Corrigé du TP Python sur les Chaines de caractères

852 - M.Lalauze - M.Junier

1 Exercice 1

```
def question1():
    a = "10"
    b = "11"
    c = a + b
    print(c)

"""
In [2]: question1()
1011
"""
```

Question 2 :

Les expressions `maChaine[len(maChaine) - 1]` et `maChaine[- 1]` permettent d'obtenir le dernier caractère de `maChaine`.

```
def question3():
    resultat = ""
    for c in "Bonsoir":
        resultat = resultat + c
    print(resultat)

"""
In [8]: question3()
Bonsoir
"""

def question4():
    resultat = ""
    for c in "Bonsoir":
        resultat = resultat + c
        print(resultat)

"""
In [10]: question4()
B
Bo
```

```
Bon
Bons
Bonso
Bonsoi
Bonsoir
"""
```

2 Exercice 2

```
def compte(chaine,caractere):
    '''prend en entrée une chaîne de caractères et un caractère,
    et affiche en sortie le nombre d'occurrence de ce caractère'''
    n = 0
    for c in chaine:
        if c == caractere:
            n += 1
    return n

"""
In [11]: chaine = "test"

In [12]: chaine.count('t')
Out[12]: 2

In [13]: compte(chaine, 't')
Out[13]: 2
"""
```

3 Exercice 3

```
def appartient(conteneur,element):
    '''détermine si un caractère donné appartient à une chaîne de
    caractères donnée. (La fonction renverra True ou False).
    Reprogrammation de l'operateur in'''
    for c in conteneur:
        if c == element:
            return True
    return False

def appartient2(conteneur,element):
    '''détermine si un caractère donné appartient à une chaîne de
    caractères donnée. (La fonction renverra True ou False).
    Reprogrammation de l'operateur in. ALgorithme de recherche séquentielle'''
    i = 0
    long = len(conteneur)
    while i < long and conteneur[i] != element:
        i += 1
```

```
    return i < long

"""
>>> chaine='The whole wolrd is a stage.'
>>> appartient(chaine,'a')
True
>>> appartient2(chaine,'a')
True
>>> appartient2(chaine,'A')
False
"""
```

Les fonctions précédentes peuvent aussi s'appliquer à des listes.

```
"""
>>> t=[1,2,3]
>>> appartient(t,1)
True
>>> appartient(t,'1')
False
>>> appartient2(t,1)
True
>>> appartient2(t,'1')
False
"""

def positions(caractere,chaine):
    '''renvoie la liste éventuellement vide des positions
    d'un caractère dans une chaîne.'''
    t = [] #listes des positions
    for i in  range(len(chaine)):
        if chaine[i] == caractere:
            t.append(i)
    return t

"""
>>> positions('a','abracadabra')
[0, 3, 5, 7, 10]
"""
```

4 Exercice 4

```
def positionp(motif, texte, p):
    '''teste si le sous-mot motif apparait dans texte à la position p'''
    if p + len(motif) > len(texte):
        return False
    for i in range(len(motif)):
        if motif[i] != texte[p+i]:
            return False
    return True
```

```
def positionp2(motif, texte, p):
    '''teste si le sous-mot motif apparait dans texte à la position p'''
    Lmotif = len(motif)
    Ltexte = len(texte)
    if p + Lmotif > Ltexte:
        return False
    i = 0
    while i < Lmotif and  motif[i] == texte[p+i]:
            i += 1
    return i == Lmotif

"""
>>> motif, texte = 'cadabra','abracadabra'
>>> [positionp(motif, texte, i) for i in range(6)]
[False, False, False, False, True, False]
>>> [positionp2(motif, texte, i) for i in range(6)]
[False, False, False, False, True, False]
"""
def sousmot(motif, texte):
    '''prend en entrée deux chaînes de caractères motif et texte et
    retournant True ou False selon que motif est un sous-mot de texte'''
    for p in range(len(texte)-len(motif) + 1):
        if positionp(motif, texte, p):
            return True
    return False

def sousmot2(motif,texte):
    '''prenant en entrée deux chaînes de caractères motif et texte et
    retournant True ou False selon que motif est un sous-mot de texte'''
    p = 0
    Lmotif = len(motif)
    Ltexte = len(texte)
    while p <= Ltexte - Lmotif  and not positionp(motif, texte, p):
        p += 1
    return p <= Ltexte - Lmotif
```

