ICS Fall 2021
Assignment 2

Please note: For some questions, starting codes are available. Please fill the blanks in starting codes. For questions without starting codes, you can write the solutions in any form you like. Since we use an auto-grader to grade your code, please strictly name the functions and the submitted files as required by the questions. Moreover, when you submit your answers, you should zip the files, not the whole folder of your answers. Lastly, the starting codes may contain some tests for you to check if your code is correct.

1. **Length of the last word**
   Write a function `lenLastWord` that one argument which is a string `s` consisting of letters, empty spaces, and punctuations (the punctuations are !()-[]{}::'"\,<>./?@#$%^&*_~), print the length of the last word in the string. Put your code in `Q1_lenLastWord.py`.

   Examples:

Input	`"hello world."`	`"Good morning  !"`
Output	`5`	`7`

   Hint:
   - You may import a package named `string`, and call `string.punctuation` to get all the punctuation.
   - You may use a string method, named, `.replace()`

2. **Number of occurrences**
   Write a function named "`fun1`" that takes two arguments, a character c and a string s, which returns the number of occurrences of c in s. Put your code in `Q2_numOfOccur.py`.

   Examples:

function call	`fun1('l', 'hello')`	`fun1('z', 'hello')`
return	`2`	`0`

3. **Integer to the Roman letter**

Write a function named `convert_to_Roman_numeral` that accepts an integer between 1 and 3999, convert it into a Roman numeral, and return the Roman number. Here is the conversion table:

'I'	'V'	'X'	'L'	'C'	'D'	'M'
1	5	10	50	100	500	1000

We take `4 as IV, 9 as IX, 40 as XL, 90 as XC, 400 as CD, 900 as CM`, and so on.

Specifically, the numbers from 1 to 10 are expressed as Roman numerals as follows:
`I, II, III, IV, V, VI, VII, VIII, IX, X.`

The system being basically decimal, tens, and hundreds follow the same pattern. Thus 10 to 100 (counting in tens, with X taking the place of I, L taking the place of V and C taking the place of X):
`X, XX, XXX, XL, L, LX, LXX, LXXX, XC, C.`

Similarly, 100 to 1000 (counting in hundreds):
`C, CC, CCC, CD, D, DC, DCC, DCCC, CM, M.`

Examples:

input	1776	1949	3999
output	MDCCLXXVI	MCMXLIX	MMMCMXCIX

Put your code in `Q3_int2Rome.py`.

Hint: What data structure you will use to connect Roman and Arabic, the list or the dict? Which one can make your code shorter?

4. **Matrix Mirror**

   Write a function named "`fun1`", which takes an `NxN` square of numbers, i.e., a 2-D list, and flip it horizontally **in place**. Put your code in `Q4_matrixMirror.py`.

   Examples:

`m is assigned as`	`[[1, 2],`
	`[3, 4]]`
		`[[1, 2, 3],`
		`[4, 5, 6],`
		`[7, 8, 9]]`
			`[[1, 2, 3, 4],`
			`[5, 6, 7, 8],`
			`[9, 10, 11, 12],`
			`[13, 14, 15, 16]]`

`After calling func2(m)`			

`m becomes`	`[[2, 1],`
	`[4, 3]]`
		`[[3, 2, 1],`
		`[6, 5, 4],`
		`[9, 8, 7],]`
			`[[4, 3, 2, 1],`
			`[8, 7, 6, 5],`
			`[12, 11, 10, 9],`
			`[16, 15, 14, 13]]`

```
input m: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
after running fun1: [[3, 2, 1], [6, 5, 4], [9, 8, 7]]
input m: [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
after running fun1: [[4, 3, 2, 1], [8, 7, 6, 5], [12, 11, 10, 9], [16, 15, 14, 13]]
```

5. **Encrypting the contents**

Julius Caesar protected his confidential information by encrypting it using a cipher. Caesar's cipher shifts each letter by a number of letters. If the shift takes you past the end of the alphabet list, just rotate back to the front of the list. In the case of a rotation by 3, w, x, y, and z would map to z, a, b, and c.

```
Original alphabet:      abcdefghijklmnopqrstuvwxyz
Alphabet rotated +3:    defghijklmnopqrstuvwxyzabc
```

You need to encrypt and decrypt a message using Caesar's cipher. But this time, you need to use an alphabet list (i.e., a codebook) which is a mixture of uppercase and lowercase, and the alphabets are shuffled, not in the alphabet order. The following figure shows an example of the codebook, which is a list.

```
Your codebook:
['t', 'O', 'v', 'A', 'N', 'u', 'L', 'r', 'T', 'E', 'Q', 's', 'Y', 'j', 'P', 'X'
'a', 'g', 'l', 'b', 'J', 'I', 'd', 'p', 'q', 'z', 'n', 'x', 'R', 'C', 'B', 'M',
'Z', 'e', 'G', 'S', 'K', 'i', 'y', 'w', 'o', 'h', 'f', 'V', 'k', 'F', 'H', 'm',
'U', 'D', 'W', 'c']
```

1) Write a function name `caesarEncrypt()` which takes three arguments: `message, codebook, shift`, where `message` is a string, `codebook` is a list of alphabets, and `shift` is a positive integer. The function returns an encrypted message.
2) Write a function name `caesarDecrypt()` which takes three arguments: `message, codebook, shift`. The function returns a decrypted message.

Put your code in `Q5_CaesarEncryption.py`.

Notes:

- The cipher only encrypts letters; symbols remain unchanged.
- The encode/decode of a character can be done by finding and computing its indices, but also, you may use the Python dictionary to simplify this process.

When you run the tests in the starting code, you will get the following output. (i.e., the message = "Hello Kitty!", codebook=the codebook given in the starting code, shift=3)

```
Origin: Hello Kitty!
Encoded: DKIIV woAAh!
Decoded Hello Kitty!
```