# Introduction au traitement d'images

Enseignement intégré
IT220 | Informatique 2A | 2023-2024

# Chapitre 3 : Couleur

**Rémi Giraud**
*remi.giraud@enseirb-matmeca.fr*
https://remi-giraud.enseirb-matmeca.fr/

# Plan du cours

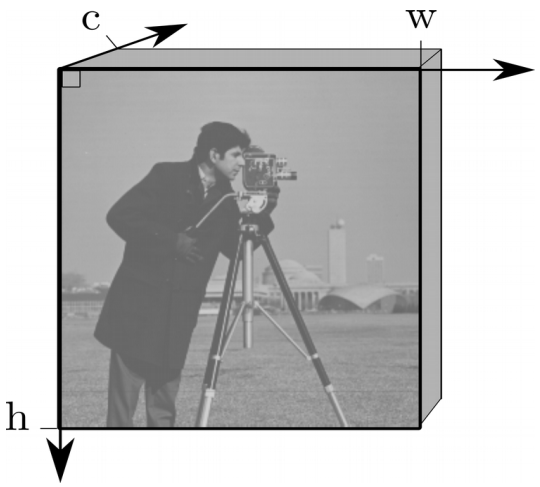
- **Introduction**
- **Formation / Acquisition**
- **Image numérique**
  - Format/Affichage/Synthèse
  - Espaces couleur caractéristiques : compression, esquisse, illusion
- **Traitements**
  - Filtrage linéaire / non linéaire : débruitage, anonymisation
  - Détection de contours : réhaussement de contraste
- **Transformée de Fourier**
  - Application : recouvrement fréquentiel
- **Compression d'images**
  - Application : algorithme JPEG
- **Transformation spatiales**

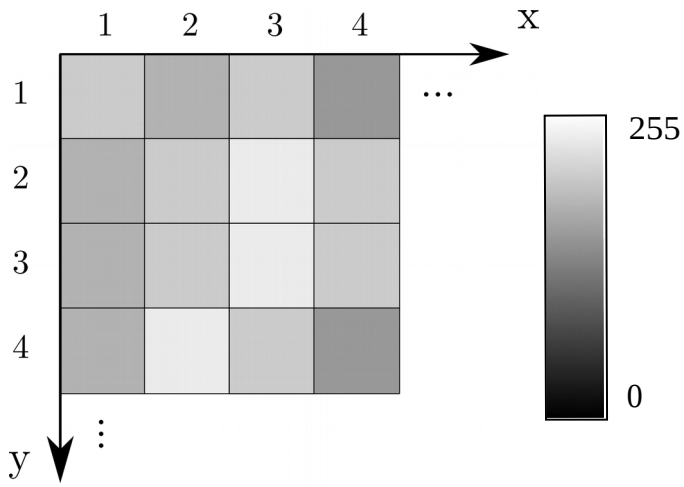
## Qu'est-ce qu'une image numérique ? (1/2)

- Image synthétisée ou obtenue par un dispositif de numérisation
- Processus de discrétisation du monde réel
- Résolution d'une image : nombre de pixels sur une longueur donnée

## Qu'est-ce qu'une image numérique ? (2/2)

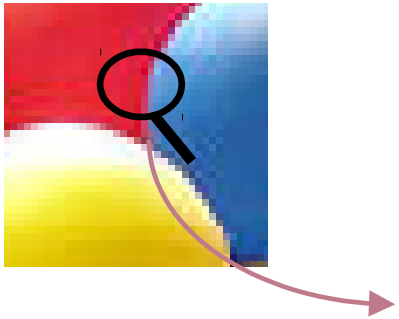
- Une image 2D I est un tableau à deux dimensions associée a :
  - Une taille : h x w
  - Un nombre de canaux : c (ex. pour une image couleur c=3)
- Chaque élément de ce tableau est un pixel, associé à :
  - Une position (i,j) avec i∈[1,h] et j∈[1,w]
  - Une couleur ou intensité I(i,j) (format usuel [0, 255])



## Pour une image couleur

- 3 composantes/canaux
  - Rouge (R), Vert (V), Bleu (B)



| | | | |
|---|---|---|---|
| R=212 G=16 B=40 | R=205 G=65 B=112 | R=103 G=120 B=176 | R=62 G=127 B=193 |
| R=201 G=26 B=43 | R=197 G=69 B=94 | R=154 G=106 B=148 | R=98 G=117 B=186 |
| R=192 G=101 B=106 | R=138 G=59 B=80 | R=127 G=96 B=137 | R=97 G=129 B=188 |
| R=255 G=250 B=250 | R=230 G=192 B=213 | R=140 G=118 B=156 | R=73 G=97 B=145 |
| R=250 G=248 B=251 | R=255 G=248 B=255 | R=255 G=246 B=255 | R=182 G=176 B=210 |

Intensité vectorielle

| | | | |
|---|---|---|---|
| 212 | 205 | 103 | 62 |
| 201 | 197 | 154 | 98 |
| 192 | 138 | 127 | 97 |
| 255 | 230 | 140 | 73 |
| 250 | 255 | 255 | 182 |

R

⊕

| | | | |
|---|---|---|---|
| 16 | 65 | 120 | 127 |
| 26 | 69 | 106 | 117 |
| 101 | 59 | 96 | 129 |
| 250 | 192 | 118 | 97 |
| 248 | 248 | 246 | 176 |

V

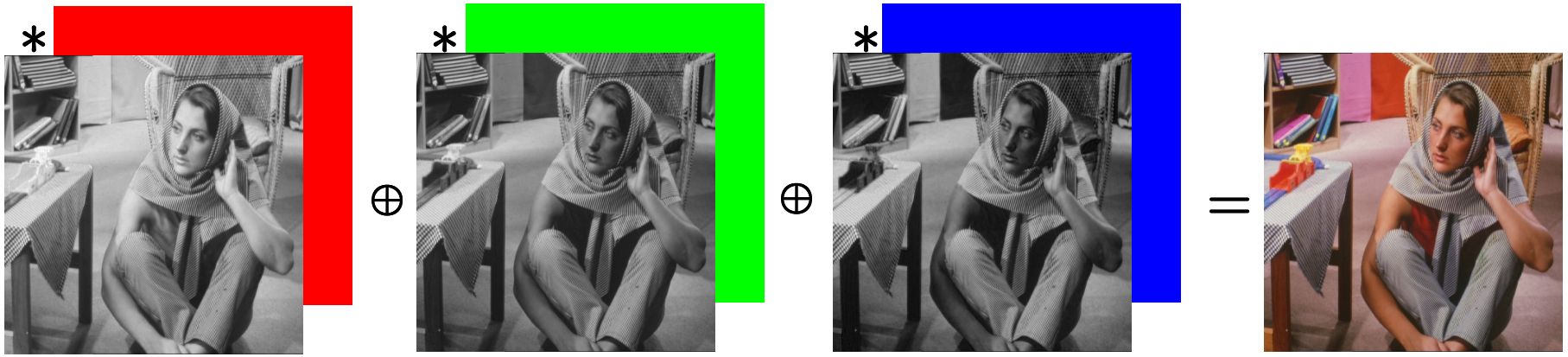
⊕

| | | | |
|---|---|---|---|
| 40 | 112 | 176 | 193 |
| 43 | 94 | 148 | 186 |
| 106 | 80 | 137 | 188 |
| 250 | 213 | 156 | 145 |
| 251 | 255 | 255 | 210 |

B

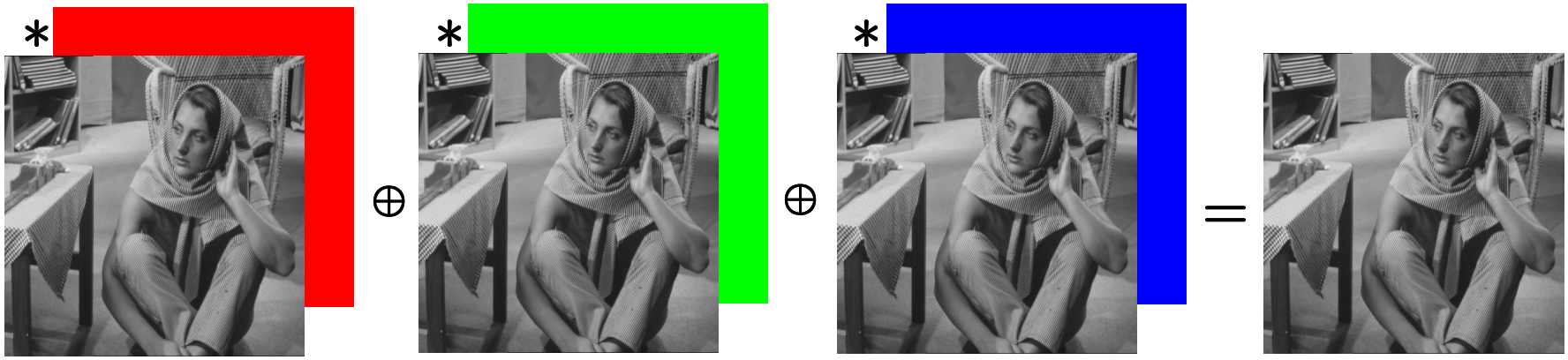
## Comment afficher une image couleur ?

- Trois canaux d'intensité associés à une couleur primaire :



## Comment afficher une image couleur ?

- En niveaux de gris : Triplication du seul canal L



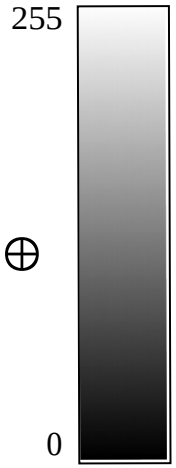

- Note : pour passer facilement d'une image couleur à une image à un seul canal qu'on peut afficher en « niveaux de gris », on peut calculer la luminance **L : (R+V+B)/3**

## Comment afficher une image avec un seul canal ?

- En fausses couleurs ou couleurs indéxées :
  - Une palette (table de correspondance couleur) associe une couleur à chaque intensité scalaire

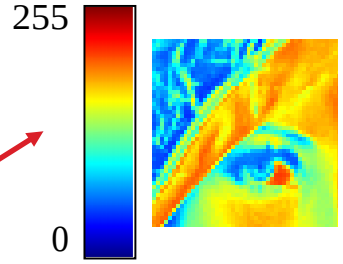
| | | | | | |
|---|---|---|---|---|---|
| 60 | 64 | 62 | 62 | 55 | 51 |
| 59 | 73 | 98 | 90 | 66 | 54 |
| 72 | 105 | 167 | 170 | 120 | 74 |
| 88 | 91 | 188 | 202 | 184 | 150 |
| 103 | 77 | 191 | 205 | 203 | 190 |
| 75 | 131 | 208 | 209 | 207 | 202 |

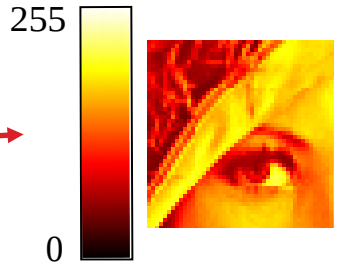
Intensité scalaire

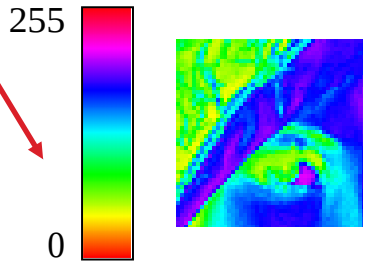



Palette
(niveaux de gris)

