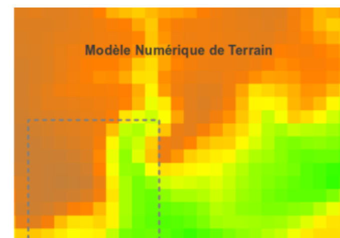
- Matrices pixels et images -

NumPy Une image, c'est un tableau de pixels https://machinelearnia.com

2D array

3D array



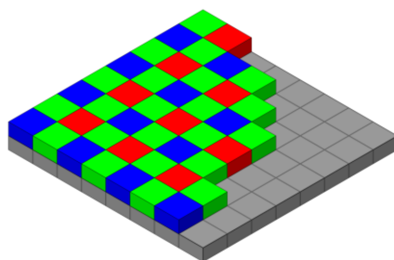
155	152	146	146	141	134	104	73	70	80
156	156	152	149	144	130	86	69	75	99
159	159	156	155	149	122	80	65	78	103
156	160	161	159	154	126	82	66	65	83
156	160	162	161	156	136	89	69	61	66
156	160	156	156	149	132	92	69	61	59
157	153	145	129	123	106	87	69	61	58
145	131	105	93	92	97	87	68	60	57
102	97	90	83	86	88	73	61	59	60

Matrice de valeurs

Palette de couleurs

155	152	146	146	141	134	104	73	70	80
156	156	152	149	144	130	86	69	75	99
159	159	156	155	149	122	80	65	78	103
156	160	161	159	154	126	82	66	65	83
156	160	162	161	156	136	89	69	61	66
156	160	156	156	149	132	92	69	61	59
157	153	145	129	123	106	87	69	61	58
145	131	105	93	92	97	87	68	60	57
102	97	90	83	86	88	73	61	59	60

Affichage de chaque couleur de la palette en fonction de la valeur du pixel de la matrice de valeurs

