CANDIDATE'S FULL NAME: Daniel Mula Tarancón

The Caesar cipher is an ancient encryption algorithm used by Julius Caesar. It encrypts letters by shifting them over by a certain number of places in the alphabet. We call the length of shift the key. For example, if the key is 3, then A becomes D, B becomes E, C becomes F, and so on. To decrypt the message, you must shift the encrypted letters in the opposite direction. This program lets the user encrypt and decrypt messages according to this algorithm.

When you run the code, the output will look like this:

Do you want to (e)ncrypt or (d)ecrypt?

> e

Please enter the key (0 to 25) to use.

> 4

Enter the message to encrypt.

> Meet me by the rose bushes tonight.

QIIX QI FC XLI VSWI FYWLIW XSRMKLS.

Do you want to (e)ncrypt or (d)ecrypt?

> d

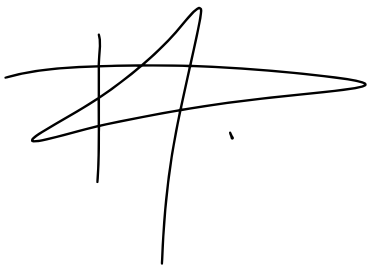
Please enter the key (0 to 26) to use.

> 4

Enter the message to decrypt.

> QIIX QI FC XLI VSWI FYWLIW XSRMKLX.

MEET ME BY THE ROSE BUSHES TONIGHT.



INDEX

1. Python libraries used [3]
2. Script structure [6]
3. Getting into details [7]
3. Conclusions [12]

1. Python libraries used

These are the libraries and modules I used to accomplish this task:

A) import string

The 'string' module provides a number of useful constants and functions for working with strings. Some of the key features of the 'string' module include:

Constants for working with ASCII and Unicode character sets, such as 'string.ascii_letters', 'string.digits', 'string.whitespace', and 'string.printable'.

Convenience functions for formatting and manipulating strings, such as 'string.capwords()' , 'string.Template()' and 'string.Formatter()'.

Functions for working with case and and whitespace in strings, such as 'string.upper()', 'string.lower()', 'string.tittle()', 'string.strip()', 'string.lstrip()', and 'string.rstrip()'.

Functions for testing and validating strings, such as 'string.startswith()', 'string.endswith()', 'string.isalpha()', 'string.isdigit()', 'string.isnumeric()', and 'string.isprintable()'.

Overall, the 'string' module is a powerful tool for working with text and strings in Python. It provides a wide range of functionality for manipulating, formatting, and testing strings, and can be a useful resource for many different types of programming tasks.

In this case I used for getting the alphabet, using the string.ascii_letters method.

B) import datetime

The 'datetime' module in Python provides classes for working with dates, times, and durations. Some of the key features of the 'datetime' module include:

Classes for working with dates and times, such as 'datetime.date', 'datetime.time', and 'datetime.datetime'. These classes allow you to create and manipulate dates and

times, extract information like the year or month, and perform arithmetic with other dates and times.

Functions for formatting and parsing dates and times, such as 'datetime.datetime.strftime() and datetime.datetime.strptime(). These functions allow you to convert between Python 'datetime' objects and human-readable string representations of dates and times.

Support for time zones and daylight saving time through the 'datetime.timezone' and 'datetime.tzinfo' classes. These classes allow you to represent and work with times in different time zones, and to handle the transition to and from daylight saving time.

Functions for performing arithmetic with dates and times, such as 'datetime.timedelta()'. These functions allow you to add or subtract a duration from a 'datetime' object, or to calculate the difference between two 'datetime' objects.

Other useful features, such as the 'datetime.datetime.now()' function, which returns the current date and time, and the 'datetime.date.today()' function, which returns the current date.

Overall, the 'datetime' module is a powerful tool for working with dates, times, and durations in Python. It provides a wide range of functionality for creating, manipulating, and formatting dates and times, and can be a useful resource for many different types of programming tasks.

In this script I used this library so the program is able to greeting the user in a proper way.

C) import sys

The 'sys' library in Python provides access to some variables used or maintained by the interpreter, as well as functions that interact strongly with the interpreter. Here are some of the commonly used features of the 'sys' library:

sys.argv —> This is a list in Python, which contains the command-line arguments passed to the script. With this, you can easily access the arguments passed to a Python script.

'sys.exit()' —> This function is used to exit the Python interpreter. If you pass an argument to this function, it will be used as the exit status.

'sys.path' —> This is a list of strings that specifies the search path for modules. Python looks for modules in these directories, in the order they are listed.

sys.platform —> This string contains the name of the platform on which the Python interpreter is running.

sys.stdin, sys.stdout and sys.stderr —> These are standard I/O streams provided by the interpreter. You can use these streams to read input from the user, write output to the console, or print error messages.

sys.version: This string contains the version number of the Python interpreter.

sys.modules: This is a dictionary that maps module objects. You can use this dictionary to get information about loaded modules.

