Queues

- Collection of objects that are inserted and removed according to the first-in, first-out (FIFO)
- Elements can be inserted at anytime, but only the element that has been in the queue the longest can be next removed

Examples:

1) printers
2) Web server responding to requests.
3) Scheduling

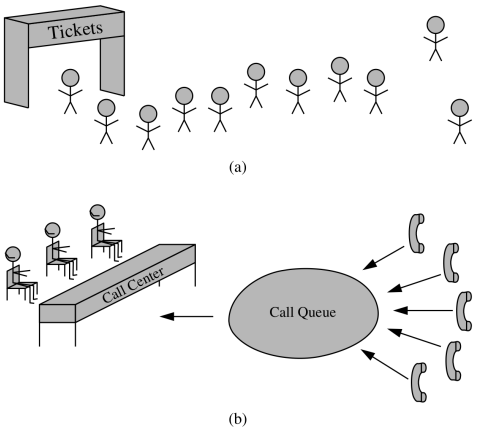
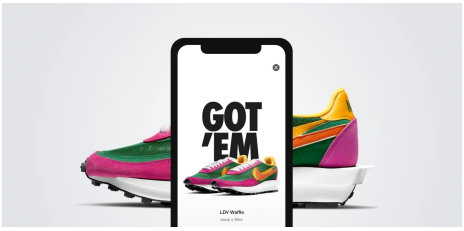


Figure 6.4: Real-world examples of a first-in, first-out queue. (a) People waiting in line to purchase tickets; (b) phone calls being routed to a customer service center.



Nike SNKR App

~ for some drops

The Queue Abstract Data Type

~ The Queue abstract data type defines a collection that keeps objects in sequence, where element access and deletion are restricted to the first elements in the queue, and element insertion is restricted to the back of the sequence.

The queue abstract data type (ADT) supports two methods for a queue Q:

Operations:

Q.enqueue(e): Add element e to the back of queue Q.

Q.dequeue(): Remove and return the first element from queue Q; an error occurs if the queue is empty.

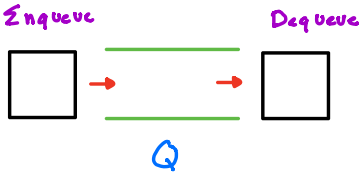
Other supporting Methods:

Q.first(): Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.

Q.is_empty(): Return True if queue Q does not contain any elements.

len(Q): Return the number of elements in queue Q; in Python, we implement this with the special method __len__.

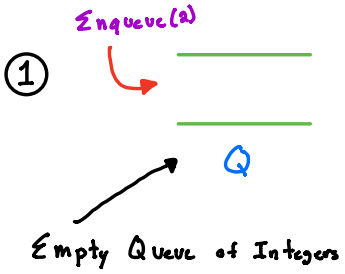
Constant time or O(1)



~ logically, a queue can be shown as a figure or container open from two sides.

~ An element can be inserted or Enqueued from one side and an element can be removed or de-queued from other side.

~ In queue, insertion and removal should happen from different sides.



Logic/Code

Enqueue (2)

Enqueue (5)

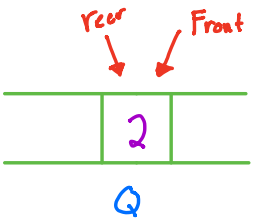
Enqueue (3)

Dequeue() → 2 return

front() → 5

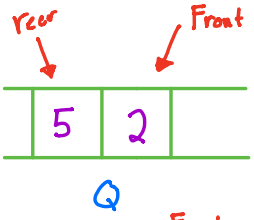
Is Empty → false

②



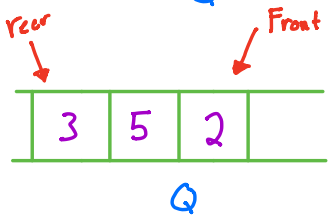
~ After this Enqueue, we have one element in the queue

③



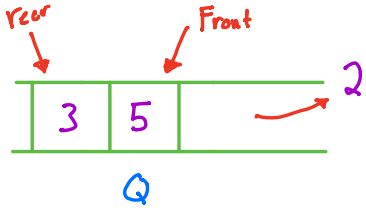
~ Insert another integer,

④



~ Insert another integer,

⑤

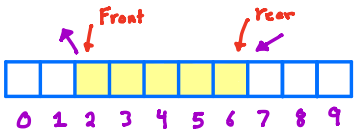


~ 2 will head out of the queue

How do we implement this in Python?

