RAPPORT DE PROJET
Romain – Mattias – Eva

OBJECTIFS ET MOTIVATIONS

Objectifs

Dans ce projet, l'objectif était d'obtenir un programme qui puisse encoder et décoder trois types de chiffres bien connus : le Morse, le chiffre de Vigenère et le chiffre de César.

L'utilisateur devrait pouvoir entrer le type de chiffrement qu'il souhaite employer et le programme réagirait en conséquence. Ainsi, si l'utilisateur choisit le Code Morse, il n'aurait besoin d'entrer que le texte, tandis que s'il choisit le chiffre de Vigenère, le programme devrait lui demander d'entrer une clef de chiffrement. De même, s'il choisit le chiffre de César, il devrait entrer la valeur du décalage qu'il souhaite faire et sa direction.

Ensuite, selon chaque chiffre, les attentes seraient différentes. Le Code Morse traduirait l'alphabet latin, les chiffres romains et les symboles de ponctuation basiques.

Le chiffre de César, lui, garderait la ponctuation et changerait les lettres et les chiffres selon la valeur du décalage demandé, tout en gardant les majuscules et les minuscules. De plus, il devrait réussir à chiffrer ou déchiffrer le texte quelle que soit la valeur du décalage (positive, négative, plus de 26, moins de -26) si c'est un nombre entier.

Enfin, le chiffre de Vigenère changerait uniquement les lettres et conserverait les chiffres et la ponctuation, gardant lui aussi les majuscules et les minuscules telles qu'entrées dans le texte à décoder.

Motivations

Pour ce projet, nous sommes partis sur de la cryptographie car c'est un sujet qui intéresse la majorité du groupe. De plus, le côté mathématique et théorique de l'encodage des différents chiffres nous attirait. Nous avons aussi pris ce projet comme une opportunité de comprendre comment ces différents codes marchent et comment ils ont pu être craqués.

ORGANISATION DU TRAVAIL EN ÉQUIPE

Pour élaborer ce projet, nous nous sommes répartis les tâches selon nos capacités. À chaque étape de l'avancée du projet, nous avons mis en commun et fignolé les détails selon ce qu'on avait convenu au préalable.

Ainsi, Mattias a proposé les idées et s'est occupé du rapport, Romain a pu s'occuper des recherches et a assemblé le programme final et Eva a fait les différentes fonctions.

DESCRIPTION DES ÉLÉMENTS DU PROGRAMME

Dans le programme, vous trouverez quatre fonctions (vigenere, cesar, morse_encodage, morse_decodage), des tests pour chacune de ces fonctions et le corps du programme en lui-même.

Tests et corps

Corps

Le corps consiste en plusieurs demande d'entrées (`quoi`, `txt`, etc.), qui sont ensuite testées. Par exemple, si l'utilisateur entre ''decoder'' au lieu de ''décoder'', ça permet d'éviter que la fonction saute une étape qui inverserait certaines valeurs. Tout cela est indenté dans une boucle `while` qui permet de recommencer si l'utilisateur le veut.

Tests

Les tests sont faits simplement en vérifiant que les fonctions rendent bien la même chose que ce que nous avions calculé à la main. Ainsi, ils retournent ''Test <nom du chiffre> OK'' si celui-ci est réussi ou ''Test <nom du chiffre> pas OK : <résultat trouvé>'' si celui-ci ne passe pas.

Fonctions

Chiffre de Vigenère

Dans cette fonction appelée `vigenere`, nous avons besoin de trois variables, `quoi`, `txt` et `clef`. La première demande si l'utilisateur veut encoder ou décoder le texte, qui est la deuxième variable. La dernière correspond à la clef de cryptage.

Après vérification qu'il n'y a pas de problème avec la clef de cryptage (vide), le programme crée des listes des ascii de chaque caractère du texte et de la clef afin de pouvoir les manipuler (`txt_ascii` et `clef_ascii`). Une fois le calcul fait pour chaque caractère (texte_crypté[i] = (texte[i] + clef[i]) %26), il injecte celui qu'il obtient dans une chaîne de caractères (`txt_final`) qui sera retourné à la fin de la fonction.

Chiffre de César

Dans cette fonction appelée `cesar`, nous avons besoin de quatre variables, `quoi` et `txt` comme pour la fonction `vigenere`, `decalage`, qui correspond à la valeur du décalage (nombre entier positif ou négatif) et `direction`, qui est sa direction (droite ou gauche).

Après avoir inversé la valeur du décalage si on l'utilisateur veut décoder ou qu'il a choisi la gauche, le programme transforme le texte en une liste d'ascii. Il ajoute ensuite le décalage et retransforme en alphabet latin et retourne le texte obtenu.

