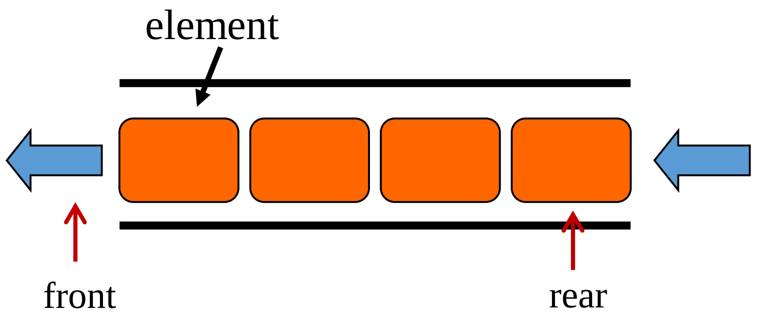
OO Implementation: Circular Queue

- Create a Circular Queue (CQ) Class and a Person Class
- We will be inserting the Person object into the CQ object
- The CQ class and Person class frame will be provided
- You must complete all the functions and check if your program works using the main function
- Each team will be required to submit the following:
  - Source code (python file *.py)
  - A power point (ppt) report explaining your implementation and test results
- Scoring:
  - How well your program works
  - How intuitive, informative, and visually expressed is your report

What is a Queue?

- FIFO(First In First Out) Structure
- Elements are added from the rear and removed from the front of the queue.

- Operations on Queue
  - init(maxSize)
  - enqueue(e)
  - dequeue()
  - multi_dequeue(count)
  - peek()
  - is_empty()
  - is_full()



Queue Functions

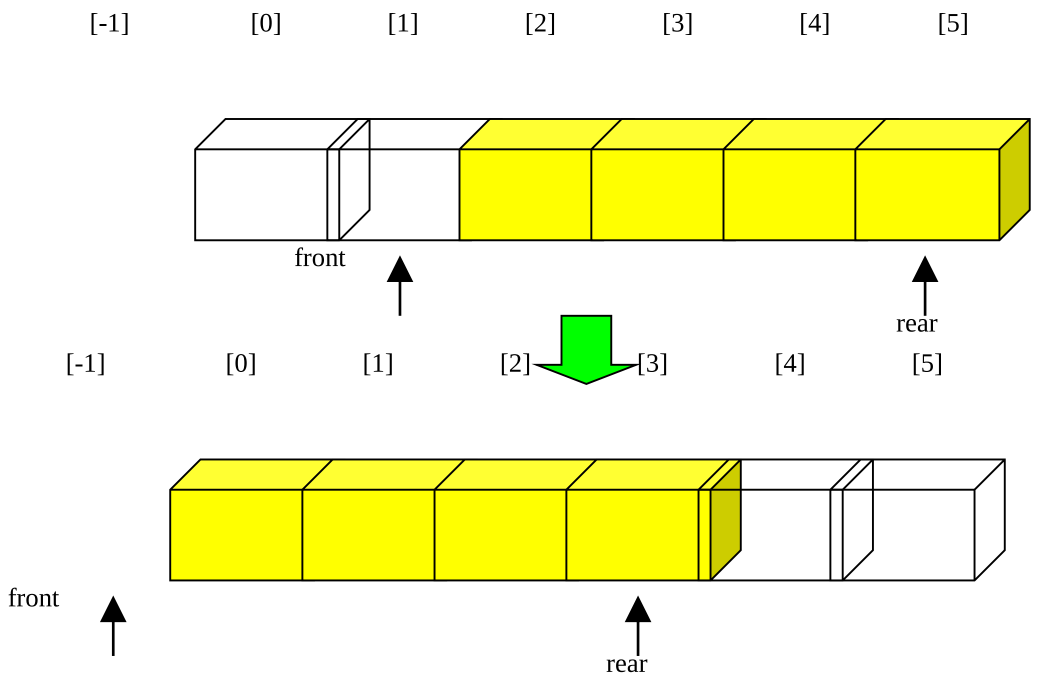
- init(maxSize) : create (initialize) the queue
- enqueue(e) : add an element at the end of the queue
  - Check if the queue is full,
    - move the rear one position
    - insert new element at the rear position