On testera suivant quatre cas : absence, présence au bord gauche, au bord droit, et au milieu.

```
"""
In [23]: texte = "crocodile"

In [24]: [sousmot(motif, texte) for motif in ["cro", "acro", "roc", "rocd", "ile", "iles"]]
Out[24]: [True, False, True, False, True, False]
"""
```

La fonction peut aussi s'appliquer aux listes.

```
"""
>>> [i for i in range(1, 5)]
[1, 2, 3, 4]
>>> [[i, i+1] for i in range(1, 4)]
```

```
[[1, 2], [2, 3], [3, 4]]
>>> m1 = [i for i in range(1, 5)]
>>> [sousmot(m1, m2) for m1 in [[i, i+1] for i in range(1, 5)]]
[True, True, True, False]
>>> [sousmot2(m1, m2) for m1 in [[i, i+1] for i in range(1, 5)]]
[True, True, True, False]
"""
```

La fonction suivante prend en entrée deux chaînes de caractères et retourne la liste (éventuellement vide) des positions dans texte où on trouve motif.

```
def listepositions(motif, texte):
    pos = [] #liste des positions
    for p in range(len(texte) - len(motif) + 1):
        if positionp(motif, texte, p):
            pos.append(p)
    return pos

"""
In [25]: positions_sous_mot('ta','taratata')
Out[25]: [0, 4, 6]
"""
```

5 Exercice 5

```
from random import randint

def chaine_aleatoire(long):
    '''Retourne une chaine aleatoire de longueur long contituée
    de caractères alphabétiques aléatoires entre "A" et "Z"'''
    binf,bsup  = ord('A'),ord('Z')
    chaine = ''
    for i in range(long):
        chaine += chr(randint(binf,bsup))
    return chaine

def chaine_aleatoire2(longueur):
    binf, bsup  = ord('A'), ord('Z')
    return ''.join([chr(randint(binf,bsup)) for i in range(long)])

"""
>>> chaine_aleatoire(12)
'HPVNVKKPUELP'
>>> chaine_aleatoire2(12)
'LPTWINONKSLX'
"""
def acgt_aleatoire(longueur):
    '''Chaine aléatoire où chaque caractère est tiré parmi "ACGT"'''
    codons = 'ACGT'
    chaine=''
```

```
    for i in range(longueur):
        chaine += codons[randint(0, 3)]
    return chaine
"""
>>> acgt_aleatoire(16)
'TGGGTAAATGGCGCAG'
"""
```

- Question 3

```
motif = acgt_aleatoire(5)
texte = acgt_aleatoire(10**4)
print('Nombre de positions où motif=%s apparait dans texte : '%motif,
len(positions_sous_mot(motif, texte)))
```

- Nombre de positions où `motif = "GGATA"` apparait dans texte : 10

```
def exo5_question2():
    nexemple, longueur = 100, 5
    cumulnpos = 0
    for i in range(nexemple):
        m1 = acgt_aleatoire(longueur)
        m2 = acgt_aleatoire(10**4)
        cumulnpos += len(positions_sous_mot(m1,m2))
    print('Sur %s exemples, moyenne du nombre de positions où une chaine\
     aléatoire de longueur %s (avec des lettres parmi ACGT) apparait dans \
     une chaine aleatoire de \
    longueur 10**4 : %.3f'%(nexemple,longueur,cumulnpos/nexemple))
```

Sur 100 exemples, la moyenne du nombre de positions où une chaine aléatoire de longueur 5 (avec des lettres parmi ACGT) apparait dans une chaine aleatoire de longueur 10^4 est expérimentalement de $9,360$.

Calculons la valeur théorique :

- Soit X le nombre de positions où m1 apparait dans m2 :

$$X = X_1 + X_2 + \cdots + X_{996}$$

  où les variables aléatoires X_i (avec i entre 1 et 996) suivent une loi de Bernoulli de paramètre $\frac{1}{4^5}$.

