https://pythonschoolkvs.wordpress.com/

CLASS-XII (SESSION 2020-21)

SUBJECT: COMPUTER SCIENCE (083) – PYTHON

INDEX

By: Vikash Kumar Yadav

PGT-Computer Science

K.V. No.-IV ONGC Vadodara

CHAPTER-1

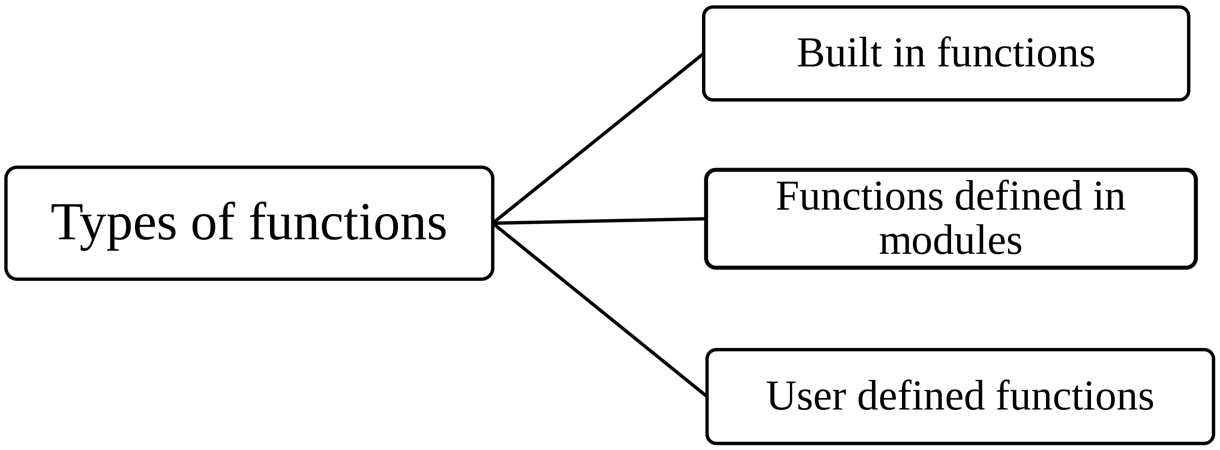
FUNCTIONS IN PYTHON

1.1 Definition: Functions are the subprograms that perform specific task. Functions are the small modules.

1.2 Types of Functions:

There are **three** types of functions in python:

1. Library Functions (Built in functions)
2. Functions defined in modules
3. User Defined Functions



1. Library Functions: These functions are already built in the python library.

2. Functions defined in modules: These functions defined in particular modules. When you want to use these functions in program, you have to import the corresponding module of that function.

3. User Defined Functions: The functions those are defined by the user are called user defined functions.

1. Library Functions in Python:

These functions are already built in the library of python.

For example: type( ), len( ), input( ) etc.

2. Functions defined in modules:

a. Functions of math module:

To work with the functions of math module, we must import math module in program.

import math

S. No.	Function	Description	Example
1	sqrt( )	Returns the square root of a number	>>>math.sqrt(49) 7.0
2	ceil( )	Returns the upper integer	>>>math.ceil(81.3) 82
3	floor( )	Returns the lower integer	>>>math.floor(81.3) 81
4	pow( )	Calculate the power of a number	>>>math.pow(2,3) 8.0
5	fabs( )	Returns the absolute value of a number	>>>math.fabs(-5.6) 5.6
6	exp( )	Returns the e raised to the power i.e. e^3	>>>math.exp(3) 20.085536923187668

b. Function in **random** module:

random module has a function randint( ).

- randint( ) function generates the random integer values including start and end values.
- Syntax: randint(start, end)
- It has two parameters. Both parameters must have integer values.

Example:

```
import random

n=random.randint(3,7)
```

*The value of n will be 3 to 7.

3. USER DEFINED FUNCTIONS:

The syntax to define a function is:

```
def   function-name ( parameters) :

          #statement(s)
```

Where:

- Keyword **def** marks the start of function header.
- A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (**:**) to mark the end of function header.
- One or more valid python statements that make up the function body. Statements must have same indentation level.
- An optional return statement to return a value from the function.

Example:

```
def display(name):
        print("Hello  " +   name    + "  How are you?")
```

1.3 Function Parameters:

A functions has two types of parameters:

1. Formal Parameter
2. Actual Parameter

1. **Formal Parameter**: Formal parameters are written in the function prototype and function header of the definition. Formal parameters are local variables which are assigned values from the arguments when the function is called.

Python supports two types of formal parameters:

i. Positional parameters
ii. Default parameters

i. **Positional parameter**: These are mandatory arguments. Value must be provided to these parameters and values should be matched with parameters.

Example:

Let a function defined as given below:

```
def Test(x,y,z):
        .
        .
        .
```

Then we can call the function using these possible function calling statements:

```
p,q,r = 4,5,6
Test(p,q,r)   # 3 variables which have values, are passed
Test(4,q,r)   # 1 Literal value and 2 variables are passed
Test(4,5,6)   # 3 Literal values are passed
```

So, x,y,z are positional parameters and the values must be provided these parameters.

ii. **Default Parameters:** The parameters which are assigned with a value in function header while defining the function, are known as default parameters. This values is optional for the parameter.

If a user explicitly passes the value in function call, then the value which is passed by the user, will be taken by the default parameter. If no value is provided, then the default value will be taken by the parameter.

Default parameters will be written in the end of the function header, means positional parameter cannot appear to the right side of default parameter.

Example:
Let a function defined as given below:

```
def CalcSI(p, rate, time=5):      # time is default parameter here
        .
        .
        .
```

Then we can call the function using these possible function calling statements:

```
CalcSI(5000, 4.5)   # Valid, the value of time parameter is not provided, so it will take
                    # default value, which is 5.
CalcSI(5000,4.5, 6)  # Valid, Value of time will be 6
```

Valid/Invalid examples to define the function with default arguments:

```
CalcSI(p, rate=4.5, time=5):      #Valid
CalcSI(p, rate=4.5, time=5):      # Valid
CalcSI(p, rate=4.5, time):          #Invalid, Positional argument cannot come after default
                                    #parameter
CalcSI(p=5000, rate=4.5, time=5):   #Valid
```

2. **Actual Parameter**: When a function is *called*, the values that are passed in the call are called *actual parameters*. At the time of the call each actual parameter is assigned to the corresponding formal parameter in the function definition.

Example :

```
def ADD(x, y):                 #Defining a function and x and y are formal parameters
    z=x+y
    print("Sum = ", z)
a=float(input("Enter first number: " ))
b=float(input("Enter second number: " ))
ADD(a,b)       #Calling the function by passing actual parameters
```

In the above example, **x** and **y** are formal parameters. **a** and **b** are actual parameters.

1.4 Calling the function:

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Syntax:

```
function-name(parameter)
```

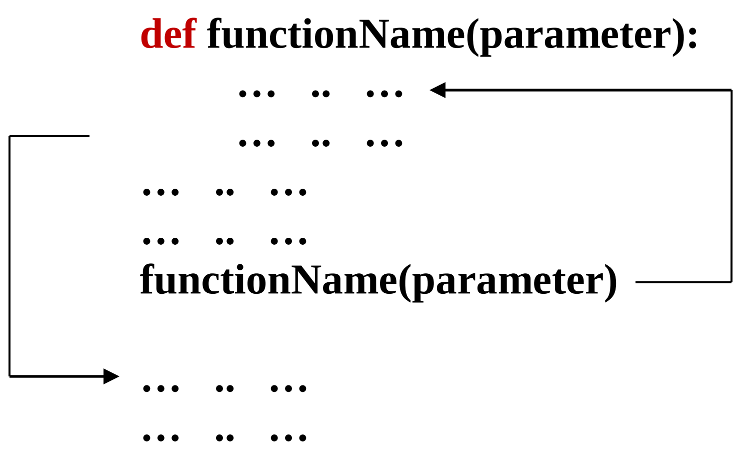
Example:

```
ADD(10,20)
```

OUTPUT:

```
Sum = 30.0
```

How function works?



1.5 The return statement:

The **return** statement is used to exit a function and go back to the place from where it was called.

