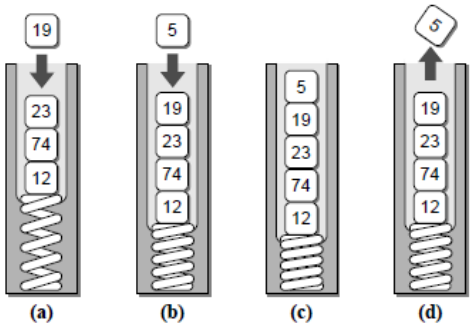
CPE111: Programming with Data Structures

Week3: Stacks, Queues, Deques

Practice I) Stacks
Mission: Complete the code in **List1**



Stack ADT

A *stack* is a data structure that stores a linear collection of items with access limited to a last-in first-out (LIFO) order. Adding and removing items is restricted to one end known as the *top* of the stack. An empty stack is one containing no items.

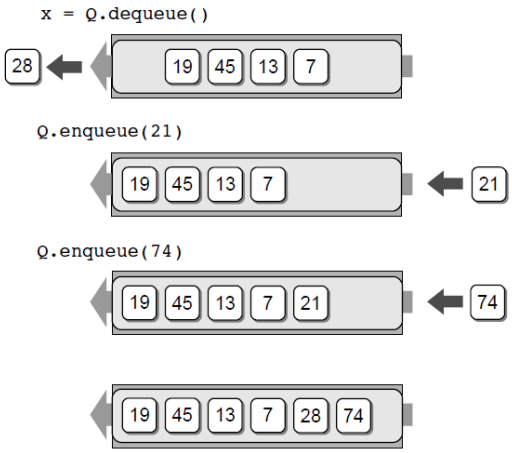
- **Stack():** Creates a new empty stack.
- **isEmpty():** Returns a boolean value indicating if the stack is empty.
- **length ():** Returns the number of items in the stack.
- **pop():** Removes and returns the top item of the stack, if the stack is not empty. Items cannot be popped from an empty stack. The next item on the stack becomes the new top item.
- **peek():** Returns a reference to the item on top of a non-empty stack without removing it. Peeking, which cannot be done on an empty stack, does not modify the stack contents.
- **push(item):** Adds the given item to the top of the stack.

List1: pyliststack.py

```
1 # Implementation of the Stack ADT using a Python list.
2 class Stack :
3       # Creates an empty stack.
4       def __init__( self ):
5               self._theItems = list()
6
7       # Returns True if the stack is empty or False otherwise.
8       def isEmpty( self ):
9               return len( self ) == 0
10
11      # Returns the number of items in the stack.
12      def __len__ ( self ):
13              return len( self._theItems )
14
15      # Returns the top item on the stack without removing it.
```

```
16      def peek( self ):
17              assert not self.isEmpty(), "Cannot peek at an empty stack"
18              return self._theItems[-1]
19
20      # Removes and returns the top item on the stack.
21      def pop( self ):
22              assert not self.isEmpty(), "Cannot pop from an empty stack"
23              [                        ] ← fill code here
24
25      # Push an item onto the top of the stack.
26      def push( self, item ):
27              [                        ] ← fill code here
```

Practice II) Queues
Mission: Complete the code in **List2**



Queue ADT

A *queue* is a data structure that a linear collection of items in which access is restricted to a first-in first-out (FIFO) basis. New items are inserted at the back and existing items are removed from the front. The items are maintained in the order in which they are added to the structure.

- **Queue():** Creates a new empty queue, which is a queue containing no items.
- **isEmpty():** Returns a boolean value indicating whether the queue is empty.
- **length ():** Returns the number of items currently in the queue.
- **enqueue(item):** Adds the given item to the back of the queue.
- **dequeue():** Removes and returns the front item from the queue. An item cannot be dequeued from an empty queue.

