Queue

&

Circular Queue

Q1) implement queue in python

Program

```
class Queue:
  def __init__(self, capacity):
     self.front = self.size = 0
     self.rear = capacity - 1
     self.Q = [None] * capacity
     self.capacity = capacity

  def isFull(self):
     return self.size == self.capacity

  def isEmpty(self):
     return self.size == 0

  def EnQueue(self, item):
     if self.isFull():
        print("Full")
        return

     self.rear = (self.rear + 1) % (self.capacity)
     self.Q[self.rear] = item
     self.size = self.size + 1
     print("%s enqueue to queue" %str(item))

  def DeQueue(self):
     if self.isEmpty():
        return "Empty"

     print("%s dequeued from queue" %str(self.Q[self.front]))
     self.front = (self.front + 1) % (self.capacity)
     self.size = self.size - 1

  def que_front(self):
     if self.isEmpty():
        print("The Queue is empty")


     print("Front item is ", self.Q[self.front])
```

```
    def que_rear(self):
        if self.isEmpty():
            print("Queue is Empty")

        print("The rear item is ", self.Q[self.rear])
queue = Queue(30)
queue.EnQueue(10)
queue.EnQueue(20)
queue.EnQueue(30)
queue.EnQueue(40)
queue.EnQueue(50)
queue.que_front()
print()
queue.DeQueue()
queue.que_front()
queue.que_rear()
print()
queue.DeQueue()
queue.que_front()
queue.que_rear()
```

Output

```
10 enqueue to queue
20 enqueue to queue
30 enqueue to queue
40 enqueue to queue
50 enqueue to queue
Front item is  10

10 dequeued from queue
Front item is  20
The rear item is  50

20 dequeued from queue
Front item is  30
The rear item is  50
```

Q3) Circular Queue Implementations in Python

Program

```
# Circular Queue implementation in Python
class MyCircularQueue():

    def __init__(self, k):
        self.k = k
        self.queue = [None] * k
        self.head = self.tail = -1

    # Insert an element into the circular queue
    def enqueue(self, data):

        if ((self.tail + 1) % self.k == self.head):
            print("The circular queue is full\n")

        elif (self.head == -1):
            self.head = 0
            self.tail = 0
            self.queue[self.tail] = data
        else:
            self.tail = (self.tail + 1) % self.k
            self.queue[self.tail] = data

    # Delete an element from the circular queue
    def dequeue(self):
        if (self.head == -1):
            print("The circular queue is empty\n")

        elif (self.head == self.tail):
            temp = self.queue[self.head]
            self.head = -1
            self.tail = -1
            return temp
        else:
```

```
            temp = self.queue[self.head]
            self.head = (self.head + 1) % self.k
            return temp
        def printCQueue(self):
            if(self.head == -1):
                print("No element in the circular queue")

            elif (self.tail >= self.head):
                for i in range(self.head, self.tail + 1):
                    print(self.queue[i], end=" ")
                print()
            else:
                for i in range(self.head, self.k):
                    print(self.queue[i], end=" ")
                for i in range(0, self.tail + 1):
                    print(self.queue[i], end=" ")
                print()

    # Your MyCircularQueue object will be instantiated and called as such:
    obj = MyCircularQueue(5)
    obj.enqueue(1)
    obj.enqueue(2)
    obj.enqueue(3)
    obj.enqueue(4)
    obj.enqueue(5)
    print("Initial queue")
    obj.printCQueue()
    obj.dequeue()
    print("After removing an element from the queue")
    obj.printCQueue()
```

Output

```
Initial queue
1 2 3 4 5
After removing an element from the queue
2 3 4 5
```