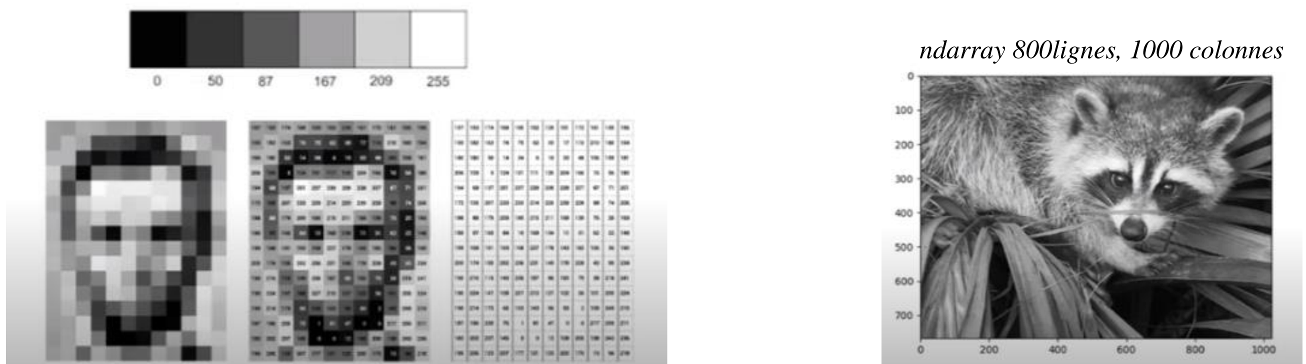


Matrices pixels et images

1. Rappels sur les tableaux

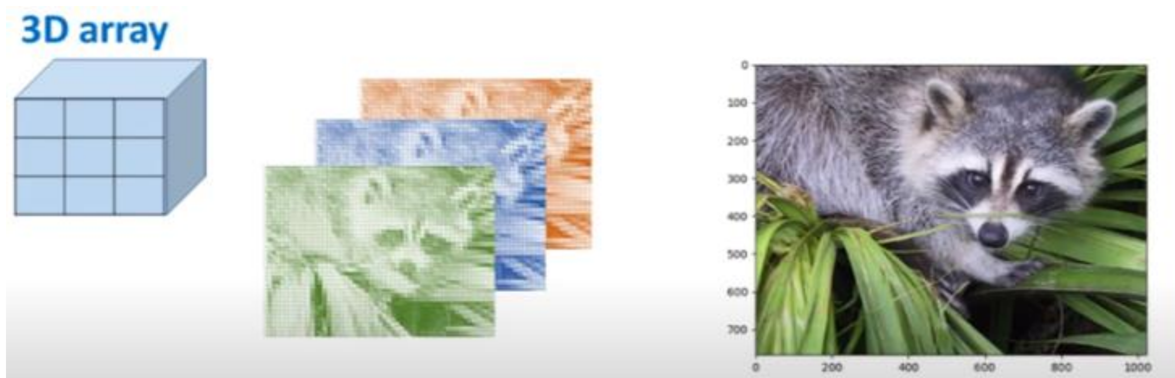
Une image est représentée par un tableau 2D ou 3D de valeurs appelés **pixels.** Si l'image est en niveau de gris, le tableau est en 2D, si elle est en couleur alors le tableau est en 3D. Les tableaux à 3 dimensions contiennent les couches Rouge, Vert, Bleu.

Voici un exemple de tableau Numpy 2D permettant de stocker les pixels d'une image en niveau de gris :



ndarray 800lignes, 1000 colonnes

Un tableau Numpy 3D pour la même image mais en couleurs (3 niveaux : vert, bleu, orange) donc 3 dimensions.



Pour info, l'assemblage des 3 couches donnera la couleur de chaque pixel par composition.

Matrices pixels et images

2. Première approche du traitement d'images avec Numpy et Scipy

L'objectif de cet exercice d'introduction est de récupérer la photo ci-dessous, zoomer légèrement dessus et appliquer un filtre avec la technique Numpy de boolean indexing (cf cours TP précédent). Cette photo est un tableau numpy, chargé depuis scipy (elle est de base dans les images de scipy).

ps : le module Scipy sera étudié un peu plus tard dans le semestre, c'est le module où se concentrent tous les outils de calculs scientifiques.

Nous allons découvrir dans cette première approche, quelques outils intéressants de traitement d'images.
Voici le code que vous pouvez recopier :

```
from scipy import misc
import matplotlib.pyplot as plt
face=misc.face()
plt.imshow(face)
plt.show()
```

Tester quelques petites choses suivantes …

Cette photo est un objet nommé face et c'est bien un tableau numpy :

```
print(type(face))
```

```
<class 'numpy.ndarray'>
```

Quelle est la dimension du tableau ?

```
print(face.shape)
```

```
(768, 1024, 3)
```

Wahou ! tableau 768 lignes, 1024 colonnes et 3 niveaux = couleurs…se sont nos pixels ! tableau 3 dimensions

Maintenant on va simplifier l'image pour l'avoir en niveau de gris.
Pour cela on va indiquer dans la 3ème et 4ème lignes que l'on veut un affichage en gris.
Modifier votre code comme suit et exécuter :

```
from scipy import misc
import matplotlib.pyplot as plt
face=misc.face(gray=True)
plt.imshow(face,cmap=plt.cm.gray)
plt.show()
```

Vérifier que cette photo est bien un tableau numpy maintenant à 2 dimensions :

```
print(face.shape)
```

```
(768, 1024)
```

→ *Oui 2 dimensions*

Matrices pixels et images

Ensuite, nous allons **zoomer de ¼ vers le milieu de cette photo** via une technique de slicing…

La première chose à faire est de créer 2 variables : hauteur et largeur qui sont les dimensions de notre image

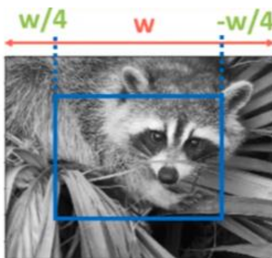
h = shape[0]
w = shape[1]

On retrouve avec shape les dimensions qu'on connaissait déjà…cool !

```
h=face.shape[0]
w=face.shape[1]
print(h,w)
```

```
768 1024
```

