Software Engineering 265
Software Development Methods
Fall 2018

Assignment 2

Due: Sunday, November 4, 11:55 pm by submission via
Gitlab (no late submissions accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the Linux machines in ELW B238 (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 script you have uploaded to the BSEng machines.

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.** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact your course instructor as soon as possible. (**Code-similarity analysis tools would be used to examine submitted programs.**)

Objectives of this assignment

- Learn to use basic features of the Python 3 language. (To enable this in your environment when logged into the lab, type in the following command at the start of your session at the shell prompt: `scl enable python33 bash`
- Use the Python programming language to write a less resource-restricted implementation of "sengfmt" (but **without using regular expressions**).
- Use gitlab 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: “sengfmt.py”

You are to write a Python version of the C program prepared for assignment #1. Some of the formatting capabilities of that C program (e.g. the ?mrgn and ?fmt commands) are to be implemented in this new program. Like the C program, it must run as a command in the shell. However, 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 “?mrgn” and “?fmt” commands **may appear at any line** of the input file. As in assignment #1, such commands are valid only if that appear at the start of a line, for commands appear in the middle of a line would be treated as text strings.

- 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 “?mrgn +5” is encountered the margin would be set to 15. If the margin currently is 20, then “?mrgn -7” would set the margin to 13, and a further “?mrgn -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 **maxwidth minus 20**. If a margin would be negative after a “?mrgn” command, then the margin is set to zero. If a margin would be greater than maxwidth minus 20, then the margin is set to maxwidth minus 20.

- If the “?maxwidth” is undefined in the text, the line would not be altered except for adding a margin when margin is greater than zero. Only the margin may apply and there will be no full justification in the format.

- If “?fmt” is turned off then ALL the formatting will not apply to the text until “?fmt” is turned on again.

- For this assignment, a “?mrgn” will never be immediately followed (i.e., on the next line) by another “?mrgn” command.

- There is no limit to the number of input lines. You may assume that each **input line** is no longer than 150 characters.

- The “?width” command will **not** appear in any of the test input files. However, a new “?maxwidth” command followed by a number will replace the feature of “?width” command, while a “fully justified” format is provided. It could also be specified relative to the value of the current page width just like “?mrgn +5”.

Here are the two new commands you need to implement in this assignment:

- ?maxwidth width: Each line following the command will be formatted such that each line has exactly maxwidth characters and is fully (left and right) justified. The original whitespace in the input text need not necessarily be preserved (i.e., **single or multiple spaces would be used to separate words in the output**). You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxwidth characters.

  Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

  For the last line of a paragraph, in real world it should be left justified, however, to simplify this assignment so that you don't need to check if it is the last line of a paragraph, we continue formatting the line to be fully justified.

  If this command does not appear in the input file, then the input text is not transformed with a limited width in the output. You can assume the value range of maxwidth is inclusively between 20 and 150.

Example 1:

```
Input:

tokens = ["This", "is", "an", "example", "of", "text",
"justification."]
maxwidth = 20
mrgn = 0

Output:
[
   "This  is  an example",
   "of              text",
   "justification.      "
]
```

Example 2:

```
Input:

tokens = ["What","must","be","acknowledgment","should","be"]
maxwidth = 20
mrgn = 0

Output:
[
  "What     must     be",
  "acknowledgment      ",
  "should            be"
]
```

Note that the last line is "shall   be" instead of "shall be   ", because the last line must be fully-justified instead of left-justified. The second line is left-justified because it contains only one word.

Example 3:

```
Input:

tokens = ["What","must","be","acknowledgment","should","be"]
maxwidth = 24
mrgn = 4

Output:
[
  "    What     must     be",
  "    acknowledgment      ",
  "    should            be"
]
```

Note that there could a combination of both margin and maxwidth, and the justification is actually starting from the position “maxwidth-mrgn”. The value of mrgn must not be greater than maxwidth minus 20.

- ?cap [off | on]: This is used to turn capitalization on and off. If the command appears with “off”, then all text below the command up to the next “?cap” command is output without being capitalized. If the command appears with “on”, then all text below the command up to the next “?cap” command is output with all capital letter. (It may be that there is no “?cap” command following, in which case the choice of “on” or “off” is used for the remaining input text.) The digits in the text are not affected by this command.

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

```
% python3 ./sengfmt.py /home/seng265/A2/in11.txt > ./myout11.txt

% diff /home/seng265/A2/out11.txt ./myout11.txt
```

where the file “myout11.txt” would be found in your current directory.

Exercises for this assignment

1. Within your Gitlab project create an “A2” subdirectory (which you should have already). Your “sengfmt.py” script must appear here. Ensure the subdirectory and script file are added to version control.

2. Write your program. Amongst other tasks you will need to:
   - read text input from a file, 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. In later assignments we will use the multiple-module features of Python.

4. Use the test files to guide your implementation effort. Start with simple cases. 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 B238.

What you must submit

- A single Python script file named “sengfmt.py” within your Gitlab repository containing a solution to Assignment #2.

Evaluation

Our grading scheme is relatively simple. Continued next page.

Requirement	**Marks**
File sengfmt.y runs without errors and can generate output	2
Code passes the first set of test cases (1 to 10)	5
Code passes the second set of test cases (11 to 15)	5
Code passes the third set of test cases (16 to 18)	6
Code passes the fourth set of test case (19 to 20)	2
Total	**20**