**Data Structures**

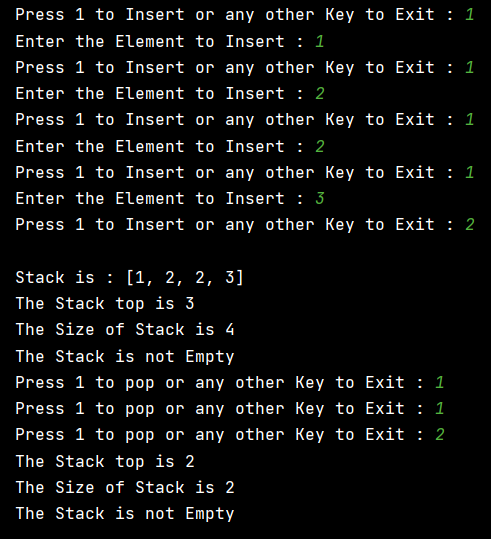
**S Abhishek**

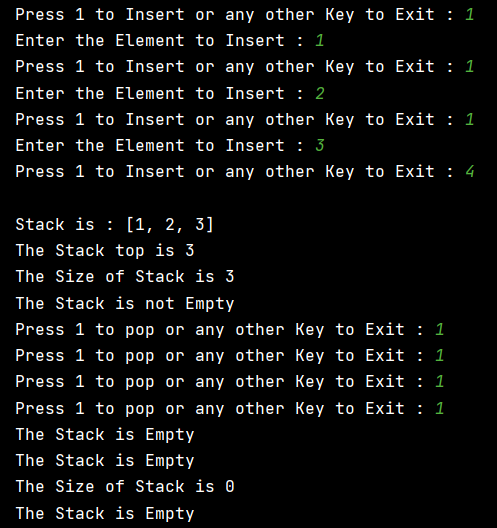
**AM.EN.U4CSE19147**

**1. Write a Python program to implement stack and its operations using array (use python list).**

def push(stack):  
 while True:  
  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 stack.append(x)  
 else:  
 return  
  
def top(stack):  
 if len(stack) == **0**:  
 print("The Stack is Empty")  
 return  
 else:  
 print("The Stack top is {}".format(stack[len(stack)-**1**]))  
  
def size(stack):  
 print("The Size of Stack is {}".format(len(stack)))  
  
def isempty(stack):  
 if len(stack) == **0**:  
 print("The Stack is Empty")  
 return  
 else:  
 print("The Stack is not Empty")  
  
def display(stack):  
 if len(stack) == **0**:  
 print("The Stack is Empty")  
 return  
 else:  
 print("\nStack is : {}".format(s))  
  
def pop(stack):  
 while True:  
 x = int(input("Press 1 to pop or any other Key to Exit : "))  
 if x == **1**:  
 if len(stack) == **0**:  
 print("The Stack is Empty")  
 return  
 else:  
 stack.pop()  
 else:  
 return

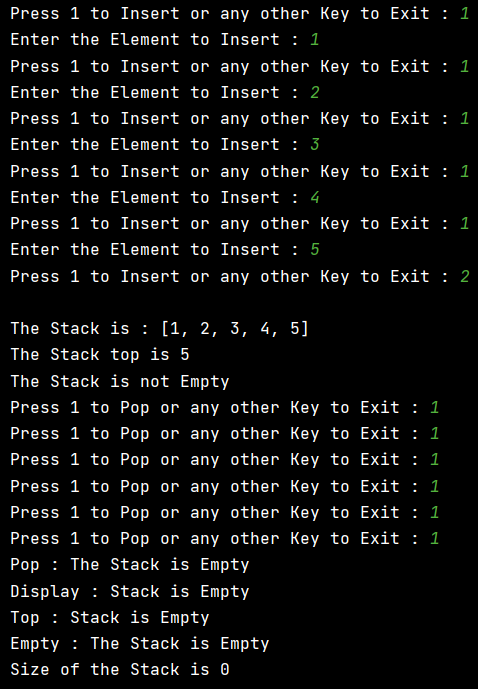
s = []  
push(s)  
display(s)  
top(s)  
size(s)  
isempty(s)  
pop(s)  
top(s)  
size(s)  
isempty(s)