There are two types of functions according to return statement:

a. Function returning some value (non-void function)

b. Function not returning any value (void function)

a. Function returning some value (non-void function) :

Syntax:

return expression/value

Example-1: Function returning one value

```
def my_function(x):
        return 5 * x
```

Example-2 Function returning multiple values:

```
def sum(a,b,c):
        return a+5, b+4, c+7
S=sum(2,3,4)        # S will store the returned values as a tuple
print(S)
```

OUTPUT:

```
(7, 7, 11)
```

Example-3: Storing the returned values separately:

```
def sum(a,b,c):
        return a+5, b+4, c+7
s1, s2, s3=sum(2, 3, 4)          # storing the values separately
print(s1, s2, s3)
```

OUTPUT:

```
7  7  11
```

b. Function not returning any value (void function) : The function that performs some operationsbut does not return any value, called void function.

```
def message():
   print("Hello")

m=message()
print(m)
```

OUTPUT:

Hello

None

1.6 Scope and Lifetime of variables:

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

There are two types of scope for variables:

1. Local Scope

2. Global Scope

1. **Local Scope:** Variable used inside the function. It can not be accessed outside the function. In this scope, The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

2. **Global Scope:** Variable can be accessed outside the function. In this scope, Lifetime of a variable is the period throughout which the variable exits in the memory.

Example:

```
def my_func():
    x = 10
    print("Value inside function:",x)

x = 20
my_func()
print("Value outside function:",x)
```

OUTPUT:

```
Value inside function: 10

Value outside function: 20
```

Here, we can see that the value of **x** is **20** initially. Even though the function my_func()changed the value of **x** to **10**, it did not affect the value outside the function.

This is because the variable **x** inside the function is different (local to the function) from the one outside. Although they have same names, they are two different variables with different scope.

On the other hand, variables outside of the function are visible from inside. They have a **global** scope.

We can read these values from inside the function but cannot change (write) them. In order to modify the value of variables outside the function, they must be declared as global variables using the keyword **global**.

1.7 Passing Strings, Lists, Tuples and Dictionaries to functions:

1.7.1 Passing Strings to Function:

Example:

```
def StrPass(s):
    for i in s:
        print(i,end='')

string="PythonClassXII"
StrPass(string)
```

Output:

PythonClassXII

1.7.2 Passing List to function:

Example:

```
def ListPass(L):
    for i in L:
        print(i)

List=['Physics','CS','Chemistry','English','Maths']
ListPass(List)
```

Output:

```
Physics
CS
Chemistry
English
Maths
```

Note: Same process for tuple and dictionaries.

Programs related to Functions in Python topic:

1. Write a python program to sum the sequence given below. Take the input ***n*** from the user.

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}$$

Solution:

```
def fact(x):
   j=1
   res=1
   while j<=x:
      res=res*j
      j=j+1
      return res
n=int(input("enter the number : "))
i=1
sum=1
while i<=n:
 f=fact(i)
 sum=sum+1/f
 i+=1
print(sum)
```

2. Write a program to compute GCD and LCM of two numbers

```
def gcd(x,y):
   while(y):
     x, y = y, x % y
   return x

def lcm(x, y):
   lcm = (x*y)//gcd(x,y)
   return lcm

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

print("The L.C.M. of", num1,"and", num2,"is", lcm(num1, num2))
print("The G.C.D. of", num1,"and", num2,"is", gcd(num1, num2))
```

CHAPTER-2

RECURSION

2.1 Definition: A function calls itself, is called recursion.

2.2 Python program to find the **factorial of a number** using recursion:

Program:

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


num=int(input("enter the number: "))
if num < 0:
        print("Sorry, factorial does not exist for negative numbers")
elif num = = 0:
        print("The factorial of 0 is 1")
else:
        print("The factorial of  ",num,"  is  ", factorial(num))
```

OUTPUT:

```
enter the number: 5
The factorial of   5   is   120
```

2.3 Python program to print the Fibonacci series using recursion:

Program:

```
def fibonacci(n):
   if n<=1:
        return n
   else:
        return(fibonacci(n-1)+fibonacci(n-2))

num=int(input("How many terms you want to display: "))
for i in range(num):
        print(fibonacci(i)," ", end=" ")
```

OUTPUT:

```
How many terms you want to display: 8
0   1   1   2   3   5   8   13
```

2.4 Sum of n Natural Numbers using recursion:

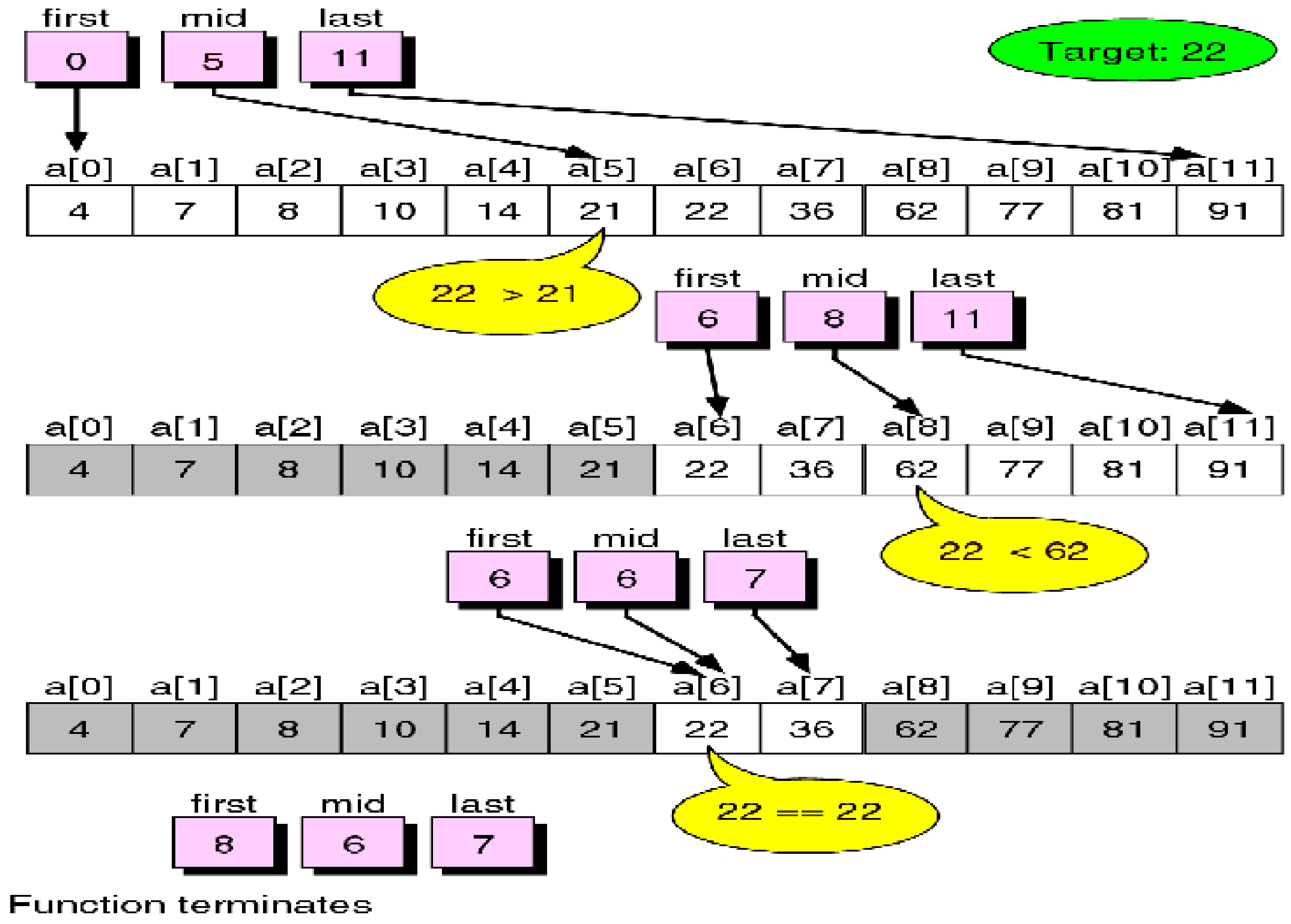
```
def SumNatural(num):
        if num<=1:
                return 1
        else:
                 return num+SumNatural(num-1)

n=int(input("Sum of How Many Natural Numbers :  "))
sum=SumNatural(n)
print("Sum of", n, "Natural numbers is:",sum)
```

OUTPUT:

```
Sum of How Many Natural Numbers :  6
Sum of 6 Natural numbers is: 21
```

2.5 Binary Search using recursion:

Note: The given array or sequence must be sorted to perform binary search.



Program:

