Des Chiffres et des Trésors

PCSI/PTSI

Consignes

† Rendu

Le devoir est à faire de manière électronique. Les consignes du rendu sont disponible sur le site web du cours

† Règles de travail

Le travail en groupe et l'entre aide sont encouragés. Toutefois le monde doit rendre son propre travail. Le transfert de code est interdit. Tous codes identiques à renommage et permutations d'instructions près seront sanctionnés par un 0 à la note finale. Pour éviter que cela arrive, la règle à suivre est simple :

- Ne pas copier-coller le code d'un camarade.
- Si un camarade vous explique la solution à partir de son propre code, prendre le temps de digérer l'explication et réécrire plus tard le code de la réponse.

† Code Python

Les consignes suivantes **doivent** être respectées sous peine d'être pénalisé par des **points négatifs**.

Le code doit être clair. - *-1 point par fonction.*

- Évitez les lignes à rallonge. S'il faut introduire une variable intermédiaire, faîte le.
- Nommez vos variables avec du sens. Évitez toutefois des noms trop longs. Auquel cas, décrivez le rôle de la variable en commentaire.
- Ne surchargez pas de commentaires. Les commentaires utiles sont :
  - Description du rôle d'une fonction
  - Description du rôle d'une variable
  - Description du rôle d'une boucle complexe.

Respectez la syntaxe Python. - *0 à l'exercice si syntaxe inventée.*

- Gardez la fiche de syntaxe à côté de vous lorsque vous codez
- Vérifiez que ce que vous écrivez respecte la syntaxe.
- Vérifiez que votre code s'exécute avant de l'envoyer.

Pour nommer une fonction, utilisez le nom donné dans l'énoncé - *0 à l'exercice sinon*

Respectez le nom est l'ordre des arguments - *-1 à l'exercice sinon*

Utilisation de `print` ou de `input` interdite - *0 à l'exercice sinon*

Le but de ce projet est d'aborder différentes techniques de dissimulation d'information, ainsi que leur limites.

I Chiffrement par Décalage

La *cryptographie* est la discipline s'attachant à protéger des messages en les rendant supposément inintelligible à tout autre personne que sont destinataire.

Le chiffrement est le procédé transformant un texte en clair en message incompréhensible à l'aide d'une clé. Le déchiffrement est le procédé permettant de retrouver le texte en clair à partir du texte chiffré et de la clé de chiffrement.

Le chiffrement par décalage consiste à remplacer une lettre par une autre, suivant un motif régulier basé sur la distance d'une lettre à une autre. La distance de deux lettres minuscules latines est basée sur leur position dans l'alphabet. En numérotant les lettres de l'alphabet latin de 0 à 25, si `l` est située avant `k` dans l'alphabet, la distance de `l` à `k` correspond à la différence des numéros de `k` et `l`. Si `l` est situé après `k`, on considère que l'alphabet est circulaire et donc que la distance de `l` à `k` correspond à la distance `l` à `'z'` puis de `a` à `k`, auxquelles ont ajoute 1. Par exemple : `d('b','j') == 8` et `d('t','c') = 9`

Le chiffrement de César est la version la plus simple adoptant ce principe. Il se nomme ainsi, car selon Suétone, Jules César l'utilisait avec l'alphabet grec.

La correspondance peut être obtenue en observant cette roue :

Décalage de 19

Le chiffrement de César consiste à remplacer chaque caractère par le caractère correspondant dans la roue. Les caractères spéciaux et espaces sont préservés. Avec un décalage de 19 on obtient donc le chiffrement suivant :

"nadine␣n'est␣pas␣un␣velociraptor." ↦ "gtwbgx␣g'xlm␣itl␣ng␣oxehvbktimhk."

On appelle alors le décalage utilisé *clé de chiffrement*. La chaîne obtenue après décalage est appelée *chiffrement de la chaîne.*. À partir du chiffrement d'une chaîne et de la clé de chiffrement, l'opération qui permet de récupérer la chaîne originale s'appelle *déchiffrement de la chaîne*.

On se restreint dans cette section à des textes dont les caractères possibles sont :

— les caractères ASCII minuscules : `"abcdefghijklmnopqrstuvwxyz"`

— `"␣"` , `"\n"`, `"'"`, `"."` et `","` ,

1.1 Chiffrement et Déchiffrement

Question 1.1. *Décalage de caractère*

Implémentez une fonction `decalage(c,n)` qui prend en entrée un caractère `c` et un décalage `n` et qui renvoie le caractère décalé de `n` s'il s'agit d'un caractère ASCII minuscule, sinon renvoie le caractère lui-même.