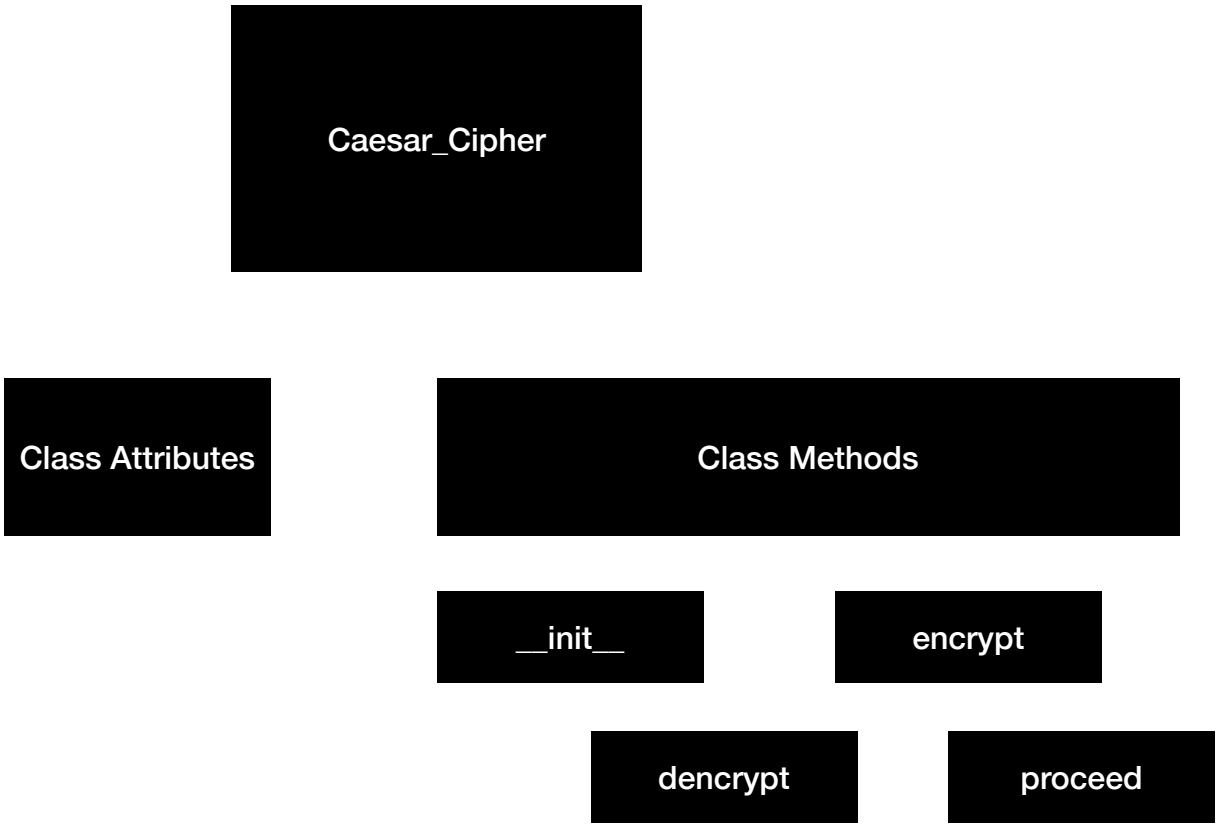
These are the its most important modules.

In this case I used the sys library so I could exit the program.

2. Script structure

In order to accomplish this challenge I decided to create a Class called Caesar_Cipher. Since I see this encryptor as an object with a structure given by this class. Different instance of my class will vary according to the key I use for perform these type of encryption. But since the encryption is the same, I thought that creating a Class will be the suitable way of providing a structure to my encryptor.

So my script is composed by the Class and a ‘dunder main’ block in which I create my instance.



3. Getting into details

In this section I'll explain each part of the script.

Let's get into the Caesar_Cipher() class.

- Class Attributes: These are attributes inherent to the class and common for all instances.

```
class Caesar_Cipher():

    british_alphabet = list(string.ascii_uppercase)
    special_characters = [' ', ',', '.', '+', '*', '-', '_', '@', '#', '%', '&', '$', '?']
    for s in special_characters:
        british_alphabet.append(s)
    british_alphabet = british_alphabet*2
```

So the attribute I create here is the british_alphabet attribute. Doing so, I've defined the letters and symbols I will be working with.

