Compsci 101
Dictionaries
Live Lecture

Announcements

- Assign 3 Transform due Today! March 18
- APT-5 due Tues, March 23

- Assign 4 Hangman out today!
  - Sakai reading quiz

- Lab 7 Friday

- Exam 2-DO NOT DISCUSS!

PFTD

- Dictionaries

Dr. Amy J. Ko

- Ph.D., Human-Computer Interaction
  - Carnegie Mellon University
- B.S., CS, Psychology
  - Oregon State University
- Professor, Informatics Program Chair
  - University of Washington, Seattle
- Director, Code & Cognition Lab
  - Information School, School of CS & Engineering, and College of Education

Sandwich Bar

APT: SandwichBar Search

Problem Statement

It's time to get something to eat and I've come across a sandwich bar. Like most people, I prefer certain types of sandwiches. In fact, I keep a list of the types of sandwiches I like.

The sandwich bar has certain ingredients available. I will list the types of sandwiches I like in order of preference and buy the first sandwich the bar can make for me. In order for the bar to make a sandwich for me, it must include all of the ingredients I desire.

Class

```
filename: SandwichBar.py

def whichOrder(available, orders):
    """
    return zero-based index of first
    sandwich in orders, list of strings
    that can be made from ingredients
    in available, list of strings
    """

    # you write code here
```

Given `available`, a list of Strings/ingredients the sandwich bar can use, and a `orders`, a list of Strings that represent the types of sandwiches I like, in order of preference (most preferred first), return the 0-based index of the sandwich I will buy. Each element of `orders` represents one type of sandwich I like as a space-separated list of ingredients in the sandwich. If the bar can make no sandwiches I like, return -1.

Sandwich Bar Example

- available = ["cheese", "cheese", "cheese", "tomato"]
- orders = ["ham ham ham", "water", "pork", "bread", "cheese tomato cheese", "beef"]

Sandwich Bar Example

- available = ["cheese", "cheese", "cheese", "tomato"]
- orders = ["ham ham ham", "water", "pork", "bread", "cheese tomato cheese", "beef"]

- Returns 4
- Can make “cheese tomato cheese”

- Ignore any duplicates!

WOTO-1 SandwichBar
http://bit.ly/101s21-0318-1

Another Trip to the SandwichBar

- Use sets to solve this!

- Idea

```
for dex in range(len(orders)):
    if canmake(orders[dex], available):
        return dex
```

- You would need to write the function canmake
- What type does it return?
- What set operation could you use?

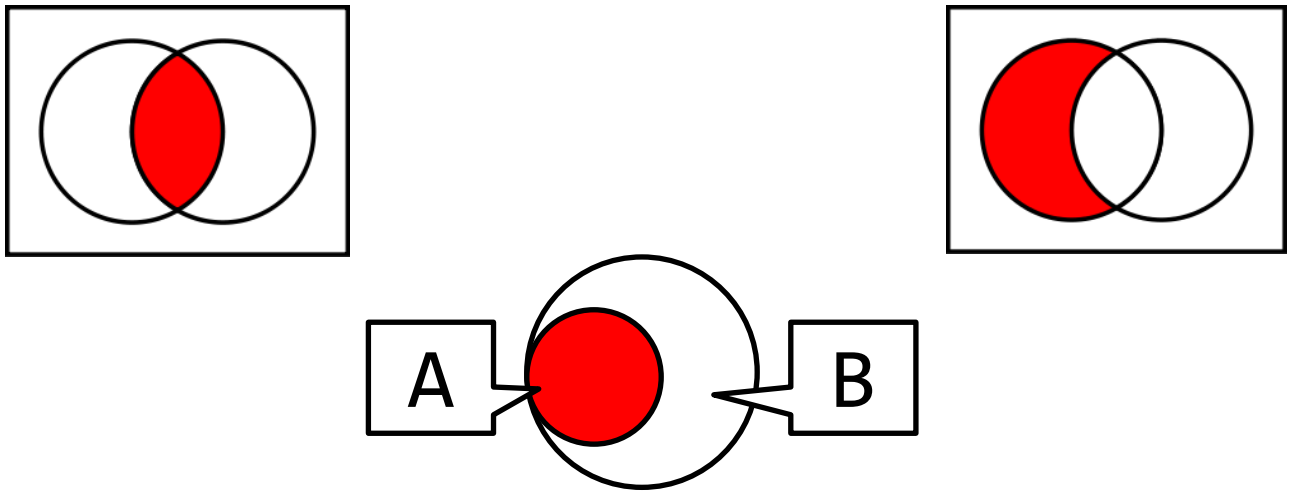
Given two lists A and B

- Determine if all elements in A are also in B
  - Examine each element in A
    - If not in B? False
  - After examining all elements? True

- Think: Could we use sets instead?

