')
def welcomeUser(name):
 return '<h1> {} We Welcome you to our Page!!!'.format(name)

if __name__ == '__main__':
 app.run(port=5000,debug=True)
```

*Figure 14.18*

When you try to execute the code, the following happens:

```
"C:\Windows\System32\chcp.com" is not recognized as an internal or external command,
operable program or batch file.

(flvirenv) C:\flask_project>python hello.py
 * Serving Flask app "hello" (lazy loading)
 * Environment: production
 WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 332-562-534
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [29/May/2020 15:28:08] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [29/May/2020 15:28:08] "GET /favicon.ico HTTP/1.1" 404 -

(flvirenv) C:\flask_project>python hello.py
Traceback (most recent call last):
 File "hello.py", line 2, in <module>
 app = Flask(__name__)
NameError: name '__name__' is not defined

(flvirenv) C:\flask_project>
```

*Figure 14.19*

## Conclusion

As you now know that Flask is a web application Frame work written in Python. It makes creation and implementation of web application easy. You have learnt the basics of how a simple web page can be displayed using Flask. With this, we've come to the end of volume 2 of Core Python Programming. You are now fully equipped to handle any type of Python Programming challenge.

# Appendix

## **Big-O Notation**

Big-O notation helps you analyze how an algorithm would perform if the data involved in the process increases or in simple words, it is simplified analysis of how efficient an algorithm can be.

Since algorithms are an essential part of software programming, it is important for us to have some idea about how long an algorithm would take to run. Only then can we compare two algorithms and a good developer would always consider time complexity while planning the coding process.

It helps in identifying how long an algorithm takes to run.

## **Importance of Big-O notation**

Big-O notation describes how fast the runtime of an algorithm will increase with respect to the size of the input. It would give you a fair idea about how your algorithm would scale and also give you an idea about the worst-case complexity of the algorithms that you might be considering for your project. You would be able to compare two algorithms and decide which would be a better option for you. Big-O is important for the following reasons:

1. It provides the algorithm complexity in terms of input size  $n$ .
2. It considers only the steps involved in the algorithm.
3. Big-O analysis can be used to analyze both time and space.

An algorithm's efficiency can be measured in terms of best-average or worst case but Big-O notation goes with the worst case scenario.

It is possible to perform a task in different ways, which means that there can be different algorithms for the same task having different complexity and scalability.

Big-O notation compares the runtime for various types of input sizes. The focus is only on the impact of input size on the runtime, which is why we use the  $n$  notation. As the value of  $n$  increases, our only concern is to see how it affects the runtime. If we had been measuring the runtime directly,

then the unit of measurement would have been time units like second, micro-second, and so on. However, in this case ‘n’ represents the size of the input and ‘O’ stands for ‘Order’. Hence,  $O(1)$  stands for order of 1,  $O(n)$  stands for order of n and  $O()$  stands for order of the square of the size of input.

Now, suppose there are two functions:

### **Constant Complexity [O(1)]**

A task that is constant will never experience variation during runtime, irrespective of the input value. For example:

```
>>> x = 5 + 7
>>> x
12
>>>
```

The statement  $x = 5+7$  does not depend on the input size of data in any way. This is known as  $O(1)$ (big oh of 1).

### **Example:**

Suppose there are sequence of steps all of constant time as shown in the following code:

```
>>> a=(4-6)+ 9
>>> b = (5 * 8) +8
>>> print(a * b)
336
>>>
```

Now, let's compute the Big-O for these steps:

$$\begin{aligned} \text{Total time} &= O(1)+O(1)+O(1) \\ &= 3O(1) \end{aligned}$$

While computing Big – O, we ignore constants because once the data size increases, the value of constant does not matter.

Hence, the *Big-O* would be  $O(1)$ .

### **Linear complexity: [O(n)]**

In case of linear complexity, the time taken depends on the value of the input provided.

Suppose you want to print table of 5, have a look at the following code:

```
>>> for i in range(0,n):
print("\n 5 x ",i,"=",5*i)
```

The number of lines to be printed, depends on ‘n’. For  $n = 10$ , only ten lines will be printed but for  $n = 1000$ , the for loop will take more time to execute.

The print statement is  $O(1)$ .

So, the block of code is  $n * O(1)$  which is  $O(n)$ .

Consider the following lines of code:

```
>>>j = 0 ----- (1)
>>> for i in range(0,n):
print("\n 5 x ",i,"=",5*i) ----- (2)
```

1. Is  $O(1)$
2. Block is  $O(n)$

$$\text{Total time} = O(1) + O(n)$$

We can drop  $O(1)$  because it is a lower order term and as the value of n becomes large (1) will not matter and the runtime actually depends on the for loop.

Hence, the Big-O for the code mentioned above is  $O(n)$ .

## Quadratic Complexity:[O0]

As the name indicates quadratic complexity, the time taken depends on the square of the input value. This can be the case with nested loops. Look at the following code:

```
>>> for i in range (0,n):
for j in range(0,n):
print("I am in ", j," loop of i = ", i,".")
```

In this piece of code, the print statement is executed times.

## Logarithmic Complexity

Logarithmic complexity indicates that in worst case the algorithm will have to perform  $\log(n)$  steps. To understand this, let's first understand the concept of logarithm.

Logarithms are nothing but inverse of exponentiation.

Now, let's take a term. Here 2 is the base and 3 is the exponent. We therefore say that base 2  $\log$  of 8 () = 3. In the same way, if, then base 10  $\log$  of 100000 () = 5.

Since the computer works with binary numbers, therefore, in programming and Big O we generally work with base 2 logs.

Have a look at the following observation:

1. ,
2. ,
3. ,
4. ,
5. ,

This means that if  $n = 2$ , number of steps = 1.

If  $n = 4$ , number of steps will be 2.

If  $n = 8$ , then number of steps will be 3.

So, as data doubles the number of steps increase by one.

The number of steps grows slowly in comparison to the growth in data size.

The best example for logarithmic complexity in software programming is Binary Search tree. You will learn more about it in chapter based on Trees.

### **Example:**

```
i = j = k =0
for i in range(n/2,n):
 for j in range(2,n):
 k = k+n/2
 j = j*2
```

Time complexity for the first for loop  $O(n/2)$ .

Time complexity for second for loop:  $O(\log n)$  because j is doubling itself till it is less than n.

$$\begin{aligned}\text{Total time} &= O(n/2) * O(\log n) \\ &= O(n \log n)\end{aligned}$$

### **Big-O notations and names:**

1. Constant –  $O(1)$
2. Logarithmic –  $O(\log(n))$
3. Linear –  $O(n)$
4. Log Linear –  $O(n \log(n))$
5. Quadratic –  $O(n^2)$
6. Cubic –  $O(n^3)$
7. Exponential –  $O(2^n)$

The worst-case time complexity in computer science means worst-case in terms of time consumed while execution of a program. It is the longest running time that an algorithm can take. The efficiency of algorithms can be compared by looking at the order of growth of the worst-case complexity.

Big-O analysis is also known as **asymptotic notation**.

It is very important to know how fast a program would run. Different computers have different hardware capabilities. The exact runtime is not considered because the results can vary depending on the hardware of the machine, speed of the processor, and the other processors that are running in the background. What is more important to understand is that how the performance of the algorithm gets affected with increase in input data.

In order to resolve this issue, the concept of asymptotic notation was developed. It provides a universal approach for measuring speed and efficiency of an algorithm. For applications dealing with large inputs, we are more interested in behavior of an algorithm as the input size increases. Big O notation is one of the most important asymptotic notations.

### **Time and space complexity**

Time complexity gives an idea about the number of steps that would be involved in order to solve a problem. The general order of complexity in

ascending order is as follows:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O()$$

Unlike time, memory can be reused and people are more interested in how fast the calculations can be done. This is one main reason why time complexity is often discussed more than space complexity. However, space complexity is never ignored. Space complexity decides how much memory will be required to run a program completely. Memory is required for:

1. Instruction space.
2. Knowledge space for constants, variables, structured variables, dynamically altered areas, and so on.
3. Execution.

As the name suggests, space complexity describes how much memory space would be required if size of input  $n$  increases. Here too we consider the worst-case scenario.

Now, have a look at the following code:

