**Practical 1:**

Apply the DDL algorithm and Create a doubly linked list and perform the following operations: insert from head, insert from tail, insert from specified position and display

**Solution:**

class Node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.prev=None

        self.next=None

class DLL:

    def \_\_init\_\_(self):

        self.head=None

        self.tail=None

    def insert\_head(self,data):

        newNode=Node(data)

        if self.head is None:

            self.head=newNode

            self.tail=newNode

        else:

            newNode.next=self.head

            self.head.prev=newNode

            self.head=newNode

    def insert\_tail(self,data):

        newNode=Node(data)

        if self.head is None:

            self.head=newNode

            self.tail=newNode

        else:

            self.tail.next=newNode

            newNode.prev=self.tail

            self.tail=newNode

    def insert\_spec(self,data,loc):

        newNode=Node(data)

        if self.head is None:

            self.head=newNode

            self.tail=newNode

        elif loc==1:

            self.insert\_head(data)

        else:

            cNode=self.head

            cLoc=1

            while cNode is not None and cLoc<loc-1:

                cNode=cNode.next

                cLoc+=1

            newNode.next=cNode.next

            cNode.next.prev=newNode

            newNode.prev=cNode

            cNode.next=newNode

    def display(self):

        temp=self.head

        while temp:

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

            temp=temp.next

        print("None")

dll=DLL()

dll.insert\_head(10)

dll.insert\_head(20)

dll.display()

dll.insert\_tail(40)

dll.insert\_tail(50)

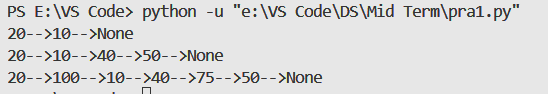
dll.display()

dll.insert\_spec(100,2)

dll.insert\_spec(75,5)

dll.display()

**output:**

****

**Practical 2:**

Apply the DDL algorithm and Create doubly linked list and perform the following operations: delete from head, delete from tail, delete from specified position and display

**Solution:**

class Node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.prev=None

        self.next=None

class DLL:

    def \_\_init\_\_(self):

        self.head=None

        self.tail=None

    def insert\_head(self,data):

        newNode=Node(data)

        if self.head is None:

            self.head=newNode

            self.tail=newNode

        else:

            newNode.next=self.head

            self.head.prev=newNode

            self.head=newNode

    def del\_head(self):

        if self.head is None:

            print("Linked list is Empty")

        else:

            self.head=self.head.next

            self.head.prev=None

    def del\_tail(self):

        if self.head is None:

            print("Linked list is Empty")

        else:

            self.tail=self.tail.prev

            self.tail.next.prev=None

            self.tail.next=None

    def del\_spec(self,loc):

        if self.head is None:

            print("Linked list is Empty")

        elif loc==1:

            self.del\_head()

        else:

            cNode=self.head

            cLoc=1

            while cNode is not None and cLoc<loc-1:

                cNode=cNode.next

                cLoc+=1

            if cNode.next.next:

                cNode.next=cNode.next.next

                cNode.next.prev=cNode

            else:

                cNode.next=None

    def display(self):

        temp=self.head

        while temp:

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

            temp=temp.next

        print("None")

dll=DLL()

dll.insert\_head(10)

dll.insert\_head(20)

dll.insert\_head(30)

dll.insert\_head(40)

dll.insert\_head(50)

dll.display()

dll.del\_head()

dll.display()

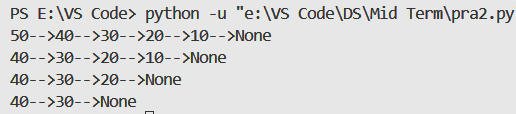
dll.del\_tail()

dll.display()

dll.del\_spec(3)

dll.display()

**Output:**

****

**Practical 3:**

Apply the SLL algorithm and Create Circular Singly linked list and perform the following operations: insert from head, insert from tail, insert from specified position and display

