Queue

First-In-First-Out (FIFO)

Dr. Nadia M. Mohammed

Queue

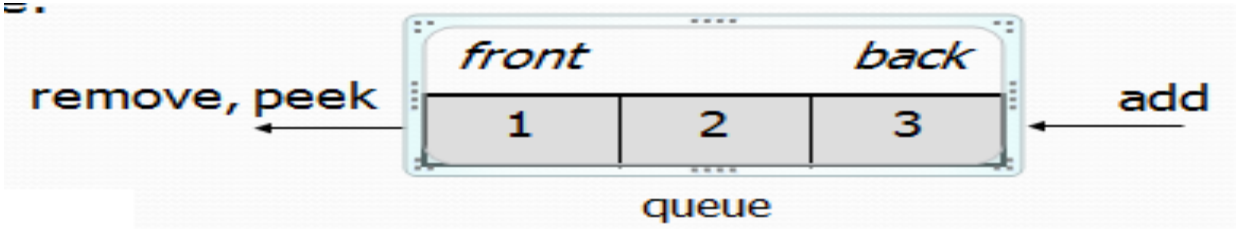
Queue: Retrieves elements in the order they were added. First-In, First-Out ("FIFO")

- Queue is a collection of data that is accessed in a first-in-first-out (FIFO) manner .
- Basic queue operations:

  add (enqueue): Add an element to the back.

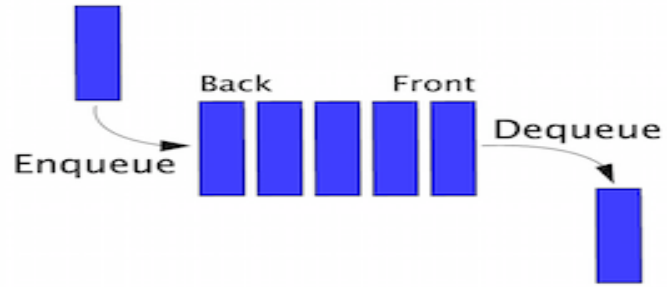
  remove (dequeue): Remove the front element.

  peek: Examine the front element.



Basic Operations in queue:

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues:



- **enqueue()**

  add an item to the queue.

- **dequeue()**

  remove an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are:

- **peek()** – Gets the element at the front of the queue without removing it.
- **isfull()** – Checks if the queue is full.
- **isempty()** – Checks if the queue is empty.

In queue, we always dequeue data (in Deletion), pointed by **front** pointer and while enqueing data (in Addition) we take help of **rear** pointer.

- Let's first learn about supportive functions of a queue:

- peek() : This function helps to see the data at the **front** of the queue. The algorithm of peek() function is as follows:

- **Algorithm**

begin procedure **peek**

  return queue[front]

end procedure

- **isfull()**: As we are using single dimension array to implement queue, we just check for the rear pointer to reach at MAXSIZE to determine that the queue is full.

- **Algorithm**

begin procedure **isfull**

```
    if rear equals to MAXSIZE
    return true
  else
    return false
   endif
```

end procedure

- **isempty()**: Algorithm of isempty() function

- **Algorithm**

begin procedure **isempty**
  if front is less than MIN OR front is greater than rear
    return true
  else return false
  endif
End procedure

- If the value of **front** is less than MIN or 0, it tells that the queue is not yet initialized, hence empty.

Enqueue Operation

- **Enqueue**: Queues maintain two data pointers, **front** and **rear**.
- The following steps should be taken to enqueue (insert) data into a queue:

Step 1 – Check if the queue is full.

Step 2 – If the queue is full, produce overflow error and exit.

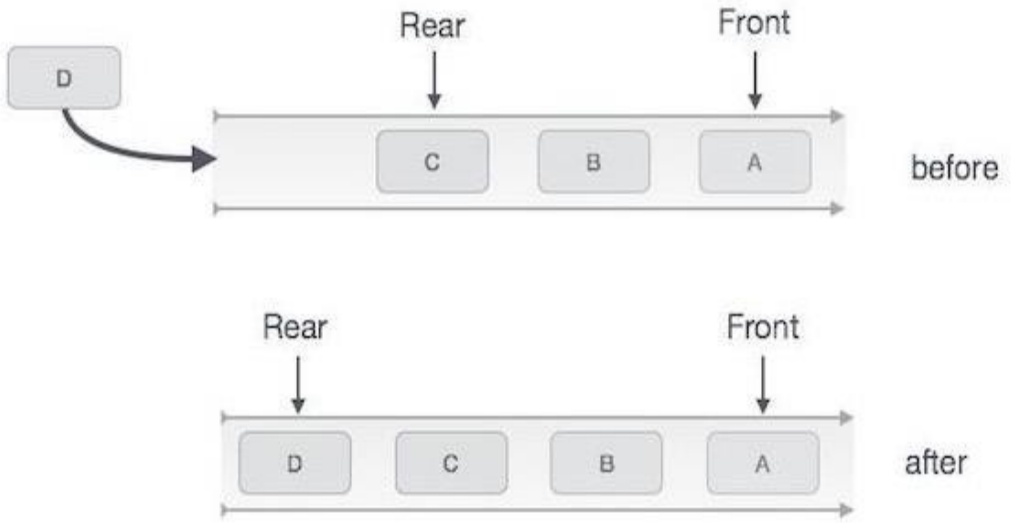
Step 3 – If the queue is not full, increment **rear** pointer to point the next empty space.

