TP1 - PROJET PYTHON

Consignes : Cette activité se compose d'une première partie guidée, suivie de suggestions. Il est conseillé de traiter en entier la partie guidée avant de choisir une ou plusieurs suggestions à explorer.

L'objectif de cette activité est d'étudier des chiffres simples et de les appliquer à des fichiers textuels.

Définition. Un chiffre est une *méthode cryptographique* qui permet de transformer un texte (que l'on veut garder secret) en un autre texte (qu'on peut rendre public) à l'aide d'une clef (une information secrète). Le texte obtenu doit être incompréhensible pour qui ne connait pas la clef. Une personne qui possède la clef doit pouvoir *dé-chiffrer* (facilement) le texte.

1 Préliminaires : Manipulation de fichier

Dans cette activité, on va appliquer un chiffre à un fichier en définissant deux opérations **bijectives** de chiffrement et de déchiffrement sur les chaînes de caractères puis en appliquant cette opération **ligne-par-ligne** à des fichiers `.txt` existants.

La construction `with open(nom, "r") as fichier` permet d'ouvrir le fichier `nom` en lecture. On peut ensuite itérer sur `fichier.readlines()` pour lire, une par une, les lignes du fichiers (qui finissent toutes par `"\n"`)

La construction `with open(nom, "w") as fichier` permet d'ouvrir le fichier `nom` en écriture. On peut ensuite appeler `fichier.write(s)` pour ajouter la chaînes de caractères `s` à ce fichier.

On donne un modèle[1] pour la manipulation de fichiers dans cette activité :

```
def identite(s : str) -> str :
  """ Renvoie la chaine s telle quelle """
  return s

def identite_texte(nom : str) -> None :
  """ Precondition : <nom>.txt est un fichier existant
  recopie le contenu du fichier <nom>.txt dans <nom>-copie.txt """
  with open(nom + ".txt", "r") as source :
      with open(nom + "-copie.txt", "w") as destination :
          ligne : str
          for ligne in source.readlines() :
              destination.write(identite(ligne))
```

La fonction `identite_texte` recopie un fichier `<nom>.txt` dans un fichier `<nom>-copie.txt`. On pourra s'en servir comme modèle pour écrire les fonctions de chiffrement et de déchiffrement de fichiers (en remplaçant, entre autres, la fonction `identite` par une fonction de chiffrement ou déchiffrement).

Pour essayer les différents chiffres proposés dans cette activité, on téléchargera un fichier `.txt` à titre d'exemple (par exemple, un roman du *Projet Gutemberg*[2]).

2 Partie Guidée : Chiffre de César

Le chiffre de César est basé sur un *décalage* des places des lettres dans l'alphabet. La clef n (entier relatif) est la valeur du décalage. Par exemple pour un décalage de 3, la lettre `A` devient `D`, la lettre `Y` devient `B`, ...

Ainsi le mot `"Cesar"` est chiffré avec la clef 3 en `"Fhvdu"`.

Pour déchiffrer un texte chiffré avec la clef n, il suffit de décaler de $-n$ places chaque lettre du texte.

[1] disponible sur Moodle

[2] un tel fichier est disponible sur Moodle

Question 1. On ne s'intéresse ici qu'à la transformation des lettres romaines majuscules et minuscules (on ne transformera pas les chiffres, les espaces ou les caractères spéciaux comme é ou !).

Ecrire deux fonctions `est_minuscule` et `est_majuscule` qui décident, respectivement, si un caractère est une des 26 lettres romaines minuscules, ou une des 26 lettres romaines majuscules.

On pourra utiliser la fonction `ord` qui renvoie le code UTF8 d'un caractère, et se souvenir que les lettres romaines minuscules occupent des codes consécutifs, dans l'ordre de l'alphabet (de même pour les majuscules).

```
assert est_majuscule("C")
assert not est_minuscule("C")
assert est_minuscule("c")
assert not est_majuscule("c")
assert not est_minuscule(" ")
assert not est_majuscule(" ")
```

Question 2. Ecrire une fonction `caractere_decale` qui prend en entrée un caractère `c` (n'importe lequel, pas forcément une lettre romaine) et un entier `n` et qui renvoie le caractère obtenu par un décalage de `n` places dans l'alphabet de `c` si `c` est une lettre romaine (majuscule ou minuscule), et `c` sinon.

