COSC 101: Introduction to Computing I
Homework 10
Fall 2014

The goal of this homework is to give you practice with file processing, string and list methods and functions.

This homework is **due on Wednesday, November, 12, 2014 11:55pm**.

1 Introduction: cryptography

In this assignment, we look at *cryptography*. With cryptography, the sender wants to send a message that only the receiver can read. An eavesdropper may see the message, but because it is "scrambled," the eavesdropper cannot make sense of it. Only the receiver can "unscramble" or decipher the message. This is possible if the sender and receiver first exchange a key: a piece of information that works like a key in a lock. The sender uses the key to lock the message and the receiver uses a copy of the key unlock it. Without the key the eavesdropper cannot unlock it... well, maybe that's not entirely true. We'll revisit that question in the next homework assignment.

In this assignment, we look at two of the simplest forms of cryptography: the Caesar cipher and Vigenère cipher.

In general, cryptography depends on the sender and receiver establishing the key, a shared secret which they use to exchange messages. But how do they go about establishing the secret key? For additional background on how two parties can establish a shared secret *even in the presence of eavesdroppers*, you are encouraged to read Ch. 4 of "Nine Algorithms That Changed the Future" by John MacCormick, available electronically through the Colgate library. This reading is entirely *optional*. It is a fun and easy read.

2 Getting started

- In the `hw10.zip` file, we include some example files that you can use to test your code.
  - `plain1.txt`
  - `plain2.txt`
  - `plain3.txt`
  - `alice.txt`

  If you double click on these files on a Windows machine, it may open these files using Notepad, in which case they do not display correctly. I suggest opening them in IDLE or Notepad++.

  Be sure to place your `hw10.py` in the same folder as these files.

- Read the background section to familiarize yourself with the basic idea of the Caesar cipher and the Vigenère cipher.
- Read Section 4 to find out exactly what you need to do.
- It is strongly recommended that you implement the Caesar cipher first and get it working completely before you attempt the Vigenère cipher. And then once you get both cipher programs working, then think about deciphering.
- After you finish, review the checklist (Section 6).

3 Background

3.1 Caesar cipher

This section explains the Caesar cipher. You will need to have a good understanding of it to complete this homework. You might also check out the Wikipedia entry: `http://en.wikipedia.org/wiki/Caesar_cipher`.

First, some terminology:

- **plaintext**: the readable ("plain") text to be encoded – i.e., the message you want to send in secret.
- **ciphertext**: the encoded (scrambled) text. Ideally, even if this gets in the hands of an eavesdropper, he or she should not be able to figure out what the plaintext is.
- **cipher**: the specific algorithm used to encode the plaintext (or decode the ciphertext). For this assignment, the first cipher is the Caesar cipher, the second is the Vigenere cipher.
- **key**: a piece of information (in this case a number) that works like a key in a lock. This should be kept secret among those who want to exchange encoded messages.

Initially, assume that the plaintext and ciphertext contain only CAPITAL LETTERS.

Now, the Caesar cipher works as follows. First, you pick a secret key which must be an integer number. Let `key` be this number. Now, let's say that our plaintext is `'TOPSECRETINFO'`. The Caesar cipher hides the secret message by "rotating" each letter in the plaintext by `key` places in the alphabet. To rotate a letter means to shift it through the alphabet, wrapping around to the beginning if necessary.

Example 1 If we choose the number 2 as the `key`, the letters of the alphabet are shifted like this:

```
original alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 shifted alphabet: C D E F G H I J K L M N O P Q R S T U V W X Y Z A B
```

Given a plaintext message of `'TOPSECRETINFO'` and a `key` of 2, the ciphertext is: `'VQRUGETGVKPHQ'`. In other words, the `T` shifts two places to `V`, the `O` shifts two places to `Q`, etc.

Example 2 If we choose the number 25 as the `key`, the letters of the alphabet are shifted like this:

```
original alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 shifted alphabet: Z A B C D E F G H I J K L M N O P Q R S T U V W X Y
```

Given a plaintext message of `'TOPSECRETINFO'` and a `key` of 25, the ciphertext is: `'SNORDBQDSHMEN'`.

The easiest way to create a ciphertext is to map each letter to a number, do some math, and then translate the resulting number back to a letter. Let's assign numeric values to each letter, such that A corresponds to 0, B corresponds to 1,

and so on until we get to Z, which corresponds to 25. Now let's consider a plaintext message that consists of the single letter D. To make the ciphertext, we will use the key of 2. The process consists of four steps.

(1) First, we obtain the numeric value of our letter. The numeric value of D is 3.

(2) Second, we add the key to this number: 3 + 2 = 5.

