Software Engineering 265
Software Development Methods
Spring 2018

Assignment 2

Due: Thursday, June 28th, 11:55 pm by "git push"
(no late submissions accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the machines in ECS B238 (i.e., these have Python 3.6 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 3.6 language. If you run `setSENG265` when first logging in, Python 3.6 is available from the command line as `python` although the actual path is a bit of a mouthful; type `which python` to discover what that pathname is! I recommend you instead use `#!/usr/bin/env python` in your script's bang path.
- Use the Python programming language to write a less resource-restricted implementation of "`uvroff.py`" (but **without using regular expressions** and **without creating a new Python class**).
- 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 (i.e., ten original tests from assignment #1 plus ten new tests) using a computer in ELW B238.

This assignment: "uvroff.py"

You are to write a Python version of `uvroff`. You solved a version of the problem in C, and you will now do so in Python – but be careful as you should not try to port your C code line-by-line into Python. That languages are very different.

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. 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 "`uvroff`.py" script, the input would be transformed into the output (here redirected to a file) and then checked:

```
% ./uvroff.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 ensure there is an "`a2`" local directory. 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 `uvroff.py` script must appear in this `a2/` directory. Ensure the subdirectory and script file are tracked by git (i.e., using `add` and `commit`).

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 may specify error-handling as part of the assignment.

6. Reasonable run-time performance of your script is expected. No test case should take longer than 15 seconds to complete on a machine in ECS B238.

What you must submit

- A single Python script file named `uvroff.py` containing a solution to Assignment #2 within your git repository in the directory named `a2/`.

Evaluation

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

- 10/10: `uvroff.py` runs without problems without warnings. All twenty tests pass. The program is clearly written and uses functions appropriately (i.e., is well structured).
- 8/10: `uvroff.py` runs without any problems. All twenty tests pass.
- 7/10: A submission completing most of the requirements of the assignment. `uvroff.py` runs with some problems; some tests do not pass.
- 5/10: A serious attempt at completing requirements for the assignment. `uvroff.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.