Projet 3c : Mosaïque (982b-3798260)

Les images au format JPG sont constituées de pixels et chaque pixel est associé à un triplet (R,V,B) de nombres compris entre 0 et 255 (uint8) avec un nombre pour chaque couleur primaire (Rouge, Vert et Bleu). Le triplet (0,0,0) correspond à un pixel noir alors que le triplet (255,255,255) correspond à un pixel blanc. A une photographie couleur est donc associé un tableau tridimensionnel d'entiers (uint8). On donnes les lignes de code suivantes pour récupérer le tableau numpy associé à la photo pomme.jpg :

In [48]:
```
import numpy as np
from PIL import Image


photo=Image.open("C:/Users/User/Documents/travail/TSI2_2024_2025/in
t=np.array(photo,dtype="uint8")#tableau tridimensionnel associé à l
# on rappelle les fonctions suivante
l,c,p=np.shape(t)#pour avoir le nombre de lignes l, de colonnes c e
tab_new=np.zeros((l,c,p),dtype="uint8")#pour créer un tableau de zé

```

1) Obtenir le tableau numpy, appelé tab, associé à cette photographie pomme.jpg en se limitant à ses 500 premières lignes et ses 500 premières colonnes.

Nous allons créer 3 nouvelles photos à partir du tableau numpy tab. Ces photos seront nommées photo_1, photo_2 et photo_3 et contiendront des pixels dont la valeur est codée en uint8:

- photo_1 est l'image en niveau de gris associée à photo. Elle est obtenue en remplaçant chaque pixel par un seul entier, dont la valeur correspond à la moyenne des trois composantes RGB de chaque pixel.
- photo_2 est une image obtenue à partir après traitement de photo. Elle est obtenue en remplaçant chaque pixel par un seul entier, dont la valeur correspond à la plus grande valeur des trois composantes RGB pour chaque pixel.
- photo_3 est une image obtenue à partir après traitement de photo. Elle est obtenue en remplaçant chaque pixel par un seul entier, dont la valeur correspond à la plus petite

valeur des trois composantes RGB pour chaque pixel.

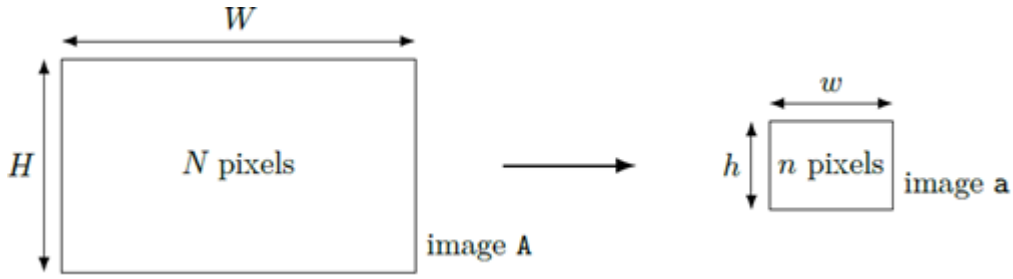
2) Ecrire une fonction photo_1 () qui prend pour argument un tableau tridimensionnel associé à une photo et renvoyant un tableau bidimensionnel associé à la photo_1. On veillera à éviter les problème d'overflow et on cherchera pour cette question.

3) Ecrire une fonction photo_2 () qui prend pour argument un tableau tridimensionnel associé à une photo et renvoyant un tableau bidimensionnel associé à la photo_2.

4) Ecrire une fonction photo_3 () qui prend pour argument un tableau tridimensionnel associé à une photo et renvoyant un tableau bidimensionnel associé à la photo_3.

On souhaite à présent réduire la taille des photo_2 et photo_3. Dans la suite, on note :

- H le nombre de lignes avant réduction (ici H=500),
- W le nombre de colonnes avant réduction (ici W=500),
- h le nombre de lignes après réduction,
- w le nombre de colonnes après réduction



On suppose ici que les dimensions de la nouvelle image a divisent celles de l'image A : H/h et W/w sont entiers. Afin d'effectuer cette réduction, on propose la fonction moyenneLocale :

In [53]:
```
def moyenneLocale(A,h,w):
    a = np.zeros((h,w),np.uint8)
    H,W=np.shape(A)
    ph, pw = H//h, W//w
    for I in range(0,H,ph):
        for J in range(0,W,pw):
            a[I//ph,J//pw]=round(np.mean(A[I:I+ph,J:J+pw]))
    return a
```

5) Expliquer en quelques lignes le principe de fonctionnement de la fonction moyenneLocale.

Un nouveau tableau réduit est généré. Les valeurs de ce nouveau tableau sont calculés à partir de valeurs moyennes obtenues en calculant le pixel moyen de chaque sous tableau de dimension ph*pw

On réalise alors la liste vignettes suivante :

In [54]:
```
vignettes=[moyenneLocale(photo_2(tab),10,10),moyenneLocale(photo_3(
```

A l'aide de la liste vignettes nous allons pouvoir composer une nouvelle image à la manière d'une mosaïque. Chaque sous tableau de dimension 10*10 de photo_1 sera remplacé par l'une de deux vignettes de la liste vignettes. Chaque pavé 10*10 de photo_1 est alors comparé à vignettes[0] puis vignettes[1] et c'est la vignette présentant l'écart le plus faible avec le pavé considéré qui sera incrustée dans la mosaïque.

Afin de comparer les vignettes et les pavés, on définit la distance L entre deux images a et b de même taille w×h par :

$$L(a,b) = \sum_{\substack{0 \le i < h \\ 0 \le j < w}} |a(i,j) - b(i,j)|$$

6) Ecrire une fonction L(a,b) qui prend pour argument deux tableaux numpy de même taille et qui calcule L. La soustraction demandée nécessite une conversion des tableaux en int16.

7) Ecrire une fonction choixVignette(pave,vignettes) qui prend pour arguments un tableau appelé *pave* correspondant à un tableau réduit et une liste *vignettes* de tableaux de même dimension que *pave* et qui renvoie l'indice i telle que L(pave,vignette[i]) est minimal.

8) A l'aide des fonctions précédentes, construire une photo-mosaïque de photo_1 à partir de la liste vignettes