**Assignment-1**

1. **Aim:** Python program to implement operations of linked list

**Code:**

class Node:

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

node.data=data

node.next=None

class LinkedList:

def \_\_init\_\_(node):

node.head=None

def display(node):

print("The Linked List is:", end=" ")

temp=node.head

while temp:

print(temp.data, end=" ")

temp=temp.next

print()

def insert\_begin(node,data):

newnode=Node(data)

newnode.next=node.head

node.head=newnode

def insert\_end(node,data):

newnode=Node(data)

if node.head is None:

node.head=newnode

return

temp=node.head

while temp.next:

temp=temp.next

temp.next=newnode

def insert\_position(node,data,pos):

newnode=Node(data)

if pos==0:

node.insert\_begin(data)

return

temp=node.head

for i in range(pos-1):

if temp is None:

print("Invalid Position")

return

temp=temp.next

newnode.next=temp.next

temp.next=newnode

def delete\_begin(node):

if node.head is None:

print("Linked List is Empty")

return

node.head=node.head.next

def delete\_end(node):

if node.head is None:

print("Linked List is Empty")

return

if node.head.next is None:

node.head=None

return

temp=node.head

while temp.next.next:

temp=temp.next

temp.next=None

def delete\_position(node,pos):

if node.head is None:

print("Linked List is Empty")

return

if pos==0:

node.head=node.head.next

return

temp=node.head

for i in range(pos-1):

if temp.next is None:

print("Invalid Position")

return

temp=temp.next

if temp.next is None:

print("Invalid Position")

return

temp.next=temp.next.next

def delete\_element(node,key):

temp=node.head

if temp and temp.data==key:

node.head=temp.next

return

prev=None

while temp and temp.data!=key:

prev=temp

temp=temp.next

if temp is None:

print("Element Not Found")

return

prev.next=temp.next

def search(node,key):

temp=node.head

index=0

while temp:

if temp.data==key:

print("Element {} found at index {}".format(key,index))

return

temp=temp.next

index=index+1

print("Element Not Found")

def replace(node,index,newdata):

temp=node.head

for i in range(index):

if temp is None:

print("Invalid Index")

return

temp=temp.next

if temp is not None:

temp.data=newdata

else:

print("Invalid Index")

def forward(node):

node.display()

def reverse(node):

def \_reverse(node):

if node is None:

return

\_reverse(node.next)

print(node.data, end=" ")

\_reverse(node.head)

print()

def insert\_after(node,key,data):

temp=node.head

while temp and temp.data!=key:

temp=temp.next

if temp is None:

print("Element Not Found")

return

newnode=Node(data)

newnode.next=temp.next

temp.next=newnode

ll=LinkedList()

print("Ahmed Ansari")

while True:

print('''1.Display List

2.Insert At Beginning

3.Insert At End

4.Insert At Position

5.Insert After An Element

6.Delete From Beginning

7.Delete From End

8.Delete At Position

9.Delete An Element

10.Search An Element

11.Replace Element At A Specified Index

12.Forward Traversal

13.Reverse Traversal

14.Exit''')

choice=int(input("Enter Your Choice: "))

if choice==1:

ll.display()

elif choice==2:

data=int(input("Enter Data: "))

ll.insert\_begin(data)

elif choice==3:

data=int(input("Enter Data: "))

ll.insert\_end(data)

elif choice==4:

data=int(input("Enter Data: "))

pos=int(input("Enter the Position: "))

ll.insert\_position(data,pos-1)

elif choice==5:

data=int(input("Enter Data: "))

key=int(input("Enter Element after which to Insert: "))

ll.insert\_after(key,data)

elif choice==6:

ll.delete\_begin()

elif choice==7:

ll.delete\_end()

elif choice==8:

pos=int(input("Enter the Position: "))

ll.delete\_position(pos-1)

elif choice==9:

key=int(input("Enter the Element to Delete: "))

ll.delete\_element(key)

elif choice==10:

key=int(input("Enter the Element to Search: "))

ll.search(key)

elif choice==11:

index=int(input("Enter the Index: "))

newdata=int(input("Enter the new Value: "))

ll.replace(index,newdata)

elif choice==12:

ll.forward()

elif choice==13:

ll.reverse()

elif choice==14:

print("Exiting...")