Autres palettes

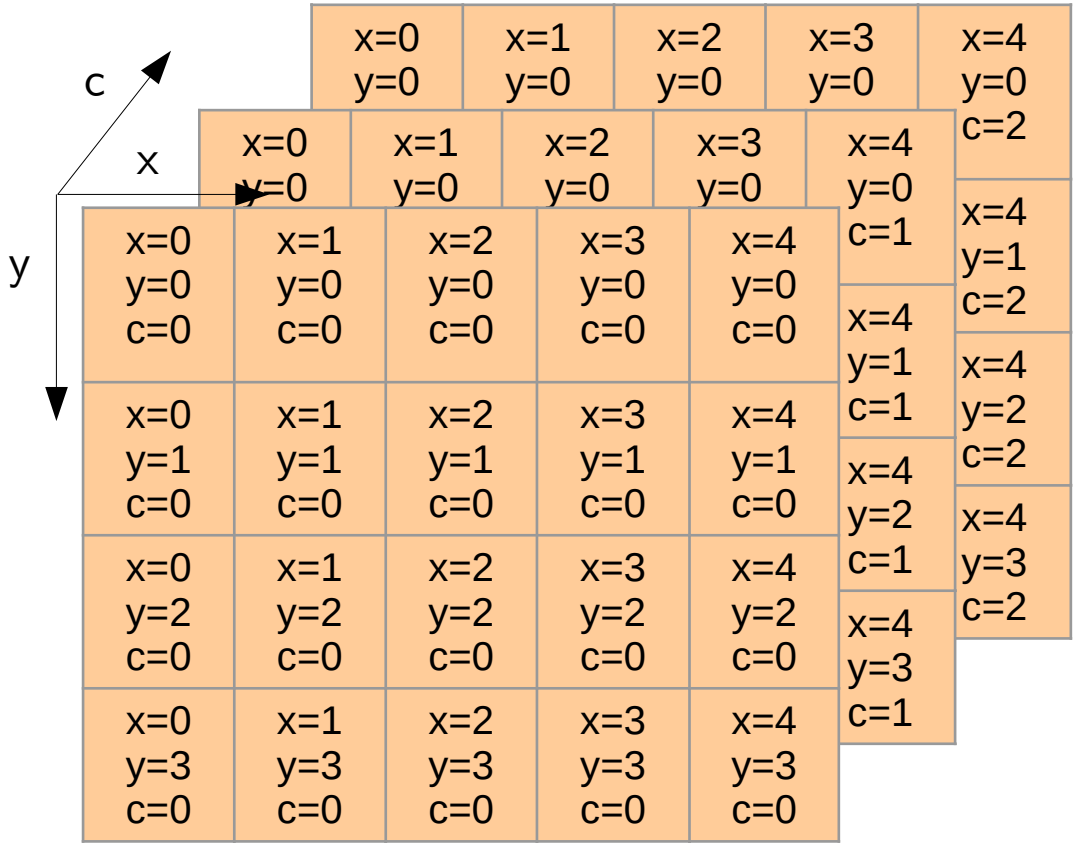


## Conventions

- Accès à une image RGB = Accès à un tableau tridimensionnel




~~img[x,y,canal]~~

img[ligne,colonne,canal]

=

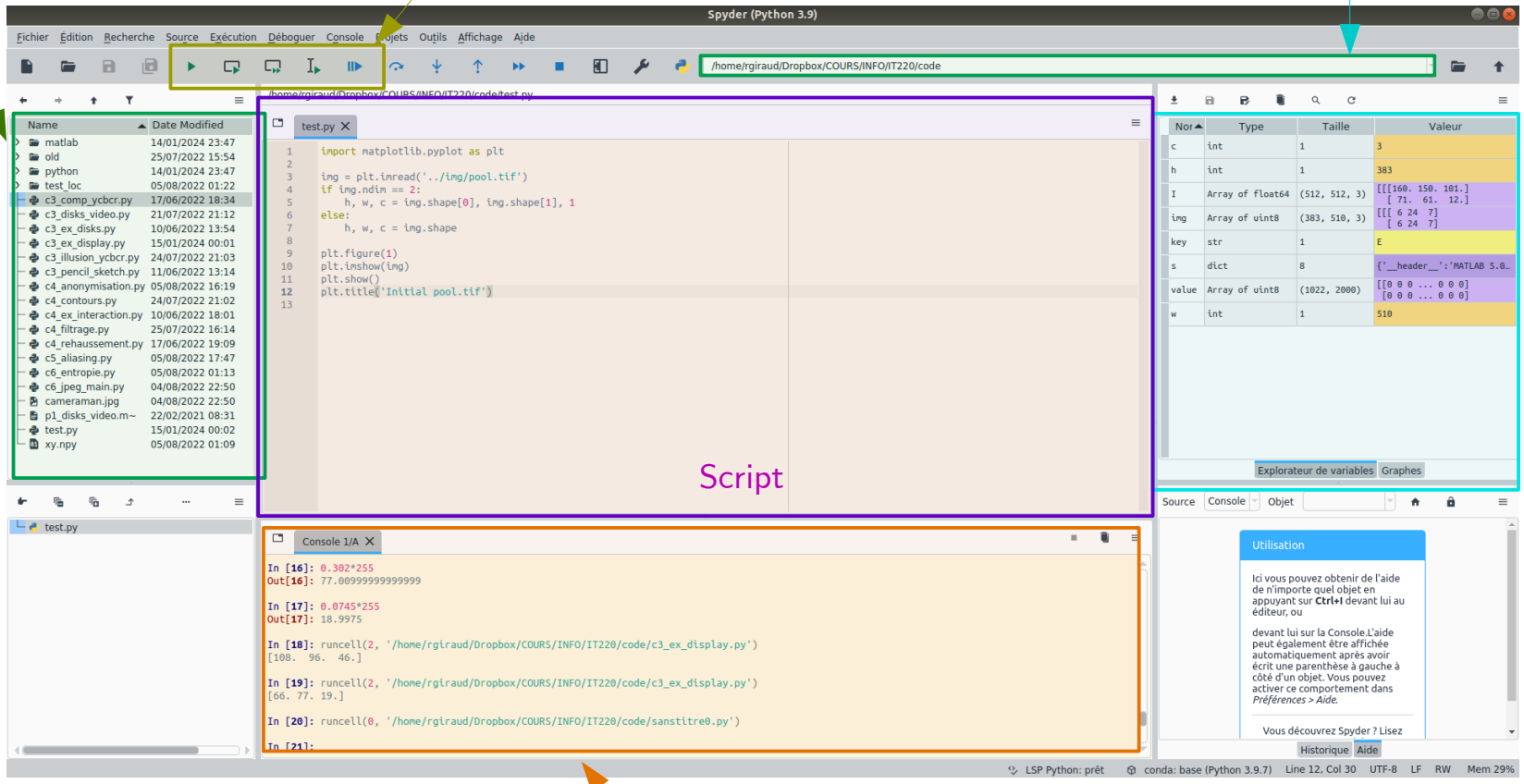
img[y,x,canal]

# Interface de Spyder

Variables déclarées
(prendre l'habitude de vérifier
leur taille)

Dossier courant (créer et se placer
dans un dossier spécifique au cours)

Exécuter le script




Console : Sortie d'affichage standard
Permet également de taper des commandes

## Dataset d'images : https://remi-giraud.enseirb-matmeca.fr/teaching/

## Image en « vraies couleurs »

```
>> import matplotlib.pyplot as plt
>> img = plt.imread('../img/bdx.jpg')
>> whos (dans la console)

Variable   Type       Data/Info
------------------------------
img        ndarray    383x510x3: 585990 elems, type `uint8`, 585990 bytes

>> plt.figure(1)
>> plt.imshow(img)
>> plt.show()
```

hauteur

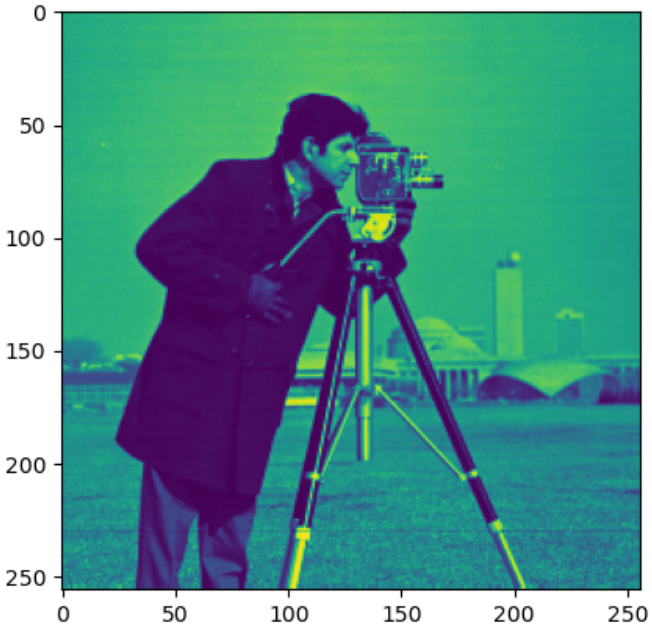
largeur

« vraies » couleurs

## Image en « fausses couleurs »

```
>> img = plt.imread('./img/cameraman.tif')
>> plt.figure(1)
>> plt.imshow(img)
>> plt.show()
```

Et dans l'explorateur de fichiers, l'image ressemble à cela ?



## Palette couleur

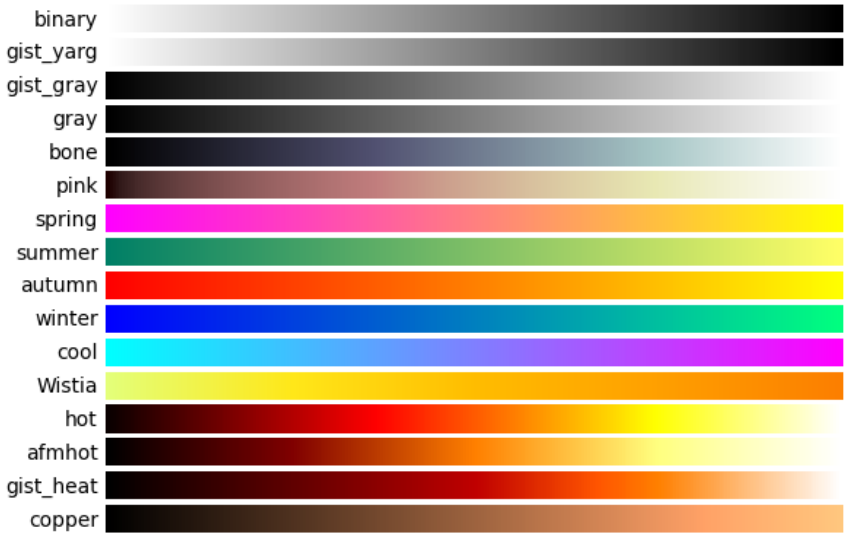
```
>> from matplotlib import cm
>> map = cm.gray(range(256))

[[0.         0.         0.         1.        ]
 [0.00392157 0.00392157 0.00392157 1.        ]
 [0.00784314 0.00784314 0.00784314 1.        ]
 ...
 [0.99215686 0.99215686 0.99215686 1.        ]
 [0.99607843 0.99607843 0.99607843 1.        ]
 [1.         1.         1.         1.        ]]
```

