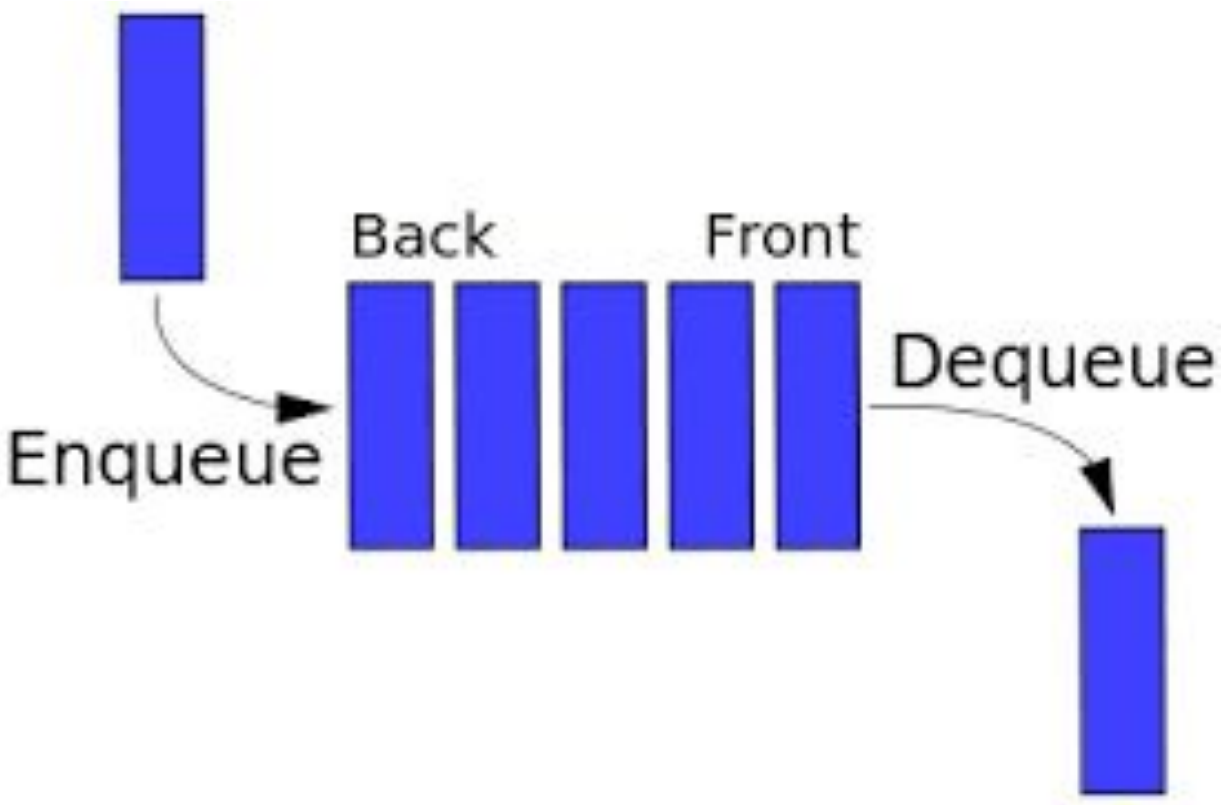
Data Structures

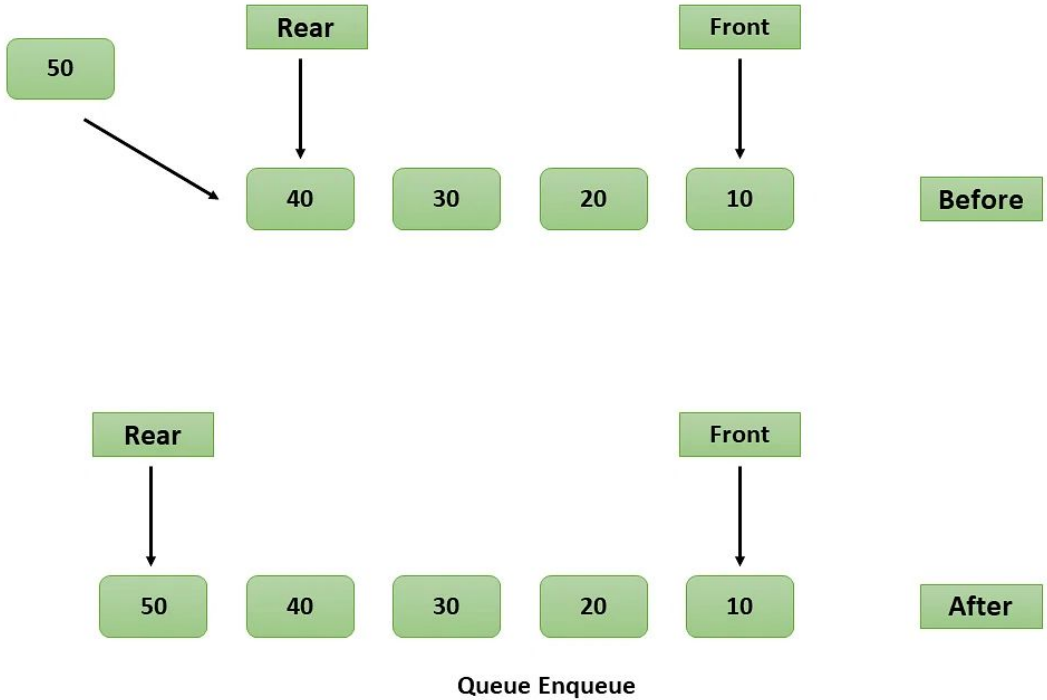
Lecture 6
Queue

Credit: Prantik Paul, Lecturer, BRAC University

Queue (Data Structure)



Queue (Enqueue)



Queue Enqueue

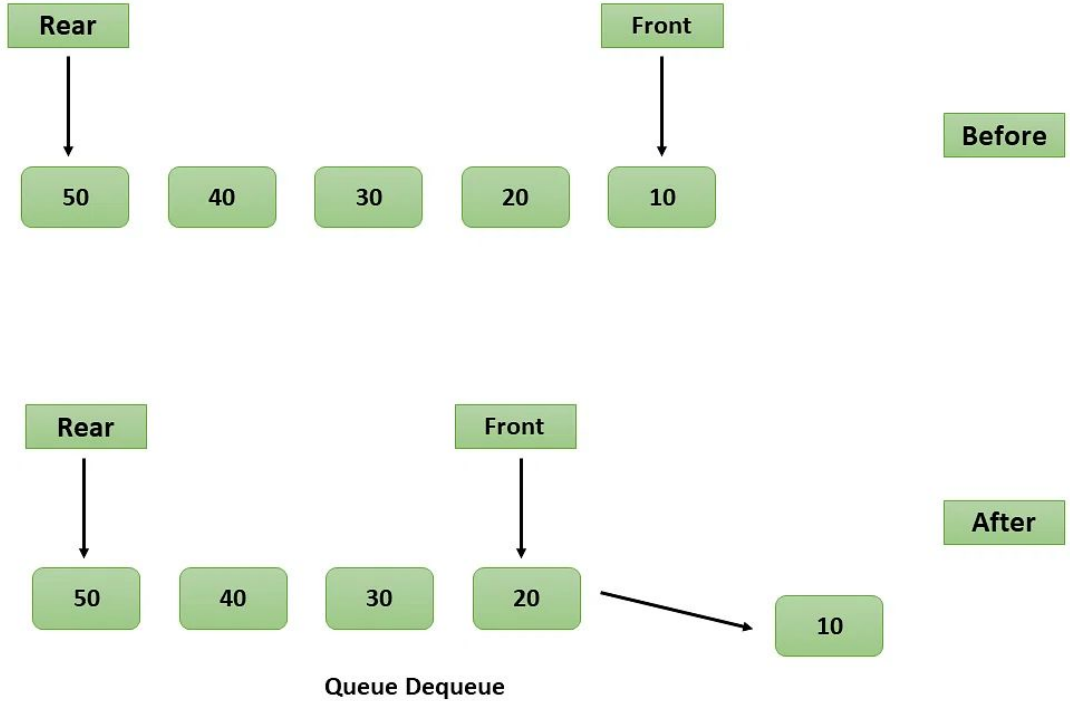
Queue (Enqueue)

```
class Queue:

  def __init__(self):
    self.front = None
    self.back = None

  def enqueue(self, elem):
    if self.front == None: # This means first item
      self.front = Node(elem, None)
      self.back = self.front # Initially front and back are same
    else:
      n = Node(elem, None)
      self.back.next = n # Inserting at the last
      self.back = self.back.next # Moving back at the end
```

Queue (Dequeue)



