PIC 16: Homework 3 (due 4/23 at 10pm)

How should your answers be submitted?

- Download `hw3.py`. Do NOT change the filename.
- Replace `mjandr` by your name.
- Delete the `pass` statements and type your answers.

  Uncomment the other functions and type your answers.

  Do NOT change the class or function names.
- Do NOT include your test cases.

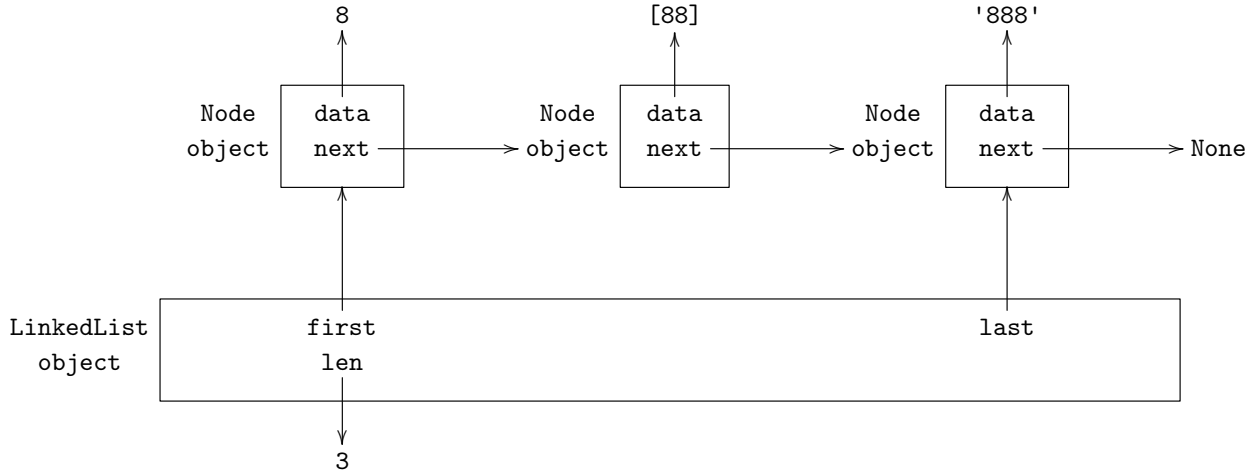
  This will hinder the grading process and might result in `0` points.
- Submit `hw3.py` to CCLE.

In this assignment, you will create a `class` called `LinkedList`. The introduction to the wikipedia article - https://en.wikipedia.org/wiki/Linked_list - describes what a linked list is.

To implement linked lists, we'll have two classes:

- `Node` which will have instance variables `data` and `next`;
- `LinkedList` which will have instance variables `first`, `last`, and `len`.

The linked list `[8 -> [88] -> '888' -> ]` will be realized using the following instance objects.



1. Make the `Node` class.

   Its initializer should take in one non-`self` parameter: `data`.

   It will initialize two instance variables `data` and `next`. `next` should be initialized to `None`.

2. Let's make the first draft for the initializer of `LinkedList`.

   In this first draft, the initializer should not use any non-`self` parameters. It simply initializes the instance variables `first`, `last`, and `len` to `None`, `None`, and `0`, respectively.

3. Write the function `append(self, data)`.

   This should take a linked list `[1 -> '2' -> [3] -> ]` and add one new `Node` so that it becomes `[1 -> '2' -> [3] -> object_referenced_by_data -> ]`. It may help to address the cases of

   - appending to an empty linked list and
   - appending to a non-empty linked list

   seperately.

   In the second case, you're going to need to create a new `Node`, have the formerly `last` node know this new node is the `next` Node, and update the `last` instance variable.

4. Write the function `prepend(self, data)`.

   This should take a linked list `[1 -> '2' -> [3] -> ]` and add one new `Node` so that it becomes `[object_referenced_by_data -> 1 -> '2' -> [3] -> ]`. It may help you to address the cases of prepending to an empty linked list and prepending to a non-empty linked list seperately.

   I'll leave it to you to think about the logic of this function. Drawing pictures is helpful: there's a reason I spent an hour drawing the diagram on the previous page!

