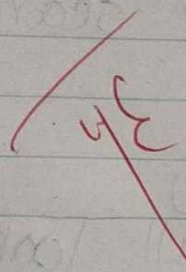
Practical 1

Aim: To search a number from the list using linear method.

Theory: The process of identifying or finding a particular record is called searching
There are two types of search
Linear search
Binary search
The linear search is further classi-fied as
SORTED
UNSORTED
Here we will look on the UNSORTED LINEAR SEARCH.
Linear search also known as sequential search is a process that check every element in the list sequentially unit the desired element is found. When the element to be searched are not specifically arranged in random manner That is what it calls unsorted linear search
The data is entered in random manner User needs to specify the element to be searched in the entered list.

Check the cond^n that whether the entered number matches if it matches then displ-y the location plus increment 1 as data is stored from location zero.
If all elements are checked on by one and elements not found then prompt message number not found.



```
found=False
a=[1,64,3,4,8,65,84,54,10,4]
search=int(input("enter the number search"))
fori in range(len(a)):
if (search==a[i]):
print("number is found at place",i+1)
found=True
break
if (found==False):
print("number dose not exist")
print("shubhang 1776")
```

```
enter the number search4
number is found at place 4
shubhang 1776
>>>
```

Practical 2

Aim: To search a number from the list using

Theory: SEARCHING and SORTING are different modes on types of data-structure

SORTING:- To basically SORT the inputed data in ascending or descending manner.

SEARCHING:- To search element and to display the same.

In searching that too in LINEAR SORTED search the data is arranged in ascending to descending or vice versa. That is all what it meant by searching through 'sorted' that is well arranged data.

14/5

SORTED Linear search

The user is supposed to enter data in sorted manner.

User has to given an element for through sorted list.

If element is found display with an updation as value is storded from location '0' If data or element not found print the same.

In sorted order list of element we can check the condition that whether the entered number lies from starting point till the last element if not then without any processing we can say number not in the list.

```
found=False
a=[11,12,13,14,15,26,27,67,68,80]
print("shubhang 1776")
search=int(input("enter the number search"))
if(search<a[0] or search>a[len(a)-1]):
print("number dose not exist")
else:
fori in range(len(a)):
if(search==a[i]):
print("number found at",i)
found=True
break
if(found==False):
print("number dose not exist")
```

```
shubhang 1776
enter the number search13
number found at 2
>>>
```

Practical 3

Aim: To search a number from the given sorted list using binary search.

Theory: A binary search also known as a half interval search, is an algorithm used in computer science to locate a specified value(key) within an array must be sorted in either ascending or descending order.

At each step of the algorithm a comparision is made and the procedure branches into one of two direction.

Specifically, the key value is compared to the array.

If the key value is less than or greater than this middle element the algorithm known which half of the array to continue sear in because the array is sorted.

This process is repeated on progressively smaller segment of the array until the vale is located.

Binary search will complete successfully in logarithmic time.

```
a=[57,58,59,72,75,79,83,85]
print("shubhang 1776")
search=int(input("enter the number search"))
l=0
r=len(a)-1
while True:
  m=(l+r)//2
if(l>r):
print("number not found")
break
if(search==a[m]):
print("number is found",m,"index number")
break
else:
if(search<a[m]):
        r=m-1
else:
        l=m+1
```

```
shubhang 1776
enter the number search58
number is found 1 index number
>>>
```

Aim: To

Theory:

Practical - 4

Aim:- To demostrate the use of stack.

Theory:

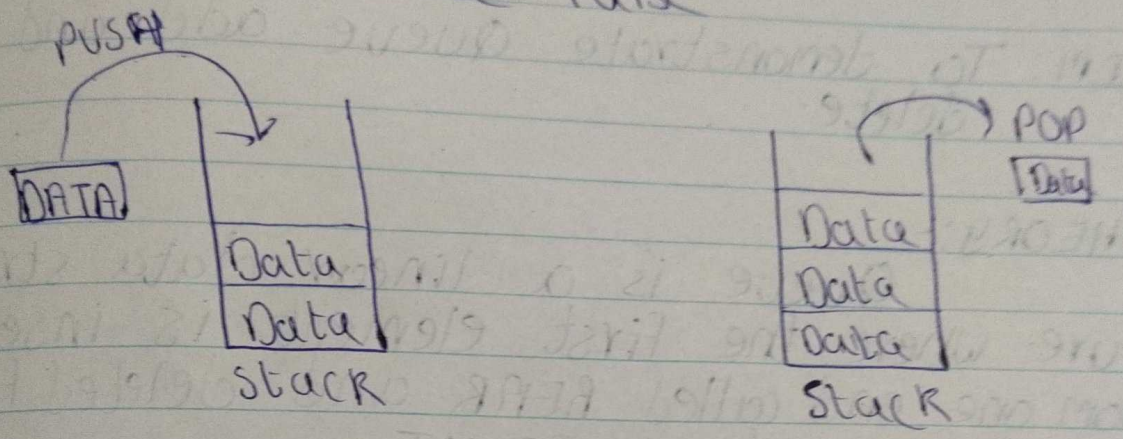
In computer science, a stack is a abstract data type that serves as a collection of elements with two principal oprn. push, which adds an element to the collection and pop, which removes the most recently added element that was not yet removed. The order may be LIFO (Last In first out) or FILO (First in Last out) Three Basic operation are performed in the stack

- PUSH :- Adds an item in the stack. If the stack is full then it is said to be over flow condition

- POP :- Removes an item from the stack The items are popped in the reverse -d order which they are pushed IF the stack is empty it is said to be under flow condn.

- Peek or Top :- Return top elemen of stack.

