CS2110 Lecture 13 Feb. 22, 2021

- HW3 due tonight
- DS4 due tomorrow night
- HW4 will be available this afternoon, due next Thursday, Mar. 4.

Last time

- Continued Ch 10: for -> while conversion, more list mutability examples and diagrams, list + vs append, == vs. is, list copies

Today

- Info about course grades
- Related to DS4: largest anagram set
- Mutability of lists when passed as arguments to function
- Functions with **side effects**
- Start Ch 12, dictionaries

About course grades

People ask if course is “curved”. Answer: no and yes

1. The “no” part: if everyone does excellent work and demonstrates mastery of all the material, then I am happy to give everyone an A.
2. There are no pre-set point/percentage -> course grade mappings like many of your are familiar with from high school (e.g. A for 90-100, B for 80-90, etc.). Instead, based on my sense of how well students mastered the material, I map score ranges to grades. Exam averages are often relatively low in a class like this - 60-75%. People who get A’s in the class generally do well on all the homework 80-100%, and 80% or more on exams. (Likely approximate ranges: A 85+, B 70+, C 50-something+, D 40-something+)
3. The “yes” part: the College of Liberal Arts has some guidelines about grade distribution (15% A, 34% B, 40%C, etc.). This is not a rule. Item 1 above takes precedence! I can give all A’s (or F’s). You are *not* competing with other students. Your job is to master the material. However, it *is* often the case that the grade distribution “curve” fits the CLAS percentages approximately well. That’s simply because 10-20% of the people do very well, 30-40% do well, etc. Many people who get Cs/Ds skip homeworks, don’t put a lot of effort into them, etc.

After the second quiz, will provide specific numbers/info on where people stand so far

Discussion section 4 example

What if we wanted to find the largest set of anagrams?

- Simple direct approach:

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

  Works okay for a couple thousand words (word5.txt) but too slow for large word sets like wordsMany.txt

- Much faster approach: we'll look at this in detail in a couple weeks when we discuss algorithm analysis but, for now, the idea:

1. associate a "key" with each word, the sorted version of that word. E.g. ["art", "art"] … ["least", "aelst"] … ["rat", "art"] … ["stale", "aelst"] … ["tar", "art"]
2. Sort this list of pairs by those "keys". Now all anagrams are neighbors in this sorted list and the largest set can be found via one simple scan through it. [.... ["least", "aelst"], ["stale", "aelst"], ..., ["art", "art"], ["rat", "art"], ["tar", "art"] ...]

(last time) Objects, equality, and identity

There is an operator in Python called **is**

```
>>> x is y
```

True if x and y refer to same object (in computer memory), False otherwise.

You don't often need to use **is** but you should be aware of when two variables refers to the same *mutable object.* This is called **aliasing.**

As we've seen:

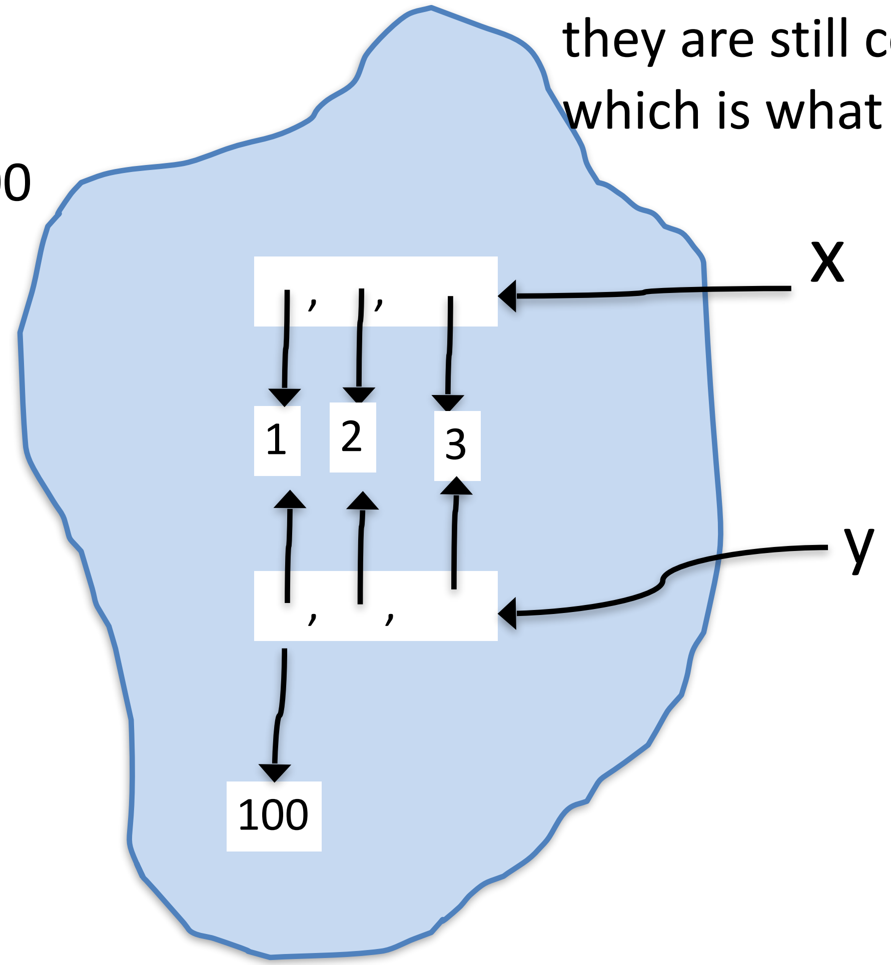
```
>>> x = [1,2,3]
>>> y = x
>>> x is y
True
>>> x[1] = 100
>>> y[1]
?
```

