Assignment: This assignment relies on the implementation of four different classes:

- `LinkedList.py`: to implement a doubly-linked list
- `Stack.py`: implements the Stack ADT, using a Python `list`
- `Queue.py`: implements the Queue ADT, currently using a Python `list`
- `Maze.py`: implements maze construction and printing, requiring you to implement DFS and BFS algorithms

The assignment itself is pretty straightforward. <u>Throughout, pay attention to doscstrings, raising exceptions when appropriate!</u>

1. Complete the details for `LinkedList.py`, using a doubly-linked list. Note that there are two pointers in a `Node` now: one to the next `Node` and one to the previous `Node`. Make sure that you update these appropriately for each of the methods, whether adding to the left or right, or removing from the left or right. (Your HW8 should have you already done with this portion.)

2. Modify your implementation of the `Queue` class to use your doubly-linked `LinkedList` class rather than a Python `list`. What changes need to be made? Any changes to `__str__`?

   The interface for `Queue` is given below:

   - `push(item: T) -> None`: inserts item @ end of the queue
   - `pop() -> T`: removes & returns item @ front of queue
   - `top() -> T`: returns reference to first item w/o removing
   - `isEmpty() -> bool`: indicates whether queue is empty
   - `__len__() -> int`: returns queue length

   Assume that a newly created queue is empty. Similar to that for `Stack`, a `pop` or `top` on an empty queue should raise an `EmptyError` (derived class of your own creation) exception.

   Test this carefully before moving on.

3. Complete the implementation of maze searching inside `Maze`. You will need to use your versions of `Stack` and `Queue` as modified/implemented above. (The `Maze` class will not directly use your `LinkedList` class, but rather will directly use `Stack` and `Queue`. The internals of `Queue` use the doubly-linked `LinkedList`.)

   You will need to complete these methods as discussed in lecture, presuming a NSWE search direction:

   - `def getSearchLocations(self, cell: Cell) -> list[Cell]:`
   - `def dfs(self) -> Cell | None:`
   - `def bfs(self) -> Cell | None:`

   (Comments are provided in the stub/skeletons of those methods for additional guidance.) Test your results on a variety of different mazes.

Reflection: In an attached PDF, comment on the DFS vs. BFS approach for solving a maze. Experiment with several different size mazes and different proportions of blocked cells. Provide output to justify your comments.

- When considering the same maze (use the same initial seed when creating the maze), compare the resulting path length for DFS vs. BFS as well as the number of cells pushed during the exploration (you will have to add additional bookkeeping to your classes to count the number of cells pushed to either the stack or queue).

  Try this on multiple different mazes, but using the exact same maze (manifested via choice of initial seed) to compare the two algorithms each time.

- How do the algorithms compare in terms of path length when there are relatively few blocked cells in the maze? Many blocked cells (but still with a path to the goal available)?
- How does the choice of search direction affect the algorithms? Experiment with each algorithm on several different mazes each using each of:
  - `SearchOrder.NSWE`
  - `SearchOrder.NESW`
  - `SearchOrder.RANDOM` (use `random.shuffle`)

  Specifically, for the exact same maze, how do the different search options affect what DFS produces? BFS? Does the size of the maze play an important role?
- Comment on and justify the big-Oh analysis for your `Stack` and `Queue` classes, specifically:
  - `Stack`'s `push` and `pop`, when using a Python `list` under the hood;
  - `Queue`'s `push` and `pop`, when using a Python `list` under the hood;
  - `Queue`'s `push` and `pop`, when using your doubly-linked `LinkedList` under the hood.

Submitting: When done, upload your four Python files (`LinkedList`, `Stack`, `Queue`, and `Maze.py`) to Lyceum along with your PDF with your reflection/discussion.