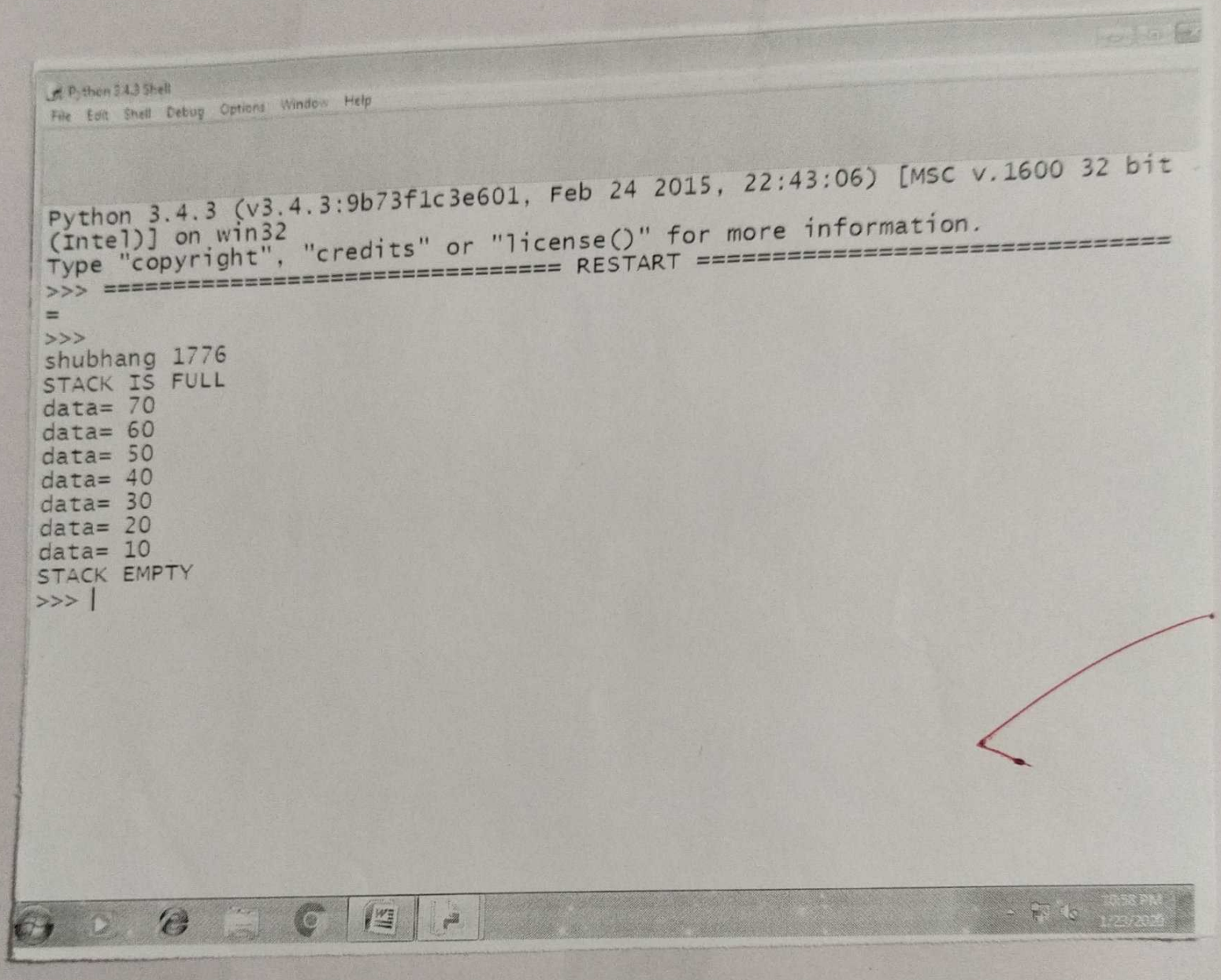
· is empty: Returns true if stack is empty else false



LIFO

PROGRAM:

```
##stack##
print("shubhang 1776")
class stack:
  global tos
  def __init__(self):
    self.l=[0,0,0,0,0,0,0,0]
    self.tos=-1
  def push(self,data):
    n=len(self.l)
    if self.tos==n-1:
      print("STACK IS FULL")
    else:
      self.tos=self.tos+1
      self.l[self.tos]=data
  def pop(self):
    if self.tos<0:
      print("STACK EMPTY")
    else:
      k=self.l[self.tos]
      print("data=",k)
      self.tos=self.tos-1
s=stack()
s.push(10)
s.push(20)
s.push(30)
s.push(40)
s.push(50)
s.push(60)
s.push(70)
s.push(80)
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
s.pop()
```


Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
=
>>>
shubhang 1776
STACK IS FULL
data= 70
data= 60
data= 50
data= 40
data= 30
data= 20
data= 10
STACK EMPTY
>>>

Practical - 5

AIM: To demonstrate Queue add and delete

THEORY:-

Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end as FRONT.

Front points to the beginning of the queue and rear points to the end of the queue.

Queue follows the FIFO (First_in_First_out) structure.

According to its FIFO structure element inserted first will also be removed first.

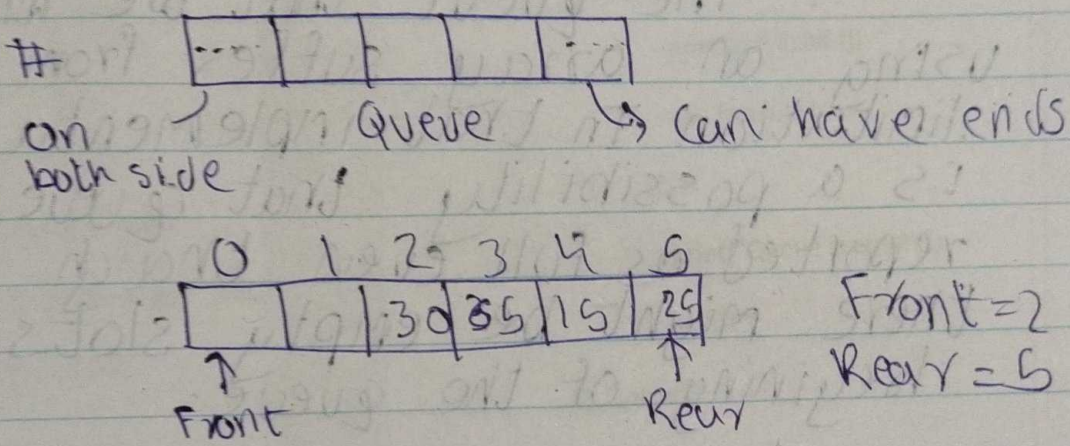
In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue) because queue is open at both of its ends.

Enqueue() can be termed as add() in queue i.e add a element in queue.

Dequeue() can be termed as delete or Remove i.e deleting or removing of element.

Front is used to get the Front data item from a queue.

