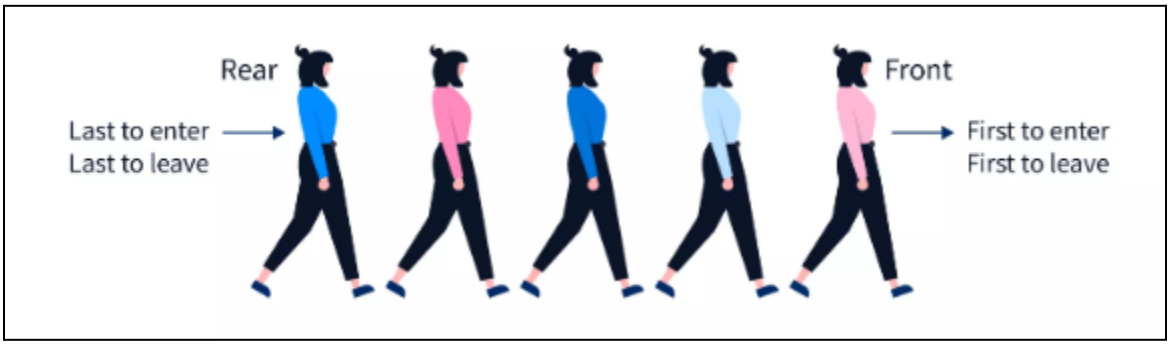
Queue

Similar to other data structures, a queue is a data structure used for storing data. In queue, the order in which data arrives is important. In general, a queue is a line of people or things waiting to be served in sequential order starting at the beginning of the line or sequence.

