Assignment Week 9: Encryption Algorithm

Application

In this assignment you will iterate strings, shift letters using ASCII (or Unicode) values, use modulus, combine strings, and use string formatting.

Background

A cipher is used to encrypt data so that is hard to know what the original message was. Julius Caesar was famous for more than his slushes, salads, and being stabbed. He also has a cipher named after him.

In a Caesar cipher, each letter is assigned an integer value as shown in the following table:

Letter	a	b	c	d	e	f	g	h	i
Value	0	1	2	3	4	5	6	7	8

Letter	j	k	l	m	n	o	p	q	r
Value	9	10	11	12	13	14	15	16	17

Letter	s	t	u	v	w	x	y	z
Value	18	19	20	21	22	23	24	25

A Caesar cipher is a simple substitution cipher wherein each letter in the message is shifted a certain number of spaces down the alphabet. The number to shift is called the key.

letter	**key shift**	**becomes**
a	1	b
c	3	f
a	13	n

If the shift runs off the end of the alphabet, it will wrap back around from the beginning.

letter	**key shift**	**becomes**
z	1	a
z	3	c
x	1	y
x	3	a
n	13	a

To apply this cipher to a text, you simply applying the shift to each letter in the text. Spaces are ignored.

text	**key shift**	**becomes**
abc	1	bcd
sos	3	vrv
hope	5	mtuj
happy day	3	kdssb gdb
not the mamma	13	abg gur znzzn

You can reverse (decrypt) the cipher if you just subtract the shift instead of adding it.

You can also shift by the difference between the key and the number of letters (26).

text	key unshift	becomes
bcd	1	abc
vrv	3	sos
mtuj	5	hope
kdssb gdb	3	happy day
abg gur znzzn	13	not the mamma

The problem with a Caesar cipher is that is it predictable. It is better to use a rolling cipher. A rolling cipher is just like a Caesar cipher, except that the key or shift changes for each letter in a predictable pattern. So, for example, if the roll was to just add one to the key for each letter, we would get the following results:

text	shift (start)	Caesar	Rolling (+1)
aaa		bbb	bcd
abc	1	bcd	bdf
sos	3	vrv	vsx
hope	5	mtuj	muwm
happy day	3	kdssb gdb	keuvf mkj
not the mamma	13	abg gur znzzn	aci kzx hwjkz

This is a simple rolling cipher, there are more powerful ones, as long at the pattern of the numbers remains consistent, it will work.

Instructions

For this assignment you will create a module and a script which will use the functions in your module.

You will create a Caesar cipher and a rolling cipher. Your solution should be able to encrypt all letters. Any character that is not a letter, will not be changed. In other words, numbers and spaces will not be shifted.

For the Caesar cipher, you will create a basic shifting cipher. We call these ROT ciphers. ROT is short for ROTating. A ROT-1 cipher shifts all letters 1 location a to b, b to c, etc. A ROT-13 cipher, shifts all letters 13, a to n, b to o, etc.

For the rolling cipher, you will use random numbers to create the rolling key. Remember in computers random numbers are pseudo random, which means that if we specify the starting point (seed), the same numbers will always be used. This will be our pattern for the rolling cipher.

Both of these approaches will be able to use the same code (function) to get the shifted letter.

Create a Python module:

1. Create a script named crypt_maker.py

2. Create a function named **shift_letter**
    a. It accepts a single letter as string and an integer value to shift.
    b. The function will shift the single letter based on the value of the shift.
        i. Make sure that if you run off the end ("z") you wrap back around to "a".
        ii. If you convert the ASCII value to start at zero (like in the table in the Background) a modulus can make this easier.
            1. Don't forget to move it back to the appropriate range if you use this approach.
    c. You will need handle for upper case letters.
        i. Hint: solve for lower case first and then adapt to handle upper case.
        ii. You are <u>not allowed</u> to use the isupper() method.
            1. Use a string conditional. Hint: < or > than what?
    d. Return the shifted letter.
    e. Examples for testing in the table ->

letter	shift	returns
a	1	b
c	1	d
z	1	a
z	3	c
a	13	n
n	13	a
A	1	B
C	1	D
Z	1	A
Z	3	C
A	13	N

3. Create a function named **caesar**
    a. It accepts a string of text and an integer key.
    b. For each letter in the string, call the shift_letter function to get a new letter based on the key.
        i. Make sure to only do letters.
            1. You are <u>not allowed</u> to use the isalpha() method.
                a. Use a string conditional. Hint: < or > than what?
        ii. If not a letter. Use the original value.
        iii. Hint: first, make sure you can get each letter and put them back together to make the original string before you shift the values.
    c. Return a string with the encoded (shifted) result.
        i. You can use the table in the **Testing** section, to validate your results.