Rear is used to get the last item from a queue



PROGRAM:

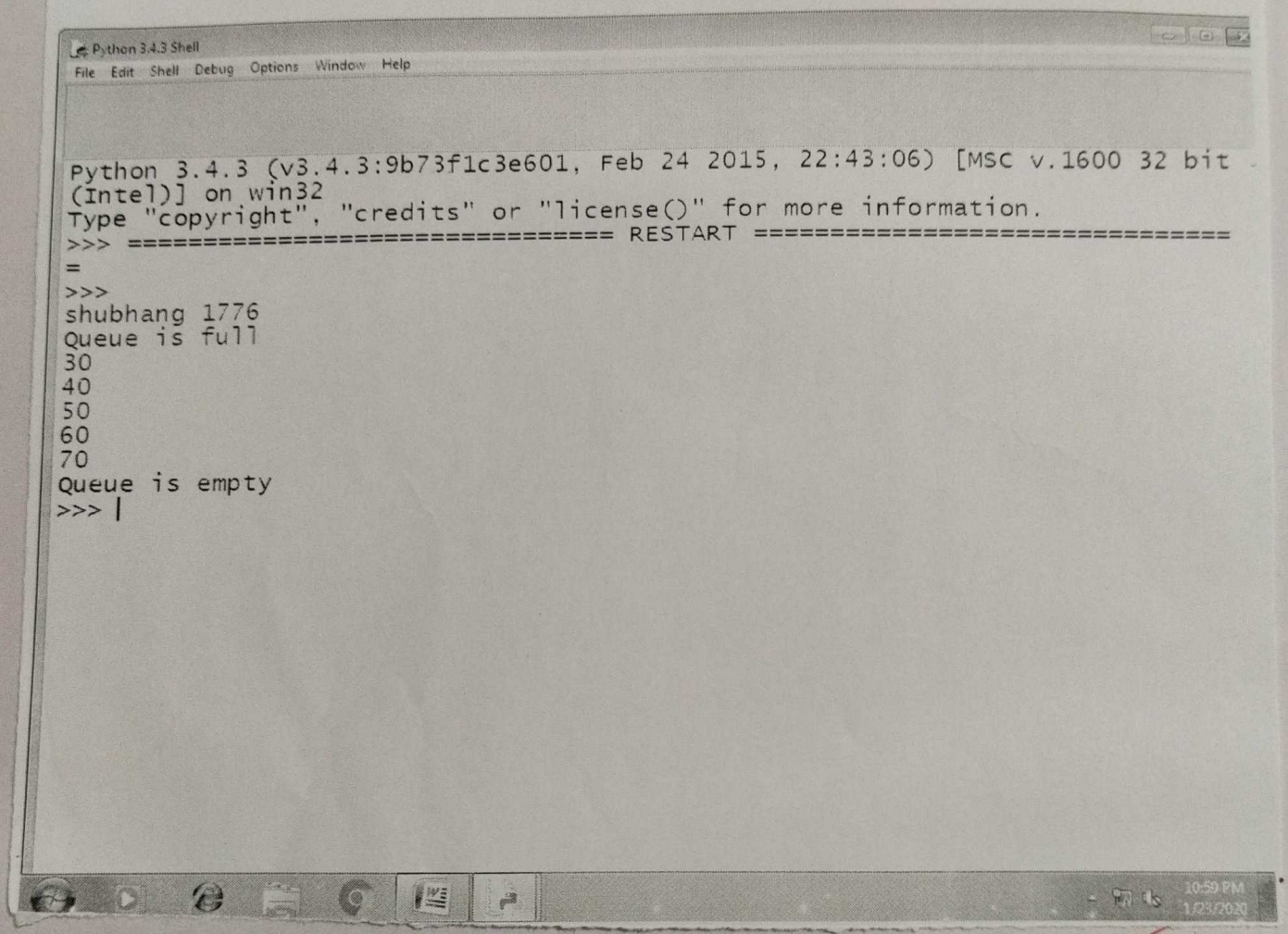
```
## Queue add and Delete ##
class Queue:
    print("shubhang 1776")
    global r
    global f
    def __init__(self):
        self.r=0
        self.f=0
        self.l=[0,0,0,0,0,0]
    def add(self,data):
        n=len(self.l)
        if self.r<n-1:
            self.l[self.r]=data
            self.r=self.r+1
        else:
            print("Queue is full")
    def remove(self):
        n=len(self.l)
        if self.f<n-1:
            print(self.l[self.f])
            self.f=self.f+1
        else:
            print("Queue is empty")
Q=Queue()
Q.add(30)
Q.add(40)
Q.add(50)
Q.add(60)
Q.add(70)
Q.add(80)

Q.remove()
Q.remove()
Q.remove()
Q.remove()
```

Q.remove()

Q.remove()

OUTPUT:



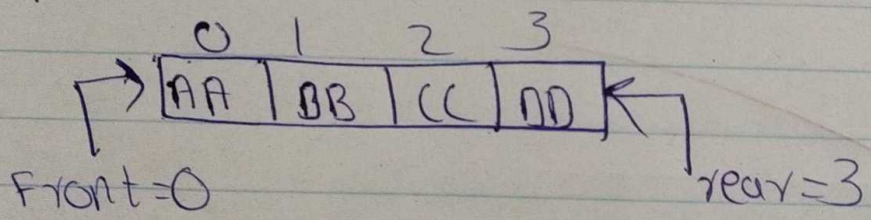
Practical-6

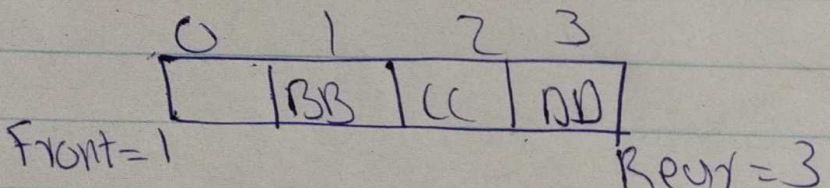
AIM: To demonstrate the use of circular queue in data-structure

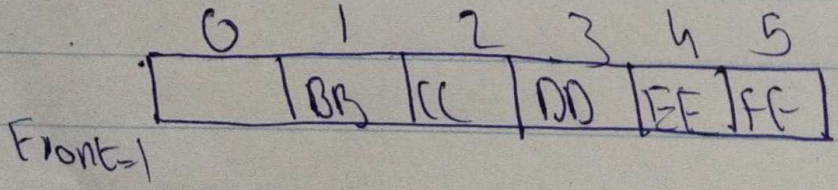
Theory:-

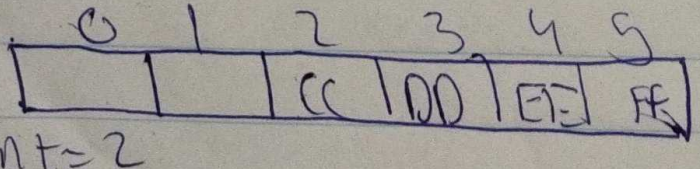
The queue that we implement using an array suffer from one limitation. In that implementation there is a possibility that queue is reported as full even though in actual there might be empty slots at the begining of the queue.