**Solution:**

class node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.next=None

class circlList:

    def \_\_init\_\_(self):

        self.head=None

    def insert\_beg(self,data):

        newNode = node(data)

        if self.head==None:

            self.head=newNode

            newNode.next=self.head

        else:

            cNode=self.head

            while cNode.next != self.head:

                cNode=cNode.next

            newNode.next=self.head

            self.head=newNode

            cNode.next=self.head

    def insert\_end(self,data):

        newNode=node(data)

        if self.head==None:

            self.head=newNode

            newNode.next=self.head

        else:

            cNode=self.head

            while cNode.next != self.head:

                cNode=cNode.next

            cNode.next=newNode

            newNode.next=self.head

    def insert\_spec(self,data,loc):

        newNode=node(data)

        if self.head==None:

            self.head=newNode

            newNode.next=self.head

        elif loc==1:

            self.insert\_beg(data)

        else:

            cNode=self.head

            cLoc=1

            while cNode.next!=self.head and cLoc<loc-1:

                cNode=cNode.next

                cLoc=cLoc+1

            newNode.next=cNode.next

            cNode.next=newNode

    def display(self):

        if self.head==None:

            print("Linked list is empty: ")

        else:

            cNode=self.head

            while cNode.next!=self.head:

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

                cNode=cNode.next

            print(cNode.data)

cl = circlList()

cl.insert\_beg(10)

cl.insert\_beg(20)

cl.display()

cl.insert\_end(40)

cl.insert\_end(50)

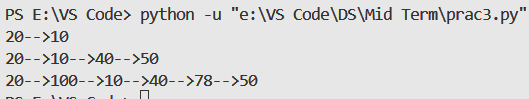
cl.display()

cl.insert\_spec(100,2)

cl.insert\_spec(78,4)

cl.display()

**Output:**

****

**Practical 4:**

Apply the Circular SLL algorithm and Create Circular Singly linked list and perform the following operations: delete from head, delete from tail , delete from specified position and display

**Solution:**

class node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class circlList:

    def \_\_init\_\_(self):

        self.head = None

    def insert\_beg(self, data):

        newNode = node(data)

        if self.head is None:

            self.head = newNode

            newNode.next = self.head

        else:

            cNode = self.head

            while cNode.next != self.head:

                cNode = cNode.next

            newNode.next = self.head

            self.head = newNode

            cNode.next = self.head

    def delete\_spec(self, loc):

        if self.head is None:

            print("Linked list is already empty")

            return

        if self.head.next == self.head and loc == 1:

            self.head = None

            print("After deletion, the list is empty")

            return

        if loc == 1:

            self.delete\_beg()

            return

        cNode = self.head

        cLoc = 1

        while cNode.next != self.head and cLoc < loc - 1:

            cNode = cNode.next

            cLoc += 1

        if cNode.next == self.head:

            print("Position out of range")

            return

        cNode.next = cNode.next.next

    def delete\_beg(self):

        if self.head is None:

            print("Linked list is already empty")

            return

        if self.head.next == self.head:

            self.head = None

            print("After deletion, the list is empty")

            return

        cNode = self.head

        while cNode.next != self.head:

            cNode = cNode.next

        self.head = self.head.next

        cNode.next = self.head

    def delete\_end(self):

        if self.head is None:

            print("Linked list is already empty")

            return

        if self.head.next == self.head:

            self.head = None

            print("After deletion, the list is empty")

            return

        cNode = self.head

        while cNode.next.next != self.head:

            cNode = cNode.next

        cNode.next = self.head

    def display(self):

        if self.head is None:

            print("Linked list is empty")

            return

        cNode = self.head

        while cNode.next != self.head:

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

            cNode = cNode.next

        print(cNode.data)

cl = circlList()

cl.insert\_beg(10)

cl.insert\_beg(20)

cl.insert\_beg(30)

cl.insert\_beg(40)

cl.insert\_beg(50)

cl.display()

cl.delete\_beg()

cl.display()

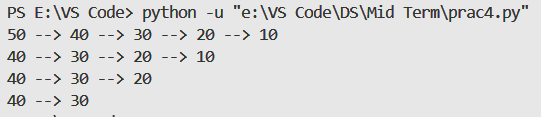
cl.delete\_end()

cl.display()

cl.delete\_spec(3)

cl.display()

**Output:**

****

**Practical 5:**

Apply the circular DLL algorithm and Create Circular Doubly linked list and perform the following operations: insert from head, insert from tail, insert from specified position and display

