Tableaux à plusieurs dimensions Images

1 Les tableaux comme listes de listes

1.1 Tableaux de dimension 2

On peut construire un tableau de dimension 2 comme une liste de **lignes**. Par exemple, pour construire

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

on utilisera l'instruction suivante :

```
A=[ [1,2,3] , [4,5,6] ]
```

1.1.1 Manipulation

1. Que renvoie l'expression : **len**(A) ?

2. Que renvoie l'expression : $A\,[1]$?

3. Comment obtenir le nombre de colonnes ?

4. Comment obtenir l'élément situé en troisième colonne et deuxième ligne ?

- Le premier index renvoie donc à l'index de la ligne.
- Le deuxième index renvoie donc à l'index de la colonne.

1.2 Définition d'un tableau sous forme de listes

Pour construire un tableau, on utilisera soit des boucles imbriquées, soit un definition par compréhensions imbriquées.

```
tab=[]
for i in range(hauteur):
    ligne=[]
    for j in range(largeur):
        ligne.append(0)
    tab.append(ligne)
```

```
tab=[ [0 for i in range(largeur)] for j in range(hauteur)]
```

1.2.1 Application 1

Ecrire un programme qui crée un tableau de 0 de 5 lignes et 5 colonnes, et où les termes diagonaux vallent 1.

1.3 Traitement d'un tableau

Le traitement d'un tableau nécessite souvent l'utilisation de boucle imbriquées.

1.3.1 Application 2

Proposer un programme qui recherche et affiche la plus grande valeur du tableau A. Il affiche aussi sa position. Combien d'itérations de boucles sont nécessaires ?

1.4 Extensions et limites

Pour construire un tableau de dimension 3, il faut utiliser une liste de listes de lignes. Par extension, il est possible de construire des tableaux de n'importe quelle dimension.

Ces tableaux héritent d'une propriété importante des listes : ils peuvent être hétérogènes.

Une limite importante dans l'usage de ce type de tableau est l'accès aux éléments de celui-ci. Par exemple, il est compliqué d'acceder directement à une colonne d'un tableau.

1.4.1 Application 3

Ecrire un programme qui extrait la quatrième colonne d'un tableau `T` de cinq colonnes et qui affiche cette colonne.

2 Le type *array* pour le calcul scientifique

Le module **Numpy** est dédié au calcul scientifique. Il propose de nouveaux types de variables et des fonctions optimisées pour le calcul scientifique. Il inclut un type de tableau dédié noté **array**, et les fonctions associées.

Les éléments contenus dans un array seront de même type (**homogénéité** du tableau). Les types proposés sont plus nombreux que sous python. On y trouve par exemple le type *uint8* : unsigned integer 8 bits (comprendre « codage en 8 bits non signés », soit de 0 à 255).

2.1 Construction d'un *array*

L'usage du type **array** nécessite l'import du module **Numpy**. Ceci fait, la construction explicite d'un tableau peut se faire par conversion d'une liste de listes en **array** :

```
import numpy as np
A=[ [1,2,3] , [4,5,6] ]
A=np.array(A)
```

Rarement utilisée, cette méthode est remplacée par des fonctions de construction de tableaux. Quatre fonctions permettent de générer automatiquement des *arrays* de propriétés particulières :

- **arange** : équivalent à *range*, mais génère un array :
  `arange(b_inf,b_sup,pas)`, où la borne inférieure est incluse, et la borne surpérieure est exclue.

```
>>> np.arange(0,2.5,.5)
array([0.,0.5,1.,1.5,2.])
```

- **linspace**(i, f, n) : construit un tableau de dimension 1 de n nombres décimaux uniformément répartis sur l'intervalle $[i; f]$:

```
>>> np.linspace(0,10,5)
array([ 0. ,2.5,5.,7.5,10.])
```

- **zeros**$([l, c])$: construit un tableau de l lignes et c colonnes rempli de 0

```
>>> np.zeros([2,3])
array([[0.,0.,0.],
       [0.,0.,0.]])
```

- **identity**(n) : construit un tableau correspondant à la matrice identité de dimension n

```
>>> np.identity(3)
array([[ 1., 0., 0.],
       [ 0., 1., 0.],
       [ 0., 0., 1.]])
```

2.2 Opération sur un *array*

Comme pour une liste, une variable de type *array* correspond à une adresse en mémoire. Lors de la copie d'une liste, il est nécessaire de prendre les dispositions nécessaires (voir chapitre **Liste**).

Un tableau de type *array* n'est pas une liste : les opérations "+" et "*" réalisent les opérations mathématiques d'addition et de multiplication :

```
>>> A+A
array([[ 2, 4, 6],
       [ 8, 10, 12]])
>>> A*A
array([[ 1, 4, 9],
       [16, 25, 36]])
```

2.2.1 Application 4

Ecrire le programme qui trace la courbe représentative de la fonction $\sin(x)$, pour $x \in [0; 2\pi]]$ sur 21 points. Vous n'avez droit qu'à trois lignes.

