Cipher Programming 101

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

For the first 3 tasks (Affine, Index Substitution ciphers, and Morse code) assume that input contains only lower case English letters, no numbers, upper case letters, spaces, or any other special symbols. For the last 2 (Caesar and Transposition ciphers) input text can contain any symbol.

HINT: Tasks are not ordered based on their difficulty. If you find one task hard to write, move to the next one, it might be easier.

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

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

Index Substitution Cipher

The first cipher is Index Substitution Cipher. This is a relatively simple cipher: Each character is replaced with its 2 digit code. 'a' is replaced with '01', 'b' with '02', 'z' with '26', etc. The Code of each character is separated into a single space.

Write `encryptIndexSubstitutionCipher` function, which is given the plain text. The function should return encrypted text based on the index substitution algorithm described above. You also should write `decryptIndexSubstitutionCipher`, a function that should decrypt given ciphered text. To decrypt the text you should reverse the above algorithm and replace each 2 digit code with an appropriate character.

Example:

```
Plain text:     'cipher'
Encrypted text: '03 09 16 08 05 18'
Decrypted text: 'cipher'
```

Morse Code

Morse code is a character-encoding scheme that allows operators to send messages using a series of electrical pulses represented as short or long pulses, dots, and dashes. It was used extensively in the XIX century. Even though it is not popular today, it still can be a fun cipher to implement.

In Morse code each character is associated with sequence of dots and underscores:

```
    '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': '__..'
```

To encrypt text, each character should be changed with an appropriate sequence and each such sequence should be separated from each other with a single space.

Write `encryptMorseCode` function, which is given the plain text. The function should return encrypted text based on the morse code. You also should write `decryptMorseCode`, a function that decrypts given ciphered text. To decrypt the text you should reverse the above algorithm and replace each sequence with an appropriate character.

Example:

```
Plain text:     'morse'
Encrypted text: '__ ___ ._. ... .'
Decrypted text: 'morse'
```

Affine Cipher

The next cipher you should implement is called the Affine Cipher. The Affine cipher is a type of monoalphabetic substitution cipher that uses a simple mathematical function to encrypt given letters. Each letter is changed with exactly one new letter.

The Affine function uses 2 coefficients (or keys): A and B. At first, the character is converted to its order in the alphabet (starting with 0), then the Affine function is applied to this order and the result is converted back to the character based on the same alphabet order.

Encription function for Affine cipher is - `e(x) = (a*x + b) % 26`.

Decription function for Affine cipher is - `d(x) = (pow(a,-1,26) * (x - b)) % 26`. `pow(a,-1,26)` denotes a in power of -1 by module 26, `pow()` is a function in python, so you can use this formula directly in your program.

For example: Let's say we want to encrypt 'c', with A equal to 11 and B equal to 9.

1. At first, convert character to it's order in alphabet: 'c' -> 2. (Do not forget ordering start from 0.)
2. Apply encription function to it: `e(2) = (11*2 + 9) % 26 = 5`.
3. Use the result as an order to get character from alphabet. 5 -> 'f'.

Finally, character 'c' will be changed with 'f' when A is 11 and B is 9.

Note: Affine Cipher does not work for all A coefficients. To decript the text A needs to be invertible module 26. You can assume that A provided to function always will be invertible.

Note: Some valid A coefficients that can be used for testing: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21...
`pow` function might fail for other numbers, but it is not a problem.

Write `encryptAffineCipher` function, which is given the plain text and 2 keys A and B. The function should return encrypted text based on the Affine cipher algorithm. You also should write `decryptAffineCipher`, a function that should decrypt given ciphered text. To decrypt the text you should use a decryption formula instead of an encryption one.

Example:

```
A: 11
B: 5
Plain text:     'affine'
Encrypted text: 'fiipsx'
Decrypted text: 'affine'
```

Caesar Cipher

The next cipher is the Caesar cipher. Caesar cipher is a shift cipher, one of the easiest and the 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, 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
```

Caesar cipher only changes letters, but as numbers might also give away some information, you should change them too with the same shifting algorithm:

```
Digits:               0123456789
Digits shifted by 2:  2345678901
Digits shifted by 12: 2345678901
Digits shifted by 10: 0123456789
```

All other symbols (i.e. space, !, ?, . and etc.) should stay unchanged. Upper case letters should be encrypted with upper case letters and lower case with lower case (i.e. if 'a' shifts to 'g', 'A' should shift to 'G').

Various modifications of Caesar cipher have been invented over time to make cracking it harder. One of such modifications was called Double-key Caesar Cipher. In this version, the cipher has two keys instead of one. 1st symbol is encrypted with 1st key, 2nd symbol with 2nd key, 3rd symbol with 1st key again, 4ths symbol with 2nd key, and so on.

For example, if the text is "Abc-01!" and keys are 2 and 3, encryption would look like this:

```
TEXT := Abc-01!

SHIFT(A, 2) -> C
SHIFT(b, 3) -> e
SHIFT(c, 2) -> e
SHIFT(-, 3) -> -
SHIFT(0, 2) -> 2
SHIFT(1, 3) -> 4
SHIFT(!, 2) -> !

RESULT := Cee-24!
```

Write `encryptCaesarCipher` function, which is given the plain text and 2 shifting keys. 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.

Example:

```
1st Key:        3
2nd Key:        2
Plain Text:     "Cipher Programming - 101!"
Encrypted text: "Fksjht Ruqjtdopkqi - 333!"
Decrypted text: "Cipher Programming - 101!"
```

Transposition Cipher

Transposition Cipher is different from the previous ciphers. Instead of changing characters to different characters, in this cipher, characters of the initial text are rearranged. You should implement this procedure in `encryptTranspositionCipher` and `decryptTranspositionCipher` functions.

We are given the key, which determines the length of the chunk. We divide the text into such chunks and write them to bellow each other (in the shape of the rectangle). After that, we are going to rearrange the text by reading columns from up to down and from left to right.

For example, if we want to encrypt "Cipher Programming - 101!" and the key is 6 we get chunks: "Cipher", " Progr" (Do not skip space in the beginning!), "amming", "!" (The last chunk may be smaller in size).

If we write them in the form of a rectangle it will look like this:

```
'Cipher'
' Progr'
'amming'
' - 101'
'!'
```

Than we read each column from top to bottom to get chunks: "C a !", "iPm-", "prm ", "hoi1", "egn0", "rrg1". Notice that the first chunk is longer than others. After concatenating them togather we get encrypted text: "C a !iPm-prm hoi1egn0rrg1".

To decrypt the text, you should reverse the encryption algorithm. At first, take the text and create the table:

```
'C a !'
'iPm-'
'prm '
'hoi1'
'egn0'
'rrg1'
```

Be careful, calculating the size of the table may be tricky. After you had done this hard part, you can restore the text by reading columns in the same way as when encrypting.

Example:

```
Key:            3
Plain Text:     "Cipher Programming - 101!"
Encrypted text: "Ch oai 1!iePgmn-0prrrmg 1"
Decrypted text: "Cipher Programming - 101!"
```

Grading

This assignment is worth 10 points. You have to implement 5 ciphers, with both encryption and decryption functions. In total, there are 10 functions, each of them worth 1 point.

- 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). Run `ciphers.py` file with `python` or `python3` command to check it.
- You do not get partial scores for each subtask, so one function is worth only 0 or 1 point.
- 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. You might be asked to explain your code during a live meeting or Teams call with TA. There are no fixed dates for the defense, it can be scheduled anytime after the deadline, but before the final exam. The initial grade will be modified based on the defense.