Software Engineering 265
Software Development Methods
Spring 2019

Assignment 3

Due: Monday, March 25, 4:30 pm by submission via git push
(no late submissions accepted)

Programming environment

For this assignment please ensure your work executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself a few days before the due date to iron out any bugs in the Python program you have uploaded to the lab machines. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

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 programs.)

Objectives of this assignment

- Understand a problem description, along with the role used by sample input and output for providing such a description.
- Use the Python programming language (specifically Python 3) and its module mechanism to write an improved version of the MTF encoder and decoder.
- Use git to manage changes in your source code and annotate the evolution of your solution with "messages" with committed changes.
- Test your code against the twenty provided test cases.

This assignment: "coding2.py"

We will continue with our MTF coding exercise. The main differences this time around are that:

- The module mechanism in Python will be used to organize your application.
- You will allow for a much larger dictionary.

Module mechanism

Two files have been provided to you named *encode.py* and *decode.py*. They refer to the functionality within some *coding2.py* module, and it is this latter module which you must write. You should assume these two provided files are located in the same directory as your *coding2.py* file. Each of the provided files calls one of the functions in the *coding2* module (i.e., passes the name of the file to be processed).

Note that we will no longer use symbolic links to your Python script as we did in previous assignment.

Larger dictionary

Our original implementations of MTF coding were seriously hampered by the limitation of a dictionary to a maximum of 120 words. In order to apply such coding to realistic texts we need a larger dictionary.

To increase the range of codes we will use the following scheme:

- A code from 1 to 120 will be encoded as (code + 128); that is, the code will fit in one char.

- Codes from 121 to 375 will be represented using two chars: the first char is (121 + 128), and the second code is (code - 121). For example, if the code would be 300, then the first output char is 0xF9, and the second would be 0xB3. (The code for 121 would be F9 00.)

- Codes from 376 to 65912 will be represented using three chars: the first char is (122 + 128), the second is ((code - 376) // 256), and third is ((code - 376) % 256). In Python integer division is indicated by the "//" symbol. For example, if the code is 30000, then the first output char would be 0xFA, the second 0x73, and the third 0xB8. (The code for 376 would be FA 00 00.)

Since the resulting MTF files will produce codes that cannot be used by earlier versions of the encoder and decoder, we need to change the magic number for our MTF files. It will now be BA5EBA12 (i.e., last hex digit is changed). Note, however, that this third program must be able to process MTF files created in the previous two assignments. (That is, your new program should check that one of the two magic numbers – BA5EBA11 or BA5EBA12 – starts the MTF file.)

You are, of course, free to use any part of your assignment #2 solution as will help with completion of assignment #3.

Exercises for this assignment

1. Within your git repo ensure there is an *a3* subdirectory. All directories and program files you create are to be source-controlled by git. You need not add the directory containing test files to your project unless you wish to do so. New test files are available at */home/zastre/seng265/a3/tests)* (and the older tests from A#1 and A#2 are still in */home/zastre/seng265/a1/tests*).

2. The scripts *encode.py* and *decode.py* are in the */home/zastre/seng265/a3* directory. ***The only modification you are permitted to make to these two files is changing the path of the Python interpreter that appears on the first line.***

3. Write your program.

4. You are welcome to use classes and regular expressions if you wish.

5. Use the test files to guide your implementation effort. Start with the simple example in test 01 and move onto 02, 03, etc. in order. (Note that test19 is particularly punishing.) **Refrain from writing the program all at once, and budget time to anticipate when things go wrong!** Use the Unix command *cmp* to compare your MTF files and *diff* to compare text files.

6. For this assignment you can assume all test inputs will be well-formed (i.e., our marking team will not test your submission for handling of input or for arguments containing errors). The next assignment might specify error-handling as part of its requirements.

7. Write two test scripts, i.e., Unix scripts that go through all of the twenty provided test cases (one script for encoding, one script for decoding) and reports which test cases fail.

What you must submit

- A single Python script named *coding2.py* within your git repository containing a solution to Assignment #3.

- Two Unix scripts (*i.e.*, one for testing encoding functionality, one for testing decoding functionality).

Evaluation

Our grading scheme is relatively simple.

- "A" grade: An exceptional submission demonstrating creativity and initiative. The code within `coding2.py` runs without any problems. Test scripts are provided. The program is clearly written and is structured in a way that also uses functions appropriate.
- "B" grade: A submission completing the requirements of the assignment. The code within `coding2.py` runs without any problems. Test scripts are provided. The program is clearly written.
- "C" grade: A submission completing most of the requirements of the assignment. The code within `coding2.py` runs with some problems. Test scripts might be missing.
- "D" grade: A serious attempt at completing requirements for the assignment. The code within `coding2.py` runs with quite a few problems. At least a handful of tests pass. Test scripts might be missing.
- "F" grade: Submission either represents very little work or cannot be executed for testing.

Appendix

Python runs on many different platforms and operating systems, and the language compensates for the different ways newlines are represented. Unfortunately this can cause us some grief. For example, here are different ways of indicating a newline:

- “\n”, or ASCII code 10
- “\r”, or ASCII code 13
- “\r\n” or ASCII codes 10 plus 13 (on Windows)

The problem for us is not so much these differences as with the way Python attempts to isolate the programmer from the differences. Specifically the problem has to do reading newline character(s). Even if a file is opened with “latin-1” encoding:

- “\n” (or 0x0a) in the file is returned as 0x0a
- “\r” (or 0x0d) in the file is returned as 0x0a

That’s no typo! If you expect to read ASCII 13, Python will instead give you ASCII 10.

There is one solution to this that involves one more parameter to open(). To ensure ASCII 13 is input as ASCII 13, use the following:

```
file = open(<filename>, encoding="latin-1", mode="r", newline="")
```

This will prevent Python from performing newline-character substitutions when reading newlines from this file object.

(Those of you with solutions reading the file as raw bytes might not encounter this situation.)