Mod 1 Homework - Python Fundamentals

This assignment has three parts. Each part will be written in a different file. With the exception of `fizz_buzz.py`, we do not provide these files - you will need to create them yourself:

- Part 1: `fizz_buzz.py`
- Part 2: `count_letters.py`
- Part 3: `is_anagram.py`

As you work on this and other problems in this class, remember:

- Unless otherwise stated, you can submit to gradescope as often as you want for automatic feedback.
- `return` and `print` are different things. If a problem asks you to return something, you must use the keyword `return`. If it asks you to print something, you must use `print`.
- Structure and readability matter. The goal of this class is to learn to write readable, maintainable code. In particular:
  - Every public module, function, class, and method needs a docstring (more info).
  - Readability is important. You can get most of the way there with good names and whitespace. You should also use comments to clarify what tricky parts of your code do.
  - Be consistent with your naming conventions. We try to write code according to the Python style guide.

  As an example, see the two implementations of `sum_k` below, one which uses docstrings, good variable names, and whitespace, and one which does not:

```python
def sum_k_readable(k):
    """Returns the sum of the first `k` integers"""
    # Initialize variables
    _sum = 0

    # Calculate sum
    for i in range(k):
        _sum += i+1

    return _sum

def sum_k_not_readable(k):
    i= 0
    for    j     in     range(    i    ):
        i+=j+1
    return i
```

Problem 1 - `fizz_buzz.py`

Write a function `fizz_buzz(start, finish)` that **prints** numbers from `start` to `finish`, inclusive (both `start` and `finish` should be printed). Replace multiples of 3 with `"fizz"`, multiples of 5 with `"buzz"`, and multiples of both with `"fizzbuzz"`.

Once you have a working algorithm, edit it to also replace numbers that contain 3 or 5 (e.g. 13, 52) with `"fizz"`, `"buzz"`, or `"fizzbuzz"`, as appropriate.

In your final solution, all conditions should work together:

- If a number is a multiple of or contains a 3, print out `fizz`
- If a number is a multiple of or contains a 5, print `buzz`
- If both of the above apply, print `fizzbuzz`.

```
>>> fizz_buzz(1, 15)
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
fizz
14
fizzbuzz
```

There are many solutions to this problem. Once you have one working, this is often a great learning tool for reducing a complicated mess of spaghetti logic to something more elegant. Think of how you can solve the problem with fewer logic branches.

Problem 2 - `count_letters.py`

Write a Python function `count_letters(data)` for counting how often individual letters appear in `data`.

- **Input**: `data`, a string
- **Output**: a dictionary of `letter:count` pairs.
  - Only include letters present in `data`.
  - Ignore characters and symbols that are not standard ascii characters (only use characters that appear in `string.ascii_lowercase`)
  - Ignore case; e.g. consider `'Treat'` as having 2 `'t'`s instead of 1 `T` and 1 `t`.

```
>>> count_letters(data_frost) # see below for initializing data_frost
{'f': 12, 'i': 23, 'r': 14, ...}
```

Note that the order of keys in a dictionary can change run-to-run (e.g. you might not see `f` as the first letter printed when you run this), but the count of letters should not (e.g. `f` should always have a value of `12` for the file `frost.txt`).

- **Reading files** - It is often helpful for us to save complicated input patterns for testing functions in a text file. In python, the command to open a file is `open`, but it creates a file object, not a string. The example below shows how to store information from a file (`frost.txt` in this case) into a string for easy use while testing:

```
>>> # First, read in contents of "frost.txt"
>>> with open("frost.txt", "r") as file:
...     data_frost = file.read()
...
>>> file.close() # ALWAYS close files after opening them
>>> print(data_frost)
Fire and Ice

Some say the world will end in fire,
Some say in ice.
From what I've tasted of desire
I hold with those who favor fire.
But if it had to perish twice,
I think I know enough of hate
To say that for destruction ice
Is also great
And would suffice.

-Robert Frost
>>>
```