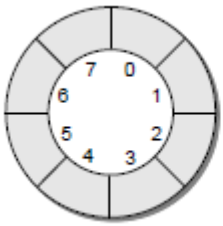
List2: pylistqueue.py

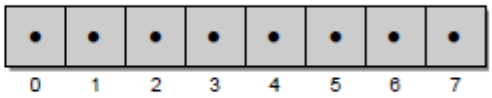
```
1 # Implementation of the Queue ADT using a Python list.
2 class Queue :
```

```
3       # Creates an empty queue.
4       def __init__( self ):
5       self._qList = list()
6
7       # Returns True if the queue is empty.
8       def isEmpty( self ):
9               return len( self ) == 0
10
11      # Returns the number of items in the queue.
12      def __len__( self ):
13              return len( self._qList )
14
15      # Adds the given item to the queue.
16      def enqueue( self, item ):
17              [                    ] ← fill code here
18
19      # Removes and returns the first item in the queue.
20      def dequeue( self ):
21              assert not self.isEmpty(), "Cannot dequeue from an empty queue."
22              [                    ] ← fill code here
```

Practice III) Queues with Circular Array

Mission: **List3** is another way to implement queues by using circular array. Complete the code in **list3**.



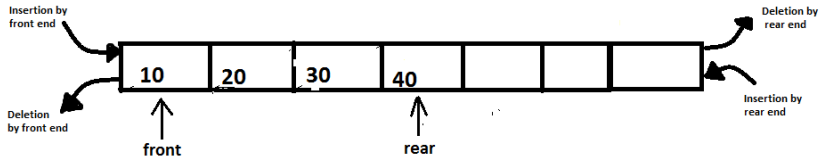


List:3 arrayqueue.py

```
1 # Implementation of the Queue ADT using a circular array.
2 from array import Array
3
4 class Queue :
5       # Creates an empty queue.
6       def __init__( self, maxSize ) :
7               self._count = 0
8               self._front = 0
9               self._back = maxSize - 1
10              self._qArray = Array( maxSize )
11
```

```
12      # Returns True if the queue is empty.
13      def isEmpty( self ) :
14              return self._count == 0
15
16      # Returns True if the queue is full.
17      def isFull( self ) :
18              [                          ] ← fill code here
19
20      # Returns the number of items in the queue.
21      def __len__( self ) :
22              return self._count
23
24      # Adds the given item to the queue.
25      def enqueue( self, item ):
26              assert not self.isFull(), "Cannot enqueue to a full queue."
27              maxSize = len(self._qArray)
28              self._back = (self._back + 1) % maxSize
29              self._qArray[self._back] = item
30              self._count += 1
31
32      # Removes and returns the first item in the queue.
33      def dequeue( self ):
34              assert not self.isEmpty(), "Cannot dequeue from an empty queue."
35              item = self._qArray[ self._front ]
36              maxSize = len(self._qArray)
37              self._front = (self._front + 1) % maxSize
38              self._count -= 1
39              return item
```

Practice IV) Double-ended queues (DEQ)
Implement Deques by yourself



Create Double-ended queues from Python list or array, according to double-ended queues ADT below:

Double-ended queue ADT

A **doubled-ended queue** is a queue-like data structure that supports insertion and deletion at both the front and the back of the queue.

- **Deque():** Create a new empty deque, which no item with in.
- **AddFirst(item)**: Add an item to the front of deque.
- **AddRear(item)**: Add an item to the back of deque.
- **DeleteFirst( )**: Remove and return the first item from deque; an error occurs if the deque is empty.
- **DeleteRear ( )**: Remove and return the last item from deque; an error occurs if the deque is empty.
- **First()**: Return (but do not remove) the first item of deque; an error occurs if the deque is empty.
- **Rear()**: Return (but do not remove) the last item of deque; an error occurs if the deque is empty.
- **isEmpty( )**: Return True if deque does not contain any items.
- **length( )**: Return the number of items in deque.

Reference

Rance D. Necaise. *Data Structures and algorithms using python*. *Chapter2*. John Wiley&Sons,Inc., 2011

Michael T.Goodrich, Roberto Tamassia, Michael H. Goodwasser. *Data Structures and Algorithms in python*. *Chapter5*. John Wiley&Sons,Inc. 2013