TP 5 : Ave Cesar (zud bdrzq)

On s'intéresse dans ce TP à un des plus anciens procédés cryptographiques : le code de César, et son amélioration, le code de Vigénère.
Les parties I, III et V sont inspirées de l'épreuve de Polytechnique 2008 en filière PC.
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 les indices dans le tableau `t`, la deuxième représente le texte et la dernière les entiers correspondants aux lettres du texte.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
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

▶ **Question 0** Sur le site `http ://mpsi2-champo.fr`, télécharger le fichier `TP5.zip`, et sauvegarder tous les fichiers de l'archive dans votre répertoire de travail. Ouvrez dans Pyzo le fichier `TP5.py`. C'est ce fichier que vous devrez compléter durant tout le TP.
Afin de gagner du temps, ce fichier contient déjà le code de certaines fonctions, que vous ne chercherez pas à comprendre durant le TP

▶ I. 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`.

En Python, les caractères d'une chaîne de caractères sont codés à l'aide d'un encodage nommé Unicode, de sorte qu'à chaque caractère correspond un unique nombre.
La fonction Python qui permet d'obtenir le code associé à un caractère est la fonction `ord`, et la fonction qui réalise l'opération inverse est la fonction `chr`.

▶ **Question 1** Essayer ces deux fonctions, et comprendre leur fonctionnement et déterminer les codes associés aux lettres minuscules de `a` à `z`.
Écrire alors deux fonctions nommées `code` et `lettre` qui à une lettre minuscule associe un entier entre 0 et 25 ($a \leftrightarrow 0, \cdots, z \leftrightarrow 25$), et le contraire.

▶ **Question 2** Écrire la fonction `codageCesar(clair,d)` qui prend en arguments une chaîne `clair` et un entier d et qui retourne une chaîne contenant le texte `clair` décalé de d positions.
Vérifier que `codageCesar('avecesar',1)` renvoie `zudbdrzq`.

▶ **Question 3** Écrire de même la fonction `decodageCesar(crypte,d)` prenant les mêmes arguments, mais qui réalise le décalage dans l'autre sens.

▶ II. Manipulation de fichiers

Réaliser le décodage d'une chaîne nécessite de connaître le décalage qui a été utilisé lors de son codage. L'approche la plus couramment utilisée pour deviner ce décalage si on ne le connaît pas est de regarder la fréquence d'apparition de chaque lettre dans le texte crypté. En effet, dans un texte suffisamment long en français, la lettre e est généralement la plus fréquente, et de loin.

Pour manipuler de grands textes, nous aurons dans la suite besoin de demander à Python d'ouvrir, de lire et d'écrire des fichiers.
Le plus pénible est d'identifier les chemins d'accès à vos fichiers. Pour cela, dans l'explorateur de fichiers, placez vous dans le répertoire contenant les fichiers du TP, choisissez un fichier, clic droit/Propriétés. Copiez alors le contenu qui se trouve à la ligne «Emplacement» (au lycée ce sera sûrement de la forme `U :\prenom.nom\...`).
Collez alors ce chemin dans la variable (de type string) `chemin` du fichier-réponse, puis remplacez tous les \\ par des \\\\, et en ajoutant également un \\ à la fin de votre chaîne.
Par exemple `chemin = 'U :\\jules.cesar\\TP5\\'`

La démarche à suivre est un peu compliquée, et n'est pas à connaître, mais il est instructif de l'avoir déjà rencontrée.

Lecture et écriture de fichiers

Pour lire le contenu d'un fichier nommé test.txt on commence par utiliser
`fichier = open(chemin + 'test.txt','r')`, où le `'r'` (pour read) signifie qu'on souhaite juste lire le fichier, et pas l'écrire (auquel cas on utiliserait `'w'`, pour write).
Puis on peut obtenir le contenu des lignes une par une en itérant sur `fichier`. Par exemple

```
for ligne in fichier :
        print(ligne)
```

On n'oublie pas, une fois la lecture terminée, de fermer le fichier à l'aide de `fichier.close()`.
Une petite subtilité est que chaque ligne termine par un caractère spécial : le saut de ligne, qui indique justement qu'on souhaite passer à la ligne. Celui-ci est représenté dans les chaînes Python par `'\n'`.
Par exemple, `print('MPSI2\n Champollion\n Grenoble')` va s'afficher sur trois lignes.

La fonction `lectureComplete` vous donne un exemple complet de fonction qui lit un fichier (dont le nom est passé en paramètre sous forme d'une chaîne de caractères), et renvoie une seule chaîne de caractères obtenue par concaténation des lignes d'un fichier, ce qui nécessite la suppression des `\n` terminaux de chaque ligne. Vous pouvez chercher à comprendre son code si vous le souhaitez.
Dans la suite, vous vous servirez par exemple de `lectureComplete` *pour importer une fois pour toutes le contenu d'un fichier et le stocker dans une variable. Par exemple* `message = lectureComplete('test.txt')`

Sur le même principe, il est possible d'écrire d'écrire une chaîne de caractères dans un fichier ouvert en écriture en utilisant `fichier.write('la chaine a écrire')`.