```
>>> x = 23 (1)
>>> y = 34 (2)
>>> sum = x + y (3)
>>> sum
57
>>>
```

In this example, we need space to store three variables:  $x$  in (1),  $y$  in (2), and  $sum$  in (3). However, this scenario will not change, and the requirement for 3 variables is constant, and hence the space complexity is  $O(1)$ .

Now, have a look at the following code:

```
word = input("enter a word : ")
word_list = []
for element in word:
 print(element)
 word_list.append(element)
print(word_list)
```

The size of the `word_list` object increases with  $n$ . Hence, in this case, the space complexity is  $O(n)$ .

If there is a function 1, which has suppose three variables and this function1 calls another function named function2 that has 6 variables, then the overall requirement for temporary workspace is of 9 units. Even if function1 calls function2 ten times, the workspace requirement will remain the same.

Look at the following code:

```
n = int(input("provide a number : "))
statement = "Happy birthday"
for i in range(0,n):
 print(i+1,".", statement)
```

The space complexity for the preceding code will be  $O(1)$  because the space requirement is only for storing value of integer variable  $n$  and string  $statement$ . Even if the size of  $n$  increases the space requirement remains constant. Hence,  $O(1)$ .

Now, find out the time and space complexity of the following code:

```
a = b = 0
for i in range(n):
 a = a + i
for j in range(m):
 b = b + j
```

### **Time complexity**

Time for first loop =  $O(n)$

Time for second loop =  $O(m)$

Total time =  $O(n) + O(m) = O(n + m)$

### **Space complexity**

$O(1)$

# Index

## Symbols

\*args [12](#)  
\*\*kwargs [13](#)

## A

acquire() method [188](#)  
actual arguments [58](#)  
anonymous functions [62](#)  
assertion methods, unittest module [264](#)  
assertions  
    about [254](#)  
    using [254](#)  
asymptotic notation [627](#)  
attributes [66](#)

## B

basic queue functions  
    dequeue() [467](#)  
    enqueue(i) [467](#)  
    isEmpty() [467](#)  
    size() [467](#)  
Big-O notation  
    about [623](#)  
    Constant Complexity [O(1)] [624](#)  
    importance [623, 624](#)  
    linear complexity [625](#)  
    logarithmic complexity [626](#)  
    quadratic complexity [626](#)  
    time and space complexity [628](#)  
binary heap  
    about [539, 540](#)  
    applications [561](#)  
    Max Heap [539](#)  
    Min Heap [539](#)  
    properties [539, 540](#)  
binary search  
    about [568](#)  
    implementing [569-572](#)  
binary tree [517](#)  
bounded methods [90](#)  
bubble sort

about [587](#)  
implementing [587-589](#)  
built-in namespace [18](#)  
button  
about [351](#)  
code for displaying [351-355](#)  
creating [355-360](#)  
syntax for creating [355](#)  
working with [351](#)

## C

Canvas  
about [362](#)  
arc, drawing [371, 372](#)  
arc, working with [373-375](#)  
associating, to root window [362](#)  
cross, drawing [367](#)  
exploring [362, 363](#)  
features for arc, defining [377, 378](#)  
line, creating [364-366](#)  
one fourth eaten pizza image, creating [380, 381](#)  
options [385](#)  
oval, creating [376, 382](#)  
polygon, drawing [383-385](#)  
polygon, writing on [376](#)  
rectangle, defining [368, 369](#)  
shapes, drawing [368](#)  
text, writing on [386, 387](#)  
chaining [574](#)  
classes  
about [64, 65](#)  
Employee class, creating [66-71](#)  
types of variables [73](#)  
class parenthesis\_match  
defining [461](#)  
class variable  
defining [73-77](#)  
code  
for finding all possible palindromic partitions in string [42](#)  
for finding nth power of [3 56](#)  
for finding sum of natural numbers from 0 to given number [51](#)  
writing, while loop used [50](#)  
collision [573](#)  
commands, while debugging [250](#)  
comparison sort [587](#)  
Condition class  
about [208](#)  
Condition variable [208, 209](#)  
methods [209](#)

Constant Complexity [O(1)] [624](#), [625](#)  