y and x are aliases for the same list object

(last time) Objects, equality, and identity

```
>>> x = [1, 2, 3]
>>> y = [1, 2, 3]
>>> x is y
False
>>> x == y
True
>>> y[0] = 100
>>> x
???
```

constructs a list containing 1, 2, 3
constructs a (new/different) list
x, y are not aliases
they are bound to different objects
they are still consider equal, though,
which is what you usually care about



(last time) Objects, equality, and identity

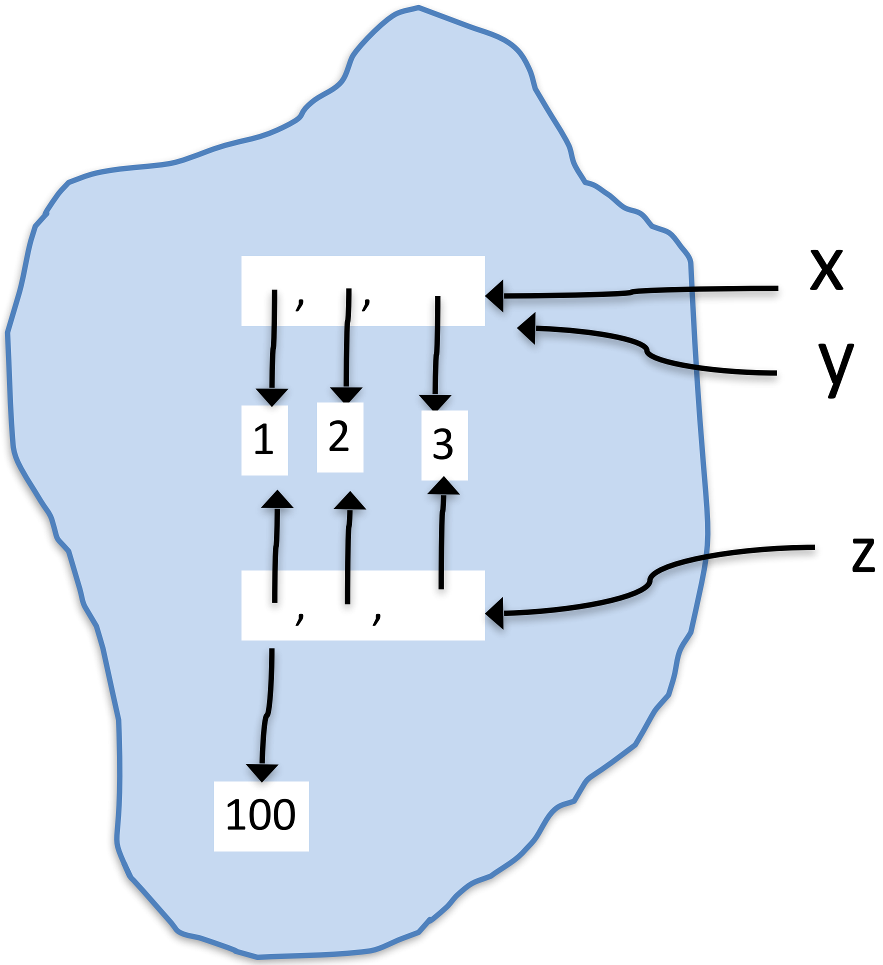
Often, we want to avoid aliasing. So, given a list, can we easily make a copy? YES!

```
>>> x = [1, 2, 3]
>>> y = x
>>> z = x[:]

>>> x is y
True
>>> x is z
False
>>> x == y
True
>>> x == z
True
>>> z[0] = 100
>>> y[0] = 50
>>> x
?
>>> y
?
```