▸ **Question 4** Écrire la fonction `nombreApparitions(texte)` qui prend comme paramètre une chaîne de caractères `texte`, et qui retourne une liste `L` de taille 26, telle que pour tout $i \in [\![0, 25]\!]$, `L[i]` contienne le nombre d'apparitions de la lettre numéro i dans `texte`.
À l'aide de cette fonction et de la fonction `lectureComplete` utilisée sur le fichier `notredame.txt`, qui contient les premiers chapitres du roman *Notre-Dame de Paris* de Victor Hugo, estimer les fréquences d'apparition de chaque lettre dans un texte français (en supposant donc que *Notre-Dame* est représentatif des textes en français).

▸ **Question 5** Écrire une fonction `decalageProbable`, qui à partir d'un texte crypté «devine» le décalage le plus probable et retourne la valeur de ce décalage.

Vous pouvez alors automatiquement essayer votre code en utilisant la fonction `decodageAuto` qui se trouve dans le fichier réponse. Elle attend deux paramètres : le nom du fichier source qui contient le texte crypté d'origine, et le nom du fichier `destination` dans lequel vous souhaitez enregistrer le texte décrypté.
Une fois de plus, si vous êtes curieux, vous pouvez bien entendu chercher à comprendre le fonctionnement de cette fonction, mais ce n'est pas indispensable.
Tester cette fonction sur le fichier `cesar.txt`

► **Question 6** Cette méthode d'analyse fréquentielle possède tout de même ses limites. Pour le constater, utiliser la fonction `decodageAuto` sur le fichier `disparition.txt` qui contient un extrait du roman *La disparition* de George Perec.
Retrouver de vous-même le bon décalage à l'aide de la fonction `decalageManuel` qui essaie tous les décalages possibles sur l'une des lignes du fichier.
Décrypter alors l'intégralité du fichier à l'aide de la fonction `decodageManuel` qui fonctionne comme `decodageAuto`, mais nécessite un troisième paramètre : le décalage que l'on souhaite utiliser pour décrypter.
Une fois le texte décrypté, savez-vous expliquer pourquoi `decodageAuto` n'a pas fonctionné ? La fonction `nombreApparitions` peut sûrement vous aider à éclaircir ce mystère. ...

► III. Codage de Vigénère

Au XVI^{ème} siècle, Blaise de Vigenère a modernisé le codage de César, dont nous venons de constater qu'il n'est pas si difficile à décoder, 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), autrement dit on décale de 24. 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

► **Question 7** Écrire la fonction `codageVigenere(clair,cle)` qui prend comme arguments deux chaînes : le texte `clair` qu'on souhaite crypter et la `cle` que l'on veut utiliser pour cela, et qui retourne une chaîne formée du texte crypté. *On pourra faire appel à la fonction* `codageCesar`.

► **Question 8** Écrire également une fonction `decodageVigenere(crypte,cle)` qui réalise l'opération inverse.

► IV. Déchiffrage de Vigénère : la méthode moderne

Dans cette partie et la suivante, 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.
Le déchiffrage de Vigénère se fait alors en deux étapes :

1. la détermination de la longueur k de la clé c
2. la détermination des lettres composant c.

Si un texte de n lettres contient n_0 fois la lettre `a`, n_1 fois la lettre `b`, etc, son indice de coïncidence est

$$I = \sum_{i=0}^{25} \frac{(n_i)(n_i - 1)}{n(n-1)}.$$

Un calcul facile de probabilités prouve qu'il s'agit de la probabilité pour que deux lettres prises au hasard dans le texte soient les mêmes.
Sur un texte en français, une analyse de la fréquence d'apparition de chaque lettre prouve que l'indice de coïncidence est d'environ 0.078.
En revanche, pour un texte aléatoire, la probabilité que deux lettres soient les mêmes est $\frac{1}{26} = 0.0385$.

▸ **Question 9** Écrire une fonction `indiceCoincidence` qui calcule l'indice de coïncidence d'une chaîne.

Pour déterminer la longueur de la clé de cryptage, on utilise le fait suivant : pour un texte crypté avec la méthode de César, l'indice de coïncidence du texte crypté est le même que celui du texte en clair, toutes les lettres ayant subi le même décalage.

Pour déterminer la longueur de la clé, on peut, pour chaque entier k (disons inférieur à la moitié de la longueur du texte), commencer par découper le message crypté en k textes plus petits : le premier est formé de la première lettre, de la $(k+1)^{\text{ème}}$, de la $(2k+1)^{\text{ème}}$, etc.
Le second est formé de la deuxième, la $(k+2)^{\text{ème}}$, ...
Si k est un multiple de la longueur de la clé, ces textes seront tous obtenus à partir d'une portion de texte initial par un chiffrage de César. Et auront donc tous un indice de coïncidence proche de celui du texte initial, donc de 0.078 (sauf texte de départ atypique).
En revanche, si k n'est pas un multiple de la clé, l'indice de coïncidence risque d'être beaucoup plus proche de celui d'un texte aléatoire, donc de 0.0385.
On peut donc considérer que k est un multiple de la longueur de la clé lorsque tous les sous-textes du texte crypté ont un indice de coïncidence qui dépasse un certain seuil, par exemple 0.055.