- dequeue() : remove first element from the front and return it
  - Check if the queue is empty,
    - move the front one position
    - return the element at the location of the front value

- multi_dequeue(count) : remove multiple elements from the queue and return it as a list
- peek() : return the first element in the queue
- is_empty() : check if the queue is empty
- is_full() : check if the queue is full

Weakness of Ordinary Queue

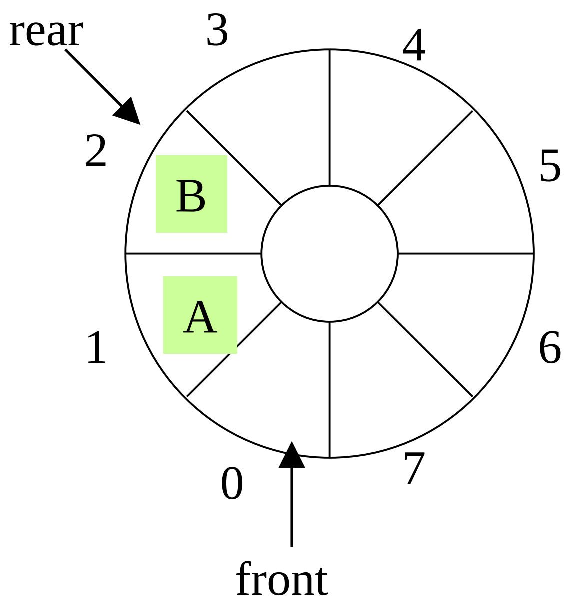
- After a insert/remove occurs, all the elements in the queue needs to be moved forward.
  - Moving elements in the queue is very expensive
  - Q(MAX_QUEUE_SIZE)



How CQ Works [1/2]

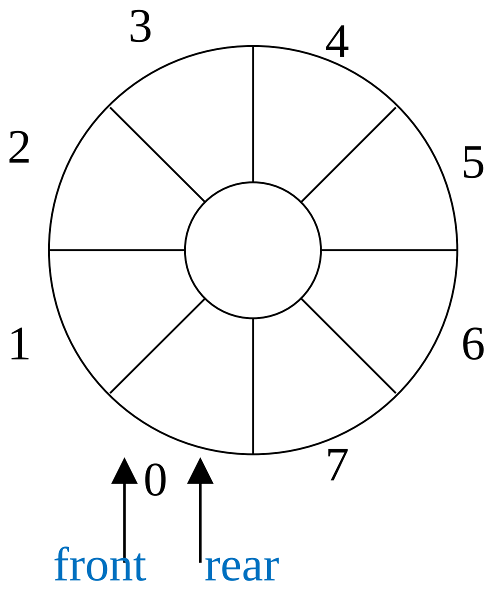
- Use a list array as a circle and implement the circular queue
- Manage the Front and Rear position using two variables

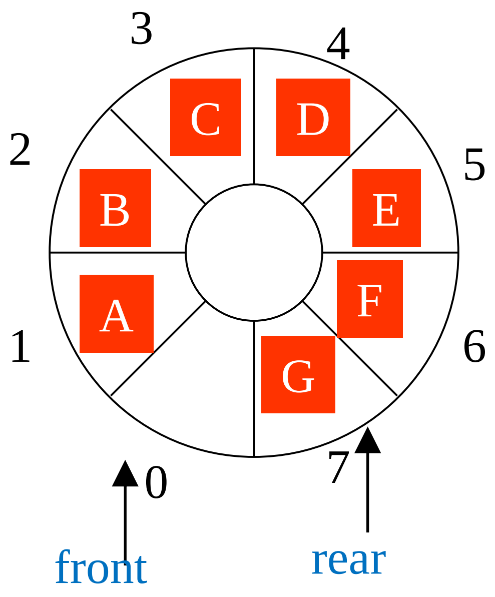
```
if (rear == MAX_QUEUE_SIZE-1)     rear=0;
else rear++;        // (rear+1) % MAX_QUEUE_SIZE와 동등
```



