Lycée Louis-Le-Grand, Paris
MPSI 4 – Informatique pour tous
A. Troesch, J.-P. Becirspahic

2013/2014

Corrigé du TP n° 2 (seconde partie)

Exercice 9.

Commençons par créer la liste des entiers compris entre 2 et n puis appliquons la démarche suivante :

1. on considère le premier élément de la liste ;
2. on supprime de la liste tous les multiples de cet éléments, s'ils se trouvent encore dans la liste ;
3. on réitère la démarche avec le second élément de la liste, puis le troisième, et ainsi jusqu'à exhaustion de la liste.

```
def crible(n):
    lst = list(range(2, n+1))
    p = 0
    while p < len(lst):
        for k in range(2, n//lst[p]+1):
            if k*lst[p] in lst:
                lst.remove(k*lst[p])
        p += 1
    return lst
```

```
>>> crible(80)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79]
```

Exercice 10.

Cet exercice peut être réalisé directement avec l'interprète ; il illustre l'expressivité de la définition d'une liste par compréhension.

```
>>> entiers = list(range(2, 101))
>>> diviseurs = list((n, list(k for k in range(1, n+1) if n % k == 0)) for n in entiers)
>>> premiers = list(n for (n, lst) in diviseurs if len(lst) == 2)
>>> premiers
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89,
 97]
```

Exercice 11.

1. Nous choisissons de retourner à l'utilisateur la liste de tous les termes de la suite jusqu'au rang n :

```
from math import sqrt

def termes():
    n = eval(input('valeur de n : '))
    lst = [0.]
    for i in range(n):
        lst.append(sqrt(3*lst[-1]+4))
    return lst
```

```
>>> termes()
valeur de n :      # ici l'utilisateur répond 5
[0.0, 2.0, 3.1622776601683795, 3.6724423726595274, 3.875219621902555, 3.952930414984264]
```

Pour la deuxième question, il n'est plus nécessaire de mémoriser dans une liste l'ensemble des termes de la suite :

```
n, u = 0, 0.
while u <= 3.99999999:
    n += 1
    u = sqrt(3*u+4)
print('rang demandé : ', n)
```

L'exécution de ce script retourne la valeur $n = 21$.

2. Pour cette question, il est nécessaire d'itérer en parallèle la suite u et la suite s définie par : $s_n = \sum_{k=0}^{n} u_k$, et ce jusqu'au rang $n = 1000$:

```
u = s = 1
for i in range(1000):
    u = 1 / (u + 1)
    s += u
print('valeur de la somme :', s)
```

On obtient $s_{1000} \approx 618.9516037633203$. Notons qu'il est facile de prouver que $(u_n)_{n\in\mathbb{N}}$ converge vers le nombre d'or $\varphi = \dfrac{1+\sqrt{5}}{2}$ et que par voie de conséquence $s_n \sim n\varphi$ (d'après le lemme de Cesàro par exemple).

3. Pour une récurrence d'ordre 2, on itère la suite (u_n, u_{n+1}) :

```
from math import sin, cos

def itere(n):
    u, v = 0, 1
    print(u, end = ' ')
    for i in range(n):
        u, v = v, sin(u) + 2 * cos(v)
        print(u, end = ' ')
    print()
```

L'observation des 1000 premiers termes de la suite ne semble pas faire apparaître une valeur limite.

4. La relation de récurrence est d'ordre 3 donc on itère la suite (u_n, u_{n+1}, u_{n+2}) :

```
def f(x,y):
    return x * cos(y) + y * cos(x)

def itere(n):
    u, v, w = 0, 1, f(1, 0)
    print(u, end = ' ')
    for i in range(n):
        if i % 2 == 0:
            u, v, w = v, w, f(w, u)
        else:
            u, v, w = v, w, f(w, v)
        print(u, end = ' ')
    print()
```

Exercice 12.

En informatique, il est maladroit de calculer les coefficients binomiaux en utilisant leur expression factorielle car cela conduit très vite à faire des calcul sur des entiers de très grande taille. On préfère utiliser la formule de Pascal pour les calculer de proche en proche. Pour autant, il n'est pas nécessaire de stocker les différentes valeurs dans un tableau bi-dimensionnel car chaque ligne du triangle de Pascal ne nécessite que la ligne précédente pour être déterminée :

$$\binom{n-1}{p-1} + \binom{n-1}{p} = \binom{n}{p}$$

Ainsi, si `lst` est une liste contenant la $(n-1)^{\text{e}}$ ligne du triangle de Pascal, l'instruction `lst[p] += lst[p-1]` permet de remplacer $\binom{n-1}{p}$ par $\binom{n}{p}$ *à condition de procéder à cette modification de la droite vers la gauche de la liste.*
La boucle qui suit permet donc de calculer la n^{e} ligne du triangle de Pascal sous la forme d'une liste :

```
lst = [1]
for k in range(n-1):
    lst.append(1)
    for p in range(k, 0, -1):
        lst[p] += lst[p-1]
```

Il nous reste à ajouter une fonction permettant à chaque étape l'affichage des éléments de la liste. Pour respecter l'alignement, nous allons allouer 5 espaces à chacune des colonnes ; il ne faudra donc pas dépasser la valeur $n = 20$, où allouer un espace plus grand à chaque colonne.

```
def affiche(lst):
    for x in lst:
        print('{:>5}'.format(x), end = ' ')
    print()

def pascal(n):
    lst = [1]
    affiche(lst)
    for k in range(n-1):
        lst.append(1)
        for p in range(k, 0, -1):
            lst[p] += lst[p-1]
        affiche(lst)
```

Exemple :

```
>>> pascal(10)
    1
    1     1
    1     2     1
    1     3     3     1
    1     4     6     4     1
    1     5    10    10     5     1
    1     6    15    20    15     6     1
    1     7    21    35    35    21     7     1
    1     8    28    56    70    56    28     8     1
    1     9    36    84   126   126    84    36     9     1
```

Exercice 13.

Notons tout d'abord que puisqu'on s'autorise des bases dépassant 10, il devient nécessaire de représenter les nombres de base b par une chaîne de caractères. La première étape consiste alors à convertir l'élément de type `str` représentant un entier en base b en un élément de type `int`.
Pour associer facilement un caractère à un entier, nous allons créer la liste `['0', '1','2', ...,'d','e','f']` ; ainsi, chaque caractère sera associé à son indice dans la liste.

```
lst = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f']
```

Nous sommes maintenant en mesure d'écrire une fonction convertissant une chaîne représentant un entier en base b en un entier de type `int` :

```
def base2dix(b, chn):
    n, p = 0, 1
    for x in reversed(chn):
        n += lst.index(x) * p
        p *= b
    return n
```

Par exemple :

```
>>> base2dix(16, '45ab29c')
73052828
```

Nous avons ensuite besoin d'une fonction réalisant l'opération inverse :

```
def dix2base(b, n):
    x, chn = n, ''
    while x > 0:
        chn = lst[x % b] + chn
        x = x // b
    return chn
```

Par exemple :

```
>>> dix2base(16, 73052828)
'45ab29c'
```

Il reste maintenant à écrire la fonction de conversion demandée :

```
def conversion(b, c, chn):
    return dix2base(c, base2dix(b, chn))
```

```
>>> conversion(3, 13, '1200211221')
'122a1'
>>> conversion(13, 3, '122a1')
'1200211221'
```