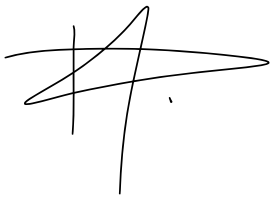
CANDIDATE'S FULL NAME: Daniel Mula Tarancón

'''This program can hack messages encrypted with the Caesar cipher from the previous project, even if you don't know the key. There are only 26 possible keys for the Caesar cipher, so a computer can easily try all possible decryptions and display the result to the user. In cryptography, we call this technique a brute-force attack.



INDEX

1. Python libraries used

A) String library

The string library in Python provides a set of useful methods and constants for working with strings. Some of the commonly used functions in the string library include:

len(): returns the length of a string

lower(): returns a lowercase version of a string

upper(): returns an uppercase version of a string

strip(): removes any leading or trailing whitespace from a string.

split(): splits a string into a list of substrings based on a delimiter

join(): joins a sequence of strings into a single string using a specified delimiter.

replace(): replaces occurrences of a substring in a string with another substring.

find(): searches for a substrings in a string and returns the index of the first occurrence, or -1 if not found.

In addition to these functions, the string library also provides some useful constants, such as ‘string.ascii_lowercase’, ‘string.ascii_uppercse’, and ‘string.ascii_letters’, which are strings containing all the lowercase letters, uppercase letters, and both lowercase and uppercase letters of the ASCII character set, respectively. The library also provides constants for other common character sets, such as ‘string.digits’, ‘string.whitespace’, and ‘string.punctuation’.

Overall, the string library is a powerful tool for working with strings in Python, and its functions and constants can be used to perform a wide range of string manipulation tasks.

B) Itertools library

The itertools library in Python provides a set of powerful tools for working with iterators and iterable objects. The library contains a number of function that can be used to generate and it is particularly useful for dealing with large or complex data sets.

Some of the commonly used functions in the itertools library include:

count(): generates an infinite that generates consecutive integers.

cycle(): cycles through an iterable indefinitely.

repeat(): generates an iterator that repeats a single value indefinitely, or a specified number of times.

chain(): combines multiple iterables into a single iterable.

product(): generates the Cartesian product of multiple iterables.

permutations(): generates all possible permutations of the elements in an iterable.

combinations(): generates all possible combinations of the elements in an iterable.

In addition to these functions, the itertools library also provides other useful tools for working with iterables, such as functions for grouping, filtering, and compressing data.

Overall, the itertools library is a powerful tool for working with iterables in Python, and its functions can be used to simplify and optimize a wide range of data processing tasks. It is particularly useful for dealing with large or complex data sets, where memory usage and processing time can be significant concerns.

2. Script Structure

For accomplishing this task, I decided to create a class called in this case Brute_Force. This class has some attributes and methods. And it will guess the password stored by a ‘Brute Force Attack’.

Also mention that the project is composed by several files.

app.py —> It is the file where you can encrypt or decrypt a secret sentence using the Caesar Cipher technique. We use this to create the secret word if it is not created.

key.txt —> In this text file we have stored the ‘key’ that defines the kind of Caesar Cipher used.

secret_word.txt —> Our secret word encrypted using the Caesar Cipher technique and the associated key.

bf.py —> This the script that answer this task and the only one we need to run for doing so.

Along this report we will only talk about the bf.py file.

3. Getting into details

```
from app import Caesar_Cipher
import string
import itertools


with open('secret_word.txt') as f:
    secret_word = f.read()

with open('key') as f:
    secret_key = int(f.read())
```

On the top of our script we import the libraries we need and in this case the class we need from the app.py file.

We also get or secret word (the word that will need to guess) and the key that specifies the kind of encryption that was applied.

```
class Brute_Force:


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

    Darth Vader
    def __init__(self): # I define the constructor (unnecessary in this case) in case I want to add something
        pass
```

Then we define our class. In this class we define some attributes. Overall, we define the list of characters that can be used to compose any secret word (in this case there are 39 characters).

Moreover, we define the constructor (unnecessary in this case since we have decided that there are not instance attributes in this class and we won't call any method when creating an instance).

```
    def generate_combinations(self, my_list, n):
        for i in range(1, n + 1):
            for combo in itertools.product(my_list, repeat=i):
                yield ''.join(combo)
```

Then I create a method called generate_combinations. This method is a generator. The goal of this method is creating a generator that yields all possible secret words that can be created with the 39 characters.

I use a generator for being more efficient.

```
    def attack(self):
        combinations = self.generate_combinations(Caesar_Cipher.british_alphabet, len(Caesar_Cipher.british_alphabet))
        guess = ''
        for i in combinations:
            print(i)
            if i == secret_word.upper():
                guess += i
                break

        key = ''

        for j in range(1, 76):
            if j == secret_key:
                key = j
                break

        print(f'''the encrypted password is:  {guess} and the key used is {key}, so using the encryptor
we can guess the password''')

        encryptor = Caesar_Cipher(key)

        print(f'The password is {encryptor.dencrypt(guess)}')
```

Eventually I define the attack method that will execute the attack. I create my generator I loop through it so I can guess the password.

However I need to guess the key too. I do that using another loop (in this case is much simper to guess).

Once I have the secret word (encrypted) and I know how it was encrypted (I know the key) I get the password using the imported class Caesar_Cipher.

4. Conclusions

Obviously this is an approach that works for guessing relative easy passwords.

In this case the program guess the password in a reasonable time if the password has five or less characters we may even go to six. Nonetheless, I will determine how much time it takes for our computer generating one guess if the password is very long (20 characters) it may tike too much time (a life time perhaps).

This is the most evident approach to guess a password but can only be used for relative easy passwords.