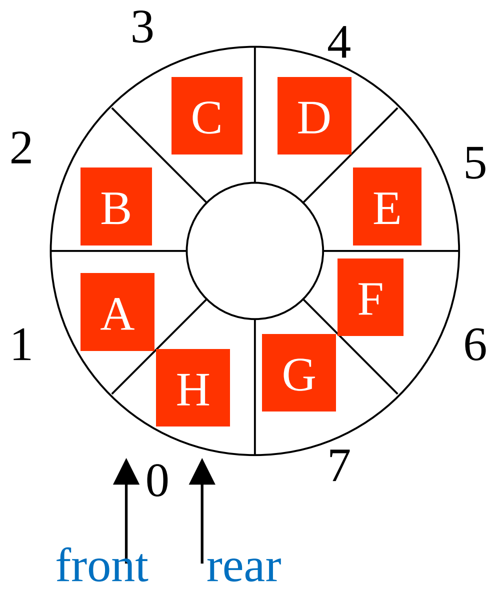
How CQ Works [2/2]

- is_empty(q) : front == rear
- is_full(q) : front % M == (rear+1) % M
  - M is the size of the circular queue





(b) Full queue



(c) error

Person Class

```
class Person:

    name = ""
    age = 0

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __add__(self, other):
        [redacted]

    def __str__(self):
        [redacted]

    def __gt__(self, other):
        [redacted]


    def __lt__(self, other):
        [redacted]


    def __repr__(self):
        [redacted]
```

CG Class [1/3]

```
class CircularQueue:

    M       = 0
    front   = 0
    rear    = 0
    queue   = [  ]

    def __init__(self, maxSize):
        self.M = maxSize
        self.queue = [None] * maxSize

    def enqueue(self, element):
```

CG Class [2/3]

```
def dequeue(self):
```

```
def multi_dequeue(self, count):
```

CG Class [3/3]

```
def peek(self):
```

[redacted]

