TP 9 - Cryptographie

Comme d'habitude, vous devez rendre votre TP dans le délai indiqué. Un fichier `cesar.py` contenant les en-têtes des fonctions et des doctests est fourni, vous devez le compléter. N'oubliez pas que vous pouvez faire vos propres tests.

Exercice 1. Code de César : cryptographie

Le but de cet exercice est de réaliser un système permettant de crypter des messages, en utilisant le code de César : on part d'un message en clair et on change toutes les lettres en les décalant d'un même nombre de positions (ce nombre est appelé la *clé*). Pour décoder le message il suffit alors de décaler dans l'autre sens, à condition de connaître la clé. Pour cet exercice, nous vous encourageons à chercher dans la doc quelles sont les fonctions qui peuvent vous être utiles. **Attention, ce qui suit ne fonctionne que pour des caractères sans accent !**

On va d'abord écrire la fonction d'encodage du texte : `encrypte(en_clair,cle)` puis une fonction de décodage. Toutes les lettres sont cryptées, mais pas les blancs et la ponctuation. Par exemple, ce code affiche les deux lignes notées en commentaire :

```
cache = encrypte('Toto est une girafe, avec des petites jambes.',12)
print(cache)                # Fafa qef gzq sudmrq, mhqo pqe bqfufqe umynqe.
print(decrypte(cache,12)) # Toto est une girafe, avec des petites jambes.
```

1. Écrire une fonction `encrypte_lettre` qui prend une seule lettre en paramètre et renvoie la lettre encodée (attention à la ponctuation et aux majuscules). Pour décaler les lettres, vous aurez besoin de 2 fonctions de bases de Python : `ord` et `chr`. La doc est là : `http://docs.python.org/3/library/functions.html`. La fonction `ord` prend une chaine contenant un seul caractère et renvoie son *code ascii* (`ord('a')` donne 97, `ord('b')` 98, etc...). Et la fonction `chr` fait l'inverse : elle renvoie le caractère en fonction du code ascii. Par exemple, le programme suivant permet de créer une liste contenant les lettres de l'alphabet (remarque : on peut obtenir le même résultat avec `string.ascii_lowercase` et `split`) :

   ```
   alphabet=[]
   for i in range(26):
     alphabet.append(chr(ord('a')+i))
   ```

   Vous devez calculer l'indice de la lettre dans l'alphabet, puis le décaler avec la clé (en prenant soin de rester entre 0 et 25 : par exemple, 'z' décalé de 2 donne 'b') et retrouver le caractère correspondant. **Attention, le nombre 97 ne doit jamais aparaître dans votre code !** Pour vérifier : lorsque l'ont fait

   ```
   for c in alphabet:
           print(encrypte_lettre(c,10), end=' '):
   ```

   on doit obtenir : `k l m n o p q r s t u v w x y z a b c d e f g h i j`.

   Ne passez pas trop de temps sur cette question, si vous êtes bloqué, appelez votre chargé de TP.

2. On constate que l'on fait 2 fois le même travail, pour une majuscule ou une minuscule. Rajouter une fonction auxiliaire qui prend 'a' ou 'A' en paramètre et permet de ne faire le calcul qu'une seule fois.

3. Enfin écrire `encrypte` qui renvoie le texte codé.
4. Passons au décodage. Écrire la fonction `decrypte(texte_cache,cle)` qui permet de décoder (et renvoyer) un texte qui a été crypté avec la clé. Soyez astucieux, ne copiez-collez pas votre code, réutilisez plutôt vos fonctions !

Exercice 2. Code de César : cryptanalyse et améliorations

1. Passons à la cryptanalyse, c'est-à-dire l'art de déchiffrer les messages sans avoir la clé ! Pour cela, nous allons essayer d'utiliser nos connaissances sur les fréquences des lettres dans un texte. En effet, vous n'êtes pas sans savoir que la lettre la plus commune en français est le 'e'... Donc, pour savoir quelle clé a permis de coder le texte, il suffit de tester toutes les clés possibles et de regarder celle qui donne le texte contenant le plus de 'e'. Écrire la fonction `devine_cle(texte_cache)` qui renvoie la clé devinée en utilisant ce principe et testez-la sur l'exemple. Attention, cette technique fonctionne d'autant mieux que le texte est long. Essayez de retrouver le texte original dans les deux cas suivants.

   Premier essai :

   ```
   Tuqd, v'mu ymzsq gzq bayyq. Mbdqe, v'mu qg pqe tmxxgouzmfuaze, bxquz
   p'tmxxgouzmfuaze.
   Vq hakmue pqe qxqbtmzfe, pqe xuoadzqe qf pqe bmzpme. Qf uxe qfmuqzf
   daeqe, fage daeqe.
   ```

   Deuxième essai :

   ```
   Jli c'rsrkkrek ul mrjzjkrj, le rezdrc rl kyfiro zeuzxf, r c'rzxlzccfe jrwire, ez le
   trwriu, ez le tyriretfe, drzj gclkfk le rikzjfe, j'rmretrzk, kirzerek le size u'rcwr.
   Zc j'rggiftyr, mflcrek c'rgcrkzi u'le tflg mzw, drzj c'rezdrc gizk jfe mfc,
   uzjgrirzjjrek urej cr elzk rmrek hl'zc rzk gl c'rjjrzcczi.
   ```

   Si vous voulez savoir pourquoi ça ne marche pas sur le second texte, cliquez là : indice.

2. **(facultatif)** Cette façon de faire n'est pas très efficace, elle force à décoder 26 fois... Il vaudrait mieux chercher la lettre la plus fréquente dans le texte et chercher la clé permettant de la faire correspondre avec un 'e'. Écrire la fonction `devine_cle2` qui utilise ce principe.
   Indice : (une fois que vous connaissez la position `pos` dans l'alphabet (entre 0 et 25) de la lettre la plus fréquente, il suffit de calculer : `( pos + ord('a') - ord('e') ) % 26`
3. On vient de voir qu'il n'est pas très difficile de décoder le message, même sans la clé. Cette méthode de cryptographie n'est donc pas très efficace. Pour empêcher de deviner en utilisant les fréquences, on va crypter en décalant de 1 la clé de codage à chaque mot. Avec l'exemple de "Toto...", ça donne : `Fafa rfg ibs vxgput, qlus uvj hwlalwk ctfuxl`. Écrire les fonctions `encrypte_mieux(texte,cle)` et `decrypte_mieux(texte,cle)` qui utilisent ce principe.
4. On se rend compte que les fonctions `encrypte` et `encrypte_mieux` sont quasiment les mêmes. Transformez-les en une unique fonction `encrypte_encore_mieux` qui prend également en paramètre le décalage utilisé pour chaque mot : il suffit de mettre 0 pour retrouver `encrypte` et 1 pour `encrypte_mieux` :

   ```
   txt='Voici le texte'
   print( encrypte(txt, 12) )       # affiche: Hauou xq fqjfq
   print( encrypte_encore_mieux(txt,12,0) )  # Hauou xq fqjfq
   print( encrypte_mieux(txt,12) )           # Hauou yr hslhs
   print( encrypte_encore_mieux(txt,12,1) )  # Hauou yr hslhs
   ```

   Pensez à utiliser cette fonction pour décrypter aussi.