Objectives

* Creating, manipulating, and using dictionaries.
* Develop a function definition.
* Import function definitions.
* Use the assert function for testing.

Due Date

This assignment is due on Tuesday, January 23, 2024, by 11:00 pm.

**Remarks:**

* It is expected you develop test cases for your code.
* When you are asked to hand in code, you cut-and-paste the definition(s) from your code and paste into this Word document immediately following the activity.
* Include with your code several test case examples and your results.
* This homework is to be individual work.

Value

This assignment is worth 20 points.

Activities

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 in a text file and ii) a percentage that shows how common the letter is in relation to all the letters in the text file.

You can import the string module in this assignment if you wish. Do not import any other modules, except for code that you write yourself.

1. Create a file letters.py for the first two function definitions.
2. 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='') # The text file has a newline

# already at the end of each

# line.

f.close()

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

* 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.
* Provide a printout of your program letters.py below this bullet. Remember to put it in courier font. Include test cases and the results.

import string  
  
def letter\_count(file):  
 *'''Counts the instances of each ascii character from a file and returns an alphabetized dictionary'''* with open(file) as f:  
 count = {} #initializes new dictionary  
 data = f.read()  
 data = data.lower() #reads and forces all characters as lower case  
 for char in data:  
 if char in string.ascii\_lowercase:  
 count[char] = count.get(char, 0) + 1 #check a-z frequency with default value of 0  
 sorted\_count = {} #initializes and sorts dictionary in alphabetical order  
 for key in sorted(count):  
 sorted\_count[key] = count[key]  
 return sorted\_count  
  
def letter\_frequency(dict\_letters):  
 *'''Given a dictionary of letters and their values, returns the ratio or frequency of each character in relation to the others in an alphabetized dictionary'''* total = sum(dict\_letters.values())   
 freqs = {}   
 for letter, count in dict\_letters.items():  
 freqs[letter] = count / total   
 return freqs  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #first test  
 frostdict = {'a': 13, 'b': 2, 'c': 6, 'd': 10, 'e': 23, 'f': 12, 'g': 2, 'h': 12, 'i': 23, 'k': 2, 'l': 6, 'm': 3, 'n': 9, 'o': 20, 'p': 1, 'r': 14, 's': 14, 't': 20, 'u': 5, 'v': 2, 'w': 8, 'y': 3}  
 assert(letter\_count('frost.txt')) == frostdict  
 #second test  
 frostfreq = {'a': 0.06190476190476191, 'b': 0.009523809523809525, 'c': 0.02857142857142857, 'd': 0.047619047619047616, 'e': 0.10952380952380952, 'f': 0.05714285714285714, 'g': 0.009523809523809525, 'h': 0.05714285714285714, 'i': 0.10952380952380952, 'k': 0.009523809523809525, 'l': 0.02857142857142857, 'm': 0.014285714285714285, 'n': 0.04285714285714286, 'o': 0.09523809523809523, 'p': 0.004761904761904762, 'r': 0.06666666666666667, 's': 0.06666666666666667, 't': 0.09523809523809523, 'u': 0.023809523809523808, 'v': 0.009523809523809525, 'w': 0.0380952380952381, 'y': 0.014285714285714285}  
 assert(letter\_frequency(letter\_count('frost.txt'))) == frostfreq

"C:\Users\artjs\OneDrive - University of Connecticut\Documents\Spring 24\CSE 2050\cse2050\hw1\.venv\Scripts\python.exe" "C:\Users\artjs\OneDrive - University of Connecticut\Documents\Spring 24\CSE 2050\cse2050\hw1\letters.py"

Process finished with exit code 0

1. 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 a new letter:frequency dictionary.
* Remember: 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)

**>>>** print(freqs)

{'a': 0.06190476190476191, 'b': 0.009523809523809525,

*(24 letter:frequency pairs omitted)*}

1. Make a new file highest\_freq.py.

* *Import* the functions you wrote in Part 1 (not include them) and use them to implement a new function highest\_freq(file) that finds the letter that has the highest frequency in a .txt file.
* Return letter and its frequency. Example:

|  |
| --- |
| **>>>** ltr, freq **=**highest\_freq("frost.txt")  **>>>** print(ltr, freq)  e 0.10952380952380952 |

* Test your code with assert statements (see “Grading” section below)
* Provide a printout of your program highest\_freq.py below this bullet. Remember to put it in courier font. Include test cases and the results.

import letters  
def highest\_freq(file):  
 *'''Given a file name, calls functions from letters.py and returns the letter with the highest frequency and its ratio'''* letter\_count = letters.letter\_count(file)  
 letter\_freq = letters.letter\_frequency(letter\_count)  
 keymax = max(letter\_freq, key=letter\_freq.get)  
 return keymax, letter\_freq[keymax]  
  
ltr, freq = highest\_freq('frost.txt')  
#third test  
assert(ltr) == 'e'  
assert(freq) == 0.10952380952380952  
  
"C:\Users\artjs\OneDrive - University of Connecticut\Documents\Spring 24\CSE 2050\cse2050\hw1\.venv\Scripts\python.exe" "C:\Users\artjs\OneDrive - University of Connecticut\Documents\Spring 24\CSE 2050\cse2050\hw1\highest\_freq.py"

Process finished with exit code 0

# Grading

Broadly speaking, we’ll grade this and all assignments on four main areas:

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 tuple:  
 (letter, frequency)

1. Tests
   * Write assert statements to test your functions.

*Print statements are not valid tests - use assert instead.*

* + assert takes a boolean as its argument, 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 function actually produces. For the letter\_count function:

|  |
| --- |
| 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 (two in letters.py and one in highest\_freq.py.)
  + round can be helpful when testing floats (e.g., assert round(expected, 3) == round(actual, 3))

1. Optimization
   * Choose optimal data structures and write efficient code.
2. Readability
   * Use whitespace to group similar code together
   * Use comments to describe what code does
   * Use docstrings for all 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. See example:

|  |
| --- |
| **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.**