range[:] is “full range” so a new list with all the elements of the original

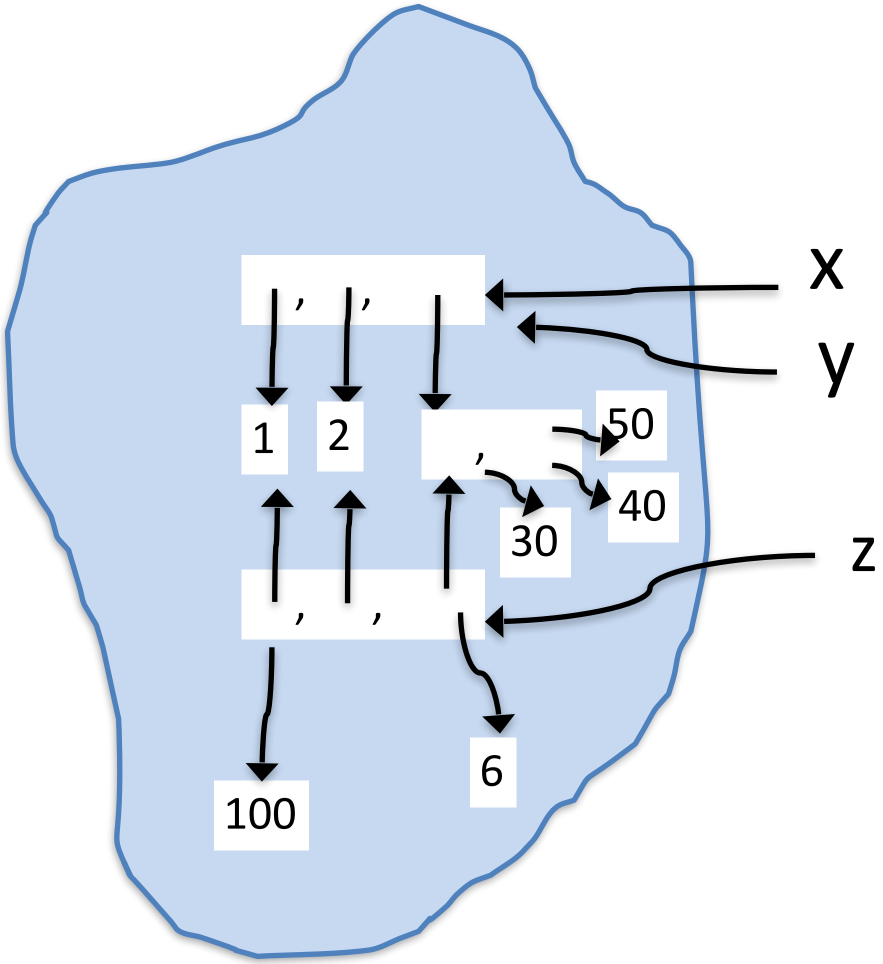
(last time) Objects, equality, and identity

```
>>> x = [1, 2, 3]
>>> y = x
>>> z = x[:]

>>> z[0] = 100
>>> x
???
```