Question 1.2. *Chiffrement de César*

En utilisant la fonction `decalage`, implémentez une fonction `chiffrement_cesar(s,n)` qui prend en entrée une chaîne de caractère `s` et une clé de chiffrement `n` et qui renvoie le chiffrement de César de la chaîne.

Question 1.3. *Déchiffrement de César*

En utilisant la fonction `chiffrement_cesar`, implémentez une fonction `dechiffrement_cesar(s,n)` qui prend en entrée une chaîne de caractère `s` et une clé de chiffrement `n` et qui déchiffre la chaîne de caractère à l'aide de la clé `n`.

Question 1.4. *Chiffrement de Fichiers*

Cette question n'est pas notée. Le code de cette question est à exécuter dans la console.

Vous pouvez aller lire la partie de cours sur les fichiers pour retrouver les fonctions qui permettent d'ouvrir un fichier, de le lire, de le parcourir et d'écrire dedans.

Voici un exemple de code qui permet de lire un fichier et d'écrire un nouveau fichier avec le contenu du premier crypté à l'aide de la clé `12`.

```
f = open('messages/texte_a_crypter.txt','r')
texte = f.read()
texte_crypte = chiffrement_cesar(texte,12)
f_crypte = open('texte_crypte.txt','w') # Ouverture du fichier en ecriture
f_crypte.write(texte_crypte) # Ecriture de texte_crypte dans le fichier
f.close()
f_decrypte.close()
```

Vous pouvez essayer d'adapter le code pour décrypter le fichier `"message_secret_1.txt"` avec la clé `7`.

Question 1.5. *Chiffrement de Vigenère*

Le chiffre de Vigenère s'appuie sur le chiffre de César, mais le décalage n'est cette fois pas le même pour toutes les lettres du texte. Le chiffrement est basé sur une clé qui prend la forme d'un mot. On répète ensuite le mot pour obtenir un texte `K` de la même longueur que le texte à crypter. Pour savoir le décalage à appliquer sur la lettre `i` du texte original, on regarde la lettre `K[i]` qui défini le décalage à appliquer. Le décalage est le numéro de la lettre dans l'alphabet, ainsi la lettre `"n"` signifie un décalage de 13 lettres.

Le texte `"nadine␣n'est␣pas␣un␣velociraptor."` chiffré avec la clé `"nadine"` donne le texte suivant :

```
"nadine n'est pas un velociraptor."  # Texte à crypter
"nadine n adi nen ad inenadinenad "  # Clé répétée
"aagqai a'evb cef uq drpbclzntgou."  # Texte chiffré
```

Implémentez une fonction `chiffrement_vigenere` et `dechiffrement_vigenere` qui prennent en entrée une chaîne de caractère et une clé de chiffrement et qui renvoie respectivement la chaîne chiffrée et la chaîne déchiffrée avec la clé.

Vous pouvez essayer de décrypter le fichier `message_secret_2.txt` avec la clé " eiffel ".

1.2 Attaques du chiffrement

La *cryptanalyse* est la technique qui consiste à déchiffrer un texte chiffré sans posséder la clé de chiffrement. Une *attaque* est un processus visant à essayer de comprendre un message crypté particulier.

Le chiffre de César n'est pas utilisable en pratique, car il peut facilement être attaqué.

Question 1.6. *Brute Force*

Une attaque par force brute consiste à tester une à une toutes les clés de chiffrement possible sur un texte dans le but de trouver la bonne clé.

Dans le chiffrement de César combien de clé de chiffrement possible existe-t-il ? Implémentez une fonction `brute_force(s)` qui prend en entrée une chaîne de caractère, `s`, et qui renvoie la liste des chaînes obtenues en essayant de déchiffrer `s` avec chaque clé de chiffrement possible.

Essayez de déchiffrer la chaîne suivante : "wz␣dofowh␣eis␣borwbs␣b'owas␣dog␣zs␣xoapcb."

Question 1.7. *Analyse de fréquence - Lettre*

Une autre attaque, si on connaît la langue d'origine du texte, consiste à analyser la fréquence d'apparition des caractères dans le texte et de les comparer à la fréquence moyenne d'apparition des lettres de l'alphabet dans cette langue.

