CSI33 Data Structures

Sharon Persinger

Fall 2019

Day 7 September 18

Python hook methods

- Python allows a programmer to write her own definitions for built-in Python functions and operations. You've already seen overloading of __lt__ in the Card class that allowed two Card objects to be compared using <. Other operators were also overloaded. The len function can be overloaded using __len__ and indexing with [] can be overloaded by defining __getitem__ nad __setitem__
- These new definitions for familiar syntax are called hook methods.
- They allow the syntax of using a newly defined class to look like familiar Python syntax. They allow Python's built-in methods like sort and max to work on newly defined classes.

Linked List ADT

- Write Python class that has the functionality of a Python list but using a linked list implementation.
  - Implement the methods needed for a Python list:
    - append
    - len
    - get item at an index
    - set item at an index
    - insert new item at an index
    - delete item at an index
    - pop item at an index
- We will need to count our way through the ListNodes to do this.

Use the LList2 class.

- Revised LList class definition in LList2.py posted.
- Let's experiment with it a bit to create some LinkedLists and use them like a Python list.

List Node class

- We will use the ListNode class as developed previously. We could add methods to it – accessors and mutators – but we won't since we will not interact with a ListNode directly, only by using methods of the LInkedList class.

```
class ListNode(object):


    def __init__(self, item = None, link = None):
        '''creates a ListNode with the specified data value and link
        post: creates a ListNode with the specified data value and link'''
        self.item = item
        self.link = link
```

LinkedList class invariant

1. The list has two instance variables – self.head and self.size.
2. self.size is the number of nodes in the list.
3. If self.size = 0, then the list is empty, and self.head is None. If the list is non-empty, self.head is a reference to the first or head ListNode object in the list.
4. The last ListNode object in the list, at position self.size -1, has it's link set to None. For every other ListNode in the list, the link refers to the next ListNode in the list.

- It is usual to manage a LinkedList through a reference to the head node object. Keeping an instance variable for the length makes a few methods easier.

LinkedList specification

```
# LList.py
from ListNode import ListNode

class LList(object):
    #--------------------------------------------------------
    def __init__(self, seq=()):
        """create an LList
        post: creates a list containing items in seq"""
    #--------------------------------------------------------
    def __len__(self):
        '''post: returns number of items in the list'''
    #--------------------------------------------------------
    def _find(self, position):
        '''private method that returns node that is at
         location position in the list (0 is first item,
         size-1 is last item)
        pre: 0 <= position < self.size
        post: returns the ListNode at the specified
         position in the list'''
    #--------------------------------------------------------
# #--------------------------------------------------------
    def append(self, x):
        '''appends x onto end of the list
        post: x is appended onto the end of the list'''
    #--------------------------------------------------------
    def __getitem__(self, position):
        '''return data item at location position
        pre: 0 <= position < size
        post: returns data item at the specified
         position'''
    #--------------------------------------------------------
    def __setitem__(self, position, value):
        '''set data item at location position to value
        pre: 0 <= position < self.size
        post: sets the data item at the specified
         position to value'''
    #--------------------------------------------------------
```

LinkedList specification, continued

```
#-----------------------------------------------------
    def __delitem__(self, position):
        '''delete item at location position from the
        list
        pre: 0 <= position < self.size
        post: the item at the specified position is
        removed from the list'''
    #-----------------------------------------------------
    def pop(self, i=None):
        '''returns and removes at position i from list;
        the default is to return and remove the last
        item

        pre: self.size > 0 and ((i is None or (0 <= i <
        self.size))
        post: if i is None, the last item in the list is
        removed and returned; otherwise the item
        at position i is removed and returned'''
```

```
#-----------------------------------------------------
    def insert(self, i, x):
        '''inserts x at position i in the list
        pre: 0 <= i <= self.size
        post: x is inserted into the list at position i
        and old elements from position i..oldsize-1
        are at positions i+1..newsize-1'''
    #-----------------------------------------------------
    def __copy__(self):
    '''post: returns a new LList object that is a
        shallow copy of self'''
#-----------------------------------------------------
```

Implementation of constructor

- Let's write a constructor that uses a procedure like the one we demonstrated in class with the example LinkedList that was built by linking together individual ListNode objects.
- That LinkedList was built by inserting ListNodes at the beginning or head of the list.
- We can traverse the parameter seq starting at the last element of the sequence instead of the first.

- First, let's write a helper function -
- Here's the specification:

def insertathead(self, data):

- '''creates a new ListNode with item data, and inserts it at the head of the LList
- post: new ListNode with data added at head of LList'''

Implementation of insertathead

```
def insertathead(self, data):
    '''post:  new ListNode with data
    added at head of LList'''
    n = ListNode(data, self.head)
    self.head = n
```

- Make a new ListNode with the data and link it to the current head of the list.
- Change the head of the list to be this new ListNode.

Constructor -

- Build the list up from seq by starting from the final item in seq and working to the first item.

__len__

- Return the value of the member variable

_find(self, position):

```
assert 0 <= position < self.size

    node = self.head
    # move forward until we reach the specified node
    for i in range(position):
        node = node.link
    return node
```

- Basically counting our way along the ListNodes.
- What is the meaning of
  - node = node.link ?
- This is an important LinkedList statement or operation.

Use _find(self, position) to write other methods

- def append(self, x):
- def __getitem__(self, position):
- def __setitem__(self, position, value):

- Used to return an element by indexing.
- Used to change an element by indexing.

_delete, delete, pop