Objects, equality, and identity

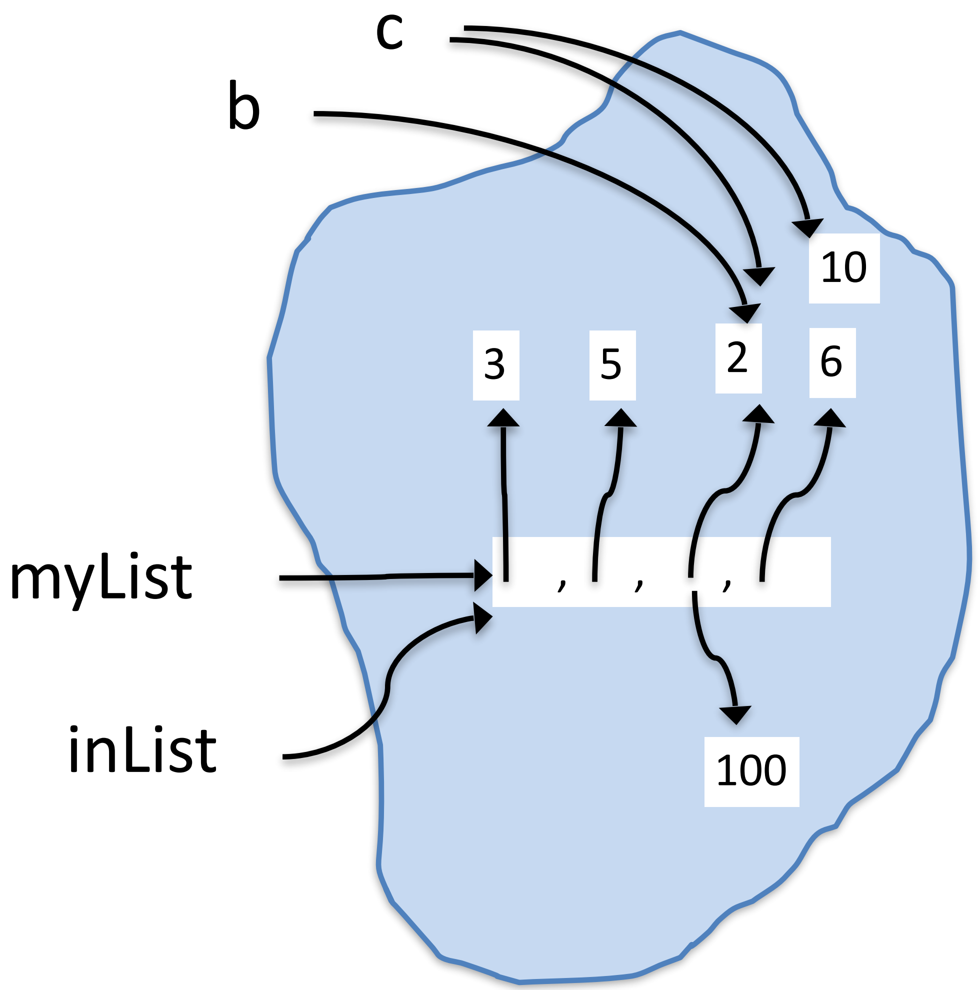
```
>>> x = [1, 2, [30, 40]]
>>> y = x
>>> z = x[:]

>>> z[0] = 100
>>> z[2][1] = 50
>>> x
???
>>> y
???
>>> Z[2] = 6
>>> z
???
>>>x
???
```



[:] is a *shallow* copy. There are ways to do *deep* copy (maybe we will discuss later in the semester)

Mutability and arguments to functions

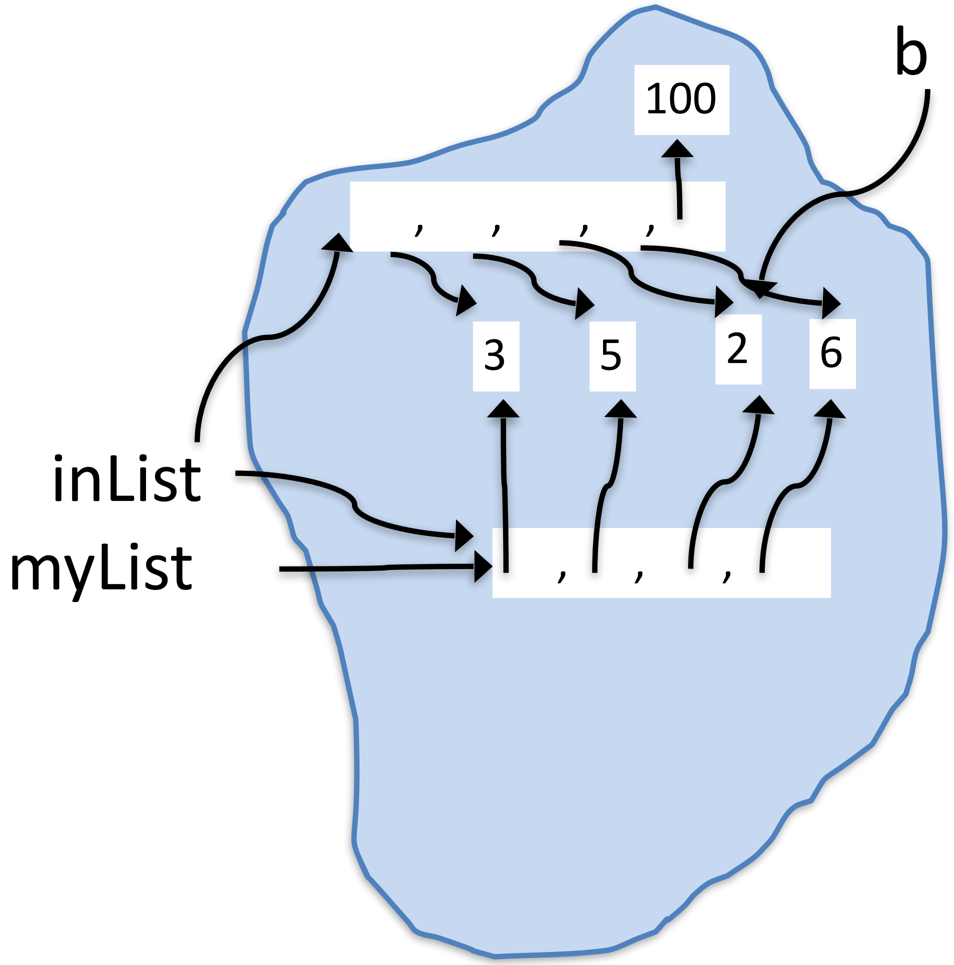
```
>>> def foo(inList, c)
...         inList[2] = 100
...         c = 10
>>>
>>> b = 2
>>> myList = [3, 5, b, 6]
>>> foo(myList, b)
>>> myList
[3, 5, 100, 6]
```



