1. **Array:**

import array as arr

a=arr.array('i',[1,2,3])

a.insert(12,15)

print(a)

b=arr.array("d",[10.24,10.54,67.34,99.43])

b.insert(4,42.67)

b.insert(5,78.42)

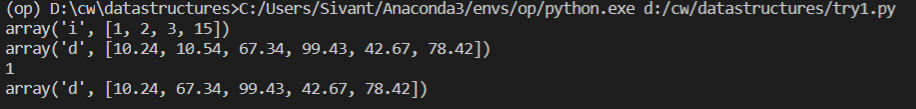
print(b)

b.remove(10.54)

print(b.index(67.34))

print(b)

**Output:**



**(2) Array of characters:**

import array as ar

a=ar.array("u",["s","i","v","a"])

print(a)

a.insert(4,"n")

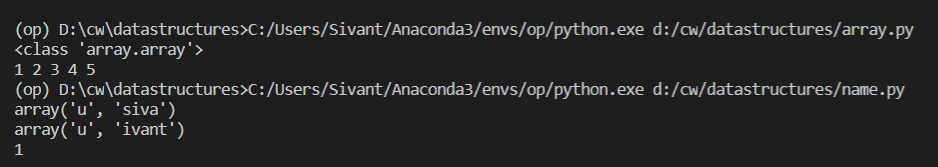
a.insert(5,"t")

a.remove("s")

print(a)

print(a.index("v"))

**Output:**



**(3) Linked list:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

class linlis:

    def \_\_init\_\_(self):

        self.first=None

        self.last=None

    def insert(self,n):

        if self.first is None:

            self.first=node(n)

            self.last=self.first

        else:

            self.last.next=node(n)

            self.last=self.last.next

    def delete(self):

        if self.last==None:

            print("cannot delete at empty linked list")

        else:

            if self.last.next==self.last:

                temp=self.first

                self.first=None

                self.last=None

            else:

                temp=self.first

                self.first=self.first.next

            print("Deleted element: ",temp.data)

            temp=None

    def display(self):

        if self.first!=None:

            temp=node(None)

            temp=self.first

            while temp != None:

                print(temp.data,end=" ")

                temp=temp.next

            print("")

        else:

            print("can't print empty list")

l=linlis()

l.insert(1)

l.insert(4)

l.insert(3)

l.insert(2)

l.insert(6)

print("Linked list: ",end=" ")

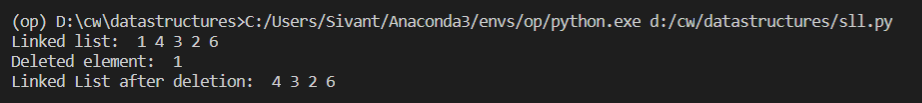
l.display()

l.delete()

print("Linked List after deletion: ",end=" ")

l.display()

**Output:**



**(4) Circular linked list:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

    def disn(self):

        return self.data

class linlis:

    def \_\_init\_\_(self):

        self.first=None

        self.last=None

    def insert(self,n):

        if self.first is None:

            self.first=node(n)

            self.last=self.first

        else:

            self.last.next=node(n)

            self.last=self.last.next

            self.last.next=self.first

    def delete(self):

        if self.last==None:

            print("cannot delete at empty linked list")

        elif self.last.next==self.last:

            self.first=None

            self.last=None

        else:

            self.first=self.first.next

            self.last.next=self.first

    def display(self):

        if self.first.next==None:

            print("--",self.first.data,"--")

        elif self.first!=None:

            temp=self.first

            print("--",end="")

            while temp.next!=None:

                print(temp.disn(),end="--")

                if temp.next==self.first:

                    break

                temp=temp.next

            print("")

        else:

            print("can't print empty list")

l=linlis()

l.insert(5)

l.insert(1)

l.insert(4)

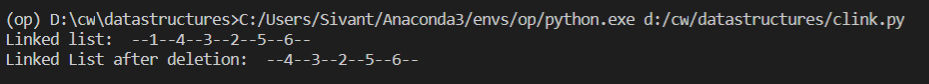
l.display()

l.delete()

l.insert(3)

l.display()

