# Traitement d'image :

## Lire et écrire des images numériques avec Python.

A.Mazza
MPSI 2-3
CPGE Oujda

21 Avril 2025

# Images vectorielles et images matricielles

On distingue deux grandes familles d'images (numériques) : les images vectorielles et les images matricielles.

- Les images vectorielles sont (re-)construites à partir d'équations mathématiques et basées sur des formes géométriques. Zoomer ou dé-zoomer sur une image vectorielle relance le calcul pour affiner la représentation en fonction de la résolution de' l'écran. Les formes paraissent lisses. Le format `.svg` est un format vectoriel
- Les images matricielles, qui font l'objet de ce cours, sont des tableaux 2D de points qu'on appelle pixels. Zoomer sur une image matricielle met en evidence la discrétisation des contours. Ce format est particulièrement adapté à la photographie ou au dessin. La première police T'EX était au format matriciel. Les formats classiques d'images matriciels sont `.bmp, .jpg, .png, ...`

# Image matricielle

- Une image matricielle est un tableau de pixels rectangulaire. Sa taille est définie par le produit du nombre de pixels en largeur et du nombre de pixels en hauteur (par exemple, 1920×1080 - format(16:9), ce qui lui donne sa taille en pixels (avec l'exemple, 2 073 600 pixels, soit environ 2 Mpx).
- Chaque pixel est une liste de 3 entiers contenant les niveaux de rouge, vert et bleu pour une image au format RGB (ou RVB en français). On peut ajouter un entier supplémentaire pour coder le canal $\alpha$ ou niveau de transparence (format RGBA).

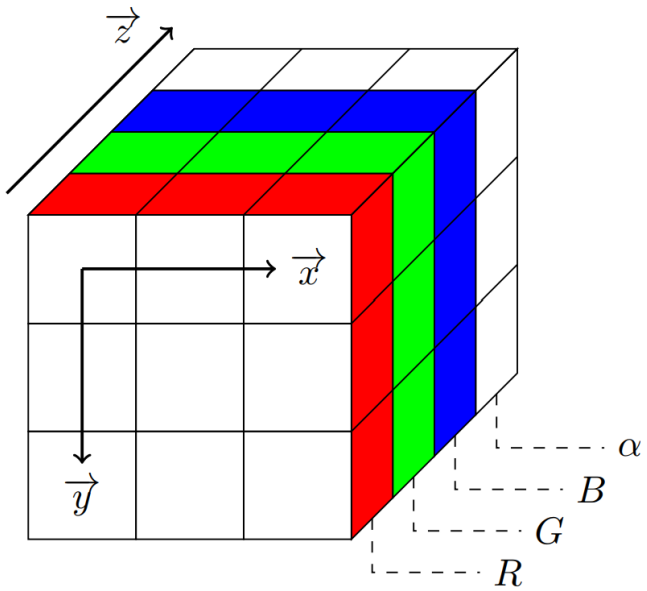


Figure: Principe de stockage des images en couleur

# Image matricielle

- Avec un codage sur 24 bits, le triplet (ou 3-uplet) $(255, 0, 0)$ code le rouge, le triplet $(0, 255, 0)$ le vert et le bleu est codé par le triplet $(0, 0, 255)$. On utilise la synthèse additive de couleurs. Ainsi le blanc est défini par le triplet $(255, 255, 255)$ et le noir par le triplet $(0, 0, 0)$. On utilise aussi une écriture hexadécimale pour coder les triplets. Le blanc devient *FFFFFF* et, par exemple, le bleu 0000*FF*.
- **Q1.** - Quel est le nombre minimal de bits nécessaire pour coder un entier sans signe, compris entre 0 et 255 ?
- **Q.2** - Donner la taille en **Kilo-Octet**, d'une image couleur représentée par une matrice de **1000** lignes et **1800** colonnes.

# Obtenir l'image sous forme de tableaux exploitables

## Avec matplotlib.pyplot

**Lecture des images avec matplotlib.pyplot :**

```
1 import matplotlib.pyplot as plt
2 for ext in ['.jpg','.jpeg', '.png', '.bmp', '.pdf']:
3     try :
4         image = plt.imread('cocci' + ext)
5         print(ext)
6         print(len(image), len(image[0]), len(image[0][0]))
7         print(image[0][0])
8     except :
9         print("Problème avec l'extension" + ext)
```

- **Remarque :** le format `PNG` conduit à coder le niveau des couleurs entre 0 et 1 et non sur un octet quand l'image est lue avec `matplotlib.pyplot`.

