Données multimédia : Python pour le traitement d'images et de données audio

Master Humanités Numériques du CESR

Clément Plancq (MSH VDL / CITERES)

Traitement d'images

- Nous travaillerons avec des *images numériques* c'est-à-dire des images acquises (par un scanner ou un appareil photo) ou créées (par un programme) sous forme binaire
- Une image numérique est stockée sous forme de fichier informatique
- Nous travaillerons avec des images matricielles (bitmap), pas des images vectorielles
- Nous travaillerons avec des images en 2D (pas en 3D, pas de stéréoscopie)

image > image numérique > image matricielle > image 2D

Définition et résolution

- Une image numérique matricielle en 2 dimensions est un tableau de points. Les points sont appelés **pixels**
- La **définition** d'une image est son nombre de pixels, exprimé en lignes x colonnes ou hauteur x largeur. Exemple : un photo de 800 x 600
- La **résolution** d'une image est le nombre de pixels par pouce (ppp) ou *pixels par inch* (ppi), elle donne une indication sur la densité de pixels

In [29]:
```
import numpy as np
import matplotlib.pyplot as plt
import requests

# Charger l'image dans un tableau numpy (ndarray)
image_path = "/content/CPR_1130x400.jpg"

image = plt.imread(image_path)
print(type(image))

# On obtient la définition de l'image (hauteur x largeur en pixels)
```

```
# avec la dimension du tableau ndarray
height, width, _ = image.shape
print(f"hauteur : {height}, largeur : {width}")
```

```
<class 'numpy.ndarray'>
hauteur : 400, largeur : 1130
```

In [28]:
```
import matplotlib.pyplot as plt

# on va quand même afficher l'image pour vérifier
plt.imshow(image)
```

Out[28]: <matplotlib.image.AxesImage at 0x7b1bf47f0e20>

Couleurs

Il existe plusieurs solutions pour le codage informatique des couleurs.

- RGB (*red*, *green*, *blue*), le plus courant. Grâce au principe de synthèse additive on peut obtenir les autres couleurs en combinant les 3 canaux rouge, vert, bleu
- CMYK utilisé dans l'impression
- TSV (Teinte Saturation Valeur) voir https://fr.wikipedia.org/wiki/Teinte_Saturation_Valeur

On s'en tiendra au RGB dans le cours. Pour chaque pixel on a donc un vecteur de trois valeurs.

In [ ]:
```
# On affiche la valeur du pixel à la position 200x100
pixel = image[200, 100]
print(pixel)
```

```
[186 178 167]
```

In [ ]:
```
# rouge, vert, bleu
print(f"rouge: {pixel[0]}, vert: {pixel[1]}, bleu: {pixel[2]}")
```

```
rouge: 186, vert: 178, bleu: 167
```

Couleurs RGB

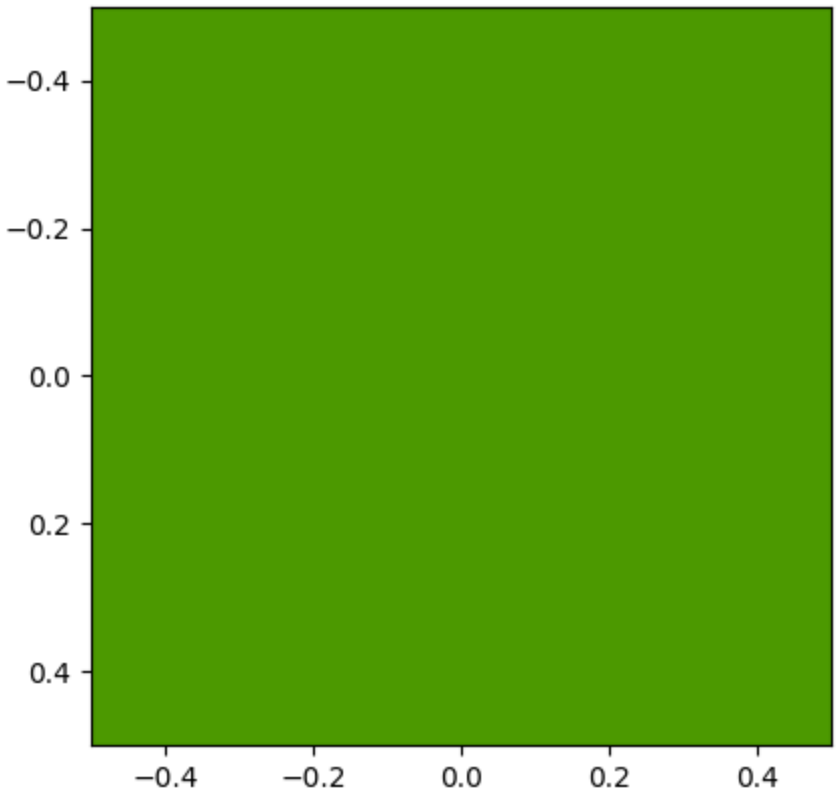
R	V	B	Couleur
0	0	0	noir
255	0	0	rouge
0	255	0	vert
0	0	255	bleu
128	128	128	gris
255	255	255	blanc

