Le traitement d'images (première partie)

Utilisation du module matplotlib

Pour enregistrer vos images au format adapté, vous pouvez utiliser par exemple le logiciel GIMP.

Il est impératif de travailler sur des fichiers .png.

Quelques clefs pour maitriser ce module :

1. Charger les modules :

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
```

2. Comment ouvrir un fichier image ?
   Les images se trouvent dans le dossier "/Users/alexandremarino/Documents/Joffre/IPTMP*/SujetsTP/TP2Images/version2/" de mon ordinateur. Pour charger la photo d'A.Einstein :

```
link='/Users/alexandremarino/Documents/Joffre/IPTMP*/SujetsTP/TP2Images/
    version2/'
img=mpimg.imread(link+'einsteincouleur.png')
```

   Pour une gestion simple des fichiers, enregistrez votre fichier TP.py dans le même dossier que les images utilisées.
   Vous constaterez que le img est une liste de listes de triplets plus précisément il s'agit d'une np.narray :
   Il s'agit de la liste des lignes de l'image. L'indice $(0, 0)$ étant le code RGB du pixel en haut à gauche.

```
imm=list(img)+list(img)
imgplot = plt.imshow(imm)
plt.show()
```

   Attention, img+img n'est pas la concaténation pour les "array" de numpy.

3. Comment est chargé une photo ? sous quel format ?
   ⇒ une grille de pixels et une "couleur" par pixel.
   — Sa taille par "img.shape" au format (Nombre de colonnes, Nombre de lignes). Nous traiterons des exemples avec des images de taille 480 sur 640.

```
taille = img.shape[:2]
```

4. Comment afficher une image ?

```
imgplot = plt.imshow(img)
plt.show()
```

5. Comment fabriquer une nouvelle image ?

   — Créer une image grise de taille (p, q) :

```
def init(p,q):
    """Fabrication d?une photo noire de format p*q"""
    noire = [ [[0.,0.,0.] for j in range(q)] for i in range(p)  ]
    return noire

plt.imshow(init(10,10))
plt.show()
```

   ou

```
def init(p,q):
    """Fabrication d?une photo noire de format p*q"""
    noire = [ [0. for j in range(q)] for i in range(p)  ]
    return noire

plt.imshow(init(10,10),cmap="gray")
plt.show()
```

6. Introduction : Comment est codé une image avec matplotlib ? (le sujet du TP détaillera cette procédure)
   Une image est donc codée par une liste de listes (une matrice) de longueur Nbre de lignes × Nbre de colonnes.
   Pour affecter la valeur d'un pixel, nous considèrerons deux possibilités :
   — La première : l'image est en "niveaux de gris". Chaque pixel est codé pour un réel entre 0 et 1
   — La seconde : l'image est en "RGB". Chaque pixel est codé pour un triplet (r,g,b) trois réels entre 0 et 1.

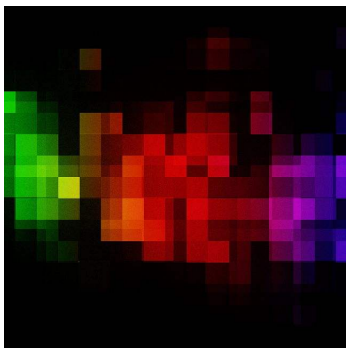
Comment sauver une image ?

N'hésitez pas à utiliser le code :