2.3 Extraction ou *Slicing*

On peut accéder de différents façon aux éléments de l'*array* A :

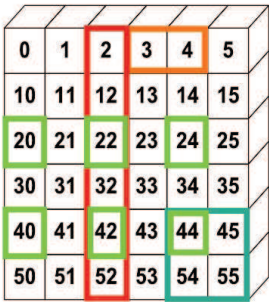
- accès à l'intégralité du tableau avec A
- accès à un élément du tableau avec $A[3, 2]$
- accès à une ligne du tableau avec $A[3, :]$
- accès à une colonne du tableau avec $A[:, 2]$
- accès à un sous tableau avec $A[2 : 4, 1 : 3]$

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



2.3.1 Application 5

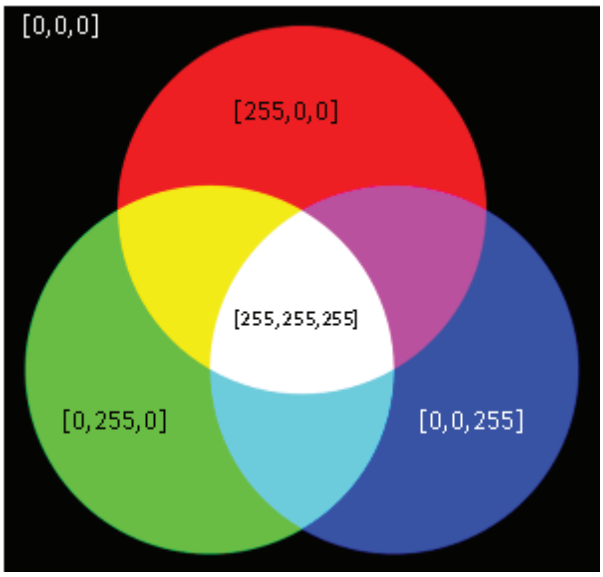
Soit A et B deux tableaux en mémoire. Afficher le produit matriciel $A \cdot B$ (on vérifiera par des assertions que les dimensions des tableaux sont adaptées).

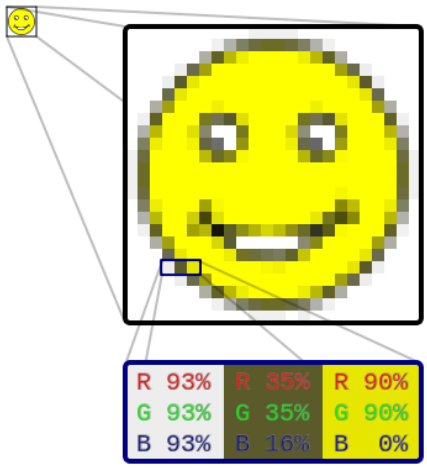
3 Images

Une image est formée de *pixel* (« picture element ») : un pixel est un carré de la plus petite dimension affichable par l'écran (ou projecteur) considéré, au quel on affecte une couleur.
A chaque pixel est associé une information :

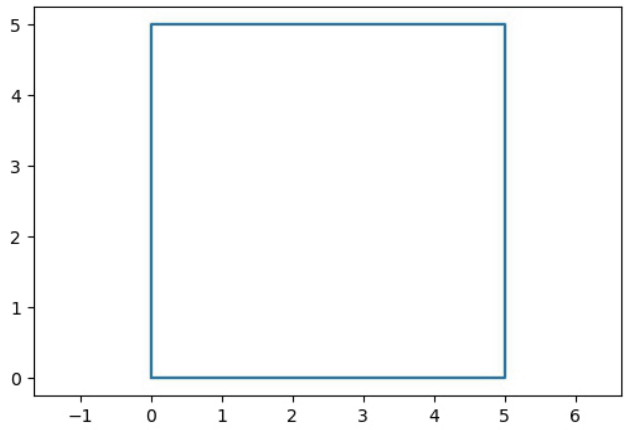
- booléenne pour une image en noir ou blanc ;
- entier naturel pour une image en nuance de gris (compris entre 0 et 255) ;
- 3 entiers naturels pour une image en couleur.

Pour définir la couleur d'un pixel, il faut préciser les trois intensités des trois couleurs primaires qui formeront la couleur du pixel.





Par exemple, le module `matplotlib` crée ses figures par défaut avec des dimensions de 6 x 4 pouces, et une résolution de 100 pixels par pouce.



