TP 12 Python – Tableaux `numpy`

Les tableaux sont des structures de données qui permettent d'effectuer facilement des opérations sur les données qu'ils contiennent. Ils permettent par exemple de représenter des **matrices**, ou encore des grilles de jeux, ou des images.
C'est un type de structure de données *homogènes* (toutes les données doivent être du même type), et *modifiable* : on peut modifier un ou une partie des éléments du tableau.
Les tableaux, de type `array`, existent déjà dans la bibliothèque standard de python. Cependant, on va privilégier la bibliothèque `numpy`, qui possède de nombreuses fonctions prédéfinies pour manipuler les tableaux. On commencera donc par importer `numpy` sous l'*alias* `np` : `import numpy as np`.
Ainsi, lorsqu'on appelle une fonction du module `numpy`, on prendra garde à utiliser le **préfixe** « `np.` ».

1. Créer un tableau

1.A) Manuellement

On peut créer un tableau manuellement, avec l'instruction :

- `t = np.array([x1, x2, ..., xn])` ou `t = np.array(L)` pour créer un tableau à une dimension à partir d'une liste
- `t = np.array([[x11, x12, ..., x1p], ...,[xn1, xn2, ..., xnp]])` ou `t = np.array(M)` pour créer un tableau à deux dimensions à partir d'une liste de listes.

Exemple

1. Créer un tableau à une dimension contenant les entiers $3, 7, 5, 2$: .....
2. Que renvoie l'instruction : `np.array([1, 1.5, 7])` ? .....
3. Que renvoie l'instruction : `np.array([1, "bla", 7])` ? .....
4. Créer un tableau représentant la matrice $\begin{pmatrix} 1 & 6 \\ -2 & 0 \end{pmatrix}$: .....
5. Même question avec I_3 : .....
6. Même question avec $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$: .....

1.B) Par compréhension

Comme avec les listes, on peut créer un tableau avec une syntaxe par **compréhension** :

- `T = np.array([f(k) for k in L])` pour un tableau unidimensionnel
- `T = np.array([ [f(j) for j in colonnes] for i in lignes])` pour un tableau bidimensionnel

où `L`, `lignes` et `colonnes` désignent des listes (ou des tableaux) déjà prédéfinis, ou alors des `range` ou des chaînes de caractères.

Exemple

1. Créer un tableau unidimensionnel contenant les 10 premiers entiers positifs au cube.
2. Créer un tableau représentant la matrice $A \in \mathcal{M}_{6,3}(\mathbb{R})$ de terme général $a_{i,j} = \dfrac{1}{i + 2^j}$.
3. Écrire une fonction qui prend en argument deux entiers $n, p \in \mathbb{N}^*$ et qui renvoie le tableau représentant la matrice $0_{n,p}$. La fonction doit faire exactement 2 lignes, par compréhension !

1.C) Avec les fonctions `numpy`

La bibliothèque `numpy` contient un certain nombre de fonctions permettant de créer des matrices particulières, comme la matrice nulle, la matrice identité, des matrices diagonale ou des matrices aléatoires.

Instruction Python	Tableau créé	Exemple(s)
`np.zeros(n)`	Tableau "ligne" avec n zéros	`np.zeros(3)` → `array([0., 0., 0.])`
`np.zeros((n,p))`	Matrice $0_{n,p}$	`np.zeros((3,2))` →
`np.ones(n)`		`np.ones(4)` →
`np.ones((n,p))`		`np.ones((2,3))` →
`np.eye(n)`		`np.eye(3)` →
`np.diag([x1, ..., xn])`		`np.diag([1, -5])` →
`np.random.rand(n,p)`		`np.random.rand(2,2)` ⟶
`np.random.randint(a, b, (n,p))`		`np.random.randint(0,2,(1,4))` ⟶
`np.arange(min, max, pas)`		
`np.linspace(min, max, nb)`		

Remarque. Ces deux dernières fonctions, très utiles pour créer des listes d'abscisses pour tracer des courbes, créent en fait des **tableaux**.

2. Accéder aux éléments d'un tableau et les modifier

On utilise globalement la même syntaxe que pour les listes. Par exemple, avec `T=np.array([-3, 1, 0, 2])` :

a) `T[2]` = b) `T[3:]` = c) `T[1:3]` = d) `T[:]` =

Mais attention, pour un tableau bidimensionnel `T`, l'élément à la position i, j est désigné par : `T[i,j]`

⚠ Ne pas confondre la syntaxe pour les listes de listes (.....) et celle pour les tableaux (.....) !