```
mpimg.imsave("nom",img)
```

Comment coder une image ?

La brique élémentaire d'une image est appelé le pixel.



Coder une image revient à l'identifier à l'aide d'une grille de pixels à l'aide d'une taille hauteur× largeur et d'une couleur ou d'un niveau de gris affecté à chaque pixel.
L'image précédente est de taille 640× 640. Ce qui donne 409600 pixels.

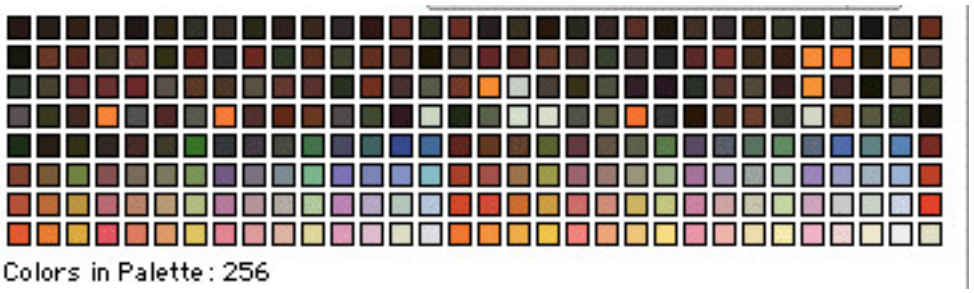
Quelle espace occupe une image ?

Comme toute information, une image est codée en binaire. Combien de bits sont nécessaires pour coder un pixel ?
— 1 bit ⇒ noir ou blanc.
— 2 bits ⇒ 4 couleurs/gris.
— 4 bits ⇒ 16 couleurs/gris.
— 8 bits =1 octet ⇒ 256 couleurs/gris.
— 24 bits=3 octets ⇒ plus de 16 millions de couleurs
Nous avons évidemment

Espace nécessaire pour l'image = Nbre de pixels × allocation par pixel

Quelques détails : allocation par pixel/ profondeur

1. **Niveaux de gris⇒ 8bits=1 octet**
   On associe à chaque pixel un réel entre 0 et 255 : de 0 (noir pur) à 255 (blanc pur).
2. **Palette de 256 couleurs (Gif, web , ...) ⇒ 8bit=1octet**
   On associe à chaque pixel un entier entre 0 et 255.



3. **RGB (16 millions de couleurs) ⇒ 24 bits=3 octets**
   La couleur est composée de 3 éléments : Rouge, Vert, Bleu. Chacun de ces éléments dispose de nuances allant de 0 à 255 : 256 couleurs/teintes. Ce qui donne au total plus de 16 millions de couleurs disponibles.

Quel espace occupe notre image ?
Elle est composée de 409600 pixels, nous devons disposer :
- Pour un stockage RGB : 409600 × 3 = 1228800 octets. Ce qui correspond à environ 1,2 Mo.
- Pour un stockage avec une palette 256 couleurs : 409600 octets. Ce qui correspond à environ 400 ko.

Combien de couleurs/nuances peut-on distinguer ?

Une estimation grossière des capacités à distinguer les couleurs donne environ 15 à 20 000 nuances identifiables. On parvient, dans les meilleures conditions d'examen, à un demi-million de couleurs. En codant plus de trente fois plus de couleurs que ce que l'oeil distingue avec le RGB, nous sommes sûr d'avoir suffisamment d'informations, pour que deux codes qui ne diffèrent que de 2 ou 3 sur les 256 valeurs possibles de chaque canal donnent des couleurs parfaitement indistinguables.

Pour comprendre le codage avec des réels entre 0 et 1 effectué par matplotlib : http ://matplotlib.org/users/colors.html

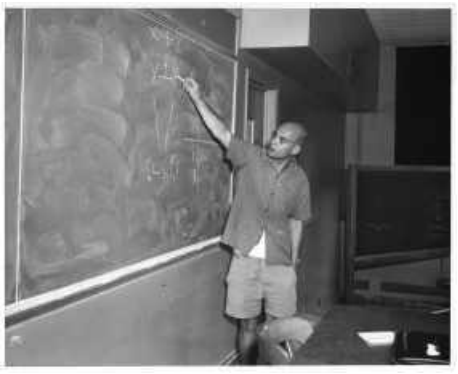
Chaque pixel en niveaux de gris est codé par matplotlib par un réel entre 0 et 1. Nous nous ramenons à un entier entre 0 et 255 en effectuant