(3) Third, we calculate the sum modulo 26 (meaning that we calculate the remainder when the sum is divided by 26). In this example, `5 % 26` is 5.

(4) Fourth, we take the result from the previous step and convert that number back to a letter. The number 5 corresponds to the letter F. Thus, when the key is 2 and the plaintext is D, the ciphertext is F.

Here's a second example, the plaintext is the letter Y and the key is 25.

(1) First, we obtain the numeric value of our letter. The numeric value of Y is 24.

(2) Second, we add the key to this number: 24 + 25 = 49.

(3) Third, we calculate the sum modulo 26. In this example, `49 % 26` is 23.

(4) Fourth, we take the result from the previous step and convert that number back to a letter. The number 23 corresponds to the letter X. Thus, when the key is 25 and the plaintext is Y, the ciphertext is X.

To map letters to numbers, you might use the `ord` and `chr` functions. You may wish to review how those functions work (Ch. 8.7). However, remember that the function call `ord('A')` converts `'A'` to a number, but the number is *not* 0! The good news is that for the characters A to Z, the numerical values returned by `ord` form a contiguous sequence – i.e., the `ord` value of B is one larger than the `ord` value of A, etc. So, you can use `ord` and `chr` but be sure to "shift" the numbers down to 0-25. Hint: what does `ord('C') - ord('A')` evaluate to? What about `ord('D') - ord('A')`?

3.2 Vigenère cipher

The Vigenere cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword: it consists of several Caesar ciphers in sequence with different shift values. Thus instead of a number for the key, the Vigenere cipher uses a word.

Specifically, the Vigenère cipher works as follows:

1. First, you pick a secret key word or phrase. Let's choose `'KEY'` as our key.
2. Now, let's say that our plaintext is `'TOPSECRETINFO'`. One way to think of the setup of the Vigenère cipher is to line up successive copies of the key with the plaintext, as shown below.

```
 plaintext: TOPSECRETINFO
cipher key: KEYKEYKEYKEYK
```

3. To create the ciphertext, consider each plaintext letter and key letter pair. We will assign numeric values to each letter, such that A corresponds to 0, B corresponds to 1, ... and Z corresponds to 25 (again, remember that `ord('A')` is *not* 0!). If we add the numeric values of each plaintext letter and key letter pair, take that sum modulo 26, converting that number back to a letter will give us the corresponding ciphertext letter. For example: T+K=19+10=29 mod 26=3, thus the ciphertext letter is D. Another example: O+E=14+4=18 mod 26=18, thus the ciphertext letter is S. (Note that the modulus operation simply serves to perform a "wraparound" when you get numbers that are larger than 25 — beyond the letter Z). Thus, the ciphertext for plaintext `'TOPSECRETINFO'` and key `'KEY'` is `'DSNCIABIRSRDY'`.

Another way to think about the Vigenère cipher is that you apply a Caesar cipher to each character but with differing keys. The Caeser key is determined by looking at the corresponding letter in the Vigenère cipher key. The cipher key of `'KEY'` corresponds to Caesar keys `[10, 4, 24]` for K, E, and Y respectively.

You might also wish to read the description on Wikipedia: `http://en.wikipedia.org/wiki/Vigenere_cipher#Description`.

3.3 Deciphering

When a user receives the ciphertext, they must decipher it to read the message.

For Caesar cipher, deciphering is very very easy. Here's a neat trick for that: deciphering is the *same process* as ciphering (which is explained above), you just need to adjust the key. In the example above, the fact that the key is 2 means that each letter was rotated right by 2 places. To decipher, we need to rotate left 2 places. However, rotating left two places is equivalent to rotating right $26 - 2 = 24$ places. So, to decipher we can simply cipher with the key of $26 - 2 = 24$. (Perhaps this analogy might help: when daylight savings ended, most people turned their clocks back one hour; however, I set my clock forward 11 hours. The result is the same.)

Thus, to decipher Caesar ciphered text, *you should not need to write a new function that "deciphers" if you already have a function that takes a key and ciphers*. Instead, just pass a different key to the ciphering function.

For the Vigenère cipher, deciphering is also "easy" but not quite as easy. Deciphering is identical to ciphering, it's just the key is different. To decipher, you might write a helper function that "rotates" the key and then just use this "rotated" key and cipher the text. Again, *you should not need to write a new function that "deciphers" if you already have a function that takes a key and ciphers*.

3.4 Whitespace, punctuation, and lowercase letters

In the descriptions above, we only considered plain text messages that contain only upper case letters – no lower case, punctuation, or white space. In your assignment, your program should be able to handle a mix of **both** upper case and lower case letters. Hint: first check if the letter is upper or lower case and then shift the letter using either `ord('A')` or `ord('a')`.

