✎ : *à faire à la maison* 60 : *niveau spé* 95 : *niveau 5/2* ℹ : *plutôt pour les optants*

T.C.D'I. — TP PYTHON

ATTAQUE DU CODE DE VIGENÈRE

Au début de ce tp, on commencera par copier le fichier `Maupassant_Le_Horla.txt` dans son répertoire personnel. On définit les constantes globales

```
LE_HORLA = "M:/Maupassant_Le_Horla.txt"
TEST_1 = "M:/test_1.txt"
TEST_2 = "M:/test_2.txt"
```

et on travaillera avec ces trois fichiers-là. Au début du tp, les deux derniers fichiers ci-dessus ne sont pas encore créés, c'est normal.

On utilisera les fonctions suivantes, permettant de lire et d'écrire sur le disque dur. La première suppose que la chaîne de caractères `source` est le nom d'un fichier existant, le deuxième crée un fichier ayant le nom et l'adresse désignés par la chaîne de caractères `cible`. Si un tel fichier existe déjà, il est *écrasé sans aucune demande de confirmation*. On fera donc très attention à ce que l'on fait.

```
# Ouverture d'un fichier, récupérage des données sous la forme
# d'une liste d'entiers entre 0 et 255
def LireFichier(source) :
    with open(source, "r+b") as f :
        contenu = f.read()
    return [int(o) for o in contenu]

# Écriture dans un fichier. Les données sont sous la forme d'une
# liste de nombres entre 0 et 255
def ÉcrireFichier(cible, contenu) :
    with open(cible, "w+b") as f :
        f.write(bytes(contenu))
```

Partie I — Codage et décodage

Le code de Vigenère consiste à prendre un texte (c'est-à-dire une liste de nombres entre 0 et 255) et de décaler chacun d'eux d'une valeur c_i (modulo 256). Cette valeur dépend de la position i du nombre dans le texte. On utilise une clé, qui est une chaîne de caractères ; et la valeur de c_i est alors donnée par c_i = `ord(clé[i % len(clé)])`.

Exercice 1 — Que fait la fonction `ord(c)` ? On pourra consulter la documentation en ligne de Python pour répondre.

Exercice 2 — Décalage avec une clé répétée
L'utilisation de `clé[i % len(clé)]` permet de faire comme si la clé se répétait à l'infini, de sorte à être au moins aussi longue que le texte à crypter.

1) Écrire un programme `DécalageAvant(texte, clé)` qui étant donné un texte et une clé renvoie le nouveau texte obtenu en appliquant la transformation décrite ci-dessus.

2) Écrire le programme `DécalageArrière(texte, clé)` qui réalise la transformation inverse (pour « revenir » au texte de départ).

3) Écrire un programe `Décalage(texte, clé, sens=1)` qui réalise le décalage soit en avant soit en arrière, selon la valeur du paramètre optionnel `sens`.

Exercice 3

1) Déduire de ce qui précède le programme `Codage(source, cible, clé)` qui prend en argument une chaîne de caractères indiquant le fichier à crypter, une chaîne de caractères indiquant le nom du fichier dans lequel sauvegarder le résultat, et une clé, et qui fait ce qu'on imagine.

2) Écrire le programme réalisation la transformation inverse `Décodage(source, cible, clé)`.

3) Tester les deux programmes précédents, en utilisant `LE_HORLA` comme source et `TEST_1` comme cible pour le codage, et `TEST_1` comme source et `TEST_2` comme cible pour le décodage. Vérifier que le fichier final est identique au fichier de départ.

Partie II — Analyse fréquentielle

Exercice 4 — Que fait la fonction `chr(x)` ?

Exercice 5

1) Écrire un programme `Fréquences(texte)` qui étant donné un texte renvoie une liste `F`, de longueur 256, telle que `F[i]` est la fréquence du nombre i dans le `texte`.