Queue Dequeue

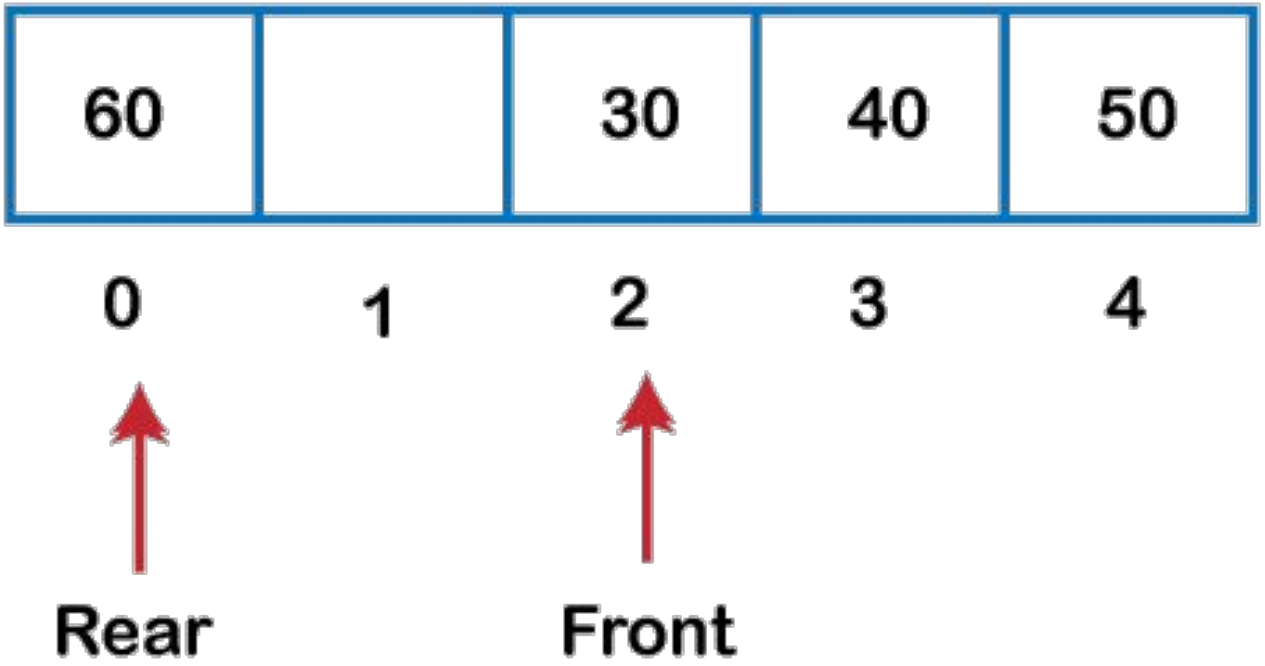
Queue (Dequeue)

```
def dequeue(self):
    if self.front == None: # For empty Queue
        print("Queue Underflow Exception")
    else:
        dequeued_item = self.front.elem
        self.front = self.front.next # Moving the front pointer
        return dequeued_item # Returning the dequeued item
```

Queue (Peek)

```
def peek(self):
    if self.front == None: # For empty Queue
        print("Queue Underflow Exception")
    else:
        return self.front.elem # Returning the front item
```