La fréquence d'apparition d'une lettre `l` dans un texte est le nombre $\frac{n_l}{n}$ avec n_l le nombre de fois que la lettre `l` apparaît dans le texte et n le nombre de lettres minuscules ASCII (hors symboles spéciaux) du texte.

Implémentez une fonction `freq_lettre(s,c)` qui prend en entrée une chaîne de caractère `s` et une lettre minuscule ASCII et qui renvoie la fréquence d'apparition de la lettre dans la chaîne.

Question 1.8. *Analyse de fréquence - Texte*

Utilisez la fonction `freq_lettre` pour implémenter une fonction `freq_texte(s)` qui prend en entrée une chaîne de caractère et renvoie une liste avec la fréquence de chaque lettre minuscule ASCII, dans l'ordre alphabétique.

Faîtes l'analyse de fréquence du texte du fichier `message_secret_3.txt` et comparez la aux fréquences théoriques d'apparition des lettres dans un texte en français :

E	17.26 %	P	3.01 %	A	8.40 %
G	1.27 %	S	8.08 %	V	1.32 %
I	7.34 %	B	1.06 %	N	7.13 %
F	1.12 %	T	7.07 %	Q	0.99 %
R	6.55 %	H	0.92 %	L	6.01 %
X	0.45 %	U	5.74 %	J	0.31 %
O	5.26 %	Y	0.30 %	D	4.18 %
K	0.05 %	C	3.03 %	W	0.04 %
M	2.96 %	Z	0.12 %		

Pour ce faire, vous pouvez exécuter les instructions suivantes dans la console.

```
f = open('messages/message_secret_3.txt','r')
texte = f.read()
freq = freq_texte(texte)
freq
```

Trouvez ainsi la clé utilisée sans faire d'attaque par force brute. Vous pouvez ensuite décrypter le message.

Question 1.9. *Analyse de fréquence avec renvoi de la clé - distance*

On définit la distance entre deux listes de même taille par la formule suivante, pour $L = [l_1, l_2, \dots, l_n]$ et $K = [k_1, k_2, \dots, k_n]$:

$$\sum_{i=1}^{n} (l_i - k_i)^2$$

Implémentez une fonction `distance(L1,L2)` qui prend en entrée deux listes de même taille et qui renvoie la distance entre ces deux listes.

Question 1.10. *Analyse de fréquence avec renvoi de la clé - clé probable*

Utilisez la fonction `distance` pour implémenter une fonction `cle(s)` qui prend en entrée une chaîne de caractère `s` et renvoie la clé la plus probable. La clé la plus probable est celle qui minimise la distance entre la fréquence du texte décryptée par cette clé est la fréquence théorique.

Vous pouvez essayer de retrouver le résultat de la question précédente par cette méthode.

Question 1.11. *Attaque du chiffre de Vigenère. Longueur de clé connue*

Lorsque l'on connaît la longueur de la clé utilisée pour chiffrer un texte par le chiffrement de Vigenère, une attaque possible consiste à se ramener au cas d'un chiffrement de César.

Supposons que la clé soit de longueur n, et soit $1 \leq i \leq n$, si on considère un caractère sur n (parmis les minuscules ASCII) du texte en commençant par le $i^{\text{ème}}$, on obtient un texte crypté par le même décalage, celui induit par le caractère i de la clé.

Une attaque par force brute reste possible, mais le nombre de textes généré est 26^n, ce qui peut rapidement devenir long à analyser manuellement. On préférera faire une analyse de fréquence indépendante pour chaque texte extrait afin d'en déduire le caractère i de la clé.

Implémentez une fonction `freq_vigenere(s,n)` qui prend en entrée un texte `s` et un entier `n` et qui renvoie la clé la plus probable de longueur `n` qui aurait servie à chiffrer le texte.

Vous pouvez essayer de décrypter le texte dans `message_secret_4.txt` en sachant que la clé a une longueur égale à `12`.

Question 1.12. *Calcul de la longueur de clé la plus probable.*

Pour trouver la longueur probable de la clé ayant servi à chiffrer un texte par le chiffre de Vigenère, on utilise l'indice de coïncidence. Cet indice calcule la probabilité, ayant pris deux lettres au hasard dans un texte, qu'elle soit égale. La formule du calcul de l'indice de coïncidence dans un texte est :

$$\mathrm{I}_c = \sum_{i=1}^{26} \frac{n_i(n_i-1)}{n(n-1)}$$

où n_i est le nombre d'apparitions de la $i^{\text{ème}}$ lettre de l'alphabet dans le texte, et n est le nombre de lettres minuscules ASCII (hors caractères spéciaux).