5. Let's finish writing the initializer of `LinkedList`. If a non-`self` parameter is specified *and* it is a `list`, the initializer should make the corresponding linked list.

   Hint: There's a reason I made this part 5 instead of part 3.
   You should NOT need to introduce any instance variables that reference a `list`.

6. Write a magic method so that you can ask for the length of an instance of a `LinkedList` in the same way that you ask for the length of a `list`.

   ```
   LL = LinkedList([-1, 1])
   print(len(LL))
   ```

   should result in `2` being printed.

   Recall: You should NOT need to introduce any instance variables that reference a list.
   I gave you all the instance variables you need. Don't cheat.

7. Write a magic method, so that you can test whether two instances of `LinkedList` are equal.

   `[d1 -> d2 -> ... -> dn -> ]` and `[D1 -> D2 -> ... -> DN -> ]` are regarded as equal when

   - `n == N`, i.e. they have the same length;
   - for all `i` in `[1,2,...,n]`, `di == Di`.

   ```
   Lempty = LinkedList()
   L01 = LinkedList([0])
   L02 = LinkedList([0])
   L1 = LinkedList([1])
   print(Lempty == L01, L01 == L02, L01 == L1)
   ```

   should print `False True False`.

8. Write a magic method so that we can print instances of `Node` in a nice human friendly way.

   What information should we print?

   I don't see the `Node` class as the important class here: it's a helper class for `LinkedList`. I'm not going to bother having you make a machine readable output (which should be unambiguous), and in this human friendly version of printing I'm not going to care about displaying `next`. We will just print the `data` of the node. However, I think it is best to print this in a machine-readable way. This is consistent with the fact `lists` always print their elements in a machine readable way.

   ```
   n1 = Node(8)
   n2 = Node([88])
   n3 = Node('888')
   print(n1, n2, n3)
   ```

   should print `8 [88] '888'` (not `8 [88] 888`).

9. Write a magic method so that we can print instances of `LinkedList` in a human friendly way.

   The easiest way for me to tell you how to format things is to give an example.

   ```
   LL = LinkedList(['8', [8], [8], '8'])
   LL.append(1)
   LL.append(2)
   LL.append(3)
   LL.prepend(-1)
   LL.prepend(-2)
   LL.prepend(-3)
   print(LL)
   ```

   should print `[-3 -> -2 -> -1 -> '8' -> [8] -> [8] -> '8' -> 1 -> 2 -> 3 -> ]`.

   I would definitely use the corresponding magic method for the `Node` class.

10. Write a magic method so that we can print instances of `LinkedList` in a machine readable way. The easiest way for me to tell you how to format things is to give an example.

```
LL = LinkedList(['8', [8], [8], '8'])
LL.append(1)
LL.prepend(-1)
print(repr(LL))
```

should print `LinkedList([-1, '8', [8], [8], '8', 1])`.

11. Write the function `insert(self, data, idx)`.

Hopefully this function describes itself. It creates one new `Node`, and inserts it at the appropriate position. It should do this by appropriately updating `next` for two different instances of `Node`.

The output of

```
LL = LinkedList([-3, -2, -1, '8', [8], [8], '8', 1, 2, 3])
LL.insert(0,5)
print(LL)
```

should be `[-3 -> -2 -> -1 -> '8' -> [8] -> 0 -> [8] -> '8' -> 1 -> 2 -> 3 -> ]`.

Don't worry about when `idx` is greater than the length of the linked list.

However, the cases when `idx` is equal to `0` or to the length of the linked list require more care: it is easiest just to call `prepend` and `append`, respectively.

12. One test case... When I ran

```
LL = LinkedList(['8', [8], [8], '8'])
LL.append(1)
LL.append(2)
LL.prepend(-1)
LL.prepend(-2)
LL.insert(0,4)

print(len(LL), LL.first, LL.last)
print(LL)
print(repr(LL))
print(eval(repr(LL)) == LL)
```

I got

```
9 -2 2
[-2 -> -1 -> '8' -> [8] -> 0 -> [8] -> '8' -> 1 -> 2 -> ]
LinkedList([-2, -1, '8', [8], 0, [8], '8', 1, 2])
True
```