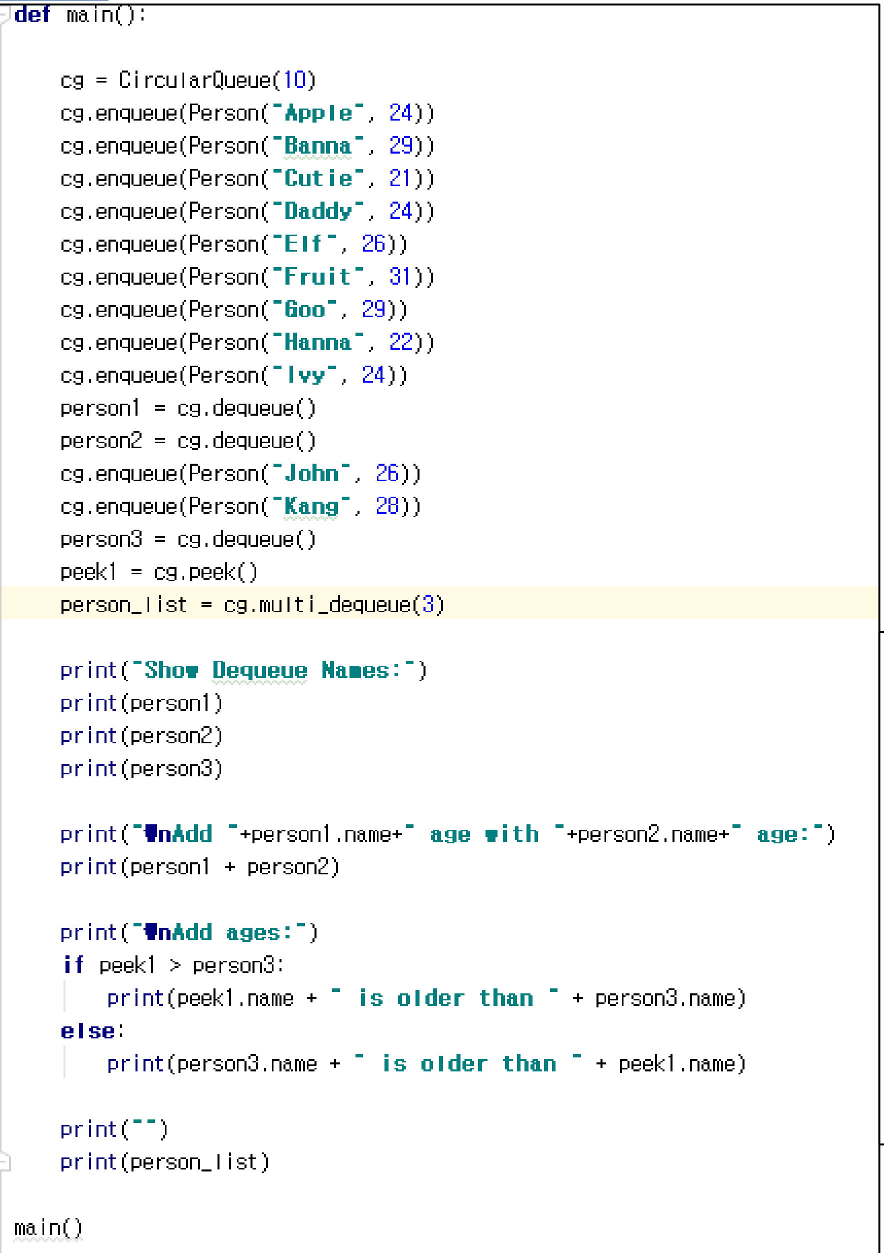
```
def is_empty(self):
```

[redacted]

```
def is_full(self):
```

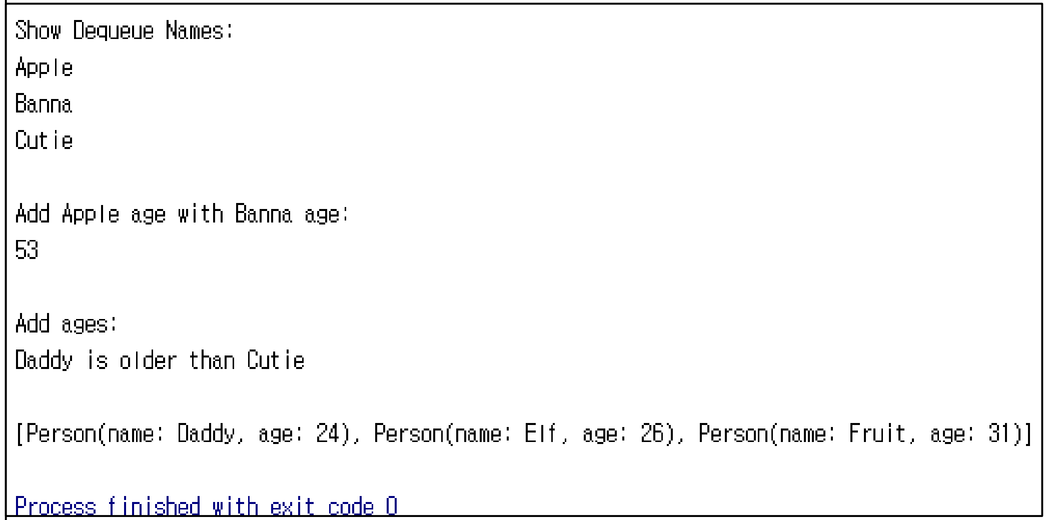
[redacted]

