How to break a substitution code

In this laboratory, we learn how frequency counting allows us to break a simple code.

As usual in old-fashioned cryptography, we assume a text file consisting of capital letters only. A substitution code then takes the plaintext (a.k.a cleartext) and changes all letters according to a simple substitution table.

For example, the substitution table could be

clear	cipher	clear	cipher	clear	cipher	clear	cipher	clear	cipher	clear	cipher
A	D	F	C	K	P	P	X	U	S	Z	F
B	A	G	Q	L	H	Q	I	V	L		
C	J	H	M	M	T	R	E	W	V		
D	G	I	R	N	B	S	Z	X	K		
E	O	J	W	O	Y	T	N	Y	U		

A message such as “Attack north gate at six thirty in the morning”, would first be normalized to

“ATTA CKNO RTHG ATEA SIXT HIRT YINT HEMO RNIN G”

and then translated letter by letter using this table to

“DNND JPBY ENMQ DNOD ZNKN MNEN UNBN MOTY EBNB Q”

(We somehow arbitrarily broke the string into groups of four letters to make it more readable.) Since there are 26! = 40 329 146 112 660 563 558 400 000 different substitution tables, this might seem to offer good protection. In fact, it offers such poor protection that with additional hints it is used to generate newspaper puzzles.

With this substitution table, the letter ‘E’ is always translated into the letter ‘O’. Since in English text, the letter ‘E’ is almost always the most frequent, the letter ‘O’ is almost always the most frequent in any cipher encrypted with this particular substitution table. Reversely, the most frequent letter in any cipher encrypted with an arbitrary substitution table is the one which is substituted for ‘E’.

Task 1: Write a function that gives the frequency of all characters in a file.

Hint: Since you want to print out the frequencies in descending order, you might want to use the counter in the collections module. Here is some code with counter that works on a string.

```
def count(string):
    ct = collections.Counter()
    for letter in string:
        ct[letter] +=1
    print (ct.most_common())
```

The counter is almost like a dictionary, but its default value is zero. It also has a handy most_common method that prints out the count in descending order.

Task 2: Write a function that converts a text file into one where all letters are in capitals and where all punctuation symbols and white spaces are suppressed. You can keep the end-line characters. Then use the function from the first task in order to count the number of letters in a number of english text files. Compare the numbers you obtained with the frequencies according to wikipedia.

You are given a cipher text encrypted with a secret substitution code. (We can figure out that it is a substitution code because the frequencies of the letters in it correspond to the frequency of English letters. This also means that we can distinguish between clear texts in various languages). By looking for the most frequent letters in the cipher and comparing them to the frequencies of English letters, you can make a number of guesses for substitutions. For instance, the letter substituted for 'E' is the most frequent letter in the cipher. This method breaks down for smaller frequency counts because the frequency counts differ between texts.

Task 3: Use the counter function in order to determine the most frequent letters in the cipher.

We can extend this method further. Notice that the lines in the cipher are not of the same length. Indeed, there appear empty lines and short lines, indicating that the encryption keeps the line structure of the text file the same. We can use this to compare the frequency of the first letters between English text and the cipher. Similarly, for the last letter.

Task 4: Determine the frequency of the first letter in a line in English text. Similarly for the last letter. Then compare with the frequency of the first, respectively last letter of each line in the cipher. Does this confirm your guesses?

Text has much more structure. We can count the frequency of bigrams — two adjoining letters in English text — and trigrams — three adjoining letters in English text.

Task 5: Implement functions that count the bigrams and trigrams in text files. You only need to print out the first 20, since only the first three or four are useful. For example, for one file, I obtained "THE" with 20390, "ING" with 8132, and "AND" with 7946 occurrences. The next frequent one, "THA" at 4415 is only slightly over half of this. I will not be able to distinguish between "ING" and "AND" in the cipher based on this observation, but together with the previously obtained information, I should be able to distinguish them.

At this point, you should have a number of guesses. The next task would be to see whether you can guess any other letters. But this is no longer Programming in Python.