TD 4 -
Listes et chaînes de caractères.

Informatique
MPSI - Lycée Thiers

2014/2015

Exercice 1 : La suite de Syracuse

1. Ecrire une fonction `syracuse()` qui prend en paramètre deux entiers > 0 a et n et qui retourne la liste des $n+1$ premiers termes $u_0, u_1, \ldots, u_n$ de la suite $(u_n)_n$ définie par la relation de récurrence :

$$u_0 = a \qquad \forall\, n \in \mathbb{N},\ u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair et } u_n \neq 1 \\ 1 & \text{si } u_n = 1 \end{cases}$$

2. L'essayer avec différentes valeurs de a, et vérifier la conjecture de Syracuse :

   Quelque soit la valeur de $a \in \mathbb{N}^*$, la suite $(u_n)_n$ est stationnaire en 1 à partir d'un certain rang.

Exercice 1 : Réponse

```
def syracuse(a,n):
    L = [a]
    for i in range(n+1):
        u = L[-1]
        if u == 1:
            L.append(1)
        elif u%2 == 0:
            L.append(u/2)
        else:
            L.append(3*u+1)
    return L
```

```
>>> 1 in syracuse(7,100)
True
```

Essayer l'instruction : `1 in syracuse(a,n)`
pour différentes valeurs de a et pour des valeurs de n suffisamment grandes.

Exercice 2 - Enoncé

Ecrire une fonction `recherche(chaine,mot)` prenant en paramètres deux chaines de caractère `chaine` et `mot` et qui retourne `True` ou `False` selon que `mot` apparaisse comme sous-mot dans `chaine` ou non.

1. A l'aide de slicing (une tranche).
2. Sans utiliser de slicing.
3. Ecrire une fonction `compte(chaine,mot)` qui compte le nombre de fois où `mot` apparaît dans `chaine`. La tester sur `compte('aaaaa', 'aa')`.

Exercice 2 - Corrigé

1) Avec slicing :

```
def cherche(chaine,mot):
    N = len(chaine)
    n = len(mot)
    for i in range(N-n+1):
        if mot == chaine[i:i+n]:
            return True
    return False
```

2) Sans slicing :

```
def cherche2(chaine,mot):
    N = len(chaine)
    n = len(mot)
    for i in range(N-n+1):
        flag = True
        k = 0
        while k < n:
            if mot[k] != chaine[i+k]:
                flag = False
                break
            k += 1
        if flag == True:
            return True
    return False
```

Exercice 2 - Corrigé

3) Pour compter le nombre d'occurrence il suffit d'utiliser un compteur que l'on incrémente dès que `mot` apparaît :

```
def compte(chaine,mot):
    N = len(chaine)
    n = len(mot)
    compteur = 0
    for i in range(N-n+1):
        if mot == chaine[i:i+n]:
            compteur += 1
    return compteur
```

Exercice 3 - Enoncé

1. Ecrire une fonction `palindrome()` prenant en paramètre un mot et qui retourne `True` ou `False` selon que le mot est ou non un palindrome. Un mot est un palindrome si il peut-être lu aussi bien de gauche à droite que de droite à gauche. Par exemple : `'radar'`, `'kayak'` sont des mots palindromes.
   1.1 A l'aide d'un slicing.
   1.2 Sans utiliser de slicing.
2. On souhaite écrire une fonction `Palindrome()` prenant en paramètre une phrase et qui retourne `True` ou `False` selon que la phrase est ou non un palindrome. Une phrase est un palindrome si elle peut-être lue aussi bien de gauche à droite que de droite à gauche, lorsque l'on ignore les espaces. Par exemple : `"Engage le jeu que je le gagne"` est une phrase palindrome.
   2.1 Ecrire une fonction prenant en paramètre une phrase et qui retourne la même phrase dans laquelle tous les espaces ont été supprimés.
   2.2 En déduire la fonction `Palindrome()`.

Exercice 3 - Correction

1.1

```
def palindrome(mot):
    return mot == mot[::-1]
```

1.2

```
def palindrome2(mot):
    n = len(mot)
    for k in range(n//2):
        if mot[k] != mot[n-k-1]:
            return False
    return True
```

2.1

```
def suppr(mot):
    res = ""
    for x in mot:
        if x != ' ':
            res = res + x
    return res
```

2.2

```
def Palindrome(mot):
    res = suppr(mot)
    return palindrome2(res)
```

Exercice 4 - Enoncé

Un algorithme de Tri prend en paramètre une liste de nombre et l'ordonne dans le sens croissant. Un exemple en est le Tro à bulle. Il s'opère de la façon suivante :

- Parcourir les éléments du tableau de gauche à droite.
  - Dès que l'on rencontre deux éléments consécutifs qui ne sont pas dans le bon ordre, on échange leur position. C'est à dire :
    `SI tableau[i] > tableau[i+1]`
    `ALORS : échanger tableau[i] et tableau[i+1]`
- Recommencer tant que l'on a changé quelque chose.

Ecrire une fonction `TriBulle()` en `python` qui prend en paramètre une liste de nombres et retourne la liste triée par l'algorithme du tri à bulle.

Exercice 4 - Corrigé

```
# Tri a bulle
def tri_bulle(L):
    n = len(L)
    chgt = True
    while chgt == True:
        chgt = False
        for k in range(n-1):
            if L[k] > L[k+1]:
                L[k], L[k+1] = L[k+1], L[k]
                chgt = True
        n = n-1
    return L
```

Après le k-ième passage dans la boucle `for` les k derniers éléments sont à leur place définitive. C'est pourquoi après chaque boucle `for` on décrémente `n`.

Exercice 5 - Enoncé

1. Montrer comment obtenir la liste des nombres premiers inférieurs à N à l'aide d'une compréhension de liste.

2. Crible d'Erathostène :

   2.1 Soit `L = [ i for i in range(100) ]` et soit a un entier $1 \leqslant a \leqslant 100$. Quelle suite d'instruction permet de changer dans `L` le `1` et tous les multiples stricts de a en des zéros.

   2.2 Ecrire une fonction `supp0()` prenant en paramètre une liste de nombre et qui retourne une liste dans laquelle tous les `0` ont été supprimés.

   2.3 Ecrire une fontion `erathostene()` prenant en paramètre un entier n et qui retourne la liste de tous les nombres premiers inférieurs ou égaux à n. L'algorithme procédera ainsi :

   – Parcourir la liste des entiers de 2 à n de gauche à droite (on pourra s'arrêter à $\lfloor n/2 \rfloor + 1$).
   – Pour chaque élément non-nul, remplacer dans la liste tous ses multiples stricts par des zéros.
   – A la fin du parcours, retourner une copie de la liste dans laquelle tous les zéros ont été supprimés.

Exercice 5 - Corrigé

1)

```
L = [i for i in range(100)]
L[1] = 0
for k in L:
    if k != 0:
        t = 2
        while k*t <= 100:
            L[k*t] = 0
            t += 1
```

2)

```
def suppr0(L):
    M = [ ]
    for x in L:
        if x != 0:
            M.append(x)
    return M
```

Exercice 5 - Corrigé

2.3)

```
def erathostene(n):
    L = [x for x in range(n+1)]
    L[1] = 0
    for k in L:
        if k!= 0:
            t = 2
            while k*t <= n:
                L[k*t] = 0
                t+= 1
    return suppr0(L)
```

```
>>> erathostene(70)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
61, 67]
```