15-110: Principles of Computing

Homework 08

Due: 21st November, 2020 at 23:59

- You must solve the tasks **individually**.
- There are 50 points.

1. (10 points) **Alphabetical Grades**

   Suppose you have a dicionary containing students' grades organized in the following way: keys are the students names (strings), and values are the student grade (floating point number).

   Implement the function `alphabeticalGrades(d)` that takes as input a dicionary as described above, and returns a list of grades of the students when listed in alphabetical order.

   For example, let

   ```
   grades = {"Barbara": 87.8,
             "Hugo": 65.6,
             "Alice": 99.7,
             "Thomas": 77.5 }
   ```

   then `alphabeticalGrades(grades)` should return `[99.7, 87.8, 65.6, 77.5]`, which are the grades of Alice, Barbara, Hugo, and Thomas.

   ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^

2. (15 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) in 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 lower case characters.

   For example, if

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

   then `wordFreq(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. Removes 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.

3. (10 points) **Number of Friends**

Given pairs of friends, we want to find out how many friends each person has. Implement the function `numberOfFriends(pairs)` that takes as input a list of pairs of friends, and returns a dictionary where the key is a person and the value is the number of friends they have. You can assume that friendship is mutual, that means that a pair `("Aisha", "Eunice")` indicates that Aisha is Eunice's friend, and vice-versa.

For example, suppose the list of friend pairs is:

```
friends = [("Alberto", "Marcin"),
           ("Marcin", "Pablo"),
           ("Alberto", "Carla"),
           ("Carla", "Miguel"),
           ("Carla", "Ivete"),
           ("Marcin", "Bart")
          ]
```

then `numberOfFriends(friends)` should return the dictionary (not necessarily in this order):

```
{ "Alberto": 2,
  "Marcin": 3,
  "Pablo": 1,
  "Carla": 3,
  "Miguel": 1,
  "Ivete": 1,
  "Bart": 1 }
```

You can assume that the same pair of friends occurs only once.

4. (15 points) **Diagnose**

Suppose you have a database of diseases and their symptoms stored in a python dictionary in the folllowing 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 in a different way? If you could choose, how would you organize the dictionary?