CS1210 Lecture 12 Sep. 20, 2021

- HW2 and DS3 scores have been posted
- HW3 due Friday
- DS4 is available, due 8pm Wed.
  - Discussion session attendance is not required but TAs will be there to discuss assignment and demonstrate wordInfo() function.

Last time

- Introduction to lists - Ch 10

Today

- For-while conversion
- More on lists: examples, and mutability and aliasing
- + vs append
- introduction to DS4

for -> while conversion

```
for var in sequence:
    …
    …
    …
```

```
index = 0
while index < len(sequence):
    var = sequence[index]
    …
    …
    …
    index = index + 1
```

Completely mechanical. No thought needed.

Body (the … lines) ***does not change***.

lec12forwhile.py

(last time) Ch 10: **lists**

- **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
>>> []
[]
>>> [1, ['a', 2, 'cat'], 3.0]
[1, ['a', 2, 'cat'], 3.0]
```

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

empty list - length is 0

"nested" list in which index 1 element is a list

(last time) 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]]
```

(last time) 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]

>>> myList.append('goodbye')
>>> myList
[1, 53, 'hello', 9, 'goodbye']
>>> myList = myList.append(3)

>>> myList2 = [3, 99, 1, 4]
>>> myList2.sort()
>>> myList2
[1, 3, 4, 99]
```

you can replace a item in a list with a new value

you can add new items to the end of a list

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 lec12.py

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

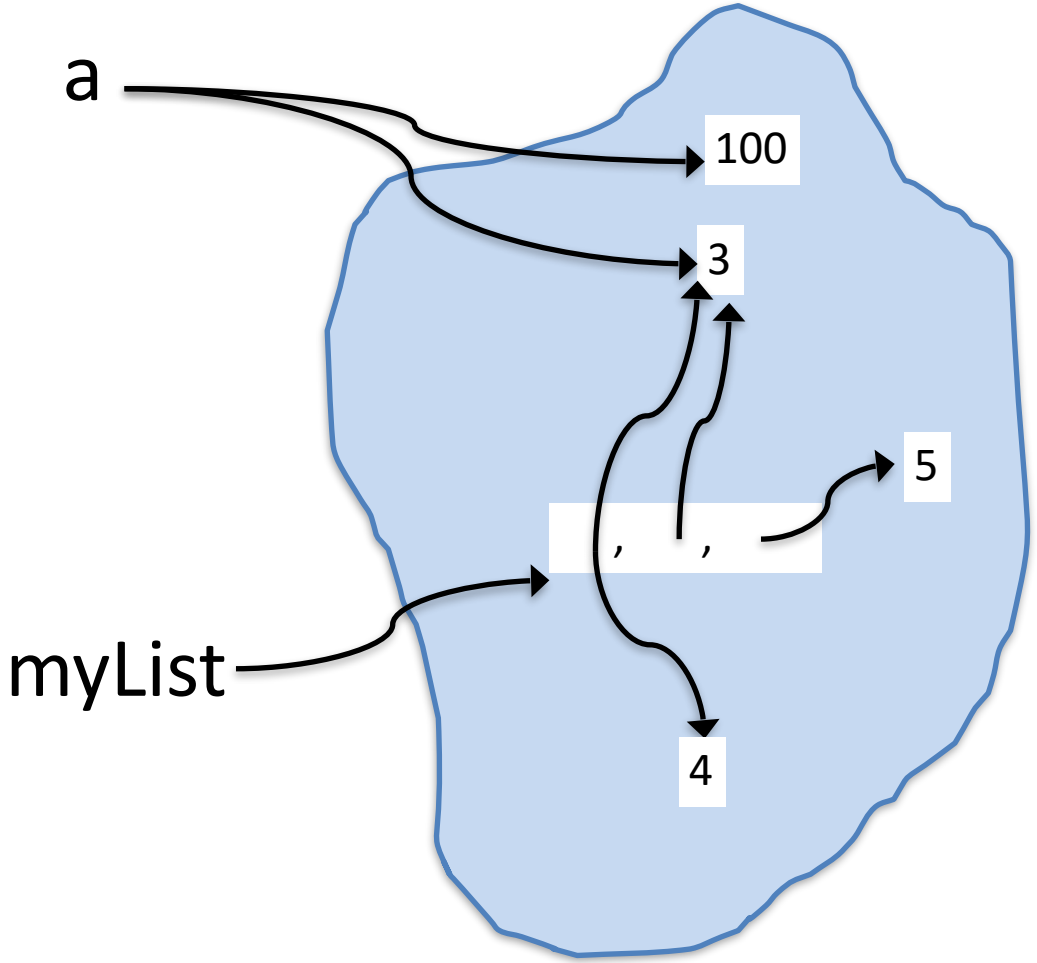
    [[1,3], [1,4], [1,5], [2,3], [2,4], [2,5]]

- 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]

List mutability

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

>>>myList
???
```

