# Image numérique

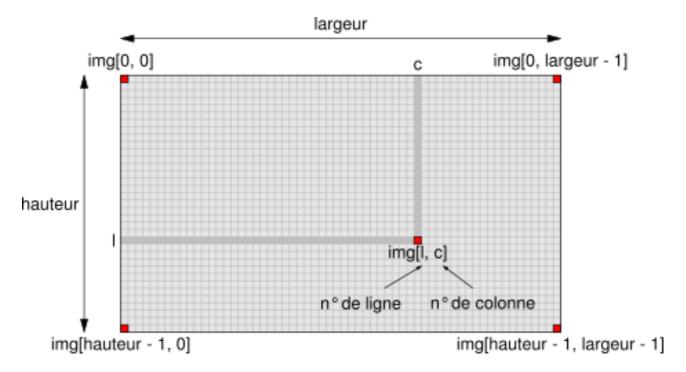
# 1. 1. Tableau de pixels

# •

# Les caractéristiques d'une image

## Quadrillage

Une image numérique se présente sous la forme d'un quadrillage - ou d'un tableau - dont chaque case est un pixel d'une couleur donnée. On peut donc repérer chaque pixel par sa ligne et sa colonne dans ce tableau (ou à l'aide de coordonnées en partant du coin en haut à gauche 1).



# Définition

La définition de l'image est le nombre total de pixels qui la composent. Celle-ci n'est pas forcément égale à la définition du capteur.

On l'obtient donc en multipliant sa <u>largeur</u> par sa <u>hauteur</u>. Par exemple, une image de 1920 pixels de <u>largeur</u> sur 1080 pixels de <u>hauteur</u> a une définition de 1920 x 1080 = 2073600 pixels soit à peu près 2 millions de pixels.

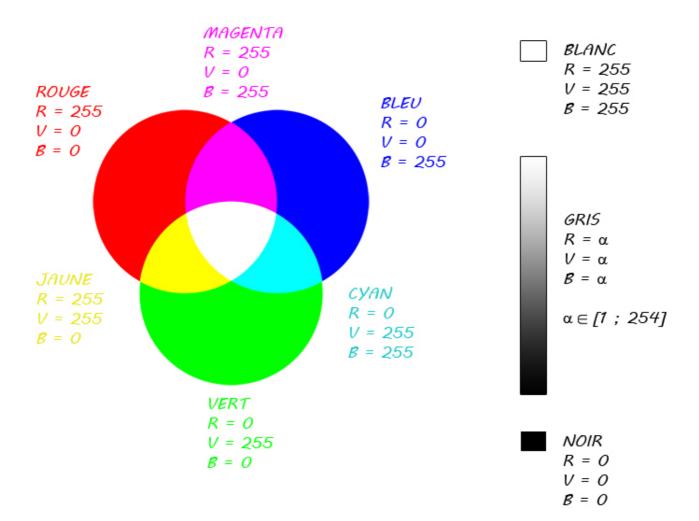
#### Résolution

La résolution de l'image, c'est-à-dire le nombre de pixels par unité de longueur, détermine sa qualité à l'impression ou sur un écran.

Par exemple, la résolution standard pour affichage sur le web est de 72 ppp (pixels par pouce) alors qu'une résolution de 300 ppp est recommandée pour l'impression.

# 1 Le codage des pixels (couleurs)

Chaque pixel correspond à un triplet de trois nombres entiers, soit les valeurs de rouge (Red), de vert (Green) et de bleu (Blue) afin de reconstituer la couleur. Chaque valeur est codée sur un octet, soit entre 0 et 255 (ou en pourcentages, ou en hexadécimal, voir ici). On parle de code RGB (RVB in french).



# À noter:

- une couleur dont les 3 composantes sont identiques correspond à un niveau de gris;
- selon les formats, une quatrième composante peut s'ajouter: le **canal alpha**. Cette valeur (sur un octet également) indique le niveau de transparence du pixel.