R, G, B

↔ 0

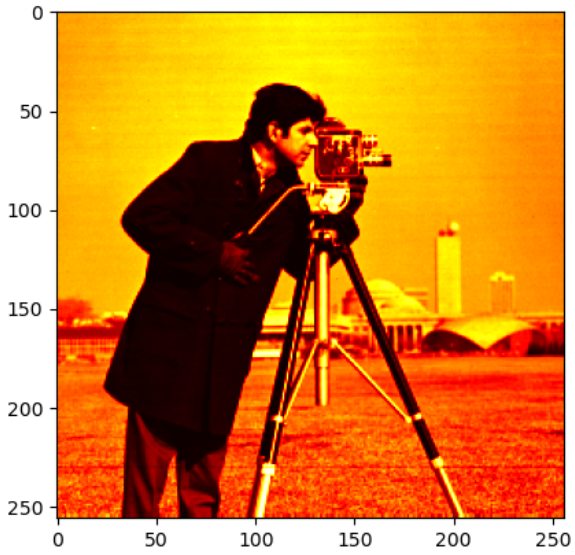
↔ 255

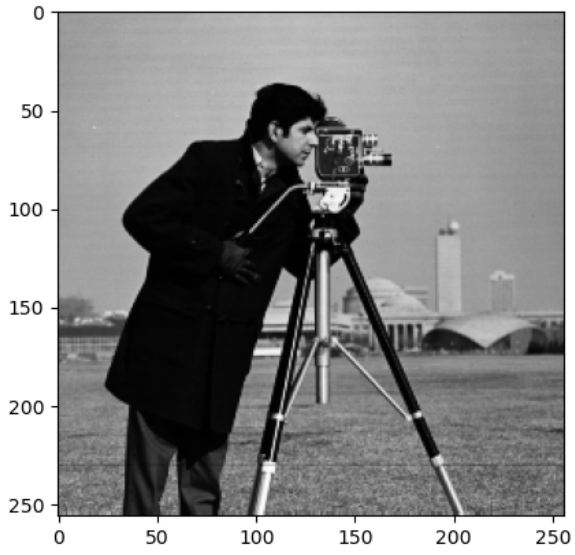
## Autres palettes



## Image en « fausses couleurs »

```
>> from matplotlib import cm
>> from matplotlib.colors import ListedColormap

>> img = plt.imread('../img/cameraman.tif')
>> map = cm.hot(range(256))
>> plt.figure(1)
>> plt.imshow(img, cmap=ListedColormap(map))
>> map = cm.gray(range(256))
>> plt.figure(2)
>> plt.imshow(img, cmap=ListedColormap(map))
```



## Comment extraire les informations d'une image ?

- Bilan des caractéristiques de l'image
  - Dimensions spatiales
  - Codage : « vraies couleurs » couleurs indexées
  - Format numérique
  - Intervalles d'intensité
  - Répartition des intensités

```
h, w, c = img.shape
```

```
whos (dans la console)
```

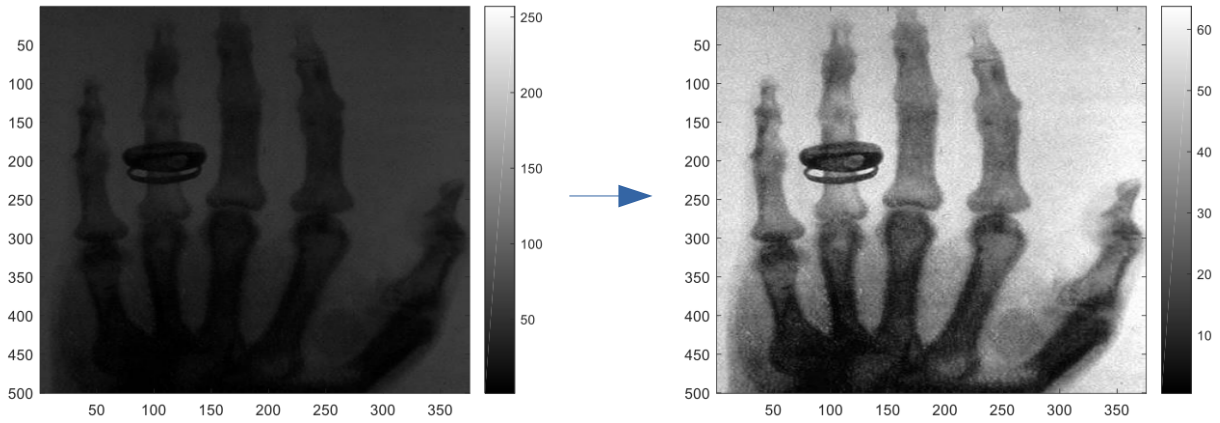
```
[np.min(I(:)) np.max(I(:))]
```

```
np.histogram(I)
```

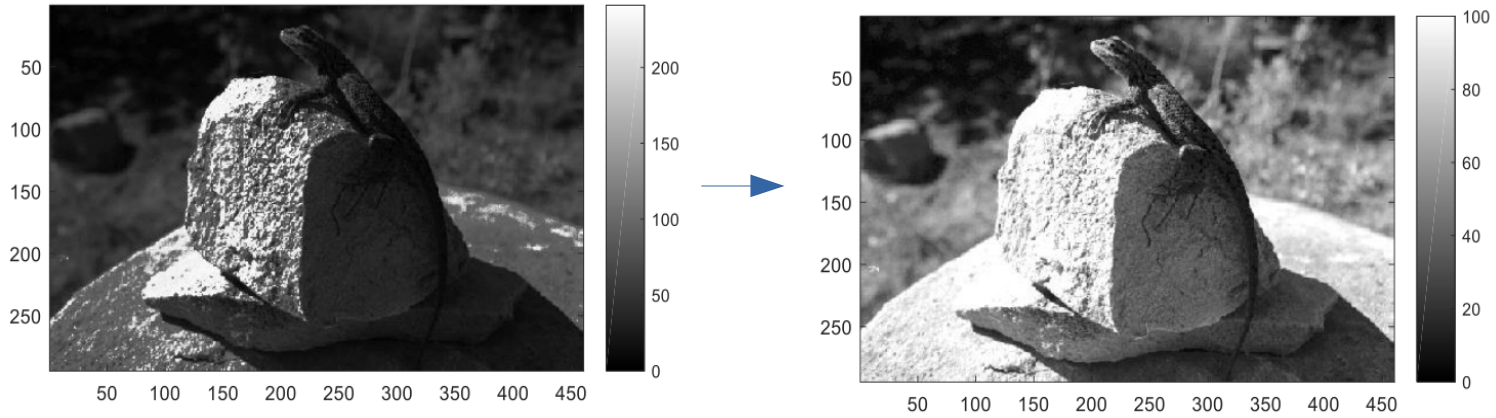
- Identification de la fonction d'affichage
  - Déterminer le contexte
    - Informationnel
    - De fidélité
    - De comparaison
  - Choix : palette, ratio L/H, intervalles d'intensité, etc.

## **Contexte « informationnel »** (couleurs indéxées)

### Étalement automatique des intensités (`imshow(I)`)




### Étalement contrôlé (`imshow(I, vmin=X, vmax=X)`)

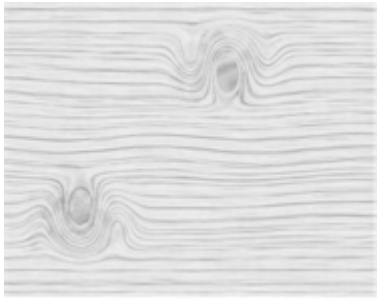


## Contexte de « fidélité »

Respect des intensités initiales
(`imshow(I, vmin=0, vmax=255)`)

*insaturée*

*sombre*

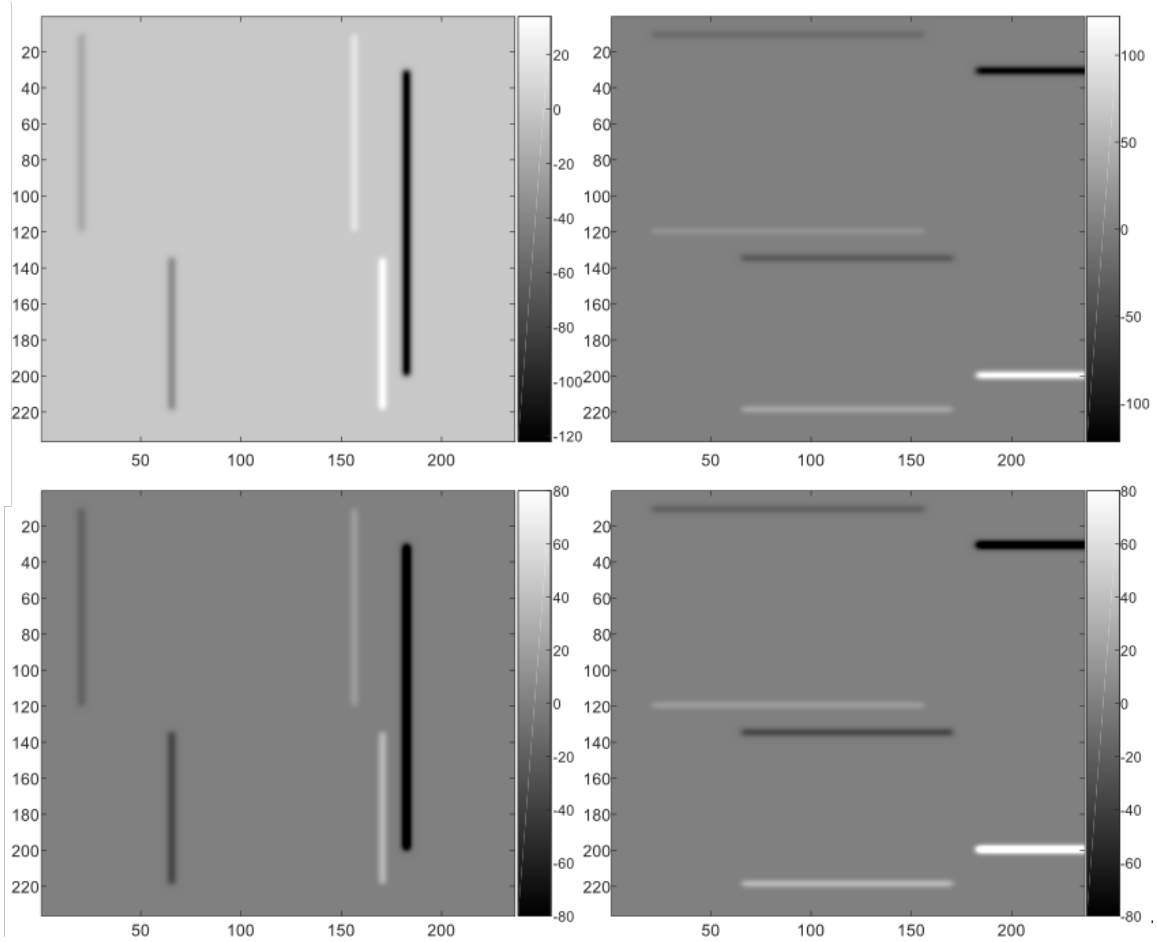


*claire*

*saturée*

## **Contexte de « comparaison »** (couleurs indéxées)

Même étalement contrôlé (`imshow(I, vmin=X, vmax=X)`)