```
def Binary_Search(sequence, item, LB, UB):
     if LB>UB:
          return -5               # return any negative value
     mid=int((LB+UB)/2)
     if item==sequence[mid]:
          return mid
     elif item<sequence[mid]:
          UB=mid-1
          return Binary_Search(sequence, item, LB, UB)
```

```
        else:
                LB=mid+1
                return Binary_Search(sequence, item, LB, UB)

L=eval(input("Enter the elements in sorted order: "))
n=len(L)
element=int(input("Enter the element that you want to search :"))
found=Binary_Search(L,element,0,n-1)
if found>=0:
        print(element, "Found at the index : ",found)
else:
        print("Element not present in the list")
```

CHAPTER-3

FILE HANDLING

3.1 INTRODUCTION:

File:- A file is a collection of related data stored in a particular area on the disk.

Stream: - It refers to a sequence of bytes.

Need of File Handling:

- File handling is an important part of any web application.
- To store the data in secondary storage.
- To Access the data fast.
- To perform Create, Read, Update, Delete operations easily.

3.2 Data Files:

Data files can be stored in two ways:

1. **Text Files**: Text files are structured as a sequence of lines, where each line includes a sequence of characters.
2. **Binary Files** : A binary file is any type of file that is not a text file.

S. No.	Text Files	Binary Files
1.	Stores information in ASCII characters.	Stores information in the same format which the information is held in memory.
2.	Each line of text is terminated with a special character known as EOL (End of Line)	No delimiters are used for a line.
3.	Some internal translations take place when this EOL character is read or written.	No translation occurs in binary files.
4.	Slower than binary files.	Binary files are faster and easier for a program to read and write the text files.

3.3 Opening and closing a file:

3.3.1 Opening a file:

To work with a file, first of all you have to open the file. To open a file in python, we use open( ) function.

The **open( )** function takes two parameters; *filename*, and *mode*. **open( )** function returns a file object.

Syntax:

file_objectname= open(filename, mode)

Example:

To open a file for reading it is enough to specify the name of the file:

f = open("book.txt")

The code above is the same as:

f = open("book.txt", "rt")

Where "r" for read mode, and "t" for text are the default values, you do not need to specify them.

3.3.2 Closing a file:

After performing the operations, the file has to be closed. For this, a close( ) function is used to close a file.

Syntax:

file-objectname.close( )

3.4 File Modes:

Text file mode	**Binary File Mode**	**Description**
'r'	'rb'	Read - Default value. Opens a file for reading, error if the file does not exist.
'w'	'wb'	Write - Opens a file for writing, creates the file if it does not exist
'a'	'ab'	Append - Opens a file for appending, creates the file if it does not exist
'r+'	'rb+'	Read and Write-File must exist, otherwise error is raised.

‘w+’	‘wb+’	Read and Write-File is created if does not exist.
‘a+’	‘ab+’	Read and write-Append new data
‘x’	‘xb’	Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

“t” – Text-Default value. Text mode

“b” – Binary- Binary Mode (e.g. images)

3.5 WORKING WITH TEXT FILES:

3.5.1 Basic operations with files:

a. **Read** the data from a file

b. **Write** the data to a file

c. **Append** the data to a file

d. **Delete** a file

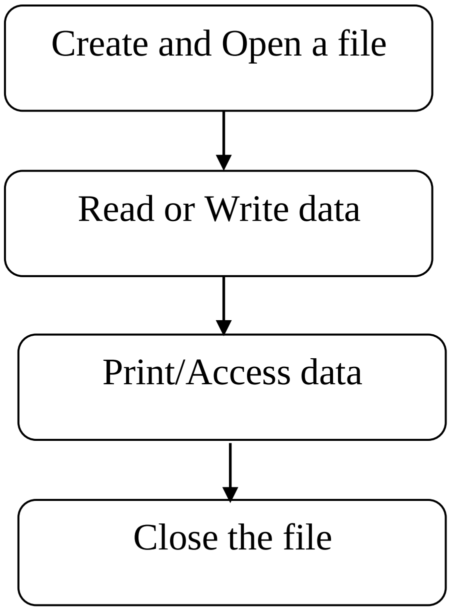
a. Read the data from a file:

There are **3 types** of functions to read data from a file.

- **read( ) :** reads n bytes. if no n is specified, reads the entire file.
- **readline( ) :** Reads a line. if n is specified, reads n bytes.
- **readlines( ):** Reads all lines and returns a list.

Steps to read data from a file:

- Create and Open a file using open( ) function
- use the read( ), readline( ) or readlines( ) function
- print/access the data
- Close the file.



Let a text file **“Book.txt”** has the following text:

“Python is interactive language. It is case sensitive language.
It makes the difference between uppercase and lowercase letters.
It is official language of google.”

Example-1: Program using read( ) function:

fin=open("D:\\python programs\\Book.txt",'r')
str=fin.read( )
print(str)
fin.close( )
	fin=open("D:\\python programs\\Book.txt",'r')
	str=fin.read(10)
	print(str)
	fin.close( )

OUTPUT: Python is interactive language. It is case sensitive language. It makes the difference between uppercase and lowercase letters. It is official language of google.	**OUTPUT:** Python is

Example-2: using readline( ) function:

```
fin=open("D:\\python programs\\Book.txt",'r')
str=fin.readline( )
print(str)
fin.close( )
```

OUTPUT:

```
Python is interactive language. It is case sensitive language.
```

Example-3: using readlines( ) function:

```
fin=open("D:\\python programs\\Book.txt",'r')
str=fin.readlines( )
print(str)
fin.close( )
```

OUTPUT:

```
['Python is interactive language. It is case sensitive language.\n', 'It makes the difference between uppercase and lowercase letters.\n', 'It is official language of google.']
```

❖ **Some important programs related to read data from text files:**

Program-a: Count the number of characters from a file. (Don’t count white spaces)

```
fin=open("Book.txt",'r')
str=fin.read( )
L=str.split( )
count_char=0
for i in L:
    count_char=count_char+len(i)
print(count_char)
fin.close( )
```

Program-b: Count the number of words in a file.

```
fin=open("Book.txt",'r')
str=fin.read( )
L=str.split( )
count_words=0
for i in L:
    count_words=count_words+1
print(count_words)
fin.close( )
```

Program-c: Count number of lines in a text file.

```
fin=open("Book.txt",'r')
str=fin.readlines()
count_line=0
for i in str:
    count_line=count_line+1
print(count_line)
fin.close( )
```

Program-d: Count number of vowels in a text file.

```
fin=open("D:\\python programs\\Book.txt",'r')
str=fin.read( )
```

```
count=0
for i in str:
    if i=='a' or i=='e' or i=='i' or i=='o' or i=='u':
        count=count+1
print(count)
fin.close( )
```

Program-e : Count the number of 'is' word in a text file.

```
fin=open("D:\\python programs\\Book.txt",'r')

str=fin.read( )

L=str.split( )

count=0

for i in L:

    if i=='is':

        count=count+1

print(count)

fin.close( )
```

b. Write data to a file:

There are **2 types** of functions to write the data to a file.

- **write( ):**Write the data to a file. Syntax:

  write(string)   (for text files)

  write(byte_string)   (for binary files)

- **writelines( )**: Write all strings in a list L as lines to file.

To write the data to an existing file, you have to use the following mode:

"w" - Write - will overwrite any existing content

Example: Open the file "Book.txt" and append content to the file:

```
fout= open("Book.txt", "a")
fout.write("Welcome to the world of programmers")
```

Example: Open the file "Book.txt" and overwrite the content:

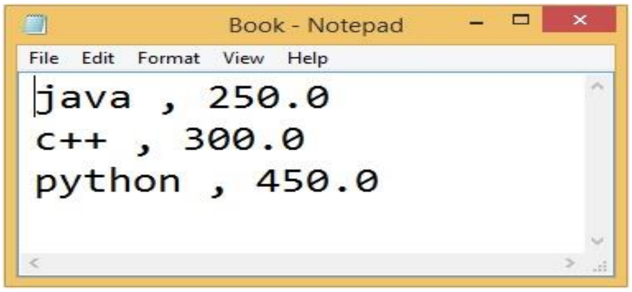
```
fout = open("Book.txt", "w")
fout.write("It is creative and innovative")
```

Program: Write a program to take the details of book from the user and write the record in text file.