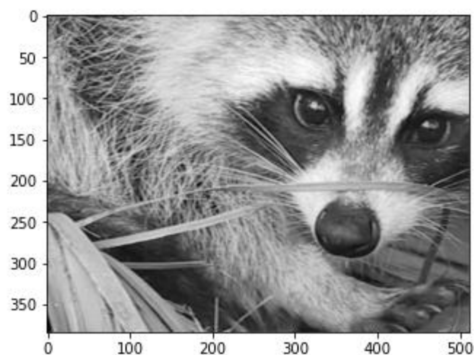
Ensuite, il faut sur l'axe vertical comme horizontal, sélectionner tous les points compris entre (h ou w)/4 et –(h ou w)/4 via slicing à partir su dernier élément (et en division entière c'est mieux pour un index !) :



image[h/4 : - h/4, w/4 : - w/4]

Voici le code :

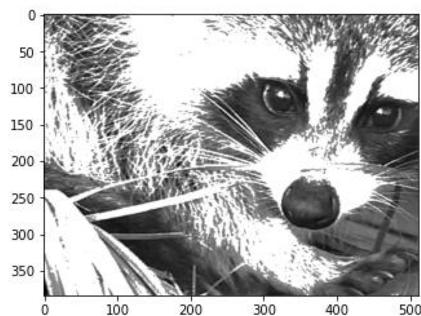
```
zoom_face=face[h//4:-h//4,w//4:-w//4]

plt.imshow(zoom_face,cmap=plt.cm.gray)
plt.show()

print(zoom_face.shape)
```




```
(384, 512)
```
⟶ *Le tableau numpy a bcq moins de valeurs ! normal*

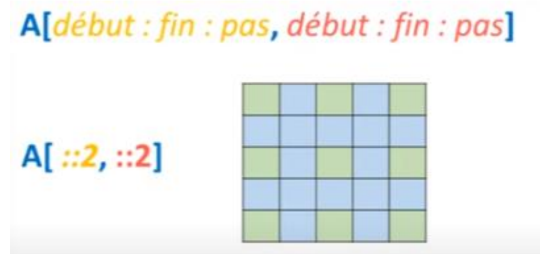
On peut aussi s'amuser pour finir à accentuer les pixels proches du blanc en les saturant à une valeur de 255 (c'est-à-dire blanc).

```
zoom_face[zoom_face>150]=255
plt.imshow(zoom_face,cmap=plt.cm.gray)
plt.show()
```

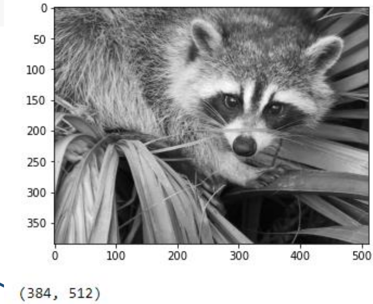


Enfin, nous allons apprendre **à réduire de moitié le poids de cette image** sur notre DD sans pour autant dégrader trop sa qualité d'affichage. C'est un peu ce que font les logiciels de compression, sauf que nous ici on fait le code…

L'idée est de prendre notre image et d'effectuer un slicing avec un pas de 2 (par exemple) sur les 2 dimensions de notre tableau d'origine (face) …on prend 1 pixel sur 2 !!




```
from scipy import misc
import matplotlib.pyplot as plt
face=misc.face(gray=True)

face=face[::2,::2]

plt.imshow(face,cmap=plt.cm.gray)
plt.show()

print(face.shape)
```



(384, 512)

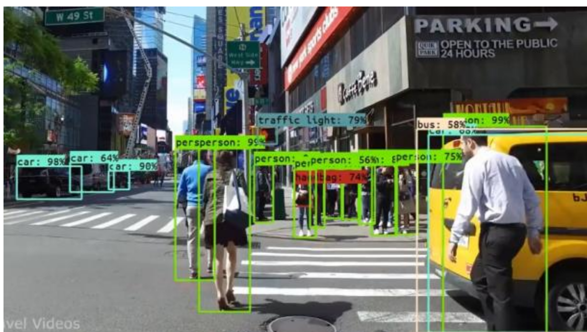
L'image est bien réduite en taille !!