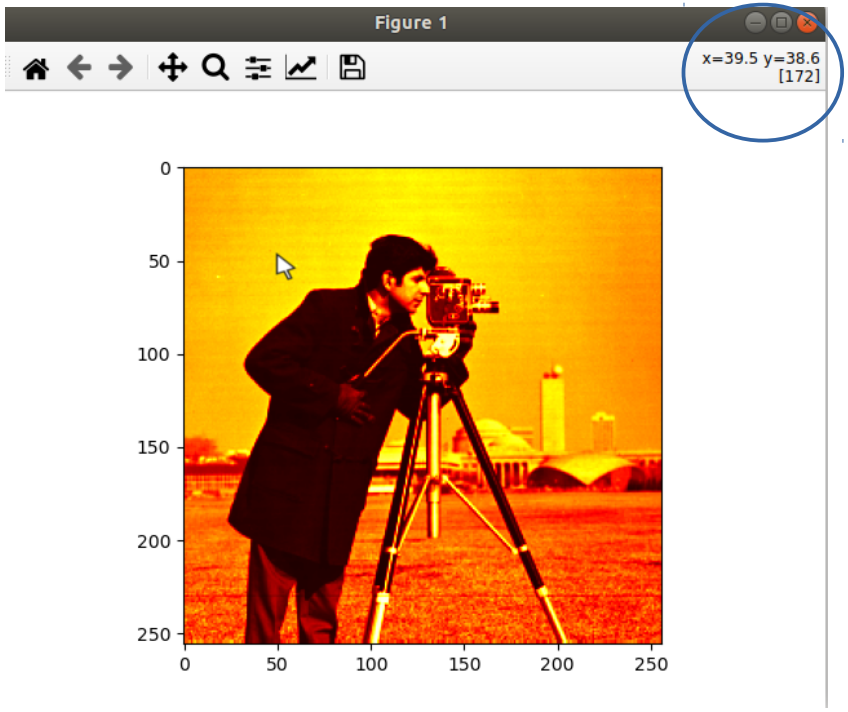


**Auto**

**Contrôlé**

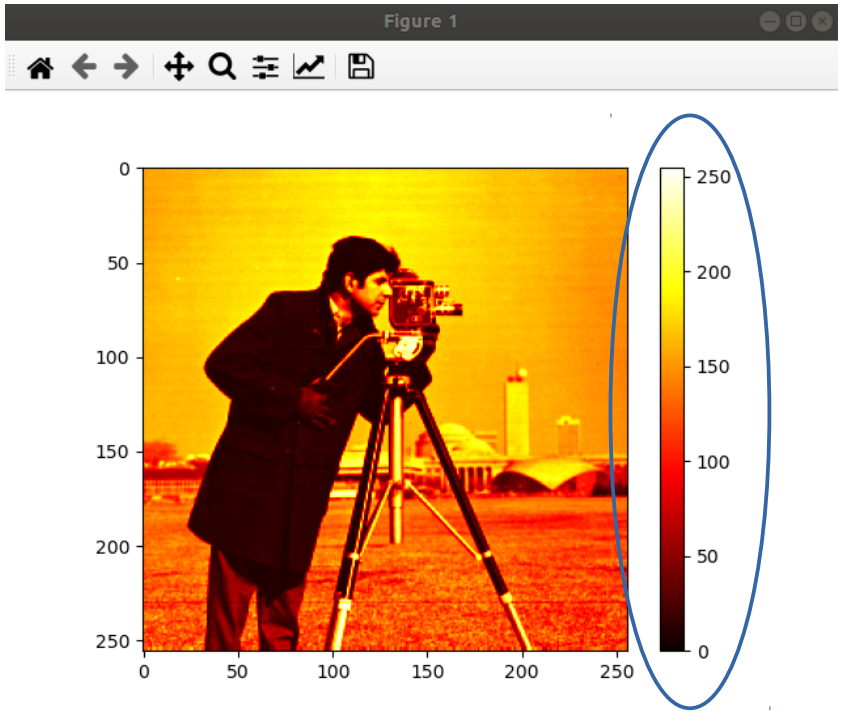
## Data cursor

Pour visualiser rapidement les intensités dans une région




## Colorbar

Pour visualiser la gamme des intensités d'une image à 1 canal (`plt.colorbar()`)



# Mémo Python (1/2)

- Bonnes pratiques :
  - Utiliser un vrai éditeur de code Python (spyder, jupyter-notebook, etc.)
  - Se placer dans un dossier dédié
  - Toujours écrire dans un script (*fichier.py*)
  - Surveiller le workspace pour voir quelles sont et surtout la taille des variables
  - Consulter l'aide des fonctions (`help(function)`)
- Modules :
  - Accéder aux fonctions d'autres modules :

```
import numpy as np #toutes les fonctions,
     possibilité de renommer le module
from signal import convolve
  #import d'une seule fonction
import subfolder.my_module
  #import de ./subfolder/my_module.py
```

- Fonctions :
  - Peuvent être écrites dans le script (avant le code appelant, comme en C)

```
def min_max(T):
    min_ = min(T)
    max_ = max(T)
    return min_, max_
```

- Exécution (*spyder*) :
  - Tout le script : **F5** ou **<Run>**
  - Par section : **ctrl+enter** ou **<Run section>**

```
h, w, c = img.shape

#%% Affichage
plt.figure(), plt.imshow(img)

#%% Vectorisation
img_vect = img(:)
```

  - Par sélection : **F9**

```
h, w, c = img.shape

#Affichage
plt.figure(), plt.imshow(img)
```

- Principales différences avec Matlab :
  - Indices [ ], à partir de 0 : tableau de taille `l`

```
tab[0] #premier élément
tab[l-1] = tab[-1] #dernier élément
```

  - Vecteur d'échantillonnage :

```
range(0,100) #0, 1, ... , 99
```

  - Opération terme à terme par défaut :

```
a = np.array([1,2,3,4])
a*a #array([1,4,9,16])
```

# Mémo Python (2/2)

- Commandes de bases :

```python
#Modules utiles
import numpy as np #tableau, opérateurs maths
import matplotlib.pyplot as plt #image, affichage
import skimage  #par ex. espace couleur ycbcr
from scipy import signal #par ex. convolution
import cv2 #opencv, par ex. vidéo, imwrite

#Manipulation d'image
img = plt.imread('path/img.png') #chargement
h, w, c = img.shape  #image couleur
print(h)

img_vect = img.ravel()  #Vectorisation
G = img[:,:,1] #accès dimension 2e canal = vert

#Mise à zéro
img = np.zeros((h,w,c)) #ones() existe aussi
img = np.copy(img*0)

#Sous-échantillonnage
img_se = img[1:h:4, 1:w:4] #ou img[::4,::4]

#Création d'un vecteur/d'une matrice
mat = [[1,1],[2,2],[3,3]]  #liste de taille 3x2
mat = np.array(mat)  #np.array

#Produits vecteurs/matriciels
vect = np.array(np.matrix(range(1,11,2))).T
           #range(début,fin,pas)  #T = transposée
vect_5_1 = vect*vect #terme à terme
vect_5_5 = np.dot(vect, vect.T) #prod. matriciel
vect_1_1 = np.dot(vect.T, vect) #prod. matriciel

#Somme sur matrice nD
sum_G = np.sum(G**2) #somme de tous les termes^2
sum_G = np.mean(img, axis=2) #conversion niv. gris
```

```python
#Seuillage sur une matrice
mask = G > 100 #mask = carte binaire (hxw)
#Équivalent à faire :
mask = np.zeros((h,w))
for i in range(0,h):
   for j in range(0,w):
      if (G[i,j]>100):
          mask[i,j] = 1

G[mask==0] = 0 #Mise à zéro des pixels de G où mask=0
G = G*mask #Équivalent à multiplication terme à terme

#changement de type
G = G.astype('uint8') #ou G = np.uint8(G)

#changement d'espace couleur
img_ycbcr = skimage.color.rgb2ycbcr(img)

#convolution image couleur (même filtrage sur R,G,B)
filter_ = np.ones([7,7,1])/49
img_f = signal.convolve(img, filter_, mode='same')

#Affichage
img_L = np.mean(img, axis=2)
plt.figure()
plt.subplot(121) #Affichage multiples 1x2
plt.plt(img_L[0,:])
plt.title('Profil de la première ligne de L')
plt.subplot(122)
plt.plot(img[0,:,0],'ro') #superposition par défaut
plt.plot(img[0,:,1],'g+')
plt.plot(img[0,:,2], color=[0,0,1])
plt.title('Profil RGB de la première ligne')
plt.xlabel('x'), plt.ylabel('Intensité')
plt.show()
```

**Autres fonctions utiles :** `np.squeeze, np.tile, plt.ginput, …`

**Liens vers les docs :** Tuto général Python  Tuto scikit-image
Tuto numpy, matplotlib, scipy

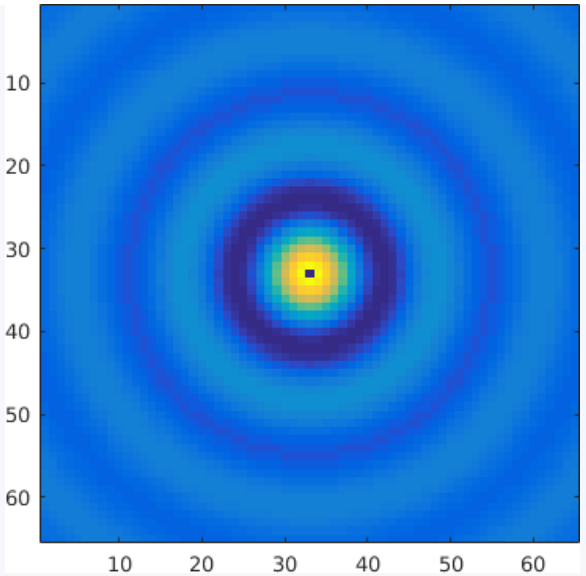
- Trouver une façon satisfaisante d'afficher les images :
  *challenge#A-C.npy*

| Nom | Donnée | Problématique |
|---|---|---|
| Mandrill | challengeA.npy | Format numérique |
| Radio | challengeB.npy | Intervalle d'intensité |
| World map | challengeC.npy | Palette discrète |

```
A = np.load('challengeA.npy')
```

## Synthèse analytique en couleurs indéxées

```
#solution longue (7 lignes)
x = range(-34,35)
y = range(-32,33)
img1 = np.zeros((len(y), len(x)))
for i in range(0, len(y)):
   for j in range(0, len(x)):
       r = np.sqrt(x[j]**2+y[i]**2)
       img1[i,j] = 1000*np.sin(r/2)/r
plt.figure(1)
plt.imshow(img1)
```

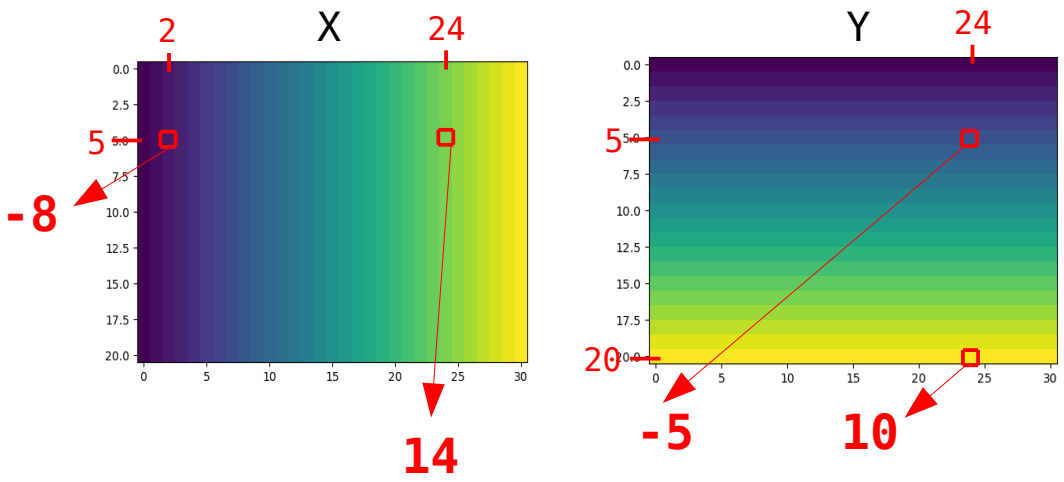



```
%solution courte (3 lignes)
X, Y = np.meshgrid(range(-34, 35), range(-32,33))
R = (X**2 + Y**2)**0.5;
img2 = 1000*np.sin(R/2)/R;
plt.figure(2)
plt.imshow(img2)
```