# **Site incontournable**

Un site pour visualiser les couleurs au formet RGB, et convertir en héxadécimal : http://www.proftnj.com/RGB3.htm

# 2. 2. Les modules

Pour manipuler les images, nous allons avoir besoin du module imageio. Ce module nécessite d'utiliser également le module numpy pour créer des tableaux d'entiers non signés sur 8 bits (un octet).



# imageio

• Ouvrir et charger une image existante (ada.png par exemple) dans une variable (img par exemple):

# **%** Script Python

```
img = imageio.imread("ada.png")
```

## 🛕 Accès à l'image

L'image doit être dans le dossier courant de travail, a fortiori le même que le fichier .py . Si ce n'est pas le cas, il faudra le modifier.

• La taille de l'image est accessible dans le triplet (hauteur, largeur, nombre de composantes) donné par:

# **%** Script Python

img.shape

• Lire/modifier un pixel: il s'agit tout simplement de travailler sur le tableau, par indices et par réaffectation.

# **%** Script Python

```
print(img[2][10])  # pour afficher le pixel ligne 2, colonne 10
img[2][10] = [0, 0, 0] # pour le mettre en noir
```

• Sauvegarder une image contenue dans une variable img:

# **%** Script Python

```
imageio.imsave("monimage.png", img)
```



## numpy

Le module numpy est un module de calcul scientifique orienté vers les matrices, qui sont des objets mathématiques bien pratiques... En gros ce sont des tableaux. On se servira uniquement de ce module pour créer des tableaux vides, au format que le module <u>imageio</u> exige pour pouvoir ensuite sauvegarder l'image (et donc la visualiser).

On utilise la fonction zeros du module numpy qui prend en paramètres un triplet (<u>hauteur</u>, <u>largeur</u>, nombre de composantes) et le type des valeurs, ici donc des entiers non signés sur 8 bits.

Par exemple pour une image de 100 pixels (de haut) sur 256 pixels (de large), avec 3 composantes (pas de canal alpha):

# **%** Script Python

```
img_vide = numpy.zeros([100,256,3], dtype=np.uint8)
```

# 3. 3. Exercices

# **Exercice 1**

#### Énoncé

- 1. Télécharger l'image ada.png ci-dessus (simple clic-gauche), puis la charger dans un programme avec le module imageio.
- 2. Trouver ses dimensions et son nombre de composantes.
- 3. Faire un crime de lèse-majesté et tracer une ligne horizontale rouge au nivau du front.

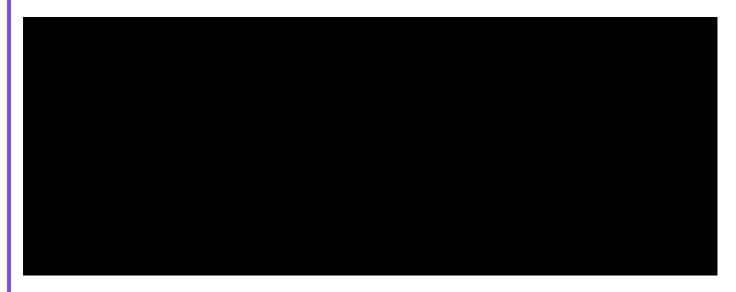
#### Solution

#### import imageio 1 2 # on charge l'image dans une variable img 3 4 img = imageio.imread('ada.png') 5 6 # on affiche les dimensions et le nombre de composatnes contenues dans le tuple 7 img.shape print('Hauteur:', img.shape[0], 'Largeur:', img.shape[1], 'Nombre de 8 composantes:', img.shape[2]) 9 10 # on parcourt la ligne 100 et on remplace tous les pixels par du rouge 11 12 for j in range(img.shape[1]): 13 img[100][j] = (255, 0, 0, 255)14 # on enregistre l'image modifiée imageio.imsave('ada\_modifie.png', img)

# **Exercice 2**

Énoncé

Cette image est-elle vraiment composée de pixels tous noirs?