break

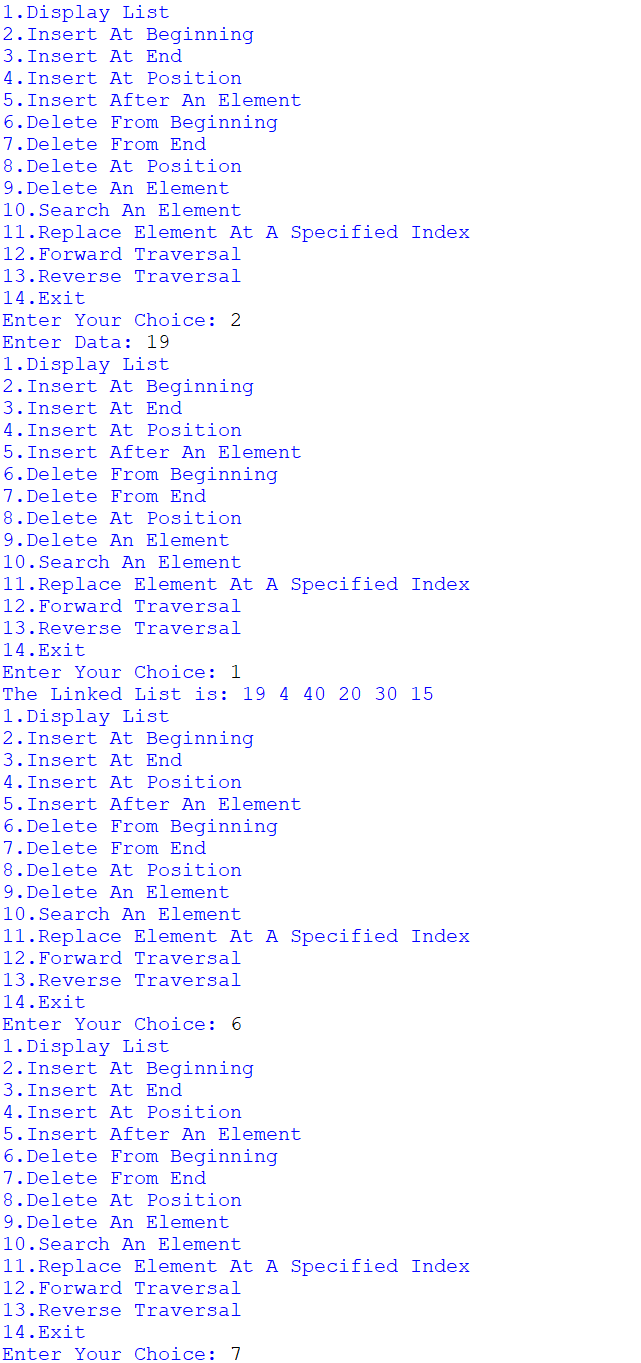
else:

print("Invalid Input")

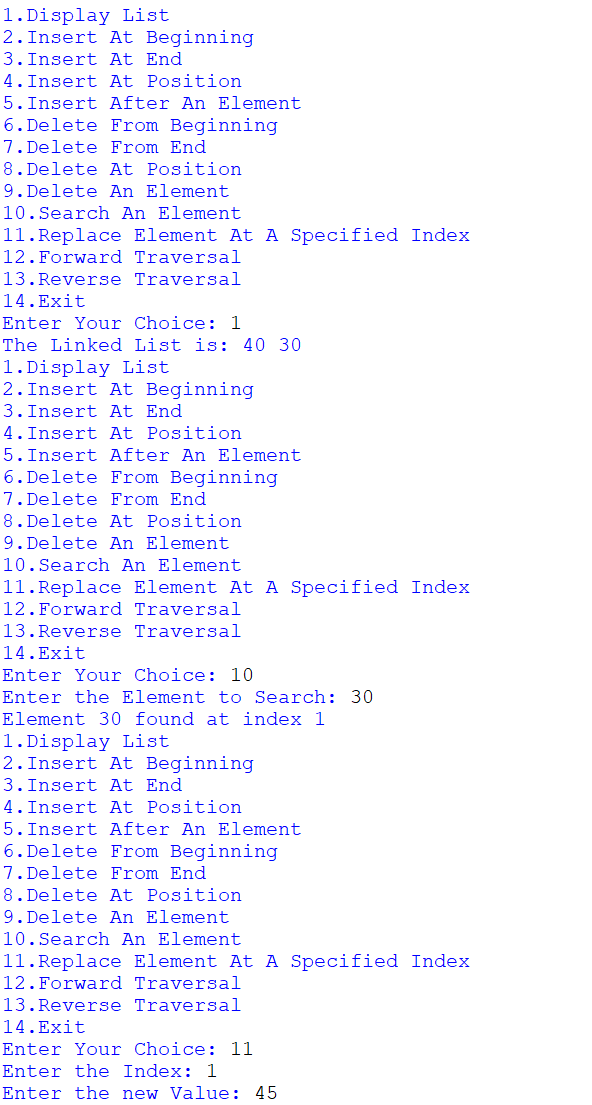
**Output:**













1. **Aim:** Python program to implement Stack data structure. Create class Stack with following functions.