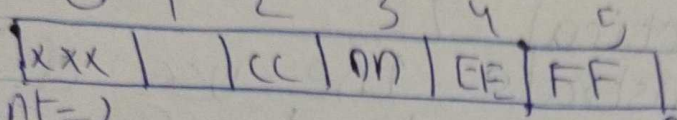
Example



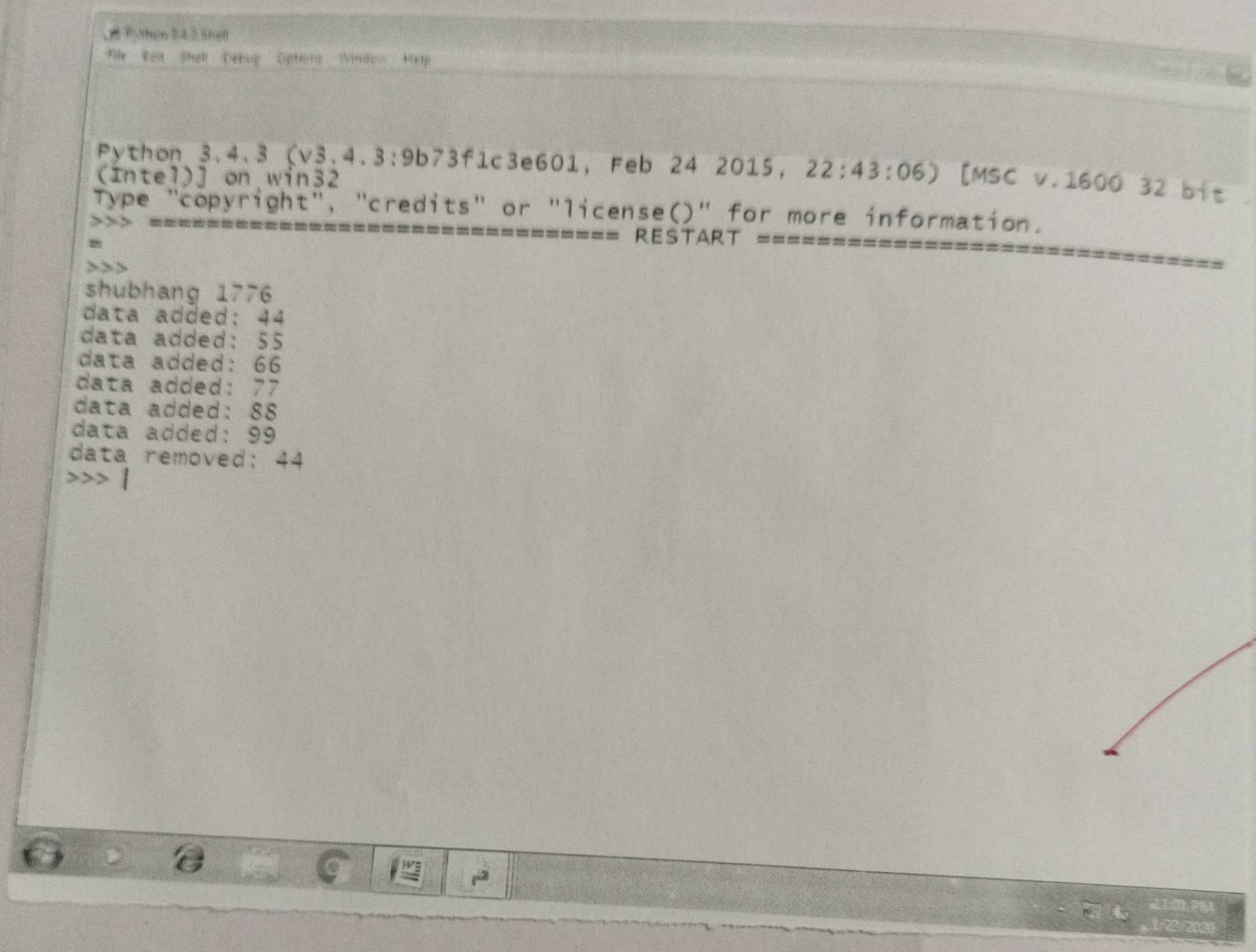







0 1 2 3 4 5
xxx
cc
nn
EE
FF
Front=2
Rear=0

```
class Queue:
  global r
  global f
  def __init__(self):
    self.r=0
    self.f=0
    self.l=[0,0,0,0,0,0]
  def add(self,data):
    n=len(self.l)
    if self.r<=n-1:
      self.l[self.r]=data
      print("data added:",dat
      self.r=self.r+1
    else:
      s=self.r
      self.r=0
      if self.r<self.f:
        self.l[self.r]=data
        self.r=self.r+1
      else:
        self.r=s
        print("Queue is full")
  def remove(self):
    n=len(self.l)
    if self.f<=n-1:
      print("data removed:",self.l[self.f])
      self.f=self.f+1
    else:
      s=self.f
      self.f=0
      if self.f<self.r:
        print(self.l[self.f])
        self.f=self.f+1
      else:
        print("Queue is empty")
        self.f=s
Q=Queue()
Q.add(44)
Q.add(55)
Q.add(66)
Q.add(77)
Q.add(88)
Q.add(99)
Q.remove()
Q.add(66)
```


Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
=
>>>
shubhang 1776
data added: 44
data added: 55
data added: 66
data added: 77
data added: 88
data added: 99
data removed: 44
>>>

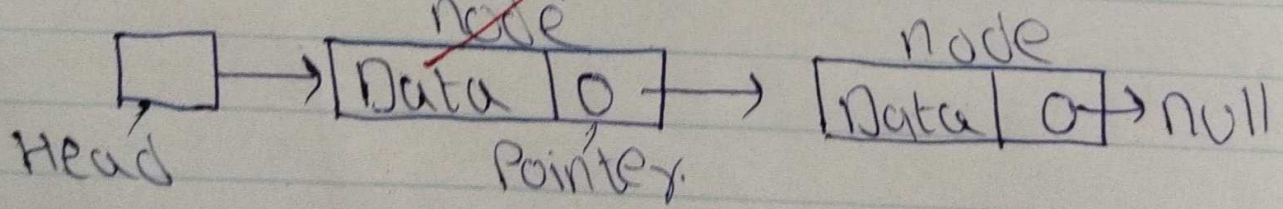
Practical - 7

AIM:- To demonstrate the use of linked list in data structure.