3. Une image est une matrice 2D, 3D ou 4D

Les données collectées par des caméras ou appareils photos font souvent l'objet de **traitements numériques**, afin notamment d'aider à la **reconnaissance ou au suivi de formes sur les images**. **L'intelligence artificielle** se base beaucoup sur cette reconnaissance pour classer les images.

Voici quelques exemples industriels concrets :

- suivi de motifs (pointage vidéo automatisé),
- reconnaissance de formes (comptage sur une ligne de production, reconnaissance faciale, …)
- à la retouche d'image automatique (reconnaissance de contours et découpage, filtres type *Instagram*), …



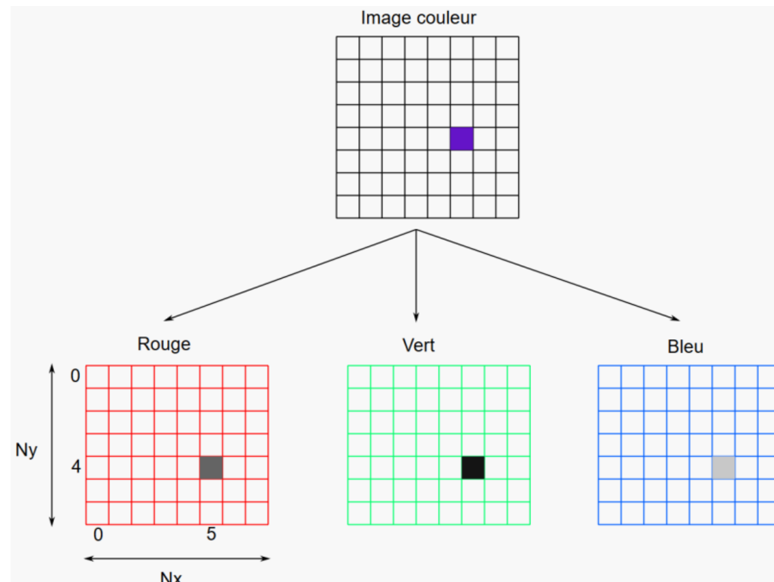
En **sciences industrielles ou en physique, le traitement d'image** doit souvent se faire en temps réel, ou très légèrement différé.

Matrices pixels et images

Exemple: traitement des mesures d'une **caméra embarquée**. Afin d'alléger les algorithmes, les concepteurs préfèreront généralement **travailler en niveaux de gris** *(cf exercice d'introduction),* même si la caméra est couleur.

Prenons une caméra couleur RVB (Rouge Vert Bleu) en 8 bits. Nous avons vu dans l'introduction que :

- en **couleur**, une image correspond à **3 matrices** : une matrice des niveaux de **rouge**, une matrice des niveaux de **vert** et une matrice des niveaux de **bleu**.



Chacune des 3 matrices R, V, B, à la même dimension que la matrice de l'image en couleur, par exemple 768 x1024 pixels (donc 768 lignes, 1024 colonnes), et chaque point de la matrice contient une valeur entre 0 et 255 (s'il s'agit d'une image 8 bits). *Ex : violet sur un pixel de l'image en couleur = 128 rouge, 0 vert, 128 bleu.*

Remarque : il est classique de voir qu'une image en RVB soit traduite comme quatre matrices, et non trois. La 4ème est alors la matrice des niveaux de transparence de l'image. 0 → pixel transparent, quelle que soit sa couleur, 255 → pixel opaque (on ne voit pas ce qui est derrière si l'image est superposée à une autre).

4. Fichier image avec matplotlib.mpimg

Nous avons vu dans l'exercice d'introduction que Numpy permet de manipuler les images comme des tableaux (ndarray) dont les cases représentent des couleurs.

Attention, avec cette bibliothèque les images sont vues comme des matrices :

- *le premier indice est celui des colonnes, il représente donc l'axe des y,*
- *le second indice est celui des lignes, il représente donc l'axe des x.*

tab[iLigne,jColonne] = image[iY,iX]

Voici les 5 méthodes les plus utiles :

- mpimg.imread(nf) : ouvre le fichier image *nf* et le convertit en `ndarray`
- mpimg.imsave(nf,tab) : enregistre le tableau *tab* (de type ndarray) en tant qu'image dans le fichier de nom *nf* (rq nf est un string : 'image.png').
- plt.imshow(tab) : affichage le ndarray en tant qu'image
- plt.axis('off') : supprime l'affiche des axes (abscisses et ordonnées) d'une figure matplotlib
- plt.set_cmap('gray') : utilise la palette des nuances de gris pour les images à un seul canal.

