**Software Engineering 265**
**Software Development Methods**
**Summer 2020**

*Assignment 2*

Due: Friday, July 3rd, 11:55 pm by "git push"
(Late submissions **not** accepted)

## Programming environment

For this assignment you must ensure your work executes correctly on the virtual machines you installed as part of Assignment #0 (which I have taken to calling `Senjhalla`). This is our "reference platform". This same environment will also be used by the course instructor and the rest of the teaching team when evaluating submitted work from students.

Any programming done outside of this reference platform might not work correctly. Such work may receive a failing grade.

## 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 will be used to examine submitted work.)

## Objectives of this assignment

- Learn to use basic features of the Python 3 language (with your submitted solution using the following bangpath: `#!/usr/bin/env python3`).
- Use Python 3 to write a less resource-restricted implementation of `Senjify` (called `senjify2.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 from A#1, plus ten new cases added for A#2).

**This assignment: `senjify2.py`**

You are to write a Python version of the formatter. You solved a version of this problem in C, but be careful as you should avoid attempting a port of your Assignment #1 solution line-by-line into Python. The languages are very different.

All of the formatting capabilities of Assignment #1 are to be implemented in the solution to this assignment. Like the C program, the program is to run as a command in the bash shell. There are also several new abilities your solution must have in addition to those from assignment #1:

- If a filename is provided to the script, then that file's contents are formatted. If no filename is provided, then the contents from `stdin` are formatted. In either case, the formatted output is output to `stdout`.
- The indent may now be specified relative to the value of the current indent position. For example, if the indent currently is `10`, then when the command `{{ +>5 }}` is encountered, the indent is changed to `15`. If the indent is currently `20`, then a command of `{{ ->7 }}` will set the indent to `13`, and a another `{{ ->7 }}` will set the indent to `6`. This is a relative-indent command.
- Valid indents must be (a) greater than or equal to 0, and (b) less than or equal to the current page width minus 20. If an indent would be negative after a relative-indent command, then the indent is set to zero after that relative-indent command. If the indent would be greater than page width minus 20, then the indent is set to the page width minus 20.
- There is no limit to the number of input lines. You may assume that each line is no longer than 132 character.

Assuming you have copied the tests inputs and outputs from the SENG servers (i.e., from `/home/zastre/seng265/assign2`) into `Senjhalla`, and that this subdirectory is named `tests/` and is located within the directory you are using for writing `senjify2.py`, then the input would be transformed into the output via the following command:

```
% cat tests/in04.txt | ./senjify2.py > testout04.txt
```

and the file `testout04.txt` produced by re-directing the output from the program appears here in the same directory as `senjify2.py`. To compare the output produced by `senjify2.py` with what is expected, you can use the Unix `diff` command as shown below (assuming you're still in the same directory as when you executed the command above):

```
% diff testout04.txt tests/out04.txt
```

In you wish, you can combine both the formatting and the testing into one longer command by using only Unix pipes:

```
% cat tests/in04.txt | ./senjify2.py | diff - tests/out04.txt
```

Also notice it is now possible to provide the input filename on the command line rather than piping input to `stdin`. This is show in the following which repeats the two commands above using `senjify2.py`:

```
% ./senjify2.py tests/in04.txt > testout04.txt
```

```
% ./senjify2.py tests/in04.txt | diff - tests/out04.txt
```

Your program must support both styles of usage (i.e., input specified as a filename to the program, or input provided as a stream of characters from `stdin`).

**Exercises for this assignment**

1. Write your program the `a2` directory within your local repo for SENG 265. Amongst other tasks you will need to:
   - read text input from a `stdin`, line by line;
   - open a text file based on the its name given as a command-line argument;
   - write output to `stdout`;
   - extract substrings from lines produced when reading a file;
   - create and use lists or tuples or dictionaries or all of these.

2. **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 #4). **A reminder: Do not use regular expressions or new Python classes for this assignment (#2).**

3. 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".

4. 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.

5. Reasonable run-time performance of your script is expected. No test case should take longer than 15 seconds to complete.

**What you must submit**

- A single Python source file named `senjify2.py` within your git repository (and within its `a2/` directory) containing your solution to Assignment #2. Ensure your work is **committed** to your local repository **and pushed** to the remote **before the due date/time**. (You may keep extra files that you have used during development within the repository.)

**Evaluation**

Our grading scheme is relatively simple.

- "A" grade: A submission completing the requirements of the assignment. Submitted code is well-structured and very clearly written. Global variables are kept to a minimum. `senjify2.py` runs without any problems; that is, all tests pass and therefore no extraneous output is produced.
- "B" grade: A submission completing the requirements of the assignment. `senjify2.py` runs without any problems; that is, all tests pass and therefore no extraneous output is produced. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. `senjify2.py` runs with some problems.
- "D" grade: A serious attempt at completing requirements for the assignment. `senjify2.py` runs with quite a few problems. *Most tests do not pass, although some tests do pass.*
- "F" grade: Either no submission given, or submission represents very little work. *No tests pass.*