##DELETION IN LINKED LIST

class node:

def \_\_init\_\_(self,data):

self.data=data

self.next=None

class Linkedlist:

def \_\_init\_\_(self):

self.head=None

def printlist(self):

temp=self.head

if not temp:

print("list is empty")

return

else:

print("start",end="")

while temp:

print(temp.data,end="-->")

temp=temp.next

print("end.")

def insert(self,data):

new\_node=node(data)

#linkedlist is empty

if self.head is None:

self.head=new\_node

#if the data data is smaller

elif self.head.data>=new\_node.data:

new\_node.next=self.head

self.head=new\_node

else:

#locate the node before the insertion point

current=self.head

while current.next and new\_node.data > current.next.data:

current=current.next

#insertion

new\_node.next=current.next

current.next=new\_node

def delete(self,key):

#ifthe list is empty

if self.head is None:

print("deleteion error:the list is empty:")

return

#if the key is in head

if self.head.data==key:

self.head=self.head.next

return

current =self.head

while current:

if current.data==key:

break

previous=current

current=current.next

#if the key was not found

if current is None:

print('deletion error :key is not found:')

else:

previous.next=current.next

#\_\_name is a python variable

if \_\_name\_\_=='\_\_main\_\_':

#create an object

LL=Linkedlist()

#insert some nodes

LL.insert(10)

LL.insert(7)

LL.insert(3)

LL.insert(9)

LL.printlist()

LL.delete(9)

LL.printlist()

Output start3-->7-->9-->10-->end.

start3-->7-->10-->end.

#creating the module

#function in any amother file as a module

import fn

fn.printing()

print(\_\_name\_\_)

@#$%^&\*(

def printing():

print("hi") ------🡪here we ctreate a file (fn) and that was accessed to create a module

print("before function")

def f1():

print("f1")

def f2():

print("f2")

def f3():

print("f3")

if \_\_name\_\_=="\_\_main\_\_":

f1()

f2()

f3()

print("name",\_\_name\_\_) #by default \_\_name\_\_ taken as the \_\_main\_\_

DOUBLE LINKED LIST

class node:

def \_\_init\_\_(self,data):

self.data=data

self.previous=None

self.next=None

class dll:

def \_\_init\_\_(self):

self.head=None

def display(self):

if self.head is None:

print("list is empty")

else :

temp=self.head

while temp:

print(temp.data," -->",end="")

temp=temp.next

n=dll()

n1=node(100)

n.head=n1

n2=node(200)

n2.prev=n1

n1.next=n2

n.display()

\*\*INSERTION AT BEGINNING

class node:

def \_\_init\_\_(self,data):

self.data=data

self.prev=None

self.next=None

class dll:

def \_\_init\_\_(self):

self.head=None

def insert\_start(self):

n=node(300)

temp=self.head

temp.prev=n

n.next=temp

self.head=n

def display(self):

if self.head is None:

print("list is empty")

else :

temp=self.head

while temp:

print(temp.data," -->",end="")

temp=temp.next

n=dll()

n1=node(100)

n.head=n1

n2=node(200)

n2.prev=n1

n1.next=n2

n.insert\_start()

n.display()

##INSERT AT ANY POSITION

()'''

class node:

def \_\_init\_\_(self,data):

self.data=data

self.prev=None

self.next=None

class dll:

def \_\_init\_\_(self):

self.head=None

def insert\_pos(self,pos):

n=node(300)

temp=self.head

for i in range(1,pos-1):

temp=temp.next

n.prev=temp

n.next=temp.next

temp.next.prev=n

temp.next=n

def display(self):

if self.head is None:

print("list is empty")

else :

temp=self.head

while temp:

print(temp.data," -->",end="")

temp=temp.next

n=dll()

n1=node(100)

n.head=n1

n2=node(200)

n2.prev=n1

n1.next=n2

n3=node(400)

n3.prev=n2

n2.next=n3

n.insert\_pos(2)

n.display()

DELETION AT THE BEGINNING

class node:

def \_\_init\_\_(self,data):

self.data=data

self.prev=None

self.next=None

class dll:

def \_\_init\_\_(self):

self.head=None

def delete\_start(self):

temp=self.head

self.head=temp.next

temp.next=None

def display(self):

if self.head is None:

print("list is empty")

else :

temp=self.head

while temp:

print(temp.data," -->",end="")

temp=temp.next

n=dll()

n1=node(100)

n.head=n1

n2=node(200)

n2.prev=n1

n1.next=n2

n3=node(400)

n3.prev=n2

n2.next=n3

n.delete\_start()

n.display()

\*\*DELETION AT THE END

class node:

def \_\_init\_\_(self,data):

self.data=data

self.prev=None

self.next=None

class dll:

def \_\_init\_\_(self):

self.head=None

def delete\_end(self):

temp=self.head.next

self.head=temp.prev

prev=self.head

while temp.next is not None:

temp=temp.next

prev=prev.next

prev.next=None

def display(self):

if self.head is None:

print("list is empty")

else :

temp=self.head

while temp:

print(temp.data," -->",end="")

temp=temp.next

n=dll()

n1=node(100)

n.head=n1

n2=node(200)

n2.prev=n1

n1.next=n2

n3=node(400)

n3.prev=n2

n2.next=n3

n.delete\_end()

n.display()

\*\*DELETION AT THE POS

class node:

def \_\_init\_\_(self,data):

self.data=data

self.prev=None

self.next=None

class dll:

def \_\_init\_\_(self):

self.head=None

def delete\_pos(self,pos):

temp=self.head.next

prev= self.head

prev=self.head

for i in range(1,pos-1):

temp=temp.next

prev=prev.next

prev.next=temp.next

prev.next=None

def display(self):

if self.head is None:

print("list is empty")

else :

temp=self.head

while temp:

print(temp.data," -->",end="")

temp=temp.next

n=dll()

n1=node(100)

n.head=n1

n2=node(200)

n2.prev=n1

n1.next=n2

n3=node(400)

n3.prev=n2

n2.next=n3

n.delete\_pos(2)

n.display()

class node:

def \_\_init\_\_(self,data):

self.data=data

self.next=None

self.tail=None

class createlist:

def \_\_init\_\_(self):

self.head=node(None)

self.next=node(None)

self.head.next=self.tail

self.tail.next=self.head

#addinf the node at the end of LL

def add(self,data):

newnode=node(data)

#is empty

if self.head.data is None:

self.head=newnode

self.tail=newnode

newnode.next=self.head

else :

self.tail.next=newnode

self.tail=newnode

#it is CLL so taril is pointing to head

self.tail.next=self.head

def display(self):

current=self.head

if self.head is None:

print("empty list")

return

else:

print("Node of the circular linked list:")

print(current.data,"-->")

while(current.next != self.head):

current=current.next

print(current.data,"-->")

class circularlinkedlist:

c1=createlist()

c1.add("S")

c1.add("M")

c1.add("I")

c1.add("L")

c1.add("E")

c1.display()