Homework 07 - Python Containers

Re-submit Assignment

Due Mar 11 by 10pm **Points** 100 **Submitting** a text entry box or a file upload

Assignment Description

> The goal of this assignment is to give you a chance to practice using `list`, `tuple`, `dict, defaultdict`, and `set`. There are three parts to this assignment. None of the individual functions require much code, but writing concise, elegant solutions may take a little thought.

Part 1: Anagrams

Two strings are anagrams if the characters in the first word can be rearranged to make the second word, e.g. 'cinema' and 'iceman' are anagrams. "dormitory" and "dirtyroom" are also anagrams.

You may assume for this assignment that both strings must have the same length to be anagrams.

Part 1.1: Anagrams using only strings and lists

`anagrams_lst(str1: str, str2: str) -> bool`

Implement (**including automated unit tests**) a function `anagrams_lst(str1, str2)` that returns `True` if str1 and str2 are anagrams, `False` if not. Your solution may **only** use `list`.

Hint: Your `anagrams_lst(str1, str2)` function needs only one line.

You'll find more hints in Canvas but see if you can come up with a solution before checking the hint.

Note: A student was recently asked to solve this problem at a job interview with Microsoft.

Part 1.2: Anagrams using defaultdict

`anagrams_dd(str1: str, str2: str) -> bool`

Part 1.1 was fun but didn't require much programming so we'll look at another solution to determine if two strings are anagrams using `defaultdict`. Write a function `anagrams_dd(str1, str2)` that also returns True or False, but uses the following strategy:

1. Create a **defaultdict of integers**, `dd`

2. Go through `str1`, adding each character to `dd` as the key and incrementing the value by 1. E.g. say `str1 == "dormitory"`. After this step, `dd == {'d': 1, 'o': 2, 'r': 2, 'm': 1, 'i': 1, 't': 1, 'y': 1}`
3. Go through each character, `c`, in `str2`. If the character is a key in `dd`, then decrement the value of `dd[c]` by 1. If `c` is not a key in `dd`, then you know that `str1` and `str2` are not anagrams because `str2` has a character `c` that is not in `str1`.
4. If you successfully processed all of the characters in `str2`, then iterate through each of the values in dd to insure that every value == 0. **This must be true if the two strings are anagrams.** In our example above, `dd == {'d': 0, 'o': 0, 'r': 0, 'm': 0, 'i': 0, 't': 0, 'y': 0}` if `str2 == "dirtyroom"`. If any value is not 0, then return `False`, else return `True`.

Hint: You might find Python's `any(iterable)` statement helpful depending upon your solution. `any(iterable)` evaluates to True if any of the elements in the iterable are True. E.g. `any(0, 0) == False`, `any(0, 1, 2, 3) == True`.

Part 1.3: Anagrams using Counters

`anagrams_cntr(str1: str, str2: str) -> bool`

Determine if two strings are anagrams using only `collections.Counters`.

Part 2: Covers Alphabet

`covers_alphabet(sentence: str) -> bool`

Write a function `covers_alphabet(sentence)` that returns `True` if sentence includes at least one instance of every character in the alphabet or False **using only Python sets**. E.g.

- `covers_alphabet("AbCdefghiJklomnopqrStuvwxyz")` is True
- `covers_alphabet("We promptly judged antique ivory buckles for the next prize")` is True
- `covers_alphabet("xyz")` is False

Hints:

- Be sure to convert the input string to lower case before comparing.
- Here's a list of all characters that you can copy/paste rather than typing it yourself: `abcdefghijklmnopqrstuvwxyz`
- Here's a few strings that **do cover** all the characters in the alphabet:
  - `"abcdefghijklmnopqrstuvwxyz"`
  - `"aabbcdefghijklmnopqrstuvwxyzzabc"`
  - `"The quick brown fox jumps over the lazy dog"`
  - `"We promptly judged antique ivory buckles for the next prize"`

- The string tested against the alphabet must include at least all of the characters in the alphabet, **but may also contain others**, E.g. the sentence, “The quick, brown, fox; jumps over the lazy dog!” is not grammatically correct but it does cover the alphabet.

Part 3: Web Analyzer

`web_analyzer(weblogs: List[Tuple[str, str]]) -> List[Tuple[str, List[str]]]`

Your boss has been collecting logs with the names of each employee and the websites visited by that employee.

Write a function `web_analyzer(weblogs)`, along with automated tests, to create a summary of the weblogs with each **distinct** site and a sorted list of names of **distinct** people who visited that site. The input to `web_analyzer` is a `list` with the following form

```
[('Nanda', 'google.com'), ('Maha', 'google.com'),
 ('Fei', 'python.org'), ('Maha', 'google.com')]
```

where each item in the list is a tuple with the employee name and the website he/she visited. **Note that each employee may visit multiple websites and each website may be visited more than once.** Your web_analyzer() function should summarize the data and return a list with the following form

```
[('website_1', ['employee_1', 'employee_2']), ('website_2', ['employee_2', 'employee_3'])]
```

where each distinct website appears once **in alphabetical order** along with a sorted list of **distinct** employees who visited that website. **The items in the result should be sorted in ascending order by website and the employees names should be sorted in ascending order.**

For example,

```

```

```
weblogs: List[Tuple[str, str]] = [
     ('Nanda', 'google.com'), ('Maha', 'google.com'),
     ('Fei', 'python.org'), ('Maha', 'google.com'),
     ('Fei', 'python.org'), ('Nanda', 'python.org'),
     ('Fei', 'dzone.com'), ('Nanda', 'google.com'),
     ('Maha', 'google.com'), ]

summary: List[Tuple[str, List[str]]] = [
     ('dzone.com', ['Fei']),
     ('google.com', ['Maha', 'Nanda']),
     ('python.org', ['Fei', 'Nanda']), ]

self.assertEqual(web_analyzer(weblogs), summary)
```

Note: The websites and employee names must be sorted in ascending order.

Hint: My implementation of the `web_analyzer()` function has only 4 lines of code and uses `defaultdict(set)`, and list comprehensions.

Deliverables

File Structure

In this assignment, you will need to **separate your code into two files** - one for code logic and one for unit test. It's always a good practice to separate the code and the test.

You are going to have **two .py files** for submission:

1. `HW07_FirstName_LastName.py -` `the logic for your functions`
2. `HW07_Test_FirstName_LastName.py -` `automated test cases for your functions`

Please let me know if you have any questions or suggestions.