Projet Chiffrement de Vigenère

1. Explications des fonctions utilisées

Note : Tout le code a été mis dans des fonctions pour simplifier la lecture et l'utilisation de notre programme.

creation_arguments() : Créer les arguments grâce à la librairie argparse qui seront affichés dans le cmd.

fichier_existe_il() : Vérifie si le fichier donné existe ou non. Il retourne un booléen.

text_contient_il_caractere_special() : Vérifie si le texte contient un caractère spécial ou non. Il retourne un booléen.

verifier_si_argument_est_null() : Vérifie si l'utilisateur a donné un argument dans le cmd. Il retourne un booléen.

trouve_index() : Trouve l'index de l'objet dans une liste.

majuscule_minuscule_ou _autre() : Regarde si la lettre est une majuscule, une minuscule ou un autre caractère. Retourne un string « Majuscule », « Minuscule », « Autre » dépendant de la réponse.

creation_message() : Lis le fichier d'entrée et retourne son contenu dans un string.

creation_clef() : Assigne la clef de chiffrement à une variable ou la crée si l'utilisateur ne l'a pas rentrée en argument.

creation_fichier_sortie() : Assigne le fichier de sortie à une variable ou le crée si l'utilisateur ne l'a pas rentrée en argument.

chiffrement_ou_dechiffrement() : Si l'utilisateur a oublié de préciser s'il voulait chiffrer ou déchiffrer, demande s'il veut chiffrer ou déchiffrer et modifie la valeur de l'argument en fonction de sa réponse.

chiffrement() : Chiffre le message avec la clef de chiffrement.

dechiffrement() : Déchiffre le message avec la clef de chiffrement. (Note : Pour obtenir le message d'origine, il faut utiliser la même clef que celle de chiffrement.)

char.ord() : Transforme le caractère en sa valeur dans le tableau « ASCII ».

os.path.exists() : Vérifie si le lien existe. Retourne un booléen.

2. Documentation sur la librairie argparse

La librairie « argparse » nous permet d'écrire des commandes d'aide qui sont utiles à l'utilisateur pour la compréhension du script dans le cmd. Elle nous permet et également de demander des informations (arguments) à l'utilisateur pour faire fonctionner notre script.

```
C:\Users\BastienVienet\OneDrive - Jobtrek\Apprentissage - Jobtrek\EPSIC\122_Scripts\Python\TP_Vigenere>Vigenere.py -h
usage: Vigenere.py [-h] [-fe] [-c] [-fs] [-ch | -de]

Programme de chiffrage Vigenère

options:
  -h, --help            show this help message and exit
  -fe , --fichierEntree
                        Fichier dans lequel se trouve le message à chiffrer ou à déchiffrer
  -c , --clef           Clef de chiffrage
  -fs , --fichierSortie
                        Fichier dans lequel se trouvera le résultat (crée le fichier s'il n'existe pas)
  -ch, --chiffrement    Permet de specifier l'action de chiffrement
  -de, --dechiffrement  Permet de spécifier l'action de déchiffrement
```

Tout d'abord, il faut importer la librairie avec la ligne de code suivante : « import argparse ». Une fois que la librairie a été importée, il faut créer un « parser ». C'est celui qui va coller les informations dans le cmd. Ensuite, il faut créer un argument que l'on souhaite utiliser avec « parser.add_argument ». Cet argument doit contenir un nom, un type et une description.

Finalement, on assigne tout les arguments à une variable via « parser.parse_args » que l'on va utiliser tout au long de notre script.

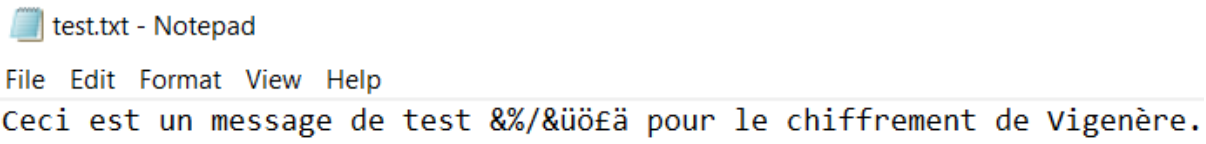
3. Documentation sur la fonction chiffrement + tests

Pour utiliser le chiffrement de Vigenère, nous devons « additionner » deux lettres. Malheureusement, on ne peut pas « additionner » deux lettres sans faire de modifications. C'est pour cela que nous devons les transformer en valeurs numériques.

Tout d'abord, nous voulions utiliser la table « ASCII » mais pour nous, il est plus simple de gérer 26 caractères plutôt que les 128 caractères présents dans la table « ASCII ». Nous avons donc créé deux strings avec toutes les lettres de l'alphabet. Un pour les lettres en majuscules et l'autre pour celles en minuscules. Ce sont les deux strings que nous allons parcourir pour chiffrer notre message.

