Problem # 622: Design Circular Queue

<https://leetcode.com/problems/design-circular-queue/>

My Solution:

class MyCircularQueue:

def \_\_init\_\_(self, k: int):

"""

Initialize your data structure here. Set the size of the queue to be k.

"""

self.maxsize = k

self.size = 0

self.head = -1

self.tail = -1

self.queue = [None] \* k

def enQueue(self, value: int) -> bool:

"""

Insert an element into the circular queue. Return true if the operation is successful.

"""

if self.isFull():

return(False)

self.tail = (self.tail + 1) % self.maxsize

self.queue[self.tail] = value

self.size += 1

if self.size == 1:

self.head = 0

return(True)

def deQueue(self) -> bool:

"""

Delete an element from the circular queue. Return true if the operation is successful.

"""

if self.isEmpty():

return(False)

self.head = (self.head + 1) % self.maxsize

self.size -= 1

if self.isEmpty():

self.head = -1

self.tail = -1

return(True)

def Front(self) -> int:

"""

Get the front item from the queue.

"""

if self.isEmpty():

return(-1)

return(self.queue[self.head])

def Rear(self) -> int:

"""

Get the last item from the queue.

"""

if self.isEmpty():

return(-1)

return(self.queue[self.tail])

def isEmpty(self) -> bool:

"""

Checks whether the circular queue is empty or not.

"""

return(self.size == 0)

def isFull(self) -> bool:

"""

Checks whether the circular queue is full or not.

"""

return(self.size == self.maxsize)

# Your MyCircularQueue object will be instantiated and called as such:

# obj = MyCircularQueue(k)

# param\_1 = obj.enQueue(value)

# param\_2 = obj.deQueue()

# param\_3 = obj.Front()

# param\_4 = obj.Rear()

# param\_5 = obj.isEmpty()

# param\_6 = obj.isFull()