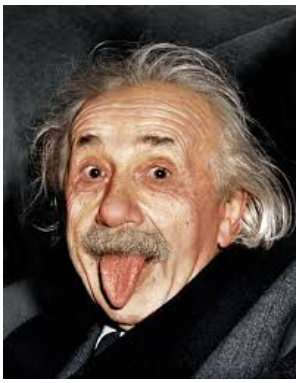
```
int(255*x)
```

(Il suffit de diviser par 255 pour revenir ensuite entre 0 et 1.)

Comment passer d'un codage à l'autre ? RGB, Niveaux de gris ?

Nous travaillerons dans le TP sur les photos de A. Grothendieck suivantes : (à retrouver sur le site de la classe)





Les images codées en RGB sont des listes de listes de listes [R,G,B]. Chaque couleur est un réel entre 0 et 1.

1. Tester les fonctions init précédentes.
2. Ecrire une fonction **nivgris** qui prend en argument une image en couleur et qui donne la même image en niveaux de gris. La valeur du pixel en niveaux de gris sera la moyenne des trois réels associés à R, G et B.
3. Ecrire une fonction **coul** qui prend en argument une image en niveaux de gris et qui donne la même image en RGB. La valeur du pixel [R,G,B] sera [N,N,N] où N est le niveau de gris du pixel.

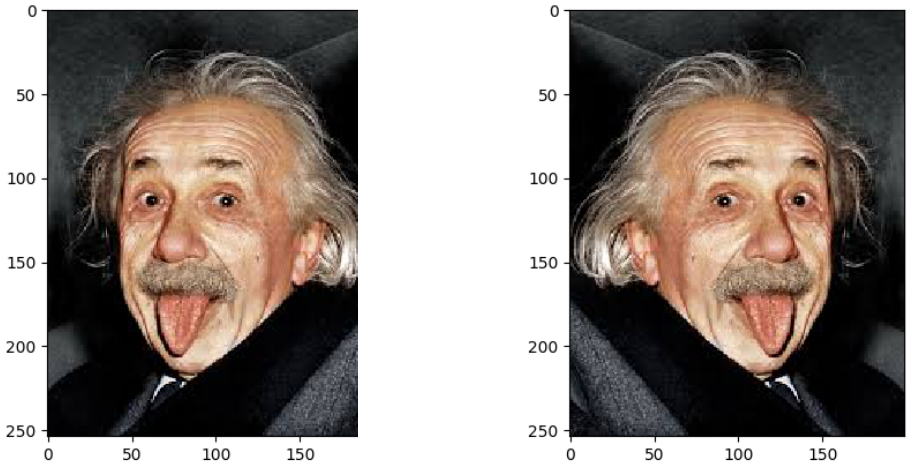
Premières transformations en niveaux de gris

Nous considérons dans la suite qu'une image de taille (H, L) (hauteur, Largeur) sera représentée par une liste de listes img. Le coefficient (i,j) de la matrice obtenu par img[i][j] contient l'information nécessaire pour représenter le pixel à l'aide du codage choisi. Par exemple, en RGB chaque coefficient de la matrice sera une liste [r,g,b].
Les indices de la matrice commencent à (0,0) qui correspond en général au pixel en haut à gauche de l'image.
Dans la suite nous ne distinguerons plus l'image et sa matrice représentative.

Nous utiliserons la fonction init précédemment définie.

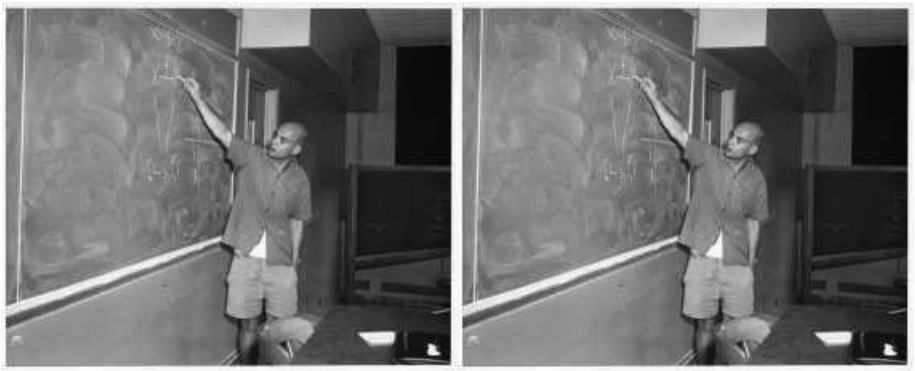
Dans la suite, suivant vos préférences vous travaillerez avec des images en niveaux de gris ou RGB .

1. Ecrire une fonction **Flip** qui prend en argument une image et qui la renverse (cf exemple ci-dessous) en réalisant une symétrie d'axe vertical passant par le "milieu" de l'image. (On ne modifiera pas la matrice, on travaillera sur une copie. Cette précaution sera à respecter dans toutes les questions.)



2. Ecrire une fonction **Inverse** qui réalise le négatif d'une image (cf exemple suivant).

3. Ecrire une fonction **Colle** qui prend deux images de même hauteur et les rassemble en une seule photo. (nous collons deux images identiques)



Egaliser les tons en niveaux de gris

---

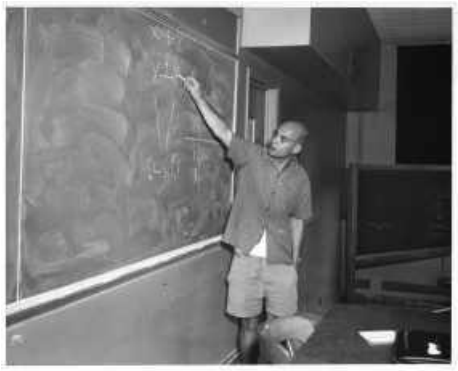
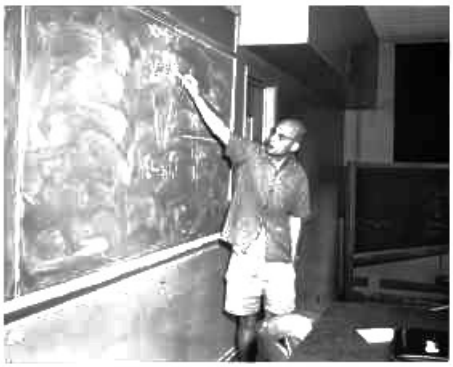