Step 4 – Add data element to the queue location, where the rear is pointing.

Step 5 – return success.

Enqueue

```
# The enqueue method add an item to the Queue
def enqueue(self, item):
    if len(self.queue) == self.size:
        print("Queue is full!!")
    else:
        self.queue.insert(0, item)
        print('Queue size is: ', len(self.queue))
```



Dequeue Operation

- **Dequeue** : Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation:

Step 1 − Check if the queue is empty.

Step 2 − If the queue is empty, produce underflow error and exit.

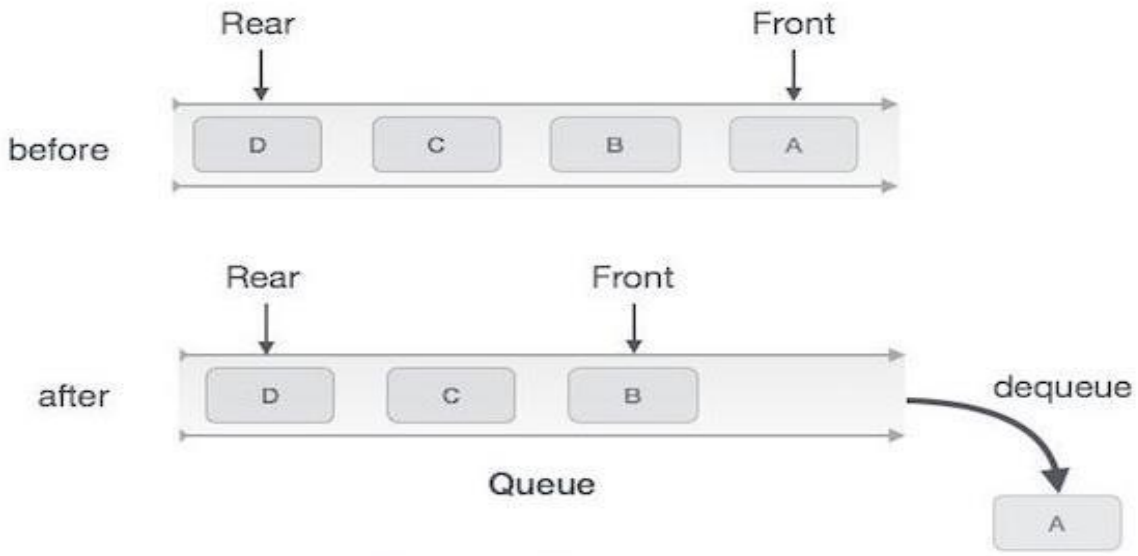
Step 3 − If the queue is not empty, access the data where front is pointing.

Step 4 − Increment front pointer to point to the next available data element.

