TP noté

Rendre le document réponse (uniquement) à la fin de la séance.

Codage et cryptographie

1 Codage de César

On cherche à crypter un texte `t` composé de caractères en majuscules (soit 26 lettres différentes) représentés par des entiers compris entre 1 et 26 (1 pour A...) ; l'espace et les tirets étant remplacés dans le texte par @ et représenté par 0.

Le codage de César reste assez rudimentaire. 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 droite de 1 ou plusieurs positions. Par exemple, en décalant les lettres de 1 position, le caractère `@` se transforme en `A`, le `A` en `B`, ..., le `Z` en `@`. Le texte `LA@POESIE` devient donc `MBAQPFTJF`.

1.1 Codage

Pour représenter les lettres par des nombres, nous allons utiliser le code `ASCII` que l'on peut obtenir par des commandes déjà existantes. Tester par exemple `chr(64)`, `ord('B')` et `chr(63)`.

Question 1. Écrire une fonction `code(t)` qui prend en argument une chaîne de caractères en majuscule (et @) `t` et la représente en une liste de nombres compris entre 0 et 26 selon le principe expliqué ci-dessus. **Compléter** le document-réponses avec `t1='NOM@PRENOM'` (en remplaçant `NOM` et `PRENOM` par votre nom et votre prénom sans accent et en majuscule).

Question 2. Écrire une fonction `decode(l)` qui prend en argument une liste de nombres compris entre 0 et 26 et ressort la chaîne de caractères en majuscule associée.

Question 3. Décoder le code secret fourni dans le tableau donné en fin de sujet et associé au codage de votre texte `'NOM@PRENOM'` et découvrez quel poète vous est associé (**répondre sur le document-réponses**).

Question 4. Réfléchir au fonctionnement de l'opérateur `%`. On pourra par exemple tester `31%30`. *Bizarre cette question, ça pourrait peut-être servir pour la suite.*

Question 5. Écrire une fonction `codageCesar(t,d)` qui prend en argument une chaîne de caractères `t` et un entier `d` et fournit en sortie la chaîne décalée de `d` (vers la droite). *Pour décaler certaines lettres, il faut bien évidemment repartir depuis le début de l'alphabet.*

Question 6. Complétez le document-réponses en codant la phrase suivante : `MON@ANNIVERSAIRE@EST@EN@MOIS` en remplaçant `MOIS` par votre mois de naissance. Vous utiliserez pour `d` la valeur $12 \times n$ où n est le "numéro" de votre mois de naissance (1 pour janvier, 2 pour février, etc.).

Question 7. Comment obtenir le décodage lorsque l'on connaît `d` ?

1.2 Décodage

Pour réaliser ce décodage (sans connaître au préalable `d`), il faut déterminer la valeur du décalage. L'approche la plus couramment employée pour cela 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.

Question 8. Écrire une fonction `NombreApparitions(t)` qui prend en argument une chaîne de caractères `t` et fournit en sortie la liste des nombres d'apparitions de chaque lettre (en commençant par @ puis `A` ...). On veillera à ne parcourir `t` qu'une seule fois. Pour ce faire, on pourra :

- déclarer initialement une liste `occurrence = [0]*27` (l'indice de la liste correspondant au numéro de la lettre),
- remplir cette liste au fur et à mesure de l'avancement dans la chaîne de caractères.

Question 9. Complétez le document-réponses en donnant le nombre d'apparition de la première lettre de votre nom dans le texte du fichier `nombre_apparition.txt`. Le script `lire_texte.py`, à télécharger ainsi que le fichier `nombre_apparition.txt`, est disponible sur votre *cahier-de-prépa*. Pour lire le fichier vous écrirez simplement les commandes suivantes :

```
from lire_texte import * ## le fichier lire_texte.py doit Etre dans votre dossier de travail

res = lire_texte('nombre_apparition.txt') ## res est la chaIne de caractEres souhaitEe
```

Question 10. Écrire une fonction `decodeAuto(t)` qui prend en argument une chaîne de caractères représentant le texte crypté `t` par la méthode César et retourne le texte d'origine (on calculera le décalage `d` en supposant que la lettre E est la plus fréquente dans le texte décrypté).