On pourra utiliser la fonction `ord` et son inverse `chr`.

```
assert caractere_decale("a", 0) == "a"
assert caractere_decale("a", 3) == "d"
assert caractere_decale("A", 3) == "D"
assert caractere_decale("V", 8) == "D"
assert caractere_decale(" ", 3) == " "
```

Question 3. Ecrire une fonction `ligne_chiffre_cesar` qui prend en entrée une chaîne de caractères `s` et un entier `n` et qui renvoie la chaîne obtenue en appliquant le chiffrement de César de clef `n` à `s`.

```
assert ligne_chiffre_cesar("Bonjour LU1IN011", 3) == "Erqmrxu OX1LQ011"
assert ligne_chiffre_cesar("Bonjour LU1IN011", 0) == "Bonjour LU1IN011"
```

Question 3. Ecrire une fonction `ligne_dechiffre_cesar` qui prend en entrée une chaîne de caractères `s` et un entier `n` et qui renvoie la chaîne obtenue en appliquant le déchiffrement de César de clef `n` à `s`.

```
assert ligne_dechiffre_cesar("Erqmrxu OX1LQ011", 3) == "Bonjour LU1IN011"
assert ligne_dechiffre_cesar("Bonjour LU1IN011", 0) == "Bonjour LU1IN011"

beaute1 : str = "Je suis belle, o mortels ! comme un reve de pierre,"
assert ligne_dechiffre_cesar(ligne_chiffre_cesar(beaute1, 12), 12) == beaute1
```

Question 4. Ecrire deux fonctions `chiffre_fichier_cesar` et `dechiffre_fichier_cesar`, basées sur le modèle de la Section 1 qui prennent en entrée un nom de fichier `nom` et une clef `n` et qui crée de nouveaux fichiers, correspondant respectivement à l'application du chiffrement et du déchiffrement de César de clef `n`.
Par exemple, le début du fichier `bovary.txt` :

```
The Project Gutenberg EBook of Madame Bovary, by Gustave Flaubert

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: Madame Bovary

Author: Gustave Flaubert
...
```

Et le début de `bovary-cesar.txt` obtenu à l'aide en appelant `chiffre_fichier_cesar("bovary", 3)`:

```
Wkh Surmhfw Jxwhqehuj HErrn ri Pdgdph Erydub, eb Jxvwdyh Iodxehuw

Wklv hErrn lv iru wkh xvh ri dqbrqh dqbzkhuh dw qr frvw dqg zlwk
doprvw qr uhvwulfwlrqv zkdwvrhyhu.  Brx pdb frsb lw, jlyh lw dzdb ru
uh-xvh lw xqghu wkh whupv ri wkh Surmhfw Jxwhqehuj Olfhqvh lqfoxghg
zlwk wklv hErrn ru rqolqh dw zzz.jxwhqehuj.ruj


Wlwoh: Pdgdph Erydub

Dxwkru: Jxvwdyh Iodxehuw
...
```

3 Suggestion : Attaque

Le problème du chiffre de César est qu'il ne propose que 26 transformations possibles (une pour chacun des 26 décalages possibles de l'alphabet). Cela rend l'attaque du chiffre assez facile : il suffit d'essayer de déchiffrer un texte chiffré avec tous les entiers compris entre 0 et 25 et de repérer visuellement le seul qui a du sens.

