INFORMATIQUE
DM3 - CORRECTION

I Fonctions utilitaires

Cette partie définit quelques fonctions qui pourront avantageusement être utilisées dans la suite du sujet.

Q1. Écrire une fonction d'entête
`def moyenne(X) -> float:`
qui prend en paramètre une séquence de nombres et qui calcule la moyenne de ces nombres. Cette fonction ne doit pas modifier le paramètre X. Par exemple : moyenne([1, 2, 3, 4]) -> 2.5

```
def moyenne(X):
    somme = 0
    for i in range(len(X)):
        somme += X[i]
    return(somme/len(X))
```

Q2. Écrire une fonction d'entête
`def variance(X) -> float:`
qui calcule la variance d'une séquence de nombres, sans la modifier. Pour rappel, la variance des n nombres $x_1, x_2, ..., x_n$ est la moyenne des carrés des écarts à la moyenne, c'est-à-dire

$$\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2=\frac{1}{n}\sum_{i=1}^{n}x_i^2-\overline{x}^2 \text{ avec } \overline{x}=\frac{1}{n}\sum_{i=1}^{n}x_i$$

Par exemple : `variance([1, 2, 3, 4]) -> 1.25`

```
def variance(X):
    moyenne_X, somme_var = moyenne(X), 0
    for i in range(len(X)):
        somme_var += (X[i]-moyenne_X)**2
    return(somme_var/len(X))
```

Q3. Écrire une fonction d'entête
`def somme(M):`
qui prend en paramètre une séquence imbriquée, de profondeur et de structure quelconques, dont tous les composants élémentaires sont des nombres, et calcule la somme de tous ces éléments.
Par exemple : `somme([[[1, 2], [3, 4, 5]], 6, [7, 8], 9]) -> 45`
Indication - *L'expression booléenne isinstance(x, numbers.Real) permet de tester si x est un scalaire numérique. Par exemple*
`isinstance(1, numbers.Real) -> True`
`isinstance(2.3e4, numbers.Real) -> True`
`isinstance([1, 2, 3], numbers.Real) -> False`

```
def somme(M):
    import numbers
    somme_t = 0
    if(isinstance(M,numbers.Real)):
        return(M)
    for i in range(len(M)):
        somme_t += somme(M[i])
    return(somme_t)
```

II Mesures expérimentales

II.A. Position de la bille

Q4. Écrire une fonction d'entête

```
def seuillage(A:np.ndarray, seuil:int) -> np.ndarray:
```

qui prend en paramètre un tableau d'entiers à deux dimensions représentant un cliché de la caméra CCD et construit un tableau de même forme contenant la valeur 1 là où la valeur des pixels de l'image originale est strictement inférieure au seuil et la valeur 0 ailleurs (pixels supérieurs ou égaux au seuil).

```
def seuillage(A, seuil) :
    n, m = A.shape
    Aseuil = np.zeros((n, m), dtype=int)
    for i in range(n) :
        for j in range(m) :
            if A[i, j] < seuil :
                Aseuil[i, j] = 1
    return Aseuil
```

Q5. Écrire une fonction d'entête

```
def pixel_centre_bille(A:np.ndarray) -> (int, int):
```

qui prend en paramètre l'image seuillée telle que produite par la fonction seuillage et renvoie les indices (ligne et colonne) du pixel le plus proche du centre de la bille (barycentre des pixels à 1).

```
import numpy as np
def pixel_centre_bille(A:np.ndarray):
    L,C = A.shape
    somme_x, somme_y,nombre_de_1 = 0, 0, 0
    for i in range(L):
        for j in range(C):
            if(A[i,j] == 1):
                somme_x += 1
                somme_y += 1
                nombre_de_1 += 1
    return(round(somme_x/nombre_de_1, somme_y/nombre_de_1))
```

On dispose de la fonction d'entête

```
def prendre_photo() -> np.ndarray:
```

qui déclenche la prise d'un cliché par la caméra CCD et renvoie l'image prise sous la forme d'un tableau à deux dimensions tel que décrit plus haut.

Q6. Écrire une fonction d'entête

```
def positions(n:int, seuil:int) -> [(int, int)]:
```

qui prend **n** photographies de la bille et renvoie la liste de ses positions dans chaque photographie en seuillant les images à la valeur seuil.
Le résultat de cette fonction est donc une liste de **n** couples de deux entiers correspondants à l'indice de ligne et de colonne des positions successives du centre de la bille au cours de son mouvement brownien.

```
def position(n:int, seuil:int):
    L = []
    for i in range (n):
        L.append(pixel_centre_bille(seuillage(prendre_photo(),seuil)))
    return(L)
```

Le capteur CCD est positionné parallèlement au plan (xOy) et ses pixels sont carrés. La caméra a été calibrée dans les conditions de l'expérience : un pixel correspond à un carré du plan (xOy) de côté t.

Q7. Définir une fonction d'entête

```
def fluctuations(P:[(int, int)], t:float) -> float:
```

qui prend en paramètre une liste de positions successives de la bille (telle que produite par la fonction positions) et la longueur correspondant à un pixel et calcule la valeur moyenne des déplacements quadratiques de la bille :
moyenne des carrés des écarts entre chaque position mesurée et la position d'équilibre de la bille (correspondant au barycentre des différentes positions observées).

```
def fluctuations(P:[(int, int)], t:float):
    P_tot = []
    for i in range(len(P)):
        P_tot.append(P[i][0])
        P_tot.append(P[i][1])
    return(t**2*variance(P_tot))
```