La couleur de chaque pixel est codée avec 3 valeurs, chacune est comprise entre 0 et 255, soit un octet (8 bits).
Pour RGB on a donc 3 octets, 24 bits par pixel

Une image sera donc représentée par un tableau numpy à 3 dimensions : hauteur, largeur, nb de canaux de couleurs

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt


# Une image avec un seul pixel de couleur

color = [76, 153, 0]
pixel = np.array([[color]])
plt.imshow(pixel)
```

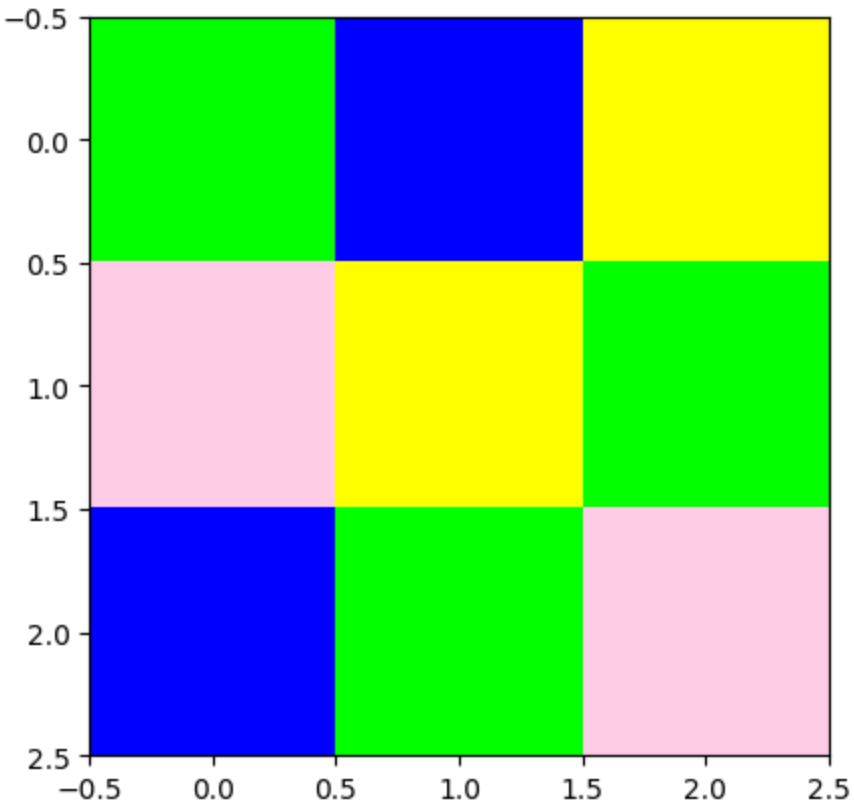
Out[8]:
```
<matplotlib.image.AxesImage at 0x7b1c34863f70>
```



✏️ Affichez un pixel rouge, un vert, un bleu puis un jaune
✏️ Affichez une ligne avec 3 pixels, un drapeau avec 3 couleurs
✏️ Affichez un carré avec 4 couleurs différentes

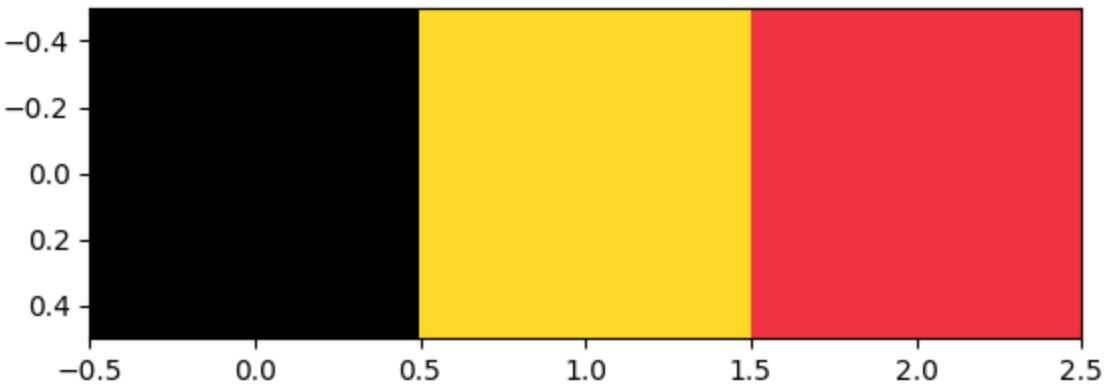
In [18]:
```
# Carré avec 4 couleurs

