CSCI 232: Data Structures & Algorithms: Project 1

Due Wednesday, September 28 @ 11pm

This project has two parts.

Part 1. Problem description

Write a program that calculates and prints statistics about the contents of a text file. In particular, we want to know how many characters, lines, and words a given text file contains. In addition to the number of words, we also want to know what the top ten most frequently occurring words in the file are, sorted by frequency (high to low).

As a short example, consider the following piece of text:

> *A long time ago, in a galaxy far, far away …*

We can see that it contains:

- One line of text. Assume that the return-line character, **\n**, is used to indicate the end of a line, and that every text file that is not empty is at least one line long.
- Forty-six characters, including spaces and punctuation.
- Ten words in total. However, note that there are only eight unique words, because ***far*** and ***A*** both occur twice.

For your program to run correctly and handle situations like those presented with the words *far* and *A* as described above, we need to give a precise definition of what we consider a word to be. For the purposes of this project, you should consider a word to have one or more characters in length, and each character is one of the lowercase characters *a* to *z*. We will ignore non-letters (e.g., digits and punctuation), and convert uppercase letters to lowercase.

There is no good way to remove punctuation symbols from some words. For example, how should you handle the apostrophe in *I'd*? If you delete it (and convert the *I* to lowercase), you get *id*, which is a different word. If you replace the apostrophe with a space, then you get *I* and *d* – one word, and one non-word. To solve this problem, we will treat apostrophes—and also hyphens—as "letters".

Running your program on the sample text given above should produce output similar to:

```
>>> filestats_reimer("test.txt")
The file 'test.txt' has:
    46 characters
    1 lines
    10 words

The top 10 most frequent words are:
 1.    2 far
 2.    2 a
 3.    1 time
 4.    1 long
 5.    1 in
 6.    1 galaxy
 7.    1 away
 8.    1 ago
>>>
```

Be sure to test your program on many different sample texts. A large data file you can test against, ***bill.txt***, is available for download from our class webpage. This 5.3 megabyte file contains the complete works of Shakespeare (free on the Project Gutenberg site, www.gutenberg.org). Running your program on ***bill.txt*** should produce output similar to:

```
>>> filestats_reimer("bill.txt")
The file 'bill.txt' has:
    5465102 characters
    124787 lines
    897585 words

The top 10 most frequent words are:
 1. 27568 the
 2. 26705 and
 3. 20115 i
 4. 19210 to
 5. 18263 of
 6. 14391 a
 7. 13606 you
 8. 12461 my
 9. 11107 that
10. 11001 in
```

```
>>>
```

Part 1. Deliverables

When you have completed and tested your program, call it **filestats_*lastname.py***, where *lastname* is your last name (e.g., filestats_reimer). Upload your finished program to the Moodle drop-box by the due date/time. Note that I expect your submission to be the result of your individual work and understanding of the project. I should be able to run your program from IDLE using a command like:

```
>>>filestats_reimer("bill.txt")
```

Part 2. Problem description

Write an encryption program that is based on the idea of shifting each letter of a plaintext message a fixed number (called the key) of positions in the alphabet. For example, if the key value is 3, the word "Computers" would be encoded as "Frpsxwhuv". Your program should adhere to the following requirements:

- It should interact with the user by prompting them to enter a phrase, the key value, and whether or not they want to **encode** or **decode** the phrase.
- Multi-word phrases should be accepted, and phrases are allowed to contain both words and integers (of any length). However, no words in a phrase will mix letters and numbers. For example:
  - "Anne teaches CSCI 232" IS acceptable (has blank between CSCI and 232)
  - "Anne teaches CSCI232 " is NOT acceptable (no blank between CSCI and 232)
- In addition to handling any possible integers in the phrase, your program only has to handle characters from 'A' to 'Z' and 'a' to 'z', and the blank space. For positioning purposes, capital letters in the alphabet should precede a blank character, which should precede the lower case letters in the alphabet.
- Your algorithm should not drop off at the end of the alphabet, but rather should circle back around. For example, the next character after the 'z' should be 'A'.
- For all words in the phrase, the key value should be used to encode it by shifting each letter a fixed number (the key) of positions in the alphabet, as described above.

- When an integer is discovered in the phrase, however, the encoding scheme is slightly different. For this, you are to create and use a text file called "digitencode.txt" for the program. This file maps a keyboard character (other than letters) to each digit as shown in the sample encryption file. Your program should read this text file in and create a **dictionary** for each pair of entries. Then, the dictionary should be used to locate the appropriate character to substitute in the encoded phrase for each digit in the integer value. For example, if the integer **5601** is encountered in the input phrase, using the sample digitencode.txt file shown, it would be encoded as **:!$(** Note that your program should work with ANY digitencode.txt file as long as the format is the same as shown (i.e., two columns of 10 rows, with one digit and one character per row, each pair separated by a blank space)
- Any original message should be recoverable by "reencoding" it using the negative of the encryption scheme. This option should run if the user selects decode when originally prompted.
- Make sure you carefully test all aspects of your program with a variety of cases/inputs.
- All parts of your program should be carefully commented so it is easy to read and understand.

Sample program execution of Encode:

```
>>>
Enter a phrase: Anne teaches CSCI 232
Enter the key value: 2
Type 0 if you would like to Encode, or 1 if you would like to Decode: 0

Your encoded message is:
CppgbvgcejgubEUEKb%{%
>>>
```

Sample program execution of Decode:

```
>>>
Enter a phrase: CppgbvgcejgubEUEKb%{%
Enter the key value: 2
Type 0 if you would like to Encode, or 1 if you would like to Decode: 1

Your decoded message is:
Anne teaches CSCI 232
>>>
```

digitencode.txt

```
0 $
1 (
2 %
3 {
4 -
5 :
6 !
7 @
8 =
9 ^
```

Part 2. Deliverables

Name your program code as **encrypt_*lastname*.py**, where *lastname* is your lastname (e.g., encrypt_reimer.py). Submit your program code and your digitencode.txt file to the Moodle drop-box by the due date/time. Note that I expect your submission to be the result of your individual work and understanding of the project.

Grading

Your project will be graded on the following elements:

- **Correctness**: For full credit here, your program should produce the correct output in ALL cases.
- **Adherence to specifications**: For full credit here, your program should follow all given specifications carefully, including application of specific implementation details when provided.
- **Input/output**: For full credit here, your program should provide correct input prompts, should format all output carefully, and should work with any input files as long as they are in the same format as shown in the specifications.
- **Code organization**: For full credit here, your program should be clearly organized and modular. Sensible variable names and function calls should be used.
- **Readability:** For full credit here, your program should be well commented and easy to read and understand. Documentation strings should be used for all modules and functions.