contents, of Python directory  
    checking, listdir() function used [98](#)  
current working directory(cwd)  
    retrieving, getcwd() method used [98](#)  
cursor [130](#)

## D

Daemon thread  
    about [210](#), [211](#)  
    setting [211-215](#)  
data  
    inserting, GUI used [430-434](#)  
database  
    creating [135](#)  
    creating, with MySQL [135](#), [136](#)  
    creating, with Python [137-141](#)  
database, working with using Python  
    about [141](#)  
    all records, fetching from table with fetchall() [147](#)  
    clause, updating [156](#)  
    one record, fetching from table with fetchone() [148-150](#)  
    ORDER BY clause, using [154](#)  
    records, deleting using DELETE command [155](#)  
    records, inserting [141-146](#)  
    records, selecting [147](#)  
    selected records, fetching from table with WHERE clause [151](#)  
data visualization  
    about [271](#), [272](#)  
    benefits [273](#)  
Deadlock condition  
    about [190-193](#)  
    locked() function, used for checking resource is locked [193-196](#)  
    RLock, resolving with [196-198](#)  
debugging  
    commands, using [250](#)  
    program [244-249](#)  
default arguments [9](#), [10](#), [61](#)  
default parameter [60](#), [61](#)  
deque  
    about [481](#)  
    code, writing for implementation [482](#)  
    implementing [482](#)  
describe() function [326](#)  
design errors [233](#)  
destructor \_\_del\_\_() method [72](#)  
directory. See Python directory  
direct recursion  
    example [32-34](#)

doubly linked list implementation  
function, for adding node at end [505](#)  
function, for removing node [506-509](#)  
Node class, creating [504](#)  
traverse forward [505](#)  
traverse reverse [505](#)

## E

edge [517](#)  
Employee class  
    creating [66-71](#)  
errors  
    about [226](#)  
    design errors [233](#)  
    logical error [232](#)  
    runtime errors [230](#)  
    semantic errors [232](#)  
    syntax errors [227-229](#)  
errors, with respect to Python  
    about [233](#)  
    exception, catching in general [237-240](#)  
    exception, raising [243](#)  
    try and catch [233](#)  
    try block with finally [242](#)  
    try...except...else statement [240-242](#)  
    try...except...finally [242](#)  
    ZeroDivisionError [234](#), [235](#)  
Excel sheet  
    dataframe, creating from [314-316](#)  
Exceptions [230-233](#)

## F

factorial of number  
    about [29](#)  
    finding, Recursion used [29-31](#)  
Fibonacci numbers  
    sequence, creating [36](#)  
Fibonacci sequence  
    writing, Recursion used [54](#)  
files  
    about [95](#), [96](#)  
    access mode, in file writing [105](#), [106](#)  
    advantages, of storing data [96](#), [97](#)  
    binary file contents, reading with “rb” access mode [112](#)  
    binary file, writing to with “wb” access mode [111](#)  
    closing, close() function used [103](#), [104](#)  
    data [97](#)  
    end of file [97](#)

file attributes [113](#)  
header [97](#)  
opening, open() function used [103](#)  
other operations, on binary file [113](#)  
reading, with r+ mode [106](#)  
strings, reading and writing in binary files [112](#)  
text, inserting at end of file [107-111](#)  
various methods [114](#)  
working with [102](#)  
writing, with r+ mode [107](#)

Flask  
about [607, 608](#)  
advantages [608](#)  
features [608](#)  
“Hello world” application, creating [615-619](#)

flask application  
debugging [620](#)

folding hash function  
about [578](#)  
implementing [579-586](#)

formal arguments [58](#)

Frame  
about [387](#)  
options [388](#)  
working with [387](#)

fruitful function [62](#)

functional arguments  
\*args [12](#)  
default arguments [9, 10](#)  
keyword arguments [11](#)  
\*\*kwargs [13](#)  
positional arguments [8, 9](#)

function, for calculating HCF  
writing, Euclidean Algorithm used [44-46](#)

