# CALCUL MATRICIEL EN PYTHON

P.PISZYNA – JUIN 2019

# LES TABLEAUX NUMPY

- numpy est une bibliothèque qui offre un type supplémentaire par rapport aux types de base Python : le **tableau**, qui s'appelle en anglais array (en fait techniquement, ndarray, pour *n-dimension array*).

- Création d'un tableau à 1 dimension :

```
import numpy as np
array = np.array([12, 25, 32, 55])
range = np.array(range(10))
f = np.arange(1.0, 2.0, 0.1)
```

- Comment accède t-on à un élément d'un tableau ?
- Aussi, si vous utilisez une bibliothèque de calcul scientifique, la quasi totalité des objets que vous serez amenés à manipuler seront des tableaux numpy.

# LES TABLEAUX NUMPY

- Les tableaux Numpy sont des collections d'éléments de même type.

- Création d'un tableau à 1 dimension :
- Affichage du type :
- D'autres types :
- Ce qu'il faut en retenir :

```
import numpy as np
a = np.array([1, 2, 4, 8])
# pour accéder au type d'un tableau
a.dtype
f = np.array([1, 2, 4, 8.])
c = np.array([1, 2, 4, 8j])
```

  - vous pouvez choisir entre bool, int, uint (entier non signé), float et complex,
  - ces types ont diverses tailles pour vous permettre d'optimiser la mémoire réellement utilisée,
  - ces types existent en tant que tel (hors tableaux),
  - Liste complète des types : https://docs.scipy.org/doc/numpy/user/basics.types.html

# LES TABLEAUX NUMPY

- Création et manipulation de tableaux multidimensionnels :

- Création d'un tableau à 2 dimensions :
- Affichage des dimensions :
- Changement des dimensions :
- Attributs liés à la forme :

```python
import numpy as np
```

```python
d2 = np.array([[11, 12, 13], [21, 22, 23]])
```

```python
# la forme (les dimensions) du tableau
d2.shape
```

```python
# l'argument qu'on passe à reshape est le tuple
# qui décrit la nouvelle *shape*
v2 = d2.reshape((3, 2))
```

```python
# le nombre de dimensions
d2.ndim
```

```python
# le nombre de cellules
d2.size
```

- Relation entre d2 et v2 ?

# LES TABLEAUX NUMPY

- Création de tableaux constants:

```
import numpy as np
```

- Création d'un tableau de zéros :

```
zeros = np.zeros(dtype=np.int64, shape=(10, 10))
```

- Avec cinq :

```
fives = 5 * np.ones(dtype=float, shape=(8, 8))
```

- Matrice identité :

```
ident = np.identity(3)
```

- Utilisation de **dtype** ?

# LES TABLEAUX NUMPY

- Accès aux éléments par les indices :
- Fonction de création d'un tableau 5 x 5 :

```
def background(n):
    tab = np.zeros(dtype=np.int8, shape=(n, n))
    for i in range(n):
        for j in range (n):
            tab[i,j] = 10 * i + j
    return tab
```

- Résultat :

```
In [2]: background(5)
Out[2]:
array([[ 0,  1,  2,  3,  4],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24],
       [30, 31, 32, 33, 34],
       [40, 41, 42, 43, 44]], dtype=int8)
```

# LES TABLEAUX NUMPY

- Accès aux éléments
- Création d'un tableau 5 x 5

```
import numpy as np
```

```
a5 = [[ 0  1  2  3  4]
      [10 11 12 13 14]
      [20 21 22 23 24]
      [30 31 32 33 34]
      [40 41 42 43 44]]
```

- Voir une ou plusieurs cellules :

```
a5[1]
```

```
a5[1][2]
```

```
a5[1, 2]
```

- Modifications :

```
a5[2][1] = 221
a5[3, 2] += 300
```

```
a5[1] = np.arange(100, 105)
```

```
a5[4] = 400
```

# LES TABLEAUX NUMPY

- Accès aux éléments
- Création d'un tableau 5 x 5

```
import numpy as np
```

```
a5 = [[ 0  1  2  3  4]
      [10 11 12 13 14]
      [20 21 22 23 24]
      [30 31 32 33 34]
      [40 41 42 43 44]]
```