Encodage du Morse

Dans cette fonction appelée `morse_encodage`, nous n'avons besoin que d'une variable, le texte (`txt`).

Ensuite, grâce à un dictionnaire comprenant l'alphabet latin, les chiffres et la ponctuation basique ainsi que leur retranscription en Code Morse (`caracteres_morse`), chaque caractère est traduit. Le texte final (`txt_final`) comprenant tout le texte en Morse est retourné à la fin, avec un espace pour séparer deux caractères et deux espaces pour séparer deux mots.

Décodage du Morse

Le décodage du Code Morse utilise la même unique variable et le même dictionnaire que l'encodage.

Pour passer la difficulté de la différentiation des caractères, un caractère n'étant pas forcément égal à une lettre (''a'' → un caractère ; ''.-'' → deux caractères), nous avons placé dans une variable tous les caractères jusqu'à tomber sur un espace dans une liste puis traduit cette liste. Cette variable est ensuite réinitialisée à la fin de la boucle. De même que pour l'encodage, deux espaces signifient un nouveau mot.

Pour la traduction, nous avons cherché à inverser le dictionnaire, c'est-à-dire qu'il nous rende la clef si nous lui donnions la valeur. Pour cela, une variable `j` prend comme valeur la position du caractère que l'on veut traduire et on ajoute au texte final la clef à cette position.

DÉVELOPPEMENT THÉORIQUE

On a choisi de faire la preuve et la complexité de la troisième boucle `for` de la fonction `cesar`.

```
113     for i in range(len(txt_ascii)):
114         if type_caractere[i]=="m":
115             txt_final+=chr(txt_ascii[i]+97)
116         elif type_caractere[i]=="M":
117             txt_final+=chr(txt_ascii[i]+65)
118         elif type_caractere[i]=="c":
119             txt_final+=chr(txt_ascii[i]+48)
120         elif type_caractere[i]=="s":
121             txt_final+=chr(txt_ascii[i])
```

Preuve

Soit **P** : La variable `txt_final` contient toujours `i` caractères.

Supposons que **P** soit vrai au tour `i` de la boucle et montrons qu'elle l'est toujours au début du tour `i+1` (soit à la fin du tour `i`).

Dans le cas où `type_caractere[i]=='m'` : un unique caractère est ajouté à la variable `txt_ascii` grâce à la commande `txt_final+=chr(txt_ascii[i]+97)`. Elle contient donc `i` caractères.

Idem pour `type_caractere[i]=='M'`, `type_caractere[i]=='c'`, ou `type_caractere[i]=='s'` puisque ces conditions englobent l'ensemble des possibilités.

Donc si **P** est vrai en entrée de la boucle, il est vrai en sortie. **P** est un invariant de boucle.

De plus, au premier tour de la boucle, `txt_final` est vide, donc **P** est tout le temps vrai à l'entrée de la boucle.

Complexité

On utilise une complexité au pire des cas en comptant le nombre d'opérations élémentaires faites au maximum à l'exécution du programme. On se trouve dans une boucle `for`.

Pour i allant de a à b :

N(opérations) = c(a)+..+c(b) où c est le nombre d'opérations effectuées quand i=a,…,i=b.

On a donc : c(b-a+1)

Trois des boucles if contiennent le maximum d'opérations élémentaires : un test (`type_caractere[i]=="<m/M/c/s>"`), une affectation (`txt_final+= chr(txt_ascii[i]+97)`) et deux opérations arithmétiques (`+=` et `txt_ascii[i]+97`).

On a donc au total quatre opérations élémentaires par tour, soit une complexité de c(n)=4n où n est la taille de l'entrée (`len(txt_ascii)`).

SITOGRAPHIE

Voici les liens de toutes les sources qui nous ont aidé pour écrire les fonctions :

La documentation Python

https://docs.python.org/release/3.5.8/reference/index.html
https://docs.python.org/release/3.5.8/library/functions.html
https://docs.python.org/release/3.5.8/py-modindex.html
https://docs.python.org/release/3.5.8/genindex-all.html
https://docs.python.org/release/3.5.8/glossary.html

Les fontions

https://www.w3schools.com/python/ref_dictionary_keys.asp
https://www.w3schools.com/python/ref_list_index.asp
https://www.quora.com/In-Python-how-do-I-convert-a-string-to-a-list-in-which-each-character-will-be-separated
https://stackoverflow.com/questions/1323364/in-python-how-to-check-if-a-string-only-contains-certain-characters

Aides pour comprendre le raisonnement à avoir

https://www.geeksforgeeks.org/morse-code-translator-python/
https://fr.wikipedia.org/wiki/Chiffre_de_Vigen%C3%A8re#Principe_math%C3%A9matique