Mod 1 Homework - Basic Python

You'll write 3 functions in 3 different files (1 each, for a total of 3 functions) for this homework. We provide 1 of the files with skeleton code for the function; you will have to create the other two yourself. Gradescope will automatically check for correct file names, function names, and docstrings in required functions when you submit. (Note - every function you write in this class should have a docstring, but gradescope only automatically checks the functions we explicitly ask for. If you write extra functions, make sure to include docstrings for them as well).

Neccesary files:

- `fizzbuzz.py` (provided)
- `anagrams.py` (not provided)
- `letters.py` (not provided)

We also provide some files you do not need to submit, but will be helpful for locally testing `letters.py`:

- `frost.txt`
- `empty_file.txt`

Imports

We do not allow imports on any homework or lab assignments, except for those explicitly specified by that assignment. Due to the ubiquitous nature of the problems we are solving, there are often modules that trivialize the assignments when imported, so we restrict imports to ensure you're learning all relevant techniques.

Do not import any modules except for those you write yourself, or any exceptions below:

- You can import `string` for Problem 3 - `count_letters`

Problem 1 - `fizzbuzz.py`

Write a function `fizzbuzz(start, finish)` that prints numbers from `start` to `finish`. 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.

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

```
fizz
14
fizzbuzz
```

Problem 2 - `anagrams.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
```

Problem 3 - `letters.py`

Write a Python function `count_letters(file)` for counting letters in a text file.

- Input: name of file relative to the present working directory `pwd` (see below), e.g. `count_letters('frost.txt')` if `'frost.txt'` is in the pwd
- Output: return the dictionary of `letter:count` pairs. Only include letters present in the file.
- Use a dictionary to hold a mapping between a letter and its number of occurrences.
- 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`.
- **Reading files** - You can use `open` to open a file. It creates an iterator that goes through the file, line by line. The example below shows how to iterate over the provided file `frost.txt`, printing every line:

```
>>> f = open('frost.txt')
>>> for line in f:
...     print(line, end='')
...
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
>>> f.close()
```

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/
|    ...
```

  there are a few directories we could be in: `cse2050/`, `homework/`, `homework1/`, `homework2/`, `labs/`, `lab1/`, and `lab2/` are all valid directories in the tree pictured. Make sure your terminal is in the directory containing `frost.txt`, or you'll get an error - this commonly happend when VSCode is open in a high level directory, like `cse2050/`, and you click the "play" button to run a python script in a nested directory:

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

  My **p**resent **w**orking **d**irectory (`pwd`) is `cse2050/`, so my terminal cannot see `frost.txt`. It is able to find `letters.py` because VSCode supplies the absolute path (`home/jas14034/...`) to that file. 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
```

Grading

Most of the functionality will be auto-graded. We will manually grade for structure and readability - ensure that:

- Every function has a docstring (more info)
- Code is well commented - you don't need a comment on every line, but don't assume because *you* know what something does that a stranger (or you in two months) will understand it.
- You use consistent variable naming conventions (see e.g. the Python style guide)

Submitting

Submit the following files:

- `fizzbuzz.py`
- `anagrams.py`
- `letters.py`

Students must submit to Gradescope individually by the due date (typically Tuesday at 11:59 pm EST) to receive credit.

Challenges

Hankering for more coding practice? We'll occasionally provide stretch-goals or more difficult variants of homework problems. These are not for extra credit and are not required, they're just here in case you want to dive a bit deeper into the content.

Problem 3 Challenge

Write a Python function `letter_frequency(dict_letters)` for finding the frequency of each letter in a dictionary.

- As input, take a dictionary of `letter:count` pairs (output of the previous function)
- Find the relative frequency of each letter (the ratio between the number of its occurrences and the total number of letters)
- Return the new `letter:frequency` dictionary.
- In Python, passing mutable objects like dictionaries can be considered as a call by reference - if you modify the dictionary passed to this function, the original dictionary will also be modified! In this problem, that's a bad thing - don't modify the original dictionary. Instead, create a new dictionary for holding frequencies.
- Example:

```
>>> counts = letter_count('frost.txt')
>>> print(counts)
{`a`: 13, `b`: 2, (24 letter:count pairs omitted)}
>>> freqs = letter_frequency(counts)
{'a': 0.06190476190476191, 'b': 0.009523809523809525, (24 pairs omitted)}
```

Next, make a new file `highest_freq.py`. Import the functions you wrote in Part 1 and use them to implement a new function `highest_freq()`. `highest_freq(file)` should find the letter that has the highest frequency in a given txt file.

- Return letter and its frequency. Example `('e', 0.12451162240025257)`
- Use the imported functions from `letters.py` to do this
- Test your code with `assert` statements (see **Grading** section below)
- Example:

```
>>> ltr, freq = highest_freq("frost.txt")
>>> print(ltr, freq)
e 0.10952380952380952
```