CS2110 Lecture 11 Feb. 17, 2021

- HW3 due Monday, 10pm

Last time

- Looping with **for**
- Discussion of HW3 Q1
- Started Chapter 10: **list**s

Today

- A debugging example and some programming advice
- more Ch10, particularly the important property:
  - lists are *mutable*

It is very important to understand the consequences of list mutability. It can be confusing if you don't take time to understand it

A debugging example

```
def is_reverse(word1, word2):
    if len(word1) != len(word2):
        return False
    i = 0
    j = len(word2) - 1

    while j >= 0:
        if word1[i] != word2[j]:
            return False
        i = i + 1
        j = j - 1

    return True
```

is_reverse should return True if word1 is the reverse of word2.
I.e. is_reverse("abc", "cba") should return True while is_reverse("ab", "ab") should return False

Is code correct?

code in lec11.py

Programming advice

Be careful with variable names:

- Don't use ..index.. when it's bound to a value other an index!
- Don't change type of thing variable is bound to – use a different variable!

```
cost1 = 23.0
cost2 = 143.
for index1 in string1:                          <—   index1 is not an index
    index2 = 0
    while index2 < len(string2):
        if string1[index1] == string2[index2]:  <—   error here
            cost1 = "The cost is:" + str(cost1) <—   dangerous to change
        index2 = index2 + 1                             change type of
…                                                       object bound to var.
…                                                       cost1 was a number,
print(cost1)                                            now a string
if (cost1 < cost2):
    print("Option 1 is the better one!")                oops, error. Forgot
                                                        cost1 now a string
```

Ch 10: **list**s

- **list** is another Python sequence type
- In a string, each item of the sequence is a character
- In a list, each item can be a value of any type! (and can be as long as you want)
- The most basic way to create a **list** is to enclose a comma-separated series of values with brackets:

```
>>> [1, 'a', 2.4]
[1, 'a', 2.4]

>>> myList = [1, 'a', 2.4]
>>> len(myList)
 3
>>> myList[0]
1
```

[] operator and len() function work on both strings and lists

Ch 10: **lists**

I said the items in a list be any type. So, can lists be elements of lists? YES!

```
>>> myList = [1, 2, [‘a’, 3]]
>>> len(myList)
3
>>> myList[2]
[‘a’, 3]
>>> myList[2][1]
3
>>> myList[1][2]
Error
```

we call this a “nested list”

Ch 10: **list**s

A list can have no elements!

```
>>> myList = []
>>> len(myList)
0
>>> myList[0]
Error
```

we call this an "empty list"

Ch 10: list operations

slices, +, * work similarly to how they work on strings

```
>>> myList = [1, 2, 3, 4, 5]
>>> myList[1:3]
[2,3]
>>> myList + myList
[1,2,3,4,5,1,2,3,4,5]
>>> myList = myList + [6]
>>> myList
[1,2,3,4,5,6]
>>> myList = myList + 6
Error
>>> myList = myList + [[6]]
>>> myList
[1,2,3,4,5,6,[6]]
>>> 2 * myList
[1,2,3,4,5,6,[6],1,2,3,4,5,6,[6]]
```

Ch 10: lists are mutable!

- Strings are immutable. You can't change them.

```
>>> myString = 'hello'
>>> myString[0] = 'j'
```
← Error

- But lists are mutable! You can update lists

```
>>> myList = [1, 2, 'hello', 9]

>>> myList[1] = 53
 >>> myList
[1, 53, 'hello', 9]
```
you can replace a item in a list with a new value

```
>>> myList.append('goodbye')
>>> myList
[1, 53, 'hello', 9, 'goodbye']
```
you can add new items to the end of a list

```
>>> myList2 = [3, 99, 1, 4]
>>> myList2.sort()
>>> myList2
[1, 3, 4, 99]
```
you can even sort! Note: Python's sort rearranges the items directly within the given list. It doesn't yield a new list with same items in sorted order (different function**, sorted**, yields new sorted list)

