CSCI 111, Lab 4
Text processing

Due date: Midnight, Tuesday, October 4, on Canvas. No late work accepted.

File names: Names of files and variables, when specified, must be EXACTLY as specified. This includes simple mistakes such as capitalization.

Individual work: All work must be your own. Do not share code with anyone other than the instructor and teaching assistants. This includes looking over shoulders at screens with the code open. You may discuss ideas, algorithms, approaches, *etc.* with other students but NEVER actual code.

One dollar words: Suppose words are worth something based on their letters. An "a" is worth one cent, a "b" is worth two cents, and so on until a "z" is worth 26 cents. The value of a word is the sum of the value of its letters. So, for example, 'word' is worth:

$$val(w) + val(o) + val(r) + val(d) = 23 + 15 + 18 + 4 = 60¢$$

How many one dollar words are there in the English language?

To answer this question we'll assume all words in the English language are in our file, `words.txt`. Assume upper and lowercase letters are worth the same. Write a module called `dollarwords.py` with the following functions (make sure you test all of them):

- `letter_value(c)`
  Returns 1 for 'a' or 'A', 2 for 'b' or 'B', *etc.*
- `word_value(w)`
  Returns 2 for 'aa', 10 for 'babe', 127 for 'syzygy', *etc.*
- `is_dollar_word(w)`
  Returns `True` if `w` is worth exactly 100, `False` otherwise.

In the same file put a loop over the lines of `words.txt` that counts the number of one dollar words. The last line prints out the total number of words, the total number of one dollar words, and the percentage of words that are worth exactly one dollar.

Your output should look similar to this, but with different numbers. These are just the two bit words (25 cents):

```
Total number of words      : 113783
Total number worth 25 cents: 70
Percentage of total        : 0.061520613799952543
```

Examples of two-bit words: abided, all, behead, bard, cabbaged, fame, pi.

Readability indices: Over the years there have been a number of readability indexes designed to determine, roughly, how difficult it is to read a certain text. Most involve counting the number of sentences, words, and syllables or characters in the text, and

then computing a formula. The idea is that if the text has a lot of big words (lots of letters or syllables per word), and/or a lot of long sentences (lots of words per sentence), then it should be harder to read.

For example, here are three in common use:

Flesch-Kincaid grade level:

$$11.8\left(\frac{\text{syllables}}{\text{words}}\right)+0.39\left(\frac{\text{words}}{\text{sentences}}\right)-15.59$$

Flesch reading ease:

$$206.835-84.6\left(\frac{\text{syllables}}{\text{words}}\right)-1.015\left(\frac{\text{words}}{\text{sentences}}\right)$$

Automated readability index:

$$4.71\left(\frac{\text{letters}}{\text{words}}\right)+0.5\left(\frac{\text{words}}{\text{sentences}}\right)-21.43$$

Coleman-Liau index:

$$5.88\left(\frac{\text{letters}}{\text{words}}\right)-29.6\left(\frac{\text{sentences}}{\text{words}}\right)-15.8$$

Each attempts to give a rough grade level for the reading material, except the Flesch reading ease which attempts to score reading material from about 0 to 100 in ease of reading. Of course, there are exceptions. One sentence in *Moby Dick* has a reading ease of -146.77.

Counting letters: In any event, each of them requires counting the number of words and the number of sentences in a text. Two of them need the number of syllables, and the other two need the number of letters. Basically, each says that shorter words and shorter sentences are easier to read.

How do we count these things in a text? Counting letters is easy. Simply run through the text and count how many letters show up. How do you know if a character is a letter? The `string` module provides a number of convenient strings, such as:

- `string.ascii_letters`
- `string.ascii_uppercase`
- `string.ascii_lowercase`
- `string.punctuation`
- `string.whitespace`

If a character is in the right string, it is a letter.

Counting tokens: Counting sentences, however, is a bit more tricky. We know sentences start with an uppercase letter, and end with a character in `'.!?'`. The trick is to run through the file and note whenever a sentence begins, and when it ends. Every time we find the end of a sentence, we can count it.

This idea is illustrated in my program `ones.py`, available in the same folder as these lab notes. It counts the number of substrings in a file that consist only of the digit `'1'`. Obviously, such strings begin with a one, and end with any other character. A boolean variable `in_ones` keeps track of whether we are inside a string of ones. It is set to true at the beginning, and false at the end.

This same idea can be generalized to count sentences in a text file. We can count words, too. They begin with a letter and end with anything that isn't a letter.