Matrices pixels et images

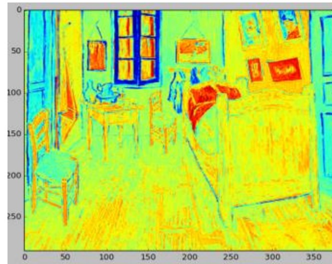
Exercice 1 : récupération d'une image

Récupérez sur google l'image de la chambre de Van Gogh et enregistrez là en format png. Puis tester le script suivant pour charger cette image depuis votre disque dur. Pensez à enregistrer votre script dans le même répertoire que l'image.

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

fileName='laChambre.png'          # nom du fichier
im1=mpimg.imread(fileName)        # imread : convertit le fichier image en ndarray
plt.imshow(im1)                   # affiche d'un ndarray en tant qu'image
plt.axis('off')                   # retirer les axes et leur graduations
plt.show()
print(im1.shape)
```

Nous allons maintenant extraire un canal de couleur (0 : rouge).

```
im2=im1[:,:,0]
plt.imshow(im2)
plt.show()
print(im2.shape)
```



On obtient l'image avec **l'extraction de la couche rouge.**

Exercice 2 : calcul de la **luminance** d'une image

C'est une image résultat qui contiendra pour chaque pixel la moyenne des valeurs max et min de chaque couche. Cela indique la puissance lumineuse du pixel.

Le calcul de la luminance se fait avec la relation : $L = (C_{max} + C_{min})/2$ où $C_{max} = \max(r,g,b)$ et $C_{min} = \min(r,g,b)$

Tester le code suivant qui permet d'afficher l'image 'im1' de l'exercice précédent en luminance :

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


fileName='laChambre.png'
im1=mpimg.imread(fileName)
plt.imshow(im1)
plt.axis('off')
plt.show()
print(im1.shape)

def getLuminance(imageIn):
    n,m,k=imageIn.shape
    imageOut=np.zeros([n,m])

    for i in range (n):
        for j in range (m):
            r= imageIn[i,j,0]    # canal R du pixel (i,j)
            g= imageIn[i,j,1]    # canal V du pixel (i,j)
            b= imageIn[i,j,2]    # canal B du pixel (i,j)
            cmin=np.min([r,g,b])
            cmax=np.max([r,g,b])
            imageOut[i,j]=(cmin+cmax)/2
    return imageOut

print(im1.shape)
im3=getLuminance(im1)
plt.imshow(im3)
plt.show()
```

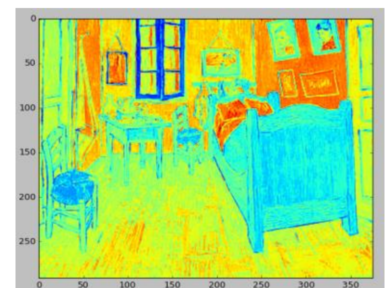


Image en luminance

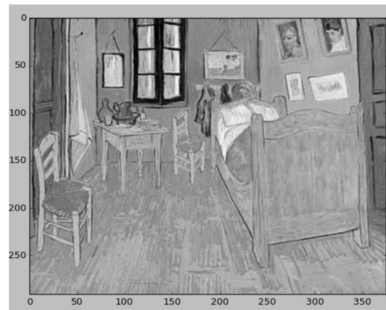
Matrices pixels et images

Exercice 3 : modification d'une image

La fonction *drawDisk(img,P0,ray,coul)* dessine un disque de centre P0(y,x) et de rayon *ray* et de couleur *coul* sur l'image *img.*

