Software Engineering 265
Software Development Methods
Spring 2018

Assignment 2

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

Programming environment

For this assignment you must ensure your work executes correctly on *linux.csc.uvic.ca*. You can use *git push* and *git pull* to move files back and forth between your account on the UVic CSC filesystem and your computer. Bugs in the kind of programming done this term tend to be platform specific, and so something that works perfectly on your own machine may end up crashing on *linux.csc*. The fault is very rarely that of *linux.csc*'s configuration. "It worked on my machine!" will be treated as the equivalent of "The dog ate my homework!"

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

- Understand a problem description, along with the role played by sample input and output for providing such a description.
- Use the Python 3 to write the next phase of a "keyword-in-context" utility. The program will be named (unsurprisingly) *kwic2.py*.
- Use *git* to track changes in your source code and annotate the evolution of your solution with "messages" provided during commits.
- Test your code against the 15 provided test cases.

This assignment: *kwic2.py*

Assignment #1 introduced you to the concept of "keyword in context" (KWIC). In this assignment we will introduce some output formatting that should help make output more readable.

Consider the following input – similar to that from in07.txt in assignment #, but with significantly different output.

```
2
::
of
and
the
too
on
who
to
that
::
that fortune
sense and sensibility
life of robert browning
the man who knew too much
legend of montrose
visit to iceland
orthodoxy
the mountains
on the track
ward of king canute
```

```
               life of robert BROWNING
                 ward of king CANUTE
                         that FORTUNE
                    visit to ICELAND
                     ward of KING canute
                  the man who KNEW too much
                              LEGEND of montrose
                              LIFE of robert browning
                           the MAN who knew too much
                    legend of MONTROSE
                           the MOUNTAINS
            man who knew too MUCH
                              ORTHODOXY
                      life of ROBERT browning
                              SENSE and sensibility
                    sense and SENSIBILITY
                       on the TRACK
                              VISIT to iceland
                              WARD of king canute
```

The biggest difference is that *index words are now left-aligned*, and some of the text from the surrounding words also appear. However, there are a few more differences, too – one of which is clear from the example above, another related one which can be seen in tests sets 09 and higher.

- Index words now begin at the 30th column of the output
- Words to the left of the index word may appear but only if they do not go *further left than column 10*.
- Words to the right of the index word may appear but only if they do not go *further right than column 60*.

To illustrate further, consider one of the lines taken from in09.txt:

```
verzierten sofa dessen polster hellgelb ueberzogen waren warf einen
```

And below are extracted from out09.txt the lines generated for this input line. (The numbers are the top are to indicate the column numbering. These must not appear in the output.)

```
0000000001111111111222222222233333333334444444444555555555566666666667
1234567890123456789012345678901234567890123456789012345678901234567890
         sofa dessen polster HELLGELB ueberzogen waren warf
                 sofa dessen POLSTER hellgelb ueberzogen
                  verzierten SOFA dessen polster hellgelb
            polster hellgelb UEBERZOGEN waren warf einen
                             VERZIERTEN sofa dessen polster
         hellgelb ueberzogen WAREN warf einen
            ueberzogen waren WARF einen
```

In the first line of output above, notice that "verzierten" does not appear even though it is in the input line. (i.e., it would go too far left in the output). Also in the first line of output above, notice that the word "einen" is also missing (i.e., it would go too far to the right). In the second line of output, "verzierten" is missing from the left, and "waren warf" are missing from the right. And so on and so forth.

Your implementation of *kwic2.py* must produce this kind of formatting of the indexed-output lines.

And in addition:

- Lines-for-input may now contain words in different cases. However, for the purposes of indexing you can ignore case (i.e., when comparing words, first convert them into all upper-case or all lower-case). If the word in a line is not an index word, however, it must appear in its capitalization (or lack thereof).
- There is no limit to the number of exclusion words, input lines, or index words.
- Lines-for-indexing can contain blank lines. These blank lines may be ignored.

Running the program

Your program will be run from the Unix command line. Input is expected from *stdin*, and output is expected at *stdout*. ***You must not provide filenames to the program, nor hardcode input and output file names.***

For example, assuming your current directory contains your script *kwic2.py* and a *tests/* directory containing the assignment's test files is also in the current directory, then the command to transform the page two's input into required output will be:

```
% cat tests/in07.txt | ./kwic2.py
```

In the command above, output will appear on the console. You may want to capture the output to a temporary file, and then compare it with the expected output:

```
% cat tests/in07.txt | ./kwic2.py > temp.txt
% diff tests/out07.txt temp.txt
```

The same thing (i.e., producing output and comparing it with the expected output) can be combined into a one-liner:

```
% cat tests/in07.txt | ./kwic2.py | diff tests/out07.txt -
```

The ending hyphen/dash informs *diff* that it must compare the contents of *tests/out07.txt* with the input piped into *diff's* stdin.

Exercises for this assignment

1. Write your program *kwic2.py* program in the *a2* directory within your git repository.
   - Read *stdin* line by line and processing each line appropriately.
   - Read and store exclusion words.
   - Read and store lines-for-indexing
   - Format output lines according to the rules described earlier in this document.

2. **Keep all of your code in one file for this assignment. In later assignments we may use the Python module mechanism. Do not use Python classes for this assignment. You must use Python 3 (i.e., what is available when you run the "python3" command on linux.csc.uvic.ca).**

3. Use the test files in */home/zastre/seng265/tests* (i.e., on the CSC filesystem) guide your implementation effort. The tests here are different from assignment 1. (The assignment #1 test files can be found in */home/zastre/seng265/tests/a1*.) Start with simple cases (for example, the one described in this writeup). In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once,

and budget time to anticipate for when "things go wrong". There are fifteen pairs of test files.

4. For this assignment you can assume all test inputs will be well-formed (i.e., the teaching team will not test your submission for handling of errors in the input). Later assignments will specify error-handling as part of the assignment.

5. Use *git add* and *git commit* appropriately. While you are not required to use *git push* during the development of your program, you **must** use *git push* in order to submit your assignment.

What you must submit

- A *single* Python script named *kwic2.py* within your git repository containing a solution to Assignment #2. Ensure your work is **committed** to your local repository **and pushed** to the remote **before the due date/time**. (You may keep extra files used during development within the repository.)

Simplifying Assumptions for Assignment #2

a) All input files will be KWIC version 2 files.
b) Indexed words will appear at most once in any line-for-indexing.
c) Decimal numbers are treated as indexed words; use lexicographic (i.e., alphabetical) ordering for determining where line-for-indexing containing such a word should appear in the output.
d) If an indexed word appears in more than one line-for-indexing, print the lines in the order they appear in the input.

Evaluation

Our grading scheme is relatively simple and is out of 10 points. We may award grades within the categories below.

- 10/10: *kwic2.py* runs without problems and compiles without warnings. All fifteen tests pass. The program is clearly written and uses functions appropriately (i.e., is well structured).
- 8/10: *kwic2.py* runs without any problems and compiles with any warnings. All fifteen tests pass.
- 7/10: A submission completing most of the requirements of the assignment. *kwic2.py* runs with some problems; some tests do not pass.

- 5/10: A serious attempt at completing requirements for the assignment. *kwic2.py* runs with quite a few problems; most tests do not pass, although some tests do pass.
- 4/10 or lower: No submission given, submission represents very little work, or no tests pass.

Through the term we will also request each student demonstrate at least one assignment to the teaching team. More about this process will be described closer to the due date of this assignment.