Lab 5: Linked Lists Solution

3) Augmenting our linked list implementation

It makes sense that our implementation of *LinkedList.__len__* is so slow; but how is the built-in *list.__len__* so much faster?

It turns out that built-in Python lists use an additional attribute to store their length, so that whenever *list.__len__* is called, it simply returns the value of this attribute.

The process of adding an extra attribute to an existing data structure is known as augmentation, and is very common in computer science. Every data structure augmentation poses a question of trade-offs:

- The benefit of augmenting is that the extra attribute makes certain operations simpler and/or more efficient to implement.

- The cost of augmenting is that this extra attribute increases the complexity of the data structure implementation.

  In particular, such attributes often have representation invariants associated with them that must be maintained every time the data structure is mutated.

1. Create a copy of your *LinkedList* class (you can pick a name for the copy), and add a new private attribute *_length* to the class documentation and initializer.

   Write down a representation invariant for this new attribute; you can use English here, but try to be precise without using the word "length" in your description. (Hint: how do we define length in terms of the nodes of a list?)

```
1    class LinkedList:
2        """A linked list implementation of the List ADT.
3        """
4        # === Private Attributes ===
5        # _first:
6        #     The first node in the linked list, or None if the list
   is empty.
7        # _length:
8        #     The number of nodes in the linked list.
9        #
```

```
10          # === Representational Invariants ===
11          # - _length >= 0
12
```

Listing 1: task_3_step_1_solution.py

2. Update each mutating method to preserve your representation invariant for this new attribute. (Why don't we need to worry about the non-mutating methods?)

3. Now let's enjoy the benefit of this augmentation!

   Modify your new class' __len__ method to simply return this new attribute.

   Use doctests wisely to ensure you've made the correct changes for this and the previous step.

4. Finally, perform some additional timing tests to demonstrate that you really have improved the efficiency of __len__.