Python Task: Crypto

Confederacy's cipher disk

This task is about cryptography. The task and this writeup are from Harvard's CS50.

Caesar's cipher encrypts (i.e., scrambles in a reversible way) messages by "rotating" each letter by k positions, wrapping around from 'Z' to 'A' as needed:

http://en.wikipedia.org/wiki/Caesar_cipher

In other words, if p is some plaintext (i.e., an unencrypted message), p_i is the i^{th} character in p, and k is a secret key (i.e., a non-negative integer), then each letter, c_i, in the ciphertext, c, is computed as:

$$c_i = (p_i + k) \% 26$$

This formula perhaps makes the cipher seem more complicated than it is, but it's really just a nice way of expressing the algorithm precisely and concisely.

For example, if the secret key is 1, then the cipher text is shifted 1 character from the plain text:

plain 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
cipher 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 A
>>> rotate(13)

So if I wanted to encrypt the message ‘DOG’ I see that ‘D’ should be encrypted as E, ‘O’ should be encrypted as ‘P’ and ‘G’ should be encrypted as ‘H’

For another example, suppose that the secret key, *k*, is 13. The relationship of the plain and cipher characters are

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

If my message, p, is “be sure to drink your Ovaltine.”

Let’s encrypt that p with that k in order to get the ciphertext, c, by rotating each of the letters in p by 13 places:

```
plaintext:  Be sure to drink your Ovaltine!
ciphertext: Or fher gb qevax lbhe Binygvar!
```

We’ve deliberately printed the above in a monospaced font so that all of the letters line up nicely. Notice how O (the first letter in the ciphertext) is 13 letters away from B (the first letter in the plaintext). Similarly is r (the second letter in the ciphertext) 13 letters away from e (the second letter in the plaintext). Meanwhile, f (the third letter in the ciphertext) is 13 letters away from s (the third letter in the plaintext), though we had to wrap around from 'Z' to 'A' to get there. And so on. Not the most secure cipher, to be sure, but fun to implement!

Incidentally, a Caesar cipher with a key of 13 is called ROT13.

Your task is to write two functions, encrypt, and decrypt.

Encrypt
The function encrypt will take two arguments, a string and an integer *k*. The output should be text with each alphabetical character rotated *k* positions.

Although there exist only 26 letters in the English alphabet, you may not assume that *k* will be less than or equal to 26. Your program should work for all non-negative values of *k*. If *k* is greater than 26 you should still have alphabetical characters in the output. For example, if *k* is 27. ‘A’ should not become ‘[‘ even though ‘[‘ is 27 positions away from ‘A’. Instead ‘A’ should become ‘B’ since 27 modulo 26 is 1.

For full XP you program must work with both upper and lower case letters and preserve case: capitalized letters should remain capitals and lower case remain lower case.

Decrypt
The function decrypt will take two arguments, a string and an integer *k.* The string has been encrypted using the key *k*. This function returns the decrypted string.

Submit
Attach the file containing your functions to an email message sent to submit.o.bot_AT_gmail_DOT_com.

Grading
60XP: a encrypt function that encrypts messages in all caps
additional 20XP for handling messages in upper and lower case.
additional 20XP for decrypt function.

Hacker edition: Vigenère's cypher

Well that last cipher was hardly secure. Fortunately, there's a more sophisticated algorithm out there: Vigenère's. It is, of course, French:

http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

Vigenère's cipher improves upon Caesar's by encrypting messages using a sequence of keys (or, put another way, a keyword). In other words, if p is some plaintext and k is a keyword (i.e., an alphabetical string, whereby A and a represent 0, while Z and z represent 25), then each letter, ci, in the ciphertext, c, is computed as:

ci =(pi +kj)%26

Note this cipher's use of kj as opposed to just k. And recall that, if k is shorter than p, then the letters in k must be reused cyclically as many times as it takes to encrypt p.

Here's an example. Suppose my keyword is *dog* and the plaintext I want to encrypt is *poodle.*

First, let's start with *dog*. If 'a' is represented by 0 then *d* is 3, *o* is 14, and *g* is 6.

I will do the same with *poodle* as shown in the first 2 columns of this table:

plain text	ord(ch)	key word	ord key word	ch + keyword % 26	Cypher text
p	15	d	3	18	s
o	14	o	14	2	c
o	14	g	6	20	u
d	3	d	3	6	g
l	11	o	14	25	z
e	4	g	6	10	k

Next I will write repeatedly the keyword next to the plain text (the key word column in the above) and also indicate the character number (the ord key word column)

Now I am going to add the following:

The *p* in *poodle* (character 15) to the *d* in *dog* (character 3) = 18 % 26 = 18
The *o* in *poodle* (character 14) to the *o* in *dog* (character 14) = 28 % 26 = 2
The *o* in *poodle* (character 14) to the *g* in *dog* (character 6) = 20 % 26 = 20
The *d* in *poodle* (character 3) to the *d* in *dog* (character 3) = 6 % 26 = 6
The *l* in *poodle* (character 11) to the *o* in *dog* (character 14) = 25 % 26 = 25
The *e* in *poodle* (character 4) to the *g* in *dog* (character 6) = 10 % 26 = 10

Finally, I convert these numbers back to characters to get the encrypted text: *scugzk*

Your final challenge this week is to write a function that encrypts messages using Vigenère's cipher. This function must take a single argument: a keyword, *keyword*, composed entirely of alphabetical characters.

```
def vignere(keyword):
```

Your program must proceed to prompt the user for a string of plaintext, which it must then encrypt according to Vigenère's cipher with *keyword*, ultimately printing the result. As for the characters in *keyword*, you must treat A and a as 0, B and b as 1, . . . , and Z and z as 25. In addition, your program must only apply Vigenère's cipher to a character in plaintext if that character is a letter. All other characters (numbers, symbols, spaces, punctuation marks, etc.) must be outputted unchanged. Moreover, if your code is about to apply the j th character of keyword to the i th character of plaintext, but the latter proves to be a non---alphabetical character, you must wait to apply that j th character of keyword to the next alphabetical character in plaintext; you must not yet advance to the next character in keyword. Finally, your program must preserve the case of each letter in plaintext. Not sure where to begin? As luck would have it, this program's pretty similar to caesar! Only this time, you need to decide which character in keyword to use as you iterate from character to character in plaintext.

How to test your program, besides predicting what it should output, given some input? Well, recall that I am a nice person. And so I have written a program that is available at

http://rosemary.umw.edu/~raz/vignere

That page also allows you to play with my own implementation of vigenere.