Assignment #2: Word Soup

Due Date:

1. **Abstract**

This project will require the use of concepts you have learned in the first few weeks of the course, such as I/O, loops, conditionals, comparisons, lists, and dictionaries. All of these concepts and more will be used in conjunction with one another to create a dictionary which maps each letter of the alphabet to a list of words starting with that letter. Your program must that each entry in the dictionary is distinct (as in, no duplicates), that everything is in alphabetical order, and that anything added to the dictionary is placed correctly.

2. **Objective**

You are tasked with implementing a dictionary that maps each letter of the alphabet to a list of words starting with that letter. Everything should be in alphabetical order, and the initial list should be created with a function that creates the list via asking for an input continuously, only ending when a terminating phrase is input (‘<_done_>’). For example, when fed an input list of [‘apple’, ‘banana’, ‘radish, ‘corn’, ‘bonbon’, ‘apple’], the dictionary, when printed, should look as follows:

{ 'a': ['apple'], 'b': ['banana', 'bonbon'], 'c': ['corn'], 'd': [], 'e': [], 'f': [], 'g': [], 'h': [], 'i': [], 'j': [], 'k': [], 'l': [], 'm': [], 'n': [], 'o': [], 'p': [], 'q': [], 'r': ['radish'], 's': [], 't': [], 'u': [], 'v': [], 'w': [], 'x': [], 'y': [], 'z': [] }

Note that ‘apple’ appears twice in the list, but only once in the dictionary. Also, note that this looks absolutely abysmal, with all the empty lists mapped to the unused keys. However, this is necessary in regard to future-proofing! If we don’t initialize the other letters with empty lists, and then try to add, say, ‘dinner’, the key ‘d’ would end up coming after ‘r’, thus breaking our alphabetical order requirement. In Python, the only way to change the order of dictionary keys is to extract them all into a list, sort them, then create a new dictionary. This is time consuming, wasteful, and quite frankly, annoying, so what we will do instead is simply create a printing function that ignores keys with empty lists (more on this in the next part).

So, instead of flat out printing the abomination seen in the previous example, our custom printing function will output this:

```
a: ['apple']
b: ['banana', 'bonbon']
c: ['corn']
r: ['radish']
```

3. **Function & Method Restrictions**

Use of the **.sort()** method and **sorted()** function are strictly prohibited. In the case of inserting new words, it would be extremely tempting to use sort after each insertion; however, this would be somewhat wasteful in terms of runtime, hence the restriction.

4. **Required Functions**

Create_List(fp)

Description: This function should take an opened text file as the input, then read every word in the file and put it into a list, with all words appearing in the same order they did in the file. Then, this function should return the list.

Parameter Restrictions: The words in the file are guaranteed to be strings composed purely of lowercase letters.

Output: This function should not print anything to the screen.

Return Value: This function should return the completed list.

Create_Dict(myList)

Description: This function will create a dictionary with 26 keys, one for each letter of the alphabet. This dictionary will map each alphabet key to a list containing each DISTINCT word in the input list (myList) that starts with that letter. Each of these lists should be sorted by alphabetical order, and any letter key that has no corresponding words in myList should simply be mapped to an empty list, [].

For example, a list of ['apple', 'banana', 'agility', 'zoo', 'apple', 'apple’] will result in a dictionary that looks like this (most unused letters omitted for the sake of space):

{'a': ['agility', 'apple'], 'b': ['banana'], 'c' : [] ... 'z': ['zoo']}

That said, this function should NOT PRINT ANYTHING!

Parameter Restrictions: The input list will be one created from the Create_List() function, and thus it is guaranteed to only contain strings made of lowercase characters. Note that it is NOT guaranteed to be sorted and may contain duplicates.

Output: This function should not print anything to the screen.

Return Value: Return the created dictionary, even if it’s completely empty.

Fancy_Print(myDict)

Description: This function will print every key and its corresponding list, with a newline between each key; if the list is empty however, neither is printed. Each line will be printed exactly as such:

key: [words in list]

See the problem statement for a more in-depth example. Note that there is no space between the key and the colon, but there is a space between the colon and the list.

Parameter Restrictions: The input dictionary will be the one created by the Create_Dict() function.

Output: Prints each key and list, assuming the list is not empty.

Return Value: This function should not return anything.

5. Testing

In testing your implementations of the above required functions, you can test the functions individually using your own test cases.

However, it is highly recommended that you make use of the provided soupTaster.py program to obtain immediate feedback on your implemented functions. Please note that the soupTaster.py program makes use of a suite of test cases, provided within a TestCases folder. This TestCases folder contains a total of **eight** .txt files: one named random, the other testcase1 through testcase7. As such, the soupTaster.py program will conduct a total of **7** tests on your implemented functions. Note that all three of the required functions **MUST** be fully implemented for proper testing. In other words, if one of the implemented functions isn't fully defined, not included in your program, or badly implemented (meaning that the way how the function was defined is incorrect, or doesn't accomplish what was asked for as specified), there is a good likelihood that many of the test cases will either fail or crash. To further elaborate, all test cases that the tester program conducts will make use of all three functions.

The soupTaster.py program will first attempt to import your program (****NOTE: IT IS IMPORTANT THAT YOUR PROGRAM IS SAVED AS wordsoup.py**). If it encounters any sort, or form, of syntax errors, the tester program will let you know if that's the case and print out the exact error encountered. This is useful for debugging purposes.

Afterwards, the soupTaster.py program will check to see if the TestCases folder is accessible to it. If it is not, it will let you know, and the program will exit. As such, it is ***highly suggested*** to create a separate, dedicated folder that includes the following **three** items: the TestCases folder, the soupTaster.py program, and your wordsoup.py program. If testing is done on Repl instead of IDLE, make sure to include those three items within Repl's working directory, not including the main.py file that Repl relies on. **WITHIN** the main.py file, included the following line of code:

```
import soupTaster
```

Once that is done, just click on "Run" within Repl, and the tester program should execute.

Please note that you are more than welcome to examine the soupTaster.py program to see how the testing is done, and receiving a good opportunity at examining some high-level Python code as well.

6. Requirements

Your program must be named “wordsoup.py”, and the .sort() method as well as the sorted() function must not be used anywhere in the program.

7. Help & Advice

If you feel overwhelmed by this assignment, don’t be. A good starting point is to first create skeleton definitions of the required functions that returns dummy values. From there, work on the function that you feel is the easiest to implement. Once defined, run the functionsTester.py program and see if you were able to successfully implement it. From this point, rinse and repeat.

If you need any kind of clarification to anything on this assignment, such as if you’re confused about the requirements for one of the functions, or if you encounter a bug error that you are unable to figure out on your own (this includes any potential errors arising from the soupTaster.py tester program), please send an email to **chsprogvol@gmail.com**. You may expect a response within 24 to 48 hours.