Given two sets A and B

- Determine if all elements in A are also in B
  - `if len(A & B) == len(A)`
  - `if len(A - B) == 0`



APT: VenmoTracker

Problem Statement

You've been asked to help manage reports on how often people spend money using Venmo and whether they receive more money than they pay out. The input to your program is a list of transactions from Venmo. Each transaction has the same form: `"from:to:amount"` where *from* is the name of the person paying *amount* dollars to the person whose name is *to*. The value of *amount* will be a valid float **with at most two decimal places.**

Specification

```
filename: VenmoTracker.py

def networth(transactions):
    """
    return list of strings based on transactions,
    which is also a list of strings
    """

    # you write code here
    return []
```

Return a list of strings that has each person who appears in any transaction with the net cash flow through Venmo that person has received. Every cent paid by the person to someone else is a pay-out and every cent received by a person is a pay-in. The difference between pay-out and pay-in is the cash flow received. This will be negative for each person who pays out more than they get via pay-in. See the examples for details.

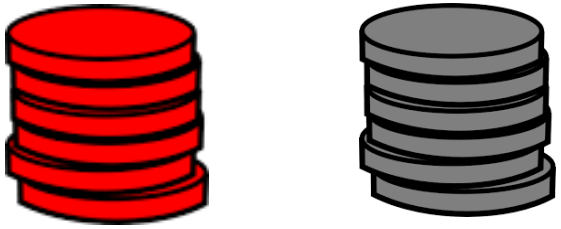
The list returned should be sorted by name. Strings in the list returned are in the format "name:netflow" where the netflow is obtained by using `str(val)` where val is a float representing the net cash flow for that person.

Store money as `int` values, multiplying by 100 and dividing by 100 as needed for processing input and output, respectively.

Motivation for Dictionary

http://bit.ly/venmotracker

- If Harry pays Sally $10.23,
  - "Harry:Sally:10.23" then Harry is out $10.23
- How do we extract sender, receiver, amount?
  - How to process
  - In Python



Tools We've Used Before

- Keep track of every person we see
  - Use a list
- Keep track of net worth: money in, money out
  - Use a parallel list
- Maintain invariant: **`names[k] <-> money[k]`**
  - k^{th} name has k^{th} money

WOTO-2 Venmo Tracker
http://bit.ly/101s21-0318-2

Some APT details

- Given a person's name, if we haven't seen it…
  - Append to **`names`**, append 0 to **`money`**
  - Why must we do both? Invariant, update!
- Find index of person's name in **`names`**
  - Update corresponding entry in **`money`**
- Use $$ * 100 to avoid floating point issues
- Sorted names: **`sorted(…)`**
  - Returns a sorted list of the passed in sequence

Seen parallel lists before

- Solution outlined is reasonable, efficient?
  - How long does it take to find index of name?
  - It depends. Why?

- **`list.index(elt)`** or **`elt in list`** – fast?
  - What does "fast" mean? Relative to what?

- Such a common idiom most languages support fast alternative: dictionary aka map aka hash …

Dictionaries….?

WOTO-3 Dictionaries
http://bit.ly/101s21-0318-3

- In your groups:
  - Come to a consensus

Short Code and Long Time

- See module WordFrequencies.py
  - Find # times each word in a list of words occurs
  - We have tuple/pair: word and word-frequency

```
37  def slowcount(words):
38      pairs = [(w,words.count(w)) for w in set(words)]
39      return sorted(pairs)
```

- Think: How many times is **`words.count(w)`** called?
  - Why is **`set(words)`** used in list comprehension?

WordFrequencies with Dictionary

- If start with a million words, then…
- We look at a million words to count # "cats"
  - Then a million words to count # "dogs"
  - Could update with parallel lists, but still slow!
  - Look at each word once: dictionary!

- Key idea: use word as the "key" to find occurrences, update as needed
  - Syntax similar to **`counter[k] += 1`**

Using fastcount

- Update count if we've seen word before
  - Otherwise it's the first time, occurs once

```
28  def fastcount(words):
29      d = {}
30      for w in words:
31          if w in d:
32              d[w] += 1
33          else:
34              d[w] = 1
35      return sorted(d.items())
```

Assignment 4: Hangman

- We give you most of the functions to implement
  - Partially for testing, partially for guiding you
- But still more open ended than prior assignments
- If the doc does not tell you what to do:
  - Your chance to decide on your own!
    - Okay to get it wrong on the first try
  - Discuss with TAs and friends, brainstorm!
- Remember: sorted(…) – cover next lecture