Devoir d'infomatique
Dictionnaires, Tris et Dichotomie

Les barèmes sont indicatifs et pourront être reconsidérés.

Recherche d'un élément dans une liste triée (7 points)

1. Parmi les fonctions suivantes, prenant en argument un élément `x` et une liste *triée dans l'ordre décroissant* `L`, donnez sans justification *toutes* celles qui renvoient `True` lorsque `x` est dans `L` et `False` si `x` n'est pas dans `L`.

```
def find1(x, L):
    bottom, top = 0, len(L)
    while top > bottom:
        mid = (top+bottom) // 2
        if x == L[mid]:
            return True
        elif x > L[mid]:
            bottom = mid + 1
        elif x < L[mid]:
            top = mid
    return False
```

```
def find2(x, L):
    for y in L:
        if y != x:
            return False
    return True
```

```
def find3(x, L):
    bottom, top = 0, len(L)
    while top > bottom:
        mid = (top+bottom) // 2
        if x == L[mid]:
            return True
        elif x < L[mid]:
            bottom = mid + 1
        elif x > L[mid]:
            top = mid
    return False
```

```
def find4(x, L):
    for i in range(len(L)):
        if L[i] == x:
            return True
    return False
```

```
def find5(x, L):
    bottom, top = 0, len(L)
    while top > bottom:
        mid = (top+bottom) // 2
        if x == L[mid]:
            return True
        elif x < L[mid]:
            bottom = mid
        elif x > L[mid]:
            top = mid + 1
    return False
```

```
def find6(x, L):
    for y in L:
        if y == x:
            return True
        else:
            return False
```

```
def find7(x, L):
    for y in L:
        if y < x:
            return False
        if y == x:
            return True
    return False
```

Décodage d'un texte (21 points)

Le but de cette section est d'étudier une technique d'attaque du chiffrement de César, dite « attaque par analyse de fréquence ».

Rappel sur le chiffrement de César On considère un texte m écrit en minuscule avec les 26 lettres de l'alphabet latin et des espaces. Le *k-chiffré* de César de m, que l'on note m_k est obtenu en changeant chaque lettre de m par celle k-position plus loin dans l'alphabet (et en recommencant à `a` lorsqu'on dépasse `z`).
Par exemple, soit $u =$ `"vive python"`, le 3-chiffré de u est

$$u_3 = \texttt{ylyh sbwkrq}$$

Notons que k peut être négatif, ce qui correspond à décaler dans l'alphabet vers la gauche (et en recommencant à `z` lorsqu'on dépasse `a`). Par exemple, le -3-chiffré de u est

$$u_{-3} = \texttt{sfsb mvqelk}$$

2. a) (1 point) Donner v_{-1} le -1-chiffré de $v =$ `bcpst`.

b) (1 point) Donner $(v_{-1})_1$ le 1-chiffré du message obtenu à la question précédente. Que remarquez vous ?

On dispose du dictionnaire

```
dico_cesar0 = {
    'a' : 0,
    'b' : 1,
    'c' : 2,
       ...
    'z' : 25
}
```

et de la chaîne de caractères

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
```

3. a) (2 points) Compléter la fonction `dico_cesar(k)` suivante qui prend en argument un entier k et renvoie un dictionnaire dont les clefs sont les lettres de l'alphabet et les valeurs leur position dans l'alphabet, de 0 à 25, décalée de k modulo 26. Par exemple, `dico_cesar(0)` doit renvoyer un dictionnaire égal à `dico_cesar0`, et `dico_cesar(1)` doit renvoyer

```
{ 'a' : 1, 'b' : 2, ..., 'y' : 25, 'z' : 0 }
```

```
def dico_cesar(k):
    dico = {}
    for clef, valeur in dico_code_cesar0.items():
        ...
    return dico
```

b) (1 point) À quoi est égal `dico_cesar(26)` ?

4. a) (1 point) Si `c` est un caratère minuscule (soit `'a'`, soit `'b'`, ..., soit `'z'`), que vaut `alphabet[dico_cesar0[c]]` ?

b) (4 point) Compléter la fonction `code_cesar(message, k)` suivante qui prend en argument

- une chaîne de caractères `message` représentant un message m composée d'espaces et de lettres minuscules
- un entier `k`

et renvoie une chaîne de caractères représentant m_k le k-chiffré de César de m.

```
def code_cesar(message, k):
    lettre_vers_position = dico_cesar(k)
    m_k = ""
    for c in message:
        if c == ' ':
            m_k += ...
        else:
            m_k += ...
    return m_k
```

c) (2 point) En déduire et écrire une fonction `decode_cesar(message_code, k)` qui prend en argument un entier `k` et une chaîne de caractères `message_code` représentant un message codé m_k, et renvoie le message original m.

Si l'on connaît k, il est donc facile de déchiffrer m_k. Le but de l'attaque par analyse de fréquence est de deviner k à partir de m_k. Supposons que l'on sait que le message m est écrit en français et est assez long. Alors la lettre `e`, qui est la lettre la plus fréquente en français, a de bonnes chances d'être présente le plus grand nombre de fois dans m par rapport à n'importe quelle autre lettre. On va donc trouver x la lettre la plus fréquente dans m_k et partir du principe que `e` a été encodé vers x, en déduire k et enfin décoder m_k.

5. a) (4 point) Écrire une fonction `caractere_le_plus_frequent(message)` qui prend en argument une chaîne de caratères `message` composée d'espaces et de lettres minuscules, et renvoie la lettre qui apparaît le plus grand nombre de fois dans `message`. S'il y a égalité entre plusieurs lettres, on renvoie n'importe quelle lettre parmi celles qui font égalité. Attention, il faut ignorer les espaces !

b) (2 point) Écrire une fonction `devine_k(x)` qui prend en argument le caractère vers qui est encodé la lettre `e` (attention, ce n'est pas forcément la lettre `x`), et renvoie `k` le décalage qui a été utilisé dans le code de César. Par exemple, `devine_k('e')` renvoie `0`, `devine_k('f')` renvoie `1` et `devine_k('d')` renvoie 25.

6. a) (2 point) Déduire des questions précédentes une fonction `craque_code_cesar(message_code)` qui prend en argument une chaîne de caratères `message_code` représentant un message en français codé par chiffrement de César et renvoie le message décodé par analyse de fréquence.

b) (1 point) La fonction précédente permet-elle de décoder le message codé tout le temps ?