1. **Histogramme** : l'histogramme d'une image permet de compter le nombre de pixels d'un niveau de gris donné. Ecrire une fonction **Histo** qui prend en argument une image et qui renvoie la liste (de longueur 256) du nombre de pixels d'un gris donné.

   Nous considérons une image de hauteur H, de largeur L d'histogramme h. Nous notons **vmin** le ton de gris le plus sombre se trouvant dans l'image. Nous égalisons les tons en transformant chaque ton v en v' à l'aide de la transformation suivante :

   $$v' = 255 \times \frac{\left(\sum_{k=0}^{v} h[k]\right) - h[vmin]}{H \times L - h[vmin]}$$

   Nous remarquons que la valeur v' n'est en général pas entière. Pensez à convertir cette valeur en un entier.
2. Ecrire une fonction **vprime(v,vmin,h, H,L)** qui prend pour arguments v, vmin, l'histogramme h et la hauteur et largeur et qui donne v' donné par la formule précédente. (vous pouvez utiliser les fonctions **sum** et **min** de python)

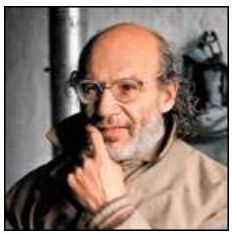
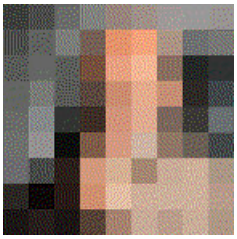
3. Ecrire une fonction **Egal** qui prend en argument une image et qui l'égalise à l'aide de la méthode précédente.

4. Que donne la fonction **Egal** sur une image complètement noire ?

Pixelisation d'une image en "niveaux de gris"

L'objectif est de réaliser la pixélisation suivante (ou sur la même photo en "niveaux de gris")

Comment pixéliser ?
Nous considérons une image. La photo précédente est de taille

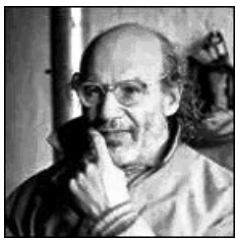
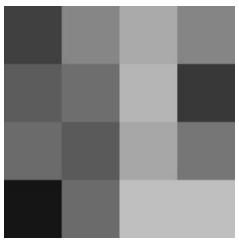
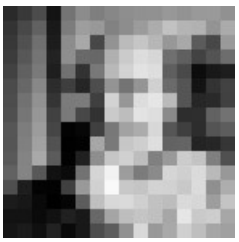
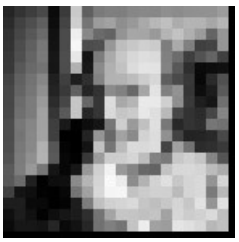
$$144 \times 144 = (o1 \times p) \times (o2 \times p)$$

Par exemple avec $p = 9$ et $o1 = o2 = 16$.

Sur cet exemple, pixéliser cette image revient à remplacer chaque carré de 16×16 par un carré de même taille dont la valeur est la valeur moyenne des 16×16 pixels.
De manière générale, pixéliser une image de taille $H \times L$ revient à remplacer chaque carré de taille $\frac{H}{p} \times \frac{L}{p}$ par un carré de même taille dont la valeur est la valeur moyenne des pixels.

1. Ecrire une fonction **pixel(mat,p)** où **mat** est la matrice de l'image et p le nombre de carrés qui renvoie la matrice de l'image pixélisé.

Nous obtenons les pixélisations suivantes (pour les valeurs $p = 4, 16, 20$)

Stéganographie ou l'art de la dissimulation

Comment dissimuler une image dans une autre ?

La photo de gauche dissimule celle de droite :


Chaque pixel en RGB est codé par matplotlib par une triplet de réels entre 0 et 1. En effectuant