# Fonction np.meshgrid

**Objectif** : Se passer des boucles de parcours Hauteur/Largeur

**Exemple** : `X, Y = np.meshgrid(range(-10,21), range(-10,11))`



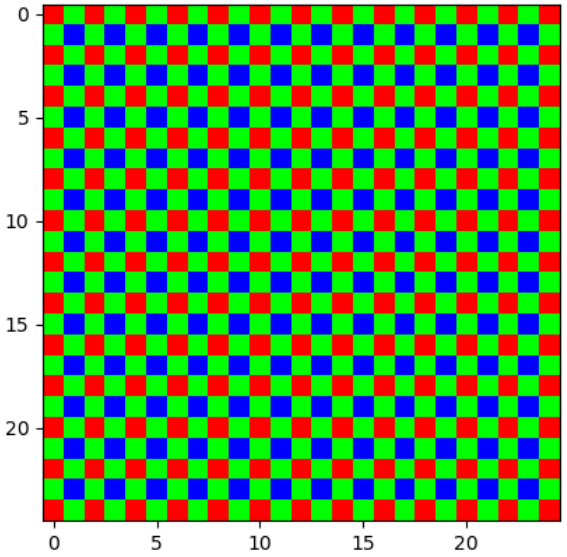

Crée deux matrices de taille 20x30 qui stockent en valeur pour chaque pixel le numéro de colonne (X) et le numéro de ligne (Y)

```
x_v = range(-10, 21)
y_v = range(-10, 11)
X, Y = np.meshgrid(x_v, y_v);
I = X**2 + Y**2
```

```
x_v = range(-10, 21)
y_v = range(-10, 11)
I = np.zeros((len(y_v), len(x_v))
for i in range(0, len(y_v)):
   for j in range(0, len(x_v)):
      I[i,j] = y_v[i]**2 + x_v[j]**2
```

## Synthèse en « vraies couleurs »

```
Nx = 25
Ny = 25
x = range(0, Nx)
y = range(0, Ny)

X, Y = np.meshgrid(x, y)

R = (1-np.mod(X,2))*(1-np.mod(Y,2))
G = np.mod(X+Y,2)
B = (1-np.mod(X+1,2))*(1-np.mod(Y+1,2))

I  = np.stack((R,G,B), axis=2).astype('double')

plt.figure(1)
plt.imshow(I)
```



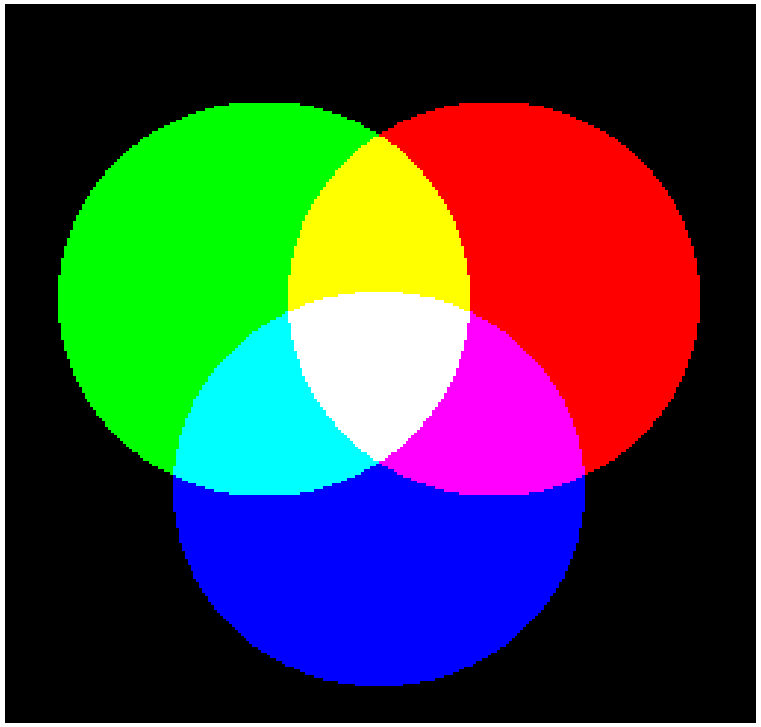
# Synthèse additive

- Écrire une fonction *disk* qui génère cette image :

  En « vraies couleurs »

  En couleurs indexées

  La taille de l'image, la largeur des cercles et leur espacement seront des paramètres.



- Remplissez le programme du slide suivant qui alterne les couleurs des cercles pour créer une vidéo de :
  - 100 images
  - 5 images/seconde (option `FrameRate`)
  - Sans compression

  Fonctions utiles (openCV) :
  - `cv2.VideoWriter, write, release`
    (Bien lire les documentations)

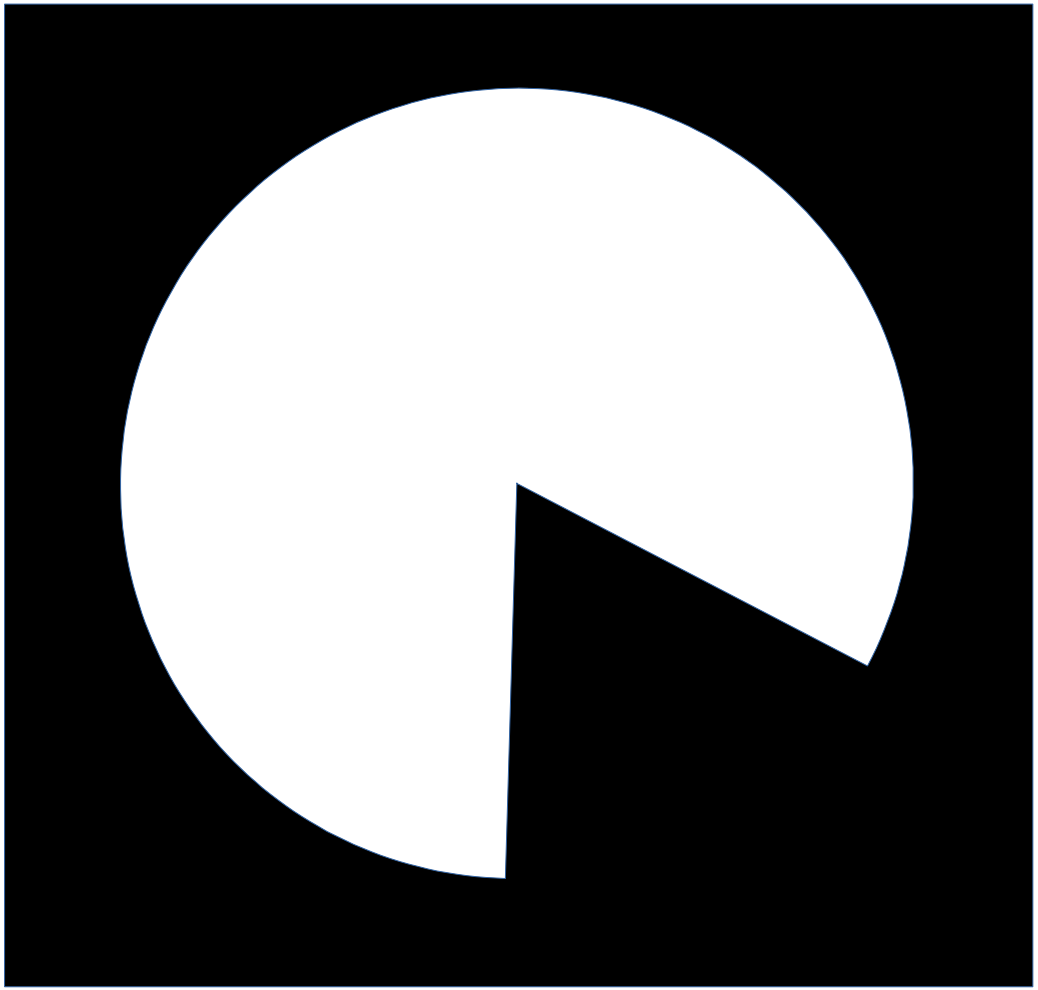
# Synthèse additive dynamique

Exercice

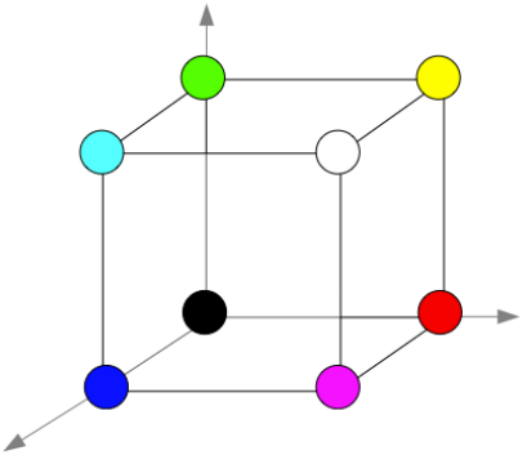
```
size = 255
radius = 70
dist = 45

R, G, B = p1_disks(size, radius, dist)

h, w = R.shape

R = R[:,:,np.newaxis]
G = G[:,:,np.newaxis]
B = B[:,:,np.newaxis]

#Créer l'objet vidéo avec
cv2.VideoWriter
#en définissant le framerate
video = ...
```

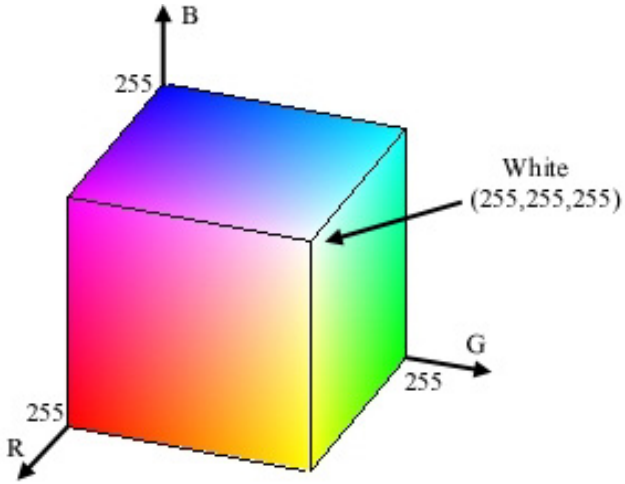
```
for n in range(0, 100):

    q = np.mod(n,3)

      if (q == 0): #C1=R,C2=G,C3=B
        img = np.concatenate((R,G,B),
axis = 2).astype('uint8')
    elif (q == 1): #C1=B,C2=R,C3=G
        img = ...
    else: #C1=G,C2=B,C3=R
        img = ...


    %Ajout du frame dans la vidéo
    ...


%Fermeture de l'objet vidéo
video.release()
```

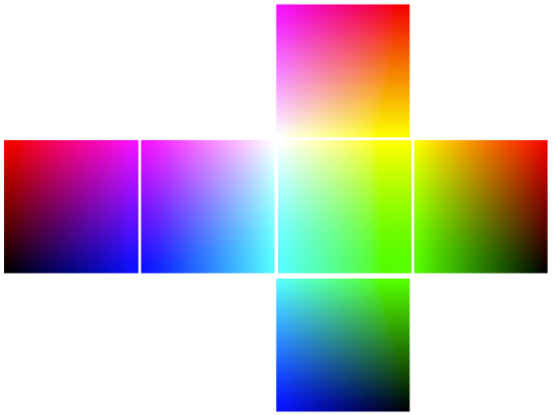
- Créer un timer animé sous forme de camembert progressif



