CS1 Lecture 10 Feb. 8, 2017

- HW3 available,due next Monday, 9:00am
  - Q1 is similar to HW2 Q3. More complex but still can be done with simple loops, if/else logic, and comparisons.
- 3rd survey available. Please complete them
- HW1 grading done. Email me questions/concerns
- Comment on discussion section work

Last time

Chapter 8:
- Strings are sequences
- Iteration with **for**
- string slices
- Strings are immutable
- Exercise 8-4

Today

Finish chapter 8:
- String methods
- **in** operator
- example from chapter 8 debugging section

Begin Ch 10 - Lists

Ch 8: string **methods**

- We use strings a lot in Python. Python provides many special built-in functions, called **methods**, for strings.
- Methods are called/invoked using a different syntax, dot notation (some people find it confusing):

```
>>> 'abcd'.upper()
'ABCD'
```

invokes the built-in string upper function

NOTE: You can think of it as upper('abcd')

- There are a quite a few: Look them up – I won't go over many of them.

Ch 8: string **methods**

```
>>> myString = ‘hello’
>>> myString.count(‘l’)
2
>>> ‘ababcab’.count(‘ab’)
3
>>> ‘eeeeeee’.count(‘ee’)
?
```

Again, you can think of it as: count(myString, ‘l’)

```
>>> myString.index(‘l’)
2
>>> myString.find(‘l’)
2
```

index and find nearly the same but look up in docs! (give different result when not found)

Ch 8: string **methods**

```
>>> 'This is a sentence.'.split()
['This', 'is', 'a', 'sentence.’]
```

*a **list** (we’ll study next)*

```
>>> '1,2,104,7,12'.split(',')
['1', '2', '104', '7', '12']
```

```
>>> '    non-whitespace  '.strip()
'non-whitespace’
```

```
>>> '.'.join(['www', 'uiowa', 'edu'])
'www.uiowa.edu’
```

Note: these three are very commonly used when reading in data from files

Ch 8: string **in** operator

- ‘a’ in myString returns True if ‘a’ is in myString, False otherwise

Write function inBoth(string1, string2) that prints all characters that appear in both:

```
def inBoth(string1, string2):
    for c in string1:
        if c in string2:
            print(c)
```

Example from Ch 8 “Debugging” section

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

    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?

Ch 9: good string exercises

- Several string exercises in the form of word puzzles
- Simple introduction to opening and reading in text from files.

```
fileStream = fopen('words.txt')
for line in fileStream:
    word = line.strip()
    print(word)
```

Yesterday and today's discussion sections cover a bit of Ch 9. But you should do more of the exercises than discussion section time allows. *Do them all if you can – very good practice!*

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
```

[] 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!

we call this a “nested list”

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

Ch 10: **lists**

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

You can “traverse” lists with **for**

```
for number in l:
    if number < 0:
        print(“negative”)
    else:
        print(“not negative”)
```

Next time

The rest of Ch 10. Much of it is related to important property of lists:

- 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!

Exercise to think about. Lists make it easy to generalize the printVowelStats(inputString) function of HW2. How would you implement printLetterCounts(inputString, letters) that prints the number of occurrences in inputString of each letter in letters

```
>>> printLetterCounts("This is a sentence containing a variety of letters", "aeiouy")
'This is a sentence containing a variety of letters' has:
        4 'a's
        6 'e's
        5 'i's
        2 'o's
        0 'u's
        1 'y's
        and 32 other letters
```