```
fout=open("D:\\python programs\\Book.txt",'w')
n=int(input("How many records you want to write in a file ? :"))
for i in range(n):
    print("Enter details of record :", i+1)
    title=input("Enter the title of book : ")
    price=float(input("Enter price of the book: "))
    record=title+" , "+str(price)+'\n'
    fout.write(record)
fout.close( )
```

OUTPUT:

```
How many records you want to write in a file ? :3
Enter details of record : 1
Enter the title of book : java
Enter price of the book: 250
Enter details of record : 2
Enter the title of book : c++
Enter price of the book: 300
Enter details of record : 3
Enter the title of book : python
Enter price of the book: 450
```



```
java , 250.0
c++ , 300.0
python , 450.0
```

c. Append the data to a file:

This operation is used to add the data in the end of the file. It doesn’t overwrite the existing data in a file. To write the data in the end of the file, you have to use the following mode:

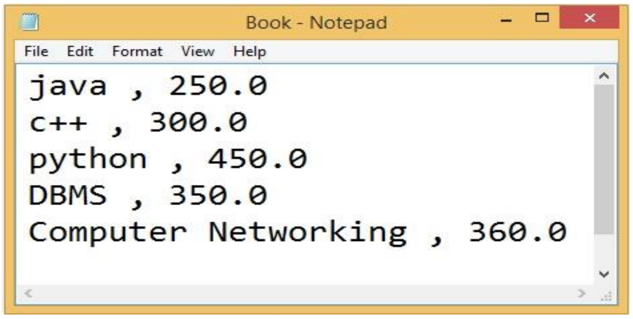
"a" - Append - will append to the end of the file.

Program: Write a program to take the details of book from the user and write the record in the end of the text file.

```
fout=open("D:\\python programs\\Book.txt",'a')
n=int(input("How many records you want to write in a file ? :"))
for i in range(n):
    print("Enter details of record :", i+1)
    title=input("Enter the title of book : ")
    price=float(input("Enter price of the book: "))
    record=title+" , "+str(price)+'\n'
    fout.write(record)
fout.close( )
```

OUTPUT:

```
How many records you want to write in a file ? :2
Enter details of record : 1
Enter the title of book : DBMS
Enter price of the book: 350
Enter details of record : 2
Enter the title of book : Computer
Networking
Enter price of the book: 360
```



```
java , 250.0
c++ , 300.0
python , 450.0
DBMS , 350.0
Computer Networking , 360.0
```

d. Delete a file: To delete a file, you have to import the **os** module, and use **remove( )** function.

```
import os
os.remove("Book.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

Check if file exists, *then* delete it:

```
import os
if os.path.exists("Book.txt"):
  os.remove("Book.txt")
else:
  print("The file does not exist")
```

3.6 WORKING WITH BINARY FILES:

Binary files store data in the binary format (0's and 1's) which is understandable by the machine. So when we open the binary file in our machine, it decodes the data and displays in a human-readable format.

(a) Write data to a Binary File: Pickle is a special python package that is used to generate data in binary format. Pickle comes with few methods like dump( ) to write data in binary format.

Example:

```
import pickle
list =[ ]    # empty list
while True:
    roll = input("Enter student Roll No:")
    sname  = input("Enter student Name :")
    student = {"roll":roll,"name":sname}   # create a dictionary
    list.append(student)                          # add the dictionary as an element in the list
    choice= input("Want to add more record(y/n) :")
    if(choice=='n'):
        break

file = open("student.dat","wb")                # open file in binary and write mode
pickle.dump(list, file)
file.close( )
```

OUTPUT:

```
Enter student Roll No:1201
Enter student Name :Anil
Want to add more record(y/n) :y
```

```
Enter student Roll No:1202
Enter student Name :Sunil
Want to add more record(y/n) :n
```

(b) Read data from a Binary File: To read the data from a binary file, we have to use load( ) function of pickle module.

Example:

```
import pickle


file = open("student.dat", "rb")
list = pickle.load(file)
print(list)
file.close( )
```

OUTPUT:

```
[{'roll': '1201', 'name': 'Anil'}, {'roll': '1202', 'name': 'Sunil'}]
```

(c) Update a record in Binary File:

```
import pickle
roll = input('Enter roll number whose name you want to update in binary file :')
file = open("student.dat", "rb+")
list = pickle.load(file)
found = 0
lst = [ ]
for x in list:
    if roll in x['roll']:
        found = 1
        x['name'] = input('Enter new name: ')
    lst.append(x)
if found == 1:
    file.seek(0)
    pickle.dump(lst, file)
    print("Record Updated")
else:
    print('roll number does not exist')

file.close( )
```

OUTPUT:

```
Enter roll number whose name you want to update in binary file :1202

Enter new name: Harish

Record Updated
```

(d) Delete a record from binary file:

```
import pickle
roll = input('Enter roll number whose record you want to delete:')
file = open("student.dat", "rb+")
list = pickle.load(file)
found = 0
lst = []
for x in list:
    if roll not in x['roll']:
        lst.append(x)
    else:
        found = 1

if found == 1:
    file.seek(0)
    pickle.dump(lst, file)
    print("Record Deleted ")
else:
    print('Roll Number does not exist')

file.close( )
```

OUTPUT:

```
Enter roll number whose record you want to delete:1201

Record Deleted
```

(e) Search a record in a binary file:

```
import pickle
roll = input('Enter roll number that you want to search in binary file :')
file = open("student.dat", "rb")
list = pickle.load(file)
file.close( )
for x in list:
    if roll in x['roll']:
        print("Name of student is:", x['name'])
        break
else:
    print("Record not found")
```

OUTPUT:

```
Enter roll number that you want to search in binary file :1202

Name of student is: Harish
```

3.7 tell( ) and seek( ) methods:

- **tell( ):** It returns the current position of cursor in file.

Example:

```
fout=open("story.txt","w")
fout.write("Welcome Python")
print(fout.tell( ))
fout.close( )
```

Output:

```
14
```

- **seek(offset) :** Change the cursor position by bytes as specified by the offset.

Example:

```
fout=open("story.txt","w")
fout.write("Welcome Python")
fout.seek(5)
print(fout.tell( ))
fout.close( )
```

Output:

```
5
```

3.8 File I/O Attributes

Attribute	Description
name	Returns the name of the file (Including path)
mode	Returns mode of the file. (r or w etc.)
encoding	Returns the encoding format of the file
closed	Returns True if the file closed else returns False

Example:

```
f = open("D:\\story.txt", "r")
print("Name of the File:  ", f.name)
print("File-Mode :  ", f.mode)
print("File encoding format : ", f.encoding)
print("Is File closed? ", f.closed)
f.close()
print("Is File closed? ", f.closed)
```

OUTPUT:

```
Name of the File:   D:\story.txt
File-Mode :   r
File encoding format :  cp1252
Is File closed?  False
Is File closed?  True
```

3.9 Pickle Module:

Python Pickle is used to serialize and deserialize a python object structure. Any object on python can be pickled so that it can be saved on disk.

Pickling: Pickling is the process whereby a **Python** object hierarchy is converted into a byte stream.

Unpickling: A byte stream is converted into object hierarchy.

To use the picking methods in a program, we have to import pickle module using import keyword.

Example:

import pickle

In this module, we shall discuss to functions of pickle module, which are:

i. **dump( ) :** To store/write the object data to the file.
ii. **load( ) :** To read the object data from a file and returns the object data.

Example:

Write the object to the file: dump( )

```
import pickle
L=[]
roll = int(input('Enter Roll Number  : '))
name=input('Enter name of student :')
marks=eval(input('Enter the marks :'))

L.append(roll)
L.append(name)
L.append(marks)

file = open('studentdata', 'wb')
pickle.dump(L, file)   #dump information to that file
file.close()
```

OUTPUT:

```
Enter Roll Number  : 1201
Enter name of student :ABC
Enter the marks :85.83
```

Read the object from a file: load( )

```
import pickle
file = open('studentdata', 'rb')
L = pickle.load(file)  # read information from the file
file.close()

print('Showing the pickled data:')

print('Roll Number :',L[0])
print('Name :', L[1])
print('Marks :',L[2])
```

OUTPUT:

```
Showing the pickled data:
Roll Number : 1201
Name : ABC
Marks : 85.23
```

3.10 CSV Files:

CSV (Comma Separated Values). A csv file is a type of plain text file that uses specific structuring to arrange tabular data. csv is a common format for data interchange as it is compact, simple and general. Each line of the file is one line of the table. csv files have .csv as file extension.