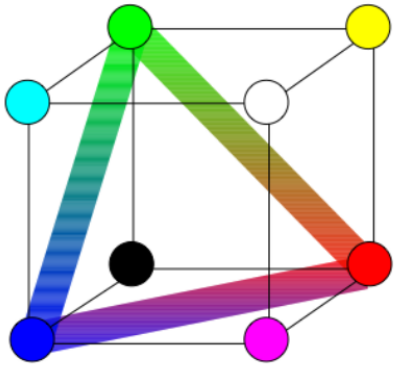
# Cube RGB



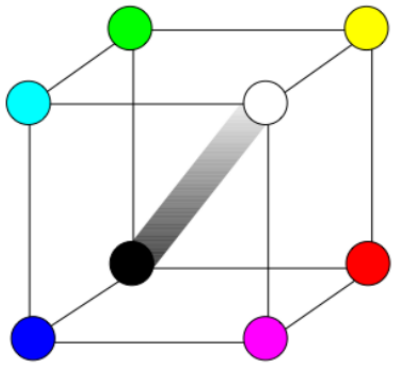


1 triangle chromatique

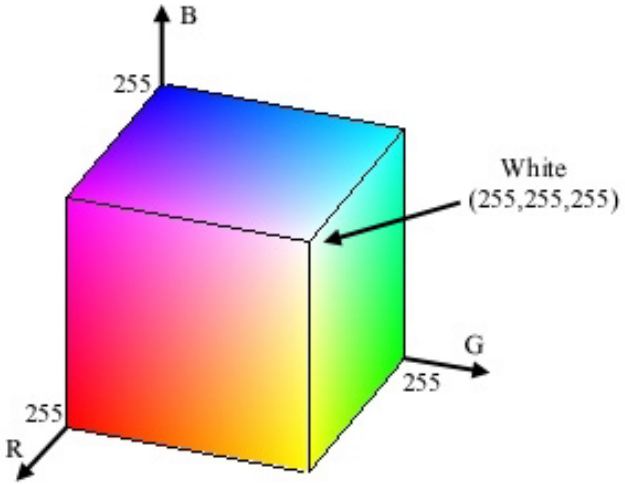


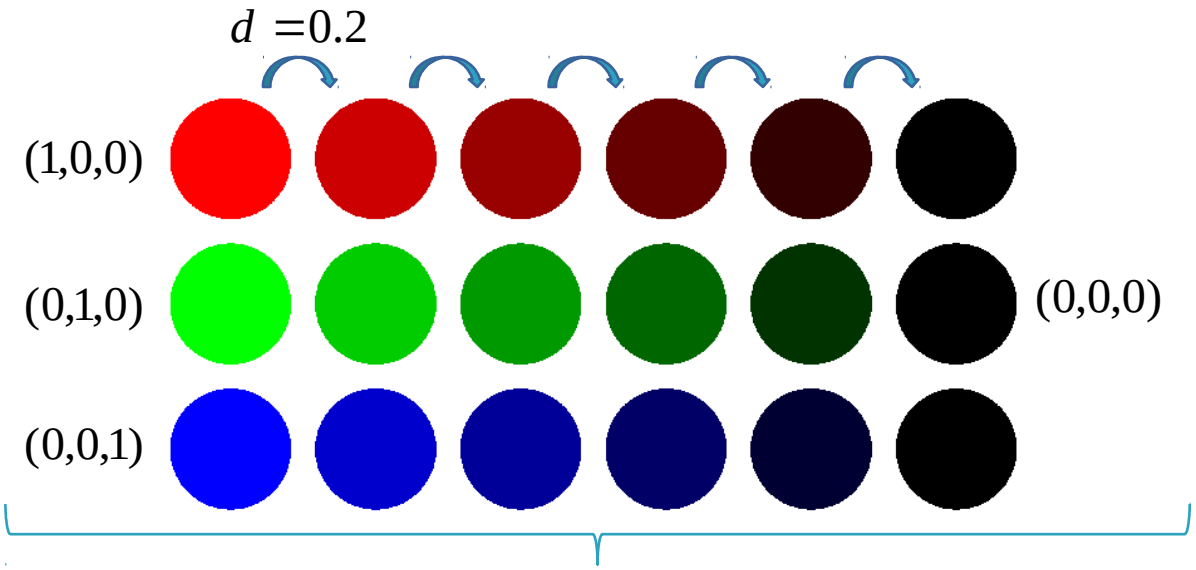
1 axe achromatique
(niveau de gris)



## Espace RGB

- Information de couleur et d'intensité mélangée dans les trois canaux
- Luminance globale donnée par **L = (R+G+B)/3**
- Perception différente selon les trois canaux



## Espace RGB

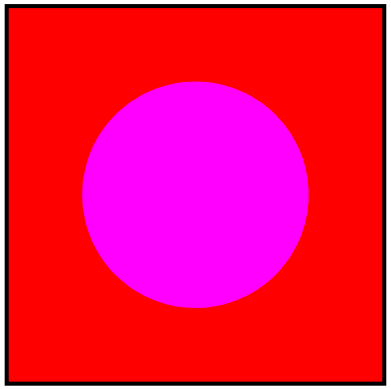
- Sensibilité aux contrastes différente sur les trois canaux


Niveaux de gris

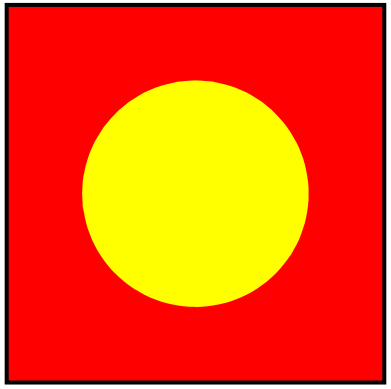
Palettes de primaires pures
(intensités identiques et autres composantes éteintes)

## Canal Y de luminance « perceptuelle »

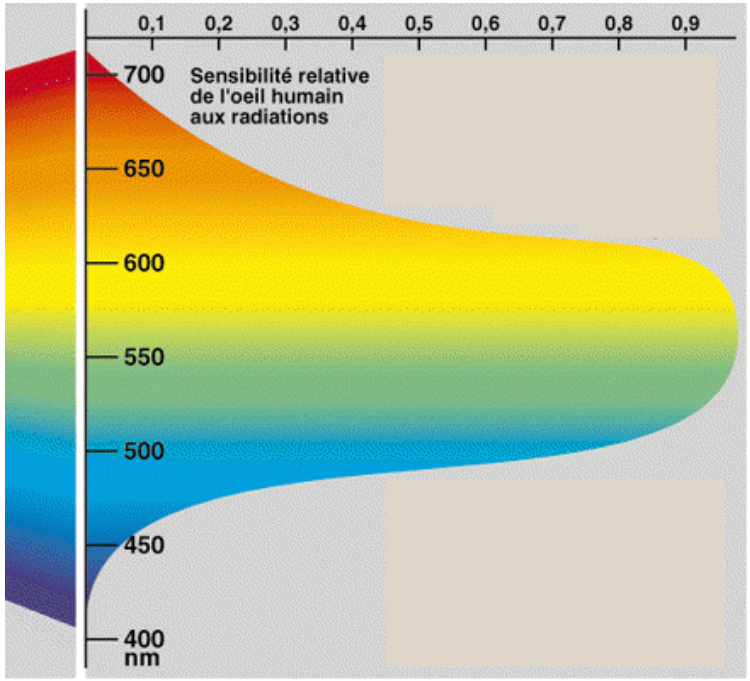
- Tient compte de la sensibilité aux couleurs de l'oeil humain



Rouge/Bleu



Rouge/Vert




$$Y = 0.299\,R + 0.587\,V + 0.114\,B$$

pondération plus faible

## Espace XYZ

- Introduction de l'espace CIE-XYZ : XYZ → LAB, XYZ → LUV, ...
- Changement d'espace linéaire

{(A,B,C) = M.(R,G,B)} $\longrightarrow$ **(A',B',C')** $\longrightarrow$ {(R'G'B') = $M^{-1}$.(A',B',C')}

$$\text{où M} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & g \end{pmatrix}$$

Par ex. : RGB$\leftrightarrow$XYZ

$$\text{M} = \begin{pmatrix} 0.41245 & 0.35758 & 0.18042 \\ 0.21267 & 0.71516 & 0.07216 \\ 0.01933 & 0.11919 & 0.95022 \end{pmatrix} \text{ et } \text{M}^{-1} = \begin{pmatrix} 3.24047 & -1.53715 & -0.49853 \\ -0.96925 & 1.87599 & 0.04155 \\ 0.05564 & -0.20404 & 1.05731 \end{pmatrix}$$