▸ **Question 10** Écrire une fonction `longueurCle` qui détermine la longueur de la clé à l'aide de la méthode décrite ci-dessus.

Une fois la longueur k de la clé connue, la suite est assez facile : on détermine la première lettre de la clé en procédant par analyse fréquentielle sur la sous-chaîne du texte crypté formée des lettres d'indices $0, k, 2k, 3k, \ldots$
Puis on détermine la seconde lettre de la clé à partir des lettres d'indices $1, k+1, 2k+1, \ldots$.

▸ **Question 11** En utilisant les fonctions de la première partie, écrire une fonction `cleVigenere` qui à partir d'un texte crypté et de la longueur de la clé de cryptage, retrouve cette clé de cryptage.
Utiliser alors cette fonction sur le fichier `recherche.txt` afin de déterminer la clé de cryptage qui a été utilisée.

Vous pouvez alors utiliser directement la fonction `decodageAutoVigenere` du document réponse pour décrypter le texte de `recherche.txt` sans effort.

▸ V. Déchiffrage de Vigénère : la méthode historique

Dans cette dernière partie, on présente la méthode découverte indépendamment par Charles Babbage et Friedrich Kasiski dans les années 1850–60 (donc antérieure à la méthode présentée dans la partie IV, qui date des années 1920).
Comme pour la méthode moderne, il s'agit de déterminer la longueur de la clé, puis d'en déduire la clé par analyse fréquentielle.

L'idée est que deux lettres identiques dans t espacées de $\ell \times k$ caractères, où ℓ est un entier et k la taille de la clé, sont codées par la même lettre dans t', car elles ont subi le même décalage. 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 $\ell \times k$ caractères, sont aussi cryptées par deux mêmes séquences dans t'.

Dans la suite de l'énoncé, on ne considère que des séquences de 3 lettres en supposant que toute répétition d'une séquence de 3 lettres dans t' ne peut être le fruit du hasard et provient nécessairement d'une séquence de 3 lettres répétée dans t. Ainsi, les distances séparant ces répétitions sont nécessairement des multiples de k. Le PGCD de ces distances doit alors être égal à k, ou au moins à un multiple de k.
Notons que ce second point n'est pas problématique, si pour un texte crypté avec la clé `concours`, on cherche une clé de longueur 14 et non 7, l'analyse fréquentielle nous donnera `concoursconcours` comme clé, ce qui permet de décoder le message de départ.

▸ **Question 12** Écrire la fonction `pgcdDesDistancesEntreRepetitions`(*crypte*, *i*) qui prend en argument le texte crypté t' et un entier i qui est l'indice d'une lettre dans `crypte` ; et qui retourne le `pgcd` de toutes les distances entre les répétitions de la séquence de 3 lettres ⟨`crypte`[i], `crypte`[$i+1$], `crypte`[$i+2$]⟩ dans la suite du texte ⟨`crypte`[$i+3$], `crypte`[$i+4$], . . . `crypte`[`len(crypte)` -1]⟩.
Cette fonction retournera 0 si la séquence en question n'est pas répétée dans `crypte`.
On pourra utiliser la fonction `gcd` *du module* `math`*, qui calcule le* ***pgcd*** *de deux nombres.*

▸ **Question 13** En utilisant la fonction de la question précédente, et en faisant varier i, écrire une fonction `longueurCle2(crypte)` qui prend en argument le texte crypté et qui retourne la longueur k de la clé de codage.
Une fois la clé obtenue, on procède comme à la partie IV par analyse fréquentielle, et on peut donc réutiliser la fonction de la question 11.
Essayer alors de décoder `recherche.txt` à l'aide de la fonction `decodageAutoVigenere2`.

Malheureusement, comme vous avez du le constater, cette approche ne fonctionne pas bien, et pour cause : nous avons supposé qu'aucune des répétitions de trois caractères n'était le fruit du hasard, alors qu'en pratique, cela se produit (surtout si le texte est long). Et comme en pratique, ces répétitions se produisent à des emplacements aléatoires, il est fréquent d'obtenir une chaîne de caractères répétée à des distances premières entre elles (= de pgcd égal à 1). Auquel cas la fonction `longueurCle2` va retourner 1.
Un moyen d'y remédier est de modifier `longueurCle2` pour qu'elle nous donne plutôt une liste de longueurs de clés possibles. Par exemple en comptant combien de fois `pgcdDesDistancesEntreRepetitions` a retourné 2, a retourné 3, etc, et retourne le pgcd des distances qui est le plus souvent apparu, hormis 1.

▸ **Question 14** Modifier la fonction `longueurCle2` comme décrit ci-dessus. Essayer alors de nouveau la fonction `decodageAutoVigenere2` sur `recherche.txt`.