CS 101 – Problem Solving & Programming 1
Program Assignment 3
Algorithm Due Sep 13, 2012
Program Due Sep 16, 2012

One of the oldest encryption methods is known as the *Caesar Cipher*, so-called because it was used by Julius Caesar. It is based on a fixed shift, substituting each letter with the one coming *k* letters later in the alphabet, wrapping back around as necessary. For example, if *k* == 3, then A becomes D, B becomes E, and so on. W becomes Z, X becomes A, Y becomes B, and Z becomes C. Thus, the sentence

```
PROGRAMMING IS FUN
```

becomes

```
SURJUDPPLQJ LV IXQ
```

Using a different value for *k* results in a different cipher. Obviously, for English there are 25 possible values (since *k* == 0, where the original text is equal to the encrypted text, is useless).

Breaking such a cipher is pretty straightforward, based on the frequency of the characters that result. For example, in English, the most common letter is E. Thus, whatever character appears most frequently in the cipher text probably corresponds to E. Likewise, the next most-common letters in English are T and A. Thus, if we look at the 3 most common letters in the cipher text, and find they all have the same shift from E, T, and A respectively, we have almost certainly found the correct shift to decipher the text.

Program specification:
Your program will provide the ability to produce a Caesar cipher for a particular shift, to decode a Caesar cipher where the shift is known, and to decode a Caesar cipher where the shift is unknown. *Your program must use functions for each of these tasks.*

Your program will give the user options to encode a message, decode a message in a file with a known shift, and decode a message in a file with an unknown shift. All output for this program will be to the screen.

If the user wants to encrypt a message, ask the user for the text to be encrypted, and the size of the shift to apply (which you can assume will be an integer from 1 to 25 inclusive). Print the encrypted version of their input. (You can assume their input is all upper-case, but may contain punctuation or spaces.)

If the user wants to decrypt a message, ask the user for the name of the file. (See below for a very quick overview of reading from a text file.) If the user knows the shift, ask for it, and again, you can assume the shift will be in the range 1-25. Apply the known shift and print the decrypted message. If the user doesn't know the shift, produce a trial decryption by finding the three most common characters and assuming they correspond to E, T, and A. By finding the difference between the most common character and 'E', we can deduce the shift. If we are correct, the same procedure using the second- and third-most common characters for T and A should produce the same shift. If all three are correct, we almost certainly have the correct shift. If two of the three have the same value, we use that value.

All text will be upper-case. Any non-letter character (spaces, punctuation, etc.) should be unchanged in encryption or decryption. Such characters are usually removed in actual application, so there are fewer clues about word length, etc., but we'll leave them in for this assignment so you can tell how well your program is working.

Deliverable: Your source code file, uploaded to Blackboard by the deadline. Also on the Blackboard site, you're provided with a plain text file, the encryption of that file (so you can test your program), and another encrypted file you can test your program against.

Development notes:
There are a few things you'll need to know to write this program. We'll talk about files later in the semester, but for now here's what you need to know about reading from a file: If the file is in the same folder your program is running, you can read the contents of the file in a single command:

```
S = open('filename.txt').read()
```

Or, if you get the file name from the user:

```
Name = input("What's the filename? ")
S = open(Name).read()
```

This reads the entire contents of the file into a single string.

The ord() function takes a character as its parameter and returns an integer value corresponding to the Unicode value of that character. For plain text files, the Unicode value is the same as the ASCII value. This means you can find the location of a character in the alphabet by computing the difference between that character and the character 'A' (for uppercase characters). Type this into a shell and see what you get: ord('C') – ord('A').

Likewise, the chr() function takes an integer as its parameter and returns a Unicode character. If we know which letter of the alphabet something is (with *A counting as 0*), we can find the letter of a number in the range 1-25 by: chr(N+ord('A')). Try this in the shell for a few values of N to see what happens.

The 'shift' which is the key for the cipher can be applied by adding it to the character value. The modulus operator can be used to handle the problem of 'wrapping around' if we go too far: CipherValue = (OriginalValue+Shift) % 26. Decryption works by reversing the process, but the modulus operator won't help us directly; we'll have to subtract out the Shift value, then add 26 if the result is below 0. (Remember that in modulus arithmetic, the first letter, 'A', will have value 0.)

Another quirk of modular arithmetic can show up if making the shift causes a letter to 'wrap around' to the beginning. Finding the shift by subtracting ordinal values can cause the shift to appear as if it's a negative number. For example, a shift of 10 causes T to encode as D. If we're trying to find the shift by subtracting values (ord('D') – ord('T')), our shift will appear to be -16. In modular arithmetic -16 and 10 are equivalent; for this program, all you have to remember is that if the shift you calculate is less than 0, add 26 to it to get the actual shift.

It sometimes happens that the letter distribution of the plaintext isn't quite typical. If the shift you calculate for E, T, and A is the same, then you have almost certainly found the right shift and should use it. If two of them agree and one is different, use the value that two of them agree on. Your program will not have to deal with text that has different computed values for all three of the most common characters.

A string has methods to count the number of occurrences of each character. Finding the second and third most-common characters seems a bit tricky, until we remember that we can remove characters from a string easily, using the replace() method. Thus, we can find the most common character, and then remove it by replacing all occurrences with the empty string. NOTE: You should remove spaces from the text before starting to count characters, otherwise the space will appear to be the most common character.

Your program must have at least three functions:

- One function will take as parameters a string where all letters are upper-case, and an integer in the range 1-25. It will return the Caesar encryption of that string using that shift. Spaces and punctuation (anything other than an upper-case letter) should be unchanged by the function. This function does no interaction with the user—nothing from the keyboard, no output.
- Another will take a string with all letters upper-case, and an integer in the range 1-25. It will produce the Caesar decryption of that string using that shift. Again, punctuation and spaces should be left unchanged. This function also produces no output and does no input.
- The third function will take a string that is Caesar-encrypted. It will deduce the most likely shift and produce the decyption of the string using that shift, on the assumption that the 3 most common letters in the ciphertext represent E, T, and A.

You may use other functions as necessary or convenient, of course, but you *must* have at least these three.