Hints

- `frost.txt` needs to be in your **p**resent **w**orking **d**irectory (`pwd`) for this to work. For instance, if your directory sturcure looks like this:

```
cse2050/
|-homework/
|   |homework1/
|   |   |-frost.txt
|   |   |-count_letters.py
|   |homework2/
|   ...
|-labs/
|    |-lab1/
|    |    |-lab1.py
|    |-lab2/
|    ...
```

  you present working directory must be `cse2050/homework/homework1` to be able to access `frost.txt` from within the file `count_letters.py`. (Technically, there are ways to access files in other directories, but we will ignore those for now). To see which directory is the `pwd`, open up a terminal (in VS Code - click `Terminal`, then `New Terminal`), type "pwd" (without the quotes), and press enter:

```
[jas14034@jas14034-vm cse2050]$ pwd
/home/jas14034/cse2050
```

  Note that our `pwd` is "cse2050" in this example. Running `count_letters.py` will result in an error like

this:

```
[jas14034@jas14034-vm cse2050]$ /bin/python3 /home/jas14034/cse2050/homework/homewor...
...k1/count_letters.py
Traceback (most recent call last):
    File "/home/jas14034/cse2050/homework/homework1/count_letters.py", line 1, in <m...
    ...odule>
        f = open('./frost.txt')
FileNotFoundError: [Errno 2] No such file or directory: './frost.txt'
```

An easy fix is to move into the directory containing the python script and text file you are interested in, then running your script with `python3` from that directory: (use `dir` instead of `cd` and `python` instead of `python3` on windows):

```
[jas14034@jas14034-vm cse2050]$ cd homework/homework1/
[jas14034@jas14034-vm homework1]$ pwd
/home/jas14034/cse2050/homework/homework1
[jas14034@jas14034-vm homework1]$ python3 ./count_letters.py
Fire and Ice

Some say the world will end in fire,
Some say in ice.
From what I've tasted of desire
I hold with those who favor fire.
But if it had to perish twice,
I think I know enough of hate
To say that for destruction ice
Is also great
And would suffice.

-Robert Frost
```

You can also open the correct directory in your IDE (go to File > Open Folder > and navigate to the "homework1" directory in VS Code).

Problem 3 - `is_anagram.py`

Write a function `is_anagram(word1, word2)` that compares two strings and returns a boolean denoting whether they are anagrams.

```
>>> is_anagram('rat', 'tar')
True
>>> is_anagram('rat', 'hat')
False
>>> is_anagram('meat', 'team')
True
```

You should import `count_letters()` from Problem 2 above to solve this problem. Note that two words are anagrams if and only if they contain the same number of each letter.

We will manually deduct credit for problem 3 if you do not import and use `count_letters()` in

your solution.

Imports

We do not allow imports on any homework or lab assignments, except for those explicitly specified by that assignment. We try to give you problems with actual applications - as a result, much of what we ask you to do can be trivially solved by certain python modules. You can and should use these modules in your personal projects, but they circumvent the point of this class - learning how to write elegant, readable, and maintainable code.

On this assignment, you should not import any modules except for:

- You can (and should) import `string` for Problem 2 - `count_letters`
- You can import any modules you write yourself, like in Part 3 where you'll import the module you wrote in Part 2. In general, you are free to write and import as many modules as you'd like, so long as you do not circumvent the point of the assignment. Always submit any files necessary for your solution code to run.
- You can import `typing` if you would like. This can help you write *very* readable code, but it is not required in this class.

Grading

For this assignment, most functionality will be auto-graded. We will manually grade for structure and readability. Ensure that you provide good:

- docstrings
- names (for variables and functions)
- whitespace (spacing within a line, and spacing between lines)

We will manually deduct credit for problem 3 if you do not import and use `count_letters()` in your solution.

Submitting

Submit the following files:

- `fizz_buzz.py`
- `count_letters.py`
- `is_anagram.py`

Students must submit to Gradescope individually within 24 hours of the due date (homework due dates are typically Tuesday at 11:59 pm EST) to receive credit.