Project(Assignment) 1

You should complete at least part1, 2, 3, 4, 6 to get full credit for this project, 5, and 7 are bonus questions. You should create new github repository and submit using github.

Part1: file reading and data types

Suppose that the government, for security reasons, wanted to store a database containing the number of times every person called every other person. There are two simple Python data structures that can store this information so that it can be easily processed.

A dictionary associating a 2-tuple of str (the caller and callee) with an int: the first value in the tuple is the caller (the person placing the call); the second value is the callee (the person receiving the call); the int is the number of times the caller called the callee). A dictionary associating a str (the caller) with another dictionary of all the people they called: this inner dictionary associates a str (the callee) with an int (the number of times the caller called the callee. For example in a simple/small database of a calling b 2 times, a calling c 1 time, b calling a 3 times, b calling c 1 time, c calling a 1 time, and c calling d 2 times, the representation of this data in the two data structures are

```
{('a','b'): 2, ('a','c'): 1, ('b','a'): 3, ('b','c'): 1, ('c','a'): 1, ('c','d'): 2}
{'a': {'b': 2, 'c': 1}, 'b': {'a': 3, 'c': 1}, 'c': {'a': 1, 'd': 2}}
```

Of course, dictionaries can print in any order, because they are not ordered.

You are required to write the following function:

```
read_calls
```

The interpretation of the function should be as following:

```
The input file will consist of some number of lines; each line can contain
information about many calls: the name of the caller followed by each
callee he/she called; the names (which can be any length) will be separated by colons.

For one example, the calls.txt file contains

b:a:c:a
a:b
c:a:d:d
b:a
a:b:c
Here is what each of the five lines mean:

b called a, then c, then a again
a called b
c called a, then d, then d again
```

```
b called a (see line 1: OK to have multiple lines with the same caller)
a called b, then c (see line 2: ditto)
Each line should initialize/update information in the dictionary for the caller
and all his/her callees. The five-line file above returns a dictionary with the
following contents:

{('a','b'): 2, ('a','c'): 1, ('b','a'): 3, ('b','c'): 1, ('c','a'): 1, ('c','d'): 2}
```

You are also required to complete a function called call1to2 which converted the the first way to the second one, the following is the detailed explaination.

The call1to2 function takes a dict argument in the form of data structure 1.

(the dictionary's keys are 2-tuples (caller,callee) whose associated value is an int: the number of times the caller called the callee) and returns an equivalent dictionary (dict or defaultdict, which print differently) in the form of data structure 2 (the dictionary's keys are str (callers); each caller is associated with a dictionary whose keys are str (callee) and whose associated value is an int: the number of times the caller called the callee). So calling

```
call1to2( {('a','b'): 2, ('a','c'): 1, ('b','a'): 3, ('b','c'):
1, ('c','a'): 1, ('c','d'): 2} )
```

returns a dictionary whose contents are

```
{'a': {'b': 2, 'c': 1}, 'b': {'a': 3, c: 1}, 'c': {'a': 1, 'd': 2}}
```

Part2: comprehension

Complete the following comprehensions, if you not doing comprehensions then you will received 60% of the total credit even if you pass the test cases.

1. Given a dictionary where the values are unique, create a new dictionary where the keys become values and the values become keys.
2. Given a dictionary, create a new dictionary where each value from the original dictionary becomes a key in the new dictionary and its value is a list of keys from the original dictionary that had the same value.
3. Given two dictionaries, merge them into a new dictionary. Where keys match, sum their values, assuming all values are numeric.
4. Create a list of integers from a nested list structure that appear more than once anywhere in the entire structure. For example, from [[1, 2], [3, 2], [1, 5, 3], [6, 5]], you'd filter for numbers that appear in multiple sublists.

Part3: Emulate Python namespace

You are tasked with creating a class NamespaceManager that can dynamically manage variables within its own namespace and provide functionality to access, modify, and delete these variables. The class should support the following methods:

1. set_variable(name, value): Sets a variable in the namespace with the given name and value.
2. get_variable(name): Retrieves the value of a variable with the given name. Raises a KeyError if the variable does not exist.
3. delete_variable(name): Deletes the variable with the given name. Raises a KeyError if the variable does not exist.
4. list_variables(): Returns a list of all variable names currently stored in the namespace.
5. execute_function that takes a string of Python code, executes it within the context of the namespace managed by the NamespaceManager class, and returns the result. This method should allow the executed code to access and modify the variables stored in the namespace. (hint: use exec(), this one is hard)

Part4: Coding problem for dict

You are given a set of buyers and a set of stocks. Each buyer has a list of stocks they prefer to purchase, ranked from most preferred to least preferred. Similarly, each stock has a list of buyers they prefer to sell to, ranked from most preferred to least preferred. Your task is to find a stable matching between buyers and stocks such that no buyer and stock would prefer each other over their current match.

You are given: A dictionary buyers_preferences where the keys are buyer names and the values are lists of stock names in the order of preference. A dictionary stocks_preferences where the keys are stock names and the values are lists of buyer names in the order of preference.

```
buyers_preferences = {
    'Buyer1': ['StockA', 'StockB', 'StockC'],
    'Buyer2': ['StockB', 'StockA', 'StockC'],
    'Buyer3': ['StockA', 'StockB', 'StockC']
}

stocks_preferences = {
    'StockA': ['Buyer1', 'Buyer2', 'Buyer3'],
    'StockB': ['Buyer2', 'Buyer1', 'Buyer3'],
    'StockC': ['Buyer1', 'Buyer2', 'Buyer3']
}
```

You should return: A dictionary representing the stable matching where the keys are buyer names and the values are the names of the stocks they are matched with. If you have the above example, you should have the following:

```
{
    'Buyer1': 'StockA',
    'Buyer2': 'StockB',
    'Buyer3': 'StockC'
}
```

We will also consider the following constrain: Each buyer can be matched with only one stock. Each stock can be matched with only one buyer. The preferences of both buyers and stocks are complete and no preferences are missing.

Part 5: AoA(Extra credit) Write it in comment in part4.py

This is a bonus question, analysis the above question in coding part4, what is the time and space complexity, why do you think it is optimal? In order to get this bonus credit, you need to: have optimal to good (do not have a growing time of exponential) solution of part4, pass all test cases in part4, have correct AoA for part4. Show your steps to get to your answer, do not just give me a single Big-O notation with no explanation.

Part 6: Iterators

This question will be released after we go over the topic.

Part 7: Github(Extra credit)

If you would like to create a local repository and link to a remote git repository, which steps should you take in Powershell and terminal?
Submit your answer in python comments.