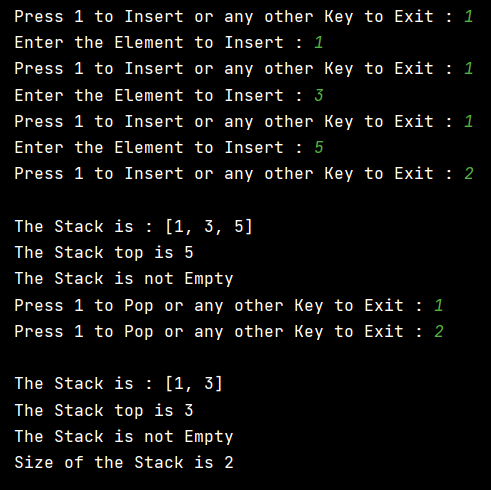




**2. Write a Python program to implement stack and its operations using linked list.**

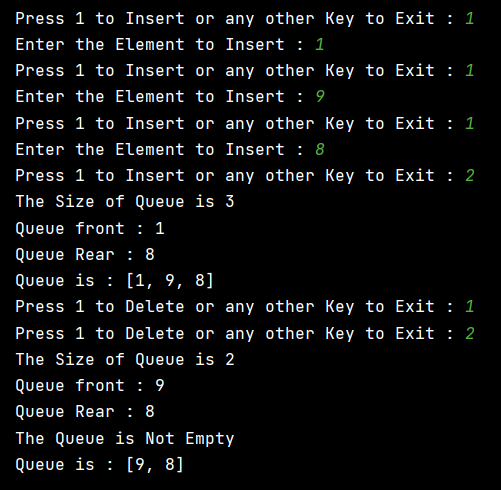
class Node:  
 def \_\_init\_\_(self**,**data):  
 self.data = data  
 self.next = None  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.head = None  
 self.size = **0** def push(self):  
 while True:  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 node = Node(x)  
  
 if self.head is None:  
 self.head = node  
 self.size = self.size + **1** else:  
 node.next = self.head  
 self.head = node  
 self.size = self.size + **1** else:  
 return  
  
 def display(self):  
 if self.size == **0**:  
 print("Display : Stack is Empty")  
 return  
 else:  
 temp = self.head  
 s = []  
 while temp:  
 s.append(temp.data)  
 temp = temp.next  
  
 s.reverse()  
 print("\nThe Stack is : {}".format(s))  
  
 def top(self):  
 if self.size == **0**:  
 print("Top : Stack is Empty")  
 return  
 else:  
 print("The Stack top is {}".format(self.head.data))  
  
 def isempty(self):  
 if self.size == **0**:  
 print("Empty : The Stack is Empty")  
 return  
 else:  
 print("The Stack is not Empty")  
  
 def pop(self):  
 while True:  
 x = int(input("Press 1 to Pop or any other Key to Exit : "))  
 if x == **1**:  
 if self.size == **0**:  
 print("Pop : The Stack is Empty")  
 return  
 elif self.size == **1**:  
 self.head = None  
 self.size = self.size - **1** else:  
 self.head = self.head.next  
 self.size = self.size - **1** else:  
 return  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 stack = Stack()  
 stack.push()  
 stack.display()  
 stack.top()  
 stack.isempty()  
 stack.pop()  
 stack.display()  
 stack.top()  
 stack.isempty()  
 print("Size of the Stack is {}".format(stack.size))  
 #print(stack.size)

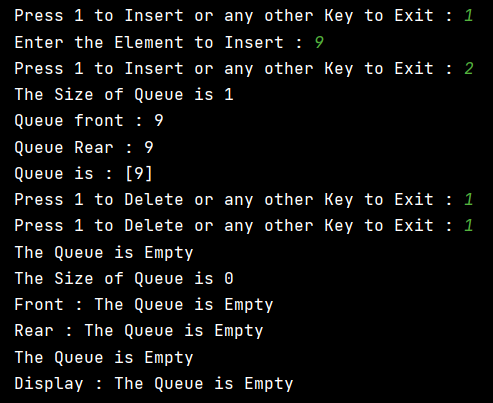
****

****

**3. Write a Python program to implement queue and its operations using array (use python list).**

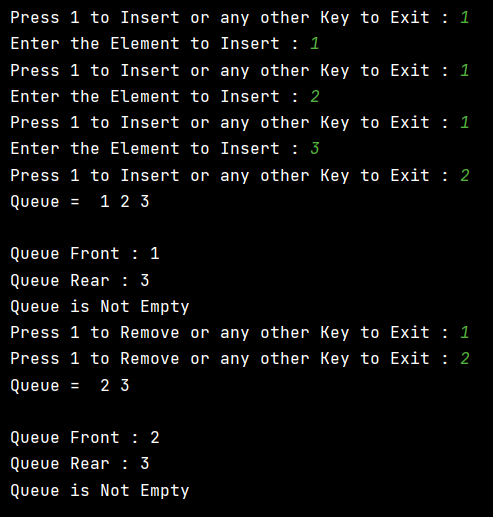
class Queue:  
 def \_\_init\_\_(self):  
 self.queue = []  
  
