MP 16-17 Feuille Info n° 01 : Révisions

Du 02/09/16 au 09/09/16

Exercice 1.

1. Ecrire une suite d'instructions pour obtenir la somme des cubes des chiffres de 2016
2. Ecrire une fonction **Som_cube** d'argument un entier naturel n et qui renvoie la somme des cubes des chiffres de n (en base 10 évidemment)
3. Ecrire une fonction **Eq_Som_cube** prenant pour argument un entier naturel n et qui renvoie la liste des entiers naturels inférieurs ou égaux à n et qui sont égaux à la somme des cubes de leurs chiffres.
4. Question mathématique. Déterminer tous les entiers naturels qui sont égaux à la somme des cubes de leurs chiffres.

Exercice 2. On considère une matrice carrée A de taille $n \times n$ (avec numpy ou directement sous forme de tableau bidimensionnel) à coefficients entiers ou flottants.

1. (a) Ecrire une fonction **som_ligne** prenant en arguments une matrice A et un entier i et qui retourne la somme des coefficients de A situés sur la ligne i pour $0 \leqslant i \leqslant n-1$
   (b) Ecrire une fonction **som_colonne** prenant en arguments une matrice A et un entier j et qui retourne la somme des coefficients de A situés sur la colonne j pour $0 \leqslant j \leqslant n-1$
   (c) Ecrire une fonction **som_diag** prenant en argument une matrice A et qui retourne la somme des coefficients diagonaux de A
   (d) Ecrire une fonction **som_secdiag** prenant en argument une matrice A et qui retourne la somme des coefficients de la diagonale secondaire de A

   On dit qu'une matrice A est magique lorsque toutes les sommes sur les lignes, les colonnes et les deux diagonales sont égales
2. Ecrire une fonction **est_magique** prenant en argument une matrice A et retournant le booléen **True** si la matrice A est magique et **False** sinon

Exercice 3. Pour tout $n \in \mathbb{N}^*$, on pose $S_n = \sum_{k=1}^{n} \frac{1}{k^2}$

1. Ecrire une fonction **somcar** prenant en argument un entier n et qui retourne la somme S_n.
   On peut montrer [1] que la suite $(S_n)_{n\in\mathbb{N}^*}$ converge vers une certaine limite ℓ et que, de plus, la distance entre le $n-$ième terme S_n et la limite l vérifie : $|S_n - \ell| \leqslant \frac{1}{n}$.
2. Ecrire une fonction **approche** prenant en argument un flottant $\varepsilon > 0$ et qui retourne une valeur approchée de la limite ℓ avec une erreur a priori inférieure à ε

Exercice 4. On rappelle que si a est un réel strictement positif, la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = A$ avec $A > 0$, et $\forall n \in \mathbb{N}, u_{n+1} = \frac{1}{2}\left(u_n + \frac{a}{u_n}\right)$ converge [2] vers $\sqrt{a}$ (et de façon assez rapide (quadratique)). On peut même montrer que si $A > \sqrt{a}$ alors la suite est décroissante.
Ecrire, sans utiliser les fonctions sqrt des modules math ou numpy, ni la puissance **0.5, une fonction **racineentiere** prenant en arguments deux entiers naturels N et p et qui retourne la chaine de caractères correspondante à une valeur approchée décimale de $\sqrt{N}$ à 10^{-p} près.
Par exemple l'appel **racineentiere**(2, 30) devra retourner '1.41421356237309504880168872420 9'

Exercice 5.

1. Ecrire une fonction **permuteretour** prenant en arguments une liste L et deux entiers naturels i et j et retournant la liste obtenue à partir de L en permutant les éléments d'indice i et j. (on supposera que l'utilisateur n'appliquera la fonction que pour des indices i et j compatibles avec la longueur de la liste)

1. Question subsidiaire : faites-le !
2. Le faire !

2. Ecrire une fonction **permute** sans retour effectuant la même permutation mais en travaillant sur une liste globale L
3. On suppose que L est une liste de nombres de longueur n et que i est un entier inférieur ou égal à $n-2$.
   Que font les instructions suivantes ?

```
for j in range(i+1, n):
  if L[j] < L[i] :
    permute(j,i)
```

4. A l'aide entre autre d'un invariant de boucles, déterminer les terminaison, correction et complexité de l'algorithme suivant :

```
n = len(L)
for i in range(n-2):
  for j in range(i+1, n):
    if L[j] < L[i] :
      permute(j,i)
```

Exercice 6. Comparer les complexités en temps et en mémoire des deux fonctions suivantes :

```
def fct1(n):
  L=[0,1]
  for i in range(2,n+1):
    L.append(L[i-1]+L[i-2])
  print(L[0:n+1])
  return(L[-1])
```

2.

```
def fct2(n):
  a, b, k = 0, 1, 0
  print(a)
  while k < n:
    print(b)
    a,b,k = b, a+b, k+1
  return(b)
```

3. Même question en débarrassant les `print` des fonctions précédentes.