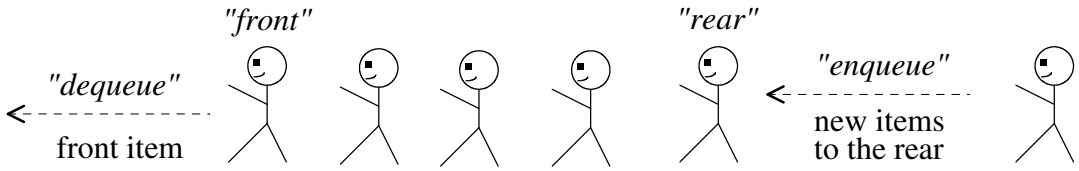
Data Structures (810:052) Lab 5 Name:_____

Objective: To understand FIFO (First-In-First-Out) queue implementations in Python including being able to determine the big-oh of each operation.

Background: A FIFO queue is basically what we think of as a waiting line.

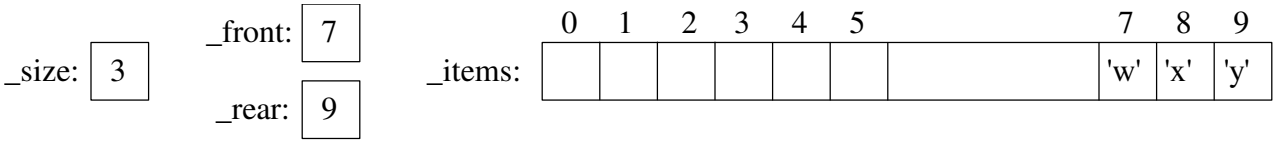


The operations/methods on a queue object, say myQueue are:

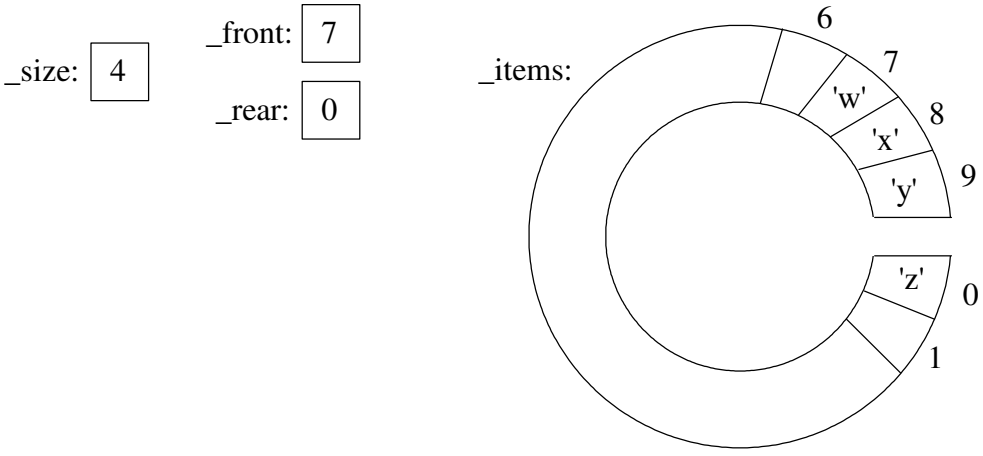
Method Call on myQueue object	Description
`myQueue.dequeue()`	Removes and returns the front item in the queue.
`myQueue.enqueue(myItem)`	Adds `myItem` at the rear of the queue
`myQueue.peek()`	Returns the front item in the queue without removing it.
`myQueue.isEmpty()`	Returns `True` if the queue is empty, or `False` otherwise.
`len(myQueue)`	Returns the number of items currently in the queue
`str(myQueue)`	Returns the string representation of the queue

To start the lab: Download and unzip the file at: www.cs.uni.edu/~fienup/cs052f10/labs/lab5.zip

Part A: As pointed out in section 15.4.2 we can avoid "shifting the items left" on a dequeue operation by maintaining the index of the front item in addition to the rear. Overtime, the used portion of the array (where the actual queue items are) will drift to the right end of the array with the left end being unused, i.e.:



Now if we enqueue another item, we'd like the rear of the queue to "wrap" around to index 0, i.e., we'd like the array to behave "circularly." After we enqueue('z'), we would have:



a) Complete the expected big-oh notation for each queue method for this circular-array queue implementation. Assuming "n" items in the queue.

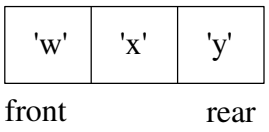
`dequeue()`	`enqueue(item)`	`peek()`	`isEmpty()`	`__len__()`	`__str__()`

b) The file `queue2.py` contains the start of the `CircularArrayQueue` class. Complete the `CircularArrayQueue` class, and thoroughly test it using the menu-driven program in the `testQueue2.py` file.

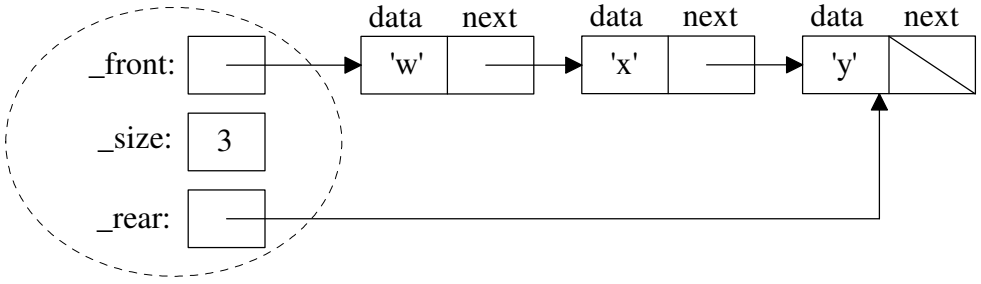
After thoroughly testing your circular-array implementation, raise you hand and demonstrate your queue.

Part B: The `Node` class defined in node.py is used to dynamically create storage for a new item added to a singly-linked list implementation of the `LinkedQueue` class in file `queue.py`. Conceptually, a `LinkedQueue` object would look like:

"Abstract Queue"



LinkedQueue Object



a) Complete the expected big-oh notation for each queue method for this linked-list queue implementation. Assuming "n" items in the queue.

`dequeue()`	`enqueue(item)`	`peek()`	`isEmpty()`	`__len__()`	`__str__()`

b) The file `queue.py` contains the start of the `LinkedQueue` class. Complete the `LinkedQueue` class, and thoroughly test it using the menu-driven program in the `testQueue.py` file.

After thoroughly testing your linked implementation, raise you hand and demonstrate your queue.

EXTRA CREDIT: Unfortunately, the `Node` class does NOT practice good object-oriented design by allowing the LinkedQueue class access to its data and next attributes without going through accessor (e.g., getData, getNext, etc.) and mutator (e.g., setData, setNext, etc.) methods. For extra credit:

- Rewrite the `Node` class to follow good object-oriented design.
- Rewrite the `LinkedQueue` class to fixed this design problem.