function, for Fibonacci series  
writing, for loop used [48, 49](#)

function, for finding factorial of number  
writing, for loop used [47](#)

function header [58](#)

functions  
about [1, 2](#)  
benefits [2, 3](#)  
built-in functions [3](#)  
creating [3](#)  
defining, to take parameter [6-8](#)  
simple functions, creating [3](#)  
user-defined functions [3](#)

## G

garbage collection [72](#)  
Global Interpreter Lock (GIL) [218](#)  
global namespace [19](#)  
global variable [60](#)  
grid layout  
    about [347](#)  
    methods [347](#)  
GUI  
    creating, for retrieving results [435-453](#)  
    table, creating with [422-426](#)  
    tkinter, importing [338-342](#)  
    used, for inserting data [430-434](#)  
    working with [338](#)

## H

hash function  
    purpose [575](#)  
hash tables  
    about [572](#)  
    folding hash function [578, 579](#)  
    implementing [572-574](#)  
    remainder hash function, implementing [575-577](#)  
“Hello world” application  
    creating, with Flask [615-619](#)

## I

indirect recursion  
    example [32-34](#)  
inheritance  
    about [77](#)  
    implementing [78-81](#)  
    subclass [77](#)  
    superclass [77](#)  
In Order Traversal [533](#)  
insertion sort  
    about [590](#)  
    implementing [591-593](#)  
instance [64](#)  
instance variable  
    defining [73-77](#)  
integration testing [253](#)

## K

keyword arguments [11, 61](#)

## L

labels  
working with [389](#)  
lambda functions  
about [22](#)  
examples [22-28](#)  
syntax [22](#)  
layout managers, widgets  
about [344](#)  
grid [347](#)  
pack [345](#)  
place [349](#)  
legend() function [293](#)  
linear complexity [625](#)  
Linear probing [575](#)  
linear search [564](#)  
linked list  
about [486](#)  
doubly linked list [486](#)  
node [486](#)  
singly linked list [486](#)  
Listbox widget  
about [393](#)  
creating [393](#)  
methods [394, 395](#)  
local namespace [19-21](#)  
local variable [60](#)  
lock object  
acquire() method [187](#)  
applying [187](#)  
release() method [187](#)  
logarithmic complexity [626](#)  
logical errors [232](#)

## M

magic methods  
examples [92](#)  
Matplotlib  
about [273](#)  
installing [273, 274](#)  
line, plotting [282-288](#)  
multiple points, plotting [281](#)  
point, plotting [275-279](#)  
PyPlot, working with [275](#)  
x and y axis, labelling [289-295](#)  
Max Heap  
building up [541-543](#)  
implementing [543-547](#)  
maximum value, finding out [551-561](#)  
memoization [36-38](#)

memory management [72](#)  
Menu buttons  
    creating [396, 397](#)  
    options [402-405](#)  
    pull down menu, creating [397-401](#)  
    working with [395, 396](#)  
Messagebox  
    working with [351](#)  
methods, threading module  
    implementing [216](#)  
    threading.active\_count() [216](#)  
    threading.activeCount() [216](#)  
    threading.current\_thread() [216](#)  
    threading.currentThread() [216](#)  
    threading.enumerate() [216](#)  
    threading.main\_thread() [216](#)  
mini project  
    stop watch [390-393](#)  
modularity [3](#)  
multiple test cases  
    defining, with pytest [262](#)  
    defining, with unittest [261, 262](#)  
MySQL  
    basic rules for writing SQL queries [127](#)  
    configuration [124, 125](#)  
    database, creating with command line tool [126, 127](#)  
    installing [122, 123](#)  
mysql connector  
    installing [128](#)  
MySQL database  
    connecting to Python, with MySQL connector [127, 128](#)  
    creating, Python used [128-132](#)  
    records, retrieving with Python [132-135](#)  
MySQLdb database  
    interacting with [420-422](#)  
MySQL essential  
    about [122](#)  
    download link [123](#)  
    setting up [123](#)

## N

namespaces  
    about [18](#)  
    built-in namespace [18](#)  
    global namespace [18, 19](#)  
    local namespace [18-21](#)  
new thread  
    implementing, threading module used [176-182](#)  
