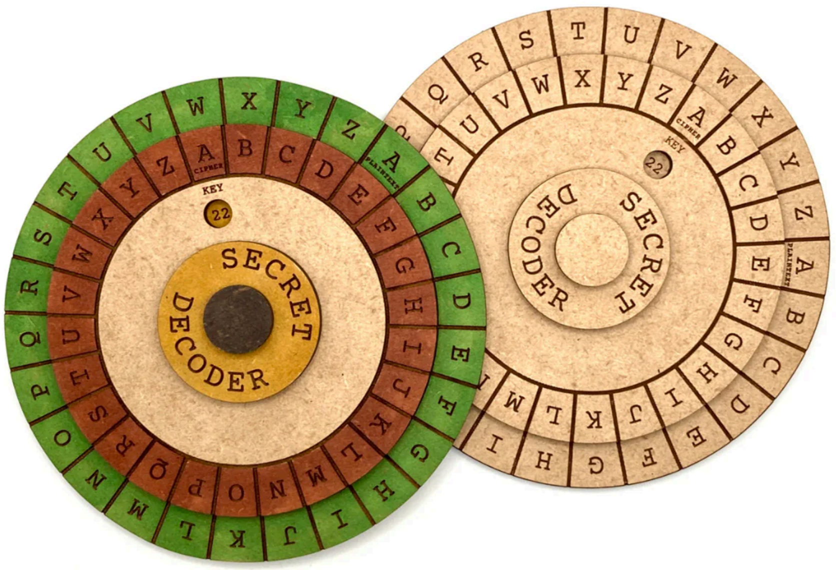

Caesar Cipher

Overview

The Caesar cipher is a simple encryption technique that was used by Julius Caesar to send secret messages to his allies.

Your job would be to construct a Caesar Cipher package that would be implemented into an API to call

steps

This project would be divided into two main steps; building the package and the API

NOTE: You can use any logic you see suits, I only provide you with some helpful steps, what matters to me is to build the package as OOP, install it and then use it's function as input to the API route

First; the package:

Implement Caesar Cipher in OOP paradigm as follows;

Your class should contain the following methods and attributes:

1. Constructor that takes the key that maps the input to the output:
    a. Self.key: a dictionary mapping each letter to another letter in the vocab.
    b. Self.antikey: a dict with a swapped values/keys of self.key

       For example key={“a”: “l”, …} then antikey={“l”: “a”, …}

       Where key would be used to encrypt and antikey would be used to decypt
2. encrypt_string(string): Method that takes the the message you want to encrypt and returns it as encrypted

       for example; caesar.encrypt_string(‘Hi”) ⇒ ul where h=u and i=l and so on based on your key

3. decrypt_string(string): Method that takes an encrypted string and tries to decrypt it

       for example; caesar.decrypt_string(‘ul”) ⇒ hi where h=u and i=l and so on based on your key
4. check_input(string): method that checks for input text validity if it contains only alphabets and spaces or not, so for example if string has “@” sign it would raise a ValueError telling that it’s not valid input, only alphabets and spaces allowed.

In another module in the same package, call the CaeserCipher class and wrap all the logic into a function to be used directly in the api; so this function should look like this:

get_cipher(string, key=None, encrypt=True), where:

String: is the string that you want either to encrypt or decrypt

Key: the key dictionary that maps between letters, it has default value of None so if the user didn’t provide an explicit key you can use the default key;

```
{"a": "d", "b": "e", "c": "f", "d": "g", "e": "h", "f": "i", "g": "j",
"h": "k","i": "l", "j": "m", "k": "n", "l": "o", "m": "p", "n": "q", "o":
"r", "p": "s","q": "t", "r": "u", "s": "v", "t": "w", "u": "x", "v": "y",
"w": "z", "x": "a","y": "b", "z": "c", " ":" "}
```

encrypt: boolean flag to direct the function; if True then encrypt this text; if False then decrypt this text

Using all this build a package that only uses this get_cipher function and does all the magic

Second the API:

Build a simple GET api that takes the payload from the user with the parameters as a dictionary and returns the result;

So the final requirement of this project is as follows:

1) The user sends a request to the api with the data required and get his results
2) Here are 4 examples of how I used the api as a final result:

```
ahmed_elbaqary@Ahmed-Elbaqary:~$ curl -G "http://127.0.0.1:5000/"
--data-urlencode 'params={"string":"Hello Data Engineers","encrypt":true}'

{

  "result": "khoor gdwd hqjlqhhuv"

}

================================================================

ahmed_elbaqary@Ahmed-Elbaqary:~$ curl -G "http://127.0.0.1:5000/"
--data-urlencode 'params={"string":"khoor gdwd hqjlqhhuv","encrypt":false}'

{

  "result": "hello data engineers"

}

================================================================

ahmed_elbaqary@Ahmed-Elbaqary:~$ curl -G "http://127.0.0.1:5000/"
--data-urlencode 'params={"string":"Hello Data Engineers this is if you want to use a
key I defined it manually for test","encrypt":true, "key": {"a": "d", "b": "e", "c": "f", "d":
"g", "e": "h", "f": "i", "g": "j", "h": "k",

            "i": "l", "j": "m", "k": "n", "l": "o", "m": "p", "n": "q", "o": "r", "p": "s",

            "q": "t", "r": "u", "s": "v", "t": "w", "u": "x", "v": "y", "w": "z", "x": "a",

            "y": "b", "z": "c", " ":" "}}'

{

  "result": "khoor gdwd hqjlqhhuv wklv lv li brx zdqw wr xvh d nhb l ghilqhg lw
pdqxdoob iru whvw"

}

================================================================

ahmed_elbaqary@Ahmed-Elbaqary:~$ curl -G "http://127.0.0.1:5000/"
--data-urlencode 'params={"string":"khoor gdwd hqjlqhhuv wklv lv li brx zdqw wr xvh
d nhb l ghilqhg lw pdqxdoob iru whvw","encrypt":false, "key": {"a": "d", "b": "e", "c":
"f", "d": "g", "e": "h", "f": "i", "g": "j", "h": "k",
        "i": "l", "j": "m", "k": "n", "l": "o", "m": "p", "n": "q", "o": "r", "p": "s",
        "q": "t", "r": "u", "s": "v", "t": "w", "u": "x", "v": "y", "w": "z", "x": "a",
        "y": "b", "z": "c", " ":" "}}'
{
  "result": "hello data engineers this is if you want to use a key i defined it manually for test"
}
```