THEORY: A linked list is a sequence of data structures linked list is a sequence of links which contains items each link contains a connection to another link.

- LINK:- Each link of a linked list can store a data called an element
- NEXT:- Each link of a linked list contains a link to the next link called NEXT
- LINKED LIST - A linked list contains the connection link to the first link called first

LINKED LIST Representation:



TYPES of LINKED LIST:

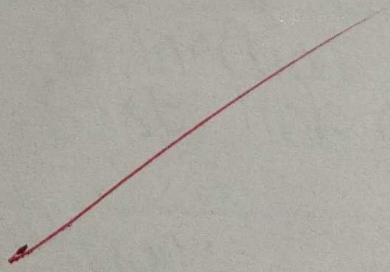
- Simple
- Doubly
- Circular

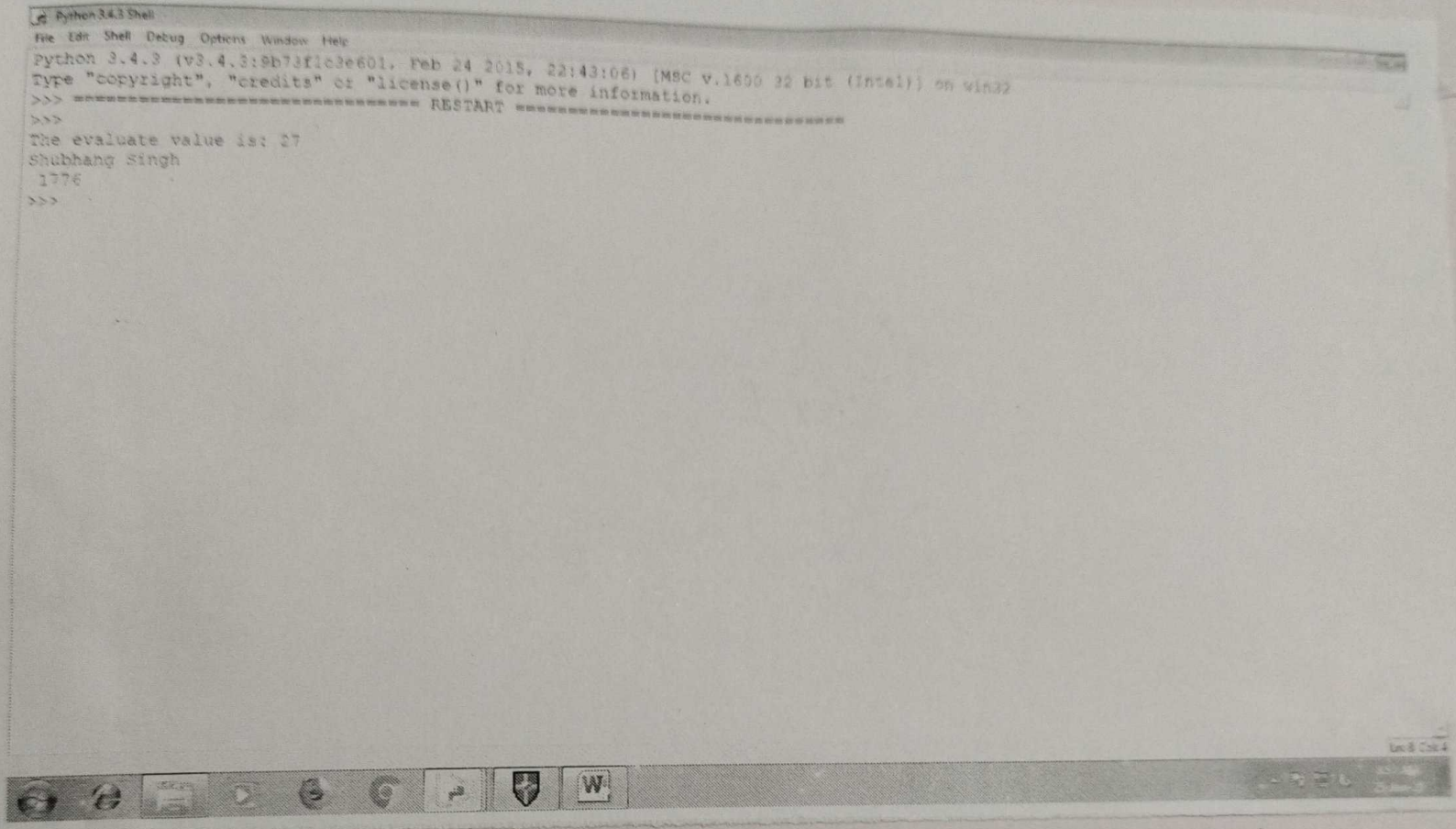
Basic operation

- Insertion
- Deletion
- Display
- Search
- Delete

```
##postfix evaluation##
def evaluate(s):
    k=s.split()
    n=len(k)
stack=[]
for i in range(n):
if k[i].isdigit():
stack.append(int(k[i]))
elif k[i]=='+':
        a=stack.pop()
        b=stack.pop()
stack.append(int(b)+int(a))
elif k[i]=='-':
        a=stack.pop()
        b=stack.pop()
stack.append(int(b)-int(a))
elif k[i]=='*':
        a=stack.pop()
        b=stack.pop()
stack.append(int(b)*int(a))
elif k[i]=='/':
        a=stack.pop()
        b=stack.pop()
stack.a
ppend(int(b)/int(a))
```

```
returnstack.pop()
s="3 4 5 + *"
r=evaluate(s)
print("The evaluate value is:",r)
print("Shubhang Singh \n 1776")
```




Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
The evaluate value is: 27
Shubhang Singh
1776
>>>

PRACTICAL-8

Aim: To evaluate postfix expression using stack

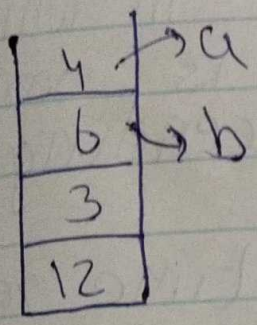
THEORY: Stack is an (ADT) and works LIFO i.e PUSH & POP operation is placed after the operand

Steps to be followed:

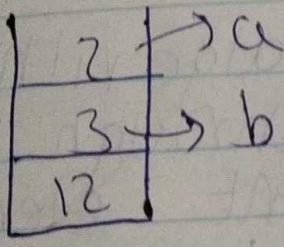
1. Read all the symbols one by one from left to right in the given postfix expression.
2. If the reading symbol is operand then push it on to the stack.
3. If the reading symbol is operator (+, -, *, /, etc.) then perform Two pop operation and store the two popped operands in two different variable (operand 1 & operand 2). The perform reading symbol operation using operand 1 & operand 2 and push result back on to the stack
4. Finally ! Perform a pop operation and display the popped value as final result.

value of postfix expression:
S= 12 3 6 4 - + *

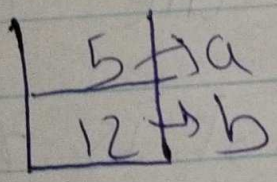
stack



b-a = 6-4 = 2 // store again in stack.



b+a = 3+2 = 5 // store result in stack



b*a = 12 * 5 = 60

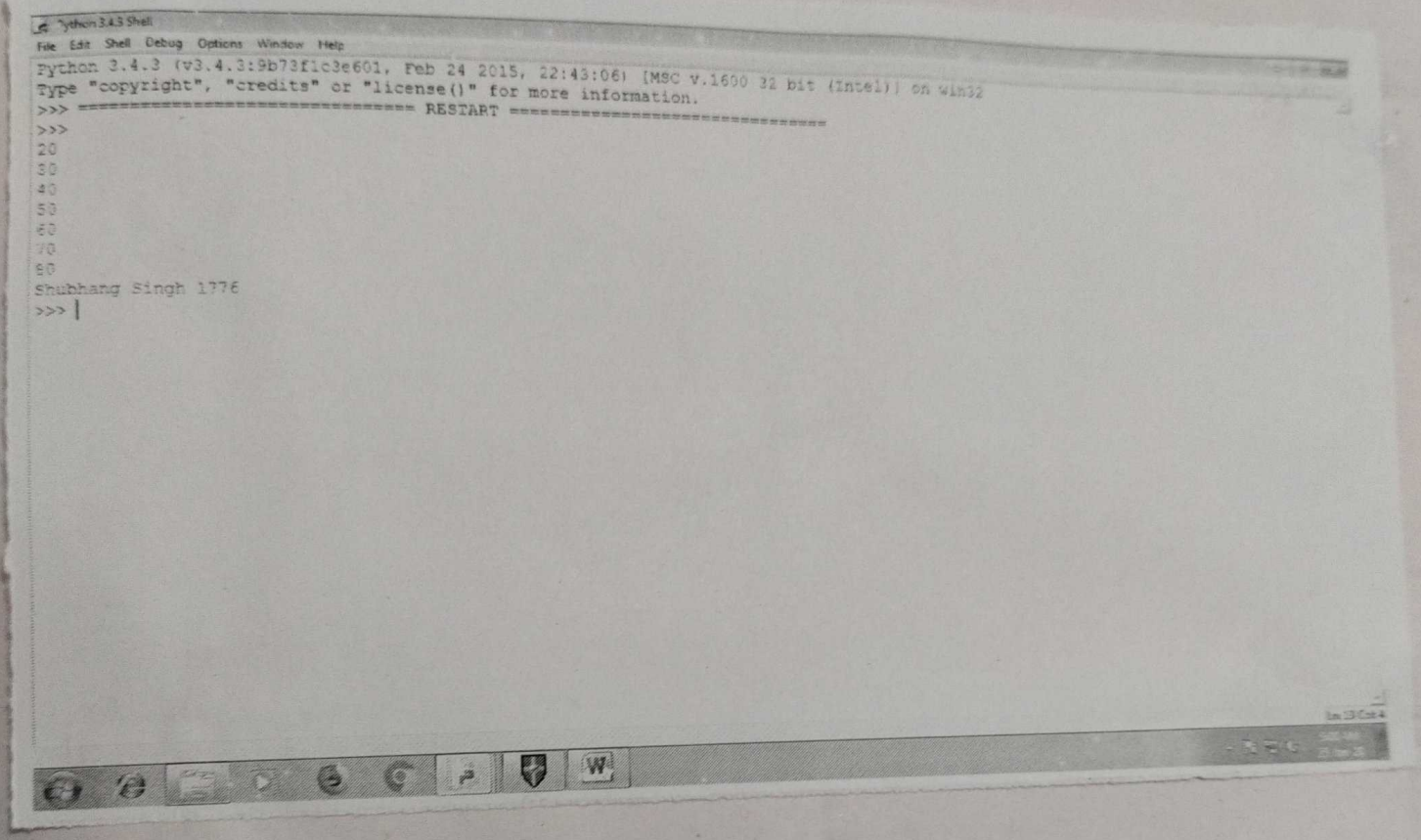
HC

```
        self.s=None
        defaddL(self,item):
        newnode=node(item)
        if  self.s==None:
        self.s=newnode
        else:
        head=self.s
        whilehead.next!=None:
        head=head.next
        head.next=newnode
        defaddB(self,item):
        newnode=node(item)
        ifself.s==None:
        self.s=newnode
        else:
        newnode.next=self.s
        self.s=newnode

        def display(self):
        head=self.s
        whilehead.next!=None:
        print(head.data)
        head=head.next
        print(head.data)
        start=linkedlist()
```

```
###After Before linkedlist(simple)###
class node:
global data
global next
def __init__(self,item):
self.data=item
self.next=None
classlinkedlist:
global s
def __init__(self):
```

```
        start.addL(50)
        start.addL(60)
        start.addL(70)
        start.addL(80)
        start.addB(40)
        start.addB(30)
        start.addB(20)
        start.display()
        print("Shubhang Singh 1776")
```


Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
20
30
40
50
60
70
80
Shubhang Singh 1776
>>>

Practical - 9

Aim:- To sort given random data by using bubble sort

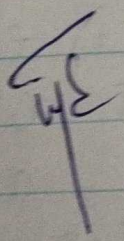
Theory:- SORTING is type in which any random data is sorted i.e arranged in ascending or descending order.
Bubble sort sometime refferred as sinking sort
It is a simple sorting algorithm that repeatedy steps through the lists, compares adjacent element and swaps them if the are in wrong order.
The pass through the list is repeated until the list is sorted.
The Algorithm which is a comparison sort is named for the way smaller or larger elements "bubble" to the top of the list.