Question 11. Décoder le poème qui est fourni dans le fichier `a_decoder.txt` (lire le fichier avec la fonction `lire_texte`). Vous stockerez le résultat dans une variable `resultat` et donnerez, sur le document-réponses, les 20 caractères du poème à décoder correspondant à `resultat[ 10*mois :  10*mois+20 ]` où `mois` représente votre mois de naissance.

2 Codage de Vigénère

2.1 Codage

Au XVI-ème siècle, Blaise de Vigenère a modernisé le codage de César qui s'avère, comme vu précédemment, très peu résistant. 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 sur le S, puis, pour la neuvième, on reprend la clé à partir de sa première lettre.

Question 12. Écrire une fonction `codeVigenere(t,CLE)` qui prend en argument une chaîne de caractères `t` et une clé `CLE` et retourne le texte crypté selon le principe expliqué ci-dessus.

Tester avec `t3='JE@VAIS@PEUT@ETRE@INTEGRER@L@ECOLE@POLYTECHNIQUE'` et `CLE1='NOM@PRENOM'` (bien entendu, vous remplacez `NOM` et `PRENOM` par les votres) ; donner le résultat sur le document réponse.

2.2 Décodage

Maintenant, on suppose disposer d'un texte `u` assez long et crypté par la méthode de Vigenère, et on veut retrouver le texte `t` d'origine. Pour cela, on doit trouver la clé `CLE` ayant servi au codage. On procède en deux temps : détermination de la longueur `k` de la clé puis détermination des lettres composant cette clé.

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 `u`. Mais cette condition n'est pas suffisante pour déterminer la longueur `k` de la clé puisque des répétitions peuvent apparaître dans `u` sans qu'elles existent dans `t`. Par exemple, les lettres P et O de `POLYTECHNIQUE` sont toutes deux codées par F dans l'exemple précédent.

Pour éviter ce problème, on recherche les répétitions non pas d'une lettre mais de séquences de lettres dans `u`, 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 `u`.

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 u 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 les 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.

Question 13. Dans un premier temps, on suppose que l'on connait la clé de chiffrage. Écrire la fonction `decodageVigenereManu(u,CLE)` où u est le texte crypté et CLE la clé de chiffrage.

Question 14. Tester cette fonction en décodant le fichier `'ci.txt'` où i représente votre mois de naissance (codé avec la clé `'POEME'`). Quel est ce texte (demandez à Google si vous ne trouvez pas). Répondre sur le document-réponses.

On s'intéresse maintenant au décodage automatique qui est plus complexe !

Question 15. Écrire une fonction `pgcd(a,b)` qui calcule le PGCD des deux entiers strictement positifs a et b.

Question 16. Écrire une fonction `pgcdDesDistancesEntreRepetitions(u,i)` qui prend en argument le texte crypté u et un entier i ($0 \leq i < n-2$) qui est l'indice d'une lettre dans u ; et qui retourne le pgcd de toutes les distances entre les répétitions de la séquence de 3 lettres `u[i]`, `u[i+1]` et `u[i+2]` dans la suite du texte `u[i+3]`, `u[i+4]`... Cette fonction retourne 0 s'il n'y a pas de répétition.

Tester avec `u='AEEETDFEEEYDHSJUEEEJHSDJVK'` et i=1.

Question 17. Écrire une fonction `longueurDeLaCle(u)` qui prend en argument le texte crypté u et qui retourne la longueur k de la clé de codage (en faisant les hypothèses expliquées ci-dessus).

Tester avec l'exemple précédent.

Question 18. Donner le nombre d'opérations réalisées par la fonction `longueurDeLaCle` en fonction de la longueur n de u ? (On ne comptera que le nombre d'appels à la fonction `pgcd`).

Question 19. Une fois la longueur de la clé connue, donner une idée d'algorithme permettant de retrouver chacune des lettres de la clé. (Il s'agit de décrire assez précisément l'algorithme plutôt que d'écrire le programme).