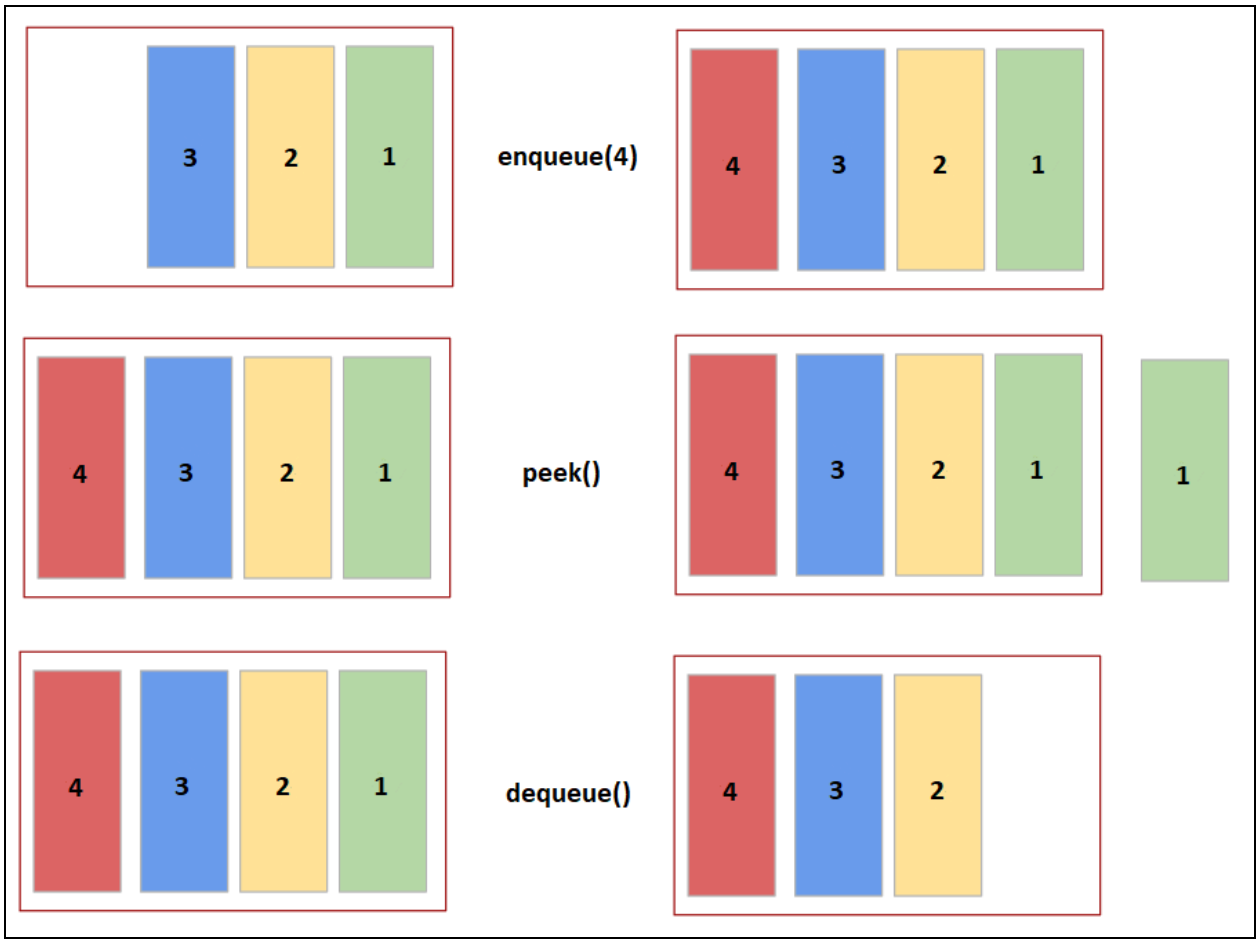


Definition: A queue is an ordered list in which insertions are done at one end (back) and deletions are done at the other end (front). The first element to be inserted is the first one to be deleted. Hence, it is called First in First out (FIFO).

Hence, it's not possible to add or remove elements from the middle of the queue. The only element that can be taken out is the one that came first.

Queue Operations:

- enqueue(obj): It stores a new object in the queue at the end. If the queue is full, this operation throws a QueueOverflow exception.
- dequeue: It removes the first object from the queue. If the queue is empty, this operation throws a QueueUnderflow exception.
- peek: This operation shows the first item in the queue. This operation does not remove any items from the queue.



Queue Implementation:

- Circular Array Based Stack:

```
class Queue:
        def __init__(self, size);
                self.queue = np.zeros(size, dtype=int)
                self.front = 0
                self.back = 0
                self.capacity = 0       #no elements
```

```
def enqueue(self, elem):
        if self.capacity == len(self.queue):
                print("Queue Overflow")
        else:
                self.queue[self.back] = elem
                self.back = (self.back+1) % len(self.queue)
                self.capacity += 1
```

```
def dequeue(self):
        if self.capacity == 0:
                print("Queue Underflow")
        else:
                dequeued_item = self.queue[self.front]
                self.queue[self.front] = None      #None or 0
                self.front = (self.front+1) % len(self.queue)
                self.capacity -= 1
                return dequeued_item
```

```
def peek(self):
        if self.capacity == 0:
                print("Queue is empty")
        else:
                return self.queue[self.front]
```

- Linked List Based Queue

```
class Node:
        def __init__(self, elem, next):
                self.elem = elem
                self.next = next
```

```
class Queue:
    def __init__(self):
        self.front = None
        self.back = None
```

```
def enqueue(self, elem):
    if self.front == None:
        self.front = Node(elem, None)
        self.back = self.front
    else:
        n = Node(elem, None)
        self.back.next = n
        self.back = self.back.next
```

```
def dequeue(self):
    if self.front == None:
        print("Queue Underflow")
    else:
        dequeued_item = self.front.elem
        temp = self.front
        self.front = self.front.next
        temp.next = None
        return dequeued_item
```

```
def peek(self):
    if self.front == None:
        print("Queue is empty")
    else:
        return self.front.elem
```

Queue Simulation:

You need to perform the following operations on an array-based queue where the length of the array is 4 and the starting index of front and back is 3.

enqueue a, enqueue b, dequeue, peek, enqueue c, peek, dequeue.

Index →	0	1	2	3	Front index	Back Index
Initial					3	3
enq (a)				a	3	0
enq (b)	b			a	3	1
deq	b				0	1
peek	b				0	1
enq (c)	b	c			0	2
peek	b	c			0	2
deq		c			1	2

Dequeued values: a, b

Peeked values: b, b