Example:
First pass
(5 1 4 2 8) → (1 5 4 2 8) Here algo compare the first two element and swaps since 5 > 1

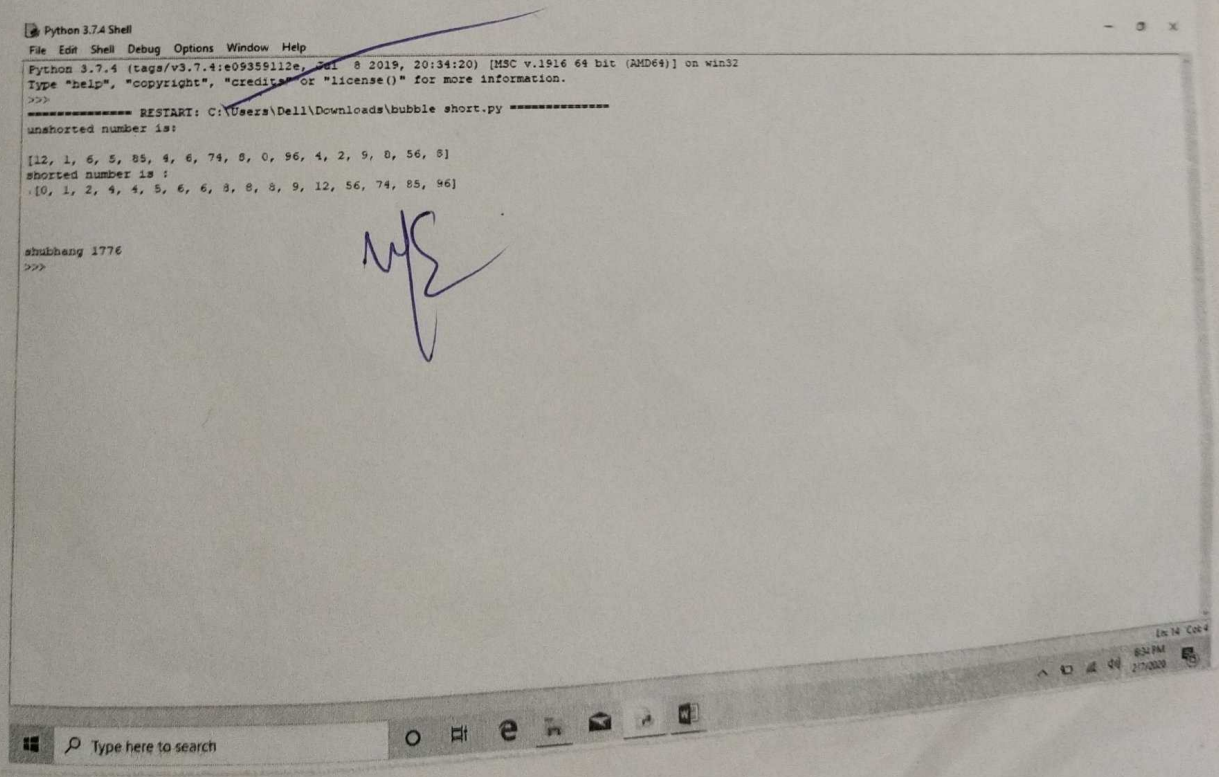
second pass:

(1 5 4 2 8) → (1 4 5 2 8) swap since (5>4)
(1 4 5 2 8) → (1 4 2 5 8) swap since (5>2)
(1 2 4 5 8) → 1 2 4 5 2

Third pass
(1 2 4 5 8) It checks and gives the data in sorted order.



```
a=[12,1,6,5,85,4,6,74,8,0,96,4,2,9,8,56,8]
print("unsorted number is:\n")
print(a)
fori in range (len(a)-1):
for j in range (len(a)-1-i):
    if (a[j]>a[j+1]):
            t=a[j]
    a[j]=a[j+1]
    a[j+1]=t
    print("sorted number is :\n",a)
    print("\n\n\nshubhang 1776")
```



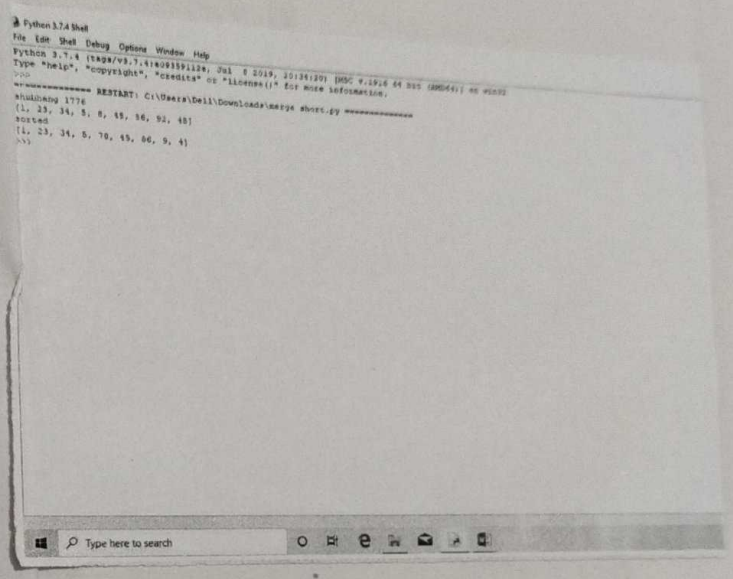
Practical - 10

Aim :- To sort using a) Merge sort

Theory :-

Merge sort is a divide and conquer algorith. based on the idea of breaking down list into several sub-lists until each sublists consist of a single element and merging those sublist in a manner that results into a sorted list. While compairing two sublists is taken into consideration. While sorting is ascending order, the element that is of lesser value becomes a new element of the sorted list. The procedure is repeated until both the smaller sublists are empty and new combine sublist comprise of all elements of both the sublist.

```
print("shubhang 1776")
def sort(arr,l,m,r):
    n1=m-l+1
    n2=r-m
    L=[0]*(n1)
    R=[0]*(n2)
for i in range(0,n1):
        L[i]=arr[l+i]
for j in range(0,n2):
        R[j]=arr[m+1+j]
i=0
    j=0
    k=l
whilei<n1 and j<n2:
if L[i]<=R[j]:
arr[k]=L[i]
i+=l
else:
arr[k]=R[j]
        j+=1
        k+=1
whilei<n1:
arr[k]=L[i]
i+=1
        k+=1
while j<n2:
arr[k]=R[j]
        j+=1
        k+=1
defmergesort(arr,l,r):
if l<r:
    m=int((l+(r-1))/2)
mergesort(arr,l,m)
mergesort(arr,m+1,r)
sort(arr,l,m,r)
arr=[1,23,34,5,8,45,86,92,48]
print(arr)
```



