Software Engineering 265
Software Development Methods
Fall 2022

Assignment 4

Due: Tuesday, December 6, 11:55 pm by "git push"
(Late submissions **not** accepted)

Programming environment

For this assignment please ensure your work executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself a few days before the due date to iron out any bugs in the C program you have uploaded to a lab workstation. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

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 the course instructor as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found online and used in your solution must be cited in comments just before where such code has been used.

Objectives of this assignment

- Use the Python 3 programming language to write a less resource-restricted implementation of the 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 .

Place the concordance-generation code in a Python class: `concord4.py`

In this assignment we will visit, for the last time, the concordance problem[1]. 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 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 words will still be lower-case (and stored alphabetically in the file).

Running the program will differ slightly from previous assignments. You are provided several files:

- `concord4.py` will contain your work (i.e. code for the `concord` class), and is never used directly at the command line.
- `driver-original.py` which is to be used on the command line in a way similar to previous assignments
- `driver-new.py` which accepts two file arguments, one for the name of the input file, one for the name of the output file

For example, to run the tenth test and compare it with expected output, and assuming all test files are in the current directory, you have two possibilities. One uses the original style of input and output:

```
$ cat in10.txt | ./driver-original.py | diff – out10.txt
```

while the other creates an output file which you then must compare in a separate command:

```
$ ./driver-new.py --in in10.txt  --out _out10.txt
$ diff out10.txt _out10.txt
```

Make sure when using `driver-new.py` that you do not overwrite the correct test-output file with your own output!

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

- The `concord4.py` contains the class named `concord`.

[1] I appreciate that some of you are heartily sick of this problem, yet please understand that you can focus on learning new language features and semantics and not be distracted by learning how to a new non-trivial problem when you have, like, five assignments from other courses due near the same date.

- The constructor for `concord` takes two string parameters – the input file name, and the output filename. However, if the input filename is `None`, then input is to be obtained from `stdin`; if the output filename ~~in~~ is `None`, output is not to be generated directly to the console. When testing your solution, however, we will only ever give two filenames or two Nones.
- The method named `full_concordance` must return a list of strings corresponding to the output lines required (i.e. as described for Assignments 2 and 3). **You must write this method, amongst others of your own choosing.** If there are to be no lines in the concordance, then an empty list must be returned from the method.
- You must also place any other methods you need to write within the `concord` class; these methods must be all "private" (or, rather, having a name starting with ~~an underscore~~ a double-underscore character as there is no true "private" in Python).

Please look at the `main` function of `driver-original.py` and `driver-original.py` to learn better how the script depends upon the constructor's signature and the method `full_concordance`. All of your Python 3 code must appear within `concord4.py` and within the class `concord`.

You are not permitted to add source-code files to your submission without first obtaining express written permission from the course instructor.

A few more observations:

- You will notice that some lines in `driver-original.py` and `driver-new.py` disable and re-enable access to `stdout` and `stderr`.
- The reason for the item above is that the concordance must be generated as a list of strings returned by `full_concordance`. This list can then later be output as newline-separated output – either to `stdout` as can be seen in `driver-original.py`, or within one of your own concord class methods as would be needed if output is to be stored in a file (i.e. as invoked by `driver-new.py`). For example, when the concord constructor is called with the input and output associated with `None`, this means that input comes from `stdin`, and the results from `full_concordance` are returned from the class instance but printed outside the class instance (i.e. the results are not output to `stdout` from within the class).
- **These two driver programs as given to you will be used when evaluating your work. That is, you cannot solve the problem by simply writing output directly to stdout.** ***However***, in order to facilitate debugging, you can comment out the lines that disable/re-enable access to `stdout/stderr`.
- You are to experiment with regular expressions.
- Global variables are forbidden.

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 `stdin`, line by line
   - read text input from a file, line by line
   - write output to the terminal
   - write output to a file
   - 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., tests in `/home/zastre/seng265/a2`). 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 `a4/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.