TD 07

Structure de données : tableaux (ou *array*)

Lycée Louis Thuillier - Informatique - PCSIB -

Il est possible en python de créer des listes de listes de nombres. On obtient alors une structure proche d'un tableau à deux dimensions : la première liste contient les nombres de la première ligne, la deuxième ceux de la deuxième ligne, ...
Pour des raisons qu'on ne détaillera pas ici, cette façon de procéder n'est pas optimale. Par exemple, on peut remarquer qu'une liste peut contenir des éléments divers. Or on souhaite ici ne faire que des tableaux de nombres : une liste est un outil trop "perfectionné" pour cela, et donc non optimal.
Il existe une structure plus adapté : les *array*, appelés parfois "tableaux".

1 Le module *numpy*

Afin de travailler avec les *array* en Python, on a besoin d'importer le module *numpy*.

1.1 C'est quoi un module

Définition. Module
Un module, appelé également bibliothèque, est un ensemble de programmes qu'on va importer (*i.e.* apprendre à notre ordinateur) lorsqu'on en a besoin.

Exemple 1 :

On veut calculer ln(3) (*Attention ! seul les francais écrivent log ln ! !)*

- *Python ne connaît pas la fonction log*
- *Je lui demande alors d'importer depuis le module math (tout un tas de fonction mathématique déjà codées) la fonction log*
- *ça marche !*

- *on peut importer toutes les fonctions mathématiques via la commande * (mis à part pour math c'est déconseillé !)*

```
>>> log(3)
Traceback (most recent call last):
  File "<console>", line 1, in <module
NameError: name 'log' is not defined

>>> from math import log

>>> log(3)
1.0986122886681098

>>> from math import *
```

Liste des modules "classiques" :

- module *math* pour les fonctions mathématiques
- module *matplotlib.pyplot* pour faire des graphes
- module *random* pour générer des nombres aléatoires

1.2 Le module numpy

On va importer le module numpy (*numerical Python*) en lui donnant un surnom, ici *np* :

import numpy as np

Remarque : *Le choix du nom np n'est pas obligatoire, on aurait pu prendre ce qu'on veut, mais il est canonique.*

A partir de maintenant, toutes les fonctions/méthodes relatives au module numpy s'écriront *np.nomdelaméthode*.

2 Définition, création et affichage d'un tableau

2.1 Définition

Un *array* est une structure ordonnée de données, **modifiable et homogène** (les données doivent être toutes de même type, souvent des nombres). Un *array* s'écrit alors *np.array* car il provient de la bibliothèque numpy.
On va se servir pour la suite d'un *array* d'exemple, qu'on nomme *Tableau*.

```
>>> Tableau=np.array([[1,2],[3,4]])
```

Un *array* à deux dimensions peut être vu comme un tableau avec des lignes et des colonnes.

```
>>> print(Tableau)
[[1 2]
 [3 4]]
```

Chaque élément du tableau est caractérisé par un ou plusieurs indice :
- ▷ $Tableau[i]$ renvoie au $i$ème élément de $Tableau$.
- ▷ $Tableau[i][j]$ revoie au $j$ème élément du $i$ème élément de $Tableau$
- ▷ ...

```
>>> Tableau[1][1]
4

>>> Tableau[0]
array([1, 2])
```

Attention ! un tableau ne peut contenir que des entiers (de type *int*) ou des nombres flottants (de type *float*), pas les deux en même temps.

Attention ! un tableau n'est pas redimensionnable : la taille d'un tableau est fixée à sa création.