**Output:**



1. **Stack:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

    def disn(self):

        return self.data

class stack:

    def \_\_init\_\_(self):

        self.top=None

    def ins(self,val):

        if self.top==None:

            self.top=node(val)

        else:

            temp=self.top

            self.top=node(val)

            self.top.next=temp

    def display(self):

        temp=self.top

        while temp!=None:

            print(temp.data,end=" ")

            temp=temp.next

        print("")

    def delete(self):

        if self.top !=None:

            if self.top.next==None:

                self.top=None

            else:

                temp=self.top

                self.top=temp.next

                temp=None

        else:

            print("Cannot pop for empty list")

s=stack()

s.ins(1)

s.ins(2)

s.ins(3)

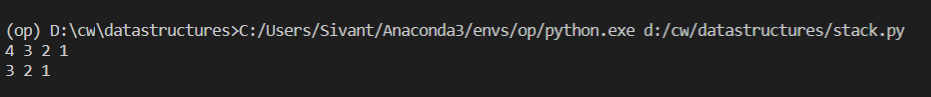
s.ins(4)

s.display()

s.delete()

s.display()

**Output:**



**(6) Queue using array:**

from collections import deque

q=deque(["sivant","starr","king","cashew"])

print(q)

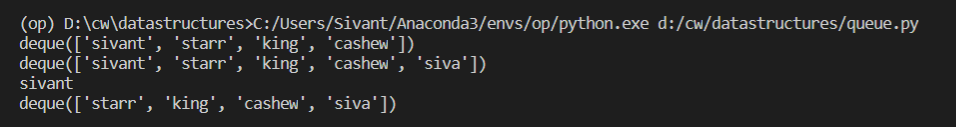
q.append("siva")

print(q)

print(q.popleft())

print(q)

**Output:**



**(7) Pallindrome:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

    def disn(self):

        return self.data

class linlis:

    def \_\_init\_\_(self):

        self.first=None

        self.last=None

    def insert(self,n):

        if self.first is None:

            self.first=node(n)

            self.last=self.first

        else:

            no=node(n)

            self.last.next=no

            self.last=self.last.next

    def delete(self):

        if self.last==None:

            print("cannot delete at empty linked list")

        elif self.last.next==self.last:

            self.first=None

            self.last=None

        else:

            self.first=self.first.next

    def display(self):

        if self.first!=None:

            temp=node(None)

            temp=self.first

            print("--",end="")

            while temp != None:

                print(temp.disn(),end="--")

                temp=temp.next

            print("")

        else:

            print("can't print empty list")

class stack:

    def \_\_init\_\_(self):

        self.top=None

    def ins(self,val):

        if self.top==None:

            self.top=node(val)

        else:

            temp=self.top

            self.top=node(val)

            self.top.next=temp

    def display(self):

        temp=self.top

        while temp!=None:

            print(temp.data,end=" ")

            temp=temp.next

def rev(l):

    s=stack()

    temp=l.first

    while temp!=None:

        s.ins(temp.data)

        temp=temp.next

    s.display()

    print("")

    return s

def pallindrome(l,s):

    t1=l.first

    t2=s.top

    while t1!=None:

        if t1.data!=t2.data:

            print("It is not a pallindrome!!!")

            return

        t1=t1.next

        t2=t2.next

    else:

        print("It is a pallindrome")

l=linlis()

l.insert(1)

l.insert(3)

l.insert(3)

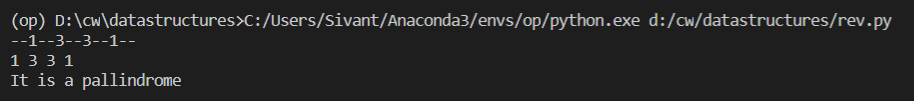
l.insert(1)

l.display()

s=rev(l)

pallindrome(l,s)