Pour les exemples, on utilisera le tableau M représentant la matrice $M = \begin{pmatrix} -1 & 2 & 0 & 3 \\ 4 & -2 & 1 & -4 \end{pmatrix}$.

Instruction Python	Résultat	Exemple(s)
`T[i,j]`	Élément à la position (i, j)	`M[1,2]` ⟶
`T[i,:]`	Ligne i	`M[0,:]` ⟶
`T[:,j]`	Colonne j	`M[:,1]` ⟶
`T[i1:i2, j1:j2]`	Lignes de i_1 à $i_2 - 1$ et colonnes de j_1 à $j_2 - 1$	`M[0:2,1:3]` ⟶

On peut modifier un élément d'un tableau avec la syntaxe `T[i,j]=x`, comme pour les listes.
On peut également, de la même manière que ci-dessus, remplacer d'un coup toute une colonne ou tout une ligne ! Par exemple, que devient M après chacune des instructions suivantes ?

- `M[0, 2] = 4`
- `M[1, :] = [0, 1, 2, 3]`
- `M[:, 2] = [1, 2]`
- `M[:, 0] = [3, -2, 6]`
- `M[0:2, 1:3] = np.zeros((2,2))`
- `M[1] = [-1, 0, 1, 0]`
- `M[0,0] = "blah"`

Enfin, on peut **supprimer une ligne ou une colonne** d'un tableau de la manière suivante :

Instruction Python	Résultat	Exemple(s)
`np.delete(T, k)`	En unidimensionnel, supprime l'élément à la position k	`np.delete(T,0)` ⟶
`np.delete(M, i, 0)`	Supprime la **ligne** i	`np.delete(M, 1, 0)` ⟶
`np.delete(M,j,1)`	Supprime la **colonne** j	`np.delete(M, 2, 1)` ⟶

Remarque. Attention, l'opération `T2=T` ne permet pas de créer un nouveau tableau ! Toute modification faite à `T` se répercutera sur `T2`.
Pour éviter ce problème, on écrira plutôt : `T2 = np.copy(T)`

3. Opérations sur les tableaux

De nombreuses opérations sur les tableau sont déjà présentes dans le module `numpy`.
On en donne ici les plus importantes. Dans les exemples, on prendra les tableaux suivants : `M = np.array([[1, 2, 3], [4, 5, 6]])`, `T = np.array([[0, 1, 2], [-1, 0, 1]])`, `u = np.array([1, 2])` et `v = np.array([[1] [2], [-1]])`.

Instruction Python	Résultat	Exemple(s)
`np.size(t)`	Nombre total d'éléments	`np.size(M)` $\longrightarrow$
`np.size(t, 0)`	Nombre de lignes	`np.size(M, 0)` $\longrightarrow$
`np.size(t, 1)`	Nombre de colonnes	`np.size(M, 1)` $\longrightarrow$
`x in t`	Test d'appartenance	`2 in M` $\longrightarrow$
`np.sum(t)`	Somme de tous les éléments	`np.sum(M)` $\longrightarrow$
`np.prod(t)`	Produit de tous les éléments	`np.prod(M, 0)` $\longrightarrow$
`np.min(t)`	Valeur du maximum	
`np.max(t)`	Valeur du minimum	

Il est possible d'appliquer des opérations aux tableaux, **coefficient par coefficient**, à l'aide des symboles usuels d'opérations. Par exemple, essayez les opérations suivantes :

a) `M + T` b) `M - T` c) `3 * M`
d) `T * M` e) `M ** 2` f) `M ** T`
g) `T / M` h) `np.log(M)` i) `np.sqrt(u)`

Remarques sur ces opérations : .....
.....
.....

Pour effectuer les opérations matricielles, on utilise les fonctions suivantes :

Instruction Python	Résultat	Exemple(s)
`np.dot(u,v)`	Produit scalaire entre 2 vecteurs	`np.dot(u,u)` $\longrightarrow$
`np.dot(M,N)`	**Produit matriciel**	`np.dot(M, v)` $\longrightarrow$
`np.transpose(M)`	Transposée	`np.transpose(T)` $\longrightarrow$
`np.linalg.inv(M)`	Inverse	`np.linalg.inv(np.diag([1, 3, -2]))` $\longrightarrow$

4. Exercices

Exercice 1 (Manipulations)**.** 1. Créer des tableau qui correspondent à chacune des matrices suivantes :

- $A = \begin{pmatrix} 2 & 3 & -6 \\ 9 & -1 & 3 \end{pmatrix}$: .....
- B la matrice diagonale de taille 3, dont les coefficients diagonaux sont 3, 7 et -4. : .....
- C la matrice carrée de taille 20 dont tous les coefficients sont des 3, sauf ceux sur la diagonale qui sont des -2. : .....
- $D \in \mathcal{M}_{10}(\mathbb{R})$ qui contient dans l'ordre tous les entiers de 1 à 100.
- (*) $E \in \mathcal{M}_{10}(\mathbb{R})$ qui contient des 1 sur le "bord" et des 0 ailleurs : à faire en 2 lignes ! .....
- M un tableau de votre choix de taille 5×3. : .....