# Obtenir l'image sous forme de tableaux exploitables

## Avec pillow

Lecture des images avec matplotlib.pyplot :

```
1  import numpy as np
2  from PIL import Image as im
3  for ext in ['.jpg', '.jpeg', '.png', '.bmp', '.pdf']:
4      try :
5          img = im.open('cocci' + ext)
6          image = np.array(img)
7          print(ext)
8          print(len(image), len(image[0]), len(image[0][0]))
9          print(image[0][0])
10     except :
11         print("Problème avec l'extension" + ext)
```

# Obtenir des informations sur une image

**Informations sur une image avec matplotlib.pyplot et pillow**

```
1 image = plt.imread('cocci.bmp')
2 print(image.shape, image.dtype)
3 image = plt.imread('cocci.png')
4 print(image.shape, image.dtype)
5 img = im.open('cocci.png')
6 print(img.format, img.size, img.mode)
```

**Rappel:** matplotlib.pyplot a un comportement different avec les images `png`. Les canaux sont codés sur un flottant de 32bits compris entre 0 et 1.

# Visualiser une image a partir d'une matrice exploitable.

**Pour visualiser la variable image définie dans la partie précédente, on peut utiliser les codes suivants:**

Code Python

```
# avec matplotlib.pyplot
plt.imshow(image)
plt.show()
```

```
# avec pillow
image_pil = im.fromarray(image)
image_pil.show()
```

# Créer/modifier une image

La partie précédente montre qu'à partir d'une image stockée sous forme de matrice telle que définie précédemment, il est possible de la visualiser directement avec `matplotlib.pyplot`. Pour la lire avec `Pillow`, il reste à la convertir en tableau numpy (si ce n'est pas déjà le cas) puis en image `Pillow`. On utilise donc les outils NumPy

pour créer des tableaux de listes à 3 éléments. Le plus rapide pour obtenir une image noire et une image blanche de *nl* lignes et *nc* colonnes est :

Codes Python:

```
1 image_noire = np. zeros ((nl , nc,3) ,dtype =np. int8 )
2 image_blanche = np. ones ((nl , nc,3) ,dtype =np. int8 )*255
```

# Sauvegarder une image.

A partir d'une image stockée sous forme de matrice (liste de listes ) telle que définie précédemment, il est possible de l'enregistrer sous différents formats :

Code Python

```
1 #avec matplotlib.pyplot
2 plt.imsave('nomdelimage.jpg', image)
3 #avec pillow
4 image_pil = im.fromarray(image)
5 image_pil.save('nomdelimage.jpg')
```

**Attention ! avec `PIL`, il faut adapter l'extension en fonction de mode de l'image : par exemple `jpg` pour `RGB`; `png` pour `RGBA`.**

# Exercices

- **Exercice 1.** Rédiger une fonction `symetrie` qui prend en argument une image et retourne une nouvelle image, obtenue par symétrie autour d'un axe vertical passant par le milieu de l'image.
  **Indication :** la fonction `np.zeros_like(m)` crée un tableau nul de même type et de mêmes dimensions que le tableau m.
- **Exercice 2.** Rédiger une fonction `rotation` qui prend en argument une image et retourne une nouvelle image, obtenue par rotation d'un angle $\pi/2$ autour du centre de l'image.
  **Indication :** la fonction `np.zeros((n, p, s), dtype=np.uint8` crée un tableau vide de dimensions *nps* dont les éléments sont de type `np.uint8`.
- **Exercice 3.** Rédiger une fonction `negatif` qui prend en argument une image et retourne son négatif, c'est-à-dire l'image dans laquelle chaque composante de couleur de chaque pixel est remplacée par sa valeur complémentaire dans l'intervalle $[0, 255]$.

# Solutions

```python
import numpy as np
from PIL import Image

def symetrie(image):
    # Convertir l'image en tableau numpy
    tab = np.array(image)
    # Créer une copie vide du même type et dimensions
    tab_miroir = np.zeros_like(tab)
    # Appliquer la symétrie verticale
    # Chaque colonne x devient colonne (largeur - 1 - x)
    largeur = tab.shape[1]
    for x in range(largeur):
        tab_miroir[:, x] = tab[:, largeur - 1 - x]
    # Reconstruire l'image à partir du tableau
    image_miroir = Image.fromarray(tab_miroir)
    return image_miroir

```

# Solutions

```python
from PIL import Image

def rotation(image):
    largeur, hauteur = image.size
    mode = image.mode
    # Créer une nouvelle image avec dimensions inversées
    image_rot = Image.new(mode, (hauteur, largeur))
    # Accès pixel par pixel
    for x in range(largeur):
        for y in range(hauteur):
            pixel = image.getpixel((x, y))
            # Nouvelle position après rotation à gauche (/2)
            image_rot.putpixel((y, largeur - 1 - x), pixel)

    return image_rot

```

# Solutions

```python
import numpy as np
from PIL import Image

def negatif(image):
    # Convertir l'image en tableau numpy
    tab = np.array(image)

    # Calcul du négatif : 255 - valeur pixel
    tab_negatif = 255 - tab

    # Reconstruction de l'image
    image_negatif = Image.fromarray(tab_negatif)
    return image_negatif


```