Let us take a **data.csv** file which has the following contents:

```
Roll No., Name of student, stream, Marks
1, Anil, Arts, 426
2, Sujata, Science, 412
```

As you can see each row is a new line, and each column is separated with a comma. This is an example of how a CSV file looks like.

To work with csv files, we have to import the **csv module** in our program.

3.10.1 Read a CSV file:

To read data from a CSV file, we have to use reader( ) function.
The reader function takes each row of the file and make a list of all columns.

CODE:

```
import csv
with open('C:\\data.csv','rt') as f:
    data = csv.reader(f)      #reader function to generate a reader object
      for row in data:
         print(row)
```

OUTPUT:

```
['Roll No.', 'Name of student', 'stream', 'Marks']
['1', 'Anil', 'Arts', '426']
['2', 'Sujata', 'Science', '412']
```

3.10.2 Write data to a CSV file:

When we want to write data in a CSV file you have to use writer( ) function. To iterate the data over the rows (lines), you have to use the writerow( ) function.

CODE:

```
import csv
with open('C:\\data.csv', mode='a', newline='') as file:
    writer = csv.writer(file, delimiter=',', quotechar='"' )
     #write new record in file
     writer.writerow(['3', 'Shivani', 'Commerce', '448'])
     writer.writerow(['4', 'Devansh', 'Arts', '404'])
```

OUTPUT:

```
['Roll No.', 'Name of student', 'stream', 'Marks']
['1', 'Anil', 'Arts', '426']
['2', 'Sujata', 'Science', '412']
['3', 'Shivani', 'Commerce', '448']
['4', 'Devansh', 'Arts', '404']
```

When we shall open the file in notepad (Flat file) then the contents of the file will look like this:

```
Roll No.,Name of student,stream,Marks
1,Anil,Arts,426
2,Sujata,Science,412
3,Shivani,Commerce,448
4,Devansh,Arts,404
```

CHAPTER-4

CREATE & IMPORT PYTHON LIBRARIES

4.1 INTRODUCTION:

Python has its in-built libraries and packages. A user can directly import the libraries and its modules using ***import*** keyword. If a user wants to create libraries in python, he/she can create and import libraries in python.

4.2 How to create Libraries/Package in Python?

To create a package/Library in Python, we need to follow these three simple steps:

1. First, we create a directory and give it a package name. Name should be meaningful.
2. Then create the python files (Modules) which have classes and functions and put these **.py** files in above created directory.
3. Finally we create an __init__.py file inside the directory, to let Python know that the directory is a package.

Example:

Let's create a package named **Shape** and build three modules in it namely **Rect, Sq** and **Tri** to calculate the area for the shapes rectangle, square and triangle.

Step-1 First create a directory and name it **Shape**. (In this case it is created under C:\Users\ViNi\AppData\Local\Programs\Python\Python37-32\ directory path.)

Step-2 Create the modules in **Shape** directory.

To create **Module-1**(**Rect.py**), a file with the name Rect.py and write the code in it.

```
class Rectangle:
    def __init__(self):
        print("Rectangle")

    def Area(self, length, width):
        self.l=length
        self.w=width
        print("Area of Rectangle is : ", self.l*self.w)
```

To create **Module-2**(**Sq.py**), a file with the name Sq.py and write the code in it.

```
class Square:
   def __init__(self):
      print("Square")
   def Area(self, side):
      self.a=side
      print("Area of square is : ", self.a*self.a)
```

To create **Module-3** (**Tri.py**), a file with the name Tri.py and write the code in it.

```
class Triangle:
   def __init__(self):
      print("Trinagle")

   def Area(self, base, height):
      self.b=base
      self.h=height
      ar= (1/2)*self.b*self.h
      print("Area of Triangle is : ", ar )
```

Step-3 Create the __init__.py file. This file will be placed inside **Shape** directory and can be left blank or we can put this initialisation code into it.

```
from Shape import Rect
from Shape import Sq
from Shape import Tri
```

That's all. Package created. Name of the package is Shape and it has three modules namely Rect, Sq and Tri.

4.3 How to use the package in a program:

Following steps should be followed to use the created package.

Step-1 Create a file main.py in the same directory where **Shape** package is located.

Step-2 Write the following code in main.py file

```
from Shape import Rect
from Shape import Sq
from Shape import Tri


r=Rect.Rectangle( )    #Create an object r for Rectangle class
r.Area(10,20)       # Call the module Area( ) of Rectangle class by passing argument

s=Sq.Square( )      #Create an object s for Square class
s.Area(10)         # Call the module Area( ) of Square class by passing argument

t=Tri.Triangle( )   #Create an object t for Triangle class
t.Area(6,8)      # Call the module Area( ) of Triangle class by passing argument
```

OUTPUT:

```
Rectangle

Area of Rectangle is :  200

Square

Area of square is :  100

Trinagle

Area of Triangle is :  24.0
```

4.4 Import the package in program:

Method-1

To use the package/library in a program, we have to import the package in the program. For this we use **import** keyword.

If we simply import the Shape package, then to access the modules and functions of **Shape** package, we have to write the following code:

Syntax:

Package-name.Module-name.function-name(parameter)

Example:

import Shape

r=Shape.Rect.Rectangle( )

s=Shape.Sq.Square( )

t=Shape.Tri.Triangle( )

Method-2:

If we want to access a specific module of a package then we can use ***from*** and ***import*** keywords.

Syntax:

from package-name import module-name

Example:

from Shape import Rect

r=Rect.Rectangle( )

s=Sq.Square( )

t=Tri.Triangle( )

Method-3:

If a user wants to import all the modules of Shape package then he/she can use ***** (asterisk).

Example:

from Shape import *

r=Rect.Rectangle( )

s=Sq.Square( )

t=Tri.Triangle( )

CHAPTER-5

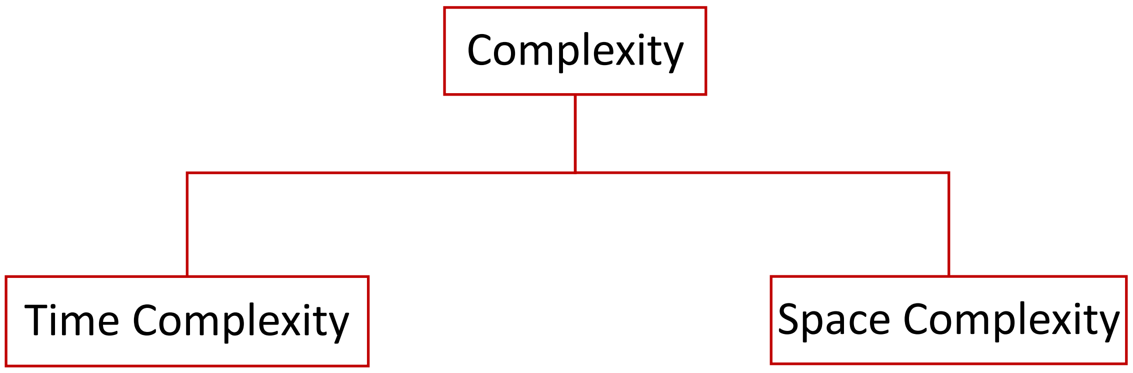
IDEA OF EFFICIENCY

5.1 INTRODUCTION:

Program efficiency is a property of a program related to time taken by a program for execution and space used by the program. It is also related to number of inputs taken by a program.

Complexity: To measure the efficiency of an algorithm or program in terms of space and time.

The program which uses minimum number of resources, less space and minimum time, is known as good program.



Time and space complexity depends on lots of things like hardware, operating system, processors, size of inputs etc.

Time Complexity of an algorithm:

- Best case
- Average case
- Worst case

- **Best case:** The minimum number of steps that an algorithm can take for any input data values.
- **Average case**: The efficiency averaged on all possible inputs. We normally assume the uniform distribution.
- **Worst case**: The maximum number of steps that an algorithm can take for any input data values.

