Software Engineering 265
Software Development Methods
Fall 2016

Assignment 2

Due: Tuesday, October 25th, 7:00 pm via push to Git
(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 3.4 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 script you have pushed to your remote repository.

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

- Learn to use basic features of the Python 3 language. This is available as `python3` from the command line, or `/opt/bin/python3` on the ELW B215 machines. Alternatively you can use `#!/usr/bin/env python3` in your bang path.
- Use the Python programming language to write a less resource-restricted implementation of "`format265`" (but without using 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 twenty provided test cases.

This assignment: "format265.py"

You are to write a Python version of the program you prepared for assignment #1. All of the formatting capabilities of that C program are to be implemented in this new program. Like the C program, it must run as a command in the shell. (Beware, though: You do not want to attempt porting you C solution into Python!) There are several additional abilities your Python script must have:

- **If a filename is provided to the script, then that file's contents are formatted. If no filename is provided, then the contents of stdin are formatted. In either case, the formatted output is sent to stdout.**

- The ".LW", ".LM" and ".FT" commands may appear at any line of the input file. As in assignment #1, such commands are valid only if they appear at the start of a line.

- The margin may now be specified relative to the value of the current margin position. For example, if the margin currently is 10, then when ".LM +5" is encountered the margin would be set to 15. If the margin currently is 20, then ".LM -7" would set the margin to 13, and a further ".LM -7" would set the margin to 6.

- Valid margins must be (a) greater than or equal to 0, and (b) less than or equal to the page width minus 20. If a margin would be negative after a ".LM" command, then the margin is set to zero. If a margin would be greater than then page width minus 20, then the margin is set to page width minus 20.

- For this assignment, a ".LM" will never be immediately followed (i.e., on the next line) by another ".LM" command.

- The linespacing command (.LS) will never have values greater than 2.

- There is no limit to the number of input lines. You may assume that each input line is no longer than 132 characters.

With your completed "format265.py" script, the input would be transformed into the output (here redirected to a file) and then checked:

```
% ./format265.py /home/zastre/seng265/assign2/tests/in11.txt > ./myout11.txt

% diff /home/zastre/seng265/assign2/tests/out11.txt ./myout11.txt
```

where the file "`myout11.txt`" would be found in your current directory.

Exercises for this assignment

1. Within your Git project create an "`assign2`" local repository. Use the test files in `/home/zastre/seng265/assign2/tests`. (Files `in01.txt` through to `in10.txt` are the same as those used for the first assignment.) Your `format265.py` script must appear in this `assign2`. Ensure the subdirectory and script file are tracked by Git.

2. Write your program. Amongst other tasks you will need to:
   - read text input from a file, line by line
   - read text input from stdin, 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 array
   - use the "diff" Unix command to test the output of your program

3. Keep all of your code in one file for this assignment. We will explore Python's object-oriented features and module-authoring mechanism in the next assignment (i.e., Assignment #3).

4. Use the test files to guide your implementation effort. Start with simple cases (such as those given in this writeup). 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 assistant will not test your submission for handling of input or for arguments containing errors). Later assignments will specify error-handling as part of the assignment.

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

What you must submit

- A single Python script file named `format265.py` containing a solution to Assignment #2 within your Git repository in a directory named `assign2`.

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 several hours the due date. Those selected for a demo must sign up for a demo slot; each demo will require

about 10 minutes. Details of the sign-up process will be posted on conneX as an announcement.j

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative. "format265" runs without any problems. The program is clearly written and uses functions appropriately (i.e., is well structured). Evaluation of your extra work would be easier if you could include a second version of your assignment with the extra features (naming the file `format265_extra.py`).
- "B" grade: A submission completing the requirements of the assignment. "format265" runs without any problems. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. "format265" runs with some problems.
- "D" grade: A serious attempt at completing requirements for the assignment. "format265" runs with quite a few problems.
- "F" grade: Either no submission given, or submission represents very little work.