Question 20. Érire la fonction `decodageVigenereAuto(u)` qui prend en argument le tableau u représentant le texte crypté ; et qui retourne le texte t d'origine. (On n'hésitera pas à recopier des parties de texte dans des tableaux intermédiaires).

Tester avec le texte dans le fichier `EssaiVigenere.txt`.

Codage de 'NOM@PRENOM'	Poète mystère à décoder
[2, 5, 14, 15, 9, 20, 0, 1, 12, 2, 1, 14, 5]	[16, 9, 5, 18, 18, 5, 0, 4, 5, 0, 18, 15, 14, 19, 1, 18, 4]
[2, 8, 1, 25, 1, 0, 13, 15, 8, 1, 13, 5, 4, 0, 20, 1, 8, 1]	[3, 1, 18, 4, 25, 0, 2]
[2, 15, 21, 20, 1, 12, 5, 2, 0, 13, 15, 8, 1, 13, 5, 4]	[10, 15, 1, 3, 8, 9, 13, 0, 4, 21, 0, 2, 5, 12, 12, 1, 25]
[3, 1, 12, 12, 5, 20, 0, 10, 15, 19, 5, 16, 8]	[22, 3, 20, 15, 18, 0, 8, 21, 7, 15]
[3, 1, 22, 1, 9, 12, 12, 5, 19, 0, 1, 18, 20, 8, 21, 18]	[3, 8, 1, 18, 12, 5, 19, 0, 2, 5, 1, 21, 4, 5, 12, 1, 9, 18, 5]
[3, 5, 18, 9, 5, 26, 0, 1, 18, 14, 1, 21, 4]	[13, 1, 9, 20, 18, 5, 0, 7, 9, 13, 19]
[3, 8, 5, 22, 1, 14, 3, 8, 5, 0, 16, 1, 21, 12, 0, 13, 1, 18, 9, 5]	[1, 18, 20, 8, 21, 18, 0, 18, 9, 13, 2, 1, 21, 4]
[3, 9, 18, 3, 9, 21, 0, 5, 4, 21, 1, 18, 4, 0, 7, 1, 2, 18, 9, 5, 12]	[16, 1, 21, 12, 0, 22, 5, 18, 12, 1, 9, 14, 5]
[3, 15, 14, 14, 1, 14, 0, 13, 1, 20, 8, 9, 5, 21]	[7, 21, 9, 12, 12, 1, 21, 13, 5, 0, 1, 16, 16, 15, 12, 9, 14, 1, 9, 18, 5]
[3, 15, 18, 14, 9, 12, 12, 5, 20, 0, 16, 8, 9, 12, 15, 13, 5, 14, 5]	[16, 1, 21, 12, 0, 5, 12, 21, 1, 18, 4]
[4, 5, 12, 22, 1, 12, 0, 25, 1, 14, 14, 9, 3, 11]	[12, 15, 21, 9, 19, 0, 1, 18, 1, 7, 15, 14]
[6, 1, 22, 18, 1, 25, 0, 5, 12, 12, 9, 15, 20]	[16, 9, 5, 18, 18, 5, 0, 4, 5, 0, 18, 15, 14, 19, 1, 18, 4]
[7, 5, 6, 6, 18, 15, 25, 0, 5, 20, 8, 1, 14]	[3, 1, 18, 4, 25, 0, 2]
[7, 9, 3, 17, 21, 5, 12, 0, 20, 8, 5, 15]	[10, 15, 1, 3, 8, 9, 13, 0, 4, 21, 0, 2, 5, 12, 12, 1, 25]
[7, 9, 18, 1, 21, 12, 20, 0, 3, 1, 13, 9, 12, 12, 5]	[22, 3, 20, 15, 18, 0, 8, 21, 7, 15]
[8, 5, 12, 12, 15, 3, 15, 0, 7, 1, 18, 1, 2, 25, 0, 24, 1, 22, 9, 5, 18]	[3, 8, 1, 18, 12, 5, 19, 0, 2, 5, 1, 21, 4, 5, 12, 1, 9, 18, 5]