Complexity of certain well known algorithms:

(a) Linear Search : O(n)

(b) Binary Search : O(log n)

(c) Bubble Sort : O(n^2)

5.2 Calculate Number of Operations:

Program (Bubble Sort):

```
L=eval(input("Enter the elements:"))

n=len(L)

for p in range(0,n-1):

    for i in range(0,n-1):

        if L[i]>L[i+1]:

            L[i], L[i+1] = L[i+1],L[i]

print("The sorted list is : ", L)
```

Step	CODING	No. of Operations
1	L=eval(input("Enter the elements:"))	**1**
2	n=len(L) #for example n=7	**1**
3	for p in range(0,n-1):	one operation for each pass (executes **6** times, 0 to 5)
4	for i in range(0,n-1):	executes 6 times for elements, same will repeat 6 times under each pass (outer loop) so 6x6=**36** operations
5	if L[i]>L[i+1]:	executes 6 times for comparisons in each pass, same will repeat 6 times under each pass (outer loop) so 6x6=**36** operations
6	L[i], L[i+1] = L[i+1],L[i]	statement related to step-5, so 6x6=**36** operations
7	print("The sorted list is : ", L)	**1**
		TOTAL: 1+1+6+36+36+36+1=117 operations

CHAPTER-6

DATA STRUCTURE-I (LINEAR LIST)

7.1 INTRODUCTION:

Definition: The logical or mathematical model of a particular organization of data is called data structure. It is a way of storing, accessing, manipulating data.

7.2 TYPES OF DATA STRUCTURE:

There are two types of data structure:

1. Linear data structure

2. Non-Linear data structure

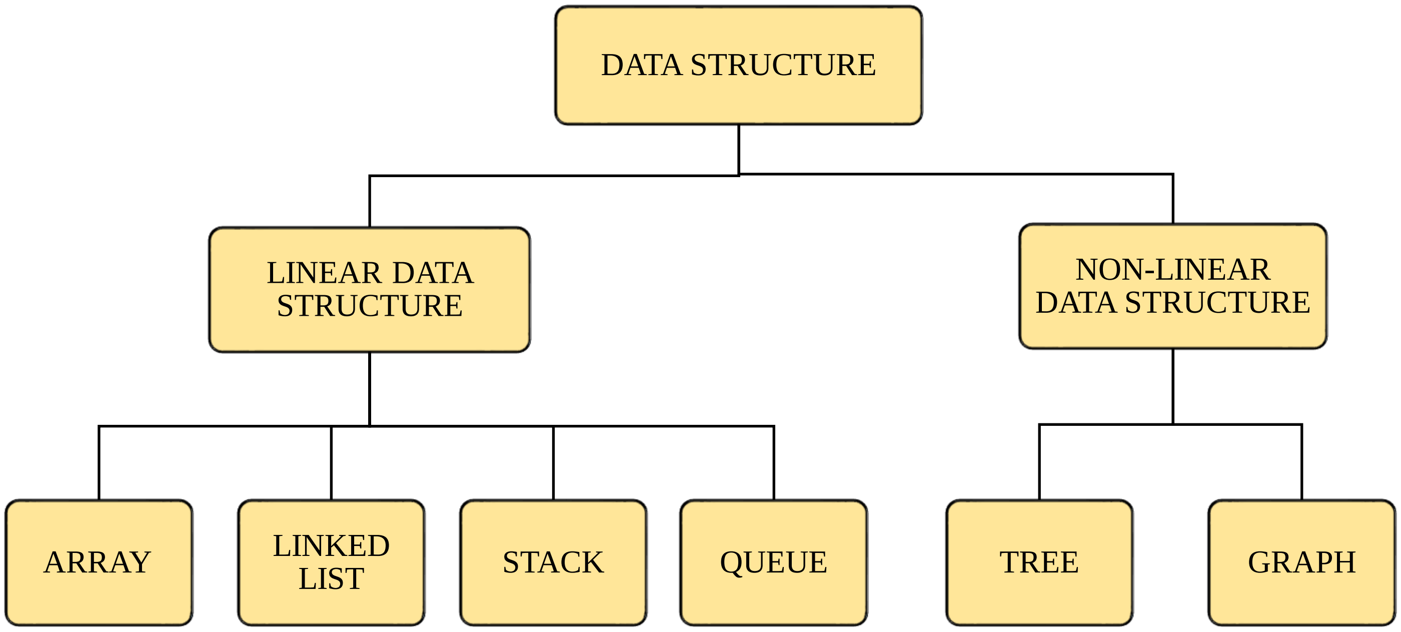


Fig: Types of data structure

1. Linear data structure: It is simple data structure. The elements in this data structure creates a sequence. Example: Array, linked list, stack, queue.

2. Non-Linear data structure: The data is not in sequential form. These are multilevel data structures. Example: Tree, graph.

7.3 OPERATION ON DATA STRUCTURE:

There are various types of operations can be performed with data structure:

1. **Traversing**: Accessing each record exactly once.
2. **Insertion**: Adding a new element to the structure.
3. **Deletion**: Removing element from the structure.
4. **Searching**: Search the element in a structure.
5. **Sorting**: Arrange the elements in ascending and descending order.
6. **Merging**: Joining two data structures of same type. (not covered in syllabus)

7.4 ARRAY (LISTS) IN DATA STRUCTURE:

An array or list is the collection of elements in ordered way. There are two types of arrays:

1. One dimensional array (1-D Lists)
2. Multi-dimensional array (Nested Lists)

7.4.1. One Dimensional array: It is the collection of homogeneous elements in an order.

a. **Traversing 1-D array (List):**

```
L=[10,20,30,40,50]
n=len(L)
for i in range(n):
    print(L[i])
```

Output:

```
10

20

30

40

50
```

b. Inserting Element in a list: There are two ways to insert an element in a list:

(i) If the array is not sorted

(ii) If the array is sorted

(i) If the array is not sorted: In this case, enter the element at any position using insert( ) function or add the element in the last of the array using append( ) function.

Example:

```
L=[15,8,25,45,13,19]

L.insert(3, 88)                # insert element at the index 3

print(L)
```

Output:

```
[15, 8, 25, 88, 45, 13, 19]
```

(ii) If the array is sorted: In this case, import bisect module and use the functions bisect( ) and insort( ).

bisect( ) : identifies the correct index for the element and returns the index.

insort( ): Inserts the element in the list in its correct order.

Example:

```
import bisect

L=[10,20,30,40,50]

print("Array before insertion the element:", L)

item=int(input("Enter the element to insert in array: "))

pos=bisect.bisect(L,item)                   #will return the correct index for item

bisect.insort(L,item)                      #will insert the element

print("Element inserted at index: ", pos)

print("Array after insertion the element : ", L)
```

OUTPUT:

```
Array before insertion the value: [10, 20, 30, 40, 50]

Enter the element to insert in array: 35

Element inserted at index : 3

Array after insertion the element :  [10, 20, 30, 35, 40, 50]
```

Note: bisect( ) works only with that lists which are arranged in ascending order.

c. Deletion of an element from a List: To delete an element from a list we can use remove( ) or pop( ) method.

Example:

```
L=[10,15,35,12,38,74,12]

print("List Before deletion of element: ", L)

val=int(input("Enter the element that you want to delete: "))

L.remove(val)

print("After deletion the element", val,"the list is: ", L)
```

OUTPUT:

```
List Before deletion of element:  [10, 15, 35, 12, 38, 74, 12]

Enter the element that you want to delete: 12

After deletion the element 12 the list is:  [10, 15, 35, 38, 74, 12]
```

d. Searching in a List:

There are two types of searching techniques we can use to search an element in a list. These are:

(i) Linear Search

(ii) Binary Search

(i) Linear Search: It is a simple searching technique.

Program:

```
L=eval(input("Enter the elements: "))

n=len(L)

item=eval(input("Enter the element that you want to search : "))

for i in range(n):

    if L[i]==item:

        print("Element found at the position :", i+1)
        break
else:
    print("Element not Found")
```

Output:

```
Enter the elements: 56,78,98,23,11,77,44,23,65
Enter the element that you want to search : 23
Element found at the position : 4
```

(ii) Binary Search: (Theory already discussed in the chapter recursion).