 def enqueue(self):  
 while True:  
  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 self.queue.append(x)  
 else:  
 return  
  
 def dequeue(self):  
 while True:  
 x = int(input("Press 1 to Delete or any other Key to Exit : "))  
 if x == **1**:  
 if len(self.queue) == **0**:  
 print("The Queue is Empty")  
 return  
 else:  
 self.queue.pop(**0**)  
 else:  
 return  
  
 def display(self):  
 if len(self.queue) == **0**:  
 print("Display : The Queue is Empty")  
 return  
 else:  
 print("Queue is : {}".format(self.queue))  
  
 def front(self):  
 if len(self.queue) == **0**:  
 print("Front : The Queue is Empty")  
 return  
 else:  
 print("Queue front : {}".format(self.queue[**0**]))  
  
 def rear(self):  
 if len(self.queue) == **0**:  
 print("Rear : The Queue is Empty")  
 return  
 else:  
 print("Queue Rear : {}".format(self.queue[len(self.queue)-**1**]))  
  
 def size(self):  
 print("The Size of Queue is {}".format(len(self.queue)))  
  
 def isempty(self):  
 if len(self.queue) == **0**:  
 print("The Queue is Empty")  
 else:  
 print("The Queue is Not Empty")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 queue = Queue()  
 queue.enqueue()  
 queue.size()  
 queue.front()  
 queue.rear()  
 queue.display()  
 queue.dequeue()  
 queue.size()  
 queue.front()  
 queue.rear()  
 queue.isempty()  
 queue.display()

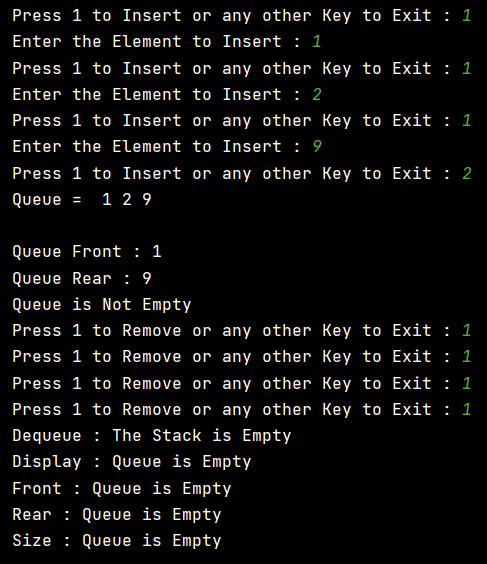
****

****

**4. Write a Python program to implement queue and its operations using linked list.**

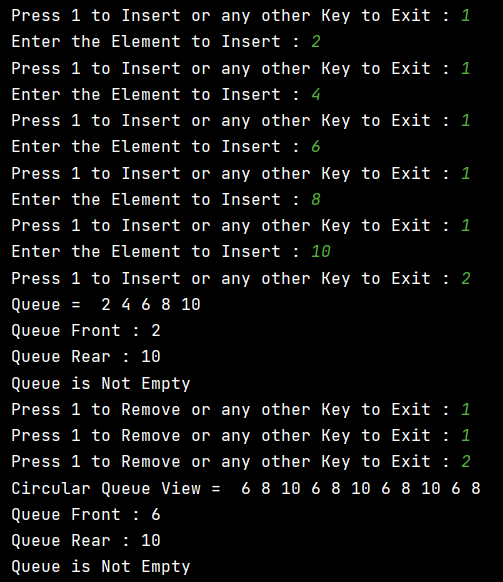
class Node:  
 def \_\_init\_\_(self**,**data):  
 self.data = data  
 self.next = None  
  
class Queue:  
 def \_\_init\_\_(self):  
 self.front = None  
 self.rear = None  
  
 def enqueue(self):  
 while True:  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 node = Node(x)  
  
 if self.rear is None:  
 self.rear = node  
 self.front = node  
 continue  
 else:  
 self.rear.next = node  
 self.rear = node  
 else:  
 return  
  