**Output:**



**(8) Insert and delete at specific pos in a linked list:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

    def disn(self):

        return self.data

class linlis:

    def \_\_init\_\_(self):

        self.first=None

        self.last=None

    def insert(self,n):

        if self.first is None:

            self.first=node(n)

            self.last=self.first

        else:

            self.last.next=node(n)

            self.last=self.last.next

    def delete(self):

        if self.last==None:

            print("cannot delete at empty linked list")

        elif self.last.next==self.last:

            self.first=None

            self.last=None

        else:

            self.first=self.first.next

    def insertpos(self,item,i):

        if i==0:

            temp=self.first

            self.first=node(item)

            self.first.next=temp

        else:

            temp=self.first

            k=0

            while temp !=None:

                if k==i-1:

                    uk=temp.next

                    temp.next=node(item,uk)

                    break

                k=k+1

                temp=temp.next

            if k<i:

                    self.last.next=node(item)

                    self.last=self.last.next

    def delpos(self,i):

        temp=self.first

        if i==0:

            self.first=temp.next

            temp=None

        else:

            k=0

            while temp!=None:

                if k==i-1:

                    t=temp

                    temp=temp.next

                    t.next=temp.next

                    break

                k=k+1

                temp=temp.next

    def display(self):

        if self.first!=None:

            temp=node(None)

            temp=self.first

            print("--",end="")

            while temp != None:

                print(temp.disn(),end="--")

                temp=temp.next

            print("")

        else:

            print("can't print empty list")

l=linlis()

l.insert(1)

l.insert(4)

l.insert(3)

l.insert(2)

l.insert(5)

l.insert(6)

l.display()

l.insertpos(13,0)

l.display()

l.delete()

l.display()

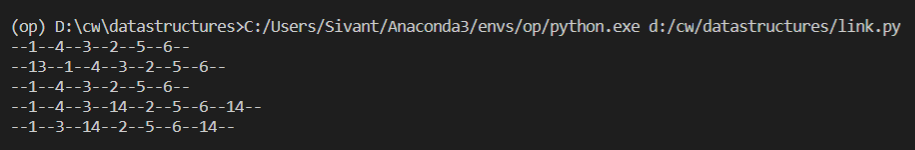
l.insertpos(14,3)

l.display()

l.delpos(1)

l.display()

**Output:**



**(9) Queue using linked list:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

    def disn(self):

        return self.data

class queue:

    def \_\_init\_\_(self):

        self.front=None

        self.last=None

    def enqueue(self,n):

        if self.front is None:

            self.front=node(n)

            self.last=self.front

        else:

            self.last.next=node(n)

            self.last=self.last.next

    def dequeue(self):

        if self.last==None:

            print("cannot delete at empty linked list")

        elif self.last.next==self.last:

            self.front=None

            self.last=None

        else:

            self.front=self.front.next

    def display(self):

        if self.front!=None:

            temp=node(None)

            temp=self.front

            while temp != None:

                print(temp.disn(),end=" ")

                temp=temp.next

            print("")

        else:

            print("can't print empty list")

l=queue()

l.enqueue(7)

l.enqueue(1)

l.enqueue(3)

l.enqueue(0)

l.enqueue(6)

l.enqueue(0)

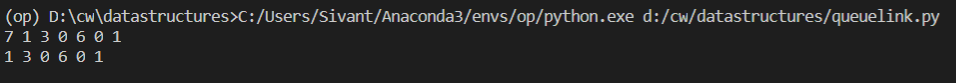
l.enqueue(1)

l.display()

l.dequeue()

l.display()

**Output:**



**(10) Max and Min in SLL:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

    def disn(self):

        return self.data

class linlis:

    def \_\_init\_\_(self):

        self.first=None

        self.last=None

    def insert(self,n):

        if self.first is None:

            self.first=node(n)

            self.last=self.first

        else:

            self.last.next=node(n)

            self.last=self.last.next

    def delete(self):

        if self.last==None:

            print("cannot delete at empty linked list")

        elif self.last.next==self.last:

            self.first=None

            self.last=None

        else:

            self.first=self.first.next

    def maxmin(self):

        if self.first!=None:

            s=0

            l=0

            temp=node(None)

            temp=self.first

            s=l=temp.data

            while temp != None:

                if(temp.data>l):

                    l=temp.data

                if(temp.data<s):

                    s=temp.data

                temp=temp.next

            print("smallest -> ",s)

            print("largest -> ",l)

        else:

            print("can't find max and min in empty list")

    def display(self):

        if self.first!=None:

            temp=node(None)

            temp=self.first

            print("--",end="")

            while temp != None:

                print(temp.disn(),end="--")

                temp=temp.next

            print("")

        else:

            print("can't print empty list")

l=linlis()

l.insert(1)

l.insert(4)

l.insert(3)

