Are you Old Enough to Read This?

Computing Reading Levels

Goals: (0) test your Anaconda version of Python using the IDE Visual Studio (VS) Code; (1) convert an existing Python script to a Jupyter Notebook; (2) demonstrate good Python documentation using docstrings and type-hinting; (3) show how Python scripts can produce output (.csv) that can be (later) professionally formatted in Tables using a spreadsheet (e.g., Excel, Google Sheets); and (4) test your creative juices regarding counting syllables.

An ability to write well is crucial in today's fast-paced, no-time-to-waste climate. One ingredient of being a "good writer" is to know your audience. If you write too simply for a sophisticated audience, the reader will ignore your message. On the other hand, if you write at a level beyond the reader's ability to comprehend, you've missed your "teachable moment". Getting your message across is very important. **But how can you determine if your writing is at an appropriate level for your reading audience?**

For years, popular word processors (e.g., Microsoft's Word©) have provided an option to report on the "grade level" of the document you are editing. Third party tools (e.g., Readable) are available, but of course, you can ask your chatBot (e.g., chatGPT). **This assignment charges you with computing and reporting a reading level for a given document.** The needs for such an analysis are many: (i) teachers want to produce appropriate level instructions, (ii) drug companies want patients to understand the side effects of their products, and (iii) companies and politicians are passionate about the degree to which their messages best target their audience.

Two of the more popular measures are: (i) The Flesch Reading Ease and (ii) Flesch-Kincaid Grade Level (see Wikipedia: "Flesch–Kincaid readability test" at: http://en.wikipedia.org/wiki/Flesch–Kincaid_readability_test). These measures are based on a set of constants and three variables: (1) total number of words, (2) total number of sentences, and (3) total number of syllables. **Your program must read in all the words from a file of text, compute the values of these three variables, and produce an spreadsheet-ready report in a new "comma separated" output file (.csv file extension).**

Counting syllables in words is the most challenging part of determing these measures.

> **syllable**, n. Pronunciation: /ˈsɪləb(ə)l/
> a. A vocal sound or set of sounds uttered with a single effort of articulation and forming a word or an element of a word; each of the elements of spoken language comprising a sound of greater sonority (vowel or vowel-equivalent) with or without one or more sounds of less sonority (consonants or consonant-equivalents); (Oxford English Dictionary)

So, a syllable is the sound of a vowel (a, e, i, o, u) that's created when pronouncing a word. In general, the number of times that you hear the sound of a vowel (a, e, i, o, u) in a word is equal to the number of syllables the word has.

How To Find Syllables[1]:

(1) Count the number of vowels ('a', 'e', 'i', 'o', 'u', and sometimes 'y') in the word.

(2) Subtract any silent vowels (like the silent 'e' at the end of a word), taking care to note certain exceptions, e.g., "the", "middle", etc.

(3) Subtract 1 vowel from every diphthong. A diphthong is when two vowels make only one sound; that is:

```
diphthongList = ["oi", "oy", "ou", "ow", "ai", "au", "ay", "aw", "oo", "ie", "ea", "ee"]
```

(4) The number you are left with should be the number of syllables in the word.

[1] The serious scholar will want to investigate the six different types of syllables. Above-average effort will be awarded to solutions that count syllables in additional ways, e.g., catching exceptions (see the Grade Key).

How can I test if my program is actually counting syllables correctly?
Consider including some haiku poems in your early **test suite**. A haiku *must be exactly* three (3) lines and the lines must follow the 5-7-5 (syllable) format:

The first line contains *only* 5 syllables.	***I am first with five***
The second line contains *only* 7 syllables.	***Then seven in the middle***
The third line contains *only* 5 syllables.	***Five again to end***

What other files might I test?
Consider your own papers. How about that last history paper? What grade level do you think it will be rated as? What about a children's story? Consider visiting `gutenberg.org` to find full novels. Note: you should use MS Word's reading level as a benchmark. While your program need not produce *exactly* the identical value, on very short files one might assume that your program's values will be quite similar to MS Word's level.

Requirements:
You must use *many* functions for this assignment. This is the review of your initial semester of Python programming. Note that each function does one task and only one task.

`getData()`: Accepts no arguments; Returns a string. Prompts the user to enter a filename from the keyboard (stdin) and if that file exists, reads and stores the entire file into a string variable to return. Note: since another function will later need to determine the number of sentences in your file, this function should insert some ending-sentence-delimiter into your string before returning it. For example, you might replace all occurrences of periods (.), question marks (?) and exclamation points (!) with some marker such as " XXX ", thus the end of each sentence will be delimited ("marked") with your symbol, thus making it easier in some later function to split the string (the entire file's contents) into individual sentences.

`remove_punctuation(s)`: Accepts one argument, `s` (a string) and returns the contents of that same string except with all punctuation symbols removed.

`syllablesPerWord(word)`: Accepts one argument, a single `word` (a string that contains *only* one word). Using the rules below, this function returns the number of vowels in that word, where the result must be at least one or greater.

