CSC148 winter 2015
linked lists, iteration, mutation — week 8

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/W14/
416-978-5899

March 3, 2015
Outline

mutation
linked lists, two concepts

There are two useful, but different, ways of thinking of linked list structures:

1. as lists made up of an item (value) and a sub-list (rest)

   ![Diagram showing value and sub-list]

2. as objects (nodes) with a value and a reference to other similar objects

   ![Diagram showing linked list]

For now, will take the second point-of-view, and design a separate “wrapper” to represent a linked list as a whole.
class LLNode:
    '''Node to be used in linked list

    nxt: LLNode -- next node
        None iff we’re at end of list
    value: object --- data for current node
    '''

    def __init__(self, value, nxt=None):
        ''' (LLNode, object, LLNode) -> NoneType

        Create LLNode (self) with data value and successor nxt.
        '''
        self.value, self.nxt = value, nxt
a wrapper class for list

The list class keeps track of information about the entire list — such as its front, back, and size.

class LinkedList:
    '''Collection of LLNodes
    front: LLNode -- front of list
    back: LLNode -- back of list'''
    size: int -- size of this

def __init__(self):
    ''' (LinkedList) -> NoneType

    Create an empty linked list.
    ',
    self.front, self.back = None, None
    self.size = 0
division of labour

Most of the work of special methods is done by the nodes:

- `__repr__`
- `__str__`
- `__eq__`

Once these are done for nodes, it’s easy to do them for the entire list.
walking a list

Make a reference to (at least one) node, and move it along the list:

cur_node = self.front
while <some condition here...>:
    # do something here...
    cur_node = cur_node.nxt
_contains_

Check (possibly) every node

cur_node = self.front
while <some condition here...>:
    # do something here...
    if cur_node is None:
        return False
    cur_node = cur_node.nxt
    keep looking
    if value is what we're looking for:
        return True
walk through list
- report if find it
- False if you reach end of list

Linked List
size 2
front 15
back 9
_getitem_

Index outside \( \rightarrow \) too big

\( \text{neg: get index len (list) + neg amount.} \)

Should enable things like

```python
>>> print(lnk[0])
5
```

... or even

```python
>>> print(lnk[0:3])
5 \rightarrow 4 \rightarrow 3 \rightarrow |
```

won't implement slices - if you want, look at help(list)
append

We’ll need to change...

- last node
- former last node
- back
- size
- possibly front

draw pictures!
We need to find the second last node. Walk two references along the list.

```python
prev_node, cur_node = None, lnk.front
# walk along until cur_node is lnk.back
while <some condition>:
    prev_node = cur_node
    cur_node = cur_node.nxt
```