Dictionary Skills

Dictionary Skills is meant to help you practice and assess your understanding of dictionaries.

Follow the directions below to review what you should have learned, practice implementing dictionaries, and submit an assessment of your understanding.

Take Aways

- know how to look up a value in a dictionary (ex: `melons['watermelon']`)
- know how to check **if** a key is in a dictionary (ex: `if 'musk' in melons`)
- know how to set or update a dictionary value (`melons['watermelon'] = 'pink'`)
- know what distinguishes dictionaries from other data structures (fast lookup! mutable, unordered)

Practice

The goal here is for you to see where you're at, review what you remember, and practice the new skills you've gained in the last week.

Directions

The file ***practice.py*** contains many functions that are not complete, but have names and docstrings that explain how each function should work.

Fill in the functions in ***practice.py*** so that each function does what its documentation string says it should do.

Begin by typing the following into the command line.

```
$ python practice.py

4 items had failures:
  3 of   3 in __main__.find_unique_common_items
  4 of   4 in __main__.get_sum_zero_pairs
  2 of   2 in __main__.top_chars
  3 of   4 in __main__.without_duplicates
 ...
***Test Failed*** 12 failures.
```

You'll see a lot of output, followed by a failure report similar to the one above.

As you finish the functions, the output from running this command will decrease. Note that the output is alphabetical, not the order that the functions appear. At the point where you've completed this skills assessment, the only output from this program should be:

```
$ python practice.py

ALL TESTS PASSED. GOOD WORK!
```

Docstrings and Doctests

Inside the docstrings are "doctests," which are runnable code samples from the standard Python interpreter shell that serve as tests for whether the function is working as expected.

A documentation string is the part of the function that explains how the function should work. Documentation strings are found on the first line of the body of the function, right after the function name and parameters are stated.

As their name suggests, doctests are tests for a function found in that function's documentation strings. Doctests are formatted just like you'd see in the Python interpreter. For example, a doctest that aims to test a function called ***all_odd*** would look like this:

```
>>> all_odd([1, 2, 7, -5])
[1, 7, -5]
```

In our case, these little code examples within documentation strings are actually run, or *evaluated*, whenever you run ***practice.py***. If the function does not behave as its doctests state that it should, those doctests are said to have "failed." Thus, the output that you see when you run ***practice.py*** provides information about each failure, if there are any.

Doctests are one of many ways that a programmer can test their Python code. Doctests are especially nice because they are easy to generate; they don't require the creation of additional files or functions outside of the very function you want to test.

> **Note: Use Dictionaries**
>
> The point of this assessment is to test your skills with dictionaries. Therefore, we expect you to **use dictionaries** to solve these problems — don't use fancy libraries.
>
> If you're worried about using a built-in Python function that isn't "allowed" for this assignment, don't worry. Each docstring has explicit instructions (if any) about what you're not allowed to use. If the docstring doesn't say you can't use something, feel free to use it.

Assessment

Using the same directions from the practice section of this assignment, complete the functions in the file ***assessment.py***.

Be honest with yourself and your adviser about how this assessment goes for you. If you don't finish it, take note of which problems you weren't able to finish, and come back to them later. If you have *ideas* about how to

solve something, but not a fully formed solution, leave your thoughts as comments in the function body.

We're assessing this on correctness and style — we want to see where you are on all of these things. So, choose good variable names, and show off good Python style. In addition, for each question, consider any "edge cases" where you might want to handle unusual input or errors and, as you are able to, think about how to handle those. If it makes sense to use a more intermediate concept, like a list comprehension, do so.

You must turn in all of your work **through FRODO** by **9PM on Sunday**.