Software Engineering 265
Software Development Methods
Summer 2014

Assignment 3

Due: Tuesday, November 15th, 7:00 pm via a push to your git remote repository.
(The due date has been chosen to account for the November reading break.)

No late submissions accepted

Programming environment

For this assignment you must ensure your work executes correctly on the Linux machines in ELW B215 (i.e., these have Python 3 installed). You are free to do some of your programming on your own computers; if you do this, give yourself a few days before the due date to iron out any bugs in the Python code as it runs in the lab.

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 may be used to examine submitted programs.)

Objectives of this assignment

- Continue to learn features of the Python 3 language.
- Use some of the object-oriented features of Python 3.
- Use some of the error-handling features of Python 3.
- Use some of the regular-expression functionality in Python 3.
- 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 twenty provided test cases.
- Be prepared to justify your script's handling of errors during the assignment demonstration.

This assignment: *textdriver.py* and *textformatter.py*

For this assignment you will wrap a solution to the formatting problem into a class. The class constructor will accept a *filename* and a *list of strings*. The formatted output will be available from the class instance via the *get_lines()* method.

- The class must be named *Formatter* and appear in a file named *textformatter.py*.

- The script that uses this class is in a file named *textdriver.py*. The file has been provided and **you must not modify it**.

- If a non-blank filename is provided to the *textdriver.py* script, then that file will be opened and its contents formatted.

- If no argument (i.e., filename) is provided to the *textdriver.py* script, then the contents of *stdin* will be formatted.

- If an instance of *Formatter* is used without being called from *textdriver.py*, and no filename is provided to the constructor (i.e., *None* is given as the filename argument), then the lines in the list provided as a parameter will be formatted. Put differently, the *Formatter* constructor will accept (besides "self") two parameters: a string and a list of strings, with the latter having the value of *None* when the constructor is called from *textdriver.py*. Have a look at the code in *formatlines.py* in the */home/zastre/seng265/assign3* directory to find out how a list is passed to the constructor. (*textformatter.py* must therefore contain an implementation of *Formatter* that provides interface as is seen to be used in *textdriver.py*.)

- **Global/module scope variables are not permitted in your solution.** However, classes can and should have instance variables. Class variables may only be used for values that are constant (i.e., more than one *Formatter* object should be permitted to exist in some program).

- The formatting specifications from the second assignment are to be used for this assignment.

- The script that uses this class will be in a file named *textdriver.py* (as mentioned above).

- You must provide some error handling in *Formatter*. Your task is to enumerate the things that could go wrong when faced with such a formatting task, and to either provide or suggest the code for handling each error item. The provided test files are not meant to contain errors, so you may create up to five of your own test files containing errors. These must be in a directory named *assign3/tests* with names starting with the "e" (e.g., *e_in01.txt* and

e_out01.txt, etc.). Document your chosen error cases—and your code's handling of them—in a Markdown file named *error_handling.md*, and place this file in your *assign3* directory. (For more information on the Markdown file format, visit *https://en.wikipedia.org/wiki/Markdown*.)

- The solution must be written in Python 3 and work correctly on the workstations in the ELW B215 laboratory.

With *textdriver.py* using the code within your completed *textformatter.py*, the input would be transformed into the output (here redirected to a file) via one of the two following UNIX commands:

```
$ ./textdriver.py /home/zastre/seng265/assign2/tests/in11.txt > \
     ./myout11.txt

% cat /home/zastre/seng265/assign2/tests/in11.txt \
      |./textdriver.py > ./myout11.txt
```

where the file *myout11.txt* would be written in your current directory.

Exercises for this assignment

1. Within your Git repository project create an *assign3* subdirectory. Use the test files in */home/zastre/seng265/assign2/tests.* The *textdriver.py* and your *textformatter.py* script files must appear here. Ensure that when you create the *assign3* directory, you immediately perform a *git add* and *git commit*.

2. Reasonable run-time performance of your script is expected. None of the test cases should take longer than 15 seconds to complete processing on a machine in ELW B215.

What you must submit

- Python 3 files named *textdriver.py* (i.e., just a copy of what has been provided to you) and *textformatter.py* (i.e., your assignment solution) your git repository.

- A text file named *error_handling.md* which enumerates the errors that are addressed by your submission.

- Test-case files for your error-handling implementation, with the files placed into *assign3/tests*.

Evaluation

Students may be selected to demonstrate their work to a member of the course's teaching team. Student demo lists will be posted onto conneX shortly before the the due date/time. Those selected for a demo must sign up for a demo slot; each demo will require about 10 minutes.

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative. *textdriver.py* and *textformatter.py* run together without any problems. All tests pass. Errors have been enumerated and are either suitably handled or a sensible response strategy has been suggested. The program is clearly written and uses functions appropriately (i.e., is well structured). Any extra functionality or features should be in a file named *textformatter_extra.py* (i.e., do not add them into your submitted *textdriver.py* and *textformatter.py*). If you choose to use more files, their purpose must be described in a file named *extra.md*.
- "B" grade: A submission completing the requirements of the assignment. *textdriver.py* and *textformatter.py* run together without any problems. All tests pass. Errors have been enumerated and are either suitably handled or a sensible response strategy has been suggested. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. *textdriver.py* and *textformatter.py* run together with some problems. Some important error cases have been overlooked.
- "D" grade: A serious attempt at completing requirements for the assignment. *textdriver.py* and *textformatter.py* run together with quite a few problems.
- "F" grade: Either no submission given, or submission represents very little work.