Program:

```
def BinarySearch(LIST,n,item):
    LB=0
    UB=n-1
    while LB<=UB:
        mid=int((LB+UB)/2)
        if item<LIST[mid]:
            UB=mid-1
        elif item>LIST[mid]:
            LB=mid+1
        else:
            return mid
    else:
        return -1
L=eval(input("Enter the elements in sorted order: "))
size=len(L)
element=int(input("Enter the element that you want to search :"))
found=BinarySearch(L,size,element)
```

```
    if found>=0:
        print(element, "Found at the position : ",found+1)
    else:
        print("Element not present in the list")
```

OUTPUT:

```
Enter the elements in sorted order: [12,23,31,48,51,61,74,85]
Enter the element that you want to search : 61
61 Found at the position :  6
```

Linear Search Vs Binary Search:

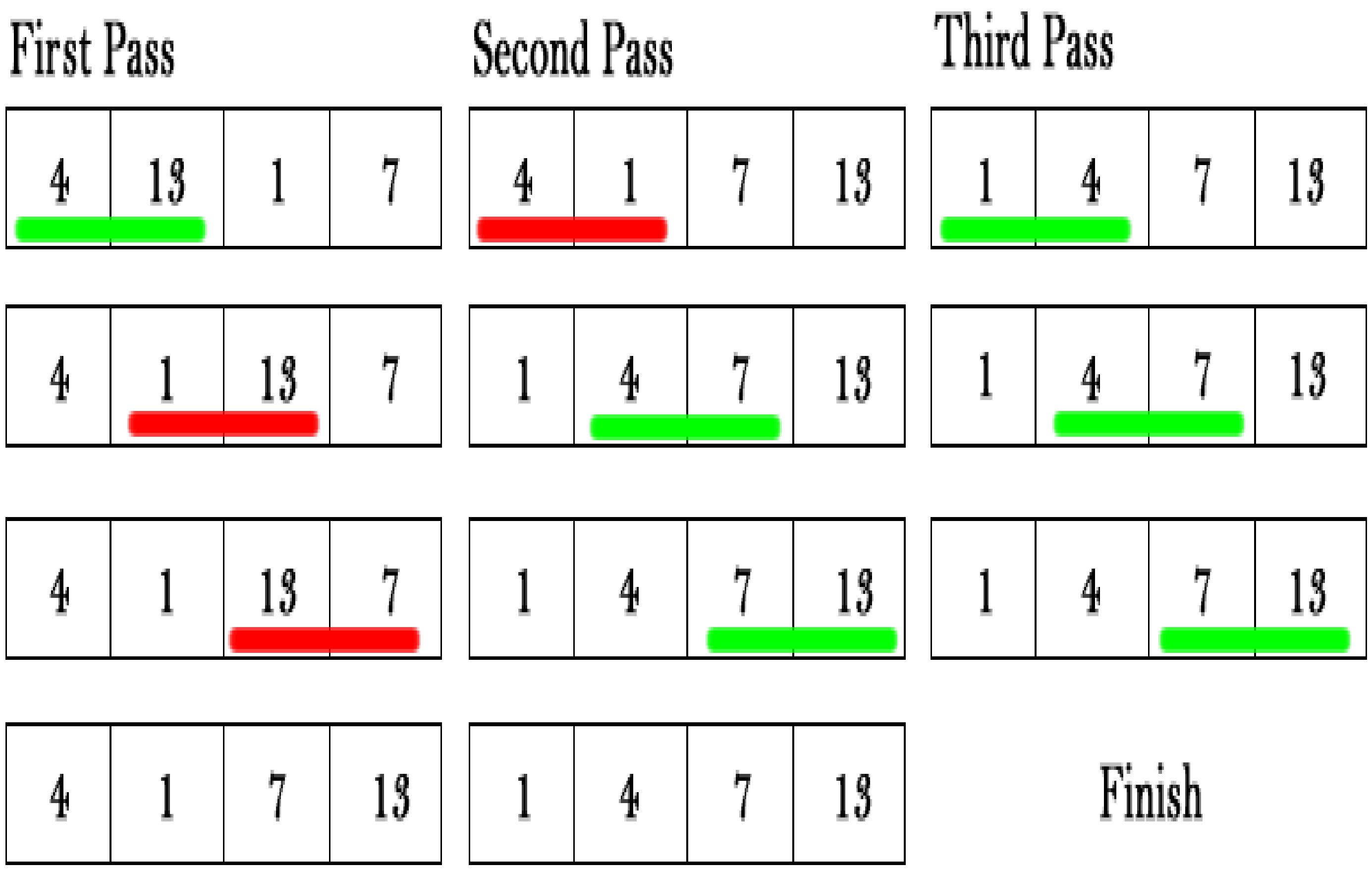
Linear Search	Binary Search
Access of elements sequentially.	Access of elements randomly.
Elements may or may not be in sorted order.	Elements must be in sorted order i.e. ascending or descending order
Takes more time to search an element.	Takes less time to search an element.
easy	tricky
Efficient for small array.	Efficient for larger array

e. Sorting: To arrange the elements in ascending or descending order. There are many sorting techniques. Here we shall discuss two sorting techniques:

(i) Bubble sort

(ii) Insertion sort

(i) BUBBLE SORT: Bubble sort is a simple sorting algorithm. It is based on comparisons, in which each element is compared to its adjacent element and the elements are swapped if they are not in proper order.



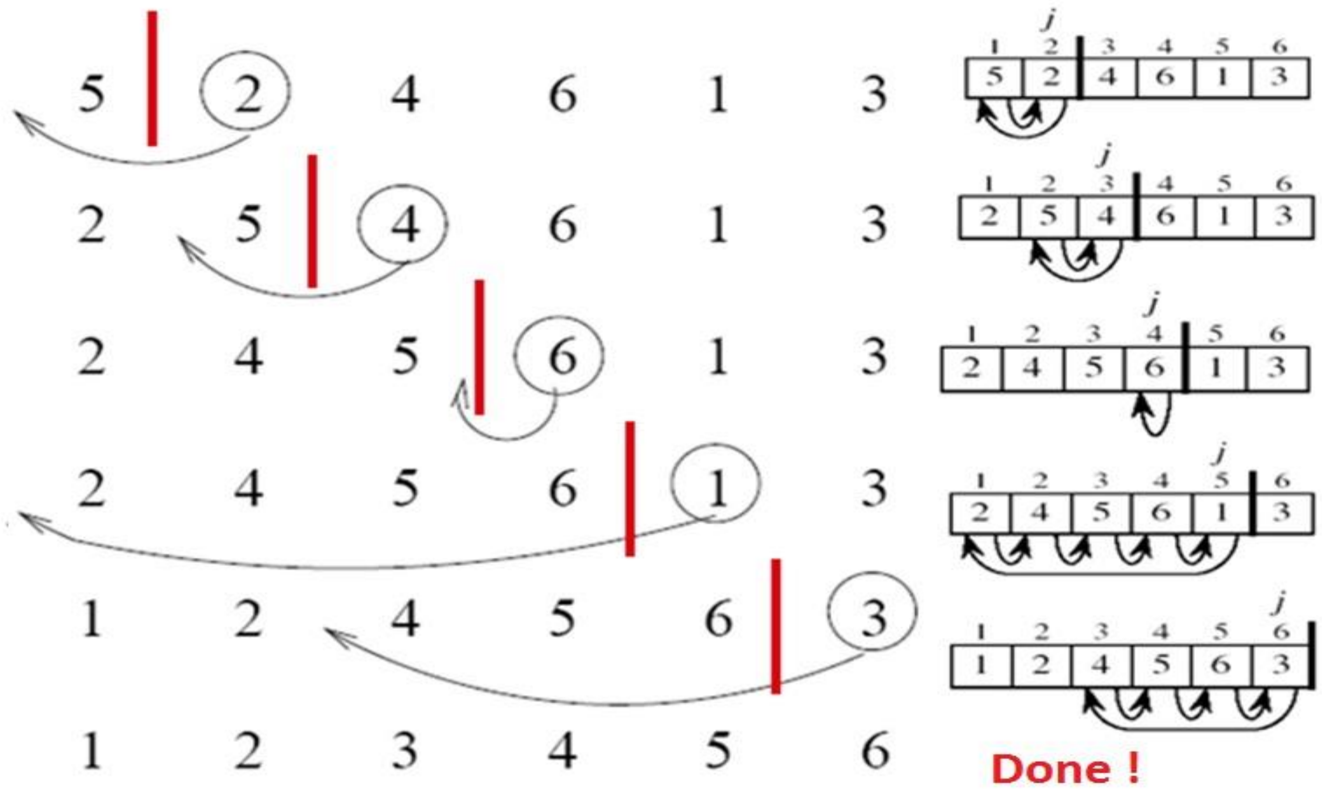
PROGRAM:

```
L=eval(input("Enter the elements:"))
n=len(L)
for p in range(0,n-1):
    for i in range(0,n-1):
        if L[i]>L[i+1]:
            L[i], L[i+1] = L[i+1],L[i]
print("The sorted list is : ", L)
```

OUTPUT:

```
Enter the elements:[60, 24, 8, 90, 45, 87, 12, 77]
The sorted list is :  [8, 12, 24, 45, 60, 77, 87, 90]
```

(ii) INSERTION SORT: Sorts the elements by shifting them one by one and inserting the element at right position.