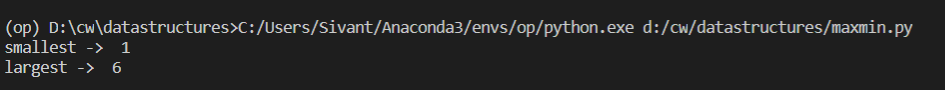
l.insert(2)

l.insert(5)

l.insert(6)

l.maxmin()

**Output:**



**(11) Middle in SLL:**

class node:

    def \_\_init\_\_(self,data=None,next=None):

        self.data=data

        self.next=next

    def disn(self):

        return self.data

class linlis:

    def \_\_init\_\_(self):

        self.first=None

        self.last=None

    def insert(self,n):

        if self.first is None:

            self.first=node(n)

            self.last=self.first

        else:

            self.last.next=node(n)

            self.last=self.last.next

    def delete(self):

        if self.last==None:

            print("cannot delete at empty linked list")

        elif self.last.next==self.last:

            self.first=None

            self.last=None

        else:

            self.first=self.first.next

    def middle(self):

        if self.first!=None:

            l=[]

            temp=node(None)

            temp=self.first

            while temp != None:

                l.append(temp.data)

                temp=temp.next

            k=len(l)

            ind=int(k/2)

            if k%2!=0:

                print("middle element -> ",l[ind])

            else:

                print("middle elements are -> ",l[ind-1]," & ",l[ind])

        else:

            print("can't find middle element in empty list")

    def display(self):

        if self.first!=None:

            temp=node(None)

            temp=self.first

            print("--",end="")

            while temp != None:

                print(temp.disn(),end="--")

                temp=temp.next

            print("")

        else:

            print("can't print empty list")

l=linlis()

l.insert(1)

l.insert(4)

l.insert(3)

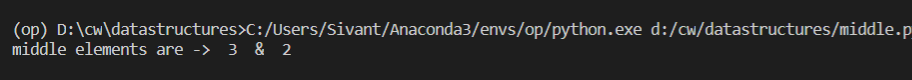
l.insert(2)

l.insert(5)

l.insert(6)

l.middle()

**Output:**



**(12) Linear Search:**

a=[-23,97,18,21,5,-86,64,0,-37]

item=int(input("Enter element to be found:"))

flag=0

for i in range(len(a)):

    if item == a[i]:

        print("Item found at index:",i)

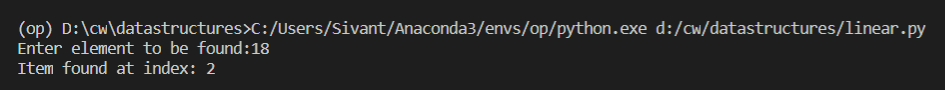
        flag=1

        break

if flag==0:

    print("Item not found")

**Output:**



**(13) Bubble Sort:**

arr=[1,3,6,5,4,9,8,2]

lenght=len(arr)

for i in range(lenght):

    for j in range(lenght-i-1):

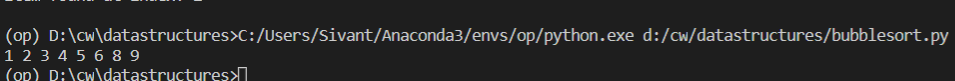
        if(arr[j]>arr[j+1]):

            arr[j+1],arr[j]=arr[j],arr[j+1]

for i in range(lenght):

    print(arr[i],end=" ")

**Output:**



**(14) Selection Sort:**

arr=[1,3,5,6,4,2,8,7]

l=len(arr)

for i in range(l):

    for j in range(i+1,l):

