ECE 364 Prelab Assignment 03

Python Collections

Passing this lab will satisfy course objectives CO2, CO3, CO7

Instructions

- You must meet all *base requirements* in the syllabus to receive any credit.
- Work in your Prelab03 directory, and copy all files from ~ee364/DataFolder/Prelab03 into your working directory:

  ```
  cp -r ~ee364/DataFolder/Prelab03/* ./
  ```

- Remember to add and commit all **requird** files to SVN. **We will grade the version of the file that is in SVN!**
- Do *not* add any file that is *not* required. **You will lose points if your repository contains more files than required!**
- Make sure you file compiles. **You will not receive any credit if your file does not compile.**
- Name and spell the file, and the functions, exactly as instructed. Your scripts will be graded by an automated process. **You will lose some points, per our discretion, for any function that does not match the expected name.**
- Make sure your output from all functions match the given examples. **You will not receive points for a question whose output mismatches the expected result.**
- Unless otherwise specified, you cannot use any external library, but you can use any module in the **Python Standard Library** to solve this lab, i.e. anything under:

  ```
  https://docs.python.org/3.7/library/index.html
  ```

- Make sure you are using Python 3.7 for your Prelab.

Python Collections

Required File `collectionTasks.py`

Description

Create a Python file named `collectionTasks.py`, and do all of your work in that file. This is the only file you need to submit. You can write any number of additional helper functions in this file. Refer to the **Python File Structure** section below for a reminder on the requirements. Note that the use of type annotation is *not* required, but it is recommended for better compile-time static analysis.

Students in some university are collaborating on several projects, where in each projects they are building multiple circuits using four component types: Resistors, Inductors, Capacitors and Transistors. You are given some files in a sub-folder, called `maps`, indicating that these files contain mapping information between two collections. These files are:

- `students.dat` containing student names and their IDs. Note that, throughout this assignment, the format of the student name should match what is used in this file.
- `projects.dat` containing project IDs and the circuits IDs that belong to each project.
- The component files, `resistors.dat`, `inductors.dat`, `capacitors.dat` and `transistors.dat`, where each file contains the component codes (or IDs) of that type, along with their prices. Any component used throughout this assignment will following the given format.

In addition, you are given another sub-folder called `circuits` containing multiple files, where each file holds information about a specific circuit that may have been used in one or more projects. The format of each file name in the folder is `circuit_<circuitID>.dat` and it contains two types of information:

- The students IDs that have collaborated in building that circuit.
- The unique set of component IDs used in that circuit.

While the format of each circuit file is consistent, the number of students participating in building each circuit varies, and so does the number of components used in that circuit. Moreover, some circuits may not have uses all component types, i.e. they may have not used transistors, or inductors, etc. Examine all of the files before you begin, to make sure you are comfortable with their format.

Note: You may hardcode the names of the two sub-folders along with the names of the files in the `maps` sub-folder. However, you cannot hardcode any of their content, nor can you hardcode the circuit IDs.

Requirements

1. **[10 pts]** Write a function called `getComponentCountByProject(projectID, componentSymbol)` that takes in a project ID, and a one letter component symbol: `"R"`, `"I"`, `"C"` or `"T"` for Resistors, Inductors, Capacitors or Transistors, respectively. The function should return the distinct number of components, of the requested type, used in all circuits within that project. If the project ID provided does not exists, raise a `ValueError` with a descriptive message.
2. **[10 pts]** Write a function called `getComponentCountByStudent(studentName, componentSymbol)` that takes in the full student name, a component symbol, and returns the distinct number of components, of the requested type, used in all circuits where that student has participated. If the student has not participated in any projects, return `0`, but if the student name passed does not exist, raise a `ValueError` with a descriptive message.

3. [**10 pts**] Write a function called `getParticipationByStudent(studentName)` that takes in the full student name and returns a `set` of the project IDs that the student has participated in. If the student has not participated in any projects, return an empty `set`. If the student name passed does not exist, raise a `ValueError` with a descriptive message.

4. [**5 pts**] Write a function called `getParticipationByProject(projectID)` that takes in a project ID, and returns a `set` of all the full names of the students who have participated in this project. If the project ID provided does not exists, raise a `ValueError` with a descriptive message.

5. [**10 pts**] Write a function called `getCostOfProjects()` that returns a `{string: float}` dictionary, where the key is the project ID, and the value is the total cost of completing that project, rounded to two decimals.

6. [**10 pts**] Write a function called `getProjectByComponent(componentIDs)` that takes in a `set` of component IDs, and returns `set` of the project IDs in which any of components passed has been used.