Circular Array



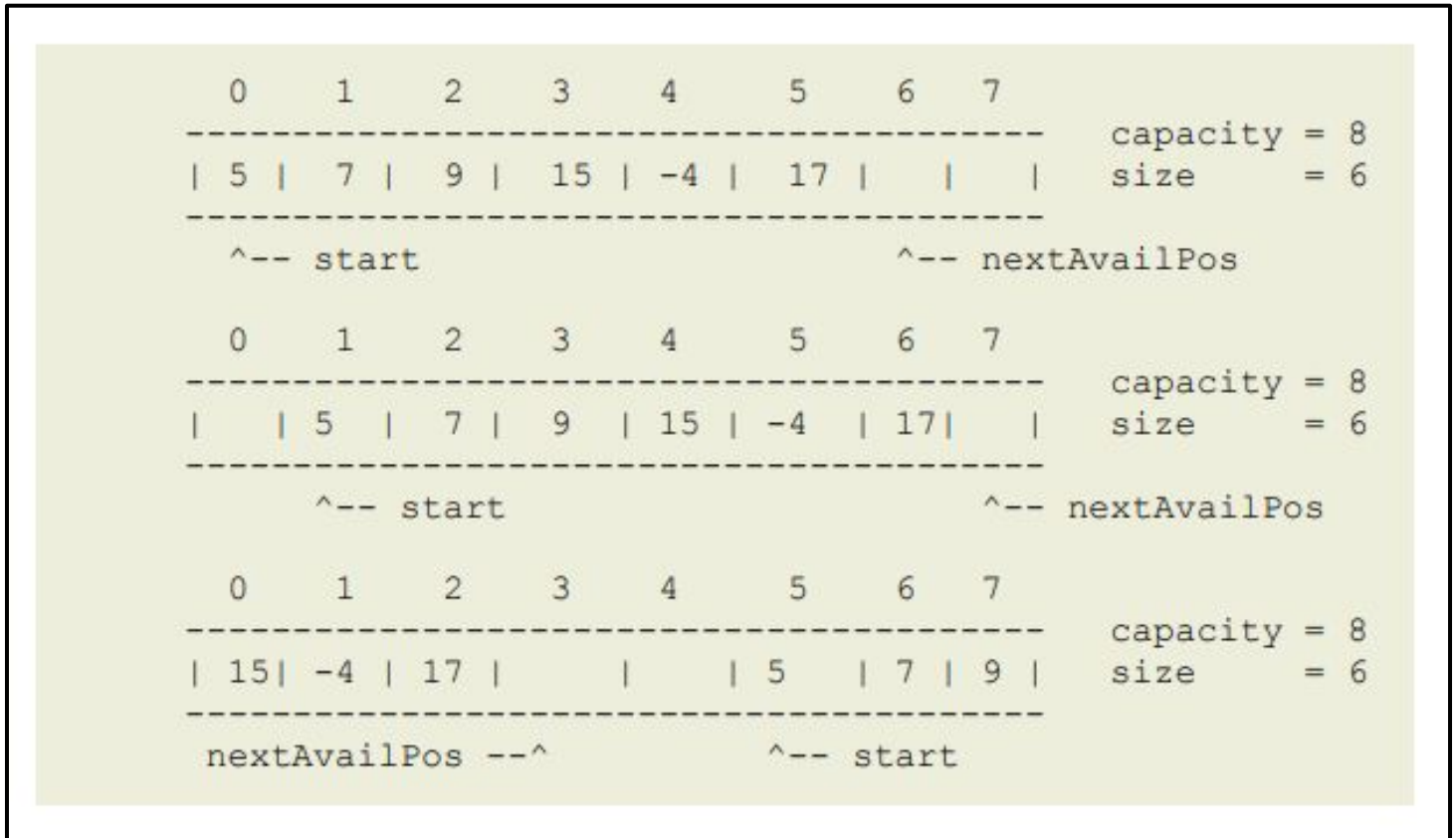
Circular Array (Indexing)

a = buffer[4 % 5]

a = buffer[(4 + 2) % 5]

a = buffer[(0 - 2) %5]

Circular Array (Structure)



Circular Array (Iteration - Forward)

```
# Forward Iteration
def forwardIteration(cir, start, size):
    k = start
    for i in range(size):
        print(cir[k])
        k = (k + 1) % len(cir)
```

Circular Array (Iteration - Backward)

```
# Backward Iteration
def backwardIteration(cir, start, size):
    k = (start + size - 1) % len(cir)
    for i in range(size):
        print(cir[k])
        k = (k - 1) % len(cir)
```