Voici comment fonctionne notre programme pour chiffrer notre message :

Texte de base :



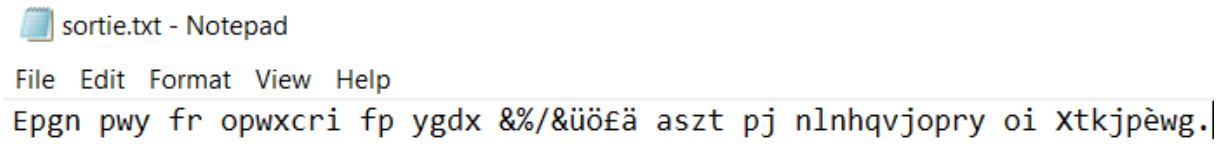
CMD :

```
C:\Users\BastienVienet\OneDrive - Jobtrek\Apprentissage - Jobtrek\EPSIC\122_Scripts\Python\TP_Vigenere>Vigenere.py
Veuillez écrire le fichier d'entrée : test.txt
Veuillez écrire la clé de chiffrement : clef
Veuillez écrire le fichier de sortie : sortie.txt
Voulez-vous chiffrer ou déchiffrer ? : chiffrer
Message chiffré avec succès dans sortie.txt
```

Exemple du même message dans le CMD en utilisant tous les arguments en même temps :

```
C:\Users\BastienVienet\OneDrive - Jobtrek\Apprentissage - Jobtrek\EPSIC\122_Scripts\Python\TP_Vigenere>Vigenere.py -fe test.txt -c clef -fs sortie.txt -ch
Message chiffré avec succès dans sortie.txt
```

Texte chiffré :

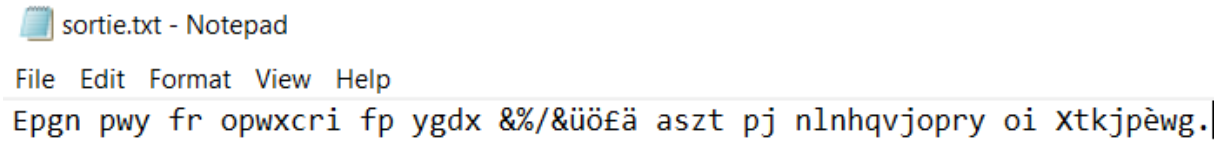


4. Documentation sur la fonction déchiffrement + tests

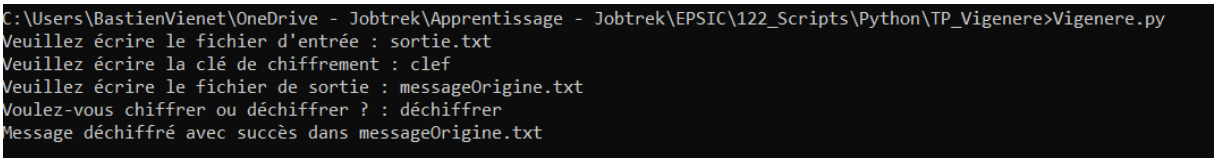
Pour déchiffrer le message, nous devons faire exactement la même chose que le chiffrement sauf que à la place « d'additionner » les lettres, nous devons les soustraire. Il est cependant important de noter qu'il faut utiliser la même clef utilisée dans le chiffrement.

Voici comment fonctionne notre programme pour déchiffrer notre message :

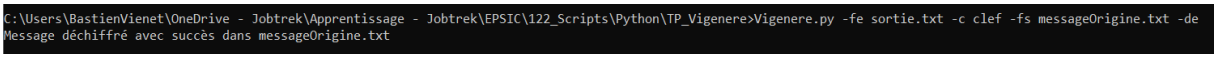
Texte de base :



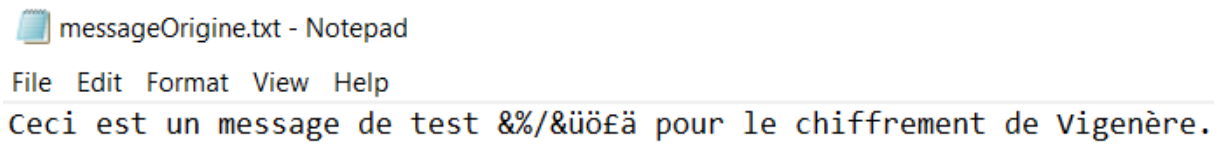
CMD :



Exemple du même message dans le CMD en utilisant tous les arguments en même temps :



Texte déchiffré :



5. Mini conclusion

Pour conclure, nous allons vous transmettre notre ressenti au niveau de ce travail pratique.

Malheureusement, nous trouvons que nous n'avons pas eu assez de temps. En effet, 1 semaine et demie pour rendre un projet pareil est très court. De plus, devoir apprendre à utiliser un nouvel outil (librairie argparse) pendant un examen est quelque chose de pas très cohérent avec ce qu'on a vu en cours. (Même chose avec les ADs dans le travail pratique powershell.) En ce qui concerne python en lui-même, l'apprentissage de ce langage a été challengeant pour les raisons suivantes : « tout est en minuscule, pas de {} pour commencer un bloc, toutes les fonctions sont des abréviations, besoin de déclarer une méthode avant de pouvoir l'utiliser, etc..

Malgré toutes les difficultés présentées ci-dessus, nous sommes très contents du résultat final et de notre rendu. On espère que Bash saura également nous captiver !