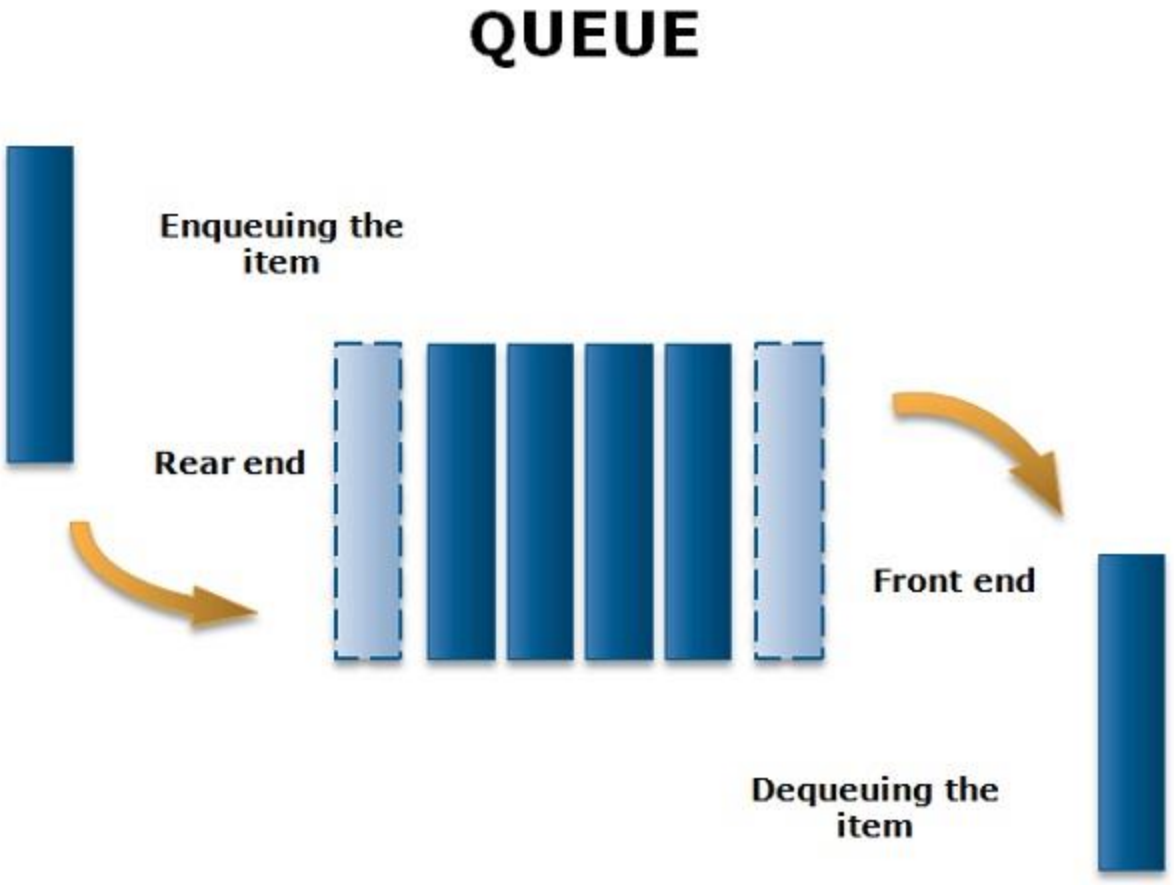
Step 5 − Return success.

Dequeue

```
    # The dequeue method delete an item from the
queue
    def dequeue(self):
        if self.queue == []:
            print("Queue is empty!")
            data = -1
        else:
            data = self.queue.pop()
        return data
```



Enqueue & Dequeue Operations



Exercise

1) Write a method **stutter** that accepts a queue of integers as a parameter and replaces every element of the queue with two copies of that element.

– front [1, 2, 3] back

becomes

front [1, 1, 2, 2, 3, 3] back

2) Write a method **mirror** that accepts a queue of strings as a parameter and appends the queue's contents to itself in reverse order.