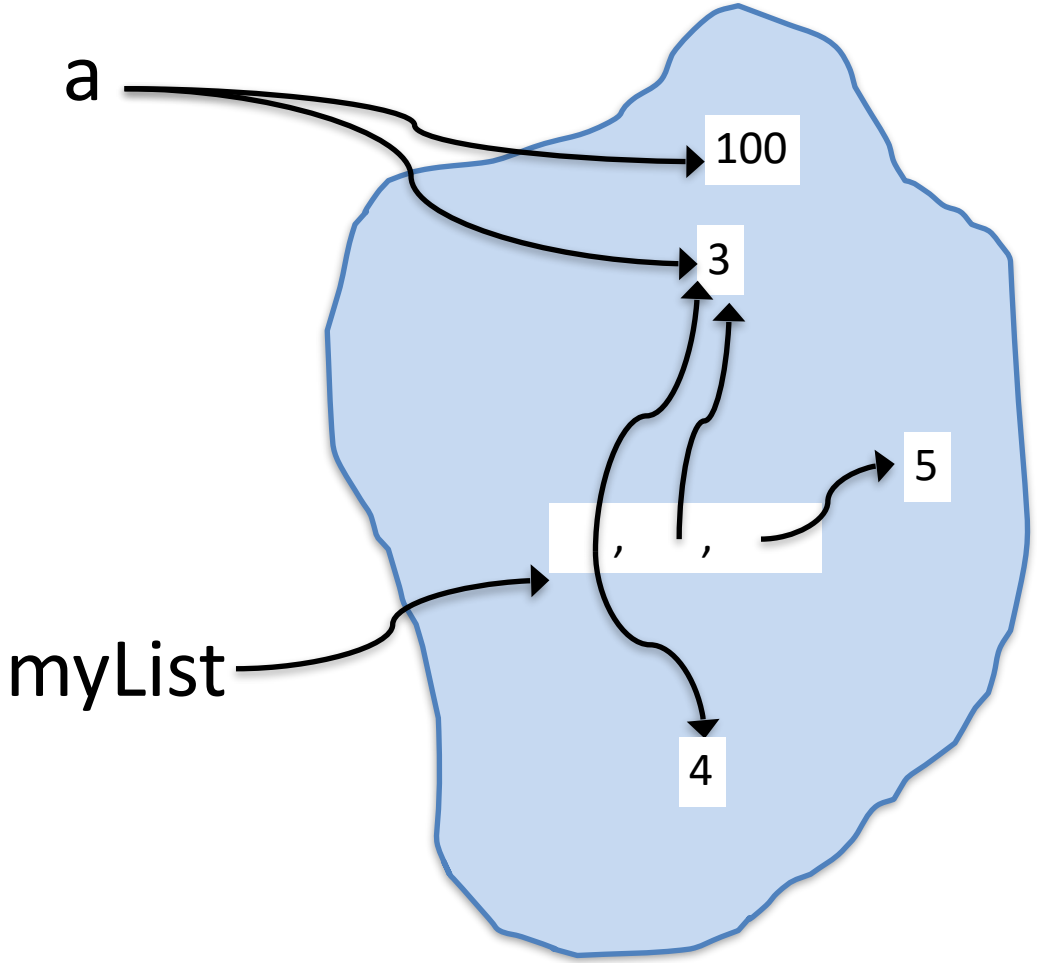
Examples: looping with lists lec11.py

- negativeListFrom(l)
- listOfBiggests(list1, list2)
- listOfBiggests2(list1, list2)
- getAverages(listOfLists)

List mutability

```
>>> a = 3
>>> myList = [a, a, 5]
>>> myList[0] = 4
>>> a = 100

>>>myList
???
```