- Slicing (coupe) :

```
a5[:, 3]
```

```
a5[:, 3] = range(300, 305)
```

- Broadcasting (diffusion) :

```
a5[:, 2] = 200
```

```
a5[:, 4] += 400
```

# LES TABLEAUX NUMPY

- Exercice :
  - On vous demande d'écrire une fonction **stairs(k)** qui crée un tableau numpy
  - La fonction prend en argument un entier **k** et construit un tableau carré de taille $(2 * k + 1)^2$
  - Aux quatre coins du tableau on trouve la valeur **0**
  - Dans la case centrale on trouve la valeur **2 * k**
  - Si vous partez de n'importe quelle case et que vous vous déplacez d'une case horizontalement ou verticalement vers la case centrale, vous incrémentez la valeur du tableau de **1**

```
In [30]: stairs(2)
Out[30]:
array([[0, 1, 2, 1, 0],
       [1, 2, 3, 2, 1],
       [2, 3, 4, 3, 2],
       [1, 2, 3, 2, 1],
       [0, 1, 2, 1, 0]])
```

```
In [29]: stairs(3)
Out[29]:
array([[0, 1, 2, 3, 2, 1, 0],
       [1, 2, 3, 4, 3, 2, 1],
       [2, 3, 4, 5, 4, 3, 2],
       [3, 4, 5, 6, 5, 4, 3],
       [2, 3, 4, 5, 4, 3, 2],
       [1, 2, 3, 4, 3, 2, 1],
       [0, 1, 2, 3, 2, 1, 0]])
```

```
In [28]: stairs(4)
Out[28]:
array([[0, 1, 2, 3, 4, 3, 2, 1, 0],
       [1, 2, 3, 4, 5, 4, 3, 2, 1],
       [2, 3, 4, 5, 6, 5, 4, 3, 2],
       [3, 4, 5, 6, 7, 6, 5, 4, 3],
       [4, 5, 6, 7, 8, 7, 6, 5, 4],
       [3, 4, 5, 6, 7, 6, 5, 4, 3],
       [2, 3, 4, 5, 6, 5, 4, 3, 2],
       [1, 2, 3, 4, 5, 4, 3, 2, 1],
       [0, 1, 2, 3, 4, 3, 2, 1, 0]])
```

# LES TABLEAUX NUMPY

- Visualisation du tableau à travers une image en niveaux de gris :
  - Une fois la fonction **stairs** testée et mise au point, on va utiliser les fonctions :
    - **astype** de **numpy** pour convertir le type des éléments du tableau
    - **imshow** de **matplotlib** pour afficher une image

```
def image():
    tab = stairs(100)
    # convertir en flottant pour imshow
    squares = tab.astype(np.float)
    # afficher avec une colormap 'gray'
    plt.imshow(squares, cmap='gray');
    return
```

- Pour les plus avancés:
  - Proposer une modification de la fonction **image** permettant un affichage en couleur
  - Quelle est la taille occupée par cette image en mémoire ?

# LES TABLEAUX NUMPY

- Opérations sur les lignes et les colonnes :
  Les opérations définies dans Numpy s'appliquent aussi aux tableaux :
  - Pour 2 tableaux **a** et **b**, l'opération **a + b** effectuera la somme des éléments de même(s) indice(s) entre eux.
  - Pour un tableau **a** et un scalaire **t**, l'opération **t x a** effectuera le produit scalaire.
  - Pour 2 tableaux **a** et **b**, l'opération **a * b** N'EFFECUE PAS le produit matriciel MAIS le produit des éléments de même(s) indice(s) entre eux. Il faut utiliser la fonction **dot(a, b)** pour cela.

- Pour le calcul matriciel sur un tableau **a** :
  - **a[i] = a[i] + t * a[j]** permet d'ajouter la ligne $\mathbf{t.L_j}$ à la ligne $\mathbf{L_i}$
  - **a[i] = t * a[i]** permet de multiplier la ligne $\mathbf{L_i}$ par **t**
  - **a[[i, j]] = a[[j, i]]** permet de permuter les lignes $\mathbf{L_i}$ et $\mathbf{L_j}$
- Ces opérations modifient le tableau **a**, et seront utilisées dans la **méthode du pivot de Gauss** (résolution d'un système d'équations linéaires).

# LES TABLEAUX NUMPY

- Opérations logiques :
  - Création d'un tableau 5 x 5 :

```
a = np.arange(25).reshape(5, 5)
```

  - Copie et modification :

```
b = np.copy(a)
b[2, 2] = 1000
```

  - Que renvoient :

```
print(a == b)
```

```
np.zeros(5).any()
```

```
np.ones(5).any()
```

  - Opérateurs à tester :

```
width = 128
center = width / 2

ix, iy = np.indices((width, width))
image = (ix-center)**2 + (iy-center)**2
# pour afficher l'image en niveaux de gris
plt.imshow(image, cmap='gray');
```

```
rayures = (ix + iy) % 8 <= 5
plt.imshow(rayures, cmap='gray');
```

```
anti_rayures = np.logical_not(rayures)
plt.imshow(anti_rayures, cmap='gray');
```

# SOURCES & RÉFÉRENCES :

- MOOC **Python 3 : des fondamentaux aux concepts avancés du langage** (Thierry Parmentelat & Arnaud Legout) :
- **https://www.fun-mooc.fr/courses/inria/41001S03/session03/about**

- **TSI Louis le Grand** (Jean-Pierre Becirspahic)
- **https://info-llg.fr/commun-mpsi/pdf/10.slide.pdf**