CSC148 Lab#4, winter 2016

learning goals

In this lab you will use iterative (looping) techniques and mutation (changing objects) to implement some methods for a LinkedList class. You may want to review the related work from our lecture materials.

You should work on these on your own before Thursday, and you are encouraged to then go to your lab where you can get guidance and feedback from your TA. There will be a short quiz during the last 15 minutes of the lab, based on these exercises.

set-up

Open file node.py in Pycharm, and save it under a sub-directory called csc148/Labs/lab4. It is worth setting your Pycharm project to this new directory — start a new project for directory lab4. This file declares both class LinkedListNode, to represent linked list nodes, and class LinkedList, to represent an entire linked list. There are also headers and docstrings for methods that you will implement, as well as some methods we have already implemented, to get you started.

We have commented-out method headers that you are to implement. You should uncomment these, one-by-one, as you work on them.

Once you have familiarized yourself with the __init__ and __str__ methods for class LinkedListNode and LinkedList, you are almost ready to proceed to the implementation of the methods below. But first, read the warnings below:

- **Draw lots of pictures!** You will need these to understand what your linked structures should look like before, during, and after operations where you change (mutate) those structures. If you skip the drawing, you are much more likely to mess up!

```
12 • —> 99 • —> 37 • —> X
```

These pictures are not just for beginners. Experienced programmers routinely draw pictures when they write code for linked structures.

- Be sure that you know exactly what attribute each part of your drawing represents. This will guide your code.

- Remember that methods that make updates to values traditionally return None. This is the case with our method append, for example. It may lead to surprises if you try to use this return value...

implement special method __setitem__

Read the docstring and examples for method __setitem__. They type for parameter index is int. You should adjust any negative index to an index in range by adding the size of the list (perhaps more than once). However, you should raise an IndexError if index is too large.

Before you write any code, decide what steps must occur and draw careful pictures of how your list should look before and after each step. Call over your TA to show her/him your drawings.

Once you’ve talked to your TA, you should implement the method. If it worked, you should be able to do something like:
>>> lnk = LinkedList()
>>> lnk.prepend(5)
>>> print(lnk)
5 ->|
>>> lnk[0] = 7
>>> print(lnk)
7 ->|

implement method _add_

Read the docstring and examples for method _add_. The aim is to provide a way of “adding” (concatenating) linked lists to produce a new one (the original lists are unchanged).

As usual, draw careful pictures of each step you carry out. Show your TA your drawings:

Once you've shown your pictures to your TA, you're ready to implement this method. You are allowed to use method append, if it helps. Once you are finished, you should be able to do things like this:

>>> lnk1 = LinkedList()
>>> lnk1.prepend(5)
>>> lnk2 = LinkedList()
>>> lnk2.prepend(7)
>>> print(lnk1 + lnk2)
5 -> 7 ->|
>>> print(lnk1)
5 ->|
>>> print(lnk2)
7 ->|

implement method insert_before

Read the docstring and examples for method insert_before. The aim is to be able to insert a new node with value v1 before the first occurrence of v2, if possible.

You will want to keep track of two nodes as you walk the list, which should look something like this:

```
while <some condition here>:
    prev_node = cur_node
    cur_node = cur_node.next
```

It is really easy to lose track of a reference between LinkedListNodes as you do this so... draw pictures.

Show your TA your diagrams and ideas, and then implement this method.

implement method delete_after

Read the docstring and examples for method delete_after. The aim is to be able to delete the node after the first occurrence of value, if such a node exists.

Again, it's really easy to mess up the updating of references, so you'll need pictures.

Show your TA your ideas before you begin implementing this method, then go ahead and implement it.
additional exercises

As usual, we have more exercises than we can fit into one lab. Work through as many unimplemented methods in node.py as you can.