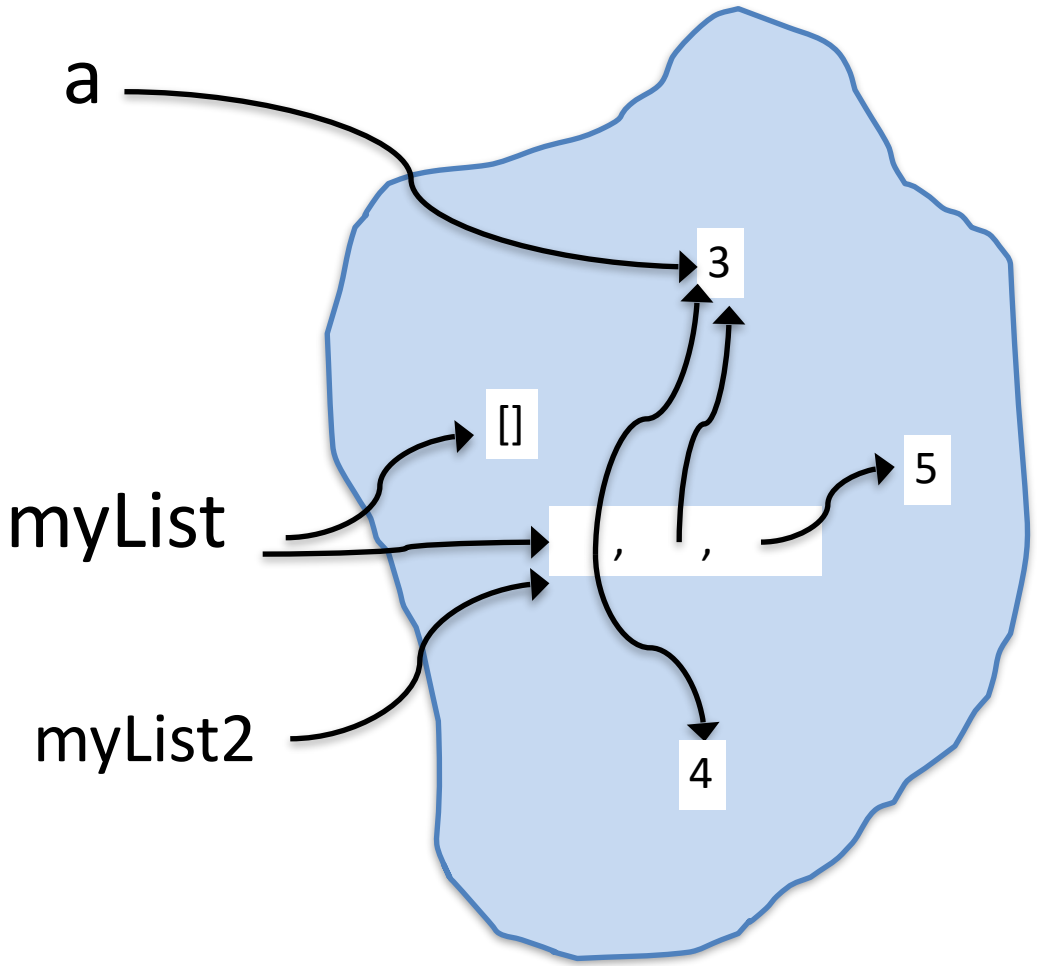


myList[0] = 4 *does not affect a's value!*

a = 100 does not affect list*!*

What happens here? Can you draw the updates?

```
>>> a = 3
>>> myList = [a, a, 5]
>>> myList2 = myList
>>> myList[0] = 4
>>>myList
???
[4, 3, 5]
>>>myList2
???
[4, 3, 5]
>>> myList = []
>>> myList
[]
>>> myList2
???
[4, 3, 5]
```