– front [a, b, c] back

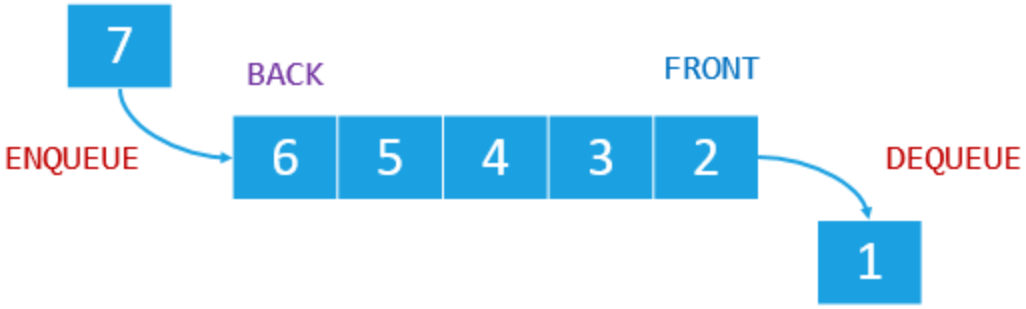
becomes

front [a, b, c, c, b, a] back

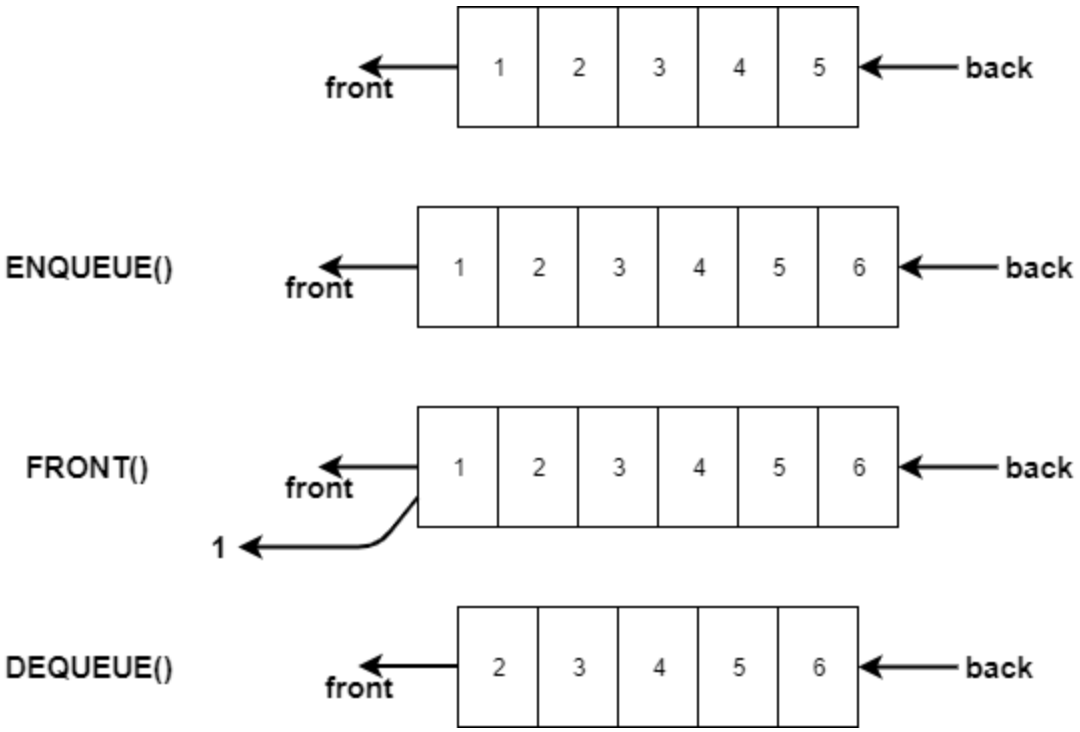
3) The following table shows a series of queue operations. Find the Return Value & queue Contents:

Operations	Return Value	Queue Contents
enqueue(1)		
enqueue(2)		
enqueue(3)		
isempty()		
enqueue(4)		
peek()		
enqueue(5)		
peek()		
dequeue()		
dequeue()		
peek()		
enqueue(1)		
dequeue()		
peek()		

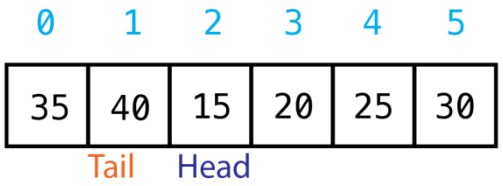
Example

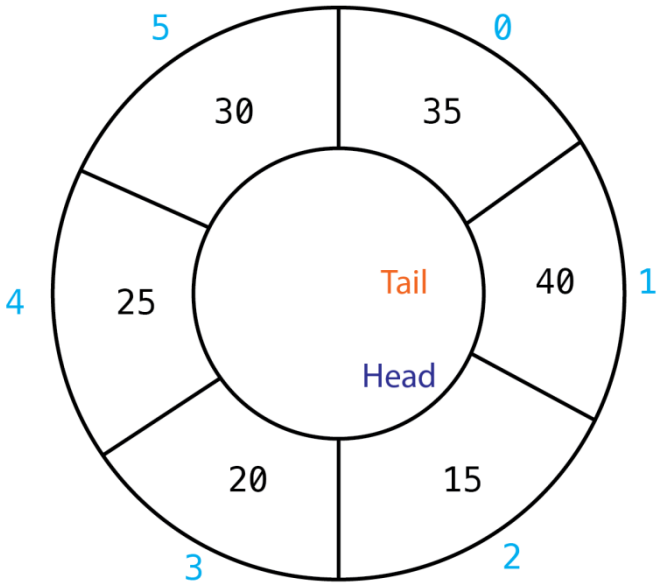


Example


front
1
2
3
4
5
back
ENQUEUE()
front
1
2
3
4
5
6
back
FRONT()
front
1
2
3
4
5
6
back
1
DEQUEUE()
front
2
3
4
5
6
back

