TP 4: Codage

Partie 1

Ave Cesar (zud bdrzq)

On cherche à crypter un texte t de longueur n composé de caractères en minuscules (soit 26 lettres différentes) représentés par des entiers compris entre 0 et 25 ($0 \leftrightarrow a$, $1 \leftrightarrow b$, . . . $25 \leftrightarrow z$). Nous ne tenons pas compte des éventuels espaces. Ainsi, le texte **ecolepolytechnique** est représenté par le tableau suivant où la première ligne représente le texte, la seconde les entiers correspondants, et la troisième les indices dans le tableau t

e	c	o	l	e	p	o	l	y	t	e	c	h	n	i	q	u	e
4	2	14	11	4	15	14	11	24	19	4	2	7	13	8	16	20	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Nous utiliserons pour le codage des fonctions suivantes :

```
def tr(texte):
    l=[]
    n=len(texte)
    for i in range(n):
        l+=[ord(texte[i])-97]
    return(l)

def rev(l):
    t=""
    n=len(l)
    for i in range(n):
        t+=chr(l[i]+97)
    return(t)
```

La fonction **tr** permet de convertir le texte en liste d'entiers entre 0 et 25 et La fonction **rev** permet de faire l'opération inverse.

Codage de César

Ce codage est le plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (et même auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet **vers la gauche** de 1 ou plusieurs positions. Par exemple, en décalant les lettres de 1 position, le caractère a se transforme en z, le b en a, ... le z en y. Le texte **avecesar** devient donc **zudbdrzq**.

1. Que donne le codage du texte **maitrecorbeau** en utilisant un décalage de 5 ?
2. Comment déterminer le reste de la division euclidienne de a par b à l'aide dee Python ?
3. Compléter la fonction **codageCesar(t, d)** qui prend en arguments la liste t et un entier d ; pour qu'elle retourne une liste de même taille que t contenant le texte t décalé de d positions. (nous pourrons raisonner modulo 26)

```
def codageCesar(t,d) :
        n=len(t)
        res=[0 for i range(n)]
        for i in range(n) :
                res[i]=.........
        return(res)
```

4. Écrire la fonction **decodageCesar(t, d)** prenant les mêmes arguments mais qui réalise le décalage dans l'autre sens.

   Pour réaliser ce décodage, il faut connaître la valeur du décalage. Une manière de la déterminer automatiquement est d'essayer de deviner cette valeur. L'approche la plus couramment employée est de regarder la fréquence d'apparition de chaque lettre dans le texte crypté. En effet, la lettre la plus fréquente dans un texte suffisamment long en français est la lettre **e**.

5. Écrire la fonction **frequences(t0)** qui prend en argument une liste $t0$ de taille n représentant le texte crypté ; et qui retourne une liste de taille 26 dont la case d'indice i contient le nombre (ou la fréquence) d'apparition du nombre i dans t ($0 \leq i \leq 26$).

   Vous pourrez utiliser si nécessaire la liste :

```
l=[i for i in range(97,97+26)]
```

6. Afficher le diagramme des fréquences.

   Nous pourrons utiliser par exemple un code similaire à :

```
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(1,26,26)
plt.plot(x,x, 'ro')
plt.ylabel('fréquences')
plt.xlabel("lettres")
plt.show()
```

7. Ecrire une fonction **max** qui prend pour argument une liste et donne le plus grand élément de cette liste ainsi que la liste des indices correspondants.
8. Écrire la fonction **decodageAuto(t0)** qui prend en argument la liste $t0$ représentant le texte crypté ; et qui retourne le texte t d'origine (en calculant la clé pour que la lettre **e** soit la plus fréquente dans le texte décrypté).

   Identifier les limites de cette méthode.
   Je vous propose de lire le roman "La Disparition" de Georges Perec.

Attention, la suite est un bonus et est plus délicate et à lire uniquement si ce qui précède est parfaitement clair ! ! !