**Solution:**

# Apply the Circular DLL algorithm and Create Circular Doubly

# linked list and perform the following operations: insert from

# head, insert from tail , insert from specified position and display

class node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.next=None

        self.prev=None

class circlList:

    def \_\_init\_\_(self):

        self.head=None

    def insert\_beg(self,data):

        newNode = node(data)

        if self.head==None:

            self.head=newNode

            newNode.next=self.head

            newNode.prev=self.head

        else:

            cNode=self.head

            while cNode.next != self.head:

                cNode=cNode.next

            newNode.next=self.head

            newNode.prev=cNode

            self.head=newNode

            cNode.next=self.head

    def insert\_end(self,data):

        newNode=node(data)

        if self.head==None:

            self.head=newNode

            newNode.next=self.head

        else:

            cNode=self.head

            while cNode.next != self.head:

                cNode=cNode.next

            newNode.prev=cNode

            cNode.next=newNode

            newNode.next=self.head

    def insert\_spec(self,data,loc):

        newNode=node(data)

        if self.head==None:

            self.head=newNode

            newNode.next=self.head

            newNode.prev=self.head

        elif loc==1:

            self.insert\_beg(data)

        else:

            cNode=self.head

            cLoc=1

            while cNode.next!=self.head and cLoc<loc-1:

                cNode=cNode.next

                cLoc=cLoc+1

            newNode.next=cNode.next

            cNode.next.prev=newNode

            cNode.next=newNode

            newNode.prev=cNode

    def display(self):

        if self.head==None:

            print("Linked list is empty: ")

        else:

            cNode=self.head

            while cNode.next!=self.head:

                print(cNode.data,end="<-->")

                cNode=cNode.next

            print(cNode.data)

    def display\_rev(self):

        if self.head == None:

            print("Linked list is empty")

        else:

            cNode = self.head

            while cNode.next != self.head:

                cNode = cNode.next

            temp = cNode

            while temp != self.head:

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

                temp = temp.prev

            print(temp.data)

cl = circlList()

cl.insert\_beg(10)

cl.insert\_beg(20)

cl.display()

cl.insert\_end(40)

cl.insert\_end(50)

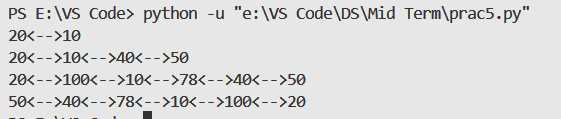
cl.display()

cl.insert\_spec(100,2)

cl.insert\_spec(78,4)

cl.display()

cl.display\_rev()

**Output:  
**

**Practical 6:**

Apply the Circular DLL algorithm and Create Circular Doubly linked list and perform the following operations: delete from head, delete from tail, delete from specified position and display

**Solution:**

class node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

        self.prev = None

class circlList:

    def \_\_init\_\_(self):

        self.head = None

    def insert\_beg(self, data):

        newNode = node(data)

        if self.head == None:

            self.head = newNode

            newNode.next = self.head

            newNode.prev = self.head

        else:

            cNode = self.head

            while cNode.next != self.head:

                cNode = cNode.next

            newNode.next = self.head

            newNode.prev = cNode

            self.head.prev = newNode

            self.head = newNode

            cNode.next = self.head

    def delete\_head(self):

        if self.head == None:

            print("Linked list is already empty")

        elif self.head.next == self.head:

            self.head = None

        else:

            cNode = self.head

            while cNode.next != self.head:

                cNode = cNode.next

            self.head = self.head.next

            self.head.prev = cNode

            cNode.next = self.head

    def delete\_tail(self):

        if self.head == None:

            print("Linked list is already empty")

        elif self.head.next == self.head:

            self.head = None

        else:

            cNode = self.head

            while cNode.next != self.head:

                cNode = cNode.next

            cNode.prev.next = self.head

            self.head.prev = cNode.prev

    def delete\_spec(self, loc):

        if self.head == None:

            print("Linked list is already empty")

        elif loc == 1:

            self.delete\_head()

        else:

            cNode = self.head

            cLoc = 1

            while cNode.next != self.head and cLoc < loc:

                cNode = cNode.next

                cLoc += 1

            if cNode == self.head:

                print("Location exceeds the length of the list")

            else:

                cNode.prev.next = cNode.next

                cNode.next.prev = cNode.prev

    def display(self):

        if self.head == None:

            print("Linked list is empty")

        else:

            cNode = self.head

            while cNode.next != self.head:

                print(cNode.data, end="<-->")

                cNode = cNode.next

            print(cNode.data)

cl = circlList()

cl.insert\_beg(10)

cl.insert\_beg(20)

cl.insert\_beg(30)

cl.insert\_beg(40)

cl.insert\_beg(50)

cl.display()

cl.delete\_head()

cl.display()