Question. Mettre en oeuvre cette méthode en écrivant une fonction `attaque-cesar` qui prend en entrée un nom de fichier (qu'on suppose chiffré avec un chiffre de César), choisit un extrait du texte de quelques lignes (le début, ou au hasard dans le texte) et écrit dans un fichier le résultat du déchiffrement de César de cet extrait pour chacune des clefs entre 0 et 25.
Par exemple, sur `bovary-cesar.txt`, la fonction `attaque-cesar` écrit un fichier ressemblant à :

```
Lignes choisies :3459-3463
================================
Decalage de : 0
Srxu duulyhu fkhc od qrxuulfh lo idoodlw, dsuèv od uxh, wrxuqhu à
jdxfkh, frpph srxu jdjqhu oh flphwlèuh, hw vxlyuh, hqwuh ghv
pdlvrqqhwwhv hw ghv frxuv, xq shwlw vhqwlhu txh erugdlhqw ghv
wurèqhv. Lov éwdlhqw hq iohxu hw ohv yéurqltxhv dxvvl, ohv
éjodqwlhuv, ohv ruwlhv, hw ohv urqfhv oéjèuhv txl v'éodqçdlhqw ghv

================================
Decalage de : 1
Rqwt cttkxgt ejgb nc pqwttkeg kn hcnnckv, crtèu nc twg, vqwtpgt à
icwejg, eqoog rqwt icipgt ng ekogvkètg, gv uwkxtg, gpvtg fgu
ockuqppgvvgu gv fgu eqwtu, wp rgvkv ugpvkgt swg dqtfckgpv fgu
vtqèpgu. Knu évckgpv gp hngwt gv ngu xétqpkswgu cwuuk, ngu
éincpvkgtu, ngu qtvkgu, gv ngu tqpegu néiètgu swk u'éncpçckgpv fgu

================================
Decalage de : 2
Qpvs bssjwfs difa mb opvssjdf jm gbmmbju, bqsèt mb svf, upvsofs à
hbvdif, dpnnf qpvs hbhofs mf djnfujèsf, fu tvjwsf, fousf eft
nbjtpoofuuft fu eft dpvst, vo qfuju tfoujfs rvf cpsebjfou eft
uspèoft. Jmt éubjfou fo gmfvs fu mft wéspojrvft bvttj, mft
éhmboujfst, mft psujft, fu mft spodft méhèsft rvj t'émboçbjfou eft

================================
Decalage de : 3
Pour arriver chez la nourrice il fallait, après la rue, tourner à
gauche, comme pour gagner le cimetière, et suivre, entre des
maisonnettes et des cours, un petit sentier que bordaient des
troènes. Ils étaient en fleur et les véroniques aussi, les
églantiers, les orties, et les ronces légères qui s'élançaient des

================================
Decalage de : 4
Ontq zqqhudq bgdy kz mntqqhbd hk ezkkzhs, zoqèr kz qtd, sntqmdq à
...
```

4 Suggestion : Chiffre affine

Afin d'éviter de s'exposer à telle attaque (dite *de force brute*), on peut complexifier le chiffre de César en un chiffre *affine*, qui utilise deux clefs a et b.

On associe à chaque lettre sa *place* dans l'alphabet (A a la place 0, B la place 1, ..., Z la place 25) et on transforme une lettre de place p en la lettre de place $(a.p + b)$ **modulo** 26.

Par exemple si a vaut 3 et b vaut 2, on transforme A en C ($0.3 + 2 = 2$ **modulo** 26) et M en M ($12.3 + 2 = 12$ **modulo** 26).

Question. Ecrire deux fonctions `chiffre_fichier_affine` et `dechiffre_fichier_affine` qui réalisent le chiffre affine sur des fichiers.

On fera particulièrement attention aux points suivants :

- L'opération $p \mapsto (a.p + b)$ **modulo** 26 n'est bijective que quand a *est premier avec* 26. Pour contourner ce problème, on pourra, par exemple, remplacer dans les calculs la clef a par le plus petit nombre plus grand que (ou égal à) a premier avec 26 (par exemple si a vaut 12, on le remplace par 15 dans les calculs).
- Pour déchiffrer un texte en connaissant les clefs a et b, il faut pour chaque lettre du texte de place p, appliquer l'opération $p \mapsto (p-b).a^{-1}$ **modulo** 26, qui nécessite de pouvoir calculer a^{-1}, l'inverse de a dans le groupe multiplicatif $\mathbb{Z}/26\mathbb{Z}$. On pourra utiliser *l'algorithme d'Euclide étendu*[3] pour déterminer cet inverse.

```
assert ligne_chiffre_affine("Bonjour LU1IN011!", 3, 7) == "Kxuixpg OP1FU011!"
assert ligne_chiffre_affine("Bonjour LU1IN011!", 2, 7) == "Kxuixpg OP1FU011!"
assert ligne_chiffre_affine("Bonjour LU1IN011!", 14, 47) == "Kxiaxjq EJ1LI011!"

assert ligne_dechiffre_affine("Kxuixpg OP1FU011!", 3, 7) == "Bonjour LU1IN011!"
assert ligne_dechiffre_affine("Kxuixpg OP1FU011!", 2, 7) == "Bonjour LU1IN011!"
assert ligne_dechiffre_affine("Kxiaxjq EJ1LI011!", 14, 47) == "Bonjour LU1IN011!"

beaute1_c : str = ligne_chiffre_affine(beaute1, 28, 2016)
assert ligne_dechiffre_affine(beaute1_c, 28, 2016) == beaute1
```

