Computer Science 112 - Fundamentals of Programming II
Lab Project 5
Due on Github 11:59pm Monday 25 October

In this project, you will experiment with applications of stacks and queues.

Part I: Finish **linkedlist.py**, **stack.py**, and **queue.py**

This is the code that we've been working on together. If you haven't saved a copy for yourself, let me know, and we'll rebuild it together. As usual, each file should have a little **`main()`** at the bottom for testing.

Part II: Write **prioirityqueue.py**

As usual, copy/paste/modify is your best bet: in this case, you can copy **queue.py** and add the necessary changes. For my test at the bottom, I pushed items out of order ('c', 'a', 'd', 'b', 'f', 'e') and then did some popping to make sure that they came out in the right order ('a', 'b', 'c', 'd', 'e', 'f').

Part III: Bribe your way to success!

If you've read the 2008 book *McMafia* or seen the recent Amazon TV series made from it, you know that hard work and talent are not enough to get ahead in international business these days: *you need to bribe people too*. For this part of the assignment, you will write a little interactive TUI program **bribery.py**, which will works as follows:

```
How much you got ? 0
Got to the end of the queue, loser!

How much you got? 100
Excellent, you are now in position #5 in the queue

etc.
```

Hints:

(1) As well as adding the new node to the appropriate place in the linked list, your **`PriorityQueue.push()`** method can return that position as a number (lower = better).

(2) Our priority queue puts smaller items at the front, whereas bribery should put *larger* items at the front. To fix this we could further complicate our **`PriorityQueue`** class, or we could exploit a simple trick in **bribery.py** to ensure that larger bribes go to the front.

Part IV: Maze runner

As we discussed in class, a classic application of stacks is solving problems like escaping from a maze, where running into a dead-end means you have to backtrack to a previous position. For this exercise, you'll augment your **`Stack`** class to support this task.

Running the **mazemodel.py** script will show you what you need to add. For example, although I mentioned that a true stack doesn't have to support iteration, you will need to add that feature for this project. Once you've added the required methods, running **mazemodel.py** should produce an output showing the a solution.