Practical - 11

Aim:- To sort using selection sort

Theory :-

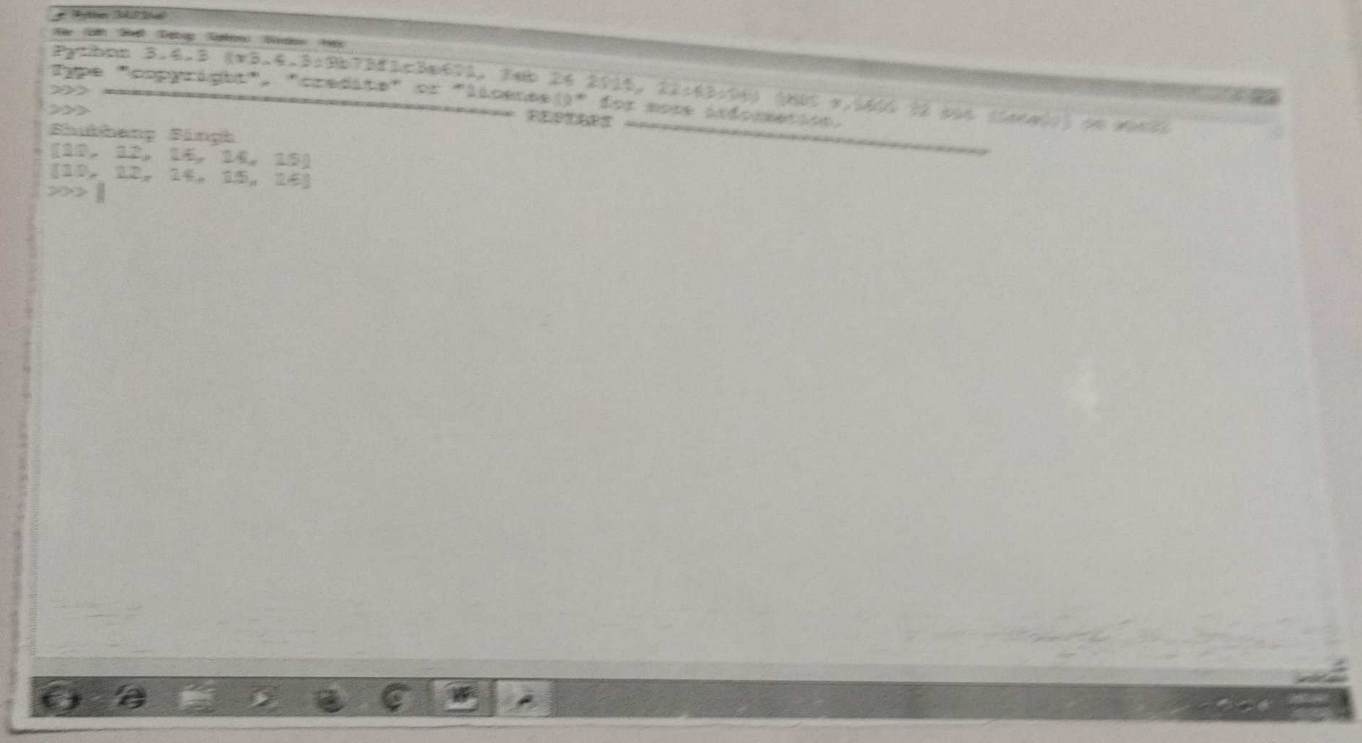
The selectio sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

The algorithm maituins two subarray in a given array.

i) The subarray which is already sorted.
ii) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked & moved to the sorted subarray

```
##Selection Sort##
print ("Shubhang Singh")
a=[10,12,16,14,15]
print(a)
for i in range (len(a)-1):
for j in range (len(a)-1):
if(a[j]>a[i+1]):
        t=a[j]
a[j]=a[i+1]
a[i+1]=t
print (a)
```


Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Shubhamp Singh
[10, 12, 14, 14, 15]
[10, 12, 14, 15, 16]
>>>

Practical - 12

Aim :- Binary Tree

Theory :-

Binary tree is a tree which supports max^m of 2 children for any node within the tree thuss any particular node can have either 0 or 1 or 2 children.

Leaf Node :- Nodes which do not have any children.

Internal Node :- Nodes which are non-leaf nodes

Traversing can be defined as a process of visiting every node of the tree exactly once

Preorder :- i) Visit the root node.
ii) Traverse the left subtrees. The left subtree in turn might have left and right subtrees
iii) Traverse the right subtree. The right subtree inturn might have left and right subtree

Inorder :- i) Traverse the left subtree. The left subtree inturn might have left and right subtree
ii) Visit the root node
iii) Traverse the right subtree right

subtree inturn might have left and right subtrees.

Postorder:- i) Traverse the left subtree.
The left subtrees inturn might have left & right subtrees
ii) Traverse the right subtree
The right subtree might have left and right subtrees
iii) Visit the root node.

```
class Node:
    global r
    global l
    global data
    def __init__(self,l):
        self.l=None
        self.data=l
        self.r=None
class Tree:
    global root
    def __init__(self):
        self.root=None
    def add(self,val):
        if (self.root)==None:
            self.root=Node(val)
        else:
            newnode=Node(val)
            h=self.root
            while True:
                ifnewnode.data<h.data:
                    ifh.l!=None:
                        h=h.l
                    else:
                        h.l=newnode
                        print(newnode.data,"added on left of",h.data)
                        break
                else:
                    ifh.r!=None:
                        h=h.r
                    else:
                        h.r=newnode
                        print(newnode.data,"added on rightof",h.data)
                        break
    def preorder(self,start):
        if start!=None:
            print(start.data)
            self.preorder(start.l)
            self.preorder(start.r)
    definorder(self,start):
        if start!=None:
            self.inorder(start.l)
            print(start.data)
            self.inorder(start.r)
    defpostorder(self,start):
        if start!=None:
            self.postorder(start.l)
            self.postorder(start.r)
            print(start.data)
T=Tree()
```

```
T.add(100)
T.add(80)
T.add(70)
T.add(85)
T.add(10)
T.add(78)
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
print("Shubhang Singh 1776")
```