7. [**5 pts**] Write a function called `getCommonByProject(projectID1, projectID2)` that returns a *sorted list* of all the distinct components that have been used in both of the projects passed. If there were no common components used between these projects, return an empty list, and if either of the project IDs passed does not exist, raise a `ValueError` with a descriptive message.

8. [**5 pts**] Write a function called `getComponentReport(componentIDs)` that takes in a `set` of component IDs, and returns a `{string: int}` dictionary, where the key is the component ID, and the value is the total number of times that component has been used in all projects.

9. [**10 pts**] Write a function called `getCircuitByStudent(studentNames)` that takes in a `set` of student names, and returns a `set` of all the circuit IDs that any of students passed has worked on.

10. [**5 pts**] Write a function called `getCircuitByComponent(componentIDs)` that takes in a `set` of component IDs, and returns a `set` of all the circuit IDs that any of components passed are used in.

Notes

Before you start working on this assignment, you need to be comfortable with how you can approach all of the problems. This assignment makes heavy use of collections, and the choices you make for using these collections can simplify the solution or complicate it significantly. Most of these functions can be completed in less than 10 lines of code, predicated on the fact that you have performed the correct file loading.

Note that, in this assignment, we have the following elements: project IDs, circuit IDs, student names, student IDs, component names (i.e. resistors, capacitors, etc.), component IDs, component costs. If you create all the possible mappings for these elements (i.e. project IDs to student Names, project IDs to student IDs, etc.) You can solve each question by using one of these mappings. However, this requires $7! = 5,040$ mappings, most of which you do not need. It is absolutely crucial to reuse the mappings across questions.

Here are some more ideas to consider as you are working:

- You *must* study and understand each collection before you use them, or you will write a lot of redundant code that performs operations already available for you. For example:
  - Lists are good for data ordering, maintaining duplication in the data, etc.
  - Sets are better at identifying distinctiveness, commonalities and differences among collections.
  - Tuples are used for immutability, grouping, pattern matching, etc.

  Additionally, you should restrict conversion between collections to a minimum (i.e. do not convert sets to lists, or vice versa, unless you absolute need to.)

- For each question, identify all the mappings that you need to do to get the answer *before* you start coding. Note that there are multiple ways to solve any question.
- When you load a file, validate its content before you use it. File formats are different and you may skip a line or more by mistake. Hence, always check that the first and last lines are loaded properly.
- Remember that for each file you can create two, and sometimes more, mappings. For example:
  - `projects.dat` can be used to create two one-to-many mappings, such that you can get circuit IDs for a given single project ID, as well as the projects IDs for a given single circuit ID. This must be done with two separate collections, which you can create using one or two procedures.
  - `students.dat` can be used to create two one-to-one mappings: name-to-ID, and ID-to-name, which, again, must be done with two separate collections that you can create using one or two procedures.
- Some of the mappings you need for a certain question might be available (i.e. you wrote the code for) in a previous question (which, of course, depends on the order you solve the questions.) For example:
  - Question 1 requires the following mappings: project ID to circuit IDs, circuit ID to component IDs, and component IDs to type/cost.
  - Question 5 requires exactly the same mappings, which means that, if written properly, you can reuse most of the code.
- Verify the answers for each question before you move to the next. The dependencies between questions can cascade the errors if an erroneous procedure is used in multiple questions. You can create procedures and sample data that help verify more than one question. For example:
  - Select two concrete project IDs.
  - For each project ID, copy the relevant circuits into a sub-folder.
  - Combine the circuits, and obtain two lists: student IDs and component IDs.

  These data sets can help verify the answer for questions: 1, 4, 5 and 7.

Final Check!

You are given a file that checks for the exact spelling of the required functions. Make sure you run this file before your final commit. Remember that **you will lose some points for any function that does not match the expected name.**

Python File Structure

The following is the expected file structure that you need to conform to:

```
#########################################################
#   Author:     <Your Full Name>
#   email:      <Your Email>
#   ID:         <Your course ID, e.g. ee364j20>
#   Date:       <Start Date>
#########################################################

import os     # List of module import statements
import sys    # Each one on a line


#########################################################
# No Module-Level Variables or Statements!
# ONLY FUNCTIONS BEYOND THIS POINT!
#########################################################

def functionName1(a: float, b: float) -> float:
    return 0.


def functionName2(c: str, d: str) -> int:
    return 1


# This block is optional and can be used for testing.
# We will NOT look into its content.
#########################################################
if __name__ == "__main__":
    # Write anything here to test your code.
    z = functionName1(1, 2)
    print(z)
```