**Code:**

class stack:

def \_\_init\_\_(st):

st.stack=[]

def push(st, data):

st.stack.append(data)

def pop(st):

if len(st.stack)==0:

print("The Stack is Empty")

return

return st.stack.pop()

def top(st):

if len(st.stack)==0:

print("The Stack is Empty")

return

return st.stack[-1]

def search(st,element):

l=len(st.stack)

for i in range(l):

if st.stack[i]==element:

print("The Element {} is Found at Index {}".format(st.stack[i],i+1))

return

print("The Element is Not Found")

return

def display(st):

if len(st.stack)==0:

print("The Stack is Empty")

else:

print("The Stack is:",st.stack[::-1])

s=stack()

print("Ahmed Ansari")

while True:

print('''1.Push an Element

2.Pop an Element

3.Top of Stack

4.Search an Element

5.Display Stack

6.Exit''')

choice=int(input("Enter Your Choice: "))

if choice==1:

data=int(input("Enter the Data: "))

s.push(data)

elif choice==2:

temp=s.pop()

if temp is not None:

print("Popped Element is ",temp)

elif choice==3:

top=s.top()

if top is not None:

print("Top Element is ",top)

elif choice==4:

element=int(input("Enter the Element to be Searched: "))

s.search(element)

elif choice==5:

s.display()

elif choice==6:

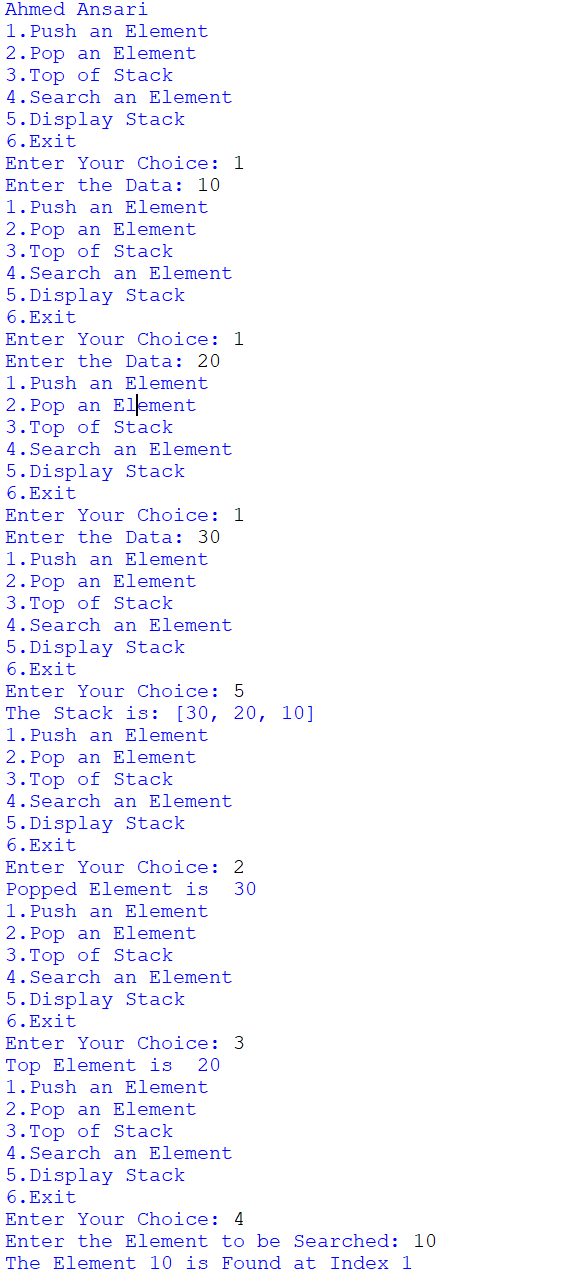
print("Exiting...")

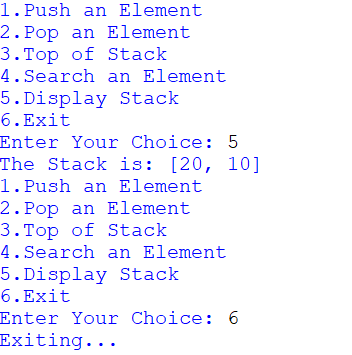
break;

else:

print("Invalid Input")

