Lab 5: Linked Lists Solution

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!)

```
        class LinkedList:
            ...
            def __len__(self) -> int:
                """Return the number of elements in this list.

                # >>> lst = LinkedList([])
                # >>> len(lst)              # Equivalent to lst.
    __len__()
                # 0
                # >>> lst = LinkedList([1, 2, 3])
                # >>> len(lst)
                # 3
                """
                curr = self._first
                count = 0

                while curr is not None:
                    curr = curr.next
                    count += 1

                return count

```

Listing 1: task_1_step_1_solution.py

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

count:

```
        class LinkedList:
            ...
            def count(self, item: Any) -> int:
                """Return the number of times <item> occurs in this
    list.

                Use == to compare items.

                # >>> lst = LinkedList([1, 2, 1, 3, 2, 1])
                # >>> lst.count(1)
                # 3
                # >>> lst.count(2)
                # 2
                # >>> lst.count(3)
                # 1
                """

                count = 0
                curr = self._first

                while curr is not None:

                    if curr.item == item:
                        count += 1

                    curr = curr.next

                return count

```

Listing 2: task_1_step_2_solution.py

index:

```
        class LinkedList:
            ...
            def index(self, item: Any) -> int:
                """Return the index of the first occurrence of <item>
     in this list.

                Raise ValueError if the <item> is not present.

                Use == to compare items.

                # >>> lst = LinkedList([1, 2, 1, 3, 2, 1])
                # >>> lst.index(1)
                # 0
                # >>> lst.index(3)
                # 3
```

```
                # >>> lst.index(148)
                # Traceback (most recent call last):
                # ValueError
                """

                index = 0

                curr = self._first

                while curr is not None:
                    if curr.item == item:
                        return index

                    curr = curr.next
                    index += 1

                raise ValueError

```

Listing 3: task_1_step_2_solution.py

__setitem__:

```
        class LinkedList:
            ...
            def __setitem__(self, index: int, item: Any) -> None:
                """Store item at position <index> in this list.

                Raise IndexError if index >= len(self).

                # >>> lst = LinkedList([1, 2, 3])
                # >>> lst[0] = 100  # Equivalent to lst.__setitem__
    (0, 100)
                # >>> lst[1] = 200
                # >>> lst[2] = 300
                # >>> str(lst)
                # '[100 -> 200 -> 300]'
                """

                curr = self.first
                i = 0

                while (curr is not None) and (i <= index):
                    if index == i:
                        curr.item = item

                    curr = curr.next
                    i += 1

                raise IndexError
```

```
27
```

Listing 4: task_1_step_2_solution.py

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:

```
1         def __init__(self, items: list) -> None:
2             """Initialize a new linked list containing the given items.
3
4             The first node in the linked list contains the first item
5             in <items>.
6             """
7
```

   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!