CSC148 - More practice with linked list traversals

Recall the basic Linked List traversal pattern:

```python
curr = self._first          # Variable initialization
while curr is not None:     # Traversal loop
    ... curr.item ...
    curr = curr.next
```

On this worksheet, you'll work on developing two different methods that modify this basic pattern. In particular, both will involve changing the *while loop condition*, and you'll practice developing non-trivial loop conditions in a logical way.

1. Here is the docstring and implementation sections for the special method `LinkedList.__eq__`.

```python
def __eq__(self, other: LinkedList) -> bool:
    """Return whether this list and the other list are equal.

    Two lists are equal when each one has the same number of items,
    and each corresponding pair of items are equal (using == to compare).
    """
    # (1) Variable initialization

    # (2) Traversal loop

    # (3) Post-loop code
```
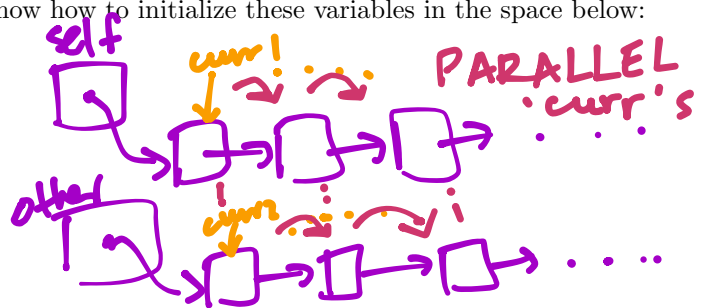
Our goal is to implement this method. Obviously, using just one variable `curr` is not enough to iterate through both lists. Instead, use two variables: `curr1` and `curr2`, one for each list. Show how to initialize these variables in the space below:

```python
# (1) Variable initialization

curr1 = self._first

curr2 = other._first
```


self
curr!
PARALLEL
curr's
other
curr

Next, let's work on (2), the traversal loop. The loop condition is a bit subtle. To make sure we get this right, we're going to go slow.

(a) First, let's think about when we want the loop to *stop*. This should be when we reach the end of `self` or the end of `other`. Write down a Python expression involving `curr1` and `curr2` that expresses this *stopping condition*. (By "expresses", we mean your expression should evaluate to `True` when the condition is true, and `False` otherwise.)

curr1 is None   or   curr2 is None

(b) The while loop condition should always be the *negation* of the stopping condition. Write down a Python expression for the while loop condition.

not (A or B)
= not A
and
not B

not (curr1 is None or curr2 is None)

⤷ curr1 is not None and curr2 is not None
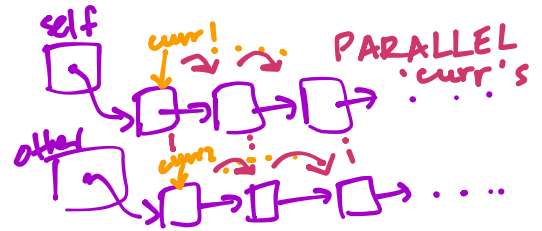
⤷ curr1   and   curr2

Using this, write down the while loop you would use to traverse the two lists. At each iteration, you should compare `curr1.item` against `curr2.item`; if this is False, you should be able to stop and return, without checking any other values!

```
# (2) Traversal loop
```

if curr1.item != curr2.item:
　　return False

curr1 = curr1.next
curr 2 = curr2.next

Finally, suppose we reach the end of the loop. We need some code to handle this case as well.

(a) What do we know about `curr1` and/or `curr2` after the loop ends? (Hint: look up at your stopping condition.)

curr1 is None   or   curr2 is None

(b) How can we use the values of `curr1` and `curr2` to check whether the lists have the same length?

curr1 is None   AND   curr is None

(c) Write the code that should go after the end of our loop. Remember that it should return `True` or `False`.

```
# (3) Post-loop code
```

return curr1 is None and curr2 is None

{ they may not be equal, so we check this}

2. Now, we're going to repeat the same process for the special method `__getitem__`, which enables indexing using square brackets: `lst[i]` is equivalent to `lst.__getitem__(i)` in Python!

```python
def __getitem__(self, index: int) -> Any:
    """Return the item at position <index> in this list.

    Raise an IndexError if the <index> is out of bounds.

    Precondition: index >= 0.
    """
```

(see linked-list.py)

For this implementation, you'll need to use two variables: `curr` for the current node in the list, and `i` for the current index in the list. Use the following strategy to implement `__getitem__` on a separate sheet of paper.

(a) Write down how to initialize `curr` and `i`.
(b) Write down the stopping condition for the loop, and then the while loop condition (negate the stopping condition).
(c) Implement the loop body.
(d) After the loop ends, use the stopping condition to remind yourself what you know about the values of `curr` and `i`. Use this to implement the post-loop code.