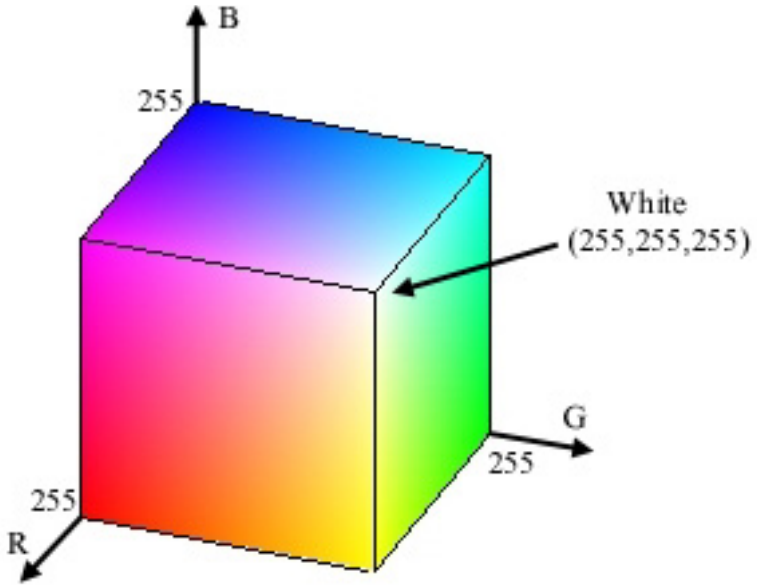
## Espace RGB

- Par défaut
  - Canaux très corrélés
  - Non perceptuel

R

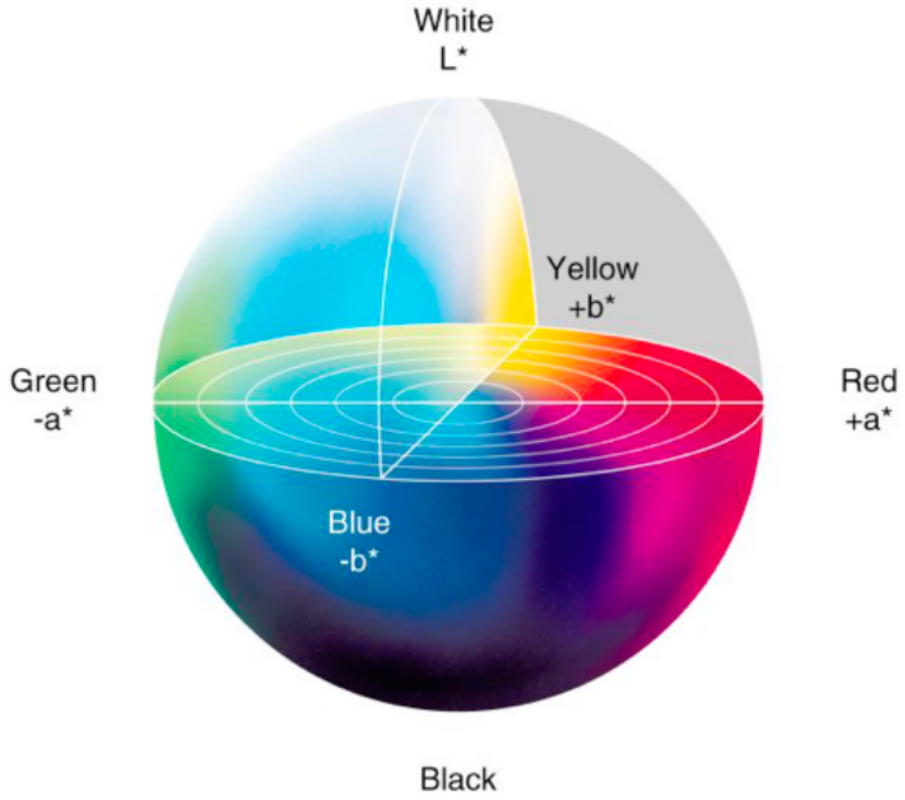
G

B



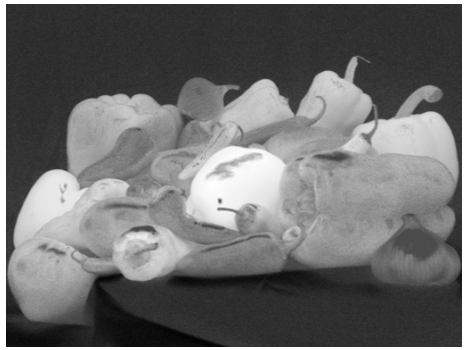
## Espace L*a*b*

- Luminance - Chrominance
  - Espace perceptuellement uniforme




L*

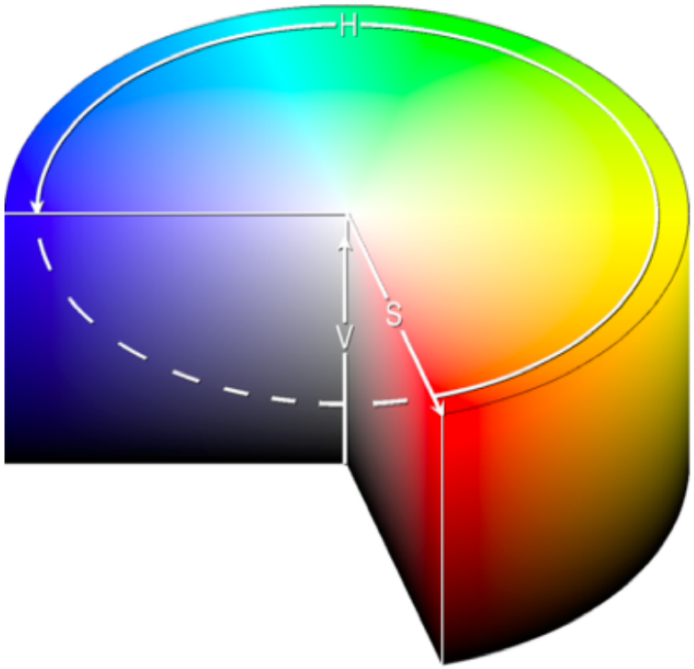
a*



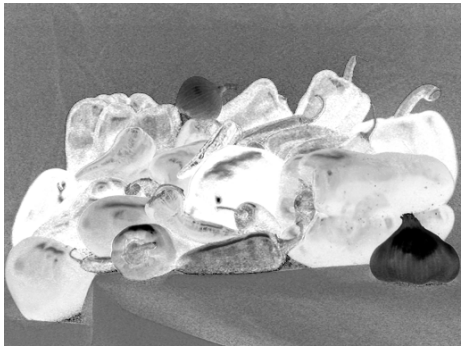
b*

## Espace HSV

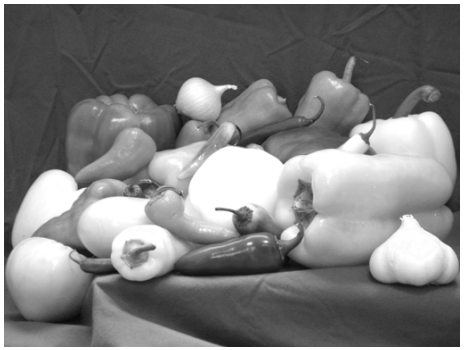
- Teinte – Saturation – Valeur
  - (Hue – Saturation – Value)
  - Utile dans les applications graphiques




H



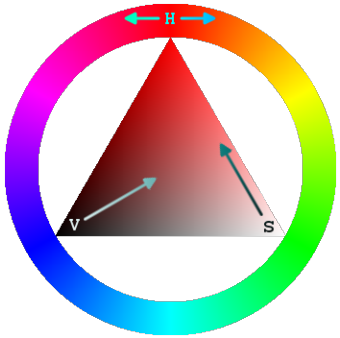
S



V

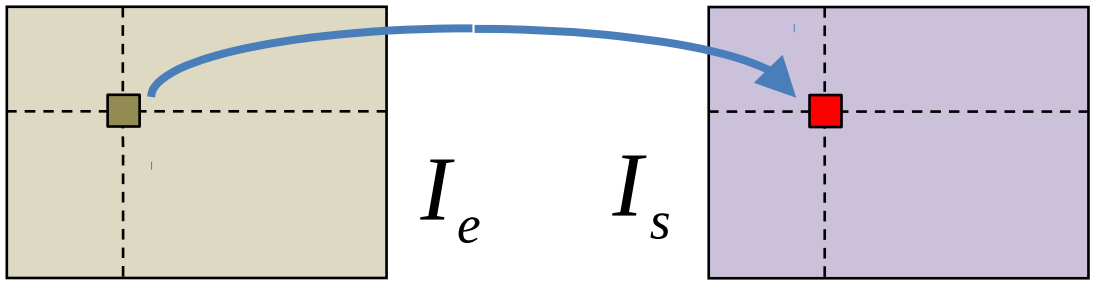
## Espace HSV

- Teinte – Saturation – Valeur
  - (Hue – Saturation – Value)
  - Utile dans les applications graphiques




## De nombreux espaces couleurs

- RGB : acquisition/restitution écran
- HSV/HSL : espace intuitif, vision humaine
- YUV/YCbCr : transmission et codage
- La*b*/Lu*v* : espace uniforme, distance entre les couleurs
- CMY : impression
- XYZ : modélisation des couleurs
- espaces non linéaires
- etc

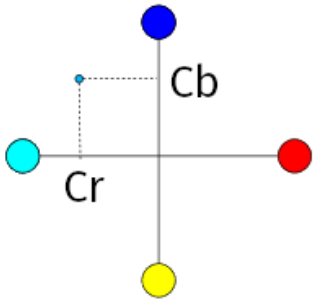


## Espace YCbCr

- Un canal de luminance **Y** et deux canaux de chrominance **Cb Cr**
- Utilisé par ex. pour la compression et la transmission hertzienne
- Les composantes sont obtenues par les formules :

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = 0.564(B - Y) + 128 \\ Cr = 0.713(R - Y) + 128 \end{cases}$$

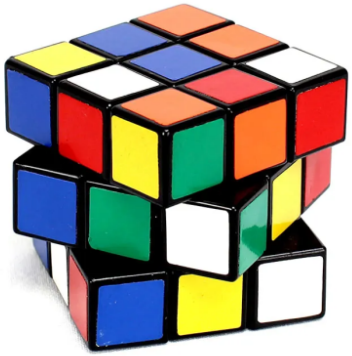
- Les canaux **Cb** et **Cr** correspondent respectivement aux contrastes Bleu/Jaune et Rouge/Cyan.

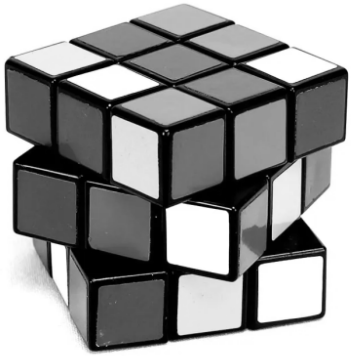


## Espace YCbCr

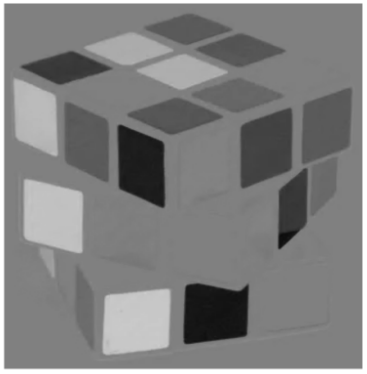
```
I = plt.imread('pool.tif')
I = I.astype(‘double’)
R = I[:,:,0]
G = I[:,:,1]
B = I[:,:,2]

Y  = 0.299*R+0.587*G+0.114*B;
Cb = 0.564*(B-Y)+128;
Cr = 0.713*(R-Y)+128;
L  = (R+G+B)/3;
```

```
plt.figure(1)
plt.imshow(I.astype(‘uint8’))
plt.title('I')

plt.figure(2)
plt.imshow(L.astype(‘uint8’))
plt.title('L')

plt.figure(3)
plt.imshow(Y.astype(‘uint8’))
plt.title('Y')
```