On note $\mathrm{I}_f = 0.074$ l'indice de coïncidence théorique d'un texte en langue française et $\mathrm{I}_a = 0.038$ l'indice de coïncidence théorique d'un texte dont les caractères sont tirés aléatoirement suivant une loi de probabilité uniforme.

On note une grande différence entre les deux indices. Si la clé est de taille n, le texte obtenu en prenant un caractère sur n devrait avoir un indice d'incidence proche de I_f. Si la clé n'est pas de taille n ,ce même texte devrait avoir un indice d'incidence plus proche de I_a.

Implémentez une fonction `meilleur_incidence` qui prend en entrée une chaîne de caractère et qui renvoie le premier entier n tel que l'indice d'incidence I_c du texte obtenu en prenant un caractère sur n est supérieur ou égal à 0.07.

Décryptez le texte du fichier `message_secret_5.txt`.

Question 1.13. *Questions ouvertes*

Non notée, pour les discussions autour d'un kebab.

Nous venons de voir des techniques de cryptanalyse permettant d'attaquer des chiffrement de César ou de Vigenère. Ces techniques de chiffrement sont donc à proscrire, car trop facilement attaquable. En cryptographie, on peut souvent rendre la tâche d'un attaquant plus difficile en combinant plusieurs techniques de chiffrement.

Est-ce que ces combinaisons de chiffrements rendent le texte chiffré plus difficile à attaquer ? Dans les trois cas, les clés utilisées dans chaque chiffrement peuvent être différente, et de tailles différentes pour Vigenère.

1. Un chiffrement de César sur un texte déjà chiffré avec le chiffrement de César.
2. Un chiffrement de Vigenère sur un texte chiffré avec le chiffrement de César.
3. Un chiffrement de Vigenère sur un texte chiffré avec le chiffrement de Vigenère.

II Stéganographie

La stéganographie est l'art de dissimuler de l'information dans un media qui semble anodin en premier approche. Elle se distingue en cela de la cryptographie. Lorsque quelqu'un regarde un contenu cryptographié il peut deviner qu'on lui cache quelquechose. En stéganographie le media en lui même n'éveille pas les soupçons.

Wikipédia - « la stéganographie consiste à enterrer son argent dans son jardin là où la cryptographie consisterait à l'enfermer dans un coffre-fort — cela dit, rien n'empêche de combiner les deux techniques, de même que l'on peut enterrer un coffre dans son jardin. »

2.1 Image dans une image

Notre premier cas d'étude est l'image en niveau de gris `flower.bmp`. Une image en noir et blanc y a été dissimulée.

Une image en niveau de gris est représentée par une matrice d'entiers entre 0 et 255 inclus. 0 correspond à *noir* et 255 à *blanc*. Une image en noir et blanc est représentée par une matrice de `False` et de `True`. `False` (0) correspond à *noir* et `True` (1) correspond à *blanc*.

Pour obtenir la matrice de l'image dissimulée il suffit de regarder la parité du niveau de gris de chaque pixel. Les pixels pair de `flower.bmp` correspondent à des pixels noirs de l'image dissimulée et les pixels impairs à des pixels blancs.

Par exemple la matrice suivante …

```
[[245,124,101],
 [9,230,114],
 [28,27,90]]
```

Dissimule la matrice :

```
[[True,False,True],
 [True,False,False],
 [False,True,False]]
```

Question 2.1. *Extraction de l'image en noir et blanc*

Implémentez une fonction `stega1inL(M)` qui prend en entrée une matrice `M` d'entiers et renvoie une matrice de même dimension contenant des booléens, obtenue en suivant le procédé décrit plus haut.

Une fois testée sur le petit exemple, vous pouvez obtenir l'image dissimulée dans `flower.bmp` à l'aide des instructions suivantes dans la console :

```
im_flower = Image.open('messages/flower.bmp')
mat_flower = np.asarray(im_flower).tolist()
mat_hidden = stega1inL(mat_flower)
img_hidden = Image.fromarray(np.asarray(mat_hidden))
img_hidden.show() # ou simplement img_hidden
```

2.2 Image en deux images

Notre second cas d'étude est l'image en niveau de gris `tv.bmp`. Une image en noir et blanc y a été dissimulée.