vert = [0, 255, 0]
bleu = [0, 0, 255]
rose = [255, 204, 229]
jaune = [255, 255, 0]

image = np.array([[ vert, bleu, jaune ], [rose, jaune, vert], [bleu, vert, r
plt.imshow(image)
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x7b1bf49dccd0>
```



In [12]:
```python
# Ligne avec 3 pixels

noir = [0, 0, 0]
gold = [253, 218, 37]
rouge = [239, 51, 64]

image = np.array([[ noir, gold, rouge ]])
plt.imshow(image)
```

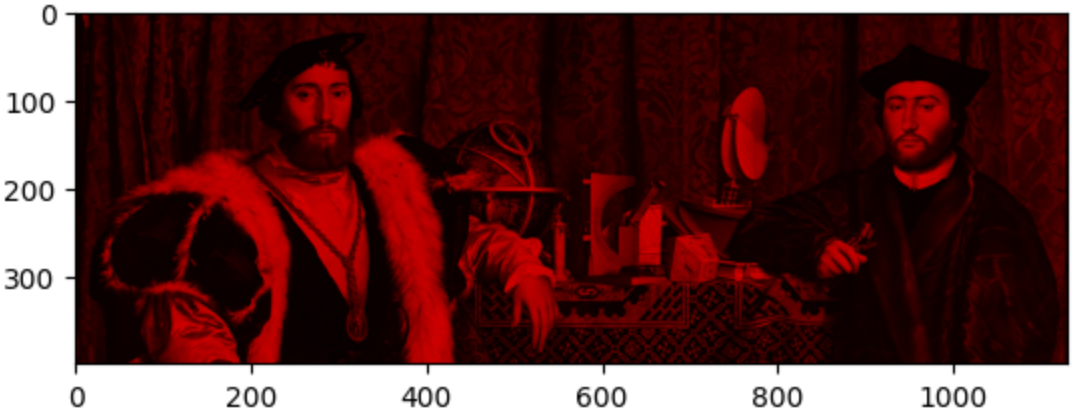
Out[12]: <matplotlib.image.AxesImage at 0x7b1bf4ee2a10>



In [32]:
```python
# Fonction qui annule les composantes vertes et bleues
def keep_red(img):
    # Créer une copie de l'image originale
    img_copy = np.copy(img)

    img_copy[:, :, 1] = 0  # Canal vert à zéro
```

```
    img_copy[:, :, 2] = 0  # Canal bleu à zéro

    return img_copy

image_red = keep_red(image)
plt.imshow(image_red)
```

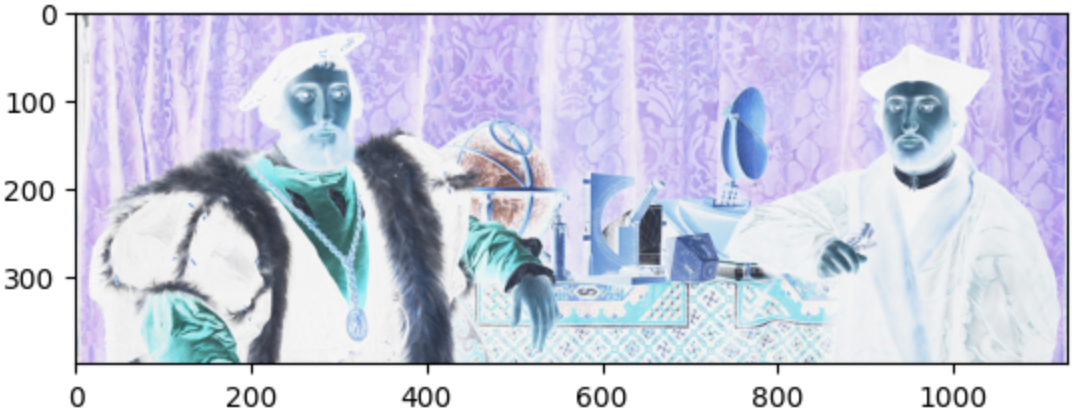
Out[32]: <matplotlib.image.AxesImage at 0x7b1bf2e17310>



✏️ Écrire la fonction keep_blue et affichez le résultat

In [33]:
```
def inverse(img):
    img_copy = np.copy(img)