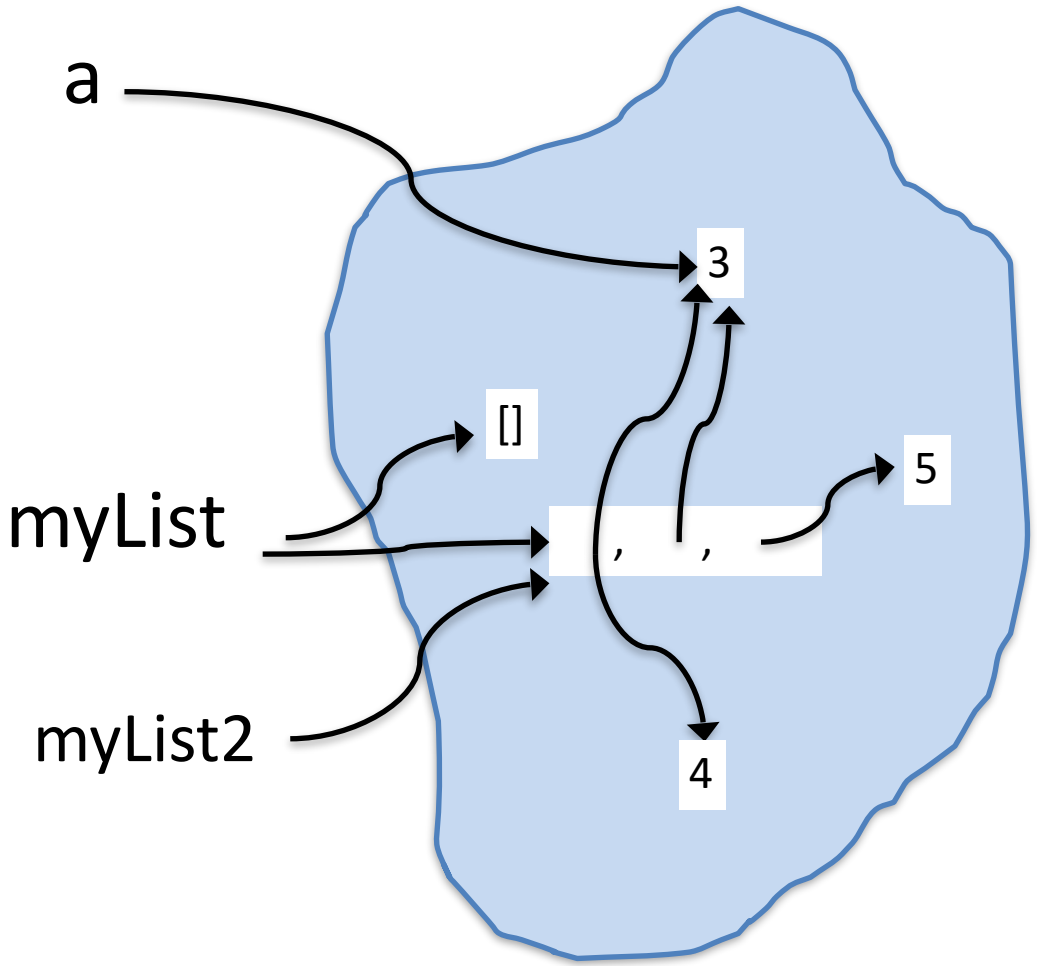


myList[0] = 4 *does not affect a's value!*

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

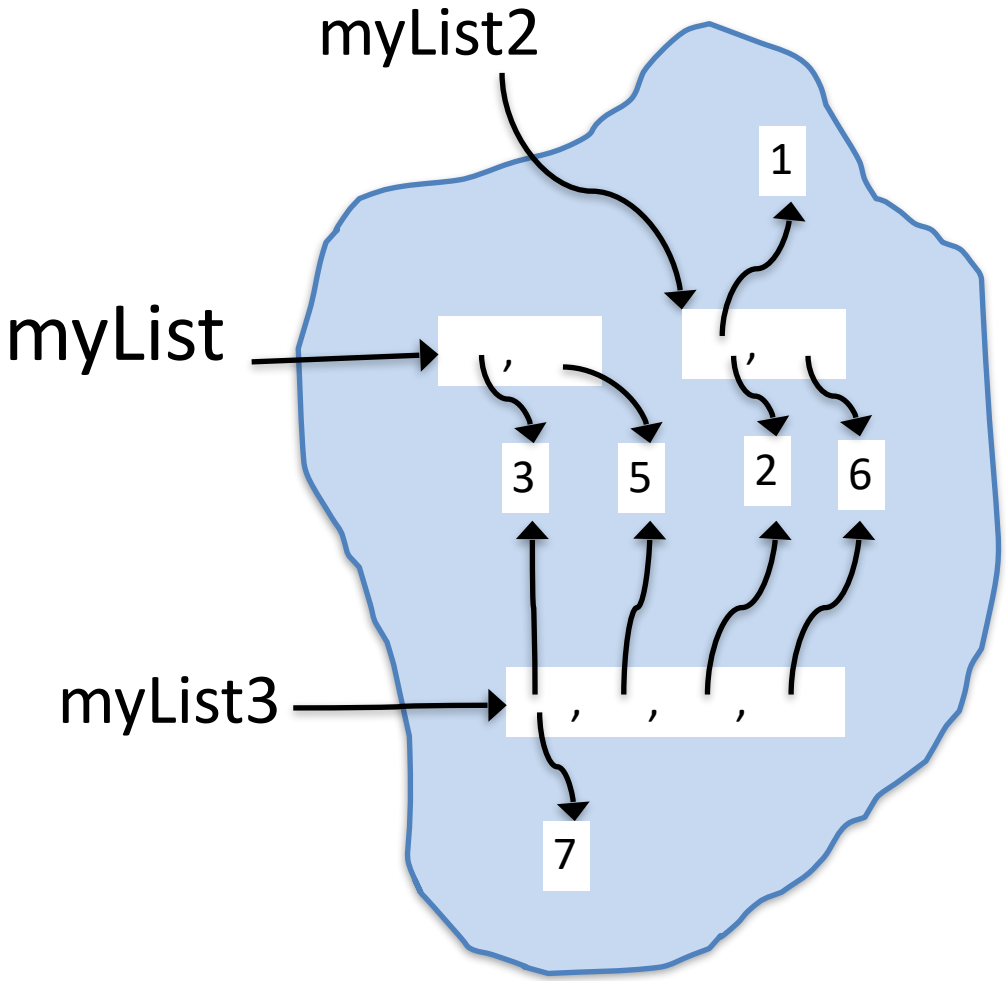
- *does not affect a's value!*
- does *affect myList2's value*

VERY IMPORTANT! CAN BE CONFUSING!

This is called **aliasing** – two or more variables referring to same mutable object

