Cipher Programming 101

In this assignment, you should implement several famous ciphers. For each cipher, you should implement encryption and decryption procedures.

HINT: Some functions, which may be useful: chr(),ord(),isalpha(),lower(),upper(),islower(),isupper().

HINT: The difficulty tags written above the functions in the template MAY be useful.

HINT: Useful links: https://www.w3schools.com/python/python_ref_string.asp, https://www.programiz.com/python-programming/methods/list, https://www.programiz.com/python-programming/methods/dictionary

Ciphers

Atbash Cipher

Atbash Cipher is one of the oldest encryption code, originally used to encrypt the Hebrew alphabet. Examples of texts encrypted in Atbash can be found in ancient texts and in the bible. The idea is to take the alphabet and map it to its reverse, so that the first letter becomes the last letter, the second letter becomes the second to last letter, and so on. For example, the Latin alphabet would look like this:

```
Plain:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
Atbash: ZYXWVUTSRQPONMLKJIHGFEDCBA
```

Write `encryptAtbashCipher` function which takes plane text as an argument and encrypts it: changes all letters according to above mentioned mapping. For example, 'a' should be changed with 'z', 'b' with 'y' and so on. For simplicity, we can assume that this cipher is used to encrypt text which contains **only** lower case english letters. You do **not** need to check input text, assume that it follows this rule.

After encryption, secret text should be decrypted to obtain initial information. Write `decryptAtbashCipher` function, which takes encrypted text in Atbash and converts it to plain text. Decryption procedure is similar to encryption, to obtain plain text 'z' should be changed with 'a', 'y' with 'b' and so on.

Example:

```
Plain Text:     "atbash"
Encrypted text: "zgyzhs"
Decrypted text: "atbash"
```

```
Plain Text:     "programming"
Encrypted text: "kiltiznnrmt"
Decrypted text: "programming"
```

Caesar Cipher

The next cipher you should implement is the Caesar cipher. Caesar cipher is a shift cipher, one of the easiest and most famous encryption systems (and really old, Yes, Julius Caesar used it).

Encryption with Caesar code is based on an alphabet shift, it is a monoalphabetic substitution cipher like Atbash, ie. the same letter is replaced with only one other letter.

For example, if the shift is 2 then 'a' is changed to 'c', 'b' to 'd', and so on. In the end, 'x' is changed to 'z', 'y' to 'a' and 'z' to 'b'.

There are 26 characters in the alphabet, but the shifting key might be greater than 26. For example, if we shift 'a' by 26 it will stay the same after one full cycle and if we shift it by 29 it will become 'd'.

Example:

```
Alphabet:                ABCDEFGHIJKLMNOPQRSTUVWXYZ
Alphabet shifted by 26: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Alphabet shifted by 3:  DEFGHIJKLMNOPQRSTUVWXYZABC
Alphabet shifted by 29: DEFGHIJKLMNOPQRSTUVWXYZABC
```

Write `encryptCaesarCipher` function, which is given the plain text and shifting key. The function should return encrypted text based on the Caesar cipher and given shifting key. You also should write `decryptCaesarCipher`, a function that should decrypt given ciphered text. To decrypt the text you should reverse the shifting process of the encryption algorithm, ie. shift the alphabet in the opposite direction.

Caesar cipher only changes letters, but as numbers and punctuation symbols might also give away some information, you should change them too, based on the given mapping:

```
    '.' :-> ','
    ',' :-> '.'
    '!' :-> '?'
    '?' :-> '!'
    '0' :-> '1'
    '1' :-> '0'
    '2' :-> '3'
    '3' :-> '2'
    '4' :-> '5'
    '5' :-> '4'
    '6' :-> '7'
    '7' :-> '6'
    '8' :-> '9'
    '9' :-> '8'
```

HINT: You can use dictionary to store this mapping and if character is not letter but it is in the dictionary get mapping from the dictionary... or you can use multiple ifs, but dictionaries are cool!

Example:

```
Key:            3
Plain Text:     "Cipher Programming - 101!"
Encrypted text: "Flskhu Surjudpplqj - 010?"
Decrypted text: "Cipher Programming - 101!"
```

NOTES:

- Upper case letters should remain upper case and lower case letters should remain lower case.
- Letters should be modified based on the Caesar algorithm, but numbers and special symbols should be modified based on the given mapping.
- If the symbol is not letter and is not included in the given mapping it should stay unchanged. Ex. ' ' -> ' ', '-' -> '-' and so on.