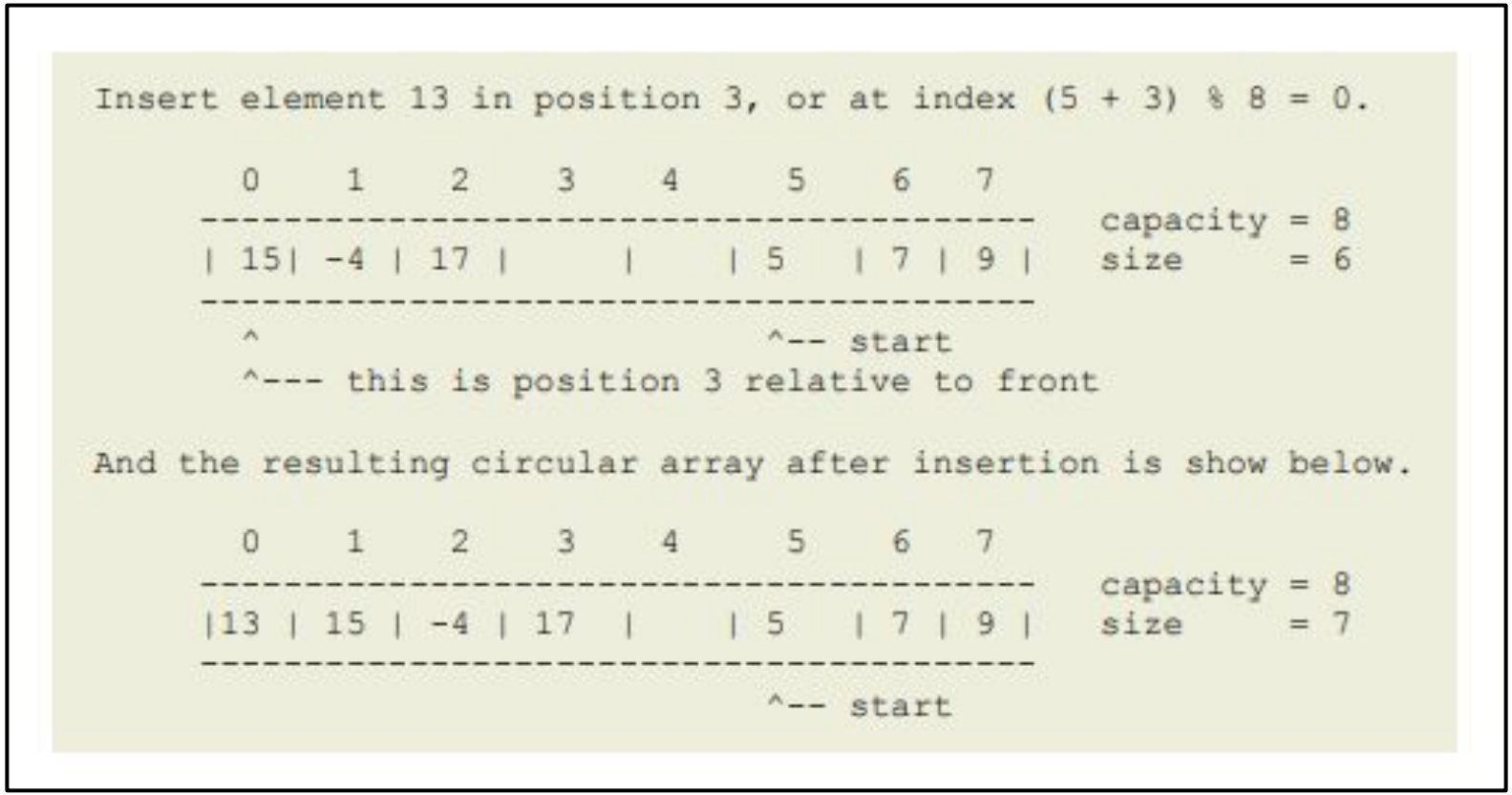
Circular Array (Insertion)

Offset i index = (start + i) % len(cir)

Last index = (start + size - 1) % len(cir)

Next available index = (start + size) % len(cir)