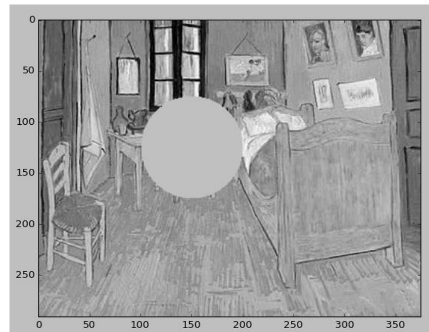
On reprend l'image im1 avec l'extraction canal rouge puis on passe l'image en nuance de gris :

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


fileName='laChambre.png'
im1=mpimg.imread(fileName)
plt.imshow(im1)
plt.show()

im2=im1[:,:,0]
plt.imshow(im2)
plt.set_cmap('gray')
plt.show()
```



Puis tester ce code permettant de dessiner un rond gris clair à un endroit de l'image:

```
def drawDisk(img,P0=[100,200],ray=50,coul=0.1):
    n,m=img.shape
    X0,Y0=P0
    for i in range(n):
        for j in range(m):
            dist2=(i-X0)**2+(j-Y0)**2
            if dist2<ray**2:
                img[i,j]=coul

drawDisk(im2,P0=[125,150],coul=0.75)
plt.imshow(im2)
plt.show()
```



Matrices pixels et images

5. TD de synthèse

Vous allez réaliser ce TD qui vous refera travailler sur les concepts vus précédemment en traitement d'images mais avec une approche différente. Votre plus grande capacité, sera, aux concours, de vous adapter aux sujets.

Problématique :
Afin de passer une image RVB en niveaux de gris, nous allons décomposer l'image initiale en 4 calques : un calque de niveaux de rouge, un de vert, un de bleu, et un de niveau d'opacité (que nous fixerons par la suite = 1).

Notre algorithme pour créer une image en niveaux de gris sera le suivant.

1. Détecter le nombre de lignes et de colonnes de l'image initiale (M lignes, N colonnes, correspondant au nombre de pixels, donc 512 et 512 dans l'exemple ci-dessus).
2. Créer trois matrices M x N, $(R_{i,j})_{\substack{1\le i\le M\\1\le j\le N}}$ correspondant aux niveaux de rouge, $(V_{i,j})_{\substack{1\le i\le M\\1\le j\le N}}$ pour le vert et $(B_{i,j})_{\substack{1\le i\le M\\1\le j\le N}}$ pour le bleu.
3. Créer une matrice $(G_{i,j})_{\substack{1\le i\le M\\1\le j\le N}}$ vierge de dimension M x N, ne contenant initialement que des 1 (couleur blanche), cette matrice correspondra aux niveaux de gris.
4. Pour chaque pixel [i, j], le niveau de gris sera calculé comme la moyenne des trois autres niveaux. Donc

$$G_{i,j} = \frac{R_{i,j} + V_{i,j} + B_{i,j}}{3}$$

Matrices pixels et images

Vous allez travailler sur une image en couleur de petite taille (32x32 pixels). Récupérez là pour la mettre dans votre répertoire de travail : *test.png*

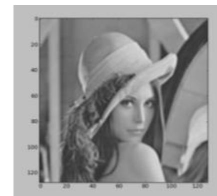
1°) À l'aide des fonctions *imread* (de la bibliothèque *matplotlib.image*) et *imshow* (de la bibliothèque *matplotlib.pyplot*), afficher cette image sur Python.

Nous allons maintenant travailler sur la photo de lena (512x512 pixels) présentée avant. Récupérez là pour la mettre dans votre répertoire de travail : *lena_128.png*

2°) Écrire alors une fonction *niv_gris(img)* prenant en argument la matrice d'une image (issue de *imread*), et renvoyant une matrice correspondant à l'image en niveaux de gris, quelle que soit la dimension initiale de l'image.

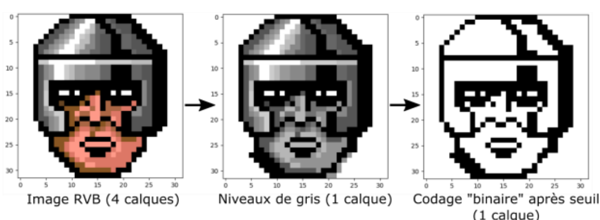
Afin de tester cette fonction, utilisez la commande suivante qui vous affichera l'image en gris correspondant à votre matrice:

