**CMPUT 274/5 - Tangible Computing** **Version 2.1-20**
**Weekly Exercise #3**

---

## Word Frequency

---

In this assignment, you will perform frequency analysis on words in an input text. You will read in a text from an input file provided as a command-line argument, divide the text into individual words, and write a sorted frequency table to an output file.

### Your Task, Part 1: Frequency Analysis

Write a program called `freq.py` that will perform text analysis on an input text using all of the following steps:

1. Take the name of an input file called `<input_file_name>` as a command line argument. For example, your program might be called using

   ```
   python3 freq.py example
   ```

   to direct you to process a plain text file called `example`. More information is given on how to do this below.

2. Read the contents of the file into your program and divide the text into a list of individual, space-separated words (for our purposes, a word is any sequence of non-blank characters).

3. Analyze the word list to count the number of times each word appears in the text. **Hint:** You may want to use a dictionary to store and track the counts.

4. When the counting is finished, your program should write a frequency table of each word, its count, and its relative frequency to an output file. The frequency table **must be sorted in lexicographic order (alphabetical with uppercase words first) by word in the text**. You may want to look up the `sorted` function to help you with this. Each line in the frequency table should look like:

   ```
   word count freq
   ```

   where

   - **word** is the word in the text
   - **count** is the number of times the word mentioned occurs
   - **freq** is a number in the range [0,1] that is the ratio of the count for the word to the total number of words found in the text. You will need to calculate this frequency once you have counted the number of times each word appears. **You must print the frequency rounded to 3 decimal places using the `round` function**.

You must write this frequency table to an output file. **The output file must have the name `input_file_name` + ".out".** For example, in the example above, the output file would be named "example.out".

### Processing Command Line Arguments

When you enter a command into the unix shell, the tokens on the line are passed as a list, `sys.argv`, to your program. Note that in order to access this list of arguments, you must import the module `sys`. You then have to interpret them as either options or file names.

For example this program, cli.py (available on eClass), shows all the options passed to it. Note that blanks are used to divide tokens, so to have a blank in an option you need to place it in quotes.

```
import sys
for t in sys.argv:
    print("|{}|".format(t))
```

This command:

```
python3 cli.py arg "hello" ' ' "" "two tokens" last
```

outputs

```
|cl1.py|
|arg|
|hello|
| |
||
|two tokens|
|last|
```

Note how the program name is the first argument in the list.

You can use this method to access a command-line argument given as input to the program from the terminal. If you are familiar with another method (eg, argparse) you are welcome to use that instead given that it produces the same result.

**Guarantees:**

- The input file given as a command line argument will always be located in the same directory as `freq.py`.
- The input file will always be plaintext.
- You can assume that the output file destination is either empty or does not exist when the program begins to run (ie, you do not have to worry about clearing the output file in between runs).

**Sample Input and Output Files:**

On eClass (or here), you will find a folder called `sample_data`. Download this folder to access three example inputs and outputs. Each input file is in the form "sampleX" (X is the test number) where the corresponding output file is called "sampleX.out". You can check if

your output matches the expected output exactly using the `diff` command. This command allows you to compare files line-by-line.

**Example Usage** (to compare `file1` and `file2`):

```
diff file1 file2
```

If the `diff` command returns nothing, the files have the same contents. You may also want to type "man diff" into the terminal to read the documentation of `diff`.

## Your Task, Part 2: Error Handling

The proper usage of your program, `freq.py`, is:

```
python3 freq.py <input_file_name>
```

where `<input_file_name>` is replaced by the name of an input file in the same directory as the program.

A user of your program may forget to include the command-line argument specifying the name of an input file. In this section, you will handle the cases where a user calls your program using too many or two few command-line arguments.

Modify the file `freq.py` to do the following:

1. If the user forgets to include the name of the input file, for example calling the program using:

   ```
   python3 freq.py
   ```

   you should:

   (a) inform the user (print) that there are **too few** command-line arguments, and demonstrate the correct usage of the program.

   (b) immediately exit the program.

2. If the user includes too many command-line arguments, for example:

   ```
   python3 freq.py hello 42 input
   ```

   you should:

   (a) inform the user (print) that there are **too many** command-line arguments, and demonstrate the correct usage of the program.

   (b) immediately exit the program.

If you wish to do additional error handling, such as checking that the filename is a string or that it exists in the directory, you may do so. However, it is not required and will not result in either additional marks or deductions.

**Requirements:**

In your final submission:

1. We expect you to **produce modular, well-designed code**. This means creating one function to do each job (there are multiple jobs in this assignment and you should not use only one function!). You are responsible in this exercise for determining how to structure your code to follow this requirement, and you may determine on your own which functions to create. **A correct solution that does not use any functions (ie, all code is in the global scope) will receive 0 marks for code design.**
2. We expect that when the program is called from the command line as specified above, your program will run the frequency analysis and write the result to the correct output file. We will not import your functions to grade this exercise; instead, we will run your entire program directly from the command line. This means that what you include under `if __name__ == "__main__"` or outside of any functions is important, because it will be run during grading!

**Submission Guidelines:**

Submit all of the required files (and no other files) as **one** properly formed compressed archive called either `freq.tar.gz`, or `freq.tgz`, or `freq.zip` (for full marks, please do **not** use `.rar`):

- when your archive is extracted, it should result in exactly *one directory* called `word_frequency` (use this exact name) with the following files in that directory:
- `freq.py` (use this exact name) contains all of your Python code
- your `README` (use this exact name) conforms with the Code Submission Guidelines.
- No other files should be submitted.

Note that your files and functions must be named **exactly** as specified above. A new tool has been developed by the TAs the help check and validate the format of your `tar` or `zip` file *prior* to submission. Instructions for how to use this validator tool will be posted to the eClass forum shortly.

If your submission passes this validation process, and all validation instructions have been followed properly, you will not lose any marks related to the format of your submission. (Of course, marks can still be deducted for correctness, design, and style reasons, but not for submission correctness.)

When your marked Weekly Exercise is returned to you, there is a 7-day window to request the reconsideration of any aspect of the mark. After the window, we will only change a mark if there is a clear mistake on our part (e.g., incorrect arithmetic, incorrect recording of the mark). At any time during the term, you can request additional feedback on your submission.