Pour obtenir la matrice de l'image dissimulée il faut couper l'image `tv.bmp` en deux moitiée pour obtenir deux matrices, `m1` et `m2` de taille 450×450. La matrice `m` de l'image dissimulée est de taille également 450×450, et le pixel `m[i][j]` de cette matrice est `True` si `m1[i][j]` $\neq$ `m2[i][j]` et `False` sinon.

Par exemple les matrices suivantes …

```
[[True,False,True],
 [True,False,False],
 [False,True,False]]

[[False,False,False],
 [True,False,True],
 [True,True,True]]
```

Dissimule la matrice :

```
[[True,False,True],
 [False,False,True],
 [True,False,True]]
```

Question 2.2. *Moit' Moit'*

Implémentez une fonction `moitie(M)` qui prend en entrée une matrice `M` de hauteur pair $2n$ et renvoie deux matrices de hauteur n correspondant aux deux moitiés de `M`.

Question 2.3. *XOR*

Implémentez une fonction `stegaXOR(M1,M2)` qui prend en entrée deux matrices `M1` et `M2` de booléens ayant la même dimension et qui renvoie une matrice de même dimension contenant des booléens, obtenue en suivant le procédé décrit plus haut.

Une fois testée sur le petit exemple, vous pouvez obtenir l'image dissimulée dans `tv.bmp` à l'aide des instructions suivantes dans la console :

```
im_tv = Image.open('messages/tv.bmp')
mat_tv = np.asarray(im_tv).tolist()
m1, m2 = moitie(mat_tv)
mat_hidden = stegaXOR(m1,m2)
img_hidden = Image.fromarray(np.asarray(mat_hidden))
img_hidden.show() # ou simplement img_hidden
```

2.3 Image dans un texte

Notre dernier cas d'étude est le texte du fichier `lipsum.txt`. Une image en niveau de gris y a été dissimulée. Le texte ne contient que des caractères minuscules ASCII, des espaces, des points et des retours à la ligne. Le principe utilisé est le suivant :

- chaque ligne du texte correspond à une ligne de l'image et contient le même nombre de `'u'` . Le $j^{\text{ème}}$ caractère `'u'` de la $i^{\text{ème}}$ ligne correspond au pixel de la $i^{\text{ème}}$ ligne, $j^{\text{ème}}$ colonne de l'image dissimulée.
- avant chaque lettre `u` il y a 2 lettres `c1` et `c2` qui représentent le niveau de gris du pixel correspondant. Si `c1` et la lettre numéro n de l'alphabet (entre 0 et 25) et si `c2` est la lettre numéro m, le niveau de gris du pixel est $n * 16 + m$.

Par exemple dans le passage `Lorem ipsoium dolor` se cache le pixel `oi`, c'est à dire $14 * 16 + 8 = 232$. Ainsi le texte suivant …

```
Lorem ipsoium dolor sit amet, consectetpjur adipiscing elit. Aliqipuam maximmpus est metnpu
s, at varipfus metckus tempglus vitae. Neiullam liu
ltrices dictldum lorem, at consectetliur metdeus aliqenuet nec. Sed ehu
ltricies nisi id magna vestibgfulocum, sit amet vestibgculejum enim fajku
cibpfus. Morbi dicthium mi iaut leo tempor dignissim. Napulla femau
```

Cache la matrice suivante :

```
[[232, 249, 143, 207, 223],
 [245,  42, 107,  72, 184],
 [179, 184,  52,  77,  71],
 [101, 226,  98,  73, 154],
 [245, 120, 128,  15, 192]]
```

Question 2.4. *Valeur de pixel*

Implémentez une fonction `hex(c1,c2)` qui prend en entrées deux caractères `c1` et `c2` et qui renvoie $n * 16 + m$ avec n et m les numéros de `c1` et `c2` respectivement.

Question 2.5. *Décodage*

Implémenter une fonction `lipsum(s)` qui prend une chaîne de caractère et renvoie la matrice cachée dans le texte.

Essayez de retrouver l'image cachée dans `lipsum.txt`.

2.4 Trésor

Section non noté.

Le fichier `nadine.txt` contient un texte qui dissimule une information.

Ou peut-être plusieurs …

Essayez de retrouver le *vrai* prénom de Nadine. Vous aurez alors trouver toutes les informations dissimulées.

A moins que … ? 😉