- Circular Array
- linked lists

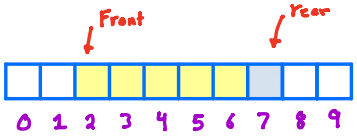
Array Implementation



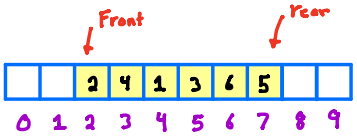
~ any order of front and rear

~ But element must be added from rear and removed from front

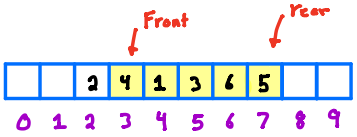
~ At any stage a segment of the array from an index marked as front till an index marked as rear is my Queue and the rest of positions in the array are free space, that can be used to expand the Queue.



~ add a new element towards the rear end, we can write the element to be inserted



~ Inserting the number 5

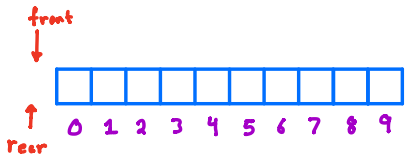


~ Removing an element from the Queue

~ By incrementing front we have discarded index 2 from the Queue

Writing Pseudo Code:

```
IsEmpty():
  if front == -1 and rear==-1:
        return True
  else:
     return False
```




```
Enqueue(x)

  if  IsFull():
        return

  else if IsEmpty():

        front, rear = 0
  else:
      rear = rear +1

  A[rear] = x
```

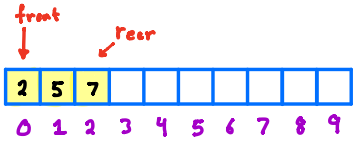
Lets Equeue some elements:

```
else:
    rear = rear + 1
    A[rear] = x
```

Enqueue (2)

Enqueue (5)

Enqueue (7)



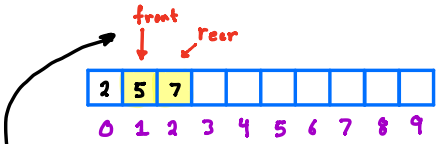
Dequeue Pseudo Code:

```
Dequeue()

  if IsEmpty():
     return

  else if front == rear:
      front, rear = A[-1]        ~ Q is empty

  else:
     front = front + 1
```

Enqueue (2)

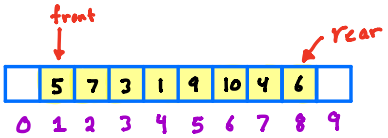
Enqueue (5)

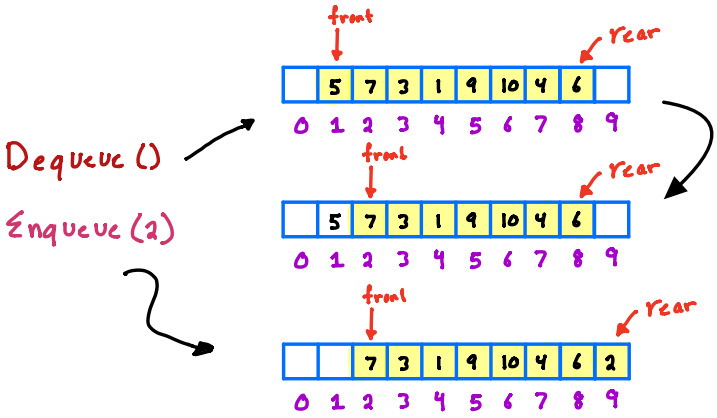
Enqueue (7)