4. Create a function named **caesar_back**
    a. It accepts a string of text and a key as string and int respectively.
    b. Reversing a Caesar cipher is really easy, just subtract the key from 26 and use this value as the new key.
    c. Call the caesar function with the text and the new key and return the result.
        i. There is no need to change the caesar function.
    d. You can use the table in the Testing section, to validate your results.

5. Create a function named **encrypt**
    a. It accepts a string of text and a key as string and int respectively.
    b. It will basically do the same thing as the function caesar. However, this will implement a rolling cipher, **but not like** the example in the Background section.
    c. The key provided is the starting seed value for a random number between 0 and 25 (inclusive). This random number will be the shift and will change for each letter.
        i. Remember that if you use the same seed, you will always get the same number sequence.
            1. You can use this table to make sure you are setting your seed correctly:

Seed	Values (0-25)
1	4, 18, 25, 24, 2, 8, 3, 15, 24, 14
4	7, 9, 3, 23, 12, 15, 4, 2, 2, 0
13	8, 9, 21, 21, 25, 5, 20, 7, 21, 4

    d. For each letter in the string, call the shift_letter function to get the value based on the rolling shift which will be the next random number in the sequence.
        i. This means that if the seed was 1, the first shift would be 4, then 18 after that, then 25, then 24, and so on and so forth until you reach the end of the text.
    e. Return a string with the encoded (shifted) result.

6. Create a function named **decrypt**
    a. It accepts a string of text and a key as string and int respectively and returns a string.
    b. It will basically do the same thing as encrypt.
        i. The only difference is that you get the key for each letter by subtracting the rolling key from 26 before you call the letter_shift function.

Create a Python script:

1. Create a script named cipher.py
2. In the script, prompt the user for the text to encrypt.
3. Prompt the user for an integer-based key.
4. Use your module to get both the Caesar and Rolling cipher results.
5. Then use your module to decrypt each of the results from the last step.
6. Display all of these results to the user nicely formatted.
    a. You must use the string format method.

```
"format string".format(a1,a2,…)
```

    b. Your results do not have to look like mine, but you must use the format method off of the string object.
    c. You must provide at least two arguments to be formatted.
    d. This is the format that uses curly braces { }

Output

The output should easily readable. As with every other assignment, your output does not need to look exactly like mine. I encourage you to be creative and descriptive in your output. Here is some example output:

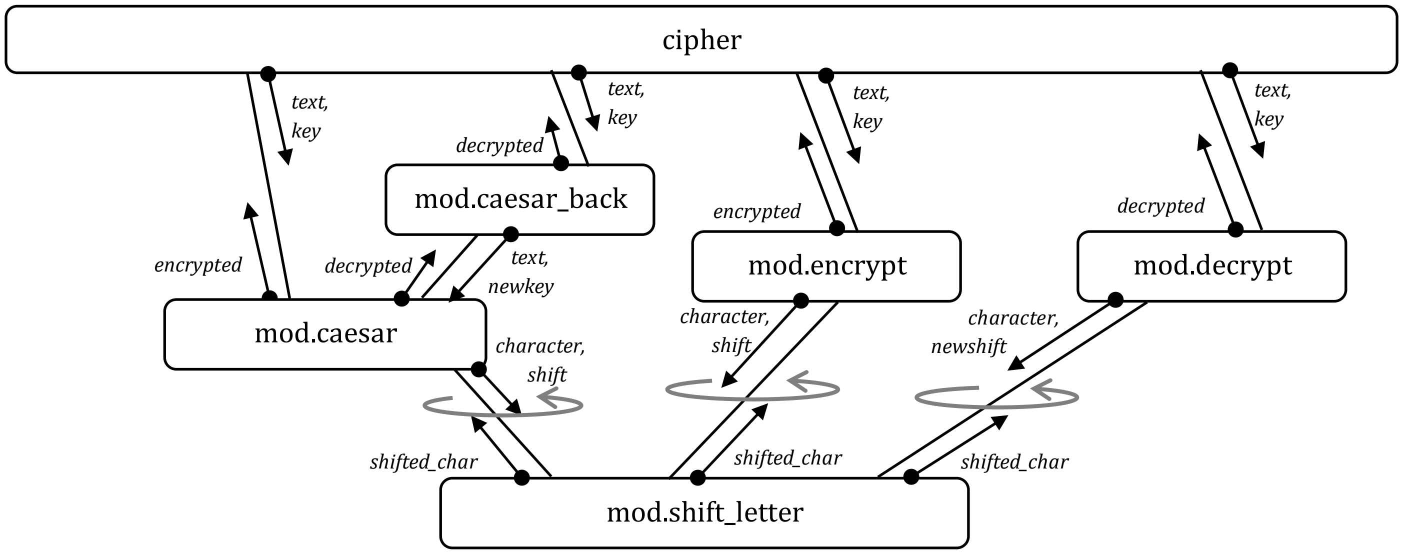
```
What do you wish to encrypt? Rabbits
What is the key? 4

  Caesar   |  Rolling
 --------  |  --------
 Veffmxw   |  Yjeyuiw
 --------  |  --------
 Rabbits   |  Rabbits
```

```
What do you wish to encrypt? How did I get 14 rabbits?
What is the key? 5

         Caesar           |          Rolling
 ------------------------ |  ------------------------
 Mtb ini N ljy 14 wfggnyx? |  Awt ohz F aut 14 fyivjyv?
 ------------------------ |  ------------------------
 How did I get 14 rabbits? |  How did I get 14 rabbits?
```

Structure chart



Testing

This table has example outputs:

function	caesar			encrypt		
key	1	4	13	1	4	13
aaa	bbb	eee	nnn	esz	hjd	ijv
bbb	ccc	fff	ooo	fta	ike	jkw
zzz	aaa	ddd	mmm	dry	gic	hiu
xxx	yyy	bbb	kkk	bpw	ega	fgs
abc	bcd	efg	nop	etb	hkf	ikx
mom	npn	qsq	zbz	qgl	txp	uxh
boo	cpp	fss	obb	fgn	ixr	jxj
New day.	Ofx ebz.	Ria hec.	Arj qnl.	Rwv bcg.	Unz amn.	Vnr yzd.
Hi, Fred:	Ij, Gsfe:	Lm, Jvih:	Uv, Serq:	La, Epgl:	Or, Ioqs:	Pr, Amdi:

Hints

Remember, code a little and test. Always keep a working program.

Scoring

1. 5% - Compiles without errors
2. 14% - shift_letter (type hinting, correct parameters)
    a. get ascii value of letter, adds shift to letter, converts ascii back to letter, returns correct string, handles upper and lower case
3. 12% - caesar (type hinting, correct parameters)
    a. iterates for each letter, only shifts letters (no is alpha), calls shift_letter, creates string, returns correct string
4. 9% - caesar_back (type hinting, correct parameters)
    a. iterates for each letter, creates new shift correctly, calls caesar, returns correct string
5. 18% - encrypt (type hinting, correct parameters)
    a. iterates for each letter, uses seed correctly, uses random to get next key, only shifts letters (no is alpha), calls shift_letter, creates string, returns correct string
6. 20% - decrypt (type hinting, correct parameters)
    a. iterates for each letter, uses seed correctly, uses random to get next key, creates new shift correctly, only shifts letters (no is alpha), calls shift_letter, creates string, returns correct string
7. 2% - prompts user for phrase to encrypt
8. 2% - prompts user for key to use
9. 2% - calls caesar properly
10. 2% - calls encrypt properly
11. 2% - calls caesar_back properly
12. 2% - calls decrype properly
13. 2% - displays results for encryption and decryption
14. 2% - output is readable and nice looking
15. 5% - Meaningful comments (header block, doc_strings, comments)