node

height of node [517](#)  
Node class implementation  
  about [487](#)  
  data [487](#)  
  doubly linked list, implementing [504](#)  
  linked list, reversing [510-513](#)  
  linked list, traversing through [489-491](#)  
  node, adding at beginning of linked list [492, 493](#)  
  node, adding at end of linked list [494, 495](#)  
  node, inserting between two nodes in linked list [497](#)  
  node, removing from linked list [499](#)  
  reference to next node [487](#)  
  values of node, printing in centre of linked list [501-504](#)  
non-daemon thread [210, 211](#)  
non-fruitful function [62](#)  
numpy  
  about [295](#)  
  arrays, creating [297-311](#)  
  installing [295](#)  
  multi-dimensional arrays, using [303-305](#)  
  shape of arrays [296](#)  
  value of elements, obtaining from array [296, 297](#)

## O

Object Oriented Programming (OOP) [64](#)  
objects [64, 65](#)  
open addressing [574](#)  
operations, in dataframe  
  about [319, 320](#)  
  columns, retrieving [320, 321](#)  
  columns, working with [322-325](#)  
  data, retrieving based on conditions [327](#)  
  data, retrieving from multiple columns [325, 326](#)  
  data sorting [333, 334](#)  
  index range [328-331](#)  
  index, resetting [332](#)  
  rows, retrieving [320, 321](#)  
operator overloading [92](#)  
optional parameter [60](#)  
ORDER BY clause  
  using [154](#)

## P

pack layout  
  about [345](#)  
  pack methods [345, 346](#)  
pandas  
  about [313](#)

dataframe, creating from .csv file [316](#)  
dataframe, creating from dictionary [317](#), [318](#)  
dataframe, creating from Excel sheet [314-316](#)  
dataframe, creating from list of tuples [318](#)  
installing [313](#)  
working with [314](#)  
parameter [58](#)  
pdb debugger [249](#), [250](#)  
performance testing [253](#)  
place layout  
about [349](#)  
place, with relative position [350](#)  
payload [517](#)  
pop function  
defining [459](#)  
pop operation [456](#), [457](#)  
positional arguments [9](#)  
Post Order Traversal [536](#)  
Preorder Traversal [530](#)  
priority queue [561](#)  
process [163](#), [164](#)  
program  
debugging [244-249](#)  
push function  
defining [459](#)  
push operation [456](#)  
pytest  
about [254](#)  
installing, pip used [254](#)  
working with [255-257](#)  
Python  
directory and file management [97](#)  
functional arguments [8](#)  
functions [3](#)  
Matplotlib [273](#)  
stack, implementing [458-464](#)  
Python Database API [127](#)  
Python directory  
contents, checking with listdir() function [98](#)  
creating, mkdir() used [98](#), [99](#)  
current working directory(cwd), retrieving with getcwd() method [98](#)  
removing, rmdir() function used [100-102](#)  
renaming, rename() function used [99](#)  
Python programming  
frequent errors [253](#)

## Q

qsHelper() function  
defining [603](#)

quadratic complexity [626](#)  
Quadratic probing [575](#)  
queue  
    about [467](#)  
    basic queue functions [467](#)  
    implementing, two stacks used [476-481](#)  
    stack implementation, using single queue [471-476](#)  
queue implementation  
    class, defining [467](#)  
    constructor, defining [467](#)  
    enqueue() function, defining [468](#)  
    equeue() function, defining [468](#)  
    isEmpty() function, defining [468](#)  
    size() function, defining [468](#)  
quick sort  
    about [599, 600](#)  
    implementing [600-603](#)

## R

Race Condition [184](#)  
Radio button widget  
    about [406](#)  
    creating [406-408](#)  
Recursion  
    about [1, 29](#)  
    advantages [35](#)  
    algorithm, for finding factorial [31](#)  
    disadvantages [35](#)  
    for Fibonacci numbers [35, 36](#)  
    types [31](#)  
    used, for finding factorial of number [29-31](#)  
Recursion, types  
    direct recursion [31, 32](#)  
    indirect recursion [31, 32](#)  
release() method [188](#)  
