Mod 1 Homework: Letter frequency

Letter frequency is the number of times letters of the alphabet appear on average in written language. According to Wikipedia, in the English language, the character "e" appears most often, then the characters "t", "a", and "o".

In this assignment, you will implement a letter frequency calculator to find i) the total number of times each letter occurs and ii) a percentage that shows how common the letter is in relation to all the letters in a text file.

Feel free to import the `string` module in this assignment. Do not import any other modules, except for code that you write yourself.

Part 1 - `letters.py`

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

- Input: the function takes a text file as an input.
- 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`.
- output: return the dictionary of `letter:count` pairs
- **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()
```

Note that your python script (`letters.py`) and the text file (`frost.txt`) need to be in the same directory for this to work. Also, **make sure you close the file after reading it**.

Next, 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)}
```

Part 2 - `highest_freq.py`

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
```

Submitting

Submit the following files:

- `letters.py`
- `highest_freq.py`

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

Grading

Broadly speaking, we'll grade this and all assignments on 4 main areas: Structure, Tests, Efficiency, and Readability (more info). We include some information/examples below for this assignment.

1) Structure

- file: `letters.py`
  - function `letter_count()`
    - input - text file
    - output - dictionary of `letter:count` pairs
  - function: `letter_frequency()`
    - input - dictionary of `letter:count` pairs
    - output - dictionary of `letter:frequency` pairs
- file: `highest_freq.py`
  - function: `highest_freq()`
    - input - a text file
    - output - highest frequency letter in form of a 2-tuple: `(letter, frequency)`

2) Tests

We'll cover testing in depth next week. For now, write a few `assert` statements to test your funcitons. **Print statements are not valid tests - use `assert` instead.**

`assert` takes a boolean as its input, and it raises an exception if that boolean evaluates as false:

```
>>> x = 2
>>> assert(x==2) # True:  nothing happens
>>> assert(x==3) # False: raises an error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

To write a test, define what you *expect* a function to produce, then use an assert statement to compare that to what the funciton *actually* produces; e.g.

```
expected_count = {'a':4,
                  'b':4,
                  'c':3,
                  'd':1,
                  ... # 22 lines omitted
                  }

actual_count - letter_count('test_file.txt')

assert(expected_count == actual_count)
```

You should test all your functions - that means you need at least three assert statements (2 in `letters.py` and 1 in `highest_freq.py`.)

`round` can be helpful when testing floats (e.g. `assert round(expected, 3) == round(actual, 3)`)

3) Optimization

Choose optimal data structures and write efficient code.

4) Readability

- Use whitespace to clump similar code together
- Use comments to describe what code does
- Use Docstrings for all of your functions. Docstrings are "documentation strings." You define them with a string at the top of your function. It's common to use triple-quotes for this string, since triple-quote strings can span multiple lines:

```
def sum_list(L):
    """
    Returns the sum of all items in L
    Fails if L is an empty collection
    """
    # initialize with first item
    temp_sum = L[0]

    # add all other items
    for i in range(1, len(L)):
        temp_sum += L[i]

    # return the sum
    return temp_sum
```

Every function you write in this course should include a docstring.