Output [1/2]



```
def main():

    cg = CircularQueue(10)
    cg.enqueue(Person("Apple", 24))
    cg.enqueue(Person("Banna", 29))
    cg.enqueue(Person("Cutie", 21))
    cg.enqueue(Person("Daddy", 24))
    cg.enqueue(Person("Elf", 26))
    cg.enqueue(Person("Fruit", 31))
    cg.enqueue(Person("Goo", 29))
    cg.enqueue(Person("Hanna", 22))
    cg.enqueue(Person("Ivy", 24))
    person1 = cg.dequeue()
    person2 = cg.dequeue()
    cg.enqueue(Person("John", 26))
    cg.enqueue(Person("Kang", 28))
    person3 = cg.dequeue()
    peek1 = cg.peek()
    person_list = cg.multi_dequeue(3)

    print("Show Dequeue Names:")
    print(person1)
    print(person2)
    print(person3)

    print("₩nAdd "+person1.name+" age with "+person2.name+" age:")
    print(person1 + person2)

    print("₩nAdd ages:")
    if peek1 > person3:
        print(peek1.name + " is older than " + person3.name)
    else:
        print(person3.name + " is older than " + peek1.name)

    print("")
    print(person_list)

main()
```

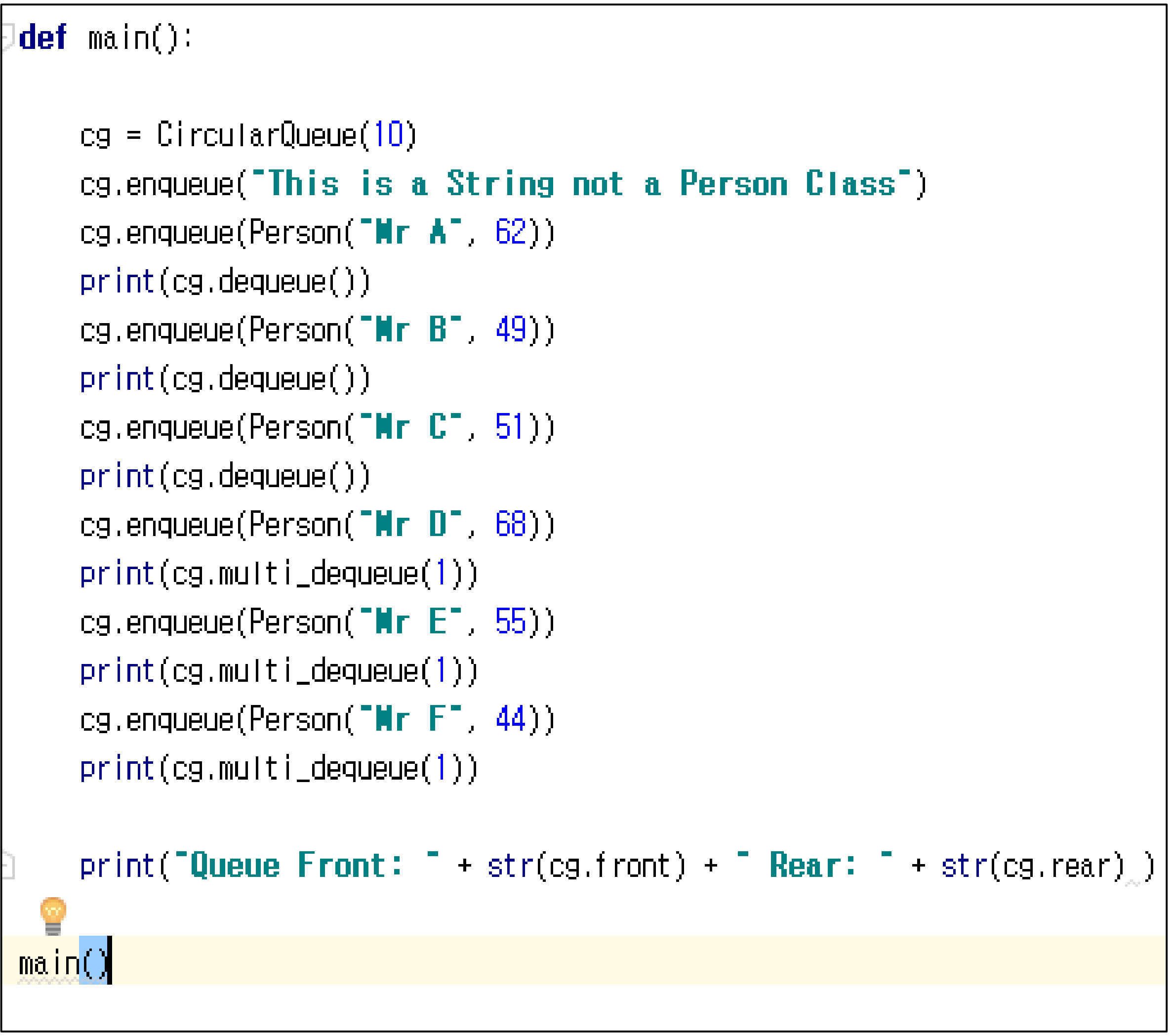
Results:



```
Show Dequeue Names:
Apple
Banna
Cutie

Add Apple age with Banna age:
53

Add ages:
Daddy is older than Cutie

[Person(name: Daddy, age: 24), Person(name: Elf, age: 26), Person(name: Fruit, age: 31)]

Process finished with exit code 0
```