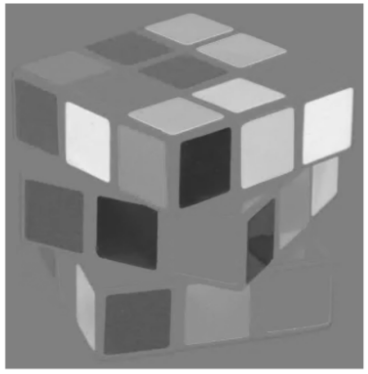




Y

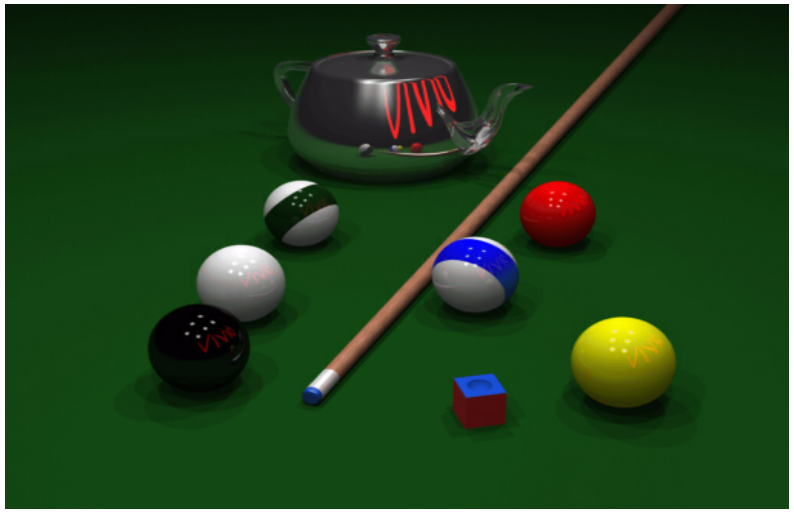


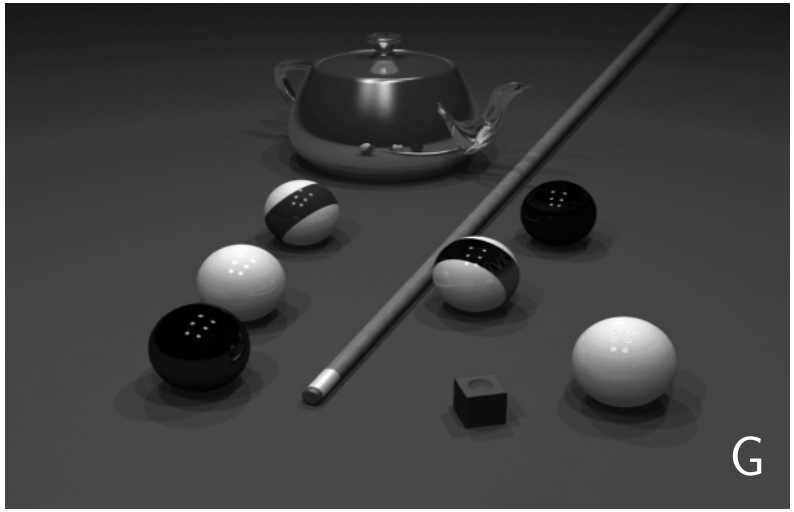
Cb



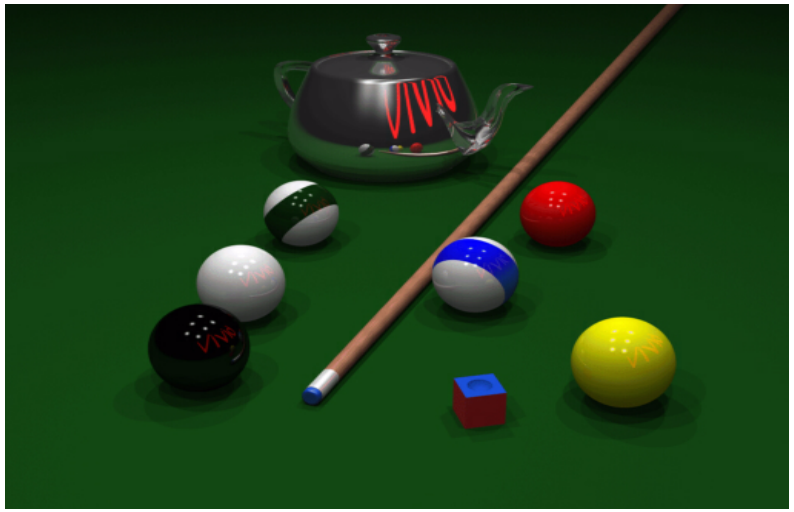
Cr

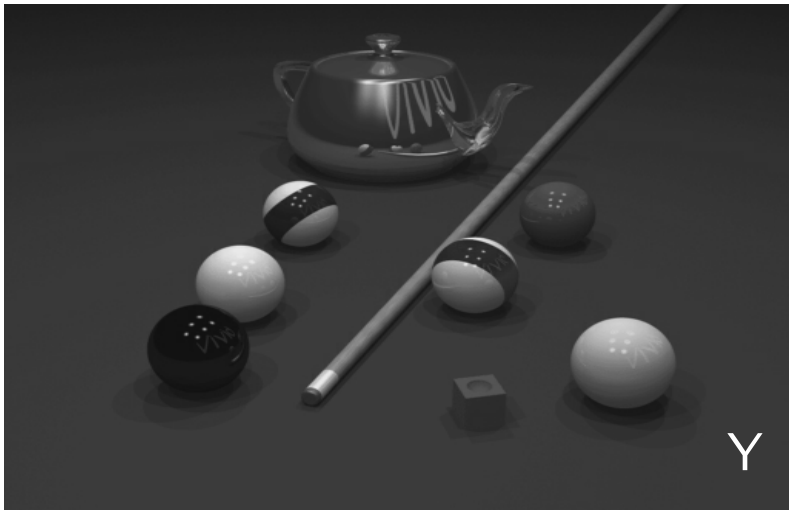
## Espace RGB vs YCbCr

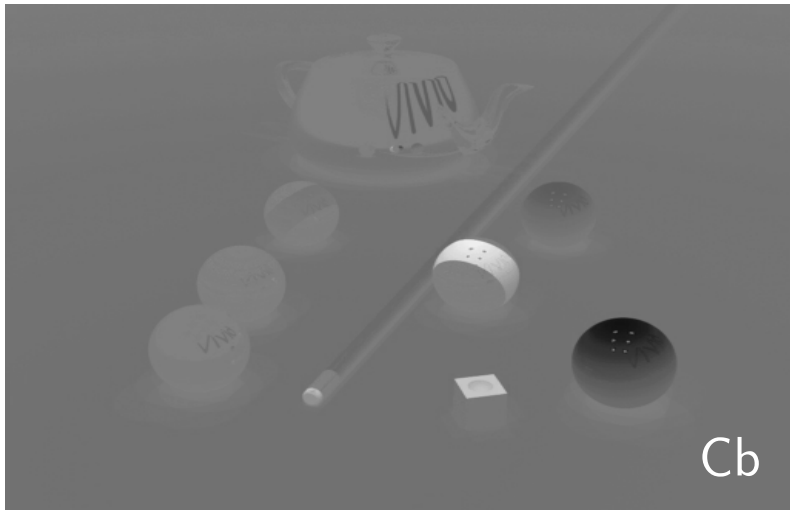


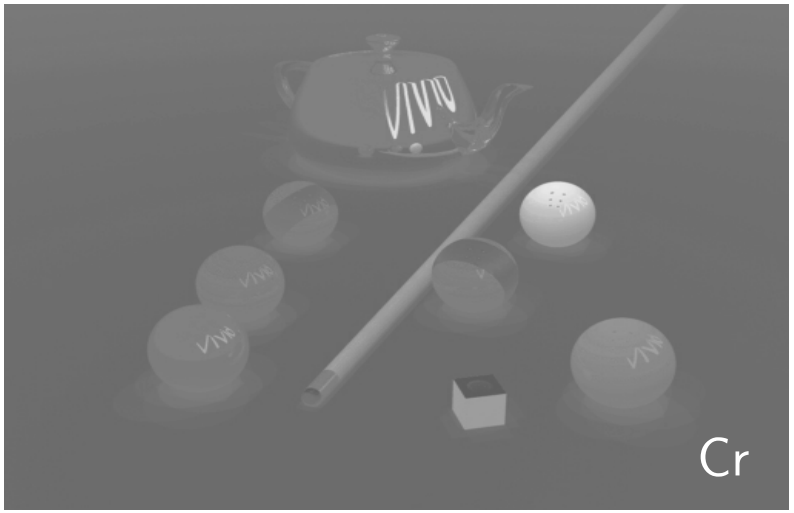




## Espace RGB vs YCbCr



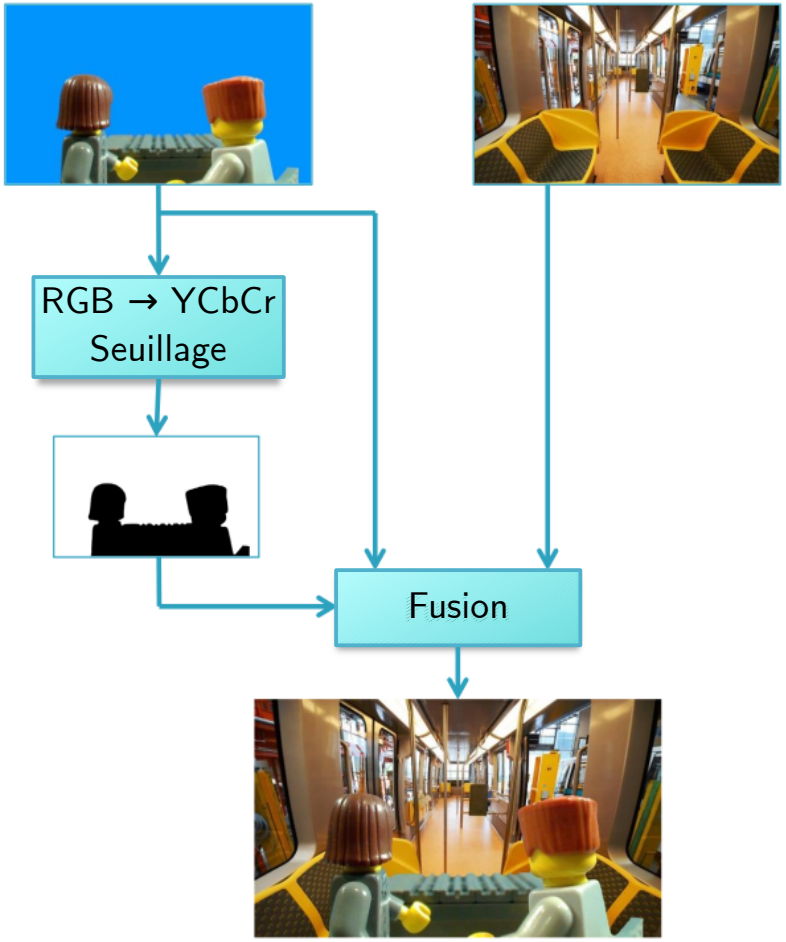


## Exemple d'application : incrustation (*chroma-keying*)

- Meilleure séparation des objets grâce aux canaux de chrominance




→ Implémentation en TP

## Compression dans l'espace YCbCr

- Convertir l'image *pool.tif* (hxwx3) en YCbCr (`skimage.color.rgb2ycbcr`)
- Compresser en taille uniquement les canaux de chrominance CbCr (`skimage.transform.resize`) par un facteur $r<1$ qu'on fera varier
- Reconstruire l'image en ramenant Cb et Cr à leur taille initiale
- Repasser en RGB (`skimage.color.ycbcr2rgb`) et comparer avec l'image initiale
- Quantifier le gain en taille mémoire

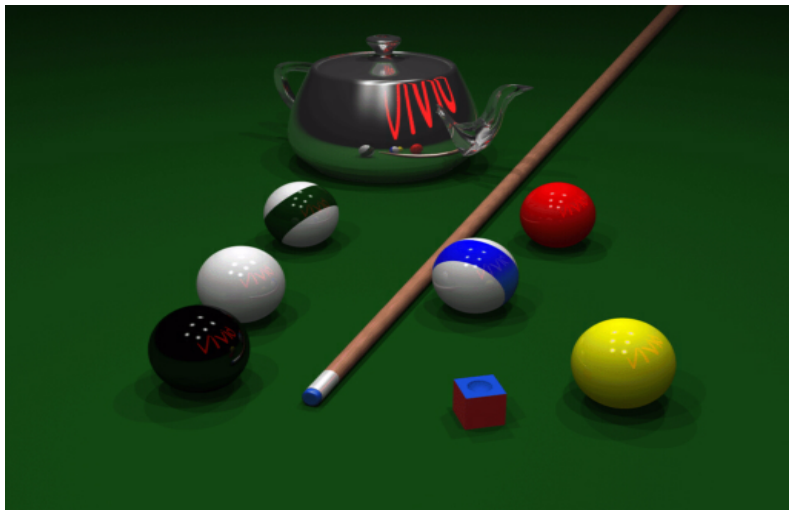


Image initiale

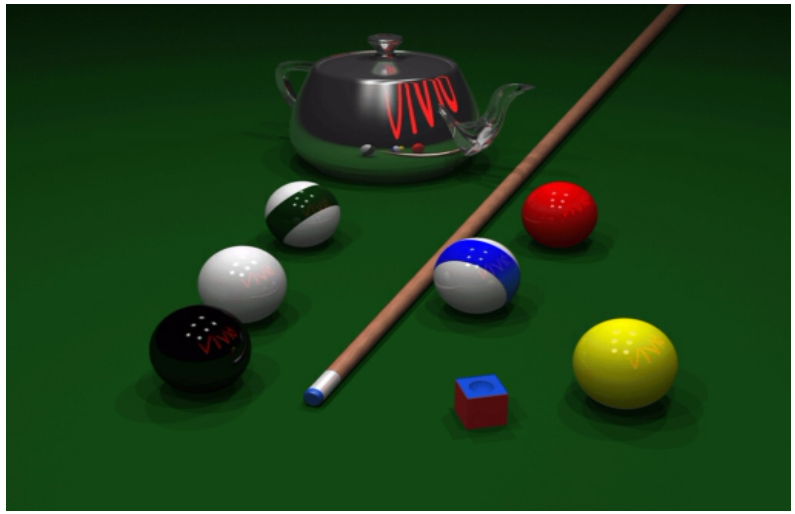


Image compressée $r = 0.5$

# Illusion d'adaptation chromatique

- Convertir l'image *campagne.jpg* dans l'espace YCbCr et en « niveaux de gris »
- Inverser les canaux de chrominance (Cb=255-Cb, Cr=255-Cr)
- Revenir dans l'espace RGB
- Dessiner un petit cercle noir au centre de l'image RGB modifiée et de l'image originale en « niveaux de gris » (`np.meshgrid`)
- Afficher l'image RGB transformée, faire une pause, puis afficher sur la même figure, l'image « niveaux de gris » (`plt.figure/plt.imshow/plt.pause(12)`)
- Lorsque l'image RGB transformée est affichée, fixer le cercle noir. Que voyez-vous lorsque l'image en « niveaux de gris » s'affiche ?

Image RGB

Image RGB
avec chrominance inversée

Image en « niveaux de gris »

## Effet Pencil Sketch

- Calculer une carte de contours C de l'image *home.jpg* (`skimage.feature.canny`)
- Convertir l'image en YCbCr, puis modifier le canal Y :

$$Y \leftarrow (255 - \alpha) \times (1 - C) + \beta$$

avec $\alpha, \beta \in [0, 255]^2$, des paramètres à régler manuellement

- Repasser en RGB pour obtenir un résultat d'esquisse :