```
image_test = mpimg.imread('lena_128.png')
plt.imshow(niv_gris(image_test), cmap=plt.get_cmap('gray'))
```



Nous allons repartir désormais de la fonction *niv_gris(img)* obtenue en Q2 et l'appliquer à l'image ci-dessous ('test.png'). Nous souhaitons désormais appliquer un seuil à l'image, afin de coder cette image en binaire : noir, ou blanc. Pour cela, il s'agit de tester la valeur de chaque niveau de gris de pixel, et appliquer la règle suivante après avoir défini un seuil compris entre 0 et 1 :

- *si sa valeur est ≤ seuil, alors le pixel est fixé à 0 (noir)*
- *sinon, le pixel est fixé à 1 (blanc).*



3°) Écrire une fonction *niv_gris(img, val_seuil),* prenant pour arguments l'image RVB ('test.png') et une valeur seuil comprise entre 0 et 1.
Cette fonction calculera le niveau de gris de l'image (on utilisera la fonction définie en Q2) et appliquera la règle ci-dessus à chaque pixel de l'image.

Vous testerez votre fonction avec l'image ci-dessus (test.png) comme suit :

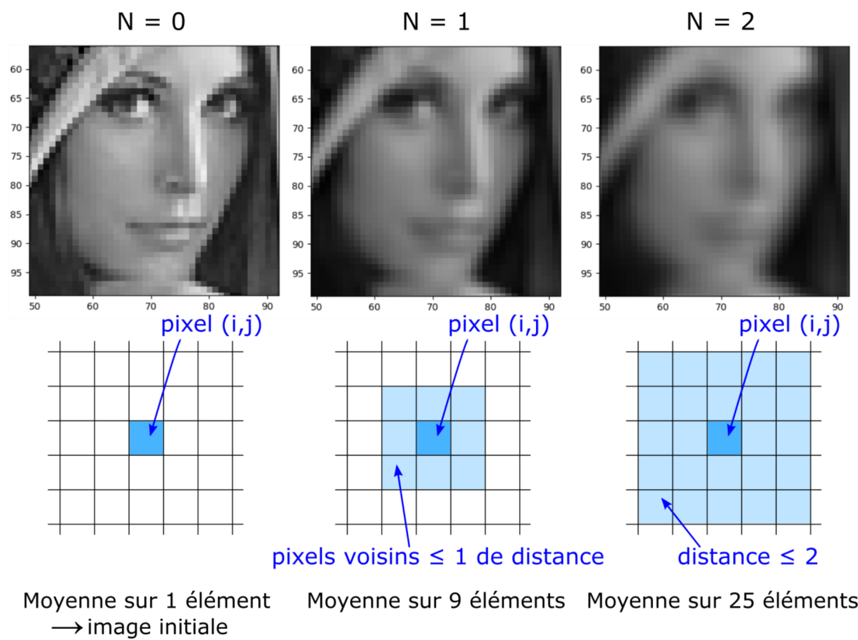
```
image_test = mpimg.imread('test.png')

