# Are you Old Enough to Read This?
Computing Reading Levels

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". Clearly, 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. Google's (advanced) search options now also allow the user to filter search results by "reading level" (Basic, Intermediate, Advanced). **This assignment charges you with computing and reporting a reading level for a given document.** The needs for such an app are many: (i) teachers want to produce appropriate level instructions; (ii) drug companies want patients to understand 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 Excel-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); also, a character or set of characters forming a corresponding element of written language. (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.

**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.

**Requirements:**
You must write *many* functions for this assignment. This is the culmination of your semester of programming and you will be evaluated on your good use of writing many (small) functions, where each function does one task and only one task. Your set of **professionally documented** functions *must* include at the very least:

**`getData()`**: Accepts no arguments. Prompts the user to enter a filename to open and if that file is available, 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, you 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 clearly 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.
(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. Double-clicking this file should cause Excel to open this file (Excel is set to automatically open `.xls`, `.xlsx`, `.tsv` (tab-delimited values), and `.csv` files by default). Even though your Python program can create and output values into Excel-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) double-click that file and then SAVE-AS this file, taking care to save the file in Excel-Workbook format; and (2) then using Excel, make the output look professional by (i) centering columns, (ii) using color/bold on headings, (iii) making borders on the cells, a legend, etc. A sample output file **`report_haiku1.txt`** after using the input "**`haiku1.txt`**" is shown below.

| **Vowels** | **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**:
I will grade this assignment with a more flexible, open-ended rubric.

**Average Effort**: Your program computes (i) at least one of the reading measures, (ii) prints the top-N words, and (iii) prints the values to an Excel (`.csv`) file. All documentation and good programming practices as required all semester are in effect.

**Above Average Effort**: Your program implements *all* of the functions listed above and documents those files appropriately. You submit a hardcopy of a professionally formatted Excel table of results for one experimental run (This hardcopy printout is due on Wednesday, December 07 in class!)

**Superior Effort:** In addition to completing "Above Average Effort" you also demonstrate your own creative twist to determining "reading level". Google's got nothing on you! 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); save the input filename; "mangle" that name with the prefix "**`report_`**" (e.g., "**`report_haiku1.txt`**" or "**`report_TaleOfTwoCities_start.txt`**") so each of the input files that you test results in its own output file, that is, not all runs will write over 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 ...*