2) Écrire le programme `PlusFréquent(fréquences)` qui étant donnée une liste de longueur 256, comme celle renvoyée par l'algorithme précédent, détermine l'indice de la case ayant la plus grande valeur dans cette liste.

3) Quel est le caractère le plus fréquent dans un texte écrit en français ? Qu'obtient-on pour le Horla ?

Exercice 6

1) Écrire un programme `FréquencesPartielles(texte, m, k)` qui étant donnés un texte et deux entiers naturels $m > 0$ et $k \geqslant 0$ renvoie une liste `F`, de longueur 256, telle que `F[i]` est la fréquence du nombre i parmi les éléments du texte dont l'indice est congru à k modulo m.

2) Est-ce que le caractère le plus fréquent est toujours le même si l'on regarde seulement les caractères d'indices k modulo m ?

Partie III — Détermination de la clé

Dans cette partie, on va voir qu'on peut retrouver la clé de cryptage à partir d'un texte codé : ceci prouve que la méthode de Vigenère n'est pas une bonne méthode pour crypter un document de type texte. L'idée est la suivante : certaines suites de caractères se répètent de nombreuses fois dans le texte de départ. Tellement souvent en fait que parfois, au moins si la clé n'est pas très longue, les répétitions tombent avec le même alignement sur la clé, et donneront donc la *même* succession de caractères cryptés.

Si l'on cherche les répétitions dans le texte crypté, il y a de très grandes chances pour que celle-ci proviennent de la situation décrite ci-dessus : ainsi l'écart de position entre les répétitions sera un multiple de la longueur de la clé.

Exercice 7 — Expliquer comment calculer la longueur de la clé en exploitant le phénomène décrit ci-dessus.

Exercice 8 — Écrire un programme qui calcule le PGCD d'une liste d'entiers naturels. On convient que le PGCD de la liste vide est 0.

Exercice 9

1) Écrire un programme `RépétitionsMotif(texte, pos, l)` qui étant donné un texte construit et renvoie la liste de tous les indices k `> pos` à partir desquels on peut lire la même succession de ℓ caractères que ce qu'on peut lire à partir de la position `pos`.

2) En déduire un programme `GrandesRépétitions(texte, l, DEB=100)` qui recherche les répétitions des motifs de longueur ℓ, parmi ceux qui démarrent sur un des `DEB` premiers caractères du texte, et qui renvoie la liste des écarts de positions entre le motif et ses répétitions.

3) Pourquoi se limite-t-on aux (par exemple) 100 premiers caractères du texte pour cherche des motifs qui se répètent ?

Exercice 10 — En déduire un programme `LongueurClé(texte)` qui étant donné un texte crypté essaie de déterminer la longueur de la clé utilisée.

Exercice 11

1) Écrire un programme `TrouverClé(texte)` qui étant donné un texte crypté tente de déterminer la clé utilisée.

2) Trouver un texte (pas trop long) dans le domaine public sur *I*nternet, et le sauvegarder dans un fichier `.txt`. Utiliser l'algorithme de Vigenère pour le crypter puis déposer le résultat dans le répertoire commun du réseau, sans dévoiler la clé évidemment.

3) Essayer de décrypter les textes déposés sur le réseau.

Solutions

Exercice 1
La fonction `ord(c)` renvoie le numéro associé au caractère c dans un certain système d'encodage. Par exemple, le caractère « e »a le numéro 101.

Exercice 2
1)

```
def DécalageAvant(texte, clé) :
    s = [0] * len(texte)
    for i in range(len(texte)) :
        s[i] = (texte[i] + ord(clé[i % len(clé)])) % 256
    return s
```

2)

```
def DécalageArrière(texte, clé) :
    s = [0] * len(texte)
    for i in range(len(texte)) :
        s[i] = (texte[i] - ord(clé[i % len(clé)])) % 256
    return s
```

3) On convient que la valeur `sens = -1` indiquera d'aller en arrière.

```
def Décalage(texte, clé, sens=1) :
    s = [0] * len(texte)
    for i in range(len(texte)) :
        s[i] = (texte[i] + sens * ord(clé[i % len(clé)])) % 256
    return s
```