list +

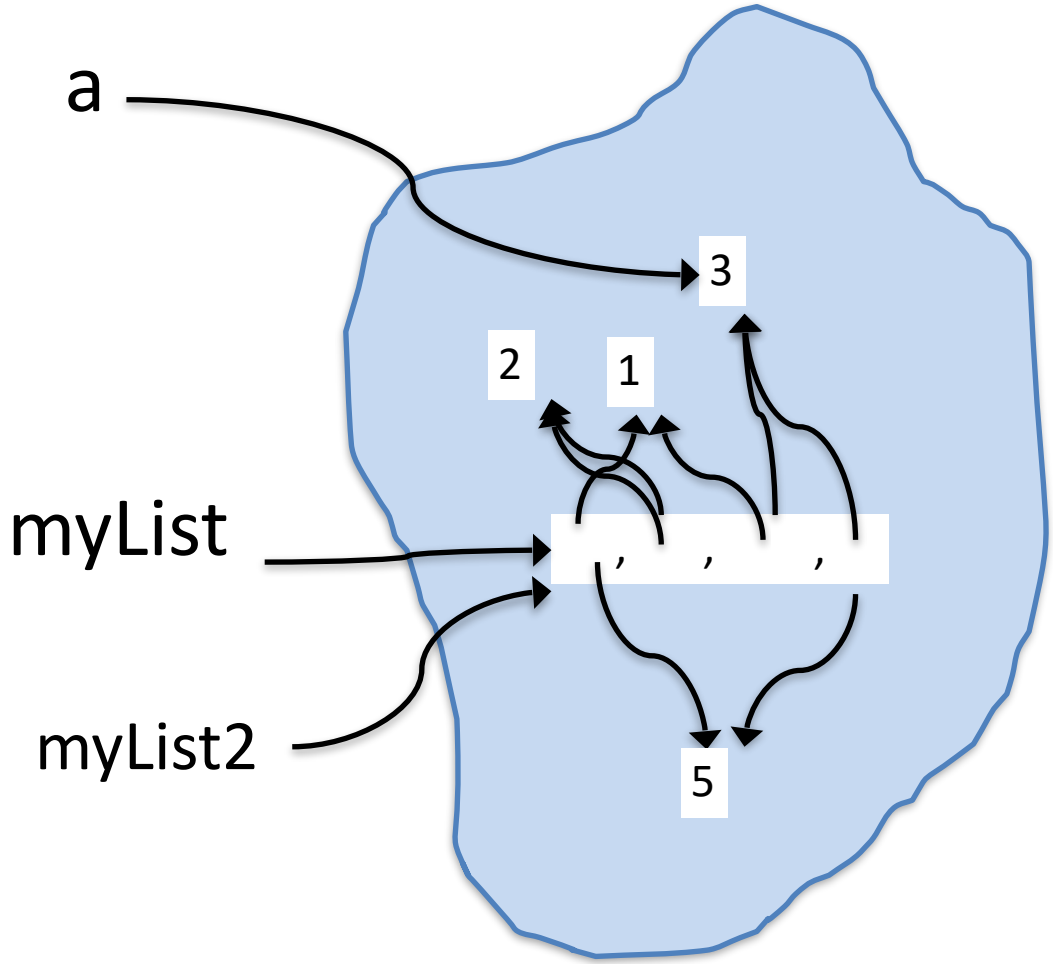
```
>>> myList = [3, 5]
>>> myList2 = [2, 6]
>>> myList3 = myList + myList2
>>> myList3
[3, 5, 2, 6]
>>> myList2[0] = 1
>>> myList3[0] = 7
>>> myList
?
>>> myList2
?
>>> myList3
?
```



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.

list + *vs.* append

```
result = []
for num in range(100000):
    result = result + [num*num]
```

```
result = []
for num in range(100000):
    result.append(num*num)
```

Is either one better?

lec12append.py

Discussion section 4

- Will work with files of many words and write code to find sets of anagrams (words with same letters but different order). E.g. art, rat, tar
- What if we wanted to find the largest set of anagrams?
  - simple direct approach

```
biggestAnagramList = []
for word in wordList:
    anagramList = getAnagramsOf(word, wordList)
    if len(anagramList) > len(biggestAnagramList):
        biggestAnagramList = anagramList
```

  - Works okay for a couple thousand words (words5.txt) but far too slow for 100+K list like wordsMany.txt
  - Problem to think about: can you efficiently find largest anagram set?

biggestAnagramSet.py

Next Time

- Finish lists
  - is operator and object identify (vs ==)
- review of HW3, Q1 hints
- One more part of Ch 10 – 10.22, list comprehensions
- Start Ch 12 - dictionaries