Programming Exercises

1. Extend the `LList` class by implementing some of the other methods that the built-in Python list supports: `__min__`, `__max__`, `index`, `count`, and `remove`.

2. Perform an experimental comparison of the efficiency of inserting at the front of a built-in Python list and of inserting at the front of an `LList`. Before you start, form a hypothesis about what you expect to see. Conduct some experiments to test your hypothesis. Write a complete lab report explaining your findings. Be sure to include a thorough description of your hypothesis and the experiments you ran. Make sure your discussion tells if your hypothesis was supported.

3. Add a `last` instance variable to the `LList` class along the lines suggested in the chapter, so that the `append` method can be implemented in $\Theta(1)$ time. This will require you to update a number of the methods to ensure `self.last` is always a reference to the last `ListNode` in the linked structure.

4. Finish the implementation of the `LListCursor` class and provide a complete set of unit tests for the `LinkedCursorList` class using the list cursor API.

5. Suppose we want our list cursors to be able to move both directions. That is, in addition to the `advance` operation, we'd also like a `backup` operation. Add this ability to the `PyListCursor`. Make sure to write complete unit tests for your updated cursor.

6. Add the capability of the previous exercise to the `LListCursor` class. To do this your cursor will have to keep track of a "trail" of previous nodes. You can use a Python list for this purpose. The predecessor of each node is appended to the list as the cursor advances and then is popped back off the end of the list when the cursor backs up.

7. Modify the linked implementation of the Python list API so that it is a *doubly-linked list*, that is, each `ListNode` has a reference to the `ListNode` before it and the `ListNode` after it. Also add a method named `reverse_iter` that iterates over the list in reverse order using the `yield` keyword. Modify your unit testing code so it also checks the reverse links. Using your doubly-linked list, modify the cursor for this new list to that it solves the previous problem without having to mantain an internal list of predecessor nodes.