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.

1.1.2 Application

Ecrire un programme qui crée un tableau de 0 de 40 lignes et 80 colonnes.

1.1.3 Traitement d'un tableau

Proposer un programme qui recherche la plus grande valeur du tableau A ? Combien d'instructions élémentaires sont nécessaires ?

1.2 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.

Par exemple, écrire 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 se fait par conversion d'une liste de listes en **array** :

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

Quatre fonctions permettent de générer automatiquement des *arrays* de propriétés particulières :

- **arange** : équivalent à *range*, mais génère un array :

```
>>> 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**$([l, n])$: 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 mathématique d'addition et de multiplication :

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

2.3 Extraction ou *Slicing*

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

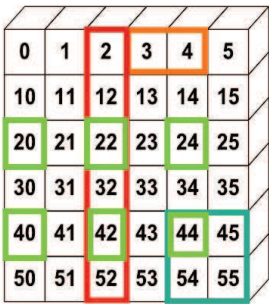
- accès à l'intégralité du tableau avec A
- accès à un élément du tableau avec $A[3, 2]$
- accès à une ligne du tableau avec $A[3, :]$
- accès à une colonne du tableau avec $A[:, 2]$
- accès à un sous tableau avec $A[2 : 4, 1 : 3]$

```
>>> 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]])
```



2.4 Application

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

2.5 Le type *mat*

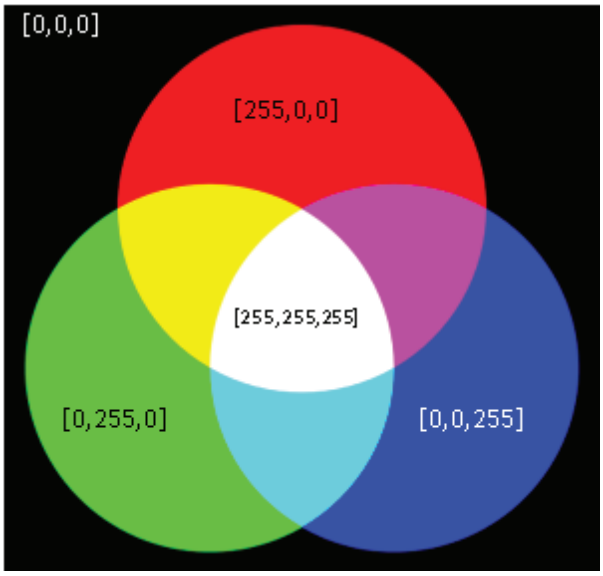
Enfin, il faut savoir qu'il existe, dans **NumPy**, un style dédié au calcul matriciel. Ceci sera détaillé plus tard dans l'année scolaire.

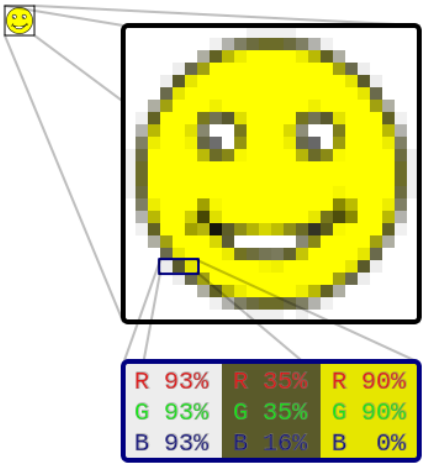
3 Images

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.





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)`