CS1210 Lecture 15 Sep. 27, 2021

- Quiz 1 scores available
  - Option to replace Quiz 1 score with Quiz 2 score (up to 18 points), essentially counting Quiz 2 twice
- Quiz 2, Wednesday, Oct. 6, in class
- HW4 and DS5 are available:
  - DS 5 due Wednesday, 8pm. Discussion section attendance is not required.
  - HW 4 due Tuesday, Oct. 5, 8pm

Last time

- Quick introduction to conditional expressions and list comprehensions (10.22). Very useful and you'll see them a lot in some Python code but you are not required to know them for quizzes.
- Started to introduce dictionaries before Internet troubles led to ending the class early.

Today

- Dictionaries - Ch 12
- Background examples for DS 5 assignment
- A few little exercises
- Overview of HW 4

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

Looping over dictionaries with for

If we have a dictionary, d, of names as keys and ages as values, we can compute averages as follows:

```
averageAge = 0
sumOfAges = 0
for nameKey in d:
   sumOfAges += d[nameKey]
if sumOfAges != 0:
   averageAge = sumOfAges / len(d)
print("The average age is: ", averageAge)
```

lec15.py

A dictionary example

- Text file with info about people – name, birth year, favorite color, weight, home city, home country
- Read and store in dictionary
  - Name as key
  - Subdictionary (and sub-sub-dictionary) for other properties

```
{'birthyear': 1980,
 'favcolor': 'red',
 ...,
 'home': {'city': 'Tokyo', 'country': 'Japan'}
}
```

- Add simple password handling, storing "hash" in dict

Files: ppldata.py, people.text

Related news

- 2015 Turing Award winners: https://www.theguardian.com/science/2016/mar/01/turing-award-whitfield-diffie-martin-hellman-online-commerce
- http://amturing.acm.org/byyear.cfm
- Remember printFirstNPrimes problem? Finding prime factors of big numbers is super Important for cryptography. Internet security depends hugely on the fact that there is no known way to find factors of very large numbers quickly

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

For next time: A few little exercises

- Given a list of numbers, find the pair with greatest difference
- Given a list of numbers find the pair with smallest difference
- 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

lec15exercises.py has solutions for first two. Has slow (and not completely correct) solution for third one. Can you think of a much faster solution using dictionaries?

HW4

http://www.cs.uiowa.edu/~cremer/courses/hw/hw4/hw4.html

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

Next Time

- “A few little exercises”
- tuples and tuple assignment (10.26-28)
- default/optional and keyword arguments to function
- Zip and sorting for HW4