Caesar Cipher 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 individually, though you can get help as usual from anyone. Make sure you are citing anyone you receive help from in your comments! Use good coding style, and think about ways you can make your program more efficient/clear when possible.

Deliverables: Your solution in the file yourusername.py.

Caesar Cipher Cracking

The Caesar cipher is named after Julius Caesar who used this type of encryption to keep his military communications secret. A Caesar cipher replaces each plain-text letter with one that is a fixed number of places down the alphabet. The plain-text is the original message; the cipher-text is the encrypted message. For example, in a shift of 3, "B" in the plain-text becomes "E" in the cipher-text, "C" becomes "F", and so on. The mapping wraps around so that "X" maps to "A", "Y" maps to "B", and so on.

Here is the complete mapping for a shift of three:

```
Plain:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC
```

An encrypted message substitutes the corresponding cipher-text letter for each plain-text letter. For example, here is an encryption of "Hello World" using our shift-three cipher:

```
Plain:   Hello World
Cipher:  Khoor Zruog
```

To decrypt the message, you would reverse the process if you knew what shift was used for encryption. However, if you get lucky enough to intercept a message from your arch-nemesis, you will need to somehow determine what shift was used just from the cipher-text. As a human, you could just try each of the 25 possibilities and look at the results, but a computer can't just "see" if a message looks correct or not, so you need a more robust method to decrypt messages automatically.

You will explore 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 letter that appears most frequently in the cipher-text, and guessing that that letter maps to "E". You find the shift amount between that most common letter and "E" and see if it holds up in mapping the 2nd most common cipher-text letter to one of the next most common English letters. 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 then find that the 2nd most common letter in the cipher-text is "Q", which when shifted 3 maps to "T". Since "T" is the 2nd most common letter in actual English, you grow confident that this is the correct shift and go ahead and decipher the text. If the 2nd most common letter had been "X" which shifted 3 maps to "A", you also would have confirmed the shift and used it to decipher the text. If the 2nd most common letter had been

anything else, however, you would be unsure about your guessed shift amount and not try to decrypt the text, instead you would tell the user that you couldn't automatically decipher in this case.

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 does not map to 'a' or 't' with your guessed shift amount, 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, determines if it can be decrypted or not, and if so writes the decrypted text to another file. The general algorithm is given here, with more details on each step below

1. Read the cipher-text from the file "coded.txt"
2. Get a count of each character in the entire cipher-text
3. Use the counts to determine which character is most common, and find the shift to it from "E"
4. Use the counts to determine which character is the 2nd most common in the cipher-text. Use the shift found in step 3 to find the corresponding plain-text character. If it is "A" or "T", go to step 5, otherwise quit.
5. Using the shift amount from step 3, decode each character of the cipher-text string, and write the resulting plain text string to the file "decoded.txt".

1. You should use the file `read()` method to save the entire file into one string variable. You will loop through this string once to get the counts, and again to do the actual deciphering.

2. You can do this very efficiently using a list of 26 "counts", one corresponding to each letter of the alphabet. You should NOT call the `count()` method 26 times on your string. You should instead initialize your list to hold 26 0's. Your list will use a mapping with the letter "a" at index 0, and the letter "z" at index 25. You can then loop through all characters in the cipher-text string, and each time you encounter an alphabetical letter, increment its corresponding count in the list. You can use the `ord()` function to easily determine the correct list index for a given letter in the alphabet. Make sure you increment the count for a letter no matter whether it's lower or upper case in the cipher-text.

3. There are many ways to find the highest number in a list, one useful list function is mentioned in the notes below. Once you determine what index in the list holds the highest count, you can use the `ord()` and/or `chr()` functions to determine which character that index corresponds to. Then you can calculate the shift amount from that character to the letter "E".

4. There are many ways to find the 2nd most common letter as well, though the method you used in part 3 may not work without modification. Let's call the 2nd most common letter *maxChar2*. Apply the shift you found in step 3 to *maxChar2* and see what plain-text letter it will be when decrypted. If that plain-text letter is either "A" or "T", this shift is likely correct, move on to step 5. If it is something other than "A" or "T", you should just output a message saying you cannot decrypt this string and end the program without writing any decrypted text to a file.

5. Create a new empty string, then loop through the original cipher-text string, adding each character as you go to the new string. If the cipher character is not an alphabetical letter, just add it as is to the new string. If it is a letter, apply the shift and add the decoded letter to the new string. Keep uppercase letters uppercase in the decrypted text, but shifted just like lowercase letters (see my Hello World example from the very beginning of the assignment description). Use the file method `write()` to write the entire decrypted string to the file.

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. In Python, to open a file to read from, use the following syntax, where "filename" is the entire name of the file, including the extension, e.g. "coded.txt".
   ```
   inputFile = open("filename")
   ```

   This will save the open file in the variable inputFile, but not actually read anything from it yet. Use the syntax
   ```
   readString = inputFile.read()
   ```
   to read the entire text of the file and save it in the string readString.

2. To open a file to write to, use a similar command as to read a file, except with an extra argument "w" in the open function ("w" stands for "write" since we want to write to this file).
   ```
   outputFile = open("filename", "w")
   ```

   To write a string to the file, use the syntax
   ```
   outputFile.write(theString)
   ```

3. Before exiting your program, you should close all open files. The same command does this, whether you opened a file for reading or writing. The syntax is `theFile.close()` For example, to close your output file, use `outputFile.close()`

4. Python has a built-in function "ord(character)" that returns an integer value (ASCII) representing the character. The letters in the alphabet are represented by the values 97 to 122 for lowercase 'a' to 'z', and 65 to 90 for uppercase 'A' to 'Z'. For example:

```
ord('a') # returns 97
ord('b') # returns 98
ord('B') # returns 66
ord('Z') # returns 90
```

5. Python has another built-in function "chr(int)" that returns the character represented by the given integer value in ASCII. For example:

```
chr(97) # returns 'a'
ord(98) # returns 'b'
ord(66) # returns 'B'
ord(90) # returns 'Z'
```

   Why do the capital letters have lower ASCII integer values than the lowercase letters? I have no idea, just the way it is...

6. You can compare strings using the same operators as comparing numbers, such as <, >, ==, etc. String comparison uses alphabetical order, just like in a dictionary, so 'a'<'b' is True, and 'camel' < 'cat' is True, and 'dog' >= 'horse' is False. Think about how you can use this to determine if a character is lowercase, capital, or something else entirely.

7. More 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 (returns index of given value)

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

   `astring.lower()` - documentation (returns lowercase version of astring)

8. Use a list of 26 ints to keep track of character counts. How could you map each character to a specific index in this list?

9. 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.

10. Do the Algorithm steps sequentially—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 list of counts correctly shows 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 cipher-text and test it.