2.2 Tableau *versus* Liste

Quelle différence entre :
- ▷ un tableau à deux dimensions

```
>>> Tableau=np.array([[1,2],[3,4]])
```

- ▷ une liste de liste

```
>>> Liste=[[1,2],[3,4]]
```

Parce que ça a l'air d'être la même chose ... Quelques différences :

1. Un tableau ne contient que des éléments du même type : que des nombres entiers ou que des réels par exemple. Dans une liste on peut "stocker" ce qu'on veut.
2. Un tableau est issu d'une bibliothèque : c'est une structure qui n'est pas inscrit dans les fonctions "de base" de Python. Il faut toujours importer au préalable la bibliothèque *numpy*.
3. un tableau permet de faire des opérations différentes d'une liste

> ***Application 1 :***
> - ▷ *Essayer de sommer $Tableau + Tableau$ et $Liste + Liste$. Que fait la commande $+$ pour un tableau ? Et pour une liste ?*
> - ▷ *Essayer de multiplier $Tableau * Tableau$ et $Liste * Liste$. Que fait la commande $*$ pour un tableau ? Et pour une liste ?*

4. on ne peut pas ajouter ou enlever des éléments d'un tableau : on peut uniquement modifier les éléments.

Propriété. *array* **ou liste**
Un tableau est une structure dérivée de la liste : il est plus restrictif (on ne peut pas ajouter des éléments, ...) et possède des méthodes qui lui sont propres (pas le même +, -, ...).
Un *array* est une liste spécialisée dans les tableau de nombres : tout va plus vite !

3 Opération sur les tableaux

L'intérêt d'un *array* c'est de pouvoir récupérer, modifier et travailler sur les nombres qu'il contient.

3.1 Taille d'un tableau

▷ La commande *shape* donne le nombre de lignes et de colonne du tableau
▷ Le nombre total d'éléments dans un tableau est donné par la commande *size*.

```
>>> tableau=np.array([[1,2,3],[4,5,6]])

>>> print(tableau)
[[1 2 3]
 [4 5 6]]

>>> print(np.shape(tableau))
(2, 3)

>>> print(np.size(tableau))
6
```

> ***Application 2 :*** *Comment récupérer le nombre de lignes d'un tableau à l'aide de la commande shape ? Comment récupérer le nombre de colonnes ?*

3.2 Accès à un ou plusieurs éléments

Accéder et modifier des éléments dans un tableau fonctionnent exactement comme pour les listes (∼ un tableau peut être vu comme une liste de listes)

▷ son premier élément $T[0]$ est un tableau à une dimension (∼ une liste)

▷ le premier élément du premier élément $T[0][0]$ est donc un nombre, ici 1

▷ le deuxième élément du premier élément $T[0][1]$ est donc également un nombre, ici 1

```
>>> print(T)
[[1 2 3]
 [4 5 6]
 [7 8 9]]

>>> T[0]
array([1, 2, 3])

>>> T[0][0]
1

>>> T[0][1]
2
```

Propriété. Accès à un élément
Soit un *array* nommé Tab.

$Tab[i][j]$ représente l'élément de Tab situé à la i-è*me* ligne, j-ième colonne

Attention ! on commence à compter à 0 !

> ***Application 3 :*** *Donner la commande qui permet d'accéder à l'élément d'un tableau situé :*
> ▷ *en haut à droite*
> ▷ *en bas à gauche*

Souvent on souhaitera accéder successivement à **tous les éléments d'un tableau**. Il faut alors parcourir toutes les lignes (⇒ une boucle for) et toutes les colonnes (⇒ une boucle for).

> ***Application 4 :*** *T est un tableau "inconnu" à deux dimensions. Compléter le programme afin qu'il affiche tous les éléments du tableau.*

```
for i in range(.......................):

    for j in range(.........................):

        print(.....................)

```

Attention ! Cette structure est **très importante** : elle permet d'accéder à touts les éléments d'un tableau !
On pourra ensuite les modifier, les utiliser, ...

3.3 Création

Il existe de nombreuses méthodes, nous allons lister les plus courantes.

Attention ! Comme un tableau est de taille fixé, on ne peut pas le construire en ajoutant progressivement des éléments à un tableaux initialement vide ⇒ **on oublie la commande** *append* **!!!** Cette technique marche pour les listes mais pas les tableaux.

1. **A la main**
   Rentrer un à un tous ses éléments à l'aide de la fonction *array* du module *numpy*. (*à n'utiliser que pour des petites tableaux !)*

   ```
   >>> Tableau=np.array([[1,2],[3,4]])
   ```

   Attention ! On n'oublie pas le *np* devant !
   Attention ! L'écriture est un peut lourde ! C'est *np.array*() dans lequel on met une liste de liste, donc entre [], des éléments.