 def dequeue(self):  
 while True:  
 x = int(input("Press 1 to Remove or any other Key to Exit : "))  
 if x == **1**:  
 if self.front is None:  
 print("Dequeue : The Queue is Empty")  
 return  
  
 elif self.front == self.rear:  
 self.front = None  
 self.rear = None  
  
 else:  
 self.front = self.front.next  
 else:  
 return  
  
 def display(self):  
 if self.front is None:  
 print("Display : Queue is Empty")  
 return  
 else:  
 temp = self.front  
 print("Queue = "**,**end=" ")  
 while temp:  
 print(temp.data**,**end= " ")  
 temp = temp.next  
  
 print("\n")  
  
 def Rear(self):  
 if self.front is None:  
 print("Rear : Queue is Empty")  
 return  
 else:  
 temp = self.front  
 while temp.next:  
 temp = temp.next  
  
 print("Queue Rear : {}".format(temp.data))  
  
 def Front(self):  
 if self.front is None:  
 print("Front : Queue is Empty")  
 return  
 else:  
 print("Queue Front : {}".format(self.front.data))  
  
 def isempty(self):  
 if self.front is None:  
 print("Size : Queue is Empty")  
 return  
 else:  
 print("Queue is Not Empty")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 queue = Queue()  
 queue.enqueue()  
 queue.display()  
 queue.Front()  
 queue.Rear()  
 queue.isempty()  
 queue.dequeue()  
 queue.display()  
 queue.Front()  
 queue.Rear()  
 queue.isempty()

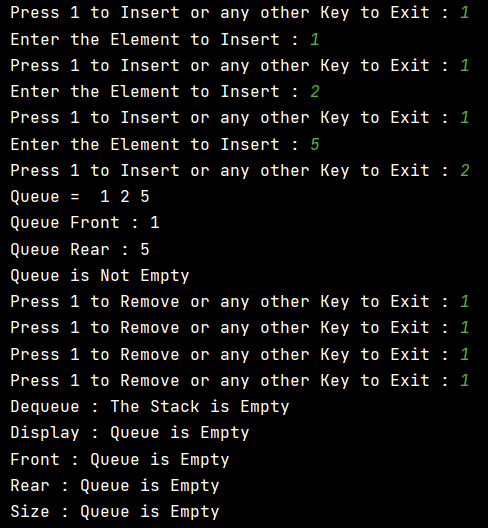
****

****

**5. Write a Python program to implement circular queue.**

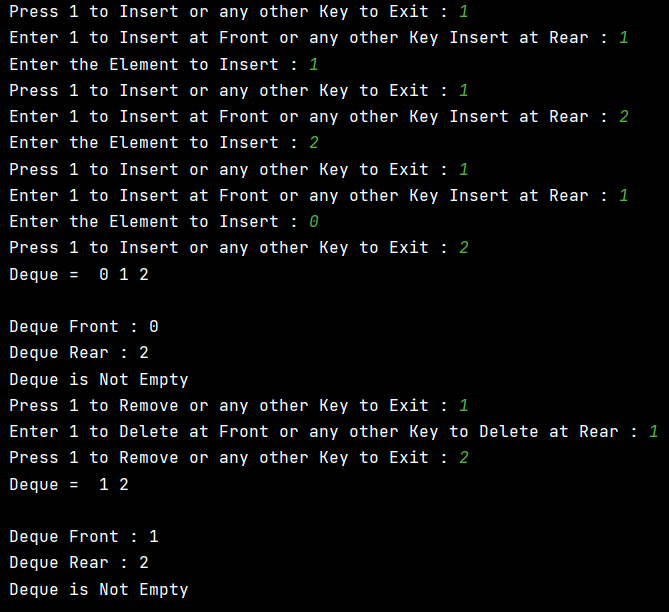
class Node:  
 def \_\_init\_\_(self**,** data):  
 self.data = data  
 self.next = None  
  
  
class Queue:  
 def \_\_init\_\_(self):  
 self.front = None  
 self.rear = None  
  
 def enqueue(self):  
 while True:  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 node = Node(x)  
  
 if self.rear is None:  
 self.rear = node  
 self.front = node  
 continue  
 else:  
 node.next = self.front  
 self.rear.next = node  
 self.rear = node  
 else:  
 return  
  
 def dequeue(self):  
 while True:  
 x = int(input("Press 1 to Remove or any other Key to Exit : "))  
 if x == **1**:  
 if self.front is None:  
 print("Dequeue : The Stack is Empty")  
 return  
  
