Lab 5: Linked Lists

1) Practice with linked lists

For this task: we have commented out the doctests in the methods. You will not be able to run them until you finish step (3) of this task, at which point you may uncomment them. We recommend you read all of the steps in this task before you begin.

1. In the starter code, find and read the docstring of the method *__len__*, and then implement it.

   You already implemented this method in this week's prep, but it's good practice to implement it again. (And if you missed this week's prep, do it now!)

2. Then, implement the methods *count*, *index*, and *__setitem__*.

3. You might have noticed that all the doctests were commented out in the previous part.

   This is because they use a more powerful initializer than the one we've started with.

   Your final task in this section is to implement a new initializer with the following interface:

```
        def __init__(self, items: list) -> None:
            """Initialize a new linked list containing the given items.

            The first node in the linked list contains the first item
            in <items>.
            """

```

   The lecture notes suggest one way to do this using *append*; however, here we want you to try doing this without using *append* (or any other helper method).

   There are many different ways you could implement this method, but the key idea is that you need to loop through *items*, create a new *_Node* for each item, link the nodes together, and initialize *self._first*.

   Spend time drawing some pictures before writing any code!

2) Timing *__len__* for linked lists vs. array-based lists

1. Most methods take longer to run on large inputs than on small inputs, although this is not always the case.

   Look at your code for your linked list method *__len__*.

   Do you expect it to take longer to run on a larger linked list than on a smaller one?

2. Pick one the following terms to relate the growth of *__len__*'s running time vs. input size, and justify.

   - constant, logarithmic, linear, quadratic, exponential

3. Complete the code in *time_lists.py* to measure how running time for your *__len__* method changes as the size of the linked list grows. Is it as you predicted?

   Now's let's assess and compare the performance of Python's built-in *list*. You can do this by simply adding it to the list of types that *list_class* iterates over. What do you notice about the behaviour of calling *len* on a built-in *list*?

3) Augmenting our linked list implementation

4) Additional exercises