5 Suggestion : Chiffre de Vigenère

Le chiffre précédent est plus difficile à attaquer que celui de César, mais ne propose que 676 (26×26) transformations possibles. Ces deux chiffres sont dits *monoalphabétiques* parce que dans tout le texte chiffré, la même lettre chiffrée correspond toujours à la même lettre initiale. Le chiffre de Vigenère n'a pas cette propriété, on dit qu'il est *polyalphabétique*.

Le principe du chiffre de Vigenère est d'utiliser comme clef un mot m composé de lettres romaines uniquement et d'appliquer au texte initial un décalage dans l'alphabet correspondant aux places successives des lettres de la clef, répétée autant de fois que nécessaire. Ainsi, si la clef a pour longueur l, la $n$ème lettre du texte initiale est décalée d'un nombre de places dans l'alphabet correspondant à la place de la (n **modulo** l)ème lettre de la clef.

Par exemple, pour chiffrer `"Bonjour"` avec la clef `"cle"` on procède ainsi :

`texte initial`	B	o	n	j	o	u	r
`place initiale`	1	14	13	9	14	20	17
`cle`	c	l	e	c	l	e	c
`decalage`	2	11	4	2	11	4	2
`place finale`	3	25	17	11	25	24	19
`texte chiffre`	D	z	r	l	z	y	t

et on obtient `"Dzrlzyt"`[4].

Pour décoder un message chiffré en connaissant la clef, il suffit d'appppliquer l'opération inverse.

Question. Implémenter le chiffre de Vigenère.

[3] à chercher sur le Web, par exemple `https://fr.wikipedia.org/wiki/Algorithme_d%27Euclide_%C3%A9tendu`

[4] plus d'information `https://fr.wikipedia.org/wiki/Chiffre_de_Vigen%C3%A8re`

```
assert ligne_chiffre_vigenere("Bonjour LU1IN011!", "AAA") == "Bonjour LU1IN011!"
assert ligne_chiffre_vigenere("Bonjour LU1IN011!", "Maths") == "Nogqggr SM1IG011!"
assert ligne_chiffre_vigenere("Bonjour LU1IN011!", "c")
          == ligne_chiffre_cesar("Bonjour LU1IN011!", 2)

assert ligne_dechiffre_vigenere(ligne_chiffre_vigenere(beaute1, "beaute"), "beaute")
          == beaute1
```

6 Suggestion : Scytale

Une *scytale* est un baton en bois permettant de chiffrer des messages. On enroule une lanière autour du baton, on écrit le message le long du baton en allant à la ligne quand c'est nécessaire, puis on déplie la lanière. Pour lire le message initial, il faut enrouler la lanière autour d'un baton *de même diamètre*[5].

Un exemple d'opération de transormation avec un baton de périmètre 3.

1. texte initial `"0123456789"`
2. on écrit le texte sur 3 lignes (en faisant attention à utiliser toutes les lignes, même si la dernière peut être incomplète) :

0	1	2	3
4	5	6	7
8	9		

3. on récupère un texte en lisant colonne par colonne `"0481592637"`

La clef du chiffre est donc le périmètre du baton.

Question. Implémenter le chiffre avec scytale.

```
assert ligne_chiffre_scytale("0123456789", 1) == "0123456789"
assert ligne_chiffre_scytale("0123456789", 2) == "0516273849"
assert ligne_chiffre_scytale("0123456789", 3) == "0481592637"
assert ligne_chiffre_scytale("0123456789", 5) == "0246813579"

assert ligne_dechiffre_scytale("0246813579", 5) == "0123456789"
assert ligne_dechiffre_scytale("0481592637", 3) == "0123456789"
assert ligne_dechiffre_scytale(ligne_chiffre_scytale(beaute1, 4), 4) == beaute1
```

7 Suggestion : Recherche Documentaire

On pourra chercher sur le Web d'autres chiffres plus ou moins simples et les implémenter pour qu'ils opèrent sur des fichiers.

[5] cf. `https://fr.wikipedia.org/wiki/Scytale`