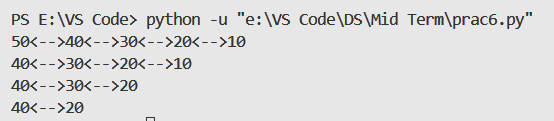
cl.delete\_tail()

cl.display()

cl.delete\_spec(2)

cl.display()

**Output:**

****

**Practical 7:**

Apply Stack Algorithm and create Stack using array and perform following operations: Push, POP, Peek

**Solution:**

class Stack:

    def \_\_init\_\_(self):

        self.stack = []

    def push(self, item):

        self.stack.append(item)

    def isEmpty(self):

        return len(self.stack) == 0

    def pop(self):

        if self.isEmpty():

            return("Underflow - Stack is empty")

        else:

            return self.stack.pop()

    def peek(self):

        if self.isEmpty():

            return("Underflow - Stack is empty")

        else:

            return self.stack[-1]

    def display(self):

        if self.isEmpty():

            return("Underflow - Stack is empty")

        else:

            print(self.stack)

s=Stack()

s.push(10)

s.push(20)

s.push(30)

s.display()

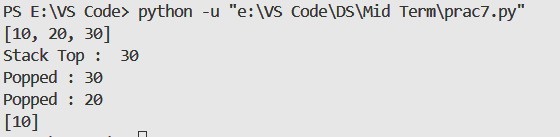
print("Stack Top : ",s.peek())

print("Popped :",s.pop())

print("Popped :",s.pop())

s.display()

**Output:**

****

**Practical 8:**

Apply Stack Algorithm and create Stack using linked list and perform following operations: Push, POP, Peek

**Solution:**# Apply Stack Algorithm and create Stack using linked list and

# perform following operations: Push, POP, Peek

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next=None

class Stack:

    def \_\_init\_\_(self):

        self.top=None

    def push(self,data):

        newNode=Node(data)

        if self.top is None:

            self.top=newNode

        else:

            newNode.next=self.top

            self.top=newNode

    def pop(self):

        if self.top is None:

            return "Stack is empty"

        else:

            popped\_node=self.top

            self.top=self.top.next

            popped\_node.next=None

            return popped\_node.data

    def peek(self):

        if self.top is None:

            return "Stack is empty"

        else:

            return self.top.data

    def display(self):

        if self.top is None:

            print("Stack is Empty")

        else:

            temp = self.top

            print("----")

            while temp:

                print(temp.data)

                print("----")

                temp=temp.next

s=Stack()

s.push(10)

s.push(20)

s.push(30)

s.push(40)

print("After pushing: ")

s.display()

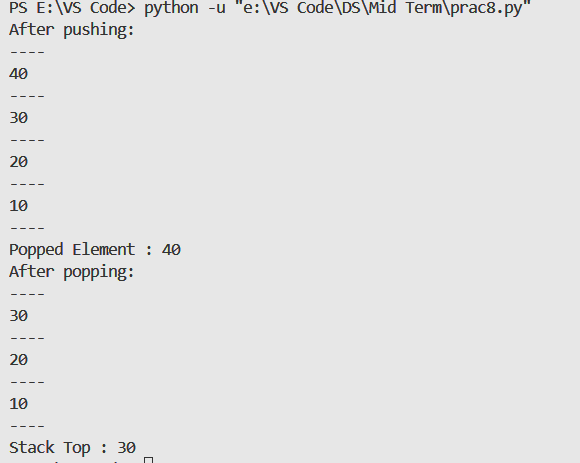
print("Popped Element :",s.pop())

print("After popping: ")

s.display()

print("Stack Top :",s.peek())

**Output:**

****