Queue Data Structure

Introduction:

==> First-In-First-Out (FIFO)
==> front and rear
==> front is always used access/delete
==> rear is always used insert

Operations on Queue:

The following are the very common operations that can be performed on Q

1) enqueue or insert
2) dequeue or delete
3) getFront element
4) getRear element
5) size
6) isempty
7) display

Applications Queue

1) Single Resource and Multiple Consumers (Queue)
2) Operating System
3) Printing Queue
4) Computer Networks
etc

Queue implementations in python

1) using list
2) collections.deque
3) queue.Queue
4) implementation of queue by using list
5) implementation of queue by using linked list

1) using list

```
q = []

print(len(q)==0) #True
q.append(10)
q.append(20)
q.append(30)
print(q) #[10, 20, 30]
print(len(q)==0) #False
print(q[0]) #front --> 10
print(q[-1]) #rear --> 30
print(q.pop(0)) #first
print(q) #[20,30]
print(q.pop(0)) #first
print(q) #[30]
```

```
C:\test>py test.py
True
[10, 20, 30]
False
10
30
10
[20, 30]
```

```
20
[30]

2) collections.deque
--------------------
from collections import deque
q = deque()

print(len(q)==0) #True
q.append(111)
q.append(222)
q.append(333)
print(len(q)==0) #False
print(q) #[111, 222, 333]
print(q[0]) #111
print(q[-1]) #333
print(q.popleft()) #111
print(q) #[222, 333]
print(len(q)) #2

C:\test>py test.py
True
False
deque([111, 222, 333])
111
333
111
deque([222, 333])
2

3) queue.Queue
--------------
from queue import Queue
q = Queue(4)
print(q.qsize()) #0
print(q.empty()) #True
q.put(10)
q.put(20)
q.put(30)
q.put(40)
print(q.qsize()) #4
print(q.empty()) #False
print(q.get()) #10

C:\test>py test.py
0
True
4
False
10

4) implementation of queue by using list
----------------------------------------
class queue:

      #creation of q
      def __init__(self,cap):
            self.que = [None]*cap
            self.cap = cap
            self.count = 0

      #is full
      def isfull(self):
            return self.count == self.cap
```

```
      #is empty
      def isempty(self):
            return self.count == 0

      #size
      def size(self):
            return self.count

      #disply
      def display(self):
            if self.isempty():
                  print("q is empty")
                  return
            for i in range(self.count):
                  print(self.que[i],end=" ")
            print()

      #insert operation
      def insert(self,data):
            if self.isfull():
                  print("q is full")
                  return
            self.que[self.count] = data
            self.count = self.count + 1

      #delete O(n)
      def delete(self):
            if self.isempty():
                  print("q is empty")
                  return
            res = self.que[0]
            for i in range(self.count-1):
                  self.que[i] = self.que[i+1]
            self.count=self.count-1
            return res


q = queue(4)
print(q.size()) #0
print(q.isempty()) #True
print(q.isfull()) #False
q.insert(10)
q.insert(20)
q.insert(30)
q.insert(40)
q.insert(50)
q.display()
print(q.size()) #4
print(q.delete())
q.display()
print(q.size()) #3
q.insert(60)
q.display()

C:\test>py test.py
0
True
False
q is full
10 20 30 40
4
10
20 30 40
```

```
3
20 30 40 60

Note: we can reduce tc of delete operation by using circular linked list

5) implementation of queue by using linked list
-----------------------------------------------
class Node:
      def __init__(self,data):
            self.data = data
            self.next = None

class MyQueue:

      #construction
      def __init__(self):
            self.front = None
            self.rear = None
            self.count = 0

      #size
      def size(self):
            return self.count

      #is empty
      def isempty(self):
            return self.count==0

      #get front
      def getfront(self):
            return self.front.data

      #get rear
      def getrear(self):
            return self.rear.data

      #insert
      def insert(self,data):
            temp = Node(data)
            if self.rear == None:
                  self.front = temp
            else:
                  self.rear.next = temp
            self.rear = temp
            self.count = self.count + 1

      #display
      def display(self):
            if self.isempty():
                  print("q is empty")
                  return
            currNode = self.front
            while currNode!=None:
                  print(currNode.data,end=" ")
                  currNode = currNode.next
            print()

      #delete
      def delete(self):
            if self.front==None:
                  return None
            res = self.front.data
            self.front = self.front.next
            if self.front == None:
```

```
                self.rear = None
            self.count = self.count - 1
            return res

q = MyQueue()
print(q.size()) #0
print(q.isempty()) #True
q.insert(10)
q.insert(20)
q.insert(30)
q.display()
print(q.size()) #3
print(q.isempty()) #False
q.display()
print(q.delete())
q.display()

C:\test>py test.py
0
True
10 20 30
3
False
10 20 30
10
20 30
```