PROGRAM:

```
L=eval(input("Enter the elements: "))

n=len(L)

for j in range(1,n):

    temp=L[j]

    prev=j-1

    while prev>=0 and L[prev]>temp:        # comparison the elements

        L[prev+1]=L[prev]              # shift the element forward

        prev=prev-1

    L[prev+1]=temp                     #inserting the element at proper position

print("The sorted list is :",L)
```

OUTPUT:

```
Enter the elements: [45, 11, 78, 2, 56, 34, 90, 19]

The sorted list is : [2, 11, 19, 34, 45, 56, 78, 90]
```

7.4.2. Multi-Dimensional array (Nested Lists): A list can also hold another list as its element. It is known as multi-dimensional or nested list.

A list in another list considered as an element.

Example:

```
>>>NL=[10, 20, [30,40], [50,60,70], 80]

>>> len(NL)

5

>>>secondlist=[1,2,3,[4,5,[6,7,8],9],10,11]

>>> len(secondlist)

6
```

Accessing the elements from nested list:

Example-1:

```
>>> L=[1, 2, 3, [4, 5, [ 6, 7, 8 ], 9 ] ,10, 11]

>>> L[1]

2

>>> L[3]

[4, 5, [6, 7, 8], 9]

>>> L[3][1]

5

>>> L[3][2][0]

6

>>> L[3][2]

[6, 7, 8]

>>> L[3][2][1]
```

```
    7
    >>> L[3][3]
    9
```

Example-2:

```
>>> L=["Python", "is", "a", ["modern", "programming"], "language", "that", "we", "use"]
>>> L[0][0]
'P'
>>> L[3][0][2]
'd'
>>> L[3:4][0]
['modern', 'programming']
>>> L[3:4][0][1]
'programming'
>>> L[3:4][0][1][3]
'g'
>>> L[0:9][0]
'Python'
>>> L[0:9][0][3]
'h'
>>> L[3:4][1]
IndexError: list index out of range
```

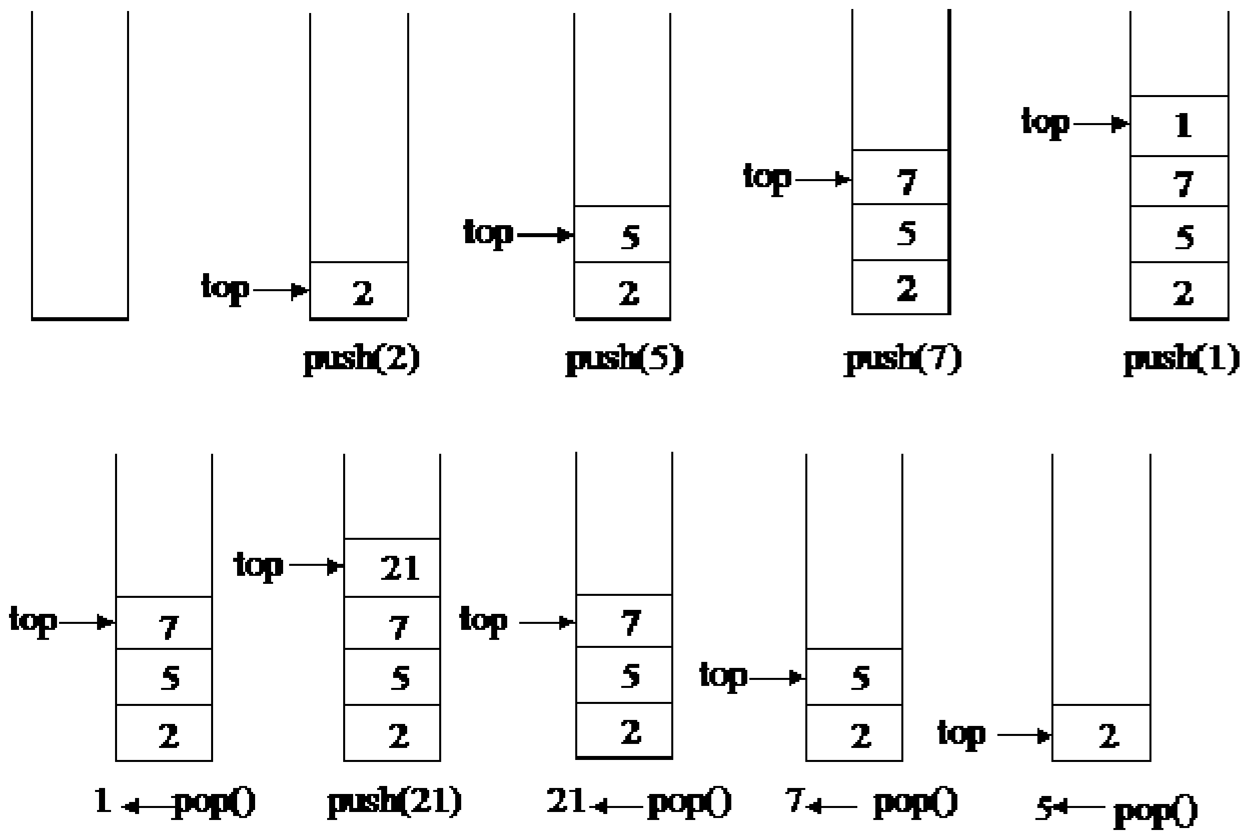
CHAPTER-7

DATA STRUCTURE-II (STACK AND QUEUE)

8.1 STACK IN PYTHON:

8.1.1 INTRODUCTION:

- Stack is a linear data structure.
- Stack is a list of elements in which an element may be inserted or deleted only at one end, called the **TOP** of the stack.
- It follows the principle **Last In First Out** (LIFO).
- There are two basic operations associated with stack:
    - **Push** : Insert the element in stack
    - **Pop** : Delete the element from stack



Example: Stack Push and Pop operations

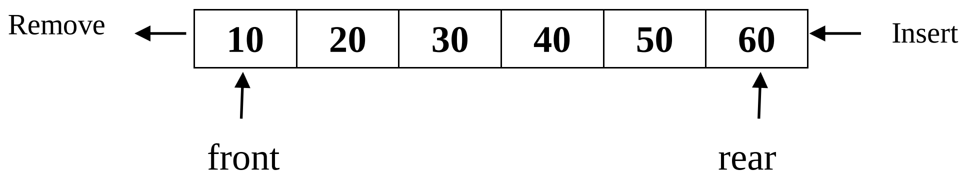
8.1.2 Program for Stack:

```
L=[ ]           #empty list
def push(item):                      #Insert an element
    L.append(item)

def pop( ):                           # Delete an element
     return L.pop( )

def peek( ):                          #Check the value of top
   return L[len(L)-1]

def size( ):                    # Size of the stack i.e. total no. of elements in stack
    return len(L)
```

8.2 QUEUE IN PYTHON:

8.2.1 INTRODUCTION:

- Queue is a linear data structure.
- Queue is a list of elements in which insertion takes place at one end, called the **REAR** and deletion takes place only at the other end, called the **FRONT**.
- It follows the principle **First In First Out** (FIFO).
- There are two basic operations associated with stack:
    - **Enqueue** : Insert the element in queue
    - **Dequeue** : Delete the element from queue



8.2.2 Program for Enqueue and Dequeue Operations:

```
L=[ ]

def Enqueue(laptop):                #Insert an element

    L.append(laptop)

    if len(L)==1:

        front=rear=L[0]
    else:
        rear=L[len(L)-1]
        front=L[0]
    print("Front is: ", front)
    print("Rear is : ", rear)

def Dequeue( ):                                 # Delete an element
    if len(L)==0:
        print("List is Empty")
    elif len(L)==1:
        print("Deleted Element is :", L.pop( ))
        print("Now list is empty")
    else:
        d=L.pop(0)
        rear=L[len(L)-1]
        front=L[0]
        print("Front is: ", front)
        print("Rear is : ", rear)
        print("Deleted Element is :", d)
```