Project 01 - Counting Characters

Due Jan 28 by 11:59pm **Points** 50 **Submitting** a file upload

Overview

In this project, you will write a Python script that counts the frequencies of characters in a text file. For example, if we have a text file containing the characters:

```
Abracadabra!
```

In this file, the following letters occur with the following frequencies:

Letter:	a	b	c	d	r	A
Count:	4	2	1	1	2	1

Our code should calculate those frequencies and print out the results, but how should those results be displayed? There are many choices, but the real question should be: how we are going to **use** the output?

One choice is to print the data in a way that users could easily read. For example:

```
The letter "a" appeared 4 times.
The letter "b" appeared 2 times.
The letter "c" appeared 1 times.
The letter "d" appeared 1 times.
The letter "r" appeared 2 times.
The letter "A" appeared 1 times.
```

While this is human-readable, it's not easy to use with other applications. What if we wanted to graph these frequencies? We'd have to translate this data into another form.

A better way is to output the data in a form that another application can understand. One such form is **CSV (https://en.wikipedia.org/wiki/Comma-separated_values)**, short for comma-separated values. In this format, each character and count appear on their own line, separated by commas:

```
"a",4
"b",2
"c",1
"d",1
"r",2
"A",1
```

This can still be read by the user, but CSVs can also be read by many other applications, like Excel or other graphing software!

What Options Do We Want?

There are some variations in how characters could be counted, such as:

- Only counting certain characters, like vowels.
- Ignoring case, so that *'a'* and *'A'* are counted as the same letter.
- Counting characters in multiple files.
- Printing out lines for characters with a frequency of zero.

All of these options are reasonable features, because they represent common ways that application could be used. Our script should support these features, so that we only have to write the code once!

To support those options, our code will support the following arguments:

```
python3 count.py [-c] [-l letters] [-z] file_1 file_2 ... file_n
```

The flags allow us to change those options:

- *-c* :: an optional flag that distinguishes between upper and lower case. For example, the file *'aA'* would count one *'a'* and one *'A'*.
- *-l* :: an optional flag with an argument, that only prints out the frequencies of the characters in the argument *letters*. For example, *'-l aeiou'* counts only vowels.
- *-z* :: an optional flag that prints a row for every character, even when it occurs zero times.

Examples

Suppose we have a text file named *test.txt* containing the following characters:

```
AA bb Cc dD
```

By default (with no flags), your script should print out the frequencies for ASCII letters only, convert all letters to lowercase, and only print characters that occur in the file.

With No Flags: *python3 count.py test.txt*

```
"a",2
"b",2
"c",2
"d",2
```

With -c Flag: *python3 count.py -c test.txt*

```
"b",2
"c",1
"d",1
"A",2
"C",1
"D",1
```

With -l Flag: *python3 count.py -l ab test.txt*

```
"a",2
"b",2
```

With -z Flag: *python3 count.py -z test.txt*

```
"a",2
"b",2
"c",2
"d",2
"e",0
"f",0
"g",0
"h",0
"i",0
"j",0
```

```
"k",0
"l",0
"m",0
"n",0
"o",0
"p",0
"q",0
"r",0
"s",0
"t",0
"u",0
"v",0
"w",0
"x",0
"y",0
"z",0
```

What to Do

Create a script *count.py*. At the top of this script, add a comment containing your name, student ID number, and a brief description of the program. In this script, you will implement two functions, described below.

Each function should have a docstring briefly describing that function.

add_frequencies(d, file, remove_case)

Write a function *add_frequencies(d, file, remove_case)*, which takes three parameters: a dictionary *d*, a file object *file*, and a boolean *remove_case*. The dictionary maps characters to frequencies. Your function should add the frequency counts of the characters in the file *file* to the dictionary *d*.

If *remove_case* is false, then the each character is the key into the dictionary. If *remove_case* is true, then the lowercase of each character is the key into the dictionary.

For example, calling *add_frequencies* with an empty dictionary, *test.txt* from the *Examples* section, and *remove_case* equal to *True* would yield:

```
{
   "a":2,
   "b":2,
   "c":2,
   "d":2
}
```

Calling *add_frequencies* with an empty dictionary, *test.txt* from the *Examples* section, and *remove_case* equal to *False* would yield:

```
{
   "b":2,
   "c":1,
   "d":1,
   "A":2,
   "C":1,
   "D":1
}
```

main()

Write a *main()* function, which handles parsing the arguments, and calling *add_frequencies(...)*. It should perform the following steps:

1. Parse the command line arguments *-c*, *-l*, *-z*
2. Create an empty dictionary
3. Add the frequencies for each file in the argument list to that dictionary
4. Print out the elements of that dictionary in CSV format

Hints

1. Do this without options first. Work incrementally, by starting by outputting the frequency counts for every character in a single file. Then, move to multiple files. Then, specify which characters you actually want to print out.
2. Add options *one at a time*. The boolean flags *-c* and *-z* are the simplest.
3. Add the string flag *-l* last. Think about how this relates to how you specified which characters to print out earlier.

What to Submit

When you are finished, please both upload your code to canvas, and put your file in a folder called Project1 and commit your *count.py* to your git repository.

It would be best to have a comp 3006 repository that you will add each project (and other practice) to. Please add myself and Sunny to the repository as collaborators

To do this, log into github in your browser. Go to your repository. Go to the Settings tab on top, then the manage access tab on the left side. Then choose invite a collaborator (and give us max access)

The Emails to add are: **dcrutch97@comcast.net (mailto:dcrutch97@comcast.net)** (for Me) and _______ (for Sunny)

Project 01 - Counting Characters - Rubric

Criteria	Ratings	Pts
Comment at beginning Required comment appears at the beginning of the script, containing the student's name, id, and a short description of the program.		2 pts
Coding Style Comments are used well to explain the code. Variables are named reasonably. Indentation matches expectations. Named constants are used instead of placing values directly into the code. Code is generally readable. Etc.		6 pts
add_frequencies() :: Docstring add_frequencies(...) function contains a docstring briefly describing what it does.		2 pts
add_frequencies() :: Functionality add_frequencies(...) behaves as described. Takes proper parameters, updates required dictionary, uses proper keys for each character (using remove_case to determine proper keys).		6 pts
main() :: Argument Parsing Parsing code separates out flags and files. Processes each flag appropriately.		6 pts
main() :: File processing Files in the argument list are processed as expected, with each having its character frequencies assessed.		6 pts
main() :: Dictionary Printing Dictionary is printed in CSV format, with each character, a comma, then its frequency. Only keys from a requested set are printed.		6 pts
-c Option -c option implemented correctly, to make the counting case-sensitive. When omitted, defaults to being case-insensitive.		4 pts
-z Option -z option used to print out chars that occurred with a frequency of zero. If omitted, chars with a frequency of zero are not printed.		4 pts
-l Option -l option used to restrict what characters are printed. The dictionary printing should reflect these options, only printing out the		6 pts

Criteria	Ratings	Pts
requested keys.		
Uploaded to Git the project is uploaded correctly to a git repository that can be accessed by the grader		2 pts
		Total Points: 50