Vigenère Cipher

Vigenere cipher is an extended version of Caesar cipher. Characters are modified in exactly same way. But, instead of one key, the Vigenere cipher has several keys. These keys are used one by one to shift characters.

For example, if keys are [1,7,3], than 1 is used to shift characters at indexes: 1,4,7,10,13..; 7 is used to shift characters at indexes: 2,5,8,11,14..; and 3 is used to shift characters at indexes: 3,6,9,12,15..; In other words i-th symbol is encrypted using key - key_list[i % key_num].

Implement `encryptVigenereCipher` and `decryptVigenereCipher` functions. Numbers and special symbols are handled in the same way as in Caesar cipher.

Example:

```
Key:            [1,3,2]
Plain Text:     "Cipher Programming - 101!"
Encrypted text: "Dlriht Stpjtbpojqi - 010?"
Decrypted text: "Cipher Programming - 101!"
```

Simplified Enigma

Enigma is the famous cipher machine that was created and used in the early to mid-20th century. It was extensively used by the German military in world war II. It was considered unbreakable, but British and polish scientists managed to break it, which determined the outcome of WW2 (btw, "The Imitation Game" a great movie about breaking the enigma and Alan Turing, the pioneer of CS).

In this last task, you should implement `encryptSimpleEnigmaCipher` and `decryptSimpleEnigmaCipher` functions, which are based on the simplified version of enigma. Enigma machine had 3 rotors, which we are going to simulate with 3 step encryption. We are going to have 3 key, where each of them represents the state of each rotor. Keys are permutations of alphabet, which are used as mappings like in the previous permutation cipher.

Let's say we have 3 keys:

```
key1 : "bcdefghijklmnopqrstuvwxyza"
key2 : "qwertyuioplkjhgfdsazxcvbnm"
key3 : "zxcvbnmlkjhgfdsaqwertyuiop"
```

and we want to decipher character 'c'. 'c' is the 3rd character in the English alphabet, so it will be changed to 3rd character from the first key, which is 'd'. This is the first step of encryption. The second one is the same, but now we use the second key, so 'd' will be transformed to 'r'. After that, comes the last, 3rd step, where we use the 3rd key, which changes 'r' to 'w'. At the end, we get 'c' -> 'd' -> 'r' -> 'w'.

We encrypt each character of the text in the same way. Upper case characters are changed identically: 'C' -> 'D' -> 'R' -> 'W'. For the simplicity of the problem, we ignore special symbols and numbers and leave them unchanged.

NOTE: Keys are passed as a tuple: (key1,key2,key3).

Example:

```
Key:            ('bcdefghijklmnopqrstuvwxyza',
                 'qwertyuioplkjhgfdsazxcvbnm',
                 'zxcvbnmlkjhgfdsaqwertyuiop')
Plain Text:     "Cipher Programming - 101!"
Encrypted text: "Wavsoz Vznkzullamk - 101!"
Decrypted text: "Cipher Programming - 101!"
```

Grading

This assignment is worth 10 points. You have to implement 4 ciphers, with both encryption and decryption functions. Each cipher is worth 2.5 points: 1.5 points if you write only encryption function and 2.5 points if you write both encryption and decryption functions.

- If your WHOLE code, does not compile it will be graded with 0. (Even if you have a small typo, so be careful and RUN the code).
- Each cipher is graded separately. You get 0, 1.5 or 2.5 points in each.
- Anything unrelated to the solution should be removed from the code, before submitting it.

You can use examples in the statements to test your programs. It is strongly recommended to come up with other input texts and keys too. If your code works for the examples, it does not mean that your code is 100% correct. During grading other inputs will be used too for testing the programs, so manual testing is highly recommended.

All functions are based on the real ciphers, but the encryption algorithms have been modified in some way to adjust this assignment. Therefore, do **not** rely on instructions and solutions from the web. If functions are not implemented as stated in the task description, they will not be accepted. **Submissions will be checked against plagiarism.** If even one function is suspected of plagiarism, the whole assignment will be annulled.