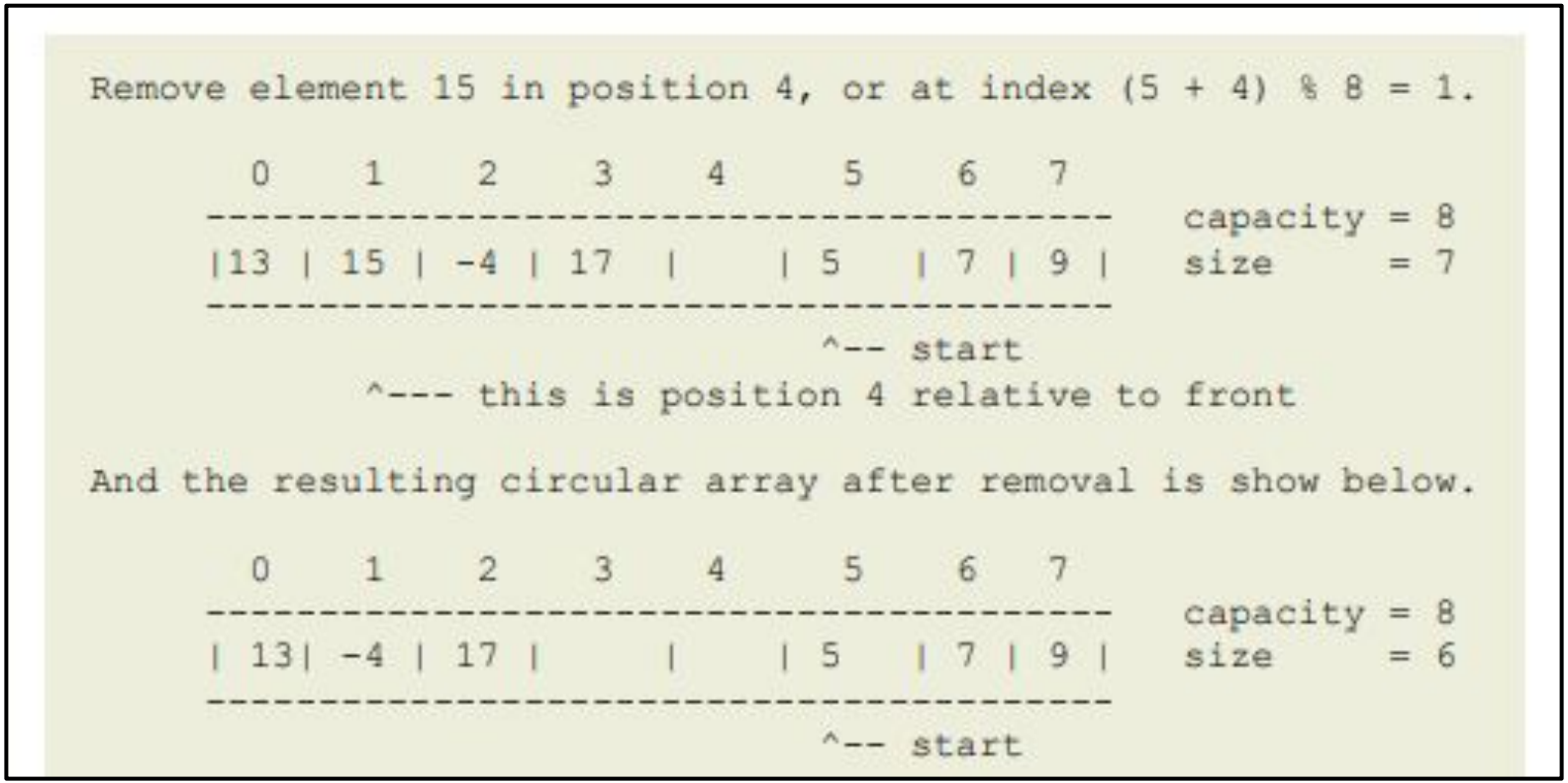
Circular Array (Insertion)



Circular Array (Insertion)

```python
# Insert in Circular Array
def insert(cir_arr, start, size, elem, pos):
  if size == len(cir_arr):
    print("No empty Spaces")
  nShifts = size - pos
  fr = (start + size - 1) % len(cir_arr)
  to = (fr + 1) % len(cir_arr)
  for i in range(nShifts):
    cir_arr[to] = cir_arr[fr]
    to = fr
    fr = (fr - 1) % len(cir_arr)
  idx = (start + pos) % len(cir_arr)
  cir_arr[idx] = elem
  size += 1
  return cir_arr
```

Circular Array (Remove)



Circular Array (Remove)

```
# Remove value from circular array by left shift
def removeByLeftShift(cir_arr, start, size, pos):
  nShift = size - pos - 1
  idx = (start + pos) % len(cir_arr)
  removed = cir_arr[idx]
  to = idx
  fr = (to + 1) % len(cir_arr)
  for i in range(nShifts):
    cir_arr[to] = cir_arr[fr]
    to = fr
    fr = (fr + 1) % len(cir_arr)
  cir_arr[fr] = None
  size -= 1
  return removed
```

Queue (with Array)

```
class ArrayQueue:
  def __init__(self):
    self.queue = [None] * 10 # Queue with size 10
    self.front = 0 # Initializing at index 0
    self.back = 0 # Initializing at index 0
    self.size = 0 # no elements
```

Queue (Enqueue)

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

Queue (Dequeue)

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

Queue (Peek)

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

Queue Simulation

len(array) = 4

enqueue a,

enqueue b,

dequeue,

peek,

enqueue c,

peek,

dequeue

Queue Simulation

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

len(array) = 4

Start index = 3

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