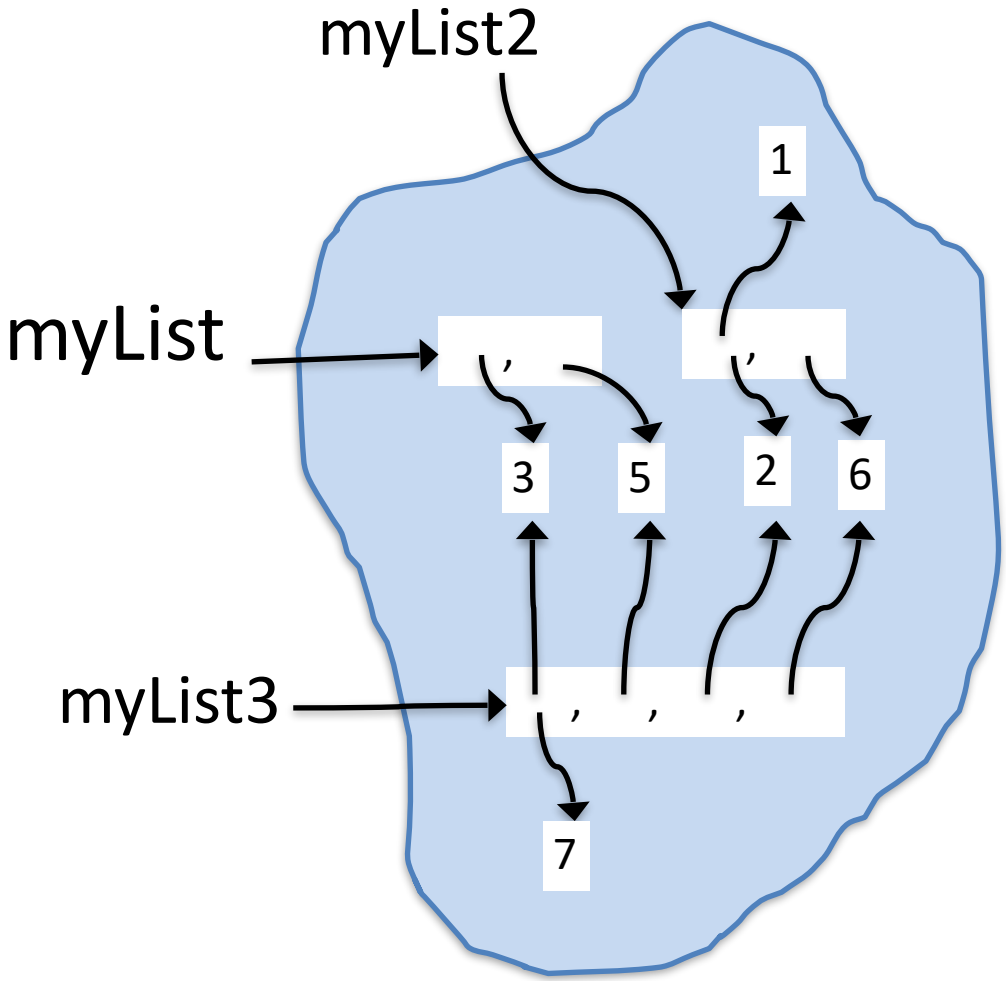
myList[0] = 4

- *does not affect a's value!*
- does *affect myList2's value*

VERY IMPORTANT! CAN BE CONFUSING!