    # Inverser les couleurs en soustrayant chaque canal RVB de 255 (valeur m
    img_copy = 255 - img_copy
    return img_copy

image_inverse = inverse(image)
plt.imshow(image_inverse)
```

Out[33]: <matplotlib.image.AxesImage at 0x7b1bf2d419f0>



Sélection d’une partie d’une image

Puisqu’une image est une matrice hauteur x largeur stockée dans un tableau numpy, on peut sélectionner une sous-partie de l’image, c’est-à-dire découper

un carré dans l'image.

In [36]:
```
ligne_1 = 60
ligne_2= 200
colonne_1 = 850
colonne_2 = 1100
extrait = image[ligne_1:ligne_2, colonne_1:colonne_2]
plt.imshow(extrait)
```

Out[36]: <matplotlib.image.AxesImage at 0x7b1bf2d8be80>

✏️ Reproduire cette image : 

Bibliothèque Pillow

Pillow offre un ensemble de fonctions de haut niveau pour le traitement d'images
On trouve un éventail des possibilités dans le tutoriel

Lire et sauver une image

In [3]:
```
from PIL import Image
```

```
im = Image.open("../img/CPR_1130x400.jpg")
print(f"format: {im.format}, taille: {im.size}, mode: {im.mode}")
```

```
format: JPEG, taille: (1130, 400), mode: RGB
```

On peut convertir facilement une image. Ici de jpg à png par exemple

In [39]:
```
im.save("/content/CPR_1130x400.png", format="PNG")
```

Accès aux données

Depuis un objet de la classe `Image` je peux accèder aux données sous forme de tableau numpy

In [42]:
```
image_array = np.asarray(im)
print(image_array[200,100])
```

```
[186 178 167]
```

Retailler une image

La fonction `resize` permet de retailler une image en indiquant la taille désirée en paramètre

✏️ Parcourez la documentation de la fonction et

1. retaillez notre image de moitié
2. agrandissez la de 20%

In [47]:
```
# 1 retaillez l'image de moitié

im = Image.open('/content/CPR_1130x400.jpg')
width = im.width // 2
height = im.height // 2
im_resized = im.resize((width, height))

print(im.size) # la taille de l'image originale
print(im_resized.size) # la taille de l'image réduite
plt.imshow(im_resized)
```

```
(1130, 400)
(565, 200)
```

Out[47]:
```
<matplotlib.image.AxesImage at 0x7b1bf4959e40>
```

In [55]:
```
# 2. Agrandir l'image de 20%

width = int(im.width + im.width * 0.2)
height = int(im.height +  im.height * 0.2)
im_resized = im.resize((width, height))

print(im.size) # la taille de l'image originale
print(im_resized.size) # la taille de l'image réduite
plt.imshow(im_resized)
```

```
(1130, 400)
(1356, 480)
```

Out[55]: <matplotlib.image.AxesImage at 0x7b1bf29dae60>

Rotation

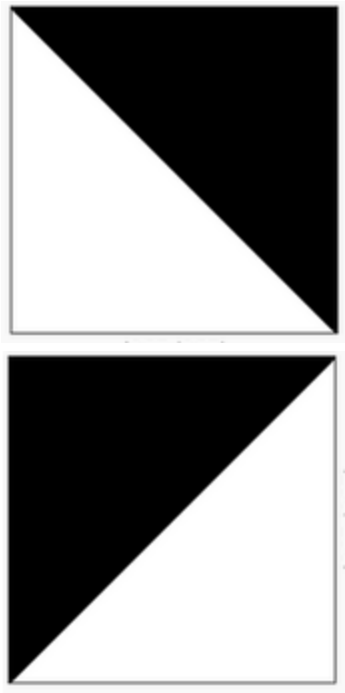
La fonction `rotate` permet de faire pivoter une image

In [10]:
```
from IPython.display import display

im_90 = im.rotate(90, expand=True)
display(im_90)
```

✏️ Quelque chose ne va pas. Parcourez la documentation de `rotate` pour que l'image retournée s'affiche correctement

✏️ Faîtes faire un tour complet à l'image `../img/noir_blanc.png` en 4 étapes

In [16]:
```
img = Image.open('../img/noir_blanc.png')

angles = (0, 90, 180, 270)
for angle in angles:
    img_res = img.rotate(angle)
    display(img_res)

#display(img)

#img_90 = img.rotate(90)
#display(img_90)

#img_180 = img.rotate(180)
#display(img_180)

#img_270 = img.rotate(270)
#display(img_270)
```



Niveaux de gris

La fonction `convert` permet de transformer une image en niveaux de gris

In [ ]:
```
im_grayscale = im.convert('L')
display(im_grayscale)
```

✏️ Comment faire cette opération depuis un tableau numpy sans passer par Pillow ?