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
```

Q2) Queue implementation using array

Algorithm

1. Declare a list and an integer MaxSize, denoting a virtual maximum size of the Queue
2. Head and Tail are initially set to 0
3. Size method
    1. Calculates the number of elements in the queue -> Size = Tail - Head
4. Reset method:
    1. Resets Tail and Head to 0
    2. Creates a new Queue (initializes queue to a new list)
5. Enqueue operation:
    1. Check if Size is less than the MaxSize:
        1. If yes, append data to Queue and then increment Tail by 1
        2. If no, print Queue full message
6. Dequeue operation:
    1. Check if Size is greater than 0:
        1. If yes, pop the first element from the list and increment Head by 1
        2. If no:
            1. Call Reset method
            2. Print Queue empty message

Program

```
class Queue:

    #Constructor
    def __init__(self):
        self.queue = list()
        self.maxSize = 8
        self.head = 0
        self.tail = 0

    #Adding elements
    def enqueue(self,data):
        #Checking if the queue is full
        if self.size() >= self.maxSize:
            return ("Queue Full")
        self.queue.append(data)
        self.tail += 1
        return True

    #Deleting elements
    def dequeue(self):
        #Checking if the queue is empty
        if self.size() <= 0:
            self.resetQueue()
            return ("Queue Empty")
        data = self.queue[self.head]
        self.head+=1
        return data

    #Calculate size
    def size(self):
        return self.tail - self.head

    #Reset queue
    def resetQueue(self):
        self.tail = 0
        self.head = 0
        self.queue = list()

q = Queue()
print(q.enqueue(1)) #prints True
print(q.enqueue(2)) #prints True
print(q.enqueue(3)) #prints True
print(q.enqueue(4)) #prints True
print(q.enqueue(5)) #prints True
print(q.enqueue(6)) #prints True
print(q.enqueue(7)) #prints True
print(q.enqueue(8)) #prints True
print(q.enqueue(9)) #prints Queue Full!
print(q.size())#prints 8
print(q.dequeue()) #prints 8
print(q.dequeue()) #prints 7
print(q.dequeue()) #prints 6
print(q.dequeue()) #prints 5
print(q.dequeue()) #prints 4
print(q.dequeue()) #prints 3
print(q.dequeue()) #prints 2
print(q.dequeue()) #prints 1
print(q.dequeue()) #prints Queue Empty
#Queue is reset here
print(q.enqueue(1)) #prints True
print(q.enqueue(2)) #prints True
print(q.enqueue(3)) #prints True
print(q.enqueue(4)) #prints True
```

Output

```
True
True
True
True
True
True
True
True
Queue Full
8
1
2
3
4
5
6
7
8
Queue Empty
True
True
True
True
```

Note: Element 9 was not added to the Queue and hence the size of the Queue remains 8

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
```

Q4) Circular Queue

Algorithm

The following steps can be seen as a flow chart to the operation of the Circular Queue:

1. Initialize the queue, size of the queue (maxSize), head and tail pointers
2. Enqueue:
    1. Check if the number of elements (size) is equal to the size of the queue (maxSize):
        1. If yes, throw error message "Queue Full!"
        2. If no, append the new element and increment the tail pointer
3. Dequeue:
    1. Check if the number of elements (size) is equal to 0:
        1. If yes, throw error message "Queue Empty!"
        2. If no, increment head pointer
4. Size:
    1. If tail>=head, size = tail - head
    2. if head>tail, size = maxSize - (head - tail)

Note: The range for the head and tail pointer should be between 0 and maxSize - 1, hence we are using the logic that if we divide x by 5, then the remainder can never be greater than 5. In other words, it should be between 0 and 4. So apply this logic to the formulae tail = (tail+1)%maxSize and head = (head+1)%maxSize. Observe that this is helps us to avoid reinitializing tail and head to 0 when the queue becomes full.

Program

```
class CircularQueue:

  #Constructor
  def __init__(self):
      self.queue = list()
      self.head = 0
      self.tail = 0
      self.maxSize = 8
```

```
    #Adding elements to the queue
    def enqueue(self,data):
        if self.size() == self.maxSize-1:
            return ("Queue Full!")
        self.queue.append(data)
        self.tail = (self.tail + 1) % self.maxSize
        return True

    #Removing elements from the queue
    def dequeue(self):
        if self.size()==0:
            return ("Queue Empty!")
        data = self.queue[self.head]
        self.head = (self.head + 1) % self.maxSize
        return data

    #Calculating the size of the queue
    def size(self):
        if self.tail>=self.head:
            return (self.tail-self.head)
        return (self.maxSize - (self.head-self.tail))
q = CircularQueue()
print(q.enqueue(1))
print(q.enqueue(2))
print(q.enqueue(3))
print(q.enqueue(4))
print(q.enqueue(5))
print(q.enqueue(6))
print(q.enqueue(7))
print(q.enqueue(8))
print(q.enqueue(9))
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
```

Output

```
True
True
True
True
True
True
True
Queue Full!
Queue Full!
1
2
3
4
5
6
7
Queue Empty!
Queue Empty!
```