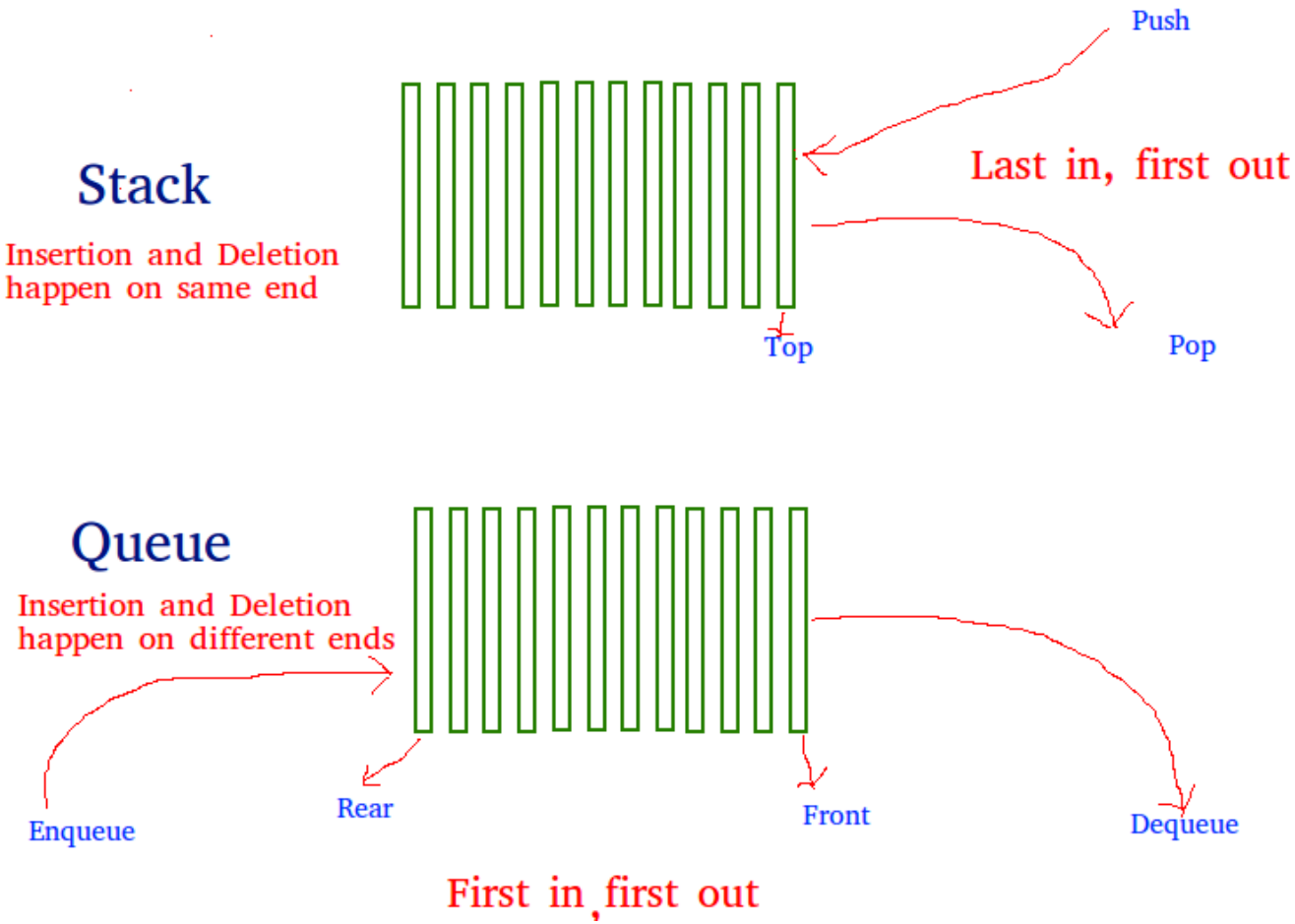
Circular Queue






```
Q = circularQueue(6)
Q.Enqueue(5)
Q.Enqueue(10)
Q.Enqueue(15)
Q.Enqueue(20)
Q.Enqueue(25)
Q.Enqueue(30)
Q.Dequeue()
Q.Dequeue()
Q.Enqueue(35)
Q.Enqueue(40)
```

Stack & Queue


Push
Stack
Insertion and Deletion
happen on same end
Last in, first out
Top
Pop
Queue
Insertion and Deletion
happen on different ends
Enqueue
Rear
Front
Dequeue
First in, first out

Difference between Stack & Queue