(1) Count the number of vowels ('a', 'e', 'i', 'o', 'u', and sometimes 'y') in the word.
(2) Subtract any silent vowels (e.g. the silent 'e' at the end of a word); take care to consider exceptions.
(3) Subtract 1 vowel from every diphthong. A diphthong is when 2 vowels make only 1 sound.

`getNumberOfTotalSyllables(s)`: Accepts one argument, `s` (a string) that contains the contents of the entire input file (as previously returned by the `getData()` function). This function should return the total number of syllables in the string.

`getNumberOfTotalWords(s)`: Accepts one argument, `s` (a string) that contains the entire file. This function should return the total number of words in this string. Note: care must be taken *not* to count any delimiter tokens (e.g., " XXX ") if those are still present.

`getNumberOfTotalSentences(s)`: Accepts one argument, `s` (a string) that contains the entire file. This function should return the total number of sentences in this string.

`compute_FleschReadingEase(s)`: Accepts one argument, `s` (a string) that contains the entire file. This file should return the Real number (float) indicating the Flesch Reading Ease score. See Wikipedia for the definition and formula.

`compute_FleschKincaidGradeLevel(s)`: Accepts one argument, `s` (a string) that contains the entire file. This file should return the Real number (float) indicating the Flesch-Kincaid Grade Level score. See Wikipedia for the definition and formula.

`printReadingLevelReport()`: (Accepts as many arguments as you deem necessary, but do *not* rely on global variables). This function should open a new file for writing (open mode = `'w'`) named: "`report.csv`", where the file extension **.csv** means "comma-separated values". All output values written to this file should thus be comma separated, including headings. When your program finishes, the user should find a *new* file named "`report.csv`" located in the local directory where the Python code is located. This .csv file can be opened later by a spreadsheet program (e.g., Google Sheets, Excel). Even though your Python program can create and output values into spreadsheet-ready columns, the file format will be very "bland". To make your output look more professionally formatted, you should (0) run your program, noticing the new `report.csv` file in your local directory; (1) open that file in a spreadsheet tool, e.g., Sheets, and (2) make the Table output look professional by (i) centering columns, (ii) using color/bold on headings, (iii) including an appropriate caption at the bottom that explains the data, and (iv) making borders on the cells, but not on the bottom caption so it appears to "float" below the Table. A sample output file `report_haiku1.csv` created from the input "`haiku1.txt`" and formatted professionally, including a nice caption at the bottom row (the row using merged columns) is shown below in Table 1.

Syllables	Words	Sentences	Reading Ease	Reading Level
17	14	3	99.37	0.56

Table 1. Reading level statistics when reading "haiku1.txt".

`printTopNwords(s, N)`: Accepts two arguments, `s` (a string) that contains an entire input file (e.g., a story) and an integer, `N` indicating the number of most frequently occurring words in this file that should be reported. For example, if `N` was 10, this function should compute and write (mode=append, `'a'`) the top-10 words and their respective frequency counts to the end of your output file, "`report.csv`" in two columns as shown below. This function does not have to return anything.

```
the          113
a            101
some          59
foo           37
 :             :
```

Grading:

You will convert the existing Python code into a Jupyter Notebook (**`.ipynb`**). You follow Python's docstring recommendations (https://realpython.com/documenting-python-code/) and document: (a) the entire Notebook at the top (of course, in a markdown cell), (b) each function includes it's own docstring, (c) each function uses **type hinting**, and (d) "inline" `#documentation` that explains blocks of the code to a reader.

I will grade this assignment with a more flexible, open-ended rubric. **Please make a .zip file of your (1) `.ipynb` and (2) a `.pdf` file showing formatted Tables of results, and submit the .zip.**

Average Effort: Your Notebook (i) computes stats, including the Reading Level and Ease, (ii) prints the top-N most frequently occurring words, and (iii) writes these values to a comma-separated (`.csv`) file.

Above Average Effort: Your program implements *all* of the functions listed above and documents those files appropriately. You submit a hard copy (printout of your .pdf) of a professionally formatted report that (i) includes a short summary of the analysis and results, (ii) two tables of results for one experimental run (Table 1 is shown above and Table 2 should format your top-N words). Each Table must contain a "floating" row without borders that contains merged-cells (bottom row) with a nicely worded caption.

Superior Effort: In addition to completing "Above Average Effort" you also demonstrate your own creative twist to determining "reading level". Your effort should be well documented, that is, the reader should be able to tell from your opening documentation how *you* are determining reading level. Some items you might implement include:

(a) Use a better name for your output (report); using the input filename, "mangle" that name with the prefix "`report_`" (e.g., "`report_haiku1.csv`" or "`report_MobyDick.csv`") so each of the input files that you test results in its own output file; that is, not all runs will over-write the file "`report.csv`".

(b) handle more situations for adequately counting syllables

(c) handle more situations for dealing with punctuation, e.g., should the word "**don't**" really be counted as "**don**" and "**t**"?

(d) Give the reader more help when interpreting the output, for example, for reading ease, print messages that follow guidelines depending on your calculated value:

Score	Notes
90.0–100.0	easily understood by an average 11-year-old student
60.0–70.0	easily understood by 13- to 15-year-old students
0.0–30.0	best understood by university graduates

(e) *Be creative …*