Software Engineering 265
Software Development Methods
Summer 2021

Assignment 2

Due: Wednesday, June 30, 11:55 pm by submission via git
(no late submissions accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the virtual machines you installed as part of Assignment #0 (which I have taken to calling `Senjhalla`). This is our "reference platform". This same environment will also be used by the course instructor and the rest of the teaching team when evaluating submitted work from students.

All test files and sample code for this assignment is available on the UVic Unix server in `/home/zastre/seng265/a1` and you must use `scp` in a manner similar to what happens in labs in order to copy these files into your `Senjhalla`.

Any programming done outside of `Senjhalla` might not work during evaluation.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Zastre).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted programs.)

Questions

If you have any questions about the assignment, please post them to our course's channel at `rocket.csc.uvic.ca` (i.e., RocketChat). By using this Slack-like environment, I can not only provide answers to queries but also give each of you a chance to write answers to questions from fellow students.

Objectives of this assignment

- Learn to use basic features of the Python language.
- Use the Python 3 programming language – more precisely Python 3.7.5 as this is installed in your Senjhalla – in order to write a less resource-restricted implementation of `concord` **but without using regular expressions or user-defined classes**.
- Use `git` to manage changes in your source code and annotate the evolution of your solution with messages provided during commits.
- Test your code against the 20 provided test cases (10 from assignment #1, 10 more added for assignment #2).

`concord2.py`: Returning to the problem

In this assignment we will re-visit the *concordance* problem given in assignment #1. However, we will eliminate several restrictions from the first assignment but will retain some others.

- There are no longer maximum values for # of input lines, # of exception words, # of keywords, lengths of words, or lengths of input lines.

- Input-file words may now be in upper and lower case. However, the words "`apple`" and "`Apple`" would both appear with the line for keyword "`APPLE`". Exception-file words will still be lower-case (and store alphabetically in the file).

- The only punctuation in the input-files, besides single spaces between words, will be the hyphen ("-") that may appear between words. For example, "`twenty-one`" would be one whole word, but words with two or more hyphens such as "`twenty--one`" or words ending in a hyphen and followed by another word such as would occur with "`twenty- or`" **would not be considered a valid word** (i.e., such words with multiple or trailing hyphens **will not appear** within input files or exception-word files).

All other behavior is as described in the first assignment. Running the program will also be similar to assignment #1. For example, to run code for test 17, either one of the following command lines is possible:

```
$ ./concord2.py -e latin-2.txt in17.txt

$ ./concord2.py in17.txt -e latin-2.txt
```

As with the first assignment, all output is to `stdout`. You must test the output of your program in the same manner as with assignment #1 (i.e., using `diff`).

However, I will place four different kinds of constraints on your Python program.

1. For this assignment **you are not to use regular expressions**. We will instead use these in assignment #4 in order to write a more powerful version of the program that processes files with words involving punctuation.
2. You must **not write your own classes** as this will be work for assignment #4.
3. You must **not use global variables** (with the only exception being the use of `UPPER_CASED_NAMED_CONSTANTS` as is the normal idiom in Python programming).
4. You must **make good use of functional decomposition**. Phrased another way, your submitted work **must not** contain one or two giant functions where all of your program logic is concentrated.

Exercises for this assignment

1. Within your `git` repo ensure there is an `a2/` subdirectory. (For convenience of testing, I have placed all necessary test files – that is, those from the first assignment and those new to this one – in my `/home/zastre/seng265/a2` directory.) Your "`concord2.py`" script must be located in your `a2/` directory.

2. Write your program. Amongst other tasks you will need to:
   - read text input from a file, line by line
   - write output to the terminal
   - extract substrings from lines produced when reading a file
   - create and use lists in a non-trivial way

3. **Keep all of your code in one file for this assignment**. In assignment #4 we will use the multiple-module and class features of Python. Please ensure you also respect all of the other constraints described earlier in this document.

4. Use the test files and listed test cases to guide your implementation efforts (i.e., information in `/home/zastre/seng265/a2/TESTS.md`). Refrain from writing the program all at once, and budget time to anticipate when "things go wrong".

5. For this assignment you can assume all test inputs will be well-formed (*i.e.*, our teaching team will not test your submission for its handling of error-formed input or for malformed command-line arguments). Assignments 3 and 4 may specify error-handling as part of their requirements.

What you must submit

- A single Python source file named `concord2.py` within your `git` repository containing a solution to assignment #2.

- No regular-expressions, global variables, or user-defined classes are to be used for assignment #2.

Evaluation

Our grading scheme is relatively simple.

- "A" grade: A submission completing the requirements of the assignment which is well-structured and very clearly written. All tests pass and therefore no extraneous output is produced.

- "B" grade: A submission completing the requirements of the assignment. `concord2.py` runs without any problems; that is, all tests pass and therefore no extraneous output is produced.

- "C" grade: A submission completing most of the requirements of the assignment. `concord2.py` runs with some problems.

- "D" grade: A serious attempt at completing requirements for the assignment. `concord2.py` runs with quite a few problems; some non-trivial tests pass.

- "F" grade: Either no submission given, or submission represents very little work, or no tests pass.