Notes

The project would go like this:

1) You build a Caesar Cipher package that can be installed in the python env as known, the package has two modules, one for the Class itself with all its methods/attributes, and another module with a function that wraps up all the logic the API needs to use
2) Build an API that can call this package functions and the user can call with data and this would be the final result

Bonus, from real life use case

Write a python function that takes two directories as input, one is source directorya and the other is destination like this; **func(source, destination)**

The source directory has so many directories in it and in each one it has many files like .txt, .png, .json files and so on

Your goal is to write a function that go inside each level of this directory and reads its content; then check its type either its an image or not

If the current file we are reading is an image, transfer it to the destination directory else do nothing,

If you are looking to a directory, go inside it and search into its content

Packages you will need; **Shutil, os, imghdr**

Search for them and find out how to do such a thing, you can search for something called **calling function recursively** to know how to map from one directory to another

Another Bonus, For anyone who likes to play around

HangMan

Overview

Hangman is an old school favorite, a word game where the goal is simply to find the missing word or words.

You will be presented with a number of blank spaces representing the missing letters you need to find.

Use the keyboard to guess a letter (I recommend starting with vowels).

If your chosen letter exists in the answer, then all places in the answer where that letter appears will be revealed.

After you've revealed several letters, you may be able to guess what the answer is and fill in the remaining letters.

Be warned, every time you guess a letter wrong you lose a life and the hangman begins to appear, piece by piece.

Solve the puzzle before the hangman dies.

steps

Write a python script in OOP fashion to simulate this game

Your class should contain the following methods and attributes:

1. Constructor that takes the word to guess and construct the following attributes in it:
    a. Self.faild_attempts: to track how many times you guessed the letter wrong
    b. Self.word_to_guess: the word you should guess, it comes from random choice from list of words;
        i. For example words=[“word_1”, “word_2”, “word_3”, …] then randomly make the program choose one of these words to play with
    c. Self.game_progress: attribute to track the length of the word and show your progress; for example, the word_to_guess is “python” then at the beginning it should look like this game_progress=[“_”, “_”, “_”, “_”, “_”, “_”]

2. fnd_indexes(letter): Method that takes the letter you guessed and returns a list with its indexes in the word you guessed;

    for example; you guess (p); and the word was (swaps) then this function returns [3,4] which are the indexes of the letters;

3. is_invalid_letter(letter): check if the user input is a letter or number etc.
4. game_status(): method to print the word and how many lives you have left.
5. Update_progress(letter, indexes): method to update the game_progress with new valid letter, this is used to reflect the change in the self.game_progress to look like game_progress=[“_”, “_”, “_”, “p”, “p”, “_”]
6. user_input(): method to get the input from the user
7. play(): the main method that gathers all this logic to play the game, it works as follows:
    a. As long as we are not out of moves, we will play, we have only 7 fail attempts
    b. first we print the game_status and then take input from the user
    c. Check if the input is valid letter and not any character, if not print a warning and go to the next move without counting the current attempt
    d. Check if we guessed this letter before; if guessed, give a warn and go to the next move without counting the current attempt
    e. Check if the letter is in the word_to_guess; if it is update the game_progress
    f. Count how many “_” left in the game_prgress; if count==0 then you won
    g. And if the letter is not in the word_to_guess, increase the faild_attempts by one
    h. When we are out of faild_attempts valid, print “you are lost!”

Notes

Use the following words to play with;

Words = [“palestine”, “freedom”, “egypt”, “advanced”, “anxity”, “attitude”, “component”, “conclude”, “army”, “conflict”, “dept”, “defense”, “element”, “emerge”, “employee”, “wireless”, “python”, “c-language”, “emptions”, “fancy”, “machine”, “ghosted”, “global”, “hate”, “illness”, “instead”, “know”, “label”, “lady”, “mirror”, “mood”, “organization”, “peak”, “perfectly”, “poetry”, “reduce”, “season”, “secret”, “show”, “signal”, “suit”, “thank”, “train”, “victim”, “violence”, “wine”, “zone”, “zionism”]

Or you can use any group of words to do so, start with a single word to check your logic then fill your list with words to play around

Use the following hangman template to show progress:

HANGMAN = [

```
‘_______________’,
‘|              | ’,
‘|              O ’,
‘|              | ’,
‘|             /|\ ’,
‘|              | ’,
‘|             / \ ’,]
```

Note, to update you can compine the faild_attempts with the string joining methods to display, so for example when we will display like this;

failed for 3 times

```
_______________
|              |
|              O
```

Failed for 7 times, in this case we will fail:

```
_______________
|             |
|             O
|             |
|            /|\
|             |
|            / \
```

Find a way to do this; it’s easy BTW