Introduction

Cette documentation décrit la structure et le fonctionnement d’un script Python servant a chiffré et déchiffré des fichiers texte en utilisant la méthode du chiffre de Vigenère. Il est exécutable en ligne de commande et prend en charge les options pour spécifier le fichier d’entrée, la clé de chiffrement et le fichier de sortie.

Fonctions employées

Je commence mon script par l’importation de trois modules dont argparse, utilisé pour gérer les arguments et permettant au script de recevoir des options et des paramètres extérieurs, sys qui est utilisé pour interagir avec le système de fichiers et vérifiant notamment les permissions des fichiers et os qui est utilisé pour les interactions liées au système.

Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement

La fonction “verify\_file”

**Objectif** :

Vérifier que l’argument spécifié pour le fichier d’entrée est un fichier existant et qu'il est accessible en lecture.

**Argument :**

**“**path**”:** le chemin du fichier à vérifier.

**Comportement** :

Si le fichier n’existe pas ou qu’il n’est pas valide une erreur l’indiquant est retournée et si le fichier ne peut pas être lu une erreur différente l’indiquant est retournée également.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

La fonction “ask\_to\_creat\_file”

**Objectif** :

Couvrir le cas où l’utilisateur spécifierait un fichier de sortie inexistant.

**Argument :**

**“**path**”:** le chemin du fichier à vérifier.

**Comportement** :

Si c’est le ca, elle demande à l’utilisateur s’il souhaite ou non créer le fichier et stocke, si la réponse est oui, il créer le fichier avec “open(path, 'w').close()” et affiche un message pour dire que c’est fait. Si l’utilisateur répond non, la fonction affiche un message indiquant qu’un script de sortie est requis pour pouvoir procéder.

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

La fonction “chiffrement\_vigenere”:

**Objectif** :

Chiffrer l’entier d’un fichier.

**Argument :**

* text (str): texte claire à chiffrer.
* key (str): clé utilisée pour le chiffrement qui doit être une série de lettres.

**Comportement** :

La fonction utilise une méthode de chiffrement par substitution où chaque lettre dans le texte va être décalée le long de l'alphabet d’un nombre de places qui dépend de la lettre correspondante de la clé de chiffrement.

Le chiffrement se fait une lettre après l’autre et si la clé est plus courte que le texte, la clé est répétée pour faire la même taille. Les lettres sont toujours décalées de la même manière et dans la même direction, donc après 'z' ou 'Z', on revient à 'a' ou 'A'.

**Tests :**

Pour tester cette fonction,voyons si le texte “AbC” avec la clé “KEY” devient “KfA”

Pour savoir comment utiliser le script exécuter le script en ligne de commande suivit de “--help”

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

Affichage de –help

Nous devons donc entrer le chemin d’accès, la clé de chiffrement et le chemin d’accès du fichier de sortie et exécuter le script.



Execution du script

Ci-dessous le fichier d’entrée.

Une image contenant texte, ligne, Police, Tracé

Description générée automatiquement

Fichier d’entrée

Et ci-dessous, le fichier de sortie contenant bel et bien les lettres “Kfa”

Une image contenant texte, ligne, Police, blanc

Description générée automatiquement

Résultat du chiffrement dans le fichier de sortie

La fonction “dechiffrement\_vigenere” :

**Objectif** :

Déchiffrer l’entier d’un fichier.

**Argument :**

* encrypted\_text (str): Texte chiffré à déchiffrer.
* key (str): Clé utilisée pour le déchiffrement qui doit être une série de lettres.

**Comportement** :

La fonction déchiffre un texte en utilisant le processus inverse du chiffrement Vigenère. Comme pour le chiffrement, la clé est répétée si elle est plus courte que le texte.

La fonction “main” :

**Objectif** :

Analyser les arguments, gérer la création de fichiers et exécuter le chiffrement ou le déchiffrement.