   > ***Application 5 :*** *Créer à la main un tableau* ***à deux dimensions*** 2×2 *contenant tous les nombres premiers entre* 0 *et* 10.
   > *Créer une variable l renvoyant à la seconde ligne du tableau. Créer une variable x renvoyant à la valeur en bas a gauche du tableau.*

2. **arrange + reshape**
   La fonction $arange(a, b, k)$ de *numpy* crée le tableau à une dimension allant de a inclus à b exclu par pas de k.

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

   > ***Application 6 :*** *Créer un tableau à une dimension "entiers_pairs" contenant tous les entiers pairs entre* 0 *et* 100.
   > *Créer un tableau à une dimension "entiers_6" contenant tous les entiers multiples de* 6 *entre* 0 *et* 100.

   ▷ La fonction $reshape(n, m)$ permet, à partir d'un tableau à une dimension qui possède $n \times m$ éléments, de créer un tableau à deux dimensions avec n lignes et m colonnes.
   Attention ! *reshape* n'est pas une méthode : elle ne modifie pas le tableau sur lequel elle s'applique !! Il faut affecter le nouveau tableau à une nouvelle variable.

```
>>> liste_origine=np.arange(0,24,1)

>>> tableau=liste_origine.reshape(4,6)

>>> print(tableau)
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]]
```

Application 7 :
- ▷ *Créer un tableau* (10×10) *entiers contenant tous les entiers entre* 0 *et* 100.
- ▷ *Créer un tableau* (5×10) *entiers*$_i$*mpairs contenant tous les entiers impairs entre* 0 *et* 100.

3. **Modifier un tableau initialement rempli de 0** (*la plus utile !*)

La fonction $zeros((n,m))$ crée un tableau à deux dimensions de valeurs nulles possédant n lignes et m colonnes.

```
>>> np.zeros((3,4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

| ***Remarque :*** *La fonction ones fonctionne de façon similaire, mais crée un tableau rempli de* 1

On va ensuite modifier un à un chaque élément pour leur donner la valeur souhaitée.

Application 8 :
1. *Créer un tableau* 50×50 *où chaque élément est égal au numéro de sa colonne*
2. *Créer un tableau* 50×50 *où chaque élément est la somme du numéro de sa ligne et de sa colonne*
3. *Créer un tableau* 50×50 *où tous les éléments sont nuls sauf ceux sur la diagonale*

4 Exercices classiques

Application 9 : ***Somme dans un tableau***
1. *Créer les fonction "ligne" et "colonne" qui prennent en argument un entier* i *et un tableau* T *et renvoie la somme des éléments de la* i*-ème ligne ou* i*-ème colonne.*
2. *Créer une fonction "diagonale" qui prend un tableau* T *et renvoie la somme des éléments de la diagonale gauche* $\rightarrow$ *droite.*
3. *Créer une fonction moyenne qui prend en entrée un tableau* T *et renvoie la moyenne de tous les éléments de* T*.*
4. *(*) Créer une fonction moyenne_locale qui prend en entrée un tableau* T *et un couple de nombres* i *et* j *et qui renvoie la moyenne des* 9 *nombres adjacent à l'élément* $T[i][j]$*. (on incluera* $T[i][j]$ *dans la moyenne)*

Application 10 : ***Sommet et multiplication entre tableaux***
Dans cet exercice, on cherche à re-coder les fonctions $+$ *et* $*$ *entre deux tableaux. Interdit donc de les utiliser !*
1. *Ecrire une fonction somme qui prend en entrée deux tableaux* $T1$ *et* $T2$ *et renvoie un tableau* T *où chaque élément est la somme des éléments de* $T1$ *et* $T2$*.*
2. *Ecrire une fonction multiplication qui prend en entrée deux tableaux* $T1$ *et* $T2$ *et renvoie un tableau* T *où chaque élément est le produit des éléments de* $T1$ *et* $T2$*.*