2. Supprimer la quatrième ligne du tableau M : .....
3. Remplacer le terme de la première ligne, deuxième colonne de ce nouveau tableau par M : .....
4. Échanger les deux premières colonnes du tableau M : .....
5. Afficher les dimensions de votre tableau : .....

Exercice 2. 1. Écrire une fonction qui détermine si deux matrices représentées par des tableaux sont égales. Pour s'éviter les problèmes liés aux égalités sur des flottants, on dira que deux flottants sont égaux si leur différence est plus petite que 10^{-6} en valeur absolue.

2. Écrire une fonction `éléments_communs(t1, t2)` qui prend en entrée deux tableaux (bidimensionnels), et qui renvoie une liste contenant tous les éléments qui sont à la fois dans `t1` et dans `t2`.
3. Écrire une fonction qui prend en entrée deux tableaux unidimensionnels, et qui leur retire tous les éléments qu'ils ont en commun.

Exercice 3. 1. Écrire une fonction qui teste si une matrice carrée est ou non triangulaire supérieure.

2. Écrire une fonction qui prend en argument une matrice et qui renvoie :
   - un message d'erreur si la matrice n'est pas triangulaire ;
   - si la matrice est triangulaire : un booléen qui indique si elle est inversible ou non.

Exercice 4. On dit qu'une matrice carrée $M \in \mathcal{M}_n(\mathbb{R})$ est orthogonale si, en notant $C_1, \dots, C_n$ ses colonnes, on a : $\forall i, j \in [\![1, n]\!]$, $C_i \cdot C_j = \begin{cases} 0 & \text{si} \quad i \neq j \\ 1 & \text{si} \quad i = j \end{cases}$ où l'opération effectuée est le produit scalaire.

1. Écrire une fonction qui prend en argument une matrice M, et qui renvoie un booléen qui indique s'il s'agit ou non d'une matrice carrée. On affichera une erreur dans le cas où la matrice n'est pas carrée.
2. On admet qu'une matrice A est orthogonale si et seulement si elle vérifie $A^\top = A^{-1}$. En déduire une seconde fonction, qui fait la même chose que la précédente, mais en beaucoup moins de lignes !
3. Bonus : montrer le résultat admis à la question précédente.

Exercice 5 (Le Jeu de la Vie)**.** Faites des recherches *rapides* sur le principe du Jeu de la Vie de Conway. Dans cet exercice, on considère une grille représentant le jeu de la vie, codée par un tableau numpy carré.

1. Écrire une fonction qui prend en argument un paramètre $p \in [0, 1]$, et qui renvoie 1 avec probabilité p, et 0 avec probabilité $1 - p$. On utilisera la fonction `random()` de la bibliothèque `random`.
2. En déduire une fonction qui prend en argument une taille n et un paramètre p, et qui renvoie une grille aléatoire où chaque case vaut 1 avec probabilité p et 0 sinon. Cette grille représente la position de départ de notre Jeu de la Vie.
3. Écrire une fonction `coords_voisins(t, coord)` qui prend en argument un tableau `t` et un couple de coordonnées valides `coord`, et qui renvoie une liste contenant les coordonnées de chacun de ses voisins directs (orthogonaux) sur la grille.
   Attention : la numérotation commence à 0, et les cases sur le bord ont mois de voisins que les autres !

4. En déduire une fonction `nombre_voisins_vivants(grille, coord)` qui prend en paramètre un tableau `grille` du jeu de la vie et un couple de coordonnées désignant une cellule, et qui renvoie le nombre de voisins vivants de cette cellule.
5. Coder finalement une fonction `génération_suivante(grille)` qui prend en paramètre une grille du jeu de la vie et qui renvoie la grille à l'étape suivante.
6. **Bonus :** Pour les plus motivés, vous pouvez :
   - tenter de poursuivre les générations jusqu'à atteindre un état stable et afficher l'état stable (attention, on n'arrive pas toujours à un état stable, on s'arrêtera après un nombre fixé d'itérations si ce n'est pas le cas).
   - essayer d'afficher les générations successives joliment pour faire une superbe animation.