 elif self.front == self.rear:  
 self.front = None  
 self.rear = None  
  
 else:  
 self.front = self.front.next  
 self.rear.next = self.front  
 else:  
 return  
  
 def display(self):  
 if self.front is None:  
 print("Display : Queue is Empty")  
 return  
 else:  
 temp = self.front  
 print("Queue = "**,** end=" ")  
 while temp != self.rear:  
 print(temp.data**,** end=" ")  
 temp = temp.next  
 print(temp.data)  
  
 def display\_circular(self):  
 if self.front is None:  
 print("Display : Queue is Empty")  
 return  
 else:  
 temp = self.front  
 print("Circular Queue View = "**,** end=" ")  
 i = **0**;  
  
 while i < **10**:  
 i = i + **1** print(temp.data**,** end=" ")  
 temp = temp.next  
 print(temp.data)  
  
 def Rear(self):  
 if self.front is None:  
 print("Rear : Queue is Empty")  
 return  
 else:  
 temp = self.front  
 while temp != self.rear:  
 temp = temp.next  
 print("Queue Rear : {}".format(temp.data))  
  
 def Front(self):  
 if self.front is None:  
 print("Front : Queue is Empty")  
 return  
 else:  
 print("Queue Front : {}".format(self.front.data))  
  
 def isempty(self):  
 if self.front is None:  
 print("Size : Queue is Empty")  
 return  
 else:  
 print("Queue is Not Empty")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 queue = Queue()  
 queue.enqueue()  
 queue.display()  
 queue.Front()  
 queue.Rear()  
 queue.isempty()  
 queue.dequeue()  
 queue.display\_circular()  
 queue.Front()  
 queue.Rear()  
 queue.isempty()

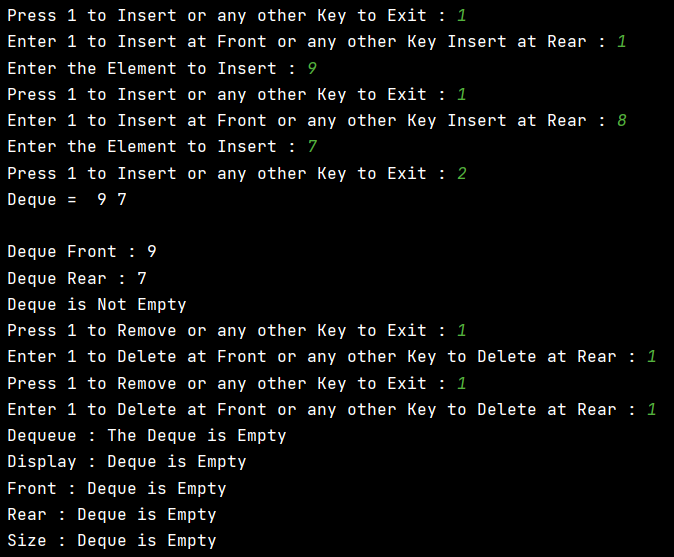
****

****

**6. Write a Python program to implement deque.**

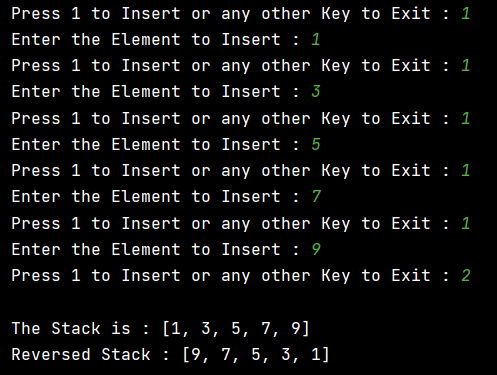
class Node:  
 def \_\_init\_\_(self**,** data):  
 self.data = data  
 self.next = None  
  
  
class Queue:  
 def \_\_init\_\_(self):  
 self.front = None  
 self.rear = None  
  
 def enqueue(self):  
 while True:  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter 1 to Insert at Front or any other Key Insert at Rear : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 node = Node(x)  
  
 if self.rear is None:  
 self.rear = node  
 self.front = node  
 continue  
 else:  
 node.next = self.front  
 self.front = node  
  
 else:  
 x = int(input("Enter the Element to Insert : "))  
 node = Node(x)  
  
 if self.rear is None:  
 self.rear = node  
 self.front = node  
 continue  
 else:  
 self.rear.next = node  
 else:  
 return  
  