Exercice 3
1)

```
def Codage(source, cible, clé) :
    texte_clair = LireFichier(source)
    texte_crypté = Décalage(texte_clair, clé)
    ÉcrireFichier(cible, texte_crypté)
```

2)

```
def Décodage(source, cible, clé) :
    texte_crypté = LireFichier(source)
    texte_clair = Décalage(texte_crypté, clé, sens=-1)
    ÉcrireFichier(cible, texte_clair)
```

3) Pour tester, on ouvre le fichier final dans le bloc note, et on compare avec le fichier de départ !

Exercice 4
La fonction `chr(x)` est la réciproque de `ord(c)` : connaissant le numéro d'un caractère, elle permet de retrouver celui-ci.

Exercice 5
1)

```
def Fréquences(texte) :
    Effectifs = [0] * 256
    for o in texte :
        Effectifs[o] += 1
    return [ eff/len(texte) for eff in Effectifs ]
```

2)

```
def PlusFréquent(fréquences) :
    ind = 0
```

```
    for i in range(1, 256) :
        if fréquences[i] > fréquences[ind] :
            ind = i
    return ind
```

3) Le caractère le plus fréquent est celui dont le numéro est 32. En calculant `chr(32)` dans la console, on voit qu'il s'agit du caractère d'espacement. Le « e » (minuscule) arrive en deuxième position.

Exercice 6
1)

```
def FréquencesPartielles(texte, m, k) :
    Effectifs = [0] * 256
    i = k ; nb = 0
    while i < len(texte) :
        Effectifs[texte[i]] += 1
        i += m ; nb += 1
    return [ eff/nb for eff in Effectifs ]
```

2) Le caractère le plus fréquent reste le même si on ne regarde qu'un caractère sur m dans le texte. C'est ce principe qui rend le cryptage de Vigenère très facile à casser.

Exercice 7
Si on a un certain nombre de répétitions d'un même motif, on calcule leur distance au motif initial, chacune est un multiple de la longueur de la clé. La longueur de la clé doit donc diviser le PGCD de tous ces nombres ; si l'on en dispose d'une quantité suffisamment importante, on va trouver *la* longueur de la clé.

Exercice 8
D'abord on calcule le PGCD de deux entiers :

```
def pgcd(a, b) :
    a_ = a ; b_ = b
    while b_ > 0 :
        (a_, b_) = (b_, a_ % b_)
    return a_
```

puis ensuite on itère la fonction précédente.

```
def PGCD(L) :
    D = 0
    for l in L :
        D = pgcd(l, D)
    return D
```

Exercice 9
1)

```
def RépétitionsMotif(texte, pos, l) :
    R = []
    for i in range(pos + 1, len(texte)) :
        if texte[pos:pos+l] == texte[i:i+l] :
            R.append(i)
    return R
```

2)

```
def GrandesRépétitions(texte, l, DEB=100) :
    décalages = []
    for pos in range(0, DEB) :
        print(pos)
        R = RépétitionsMotif(texte, pos, l)
        for r in R :
            décalages.append( r - pos )
    return décalages
```

3) On se limite aux motifs commençant sur les quelques premiers caractères du texte car l'exploration de toutes les possibilités de motifs serait trop longue.

Exercice 10 On se limite à chercher des répétitions de motifs de longueur au plus 5.

```
def LongueurClé(texte) :
    l = 5
    décalages = GrandesRépétitions(texte, l)
    D = PGCD(décalages)
    while D <= 1 :
        l -= 1
        décalages = GrandesRépétitions(texte, l)
        D = PGCD(décalages)
    return D
```

Exercice 11
1)

```
def TrouverClé(texte) :
    m = LongueurClé(texte)
    clé = ""
    for k in range(m) :
        p = PlusFréquent(FréquencesPartielles(texte, m, k))
        clé += chr((p - 32) % 256)
    return clé
```