### Solution

- on parcourt tous les pixels de l'image avec deux boucles for imbriquées: i sur la <u>hauteur</u> (les lignes) et j sur la <u>largeur</u> (les colonnes);
- on regarde si le pixel img[i][j] est noir, c'est-à-dire que ses 3 composantes sont égales à 0;
- si c'est le cas, on le remplace par un pixel blanc;
- sinon on ne fait rien: le pixel restera sur une teinte proche du noir.

```
import imageio
img = imageio.imread('message.png')

img = imageio.imread('message.png')

img = imageio.imread('message.png')

img = imageio.imread('message.png')

img = imageio.imread('message.png')
```

# **Exercice 3**

Énoncé

Incruster John Travolta devant le lycée



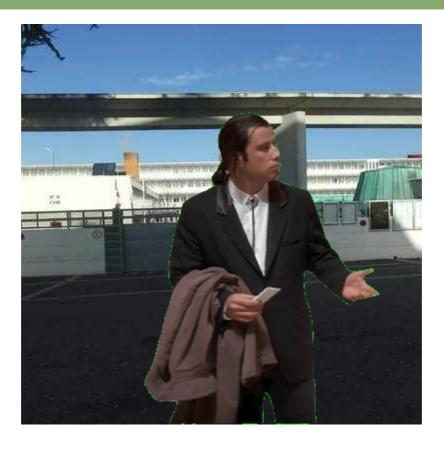


### Solution

- on charge les deux images dans deux variables;
- on parcourt l'image sur fond vert, et si le pixel est vert, on le remplace par le pixel (aux mêmes coordonnées) de l'autre image.
- on peut bien entendu faire le contraire...

```
import imageio
1
2
     img_john = imageio.imread('john.bmp')
     img_lycee = imageio.imread('lycmdv_crop.jpg')
3
4
5
     for i in range(img_john.shape[0]):
         for j in range(img_john.shape[1]):
6
             if img_john[i][j][0] == 0 and img_john[i][j][1] == 255 and img_john[i][j]
7
8
     [2] == 0:
                 img_john[i][j] = img_lycee[i][j]
9
10
     imageio.imsave('john_devant_lycee.bmp', img_john)
```

On obtient:



# 4. 4. Création d'effets

Dans cette dernière partie, on va recréer des effets que des logiciels de retouche d'image (GIMP, Photoshop, ...) proposent.

On travaillera (par exemple) sur l'image ci-dessous:



# Effets

Filtre rouge

Pour créer un filtre rouge il suffit de conserver la composante rouge et de remplacer les autres composantes par 0.

Si vous n'aimez pas le rouge, faites un filtre vert. Ou bleu.



```
Correction

Script Python

import imageio
img = imageio.imread('VanGogh_Arles.png')

for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        r = img[i][j][0]
        img[i][j] = (r, 0, 0)

imageio.imsave('img_filtre.png', img)
```

Négatif

Pour obtenir le négatif d'une image, il faut remplacer chaque composante RGB par son complémentaire à 255.

Par exemple, si une composante vaut 42, il faut la remplacer par 213 (= 255 - 42).



```
Correction
     import imageio
1
     img = imageio.imread('VanGogh_Arles.png')
 3
     for i in range(img.shape[0]):
 5
        for j in range(img.shape[1]):
 6
            r = 255 - img[i][j][0]
7
             g = 255 - img[i][j][1]
8
            b = 255 - img[i][j][2]
9
             img[i][j] = (r, g, b)
10
11
     imageio.imsave('img_negatif.png', img)
```

Niveaux de gris

Dans sa norme 709, la Commission Internationale de l'Éclairage propose de remplacer les 3 composantes d'un pixel (r, g, b) par la valeur suivante :



```
Correction
     import imageio
1
     img = imageio.imread('VanGogh_Arles.png')
 3
     for i in range(img.shape[0]):
 5
        for j in range(img.shape[1]):
 6
             r = img[i][j][0]
7
             g = img[i][j][1]
8
             b = img[i][j][2]
9
             m = int(0.2126*r + 0.7152*g + 0.0722*b)
10
             img[i][j] = (m, m, m)
11
     imageio.imsave('img_gris.png', img)
12
```

Flip

On retourne l'image horizontalement.



```
Correction
     import imageio
1
     img = imageio.imread('VanGogh_Arles.png')
 3
     # on doit commencer par faire une copie de l'image
 5
     img_miroir = img.copy()
 6
7
     for i in range(img.shape[0]):
8
         for j in range(img.shape[1]):
9
             img_miroir[i, img.shape[1]-j-1] = img[i, j]
10
     imageio.imsave('img_miroir.png', img_miroir)
11
```

**Photomaton** 

C'est une transformation réversible, puisqu'on envoie un pixel sur quatre dans chaque carré...