- Si m1 apparait dans la sous-chaine de m2 commençant à la position i aalors $X_i = 1$ et $X_i = 0$ sinon.
- On a pour tout entier $1 \leqslant i \leqslant 9996$, $\mathrm{E}(X_i) = \frac{1}{4^5}$.
- Donc par linearité de l'espérance : $\mathrm{E}(X) = 9996 \times \mathrm{E}(X_i) = \frac{9996}{4^5}$

```
>>> 9996/4**5 #valeur théorique
9.76171875
```

6 Exercice 6

```
def inversion(chaine):
    '''Retourne l\'inverse de la chaine passée en paramètre'''
    inv = ''
    for c in chaine:
        inv = c + inv
    return inv

"""
In [16]: chaine = 'test'

In [17]: chaine[::-1]  #inversion par slicing
Out[17]: 'tset'

In [18]: inversion(chaine)
Out[18]: 'tset'
"""
def mot_palindrome(mot):
    '''teste si un mot est un palindrome'''
    mot = mot.lower() #conversion en minuscules
    mot = convert_accents(mot) #élimination des accents
    return inversion(mot) == mot

"""
>>> mot_palindrome('radar')
True
>>> mot_palindrome('baobab')
False
"""
def texte_palindrome(chaine):
    '''teste si un texte est un palindrome
    La variable texte etant de type string ses modifications dans le
    contexte local de la fonction ne modifient pas sa valeur dans le
    contexte global du programme'''
    chaine = chaine.lower() #conversion de chaine en minuscules
    accent, nonaccent = "âàéèêëîïôûüç", "aaeeeeiiouuc"
    ponctuation = " !?;,:."
    newchaine=''
    i = 0
    while  i < len(chaine):
        caractere = chaine[i]
        #liste des positions de caractere dans la chaine des caracteres accentués
        p = positions(caractere,accent)
        #si p est un caractere acentué
        if p!=[]:
            newchaine += nonaccent[p[0]]
        #sinon si caractere n'est pas un caractere de ponctuation
        elif caractere not in ponctuation:
            newchaine += caractere
```

```
        i += 1
    #pour tester
    #print('Nouvelle chaine nettoyée : ', newchaine)
    return inversion(newchaine) == newchaine

"""
>>> texte_palindrome('Ésope reste ici, et se repose')
Nouvelle chaine nettoyée :  esoperesteicietserepose
True
"""
```

7 Exercice 7

```
def occurences(chaine):
    chaine = chaine.lower() #on convertit la chaine en minuscules
    chaine = convert_accents(chaine) #conversion des caracteres accentues
    tab = [0]*26
    #on crée un tableau de 26 zéros correspondant aux occurences
    #de chaque lettre
    for lettre in chaine:
        if 97 <= ord(lettre) <= 122:
            tab[ord(lettre)-97] += 1
    return tab


def position_maxi(tab):
    '''retourne le maximum et la liste des positions où il est atteint'''
    tmaxi,maxi = [0],tab[0]
    for indice in range(1,len(tab)):
        terme=tab[indice]
        if terme > maxi:
            tmaxi, maxi = [indice],terme
        elif terme == maxi:
            tmaxi.append(indice)
    return maxi,tmaxi


def occurences_maxi(chaine):
    '''détermine les lettres de l'alphabet les plus fréquentes dans une
    chaîne de caractères.'''
    histo = occurences(chaine) #histogramme des occurences des lettres
    maxi,tmaxi = position_maxi(histo)
    for i in tmaxi:
        print('La lettre',chr(97+i),' est apparue ',maxi,' fois dans la chaine')

"""
>>> occurences_maxi('taratatta')
La lettre a  est apparue  4  fois dans la chaine
La lettre t  est apparue  4  fois dans la chaine
```

```
>>> occurences_maxi('çiçi éécè')
La lettre c  est apparue  3  fois dans la chaine
La lettre e  est apparue  3  fois dans la chaine
"""
```

8 Exercice 8

```
def est_anagramme(m1,m2):
    '''teste si le mot m1 est anagramme de m2.
    On redéfiniti la fonction occurences pour traiter le ces des traits
    d'union'''
    def occurences2(chaine):
        chaine = chaine.lower() #on convertit la chaine en minuscules
        chaine = convert_accents(chaine) #conversion des caracteres accentues
        tab = [0]*27
        #on crée un tableau de 26 zéros correspondant aux occurences
        #de chaque lettre
        for lettre in chaine:
            if 97 <= ord(lettre) <= 122:
                tab[ord(lettre)-97] += 1
            elif lettre=='-':
                tab[26] += 1
        return tab

    return occurences2(m1)==occurences2(m2)

"""
>>> est_anagramme('beausejour','jourseueba')
True
>>> est_anagramme('beausejour','jourseuebz')
False
>>> est_anagramme('beau-sejour','jourseueba')
False
>>> est_anagramme('beau-sejour','jo-urseueba')
True
"""
```