TP Python 7 - Matrices de pixels et images

Dans ce TP on utilisera le module `matplotlib.pyplot` de Python qu'on importera sous le nom `plt` ainsi :

```
import matplotlib.pyplot as plt
```

Images en niveaux de gris

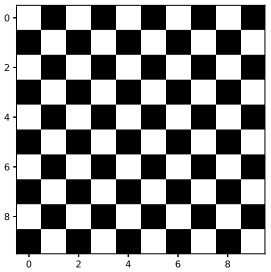
On peut représenter une image en niveaux de gris par un tableau bidimensionnel (ici une liste de listes) dont le $(i,j)^e$ élément est un entier compris entre 0 (noir) et 255 (blanc) correspondant au niveau de gris du $(i,j)^e$ pixel de l'image.

```
img = [[0, 50, 100], [150, 200, 250]]
```

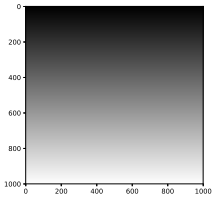
Pour afficher une telle image avec Python, on peut utiliser la fonction `plt.imshow` du module `matplotlib.pyplot` de la manière suivante :

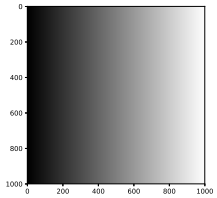
```
plt.imshow(img, cmap='gray', clim=(0, 255))
plt.show()
```

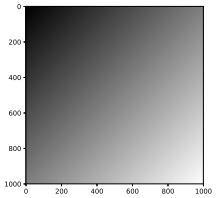
Exercice 1 Créer un damier noir et blanc 10×10 comme celui-ci :



Exercice 2 Créer des dégradés verticaux, horizontaux et diagonaux :







Images en couleurs

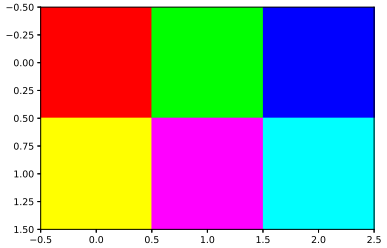
Un pixel d'une image en couleurs peut être représenté par un triplet d'entiers compris entre 0 et 255 correspondant aux niveaux respectifs de rouge, vert et bleu du pixel.

On peut ainsi représenter une image en couleurs par un tableau tridimensionnel (une liste de listes de listes). Par exemple, dans le programme suivant, on crée une image 2×3 pixel par pixel :

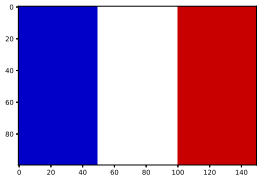
```
image = [[[0, 0, 0] for j in range(3)] for i in range(2)]
image[0][0] = [255, 0, 0]    # rouge
image[0][1] = [0, 255, 0]    # vert
image[0][2] = [0, 0, 255]    # bleu
image[1][0] = [255, 255, 0]  # jaune
image[1][1] = [255, 0, 255]  # magenta
image[1][2] = [0, 255, 255]  # cyan
```

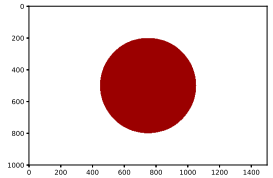
La fonction `plt.imshow` permet à nouveau d'afficher l'image.

```
plt.imshow(image)
plt.show()
```



Exercice 3 Créer le drapeau de la France, puis le drapeau du Japon.





Importer une image

La fonction `plt.imread` permet de convertir une image .png en tableau du type précédent. Par exemple, pour importer une image nommée `chaton.png` présente dans le dossier `D:\Info\Python`, on procède ainsi :

```
t = plt.imread('D:\\Info\\Python\\chaton.png')   # noter les \\
```

La fonction `plt.imread` renvoie en fait un tableau de flottants compris entre 0 et 1. On peut le transformer en tableau d'entiers compris entre 0 et 255 comme suit :

```
chaton = [[[int(255*t[i][j][k]) for k in range(3)] for j in range(len(t[0]))]
          for i in range(len(t))]
```

Transformations d'images

On pourra tester les fonctions avec l'image `chaton.png` du dossier de la classe.

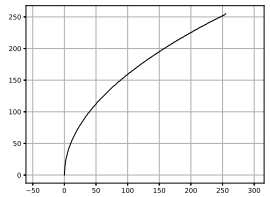
Exercice 4

1) Écrire une fonction `negatif` qui, recevant un tableau associé à une image, renvoie le tableau associé à l'image obtenue en inversant les couleurs de l'image d'origine : si les niveaux de couleurs d'un pixel sont (x, y, z) ils doivent devenir $(255 - x, 255 - y, 255 - z)$.

2) Écrire une fonction `daltonien` qui, recevant un tableau associé à une image, renvoie le tableau associé à l'image obtenue en échangeant les niveaux de rouge et de vert de l'image d'origine.

Exercice 5

Pour augmenter la luminosité d'une image ou renforcer une de ses couleurs, on peut appliquer une fonction bien choisie à ses pixels, par exemple $n \mapsto \left\lfloor 255\sqrt{\frac{n}{255}} \right\rfloor$, dont le graphe est :



Écrire une fonction `augmenter_luminosite` qui, recevant un tableau associé à une image, renvoie l'image obtenue en appliquant la fonction précédente à ses pixels.

Exercice 6

1) Écrire une fonction `renversement_vertical` qui, recevant un tableau associé à une image, renvoie le tableau associé à l'image symétrique de celle d'origine par rapport à son axe vertical.

2) Écrire une fonction `renversement_horizontal` qui, recevant un tableau associé à une image, renvoie le tableau associé à l'image symétrique de celle d'origine par rapport à son axe horizontal.

3) Écrire une fonction `rotation_gauche` qui, recevant un tableau associé à une image, renvoie le tableau associé à l'image ayant subi une rotation d'un quart de tour dans le sens trigonométrique.

Exercice 7

On peut transformer une image en remplaçant chacun de ses pixels par la moyenne des pixels qui l'entourent, moyenne pondérée par les coefficients d'une matrice appelée **noyau** ou **matrice de convolution**. On peut ainsi appliquer différents effets : augmentation du contraste, floutage, détection de contours, etc.

Écrire une fonction `convolution` qui, recevant un tableau associé à une image et une matrice carrée d'ordre 3, renvoie le tableau associé à l'image transformée par cette méthode. On éliminera les pixels situés sur les bords de l'image.

On pourra tester la fonction avec les matrices de convolution suivantes :

Floutage : $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

Augmentation du contraste : $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -5 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

Détection de contours : $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ ou $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$

Attention à ne pas diviser par 0 lorsque la somme des coefficients de la matrice est nulle.