[8, 5, 18, 22, 25, 0, 13, 1, 20, 20, 8, 9, 5, 21]	[13, 1, 9, 20, 18, 5, 0, 7, 9, 13, 19]
[10, 1, 14, 0, 13, 1, 20, 8, 9, 5, 21]	[1, 18, 20, 8, 21, 18, 0, 18, 9, 13, 2, 1, 21, 4]
[11, 8, 1, 3, 8, 9, 4, 26, 5, 0, 14, 9, 11, 15, 12, 15, 26]	[16, 1, 21, 12, 0, 22, 5, 18, 12, 1, 9, 14, 5]
[12, 1, 13, 2, 5, 18, 20, 0, 1, 4, 5, 14, 15, 18]	[7, 21, 9, 12, 12, 1, 21, 13, 5, 0, 1, 16, 16, 15, 12, 9, 14, 1, 9, 18, 5]
[12, 1, 13, 2, 5, 18, 20, 0, 2, 1, 16, 20, 9, 19, 20, 5]	[16, 1, 21, 12, 0, 5, 12, 21, 1, 18, 4]
[12, 1, 14, 15, 5, 0, 1, 18, 20, 8, 21, 18]	[12, 15, 21, 9, 19, 0, 1, 18, 1, 7, 15, 14]
[12, 5, 0, 2, 1, 9, 12, 0, 12, 1, 21, 18, 5]	[16, 9, 5, 18, 18, 5, 0, 4, 5, 0, 18, 15, 14, 19, 1, 18, 4]
[12, 5, 0, 12, 1, 25, 0, 12, 21, 2, 9, 14]	[3, 1, 18, 4, 25, 0, 2]
[12, 5, 0, 13, 1, 18, 3, 8, 1, 14, 4, 0, 1, 18, 20, 8, 21, 18]	[10, 15, 1, 3, 8, 9, 13, 0, 4, 21, 0, 2, 5, 12, 12, 1, 25]
[12, 5, 0, 13, 5, 8, 1, 21, 20, 5, 0, 7, 15, 21, 12, 22, 5, 14]	[22, 3, 20, 15, 18, 0, 8, 21, 7, 15]
[13, 5, 12, 15, 21, 0, 1, 12, 5, 24, 9, 19]	[3, 8, 1, 18, 12, 5, 19, 0, 2, 5, 1, 21, 4, 5, 12, 1, 9, 18, 5]
[13, 5, 14, 7, 21, 25, 0, 13, 1, 20, 5, 15]	[13, 1, 9, 20, 18, 5, 0, 7, 9, 13, 19]
[14, 15, 25, 1, 12, 5, 20, 0, 1, 12, 5, 24, 9, 19]	[1, 18, 20, 8, 21, 18, 0, 18, 9, 13, 2, 1, 21, 4]
[16, 18, 15, 5, 18, 5, 19, 0, 18, 15, 13, 1, 9, 14]	[16, 1, 21, 12, 0, 22, 5, 18, 12, 1, 9, 14, 5]
[17, 21, 5, 12, 12, 9, 5, 14, 0, 13, 1, 20, 8, 5, 15]	[7, 21, 9, 12, 12, 1, 21, 13, 5, 0, 1, 16, 16, 15, 12, 9, 14, 1, 9, 18, 5]
[18, 1, 12, 12, 9, 5, 18, 0, 4, 21, 0, 2, 1, 20, 25, 0, 1, 21, 7, 21, 19, 20, 9, 14]	[16, 1, 21, 12, 0, 5, 12, 21, 1, 18, 4]
[18, 15, 21, 1, 21, 12, 20, 0, 3, 12, 1, 18, 5, 14, 3, 5]	[12, 15, 21, 9, 19, 0, 1, 18, 1, 7, 15, 14]
[19, 21, 9, 19, 19, 5, 0, 4, 5, 0, 19, 1, 9, 14, 20, 5, 0, 3, 12, 1, 9, 18, 5, 0, 8, 21, 7, 15]	[16, 9, 5, 18, 18, 5, 0, 4, 5, 0, 18, 15, 14, 19, 1, 18, 4]
[22, 9, 14, 1, 25, 0, 16, 1, 21, 12]	[3, 1, 18, 4, 25, 0, 2]