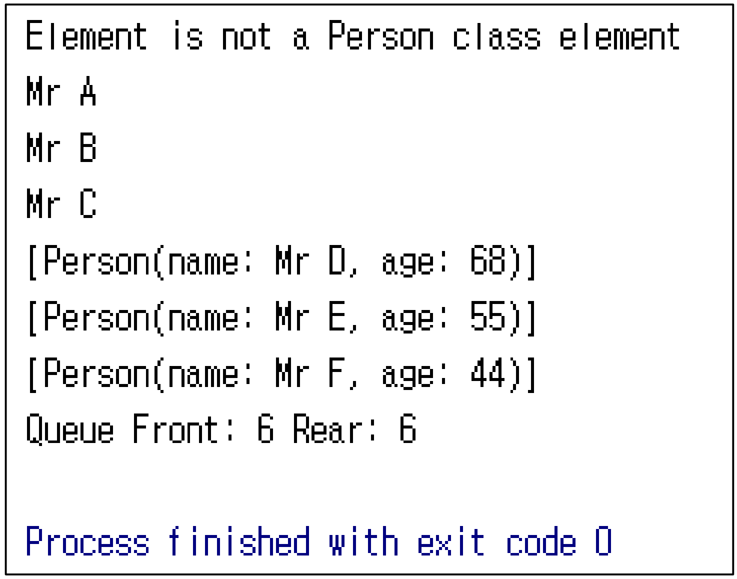
Output [2/2]

Main Function:



```python
def main():

    cg = CircularQueue(10)
    cg.enqueue("This is a String not a Person Class")
    cg.enqueue(Person("Mr A", 62))
    print(cg.dequeue())
    cg.enqueue(Person("Mr B", 49))
    print(cg.dequeue())
    cg.enqueue(Person("Mr C", 51))
    print(cg.dequeue())
    cg.enqueue(Person("Mr D", 68))
    print(cg.multi_dequeue(1))
    cg.enqueue(Person("Mr E", 55))
    print(cg.multi_dequeue(1))
    cg.enqueue(Person("Mr F", 44))
    print(cg.multi_dequeue(1))

    print("Queue Front: " + str(cg.front) + " Rear: " + str(cg.rear) )

main()
```

Results:



```
Element is not a Person class element
Mr A
Mr B
Mr C
[Person(name: Mr D, age: 68)]
[Person(name: Mr E, age: 55)]
[Person(name: Mr F, age: 44)]
Queue Front: 6 Rear: 6


Process finished with exit code 0
```

Rules & Guide

- Cannot use any list functions (len, append, split [:])
- When enqueue, you must type check the element, if it is not an Person class object then your must print “Element is not a Person class element”
- When dequeue and the queue is empty, you must print “Queue is empty”
- When queue is full, you must print “Queue is full!”