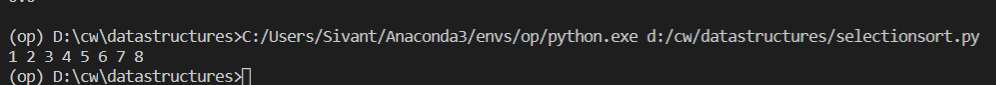
        if arr[j]<arr[i]:

            arr[j],arr[i]=arr[i],arr[j]

for i in arr:

    print(i,end=" ")

**Output:**



**(15) Merge:**

l1=[1,3,4,5,7]

l2=[2,4,6,8,10]

l3=[]

i=0

j=0

a=int(len(l1))

b=int(len(l2))

while i<a and j<b:

    if l1[i]>l2[j]:

        l3.append(l2[j])

        j+=1

    else:

        l3.append(l1[i])

        i+=1

while i<a:

    l3.append(l2[i])

    i=i+1

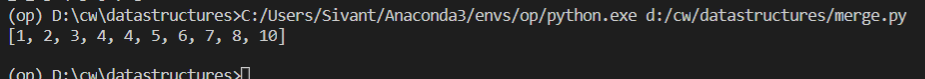
while j<b:

    l3.append(l2[j])

    j=j+1

print(l3)

**Output:**



**(17) Merge Sort:**

def mergesort(arr):

    if len(arr)>1:

        mid=len(arr)//2

        l=arr[:mid]

        r=arr[mid:]

        print(l,end=" ")

        print(r)

        mergesort(l)

        mergesort(r)

        i=j=k=0

        while i<len(l) and j<len(r):

            if l[i]<r[j]:

                arr[k]=l[i]

                i=i+1

            else:

                arr[k]=r[j]

                j=j+1

            k=k+1

        while i<len(l):

            arr[k]=l[i]

            i=i+1

            k=k+1

        while j<len(r):

            arr[k]=r[j]

            j=j+1

            k=k+1

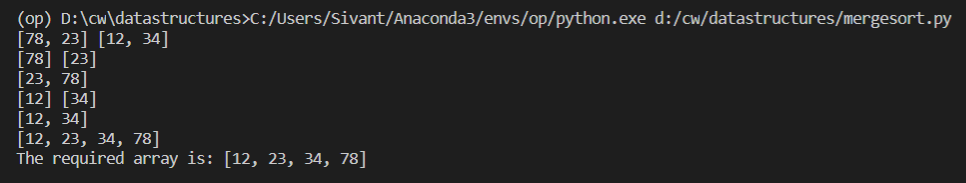
        print(arr)

arr=[78,23,12,34]

mergesort(arr)

print("The required array is:",arr)

**Output:**



**(17) Is Binary Tree a BST:**

class node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.left=None

        self.right=None

class bst:

    def \_\_init\_\_(self):

        self.curnode=None

    def insert(self,cn,val):

        if self.curnode==None:

            self.curnode=node(val)

        else:

            if cn==None:

                cn=node(val)

            else:

                if val<cn.data:

                    cn.left=self.insert(cn.left,val)

                elif val>cn.data:

                    cn.right=self.insert(cn.right,val)

            return cn

    def display(self,cn):

        if self.curnode==None:

            print("List is empty")

        else:

            if cn==None:

                return

            else:

                self.display(cn.left)

                self.display(cn.right)

                print(cn.data,end=" ")

    def isbst(self):

        f1=0

        f2=0

        cn=self.curnode

        while cn!=None:

            if cn.left==None:

                break

            else:

                if cn.data<cn.left.data:

                    f1=1

                    break

                else:

                    cn=cn.left

        while cn!=None:

            if cn.right==None:

                break

            else:

                if cn.right.data<cn.data:

                    f2=1

                    break

                else:

                    cn=cn.right

        if f1==0 and f2==0:

            print("It is a BST")

        else:

            print("It is not a BST")

b=bst()

b.insert(b.curnode,4)

b.insert(b.curnode,2)

b.insert(b.curnode,5)

b.insert(b.curnode,1)

b.display(b.curnode)

print("")

b.isbst()

**Output:**