Dequeue () → It will increment front

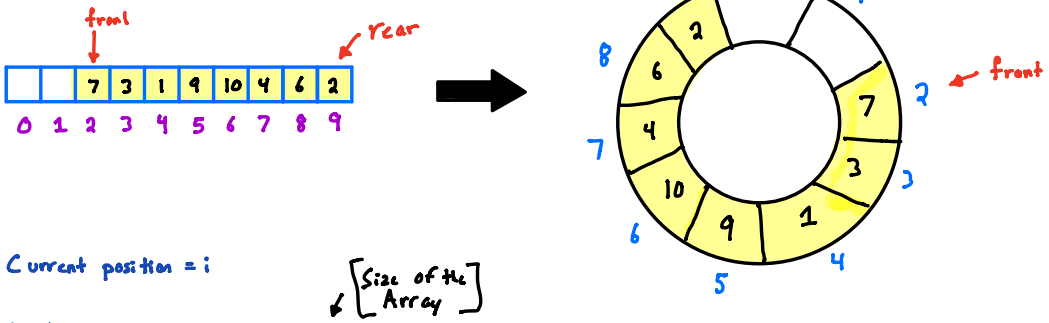
Enqueue (3)
Enqueue (1)
Enqueue (9)
Enqueue (10)
Enqueue (4)
Enqueue (6)





At this point, we cannot Enqueue an element anymore

Circular Array



Current position = i

Next position = (i+1) % N ← [Size of the Array]

previous position = (i+N-1) % N

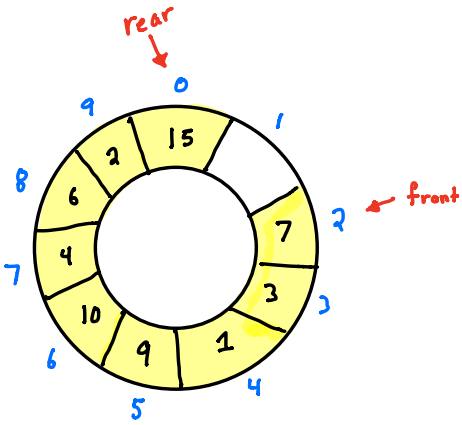
i = N-1

Next = N % N = 0

Modify Sudo Code:

```
IsEmpty():
  if front == -1 and rear==1:
        return True
  else:
     return False
```

```
Enqueue(x)

   if (rear+1)%N == A[Front]:
        return

   else if IsEmpty():

         front, rear = 0
   else:
      rear = (rear+1) %N

   A[rear] = x
```

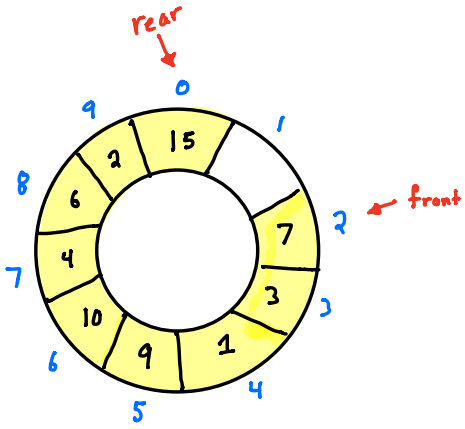


Enqueue(15)

```
Enqueue(x)

if (rear+1)%N == A[Front]:
     return

else if IsEmpty():

      front, rear = 0
else:
   rear = (rear+1) %N

A[rear] = x
```

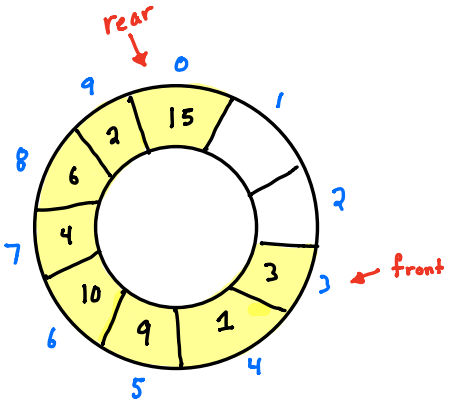
rear = (9+1) % 10 = 0

```
Dequeue()

if IsEmpty():
    return

else if front == rear:
    front, rear = A[-1]          ~ Q is empty

else:
    front = (front + 1) % N
```




```
Dequeue()

if IsEmpty():
    return

else if front == rear:
    front, rear = A[-1]          ~ Q is empty

else:
    front = (front + 1) % N
```

Dequeue()




```
front()

return A[front]
```