Caesar Cipher - part 2: decryption

The purpose of this project is to familiarize you with file reading, lists, and more string methods. You will create a program to automatically decrypt a message. You will complete this assignment with your same partner from the encrypt assignment since it's related, and then we will change up for the next assignment. Don't forget good coding style and comments, as well as the quality/efficiency of your solution!

Deliverables: The usual zip folder named username1_username2 containing the file decrypt.py.

Caesar Cipher Cracking

To find the "shift" that was used to create a given ciphertext from plaintext, you need to know about cracking Caesar ciphers using a technique that has been around for over a thousand years. Any language such as English has a known distribution for each letter. For example, the letter "E" is the most common letter in English making up 12.702% of the letters on average (ignoring case). The letter "T" is next (9.056%), followed by "A" (8.17%), and so on. The order "E"-"T"-"A" is what matters, not the percentage, but Wikipedia explains the complete distribution for the curious:

http://en.wikipedia.org/wiki/Letter_frequency#Relative_frequencies_of_letters_in_the_English_language

The procedure begins by finding the most common letter. You can guess that the most common letter maps to "E". You can now find the "shift" from the most common letter in the cipher-text to the expected most common letter "E". For example, if the most common letter in the cipher-text is "H", you know that the shift from "E" to "H" is 3, and so you guess that a shift of 3 was used to encode this text. You should do a check of your guessed shift using the knowledge that the next most common letters are "A" and "T". Find the 2nd most common letter in the **ciphertext** and see if it matches **either** "A" or "T" shifted by your guessed amount. If so, you can assume it is the correct shift, then apply the shift to all the letters in the ciphertext and get the original plain-text message. If it doesn't match either of these print a message saying you cannot decrypt the file **and quit**.

Just like your previous assignment you should not try and shift any non-alphabet characters, just leave them in the message as they are. You should **NOT** use any dictionaries in this project (in case you're looking ahead); you can do everything you need more efficiently with lists and strings.

Examples:
ciphertext: *Uif dbu dmbxfe bu uif qfstpo't tipfmbdf*.
- most common letter is 'f' (6 total). Shift 'e' to 'f' is 1, so guess encoding shift was 1.
- 2nd most common letter is 'u' or 'b' (each 4 total). In case of a tie you can choose either, I will go with 'u'. Determine that 't' is letter that would shift to 'u' with a shift of 1, so I am confident my shift is correct and I shift every letter backwards 1 to get back to the original message: *The cat clawed at the person's shoelace*.

ciphertext: *Hyhubrqh oryhv frpsxwhu vflhqfh*!
- most common letter is 'h' (7 tot). Shift 'e' to 'h' is 3, so guess encoding shift was 3.
- 2nd most common letter is 'f' or 'r', going with 'f' determine that 'c' is letter that would shift to 'f' with shift of 3. 'c' is not 'a' or 't' so output that this text cannot be decoded with any certainty.

Note that in the 2nd example a shift of 3 was actually correct, try shifting the message above 3 to the left and see what you get. However the computer does not know that, so if the 2nd most common letter is not 'a' or 't' you should not even try to decrypt the message, just output that you can't.

Your Task

Create a program that reads in encrypted text from a file named "coded.txt". You have to read the entire file to get the letter frequencies and determine the shift before you can start decrypting. It is inefficient to read the file twice, instead you should save the text of the file in a string so that after determining the letter frequencies you can simply go through the saved string applying the correct shift to actually decrypt the message. You will write the decrypted text to another file called "decoded.txt".

You must create and use at least 2 non-trivial functions of your own in your solution other than main(). For example you could create a function for step 1 below called *readCipher* that opens and reads from the file "coded.txt", returning a string with all of the text from the file. Or you could create a function for step 2 below called *getCount* that takes the string of text as an argument and returns a list of the counts for each letter of the alphabet. Or really you could encapsulate any individual step below into a reasonable function.

Your high level algorithm will be:

1. Read the cipher-text from the file "coded.txt"
2. Get a count of each character in the entire ciphertext (ignore non-alphabetic characters)
3. Find the 2 most common characters
4. Find the shift from "e" to the first most common character.
5. Check that the same shift translates the second most common character to either "a" or "t", if it does not simply print a message and quit.

6. Using the shift, decode each character of the cipher-text and write the plain text to the file "decoded.txt".

I have included 2 test files. cryptWiki.txt is an encrypted version of the Wikipedia page on cryptography with a few extra words at the end to make the frequencies work. You SHOULD be able to successfully decrypt this file. The file fails.txt should NOT be decryptable by your program. If you are not getting the correct results, make sure that you are counting both upper and lower case appearances of each letter in the same count! Feel free to test it on whatever else you want as well, but **note that it may not work well for small amounts of text.** In fact it took me a few tries to find a short example message that was decryptable using the criteria of this assignment.

Notes

1. Helpful functions: here are some functions that may be useful in your solution. Read the documentation for each of these and play around with them until you understand what they do! Remember you can type help(type) to see all of the functions available. For instance help(list) will show you all the list functions and help(str) all the string functions. I've also included a link to the documentation for each and a little help understanding it in some cases.

   `max(alist)` - documentation (returns the max value in the given list)

   `alist.index(value)` - documentation

   `astring.strip()` - documentation (by default strips all whitespace)

   `astring.lower()` - documentation

2. Think about the best way to keep track of the character counts. It makes sense to use a list of 26 ints for this, how could you map each character to a specific index in this list?

3. Make sure you count both upper and lower case characters when determining the frequencies. For instance both "C" and "c" should increment the count at index 2 in your frequency list. This is where the `lower()` function may come in handy.

4. Do the Algorithm steps separately—test each part before moving on to the next.

   a. Make sure you can read the file properly and write it back out as is.

b. Write the code to create the list of frequency counts of each letter and print the list. Test that it works on small, simple files. For example create a file with the text “aaaAAAbcdefghijklmnopqrstuvwxyz” and see if your frequency list is correctly 6 a’s, 1 b, 1 c, …, 1 z.
c. Determine the first and second most frequent characters and again use simple files to check that they are correct.
d. Calculate the shift from “e” to the most frequent character, check that it is correct.
e. Write code to check that the second most frequent character represents “a” or “t” given the shift above, test that it works.
f. Finally write the code to decrypt the ciphertext and test it.