```
int(255*x)
```

nous ramenons ce code en un triplet d'entiers entre 0 et 255. (Il suffira de diviser par 255 pour revenir ensuite entre 0 et 1.)
Chaque pixel en RGB est codé par 3 octets. Une couleur est donc codée par 8 bits ou un nombre entier entre 0 et 255. Par exemple,

$$69 = 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 1$$

est codé par

0	1	0	0	0	1	1	0

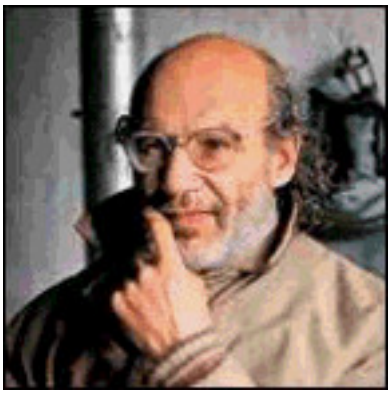
Question essentielle : est-ce que l'image est "visuellement proche" entre (69 et 63) ?

0	1	0	0	0	1	1	0

et

0	1	0	0	0	0	0	0

1. Dans la suite, nous pourrons utiliser les opérations sur l'écriture binaire des entiers :
   (nous avons déjà utilisé a^b dans le TP précédent)
   — & : et
   — | : ou
   — ^ : ou exclusif
   — » : décalage d'un bit à droite (correspond à une division par 2)
   — « : décalage d'un bit à gauche (correspond à une multiplication par 2)
   Lorsque a est une entier entre 0 et 255 que fait (a«4)»4 ?
2. Ecrire une fonction **zero(image,n)** qui prend une image et donne l'image dont les n derniers (correspondants aux puissances les plus petites) bits de chaque couleur sont ramenés à 0. Nous obtenons par exemple pour $n = 4$ et $n = 7$.



Nous proposons une procédure de dissimulation d'une image 2 dans une image 1 basée sur la méthode suivante : Le schéma suivant est la brique élémentaire de notre transformation :

image 1	a7	a6	a5	a4	a3	a2	a1	a0

et

image 2	b7	b6	b5	b4	b3	b2	b1	b0

$\Downarrow$

a7	a6	a5	a4	b7	b6	b5	b4

3. Ecrire une fonction **masquer(image1,image2)** qui prend 2 images et donne l'image 1 dans laquelle l'image 2 est dissimulée.
4. Ecrire une fonction **voir(image1)** qui prend l'image1 et donne l'image 2 dissimulée dans celle-ci.