CS 142
Finish Python Memory Model
Linked Lists

Rhodes College

Notes and figures adapted from Problem Solving with Algorithms and Data Structures
http://interactivepython.org/courselib/static/pythonds/BasicDS/linkedlists.html

Shallow Copies vs. Deep Copies

- Shallow copy – underlying objects point to the same memory address, but the memory address of object name is different
- Deep copy – completely separate copy creates both references and new data objects (if necessary) at all levels

```
>>> import copy
>>> b = [3, 4, [5, 6], 8]
>>> c = b
>>> d = copy.copy(b)     #creates a shallow copy
>>> e = copy.deepcopy(b) #creates a deep copy
>>> b
[3, 4, [5, 6], 8]
>>> c
[3, 4, [5, 6], 8]
>>> d
[3, 4, [5, 6], 8]
>>> e
[3, 4, [5, 6], 8]
>>> b is c, b == c
(True, True)
>>> b is d, b == d
(False, True)
>>> b is e, b == e
(False, True)
>>> id(b), id(c), id(d), id(e)
(43945416, 43945416, 44376352, 747400)
```


Shallow Copies vs. Deep Copies

```
>>> c[0] = 1
>>> b, c, d, e
([1, 4, [5, 6], 8], [1, 4, [5, 6], 8], [3, 4, [5, 6], 8], [3, 4, [5, 6], 8])
>>> d[0] = 2
>>> b, c, d, e
([1, 4, [5, 6], 8], [1, 4, [5, 6], 8], [2, 4, [5, 6], 8], [3, 4, [5, 6], 8])
>>> e[0] = 9
>>> b, c, d, e
([1, 4, [5, 6], 8], [1, 4, [5, 6], 8], [2, 4, [5, 6], 8], [9, 4, [5, 6], 8])
>>> d[2][0] = 2
>>> b, c, d, e
([1, 4, [2, 6], 8], [1, 4, [2, 6], 8], [2, 4, [2, 6], 8], [9, 4, [5, 6], 8])
>>> e[2][1] = 3
>>> b, c, d, e
([1, 4, [2, 6], 8], [1, 4, [2, 6], 8], [2, 4, [2, 6], 8], [9, 4, [5, 3], 8])
```


Passing Parameters

```
def foo(e, f, g):
    print(id(e), id(f), id(g))
    e += 3
    f.append(9)
    g = [4, 8]
    print(id(e), id(f), id(g))
    print(e, f, g)

def main():
    b = 0
    c = [1, 2, 3]
    d = [5, 6, 7]
    print(b, c, d)
    print(id(b), id(c), id(d))
    foo(b, c, d)
    print(id(b), id(c), id(d))
    print(b, c, d)

main()
```

Output:

```
0 [1, 2, 3] [5, 6, 7]
504610360 43353192 4989336
504610360 43353192 4989336
504610408 43353192 43379128
3 [1, 2, 3, 9] [4, 8]
504610360 43353192 4989336
0 [1, 2, 3, 9] [5, 6, 7]
```

Node Class

- Need to keep track of 2 pieces of data for each "node"
  - The list item (data being stored)
  - Reference to the next node
- Required methods
  - Constructor
  - Getters for 2 pieces of data
  - Setters for 2 pieces of data

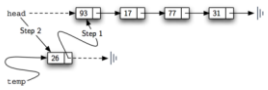
Unordered Linked Lists

- Creating an empty list

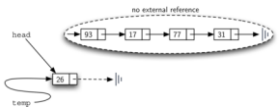
- Need method to check if list is empty
- Need a way to grow our list to add new nodes

Adding New Nodes

- 2 step process
  - 1. Create new node with link to first node in current list
  - 2. Updated head link to point to the new node

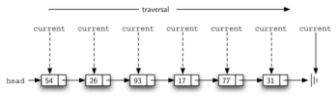


- Result of reversing the order of the 2 steps = BAD!



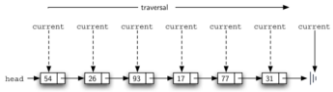
Traversing Thru a Linked List

- Traversal refers to the process of systematically visiting each node.
  - Use an external reference that starts at the first node in the list
  - As we visit each node, we move the reference to the next node by "traversing" the next reference.



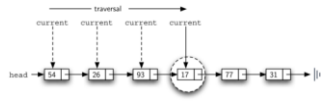
Size Method

- Returns total number of Nodes in the list
  - If list is empty, returns 0



Search Method

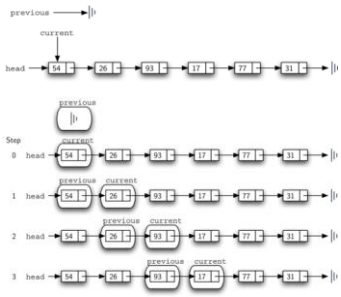
```
mylist.search(17)   #Returns True or False
```



Remove Method

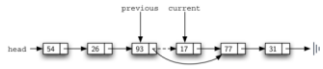
2 step process again

1. Find the node



Remove Method

2. Remove node from the linked list



Removing an Item from the Middle of the List

Removing the First Node from the List

Ordered Linked List

- What are the differences?
- How do we keep the list in order at all times?

Runtimes of Methods

Method	Linked List (Unordered)	Linked List (Ordered)
isEmpty	O(1)	O(1)
Add	O(1)	O(n)
Size	O(n)	O(n)
Search	O(n)	O(n)
Remove	O(n)	O(n)

Method	Python List
Append	O(1)
Insert	O(n)
Remove	O(n)
In (contains)	O(n)
Len	O(1)