15-110: Principles of Computing

Homework 09

Due: 23rd October, 2022 at 10:00pm

- You need to complete the Python file for this assignment, and submit it to Gradescope.
- There are 100 points.
- You must solve the tasks **individually**, always abiding by the course and CMU's academic integrity policy.

1. (20 points) **Diagnose**

   Suppose you have a database of diseases and their symptoms stored in a python dictionary in the following way:

   ```
   symptoms = {
     "common cold": ["sore throat", "sneezing", "cough", "runny nose"],
     "flu": ["fever", "headache", "weakness", "body aches", "cough"],
     "asthma": ["wheezing" , "shortness of breath", "tight chest", "cough"],
     "conjunctivitis": ["red eyes", "eye discharge"]
   }
   ```

   Implement the function `diagnose(d, s)` that takes as input a dictionary `d` like the one above (but not exactly that one), a symptom `s`, and returns a list of possible diseases that might be causing that symptom, in alphabetical order.

   For example, `diagnose(symptoms, "cough")` should return the list `["asthma", "common cold", "flu"]`.

   Think: Would your implementation be simpler if the dictionary was organized differently? If you could choose, how would you organize the dictionary?

2. (25 points) **Antisocial Score**

   You have created a survey to find out who is friends with whom, and the result of this survey is given to you as a (multiline) string in the following format:

   ```
   '''
   Homer: Apu, Moe
   Flanders: Homer, Apu, Moe
   Apu:
   Moe: Homer
   '''
   ```

   That means that Homer answered that his friends are Apu and Moe; Flanders answered that his friends are Homer, Apu, and Moe; Apu answered that he has no friends, and Moe answered that Homer is his only friend.

   As you can see, friendship does not go both ways in Springfield. Whenever person A does not list B as their friend, but person B lists A, we can say that A is antisocial towards B. This way, we can

compute a person's *antisocial score*. We will consider person A's antisocial score to be the number of people B such that A does not list B as a friend, but B does list A.

For example, Apu's antisocial score is 2 because Homer and Flanders list Apu as a friend, but Apu lists neither as a friend.

Implement the function `antisocialScore(s)` that takes as input a string as above, and returns a dictionary where the keys are each person's name, and the value is an integer representing that person's antisocial score.

For example, if `friends` is the string above, then `antisocialScore(friends)` should return the dictionary:

```
{
  'Homer': 1,
  'Flanders': 0,
  'Apu': 2,
  'Moe': 1
}
```

3. (25 points) **Word Count**

   Search engines are tools that help people search for documents in a database. The most common search engine is likely Google, which helps people search for webpages (documents) on the internet (database). One way (out of many!) that search engines have to figure out how relevant a document is, is to verify how many times the word looked for occurs in a document.

   For example, if you search for `"python"`, a webpage where the word `"python"` occurs 15 times is probably more relevant than a webpage where the word `"python"` occurs once.

   Given the volume of searches, it is useful to keep a dictionary of words and their number of occurrences for each document. Implement the function `wordCount(doc)` that takes a document (a string) as the input, and returns a dictionary where the keys are the words (strings) and values are the number of occurrences of that word (int).

   Observe that:

   - Punctuation marks do not count.
   - Capitalization is irrelevant (so `"the"` should be treated as the same word as `"The"`).
   - The words used as keys in the dictionary must be in lowercase characters.

   For example, if

   ```
   S = "She was young the way an actual young person is young."
   ```

   then `wordCount(S)` should return the dictionary (not necessarily in this order):

   ```
   {'actual': 1,
    'an': 1,
    'is': 1,
    'person': 1,
    'she': 1,
    'the': 1,
    'was': 1,
    'way': 1,
    'young': 3}
   ```

   HINT 1: Think about pre-processing the string first. Remove characters that are not relevant, and fix capitalization.

   HINT 2: In order to identify characters that are alphanumeric (‘"a"‘ to ‘"z"‘, ‘"A"‘ to ‘"Z"‘, and ‘"0"‘ to ‘"9" ‘) or whitespaces (spaces, linebreaks, tabs, etc), you can use the string functions:

- `s.isalnum()` returns `True` if `s` is composed of only alphanumeric characters, and is not the empty string.
- `s.isspace()` returns `True` if `s` is composed of only whitespace characters, and is not the empty string.

Some of the string constants from `https://docs.python.org/3/library/string.html` may also be useful.

4. (30 points) **Groceries Shopping**

   The latest trend nowadays is online shopping, but although there are many ways to shop for clothes, shoes, and electronic products online, few grocery shops and supermarkets offer this practical service. Knowing how to develop programs, you decided to offer supermarkets a system where their clients can shop online. Your system needs to keep track of items in stock and their prices. Having learned about dictionaries, you think this will be a great way to store all this information.

   You have decided to keep two separate dictionaries, one for stock and one for prices. The keys of both dictionaries are the product names, and the values are numbers. The values are integers for the stock dictionary, whereas the values are floats for the price. Here is an example of both dictionaries:

```
stock = {
    "tomato": 6,
    "cucumber": 0,
    "lettuce": 32,
    "spinach": 15
}

prices = {
    "tomato": 4,
    "cucumber": 2,
    "lettuce": 1.5,
    "spinach": 3
}
```

   The supermarket system must provide a few simple functionalities, which you will implement for this task.

   (a) Sometimes the supermarket will need to adjust the price of one of the products. Implement the function `adjustPrice(pd, prod, adj)` that takes as input a price dictionary `pd`, a product `prod`, and a price adjustment `adj`; and returns a new price dictionary with `prod`'s price updated. All other prices remain the same.

   The price adjustment is given in terms of a percentage. For example if `adj` is 20, that means the price goes *up* 20%; if `adj` is -10, that means the price goes *down* 10%.

   If `prices` is the price dictionary above, then `adjustPrice(prices, "tomato", 10)` should return the dictionary:

```
{
    "tomato": 4.4,
    "cucumber": 2,
    "lettuce": 1.5,
    "spinach": 3
}
```

   (b) Supermarkets need to keep refilling the shelves so it receives new products for restocking with a certain regularity. The warehouse system sends to your system the new items that have arrived and their quantities as a dictionary as well. For example:

```
new_items = {
    "tomato": 20,
    "spinach": 10,
    "zucchini": 5
}
```

Implement the function `restock(sd, new)` that takes as input two dictionaries containing the stock and the newly arrived items, and returns a new stock dictionary with the values updated. If new items have arrived, they must be included in the stock dictionary.

For example, `restock(stock, new_items)` should return:

```
{
    "tomato": 26,
    "cucumber": 0,
    "lettuce": 32,
    "spinach": 25,
    "zucchini": 5
}
```

(c) Eventually, people are going to start shopping online, so you need to provide the functionality to compute someone's total bill, given their shopping cart. A customer's shopping cart is a dictionary of items and the quantities they want to buy:

```
my_cart = {
    "tomato": 5,
    "lettuce": 1,
    "spinach": 2
}
```

You can assume that if an item could be added to the shopping cart in the desired quantity, it is available in stock and has a price.

Implement the function `computeTotal(pd, cart)` that takes as input a price and a shopping cart dictionaries, and returns the total amount the customer has to pay.

For example, `computeTotal(prices, my_cart)`, should return 27.5.