list +

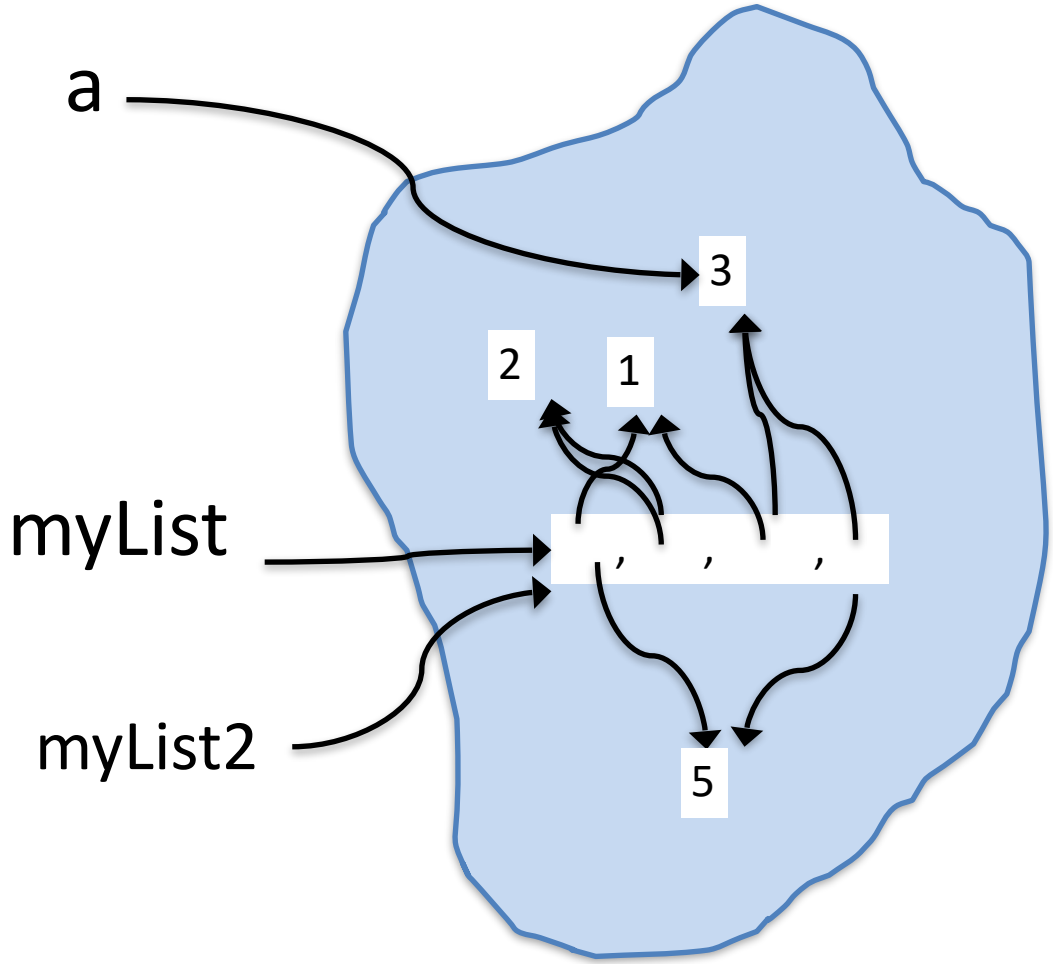
```
>>> myList = [3, 5]
>>> myList2 = [2, 6]
>>> myList3 = myList + myList2
>>> myList3
[3, 5, 2, 6]
>>> myList2[0] = 1
>>> myList3[0] = 7
>>> myList
?
>>> myList2
?
>>> myList3
?
```



IMPORTANT: + on lists yields a NEW list

append and sort

```
>>> a  = 3
>>> myList = [5, 2, 1]
>>> myList2 = myList
>>> myList.append(a)
>>> myList2.sort()
>>>myList
?
>>>myList2
???
```



SUPER IMPORTANT: unlike +, which does NOT modify the lists involved, append and sort MODIFY the list.

Ch 10: examples

- Write a function that takes two lists as input and returns a list of all pairs [i1, i2] where i1 in an item from the first list and i2 is an item from the second list pairs
  - e.g. [1,2] and [3,4,5] ->

    [[1,3], [1,4], [1,5], [2,3], [2,4], [2,5]]

- Write a function that, given a list of zero or more sublists of zero or more numbers, returns a list of numbers in which the ith number is the sum of the numbers in the ith sublist.
  - e.g [[2,3], [23], [1,1,1]] -> [5, 23, 3]

lec11b.py

Next Time

Finish Chapter 10

- more on list mutability
- *“aliasing”*
- lists as arguments to functions

List comprehensions