Software Engineering 265
Software Development Methods
Summer 2021

Assignment 4

Due: Tuesday, August 3, 11:55 pm by submission via git
(no late submissions accepted)

~~Note: This assignment is not required for completion of the course.~~

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 starter code for this assignment is available on the UVic Unix server in `/home/zastre/seng265/a4` and you must use `scp` in a manner similar to what happens in labs in order to copy these files into your `Senjhalla`. The tests used for assignment #2 (i.e. those in `/home/zastre/seng265/a2`) will be used for assignment three with the exception of test 18 (i.e. this test will not be used). Five additional tests are provided (21 through to 25) for this fourth and last assignment.

Any programming done outside of `Senjhalla` might not work during evaluation. (I know many of you are using VS Code, but you should try to get comfortable with `vim` as you never know when you'll be confronted with an environment without an IDE.)

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 about 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.)

Objectives of this assignment

- Use the Python 3 programming language to write a less resource-restricted implementation of CONCORDANCE, this time using *user-defined classes* and *regular expressions*.
- 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 provided test cases .

`concord4.py`: Place concordance code into a class

In this assignment we will visit, for the last time, the CONCORDANCE problem. As in assignment #3, the following is now true:

- 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).

However, some punctuation may now appear in the input-files. This is described in more detail later in this assignment description.

Running the program will differ slightly from previous assignments. You are provided with a file named `tester4.py` designed to accept command-line arguments from the shell and call your implementation of the `Concordance` class in `concord4.py`.

```
$ ./tester4.py -e latin-2.txt in17.txt

$ ./tester4.py in17.txt -e latin-2.txt
```

There are a few more important differences in what you are asked to do:

- The constructor for `Concordance` must accept two string parameters – the name of the input file, and the name of the exception-words file. You must write this constructor.
- The method named `full_text` must return a list of strings corresponding to the `Concordance` output required. You must write this method.
- You must also write any other methods needed within the class; these methods should be all "private" (or, rather, having a name starting with an underscore character as there is no true "private" in Python).

Please look at the `main` function of `tester4.py` to learn better how the script depends upon the constructor signature and the method `full_text`. All of your Python 3 code must appear within `concord4.py` and within the class `Concordance`.

With respect to punctuation:

- As with the previous two assignments, words may contain dashes, and the words may contain upper- and lower-case letters yet still be considered the same word (e.g., "Who" and "who").
- Words may now end with `'s` and `'t` (as in "Thor's hammer" or in "Thor can't come to the door right now"). The `'s` and `'t` are to be considered part of the word.
- Double-quotes may now appear before and after words, as may parentheses. ***These are not to be considered part of the word.***
- Other punctuation such as commas (","), periods ("."), colons (":"), semi-colons (";"), exclamation marks ("!") and question marks ("?") may appear at the end of a word. ***These are not to be considered part of the word.***

Exercises for this assignment

1. Within your `git` repo ensure there exists an `a4/` 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/a4` directory.) Your "`concord4.py`" file must be located in your `a4/` 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
   - write a user-defined class
   - use regular expressions

3. Use the test files and listed test cases to guide your implementation efforts (i.e., information in `/home/zastre/seng265/a4/TESTS.md`). Refrain from writing the program all at once, and budget time to anticipate when "things go wrong".

4. 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).

What you must submit

- A single Python source file named `concord4.py` within your `git` repository containing a solution to assignment #4.

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. The code written in `concord4.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 The code written in `concord4.py` runs with some problems.
- "D" grade: A serious attempt at completing requirements for the assignment. The code written in `concord4.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.