As for punctuation and whitespace, these characters should **not** be encoded using the cipher. Instead, they should just "pass through" and be written to the output. And for the Vigenère cipher in particular, the non-alphabetical characters should be ignored when it comes to aligning successive copies of the key to plain text. For example, supose the plain text is `'TOP SECRET INFO'`. They key is aligned only with the alphabetical letters.

```
 plaintext: TOP SECRET INFO
cipher key: KEY KEYKEY KEYK
```

The cipher text would be `'DSN CIABIR SRDY'`.

More examples are provided in Section 5.

4 Your task

Your task is write a program that allows the user to either cipher or decipher some text.

Specifically, write a function called `main` that prompts the user to choose between ciphering and deciphering. Then they choose between the Caesar cipher or Vigenère cipher. The next input is the value of the key. For Caesar, this will be a number and for Vigenère, this will be a word. Then ask for the name of an existing file to read that contains plain text and the name of a file into which to write the cipher text. Some examples are given in the next section.

Important: To facilitate testing of your code, it is very important that you ask for user input in *exactly* this order:

1. Choice of cipher or decipher
2. Choice of Caesar or Vigenere
3. Key
4. Input file name
5. Output file name

Of course, you are expected to write additional "helper" functions to support the `main` function. We leave the program design up to you. All functions should have appropriate docstrings.

5 Examples

5.1 Encoding or ciphering – simple examples

Your program should produce output that looks like this:

```
Do you want to (c)ipher or (d)ecipher? c
Which cipher (c)aeser or (v)igenere? c
Enter key: 1
Name of file to read: plain1.txt
Name of file to write: cipher1.txt
```

Since `plain1.txt` just contains one line of text, `'ABCDEFG'`, the cipher text contains this portion of the alphabet shifted by 1. After my program ran, here is what `cipher1.txt` looks like:

```
BCDEFGH
```

A second example, here is `plain1.txt` encoded with the Vigenère cipher and key of `'ABC'`.

```
Do you want to (c)ipher or (d)ecipher? c
Which cipher (c)aeser or (v)igenere? v
Enter key: ABC
Name of file to read: plain1.txt
Name of file to write: cipher1_v.txt
```

After my program ran, here is what `cipher1_v.txt` looks like:

```
ACEDFHG
```

Notice that because the key begins with `'A'` and is three characters long, every third character (A, D, and G) is not shifted at all.

5.2 Encoding – white space and punctuation

If I cipher `plain2.txt` using a Caesar cipher with a key of 25, I get:

```
ZABC defg HIJK
lmno PQRS tuvw
XY
```

If I cipher `plain3.txt` using a Vigenère cipher with a key of `'BAD'`, I get:

```
BA      ec!!     zc??
   fxwsa    vqaffs  !
```

5.3 Encoding – big files

Try out your program on larger files like `alice.txt`. This file starts out like this:

```
Project Gutenberg's Alice's Adventures in Wonderland, by Lewis Carroll

This eBook is for the use of anyone anywhere at no cost and with
```

If I apply the Vigenère cipher with a key of `'ALICE'`, then I get:

```
Pcwlice Owxeyjgvg'd Inmcp'a Chvpvvyrpa kr Wzvfirwiph, bj Tgaid Kcvrztn

Xhta gFozs kw fzz vle fag sf lvasnp ipcwsmti ae vq godb crd hqvl
```

5.4 Decoding or deciphering

If I choose to decode or deciper, I essentially run the process in reverse. (Hint: if you think carefully about the problem, deciphering requires writing only a few lines of code). In fact, if you take any of the examples above and just decipher the ciphered text, you will get back the original plain text. Here is one concrete example:

Recall the example above where I encoded `plain1.txt` with a key of 1 and saved the result to a file called `cipher1.txt`. In this example, I decipher that file.

```
Do you want to (c)ipher or (d)ecipher? d
Which cipher (c)aeser or (v)igenere? c
Enter key: 1
Name of file to read: cipher1.txt
Name of file to write: decipher1.txt
```

The file that is written, `decipher1.txt` is *identical* to `plain1.txt`.

6 Checklist

- So that we can more easily test your code, make sure that you ask the user for inputs in *exactly* the same order as shown in the examples above.
- Program design. The description only specifies the behavior of the program, not what functions you should write. It is up to you to decide how to break this program down into functions. A well-designed program will include appropriate helper functions that take care of some of the low-level details. You are responsible for coming up with a reasonable program design. Be sure to review the SOFA criteria.
- Does each one of your functions have a docstring? If a function does not return value but rather writes to a file, then the docstring should not include examples. Your helper functions should have docstrings as well, possibly with examples if they return a value.