remainder hash function  
    about [575](#)  
    implementing [576, 577](#)  
result set [130](#)  
return statement [59](#)  
    about [14, 15](#)  
    examples [15-17](#)  
RLock  
    resolving with [196](#)  
runtime errors  
    about [230](#)  
    example [230, 231](#)

# S

scope of variable [59](#)  
scopes [18](#)  
scrollbar widget  
    about [408](#)  
    creating [410](#)  
    horizontal scroll bar, adding [411](#)  
    main structure, creating [408](#)  
    vertical scroll bar, creating [409](#)  
searching  
    sequential search [564](#)  
selection sort  
    implementing [589](#), [590](#)  
semantic errors [232](#)  
semaphore  
    about [199](#)  
    using [199-203](#)  
sequential search  
    about [564](#)  
    implementing [565](#)  
    implementing, for ordered list [566-568](#)  
setDaemon() method [211](#)  
shell sort  
    about [595](#), [596](#)  
    implementing [597](#), [598](#)  
simple functions  
    creating [3](#)  
    definition of function [4](#)  
    function body [4](#)  
    function, calling [4-6](#)  
simple tree representation [518-523](#)  
sinking sort [587](#)  
software testing [252](#)  
sort\_values() function [333](#)  
space complexity [628](#), [629](#)  
spinbox widget [415](#)  
stack  
    about [456-458](#)  
    BASE [456](#)  
    implementing, in Python [458-464](#)  
    implementing, single queue used [471-476](#)  
    TOP [456](#)  
Stack Class  
    defining [458](#)  
static decorator  
    using [91](#)  
static methods [90](#)  
string concatenation operation [248](#)  
Structured Query Language (SQL) [122](#)

subtrees [517](#)  
syntax errors  
    about [227](#), [228](#)  
    example [227-229](#)  
system-level testing [253](#)

## T

table  
    creating, GUI used [422-426](#)  
test-driven development [251](#)  
testing  
    integration testing [253](#)  
    performance testing [253](#)  
    system-level testing [253](#)  
    unit testing [252](#)  
text widget  
    about [413](#)  
    creating [413-415](#)  
thread  
    about [163-165](#)  
    creating [165](#), [166](#)  
    Daemon [210](#)  
    non-daemon thread [210](#), [211](#)  
threading [165](#)  
threading module  
    about [165](#)  
    example [167-174](#)  
    importing [166](#)  
    methods [215](#)  
    used, for implementing new thread [176-182](#)  
thread synchronization  
    about [183-187](#)  
    event object, using [204-208](#)  
    lock, applying [187-190](#)  
    Race Condition [184](#)  
time complexity [627](#), [628](#)  
tkinter module  
    about [337](#), [338](#)  
    importing [338-342](#)  
Tool Command Language (TCL) [338](#)  
tree representation, as list of lists  
    about [523-528](#)  
    Tree Traversal Methods [530](#)  
trees  
    about [516](#)  
    definition [523](#)  
    depth [516](#)  
    external nodes [516](#)  
    internal nodes [516](#)

- leaf node [517](#)
- leaves [516](#)
- level of a node [516](#)
- recursive definition [517](#)
- root [516](#)
- simple tree representation [518-523](#)

Tree Traversal Methods

- In Order Traversal [533](#)
- Post Order Traversal [536](#)
- Preorder Traversal [530](#)

types of variables

- class variable or static variable [73](#)
- instance variable [73](#)

## U

- unbounded methods [90](#)

unit testing

- about [251](#)
- benefits [251, 252](#)
- objectives [252](#)

unit testing, in Python [253](#)

unittest module

- about [258](#)
- assertion methods [264](#)
- rules, for writing test methods [258](#)
- working with [258-261](#)

## V

- variable-length arguments [61](#)

virtual environment

- creating [612-614](#)
- installing, pip used [609-611](#)

void function [62](#)

## W

- web server gateway interface (WSGI) [608](#)

Where clause

- using, with Python [151-153](#)

widgets

- about [343](#)
- layout management [344](#)
- working with [343](#)

## Z

- ZeroDivisionError

- about [234, 235](#)

handling [236](#)