- Class Methods: These are the functions that will determine the class's behavior.

```
    def __init__(self, key):
        self.key = key
```

The constructor. Once I create an instance I must pass an argument (the key in this case) so the class can create the structure.

```
def host(self):
    current_time = datetime.datetime.now()
    moment = ''
    if  6 <= current_time.hour and current_time.hour < 12:
        moment = 'morning'
    elif 12 <= current_time.hour and current_time.hour < 18:
        moment = 'afternoon'
    elif 18 <= current_time.hour and current_time.hour < 22:
        moment = 'evening'
    elif  22 <= current_time.hour and current_time.hour < 6:
        moment = 'night'
    return f'''\nGood {moment}, I will help you encrypting or decrypting the message you desire'''
```

The host method will allow the program to greet the user in a proper way. Depending on the moment of the day the user is in.

```
def encrypt(self, word):
    ciphertext = []
    ciphertext_string = ''
    for w in word.upper():
        for l in Caesar_Cipher.british_alphabet:
            if w == l:
                initial_index = Caesar_Cipher.british_alphabet.index(l)
                final_index = initial_index + self.key
                ciphertext.append(Caesar_Cipher.british_alphabet[final_index])
                break
            else:
                continue
    for w in ciphertext:
        ciphertext_string += w
    with open ('secret_word.txt','w') as f:
        f.write(ciphertext_string)
    return ciphertext_string.capitalize()
```

This method encrypts the given word or sentence and it stores it values into a new file. And eventually returns the word/sentence encrypted.

```
def dencrypt(self, ciphertext):
    original_word = []
    original_word_str = ''
    for w in ciphertext.upper():
        for l in Caesar_Cipher.british_alphabet:
            if w == l:
                initial_index = Caesar_Cipher.british_alphabet.index(l)
                final_index = initial_index - self.key
                original_word.append(Caesar_Cipher.british_alphabet[final_index])
                break
            else:
                continue
    for w in original_word:
        original_word_str += w
    return original_word_str.capitalize()
```

This method take the word/sentence encrypted and decrypts it. Eventually returns the original secret word/sentence.

```
    def proceed(self):
        connected = True
        while connected or ValueError:
            print(self.host())
            try:
                choice = input('\nWhat would you like to do [enter "e" for encryption or "d" for decryption]? ').lower()
                if choice not in ['e','d']:
                    raise ValueError('Please you must enter "e" or "d"')
            except ValueError as err:
                print(f'{err}')
            else:
                if choice == 'e':
                    try:
                        secret_word = input('Would you be so kind to entering the secret word please: ')
                        for w in secret_word:
                            if w.upper() not in Caesar_Cipher.british_alphabet:
                                raise ValueError(f'Your password must be composed by letters from the British Alphabet and you can add thes
                            else:
                                continue
                    except ValueError as err:
                        print(f'{err}')
                    else:
                        ciphertext = self.encrypt(secret_word)
                        print(f'''Your secret word is now encrypted and I also created a backup in case you forget the ciphertext,
which is {ciphertext}''')
```

```
                elif choice == 'd':
                    try:
                        ciphert = input('''I will need you to enter the ciphertext so I can decrypt it.
Or if you just press enter I will use the ciphertext store in the backup if there is any. ''')
                        for w in ciphert:
                            if w.upper() not in Caesar_Cipher.british_alphabet and w != '':
                                raise ValueError(f'Your password must be composed by letters from the British Alphabet and you can add thes
                            else:
                                continue
                    except ValueError as err:
                        print(f'{err}')
                    else:
                        if ciphert == '':
                            with open('secret_word.txt', 'r') as f:
                                ciphert = f.read()
                            original_w = self.dencrypt(ciphert)
                            print(f'Very well here is the secret word {original_w}')
                        else:
                            original_word = self.dencrypt(ciphert)
                            print(f'Very well here is the secret word {original_word}')
                do_now = input('''What would you like to do now? If you wish to restart the app press "n" and if you wish to exit the progr
press any other key: ''')
                if do_now == 'n':
                    connected = True
                else:
                    sys.exit()
```

Finally the proceed method what it does is running the whole program.

Firstly it calls the host method to greet the user.

Then program prompts the user asking him what he desires encrypt or decrypt. If the users chooses encryption the encrypt method is called and the encrypted word/sentence is save in a file. On the other hand, if the if the user chooses decryption the program ask the user for the encrypted word/sentence or the user can use easily the word/sentence stored.

Finally the program asks the user if he/she wants to leave the program or repeat the process.

- Dunder main block

```
if __name__ == '__main__':

    while ValueError:
        try:
            key = int(input('Enter the key so I can configurate the encryptor: '))
            if key not in range(1,76):
                raise ValueError
        except ValueError:
            print('You must enter an integer lower than 76')
            pass
        else:
            encryptor = Caesar_Cipher(key)
            encryptor.proceed()
```

All the program's execution is handle in a proper way inside a 'dunder main' block.

Inside that block the user is asked for the key that will define the structure of this kind of encryption.

Then as you can see I create an instance of that class passing the key as an argument. And finally I call the proceed method so everything can run.

4. Conclusions

Overall, the program looks great. This sama type of encryption can be more personalized by deciding if we only consider words or if we want to include numbers, etc.