plt.figure()
plt.imshow(seuil(image_test,0.05),cmap=plt.get_cmap('gray'))
plt.show()
```

Matrices pixels et images

Nous allons finir ce TD par un travail sur **le flou.**

La plupart des **opérations de filtrage** utilisées en traitement d'image numérique sont réalisées à l'aide **d'opérateurs matriciels appliqués à chaque pixel**. Nous allons ici voir **l'exemple simple du « flou », opéré par une moyenne glissante sur l'image**.

Pour flouter une image en niveaux de gris, nous allons **pour chaque pixel (i, j)** créer une fenêtre centrée sur ce pixel, et calculer la moyenne des niveaux de gris sur cette fenêtre. La valeur de la moyenne sera alors affectée au pixel central. Nous noterons N la taille de la demi-fenêtre, donc que la moyenne sera réalisée en incluant tous les pixels voisins du pixel (i, j) situés à une distance $\leq N$ de ce pixel central, c'est-à-dire que la fenêtre sera carrée de côté $2N + 1$ pixels, centrée sur (i, j).

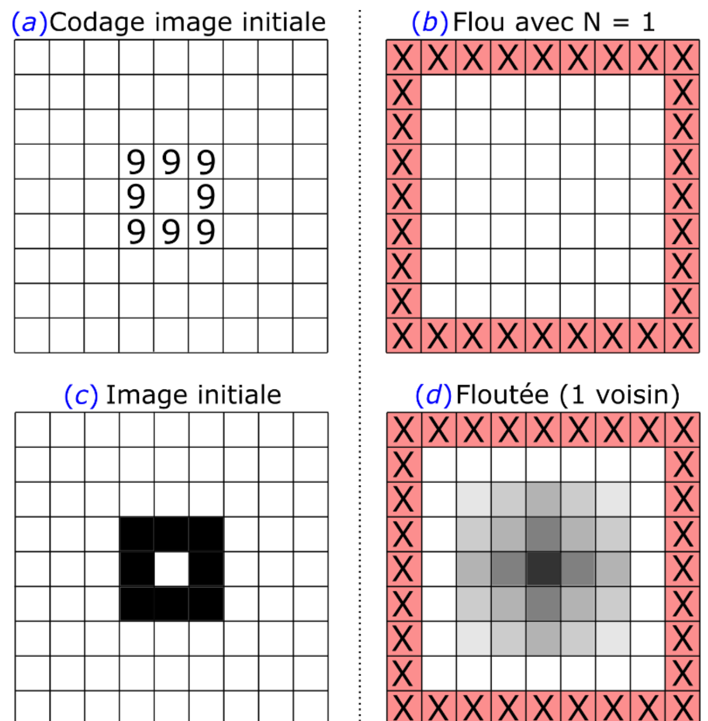


Remarque : la moyenne est égale à la somme des niveaux de gris de tous les pixels de la fenêtre, divisée par le nombre de pixels contenus dans la fenêtre, c'est-à-dire $(2N + 1)^2$.

Soit, ci-contre, une image où nous supposerons que les niveaux de gris sont codés sur 10 niveaux (de 0 = blanc à 9 = noir).

La table *(a)* correspond à l'image initiale, représentée en *(c)* (remarque : les 0 correspondant au blanc sont ici représentés par des cases vierges afin de ne pas surchargerle dessin).

La table *(b)* est à remplir en appliquant une fenêtre glissante avec $N = 1$ à la table *(a)*. Le résultat correspondant en niveaux de gris est représenté en *(d)*.



Matrices pixels et images

4°) Calculer les termes de la table *(b)*. Pourquoi, sur cette table comme sur la figure *(d)*, les bords sont-ils barrés ?

Afin de balayer totalement l'image, nous allons donc, pour chaque pixel (i, j), calculer la moyenne des pixel voisins (verticalement et horizontalement) distants de *N* ou moins.

Pour balayer tous les pixels centraux (i, j), il faudrait 2 boucles *for*. Mais pour un couple (i, j) donné, il faut balayer la fenêtre glissante, en vertical et horizontal. Nous partirons donc sur 4 boucles for imbriquées.

Soit une image initiale, en niveaux de gris, de taille $X_{\text{pix}} \times Y_{\text{pix}}$ sur laquelle on souhaite appliquer un flou de taille *N*. Sur feuille, décrire les bornes des quatre boucles *for*.

Pour *i* allant de ____________ à ____________

Pour *j* allant de ____________ à ____________

Pour *k1* allant de ____________ à ____________

Pour *k2* allant de ____________ à ____________

5°) Écrire alors une fonction *flou(img, demi_fen)* prenant en argument une image en couleur et un entier *N* correspondant à la taille de la demi-fenêtre, et renvoyant une matrice des niveaux de gris floutée par cette fenêtre glissante.

Tester cette fonction sur l'image utilisée en partie **B**, pour plusieurs valeurs de *N* différentes.

```
image_test = mpimg.imread('lena_128.png')

# montre l'image floutée
plt.figure()
plt.imshow(flou(image_test,2),cmap=plt.get_cmap('gray'))
```