# Correction

L'idée est d'«envoyer» chaque pixel dans l'un des 4 carrés, en considérant la parité des indices de ligne i et de colonnes j :

- les pixels sur une ligne paire sur les carrés du haut;
- les pixels sur une ligne impaire sur les carrés du bas;
- les pixels sur une colonne paire sur les carrés de gauche;
- les pixels sur une colonne impaire sur les carrés de droite;

```
1
     import imageio
2
     img = imageio.imread('VanGogh_Arles.png')
3
    img_photomaton = img.copy()
4
5
    for i in range(img.shape[0]):
 6
7
        for j in range(img.shape[1]):
             img_photomaton[i//2 + 128*(i\%2)][j//2 + 128*(j\%2)] = img[i][j]
8
9
10
     \verb|imageio.imsave('img_photomaton.png', img_photomaton)|\\
```

#### Pop-art

Le principe est, pour chaque pixel, d'appuyer sur la composante majoritaire: on récupère la composante maximale et on l'augmente d'une certaine valeur (par exemple 50 sur l'image ci-dessous) sans dépasser 255 bien entendu.



✓ Correction

Avec une fonction...

```
1
     import imageio
 2
     img = imageio.imread('VanGogh_Arles.png')
 3
     def popart(pix: list, val: int) -> list:
 4
 5
 6
         renvoie la liste des composantes de pix, en ayant augmenté la valeur maximale
 7
         de la valeur val, sans dépasser 255 bien évidemment
 8
9
         m = max(pix)
10
         t = []
         for composante in pix:
11
12
             if composante == m:
13
                 t.append(min(255, composante+val))
14
             else:
15
                 t.append(composante)
16
         return t
17
18
     for i in range(img.shape[0]):
19
         for j in range(img.shape[1]):
20
             img[i][j] = popart(img[i][j], 50)
21
22
     imageio.imsave('img_popart.png', img)
```

**Pixellisation** 

Je vous laisse deviner...



# ✓ Correction

Le principe est de décider tout d'abord d'une taille de «carrés» qui vont composer l'image pixellisée. Bien entendu, cette taille doit être un diviseur commun de la <u>hauteur</u> et de la <u>largeur</u> de l'image. Ici comme l'image fait 256x256, on peut choisir n'importe quelle puissance de 2. Par exemple, prenons 8 pixels. Il y aura donc 256//8 = 32 carrés en hauteur et en <u>largeur</u>.

Ensuite on va définir la couleur «moyenne» qu'on va mettre dans chaque carré: on fait la moyenne des composantes dans le carré (moyenne des rouges, moyenne des verts et moyenne des bleus).

On affecte enfin cette couleur à chaque pixel du carré.

```
import imageio
1
 2
     img = imageio.imread('VanGogh_Arles.png')
 3
 4
     cote = 8
 5
     def couleur_moyenne(ligne, colonne):
 6
 7
        moy = 3 * [0]
 8
         for i in range(cote):
 9
             for j in range(cote):
10
                 for k in range(3):
                     moy[k] += img[cote*ligne + i,cote*colonne + j][k] / (cote**2)
11
12
         return moy
13
14
     for ligne in range(img.shape[0]//cote):
15
         for colonne in range(img.shape[1]//cote):
16
17
             for i in range(cote):
18
                 for j in range(cote):
                     img[cote*ligne + i,cote*colonne + j] = couleur_moyenne(ligne, colonne)
19
20
     imageio.imsave("img_pixellisee.png", img)
21
```

**Floutage** 

Je vous laisse deviner...





1. en fait cela dépend de l'outil (module) utilisé pour lire et écrire des images. ←