What about syllables? These are very tough. They generally depend on a vowel between non-vowels, but some people pronounce "evening" with two syllables, and some three. Some words ending with a single 'e' have a syllaBLE there, and soME don't.

We will take a pragmatic approach. Syllables start with a vowel in `'aeiou'` and end with a character not in that string. We will count some extra syllables, and miss some others (why?), but hope that our count of syllables won't be too far off the mark.

Global variables: You will notice that my program, `ones.py`, uses two global variables. You will need at least six. I know I told you once to eschew global variables, but this will be an exception. Later on, we'll see a better way to handle this problem, using objects, but for now just use global variables.

Notice! Global variables must be declared in each function that accesses them. There are some exceptions to this rule, but it is always best to have an overabundance of clarity! Declare all global variables in all functions that use them!

The program: Write a module called `reading.py` following these ideas that takes some texts, such as the ones provided in this lab folder, and calculates the four reading indices above. Your output should be similar to those provided at the end of this document. Your numbers may be slightly different from mine, but not by much.

In your module, you should automatically process *Moby Dick*, *Green Eggs and Ham*, and one other document of your choice from Project Gutenberg. The print out should look similar to my results, below, but with only three documents. **Include all three documents in your zipped handin so I can run your program without downloading anything else or copying documents from my folders to yours!**

Note: during development it's probably wise to create a simple file. For example, a file with three sentences of three or four words each (they don't even have to be real words). Then you know your program is working right if it gets the right answer for very small files. You should include some lines above and below your text, with `*** START` and `*** END` marking them off so you test that part, too (see below).

UTF-8: You can find lots of good texts at Project Gutenberg:

- `https://www.gutenberg.org/`

The ones I've provided were downloaded from there.

IMPORTANT! Make sure you download the plain text versions, which are marked as "UTF-8" on Project Gutenberg. They will have no formatting information, just the characters themselves. UTF-8 is a generalizaiton of ASCII, but to process these files in Python you have to tell it that the file is encoded this way:

```
    fin = open(fname, encoding='utf8')
```

Skipping the beginning and end:

ALSO IMPORTANT! Files from Project Gutenberg include a lot of information about themselves at the beginning and end of the file. You should NOT use these sentences to contribute to your calculations. You should skip all lines until you get to the line after the one that starts with

```
*** START
```

and you should ignore all lines starting with the one that starts with

```
*** END
```

You can easily make some test files with these lines in them to make sure your program ignores them and anything outside of them.

Turn in: Zip your programs into a `lab04` folder and turn in on Canvas before next Tuesday.

Remember to include the text documents from Project Gutenberg in your folder before you hand it in!

A sample run from my solution to the reading ease problem

```
================================
lordjim.txt 11305 lines
Sentence count:  8001
Word count:      132430
Syllable count:  182332
Character count: 550378
CLI: 6.848894057237786
FKG: 7.111607315886847
ARI: 6.420562108698881
FRE: 73.55624588906113
================================
mobydick.txt 22276 lines
Sentence count:  10423
Word count:      219273
Syllable count:  311725
Character count: 955784
CLI: 8.423178959561824
FKG: 9.389822692095116
ARI: 9.619018512830174
```

```
FRE: 65.2121524236065
==================================
senseandsensibility.txt 12703 lines
Sentence count:  5971
Word count:      120869
Syllable count:  174364
Character count: 525803
CLI: 8.316854114785428
FKG: 9.32716434233193
ARI: 9.18072687397012
FRE: 64.24586045590115
==================================
tarzanoftheapes.txt 11035 lines
Sentence count:  4556
Word count:      87142
Syllable count:  124254
Character count: 378028
CLI: 8.160283674921395
FKG: 8.694856877965872
ARI: 8.565737350291528
FRE: 66.79181728347318
==================================
thesunalsorises.txt 10259 lines
Sentence count:  8095
Word count:      70604
Syllable count:  87384
Character count: 270171
CLI: 3.3064738541725696
FKG: 2.4159819189361897
ARI: 0.9540983659778597
FRE: 93.27590439360452
==================================
thefederalistpapers.txt 19961 lines
Sentence count:  6519
Word count:      196335
Syllable count:  325066
Character count: 949296
CLI: 11.647465199786076
FKG: 15.692674057909631
ARI: 16.401915074246382
FRE: 36.19619531012103
==================================
greeneggsandham.txt 137 lines
Sentence count:  139
Word count:      802
Syllable count:  3
Character count: 2418
CLI: -3.2021945137157104
FKG: -13.295644521789052
ARI: -4.344634098207717
FRE: 200.66221021188036
```