**Output:**





**3) Aim:** Python program to implement Queue data structure create class Queue with following functions.

**Code:**

class Queue:

def \_\_init\_\_(q):

q.queue=[]

def insert(q,data):

q.queue.append(data)

def remove(q):

if len(q.queue)==0:

print("Queue Underflow")

else:

temp=q.queue.pop(0)

print("Deleted Element is: ",temp)

def search(q,key):

for i in range(len(q.queue)):

if key==q.queue[i]:

print("Element {} found at Position {}".format(key,i))

return

print("Element Not Found")

return

def display(q):

if len(q.queue)==0:

print("Queue Underflow")

else:

print("Queue Elements are:",end=" ")

for i in range(len(q.queue)):

print(q.queue[i],end=" ")

print()

q=Queue()

print("Ahmed Ansari")

while True:

print('''1.Insert an Element

2.Remove an Element

3.Search an Element

4.Display Queue

5.Exit''')

choice=int(input("Enter Your Choice: "))

if choice==1:

data=int(input("Enter Data: "))

q.insert(data)

elif choice==2:

q.remove()

elif choice==3:

key=int(input("Enter the Element to be Searched: "))

q.search(key)

elif choice==4:

q.display()

elif choice==5:

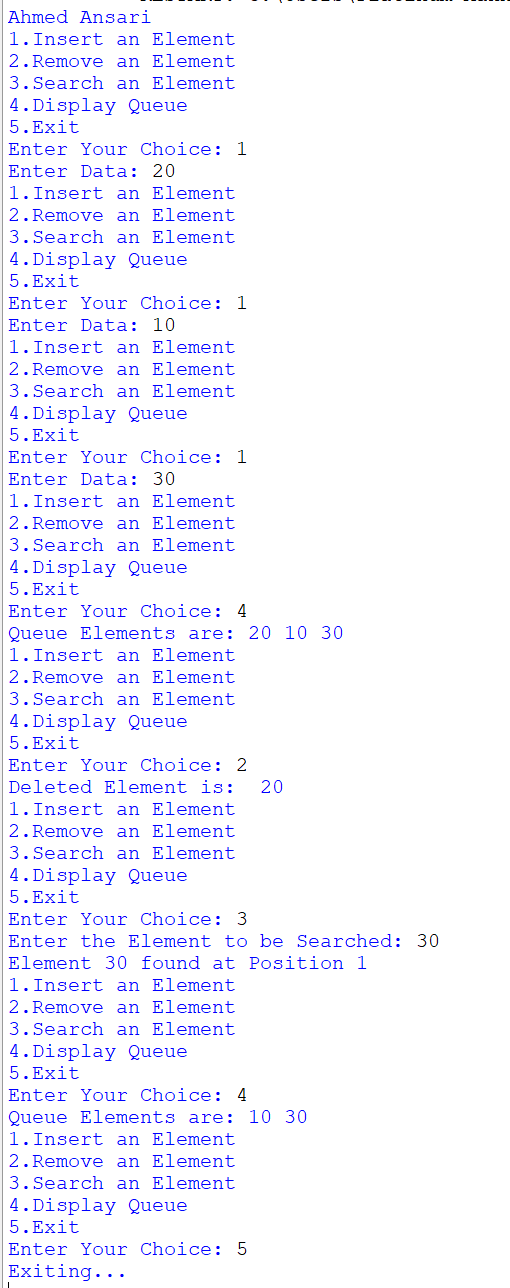
print("Exiting...")

break;

else:

print("Invalid Input")

**Output:**



**4) Aim:** Python program to use deque class from collections with following functions.

**Code:**

from collections import deque

class DequeOperations:

def \_\_init\_\_(dq):

dq.deque=deque()

def add\_front(dq,data):

dq.deque.appendleft(data)

print("{} added to the front".format(data))

def remove\_front(dq):

if dq.deque is None:

print("Deque is Empty")

else:

print("Removed {} from the Front".format(dq.deque.popleft()))

def add\_rear(dq,data):

dq.deque.append(data)

print("{} added to the rear".format(data))

def remove\_rear(dq):

if dq.deque is None:

print("Deque is Empty")

else:

print("Removed {} from the rear".format(dq.deque.pop()))

def search\_element(dq,key):

for i in range(len(dq.deque)):

if key==dq.deque[i]:

print("Element {} found at Index {}".format(key,i))

else:

print("Element Not Found")

def display(dq):

if dq.deque is None:

print("Deque is Empty")

else:

print("Deque Elements:",list(dq.deque))

dq=DequeOperations()

print("Ahmed Ansari")

while True:

print('''1.Add Element at Front

2.Remove Element from Front

3.Add Element from Front

4.Remove Element from Rear

5.Search for an Element

6.Display

7.Exit''')

choice=int(input("Enter Your Choice: "))

if choice==1:

data=int(input("Enter Data: "))

dq.add\_front(data)

elif choice==2:

dq.remove\_front()

elif choice==3:

data=int(input("Enter Data: "))

dq.add\_rear(data)

elif choice==4:

dq.remove\_rear()

elif choice==5:

key=int(input("Enter the Element to be Searched: "))

dq.search\_element(key)

elif choice==6:

dq.display()

elif choice==7:

print("Exiting...")

break

else:

print("Invalid Input")

**Output:**

