Software Engineering 265
Software Development Methods
Fall 2018

Assignment 3

~~Due: Sunday, November 25th, 11:55 pm by submission via Gitlab~~

Due: Sunday, December 2nd, 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 lab 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 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. (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: `setSENG265`
- 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 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 ten provided test cases.
- Be prepared to justify your script's handling of errors for this assignment.

This assignment: "sengfmt2.py" & "formatter.py" for object–oriented features

For this assignment you will be **wrapping a solution** to the formatting problem into a class. The class constructor will be given either a filename or a list as a parameter. 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 "formatter.py" which contains most of the methods you previously developed in "sengfmt.py".

- The script that uses this class will be in a new file named "sengfmt2.py". (similar to driver.py but can handle fileinput, rather than a static long string).

    - If a non–blank filename is provided to the sengfmt2.py script, then that file will be opened and its contents formatted.

    - If no name is provided to the sengfmt2.py script, then the contents of stdin will be formatted.

- If the formatter.py object is used without being called from sengfmt2.py, and no filename is provided to the constructor (i.e., None is passed as filename argument), then the lines in the list provided as a parameter will be formatted. Put differently, the Formatter constructor will accept (besides "self") a string and a list as parameters. **Have a look at the code in driver.py in the a3_files directory to find out how a list is passed to the constructor in sengfmt2.py. (driver.py must therefore work with your code!)**

- The formatting specifications from the second assignment are, for the most part, to be used for this assignment. However, there are two small changes: two "?mrgn" commands may appear right after each other in an input file; and a file may have more than one "?maxwidth" command.

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

- ?replace pattern1 pattern2: Each line following the command will be formatted such that any matched sub-strings with the first argument pattern1 will be replaced by the second argument pattern2. You need to use the "re" module in Python 3 and complete the replacement before applying any other formats. You can consider both pattern1 and pattern2 can be generalized to formal regular expressions and they are separated by a single white space.

Example1:

```
Input:

[
  "?maxwidth 24",
  "?mrgn 4",
  "?replace What You",
  "What must acknowledgment shall be"
]

Output:
[
  "    You             must",
  "    acknowledgment      ",
  "    shall             be"
]
```

Example2:

```
Input:

[
  "?maxwidth 24",
  "?mrgn 4",
  "?replace acknowledgment know",
  "What must acknowledgment shall be"
]

Output:
[
  "    What    must    know",
  "    shall             be"
]
```

- ?monthabbr [off/on]: Each line following the command will be formatted such that any recognizable date in the format of "mm/dd/yyyy" or "mm-dd-yyyy" or "mm.dd.yyyy" (not a regular expression, you need to create one) will be replaced with the new date format "MMM. dd, yyyy" in which "MMM" stands for the month abbreviation. You can import and use the calendar module to convert the month number to month abbreviation. You can only focus on the above three variants and ignore other date formats like "yyyy-mm-dd". You can show your concerns in the error handling file.

```
>>> import calendar
>>> month_number = 1
>>> print(calendar.month_abbr[month_number])
'Jan'
```

Example3:

```
Input:

[
  "?maxwidth 24",
  "?mrgn 4",
  "?monthabbr on",
  "I dropped the course on 12/07/2018"
]

Output:
[
  "    I dropped the course",
  "    on   Dec.  07,  2018"
]
```

- Note that if "?fmt" command is turned off, then all the formatting will NOT apply.

You must now provide some error handling.

- Your task is to enumerate the things that could go wrong when faced with such a formatting task in a plain text file named "error_handling.txt", and to either provide or suggest the code for handling each error item.

- The solution must be written in Python 3 and work correctly on the workstations in the ELW B238 laboratory.

With your completed "sengfmt2.py" script, the input would be transformed into the output (here redirected to a file) via one of the two following UNIX commands:

```
% python3 ./sengfmt2.py /home/seng265/assign3/in11.txt > ./myout11.txt

% cat ~/seng265/assign3/in11.txt | python3 ./sengfmt2.py > ./myout11.txt
```

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

Exercises for this assignment

1. Within your Gitlab project create an "A3" subdirectory. Use the test files in a3_files.zip. (Files in01.txt through to in20.txt are based on those from the second assignment, but letters with diacritics have been transliterated into two separate characters.) Your "sengfmt2.py" and "formatter.py" script files must appear here. Ensure the subdirectory and files are added to Gitlab version control.

2. 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

- Python 3 files named "sengfmt2.py" and "formatter.py" within your subversion repository (i.e., containing a solution to Assignment #3).

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

Evaluation

Our grading scheme is relatively simple.

Requirement	**Marks**
File "sengfmt2.py" and "formatter.py" runs without errors.	2
The program is clearly written and uses **functions** appropriately (i.e., is well structured).	3
Errors have been enumerated and are either suitably handled or a sensible response strategy has been suggested.	3
Program can handle both stdin and file inputs.	2
Code passes the first set of test cases for multiple "?mrgn" (1 to 3)	Bonus +1
Code passes the second set of test cases for multiple "?maxwidth" (4 to 6)	Bonus +1
Code passes the third set of test cases for "?replace" (7)	5
Code passes the second set of test cases for "?replace" combined with "?mrgn" and "?maxwidth" (8)	Bonus +1

Code passes the third set of test cases for “?monthabbr”(9)	5
Code passes the second set of test cases for “?monthabbr” combined with “?mrgn” and “?maxwidth” (10)	Bonus +1
Total	**20 + (4)**