 def dequeue(self):  
 while True:  
 x = int(input("Press 1 to Remove or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter 1 to Delete at Front or any other Key to Delete at Rear : "))  
 if x == **1**:  
 if self.front is None:  
 print("Dequeue : The Deque is Empty")  
 return  
  
 elif self.front == self.rear:  
 self.front = None  
 self.rear = None  
  
 else:  
 self.front = self.front.next  
 else:  
 if self.front is None:  
 print("Dequeue : The Deque is Empty")  
 return  
  
 elif self.front == self.rear:  
 self.front = None  
 self.rear = None  
  
 else:  
 temp = self.front  
 while temp.next:  
 temp = temp.next  
 temp.next = None  
 self.rear = temp  
 else:  
 return  
  
 def display(self):  
 if self.front is None:  
 print("Display : Deque is Empty")  
 return  
 else:  
 temp = self.front  
 print("Deque = "**,** end=" ")  
 while temp:  
 print(temp.data**,** end=" ")  
 temp = temp.next  
  
 print("\n")  
  
 def Rear(self):  
 if self.front is None:  
 print("Rear : Deque is Empty")  
 return  
 else:  
 temp = self.front  
 while temp.next:  
 temp = temp.next  
  
 print("Deque Rear : {}".format(temp.data))  
  
 def Front(self):  
 if self.front is None:  
 print("Front : Deque is Empty")  
 return  
 else:  
 print("Deque Front : {}".format(self.front.data))  
  
 def isempty(self):  
 if self.front is None:  
 print("Size : Deque is Empty")  
 return  
 else:  
 print("Deque is Not Empty")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 queue = Queue()  
 queue.enqueue()  
 queue.display()  
 queue.Front()  
 queue.Rear()  
 queue.isempty()  
 queue.dequeue()  
 queue.display()  
 queue.Front()  
 queue.Rear()  
 queue.isempty()

****

****

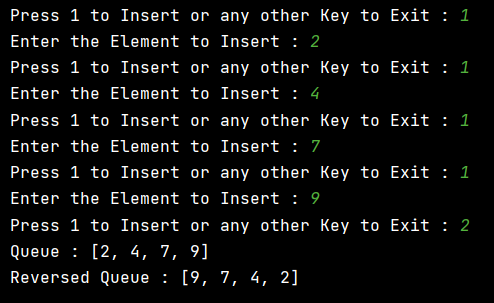
**7. Write a Python program to reverse a stack.**

class Node:  
 def \_\_init\_\_(self**,** data):  
 self.data = data  
 self.next = None  
  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.head = None  
 self.size = **0** self.queue = []  
  
 def push(self):  
 while True:  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 node = Node(x)  
  
 if self.head is None:  
 self.head = node  
 self.size = self.size + **1** else:  
 node.next = self.head  
 self.head = node  
 self.size = self.size + **1** else:  
 return  
  
 def display(self):  
 if self.size == **0**:  
 print("Display : Stack is Empty")  
 return  
 else:  
 temp = self.head  
 while temp:  
 self.queue.append(temp.data)  
 temp = temp.next  
  
 self.queue.reverse()  
 print("\nThe Stack is : {}".format(self.queue))  
  
 def reverse(self):  
 self.queue.reverse()  
 print("Reversed Stack : {}".format(self.queue))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 stack = Stack()  
 stack.push()  
 stack.display()  
 stack.reverse()

****

**8. Write a Python program to reverse a queue.**

class Node:  
 def \_\_init\_\_(self**,** data):  
 self.data = data  
 self.next = None  
  
  
class Queue:  
 def \_\_init\_\_(self):  
 self.front = None  
 self.rear = None  
 self.list = []  
  
 def enqueue(self):  
 while True:  
 x = int(input("Press 1 to Insert or any other Key to Exit : "))  
 if x == **1**:  
 x = int(input("Enter the Element to Insert : "))  
 node = Node(x)  
  
 if self.rear is None:  
 self.rear = node  
 self.front = node  
 continue  
 else:  
 self.rear.next = node  
 self.rear = node  
 else:  
 return  
  
 def display(self):  
 if self.front is None:  
 print("Display : Queue is Empty")  
 return  
 else:  
 temp = self.front  
 while temp:  
 self.list.append(temp.data)  
 temp = temp.next  
 print("Queue : {}".format(self.list))  
  
 def reverse(self):  
 self.list.reverse()  
 print("Reversed Queue : {}".format(self.list))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 queue = Queue()  
 queue.enqueue()  
 queue.display()  
 queue.reverse()

****

**Thankyou!!**