Partie 2

Codage de Vigenère

Au XVIème siècle, Blaise de Vigenère a modernisé le codage de César très peu résistant de la manière suivante. Au lieu de décaler toutes les lettres du texte de la même manière, on utilise un texte clé qui donne une suite de décalages. Prenons par exemple la clé **concours**. Pour crypter un texte, on code la première lettre en utilisant le décalage qui envoie le a sur le c (la première lettre de la clé). Pour la deuxième lettre, on prend le décalage qui envoie le a sur le o (la seconde lettre de la clé) et ainsi de suite. Pour la huitième lettre, on utilise le décalage a vers s, puis, pour la neuvième, on reprend la clé à partir de sa première lettre. Sur l'exemple **ecolepolytechnique** avec la clé concours, on obtient : (la première ligne donne le texte, la seconde le texte crypté et la troisième la lettre de la clé utilisée pour le décalage)

e	c	o	l	e	p	o	l	y	t	e	c	h	n	i	q	u	e
g	q	b	n	s	j	f	d	a	h	r	e	v	h	z	i	w	s
c	o	n	c	o	u	r	s	c	o	n	c	o	u	r	s	c	o

1. Donner le codage du texte **becunfromage** en utilisant la clé de codage **jean**.
2. Écrire la fonction **codageVigenere(l, c)** qui prend comme arguments une liste l représentant le texte à crypter, et une liste c donnant la clé servant au codage ; et qui retourne une liste contenant le texte crypté.

   Maintenant, on suppose disposer d'un texte t' assez long crypté par la méthode de Vigenère, et on veut retrouver le texte t d'origine. Pour cela, on doit trouver la clé c ayant servi au codage. On procède en deux temps :

   (a) Détermination de la longueur k de la clé c

   (b) Détermination des lettres composant c.

   La première étape est la plus difficile. On remarque que deux lettres identiques dans t espacées de $l \times k$ caractères (où l est un entier et k la taille de la clé) sont codées par la même lettre dans t'. Mais cette condition n'est pas suffisante pour déterminer la longueur k de la clé c puisque des répétitions peuvent apparaître dans t' sans qu'elles existent dans t. Par exemple, les lettres t et n sont toutes deux codées par la lettre h dans le texte crypté à partir de **ecolepolytechnique** avec **concours** comme clé. Pour éviter ce problème, on recherche les répétitions non pas d'une lettre mais de séquences de lettres dans t' puisque deux séquences de lettres répétées dans t, dont les premières lettres sont espacées par $l \times k$ caractères, sont aussi cryptées par deux mêmes séquences dans t'.

   Hypothèse extrêmement forte :
   dans la suite de l'énoncé, on ne considère que des séquences de taille 3 en supposant que toute répétition d'une séquence de 3 lettres dans t' provient exclusivement d'une séquence de 3 lettres répétée dans t.

   Ainsi, la distance séparant ces répétitions donne des multiples de k.
   La valeur de k est obtenue en prenant le PGCD de tous ces multiples. Si le nombre de répétitions est suffisant, on a de bonnes chances d'obtenir la valeur de k. On suppose donc que cette assertion est vraie.

3. Que fait le programme suivant :

```
def pgcd(a, b)
        while b
                a, b = b, a%b
        return a
```

4. Écrire la fonction **pgcdDesDistancesEntreRepetitions(l', i)** qui prend en argument la liste l' associée au texte crypté t' et un entier i ($0 \leq i < n - 2$) qui est l'indice d'une lettre dans t' ; et qui retourne le pgcd de toutes les distances entre les répétitions de la séquence de 3 lettres t[i], t[i + 1], t[i + 2] dans la suite du texte t[i + 3], t[i + 4], . . . ,t[n-1]. Cette fonction retourne 0 s'il n'y a pas de répétition.
5. Écrire la fonction **longueurDeLaCle(l')** qui prend en argument la liste l' associée au texte crypté ; et qui retourne la longueur k de la clé de codage.

La suite est un bonus

6. Une fois la longueur de la clé connue, décrire un algorithme permettant de retrouver chacune des lettres de la clé.
7. Écrire la fonction **decodageVigenereAuto(l')** qui prend en argument une liste l' associée au texte cripté ; et qui retourne le texte t d'origine.