But what if body of foo is instead: inList = inList + [10]?

Mutability and arguments to functions

```
>>> def foo2(inList)
...         inList = inList + [100]
>>>
>>> b = 2
>>> myList = [3, 5, b, 6]
>>> foo2(myList)
>>> myList
[3, 5, 2, 6]
>>> b
```



Advice/comments on functions

- Some functions compute something and return a value without *side effects*. That is, they do any output and don't change the values of any objects that exist outside of the execution of that function.
- Other functions do have side effects. They either print something (or affect GUI elements) or change values of objects that exist outside the function execution. Such functions often don't return anything. And such functions can maybe helpfully be thought of as commands.

```
# return new list that is like inList
# but without 1st and last elements
def middle(inList):
    return inList[1:len(inList)-1]
```

```
# remove the first and last
# elements from inList
def chop(inList)
    del inList[0]
    del inList[len(inList)-1]
```

We use these differently.
Consider:

```
def bar(inList):
    …
    middle(inList)
    …
    …
```

What can you say about this?

And

```
def baz(inList):
    …
    chop(inList)
    …
    ..
```

And how about this?

Look at the code in lec13.py and make sure you understand the differences between bar, bar2, and baz

Thoughts on this function?

```
~~# Given a list of 1 or more numbers or strings, return the smallest~~
~~# and largest items.~~
~~#~~
~~def getSmallestAndBiggest(listOfStuff):~~
~~        listOfStuff.sort()~~
~~        return (listOfStuff[0], listOfStuff[-1])~~
```

Functions should not modify their input unless the function specification explicitly says to do so!

```
# Given a list of 1 or more numbers or
# strings, return the smallest and largest
# items.
#
def getSmallestAndBiggest(listOfStuff):
        sortedCopyOfList = sorted(listOfStuff)
        return (sortedCopyOfList[0], sortedCopyOfList[-1])
```

```
# Given a list of 1 or more numbers or
# strings, sort the list and return the smallest and
# largest items.
#
def sortAndGetSmallestAndBiggest(listOfStuff):
        listOfStuff.sort()
        return (listOfStuff[0], listOfStuff[-1])
```

Chapter 12: Dictionaries

- Python supports the extremely useful **dictionary** ‘dict’ type in Python
- Dictionaries are:
  - collections of key – value pairs
- Similar to but importantly different from lists
  - could think of lists as *ordered* collection of key-value pairs, where the keys are integers 0, 1, 2, …
  - with dictionaries, the collection is *unordered* but the interesting thing is that the *keys can be any immutable values*
  - *E.g.* create dictionary numlegs

    >>> numlegs = { ‘frog’: 4, ‘human’: 2, ‘ant’:6, ‘dog’:4}

Dictionaries

- create with { k1:v1, k2:v2, …}
- empty dictionary: {}
- retrieve value: dict[key]
- modify (or insert) value for key: dict[key]=value

- one important feature of dictionaries is that they provide *very fast* access (we might discuss *how* later in term) to values associated with keys despite being more flexible (not restricted to integer keys, etc.) than lists (demo: dicttest.py for speed comparison with lists)

Dictionary operations

- len(d)
- d.keys()
- d.values()
- k in d
- del d[k]
- for key in dict:
- d.get(key, defaultVal) when you don't want possible KeyError for d[key]

But note: no slice – d[key1:key2] doesn't make sense

Next Time

More on dictionaries, Ch 12