TD1 : Exercices en python

1. On dispose de nombres (entiers ou flottants) rangés dans une liste L.
   a) Écrire une fonction qui calcule la valeur moyenne m de ces nombres.
   b) Écrire de même une fonction qui calcule l'écart quadratique moyen e de ces nombres. On rappelle que :
   $$e = \frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2$$
   où m est la valeur moyenne des n nombres x_1,..., x_n.
   c) Toujours avec une liste L de n nombres, écrire une fonction qui renvoie le plus petit élément ; qui renvoie le plus grand élément.
   d) Écrire une fonction qui teste si un nombre x est présent dans une liste de nombres. Modifier cette fonction pour qu'elle calcule le nombre de fois où x est présent dans L.
2. On travaille maintenant avec des entiers naturels. On considère un entier $n \in \mathbb{N}$.
   a) Écrire une fonction qui teste si $n > 0$ et, dans l'affirmative, calcule la somme des entiers de 1 à n.
   b) Écrire une fonction qui calcule la factorielle de n.
   c) Écrire une fonction qui teste si $n > 0$ et, le cas échéant, calcule le produit des nombres pairs compris entre 1 et n.
   d) Écrire une fonction qui teste si n est premier.
3. Écrire une fonction qui calcule la somme :
   $$S = \sum_{n=1}^{\infty}\frac{1}{n^2}$$
   On arrête la boucle lorsque le dernier terme de la somme est inférieur à 10^{-15}, ce qui s'écrit 1e-15.
4. Écrire une fonction qui calcule une valeur approchée de π par la formule de Wallis :
   $$\pi = 2\prod_{i=1}^{\infty}\frac{4i^2}{4i^2-1}$$
   On réfléchira à la condition d'arrêt de la boucle.
5. Écrire une fonction qui reçoit une durée exprimée en secondes et construit une liste [jours, heures, minutes, secondes] désignant la même durée.

Programmation itérative et récursive

Rappelons que programmer de façon itérative consiste à utiliser des boucles for et/ou while pour effectuer des tâches répétitives tandis que prorammer de façon récursive consiste à écrire une fonction qui peut s'appeler elle-même dans le but de réaliser ces mêmes tâches.

6. Considérons la suite $u_n = 3u_{n-1} - 1$ pour $n > 1$ et $u_0 = 2$. Écrire une fonction `u_itera(n)` qui calcule u_n en utilisant une méthode itérative, puis une autre fonction `u_rec(n)` faisant la même chose mais avec méthode récursive.
7. Considérons la somme $S(n) = \sum_{k=1}^{n}\frac{1}{k(k+1)}$. Écrire une fonction `somme(n)` qui renvoie $S(n)$ en utilisant une méthode itérative puis une autre fonction `somme2(n)` qui calcul la même somme mais avec une méthode récursive.
8. Considérons la suite de Fibonacci, générée par la récurrence suivante :
   $$u_n = \begin{cases} u_{n-1} + u_{n-2} & \text{si} \quad n > 1 \\ 1 & \text{si} \quad n = 1 \\ 0 & \text{si} \quad n = 0 \end{cases}$$

a) Écrire une fonction `fibo_itera(n)` qui calcule cette suite en utilisant une approche itérative.

b) Réécrire la fonction `fibo_rec(n)` avec une approche récursive. La lancer pour calculer `fibo_rec(6)` puis `fibo_rec(200)`. Que constate-t-on dans ce dernier cas ?

c) Pour y voir plus clair, placer un compteur mouchard `cmpt` dans le programme, défini comme une variable globale et l'augmenter de 1 à chaque appel de `fibo_rec`. Que constate-t-on ?

d) Tracer l'arbre des appels récursifs de `fibo_rec(6)`. Cet arbre permet de visualiser l'ensemble des appels récursifs ainsi que l'ordre de ces appels.