**Comportement** :

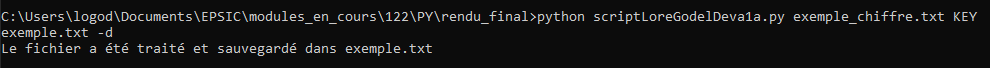
Cette fonction configure le parseur d'arguments et parse les arguments puis, elle vérifie si le fichier de sortie est déjà existant et si ce n'est pas le cas, elle propose à l'utilisateur de le créer.

Selon les options spécifiées, elle chiffre/déchiffre le fichier d’entrée puis sauvegarde le résultat dans le fichier de sortie.

Elle affiche des messages d’erreur appropriés pour les erreurs pouvant subvenir durant l'ouverture des fichiers ou pendant le processus de chiffrement/déchiffrement.

**Tests :**

Le fonctionnement est le même que pour le chiffrement sauf que nous devons préciser “-d” à la place de “-c” :



Execution du script

Ci-dessous le fichier d’entrée.

Une image contenant texte, ligne, Police, capture d’écran

Description générée automatiquement

Fichier d'entrée

Et ci-dessous, le fichier de sortie

Une image contenant texte, ligne, Police, capture d’écran

Description générée automatiquement

Résultat du chiffrement dans le fichier de sortie

Argparse

Argparse (qui sous-entend argument parser ou analyseur d'arguments) est un module (fichier python contenant un script réutilisable importé dans un autre script à l’aide du mot-clé “import” ) faisant partie de la bibliothèque standard Python.

*“La bibliothèque standard Python (The Python Standard Library en anglais) à toujours fait partie intégrante des fichiers d'installation de python et elle contient les modules et paquets pour la plupart des fonctionnalités de base*.”

Le module Argparse arriva avec la version 2.7. de Python et rendit plus facile la conception d’interfaces en lignes de commandes. Il gère les arguments postitionels (obligatoires) et les arguments optionnels (les options).

Dans ces fonctionnalités on retrouve :

La gestion des arguments que l’utilisateur peut passer au script lors de son exécution:

* Pour créer l’analyseur d’argument utiliser : “ArgumentParser()”, vous pouvez spécifier des paramètres tels que “description” pour décrire le script ou “formatter\_class” pour personnaliser l’affichage de l’aide.
* Pour ajouter des arguments utiliser : “add\_argument()”, vous pouvez aussi spécifier des paramètres tels que “help” qui donne une description de ce que fait l’argument ou “type” qui définit le type auquel l’argument doit être converti (char, float, Boolean, etc).
* Pour annalyser les arguments entrés par l’utilisateur et les renvoyer au script utiliser : “parse\_args()”

Le retour automatique d’erreur

* Si un argument obligatoire n’a pas été fourni, “parse\_args()” généra automatiquement un message d’erreur indiquant qu’un argument est manquant.
* Si les arguments ne correspondent pas aux critères définis, un message d'erreur approprié sera généré et l’exécution du programme sera arrêtée.

L’analyse et l’interprétation des arguments

* Pour analyser les arguments de la ligne de commande utiliser : “parse\_args()”. Cette méthode examine chaque argument, valide son type, applique les règles spécifiées et si les arguments sont valides, “parse\_args()” retourne un objet de type Namespace ayant des attributs qui correspondent aux noms des arguments définis dans l’analyseur ce qui permet d’accéder finalement à leurs valeurs.

Conclusion

Encore débutante en python, ce projet m'a servi à mettre en pratique les bases acquises en classe, j’ai pu découvrir ce qu’était la bibliothèque principale Python et que c'est en partie ce qui rendait Python agréable d'utilisation. J’ai connu la joie réelle en voyant mon premier fichier chiffré, merci à vous !

Bibliographie

Python Documentation:

[*https://docs.python.org/3/howto/argparse.html*](https://docs.python.org/3/howto/argparse.html) *,* [*https://docs.python.org/3/library/argparse.html*](https://docs.python.org/3/library/argparse.html)

Tutorial on Argparse*:*

[*https://www.youtube.com/watch?v=FbEJN8FsJ9U*](https://www.youtube.com/watch?v=FbEJN8FsJ9U)