3.1 module PIL

La bibliothèque PIL propose un type informatique pour gérer les images. La fonction *PIL.Image.open* permet de charger en mémoire une image depuis un fichier. Les différents formats habituels sont acceptés, comme TIFF, JPEG, BMP... Elle renvoie un objet du type PIL.Image qui dispose de nombreuses méthodes et de nombreux attributs. Cette image peut alors être convertie en

- liste : `list(Image)`
- *array* : `np.array(Image)`

3.2 Traitements d'image

3.2.1 Déplacer , redimensionner

Une rotation ou une symétrie d'image implique le déplacement des pixels au sein de celle-ci. On pourra utiliser des syntaxes qui modifie l'image :

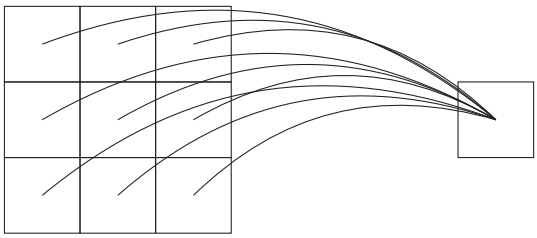
- point par point ;
- ligne par ligne, ou colonne par colonne ;
- l'image intégralement.

Il y aura, en général, autant de pixels à déplacer qu'il y a de pixel dans l'image.

Conseil : bien analyser la transformation géométriques pour définir les règles de déplacement nécessaires.

Application 7 : mirroir Une image est stockée en mémoire vive sous forme d' `array`. Ecrire un programme qui inverse l'image selon un axe de symétrie vertical.

Application 8 : compression par changement de résolution Une image en nuance de gris est stockée en mémoire vive sous forme d' `array`. Ecrire un programme qui réduise chaque dimension de l'image par trois, par moyennage de blocs de 9 pixels. L'image initiale fait 900 pixels de largeur, et 300 de hauteur.



3.2.2 Modifier l'aspect

Couleur vers noir et blanc

Pour un pixel, on note `L[R,G,B]` la liste des trois intensités de couleur du pixel. La transformation en un pixel gris `N` peut être fait par moyennage :

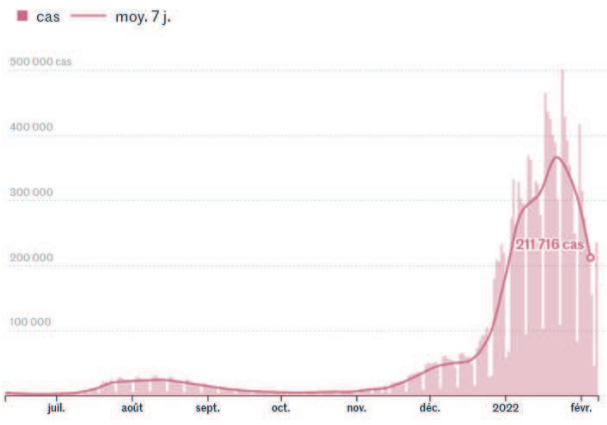
N=1/3*(R+G+B)

Ou en passant par un produit scalaire : N=[1/3,1/3,1/3] · [R,G,B]= H · L

Cette opération est aussi appelée convolution entre 2 vecteurs de même dimension.

En modulant les composantes de H, on peut donner plus de poids à certaines des couleurs primaires.

Paranthèse : filtrage par moyenne glissante



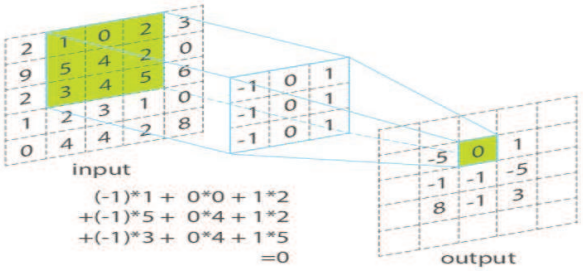
Evolution de la pandémie Covid en France

Moyenne glissante sur 3 jours :

2	15	41	45	53	40	31	15
		1/3	1/3	1/3			
	19,3	33,7	46,3	46,0	41,3	28,7	

Convolution sur les images

Le principe de la convolution peut être utilisé sur des objets de dimension supérieure à 1. Dans le cas d'une image (à deux dimensions), la transformation s'appuie sur une convolution par une matrice dont le contenu définit la transformation.



Exemple :

flou par moyennage	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	
flou Gaussien	$\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$	
détection de contour	$\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$	

3.3 Mémo – Manipulation d'images

```python
import numpy as np                  #bibliothèque de gestion des tableaux
from PIL import Image               #bibliothèque de gestion des images
import matplotlib.pyplot as plt     #bibliothèque de gestion des graphiques

T = np.array(Image.open("image.png")) #charge une image dans un tableau

#Pour afficher l'image d'un tableau T, quatre instructions sont nécessaires.
plt.figure()
plt.imshow(T)                       #pour une image en couleurs RGB(A)
plt.imshow(T,cmap='gray')           #pour une image noir & blanc
                                    #ou en nuances de gris
plt.axis("off")
plt.show()

Image.fromarray(T).save("fichier.png") #enregistrer une image en "fichier.png"
```