CS2110 Lecture 15 Feb. 26, 2021

- "DS5" assignment available this evening. Due Monday (there are no discussion sections on Tuesday - it's a "no instruction" day). A dictionary problem easier than the ones on HW4
- HW 4 due Thursday
- Quiz 2 next Friday

Last time

- Dictionaries - Ch 12

Today

- DS5 info
- HW4
- Exercises from last time, plus a few variants
- Tuples and tuple assignment

Background examples for this discussion section assignment

- birdDict.py example
- How would you implement printLetterCounts(inputString, letters) that prints the number of occurrences in inputString of each letter in letters? E.g

```
>>> 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 letter
```

list version: letterCountsWLists.py
In discussion section assignment
you will redo this with dictionaries

HW4

It is interesting, and not hard if you do a little bit at a time. Get it working bit by bit.

1. Read the file, storing all the messages and their labels (spam/ham). E.g.
   - Two separate lists: ham list [[‘text’, ‘me’, ‘later!’], [‘…’, …], ...] and spam list [[‘call’, ‘1412’, ‘to’, ‘win’], …] *(I recommend this option)*
   - Or one list [[‘spam’, [‘call’, ‘1412’, ‘to’, ‘win’]], [‘ham’, [‘text’, ‘me’, ‘later!’]], […], …]
   - Note: don’t keep ham/spam label/tag as part of message. I’ve seen people do this and then write special case code to ignore ‘ham’/’spam’ when processing message words in step 2 below – this can yield errors.
2. Create a ham and a spam dictionary. For each message, extract its words, and update spam or ham dictionary of word counts accordingly
   - for ‘text me later!’ increment ‘text’, ‘me’, ‘later’ entries in ham dict
3. Use the two dictionaries to compute and print some statistics
   - get total spam/ham word counts and unique word counts
   - extract most common words from dictionaries
   - print stats

HW4

1. File has some non-Ascii characters.

   use: open(fileName, encoding = 'utf-8')

2. To break line into tokens – individual elements of a line, learn how to use string **split**

   for line in fileStream:

   lineAsList = line.split()

   lec15split.py

3. get rid of extra stuff "…cool!?" learn how to use string **strip** (and/or lstrip, rstrip)

   - I *strongly recommend against* using replace() method
   - *Don't put "" (empty string) as a word in your dictionaries*

for HW4, sorting is very helpful

Why?

You'll have two dictionaries of the form:
{'free': 23, 'you': 50, 'go': 10, 'zoo': 1}

You'll need to extract words in order from most to least frequent?

sort/sorted "work" on dictionaries but do they do what we want?

```
>>> d = {'free': 23, 'you': 50, 'go': 10, 'zoo': 1}
>>> sorted(d)
['free', 'go', 'you', 'zoo']
```

helpful???

In Monday's lecture, will show how to "sort" dictionary in the way you need

A few little exercises

1. Given a list of numbers, find the pair with greatest difference ✔
2. Given a list of numbers find the pair with smallest difference ✔
3. Given a list of numbers and a target number (call it k), find two numbers (if they exist) in the list that sum to k

lec14exercisesB.py

Small variants of/questions about third problem

4. Suppose:
   - No dictionaries allowed/available
   - Numbers in lists are know to have limited magnitude. E.g. all numbers between 0 and 10000

   Fast solution?
5. Modify findKPairFast to provide indices/location of found pair in list
6. Question: if we generate a list of, say, 10,000 random numbers between -1,000,000 and 1,000,000 how likely is it that the list contains a pair that sums to k for any k in, say, 0…999?

lec14exercisesB.py

Ch 10.26

- Another sequence type (you've seen list, string, range so far) is **tuple**
- Tuples are just like lists except they are *immutable*!
- Create by typing a comma-separated list of values
  - Standard practice is to enclose the list in parentheses (but that's not required)

```
>>> myTuple = (1,2,3)
>>> myList = [1,2,3]
>>> len(myTuple)
3
>>> myList[0] = 4
>>> myList
[4, 2, 3]
>>> myTuple[0] = 4
…
TypeError: 'tuple' object does not support item assignment
>>> newList = list(myTuple)
>>> newList
[1, 2, 3]
```

Tuples

```
>>> myTuple = 10, 11, 12
>>> myTuple
(10, 11, 12)
>>> myTuple = (3)
>>> myTuple
3
>>> myTuple = (3,)
>>> myTuple
(3,)
>>> myTuple = ()
```

NO! This is just 3

Tuple of length 1

Empty tuple – length 0

Adding tuples works:

```
>>> myTuple = (1,2,3)
>>> myTuple + myTuple
(1,2,3,1,2,3)
```

But append and other operations that mutate lists do not

Tuples

You don't *need* tuples. You could use lists anywhere instead. But the immutability is sometimes desirable.

E.g. sometimes good to pass tuples as arguments to functions. The function can't then (accidentally or purposely) modify your data!

Easy to create a tuple from a list:

```
>>> importantList = [10, 11, 12]
>>> safeTuple = tuple(importantList)
>>> safeTuple
(10, 11, 12)
```

Tuple assignment

Some of you have learned you can write things like:

```
>>> a, b = 1, 2                    (or even (a,b) = (1,2))
>>> a
1
>>> b
2
```

This is called tuple assignment (even though you don't really need to think about tuples here)

Tuple assignment

Useful for swapping values

```
>>> a, b = 1, 2
>>> a = b
>>> b = a
```

Does not correctly swap!

```
>>> temp = a
>>> a = b
>>> b = temp
```

Does correctly swap.

But so does

```
>>> a, b = b, a
          2, 1
    a, b = 2, 1

>>> a
2
>>> b
1
```

Why does this work?

Remember our old rule:

1. evaluate right hand side
2. update values var names refer t

Tuple assignment

Also useful for “unpacking” results from functions that return list or tuple of results

```
def twiceAndThriceN(n):
        return (2*n, 3*n)                              [2*n, 3*n] also ok

>>> twoN, threeN = twiceAndThriceN(5)
>>> twoN
10
>>> threeN
15
>>> result = twiceAndThriceN(5)              also works, of course
>>> result
(10, 15)
```

Next Time

- HW4 Information and advice
  - zip and sorting for HW4
- default and keyword arguments to functions
- conditional expressions