Tableaux (array)

Informatique pour tous

Rappel sur les listes

Ce qu'on peut faire avec une liste `L` (de type `list`):

1. Contenir des éléments de différents types: `[1, "2", [3., 4]]`

Rappel sur les listes

Ce qu'on peut faire avec une liste `L` (de type `list`):

1. Contenir des éléments de différents types: `[1, "2", [3., 4]]`
2. Ajouter un élément `e` à `L`, en utilisant `L.append(e)`

Rappel sur les listes

Ce qu'on peut faire avec une liste `L` (de type `list`):

1. Contenir des éléments de différents types: `[1, "2", [3., 4]]`
2. Ajouter un élément `e` à `L`, en utilisant `L.append(e)`
3. `L1 + L2` créé une nouvelle liste contenant les éléments de `L1`, puis les éléments de `L2`, en complexité O(`len(L1) + len(L2)`)

Rappel sur les listes

Ce qu'on peut faire avec une liste `L` (de type `list`):

1. Contenir des éléments de différents types: `[1, "2", [3., 4]]`
2. Ajouter un élément `e` à `L`, en utilisant `L.append(e)`
3. `L1 + L2` créé une nouvelle liste contenant les éléments de `L1`, puis les éléments de `L2`, en complexité $O(\texttt{len(L1)} + \texttt{len(L2)})$
4. Si $n \in \mathbb{N}$, `n * L` est un raccourci pour $\underbrace{L + ... + L}_{n}$

Rappel sur les listes

Ce qu'on peut faire avec une liste `L` (de type `list`):

1. Contenir des éléments de différents types: `[1, "2", [3., 4]]`
2. Ajouter un élément `e` à `L`, en utilisant `L.append(e)`
3. `L1 + L2` créé une nouvelle liste contenant les éléments de `L1`, puis les éléments de `L2`, en complexité $O(\texttt{len(L1)} + \texttt{len(L2)})$
4. Si $n \in \mathbb{N}$, `n * L` est un raccourci pour $\underbrace{L + ... + L}_{n}$

`L1 * L2` donne une erreur.

Type array

`array` (tableau) est un type similaire à `list`, défini dans le module `numpy`.
Pour l'utiliser:

```
In [7]: import numpy as np
```

Pour utiliser une fonction `f` de numpy, on doit ensuite écrire:

```
np.f(...)
```

Type array

On peut créer un tableau à partir d'une liste `L` avec `np.array(L)`:

```
In [15]: np.array([1, 2, 3])
Out[15]: array([1, 2, 3])
```

Un `array` doit contenir des éléments **tous de même type**:

```
In [18]: np.array([1, "2", [3., 4]])
---------------------------------------
ValueError
```

Type array

Il est possible de changer la valeur d'un élément d'un tableau, comme pour une liste:

```
In [23]: T = np.array([1, 2, 3])

In [24]: T[1] = 4

In [25]: T
Out[25]: array([1, 4, 3])
```

Type array

Il est possible de changer la valeur d'un élément d'un tableau, comme pour une liste:

```
In [23]: T = np.array([1, 2, 3])

In [24]: T[1] = 4

In [25]: T
Out[25]: array([1, 4, 3])
```

Si on essaie de mettre un type qui n'est pas le bon, Python le convertit automatiquement:

```
In [27]: T[0] = 3.6

In [28]: T
Out[28]: array([3, 4, 3])
```

Type array

Le type commun aux éléments d'un tableau `T` est stocké dans la variable `T.dtype`:

```
In [29]: T = np.array([1, 2, 3])

In [30]: T.dtype
Out[30]: dtype('int64')
```

Par défaut, les entiers de `numpy` sont des entiers relatifs (codage en complément à 2) sur 64 bits.

Type array

Le type commun aux éléments d'un tableau `T` est stocké dans la variable `T.dtype`:

```
In [29]: T = np.array([1, 2, 3])

In [30]: T.dtype
Out[30]: dtype('int64')
```

Par défaut, les entiers de `numpy` sont des entiers relatifs (codage en complément à 2) sur 64 bits.

```
In [33]: T[0] = 2**63-1

In [34]: T[0] = 2**63
---------------------------
OverflowError
```

Type array

Un `array` n'est **pas redimensionnable**: sa taille est fixée une fois pour toute à sa création!

`len(T)` donne la taille d'un tableau `T`.

Type array

Un `array` n'est **pas redimensionnable**: sa taille est fixée une fois pour toute à sa création!

`len(T)` donne la taille d'un tableau `T`.

Il n'existe pas de fonction `T.append`, `T.pop`...

Avantage des array

L'avantage des tableaux est qu'ils permettent de faire simplement et très rapidement des calculs vectoriels et matriciels.

Opérations sur le type array

Si `T1` et `T2` sont deux tableaux de **même taille**:

Opérations sur le type array

Si `T1` et `T2` sont deux tableaux de **même taille**:

1. `T1 + T2` est le tableau obtenu par **addition terme à terme** des éléments de T1 et T2.
   Attention: le + est très différent sur les listes!

Opérations sur le type array

Si `T1` et `T2` sont deux tableaux de **même taille**:

1. `T1 + T2` est le tableau obtenu par **addition terme à terme** des éléments de T1 et T2.
   Attention: le + est très différent sur les listes!
2. `T1 * T2` est le tableau obtenu par **multiplication terme à terme** des éléments de T1 et T2.
3. ...

Opérations sur le type array

```
In [84]: T1 = np.array([1, 2, 3])

In [85]: T2 = np.array([4, 5, 6])

In [86]: T1 + T2
Out[86]: array([5, 7, 9])

In [87]: T1 - T2
Out[87]: array([-3, -3, -3])

In [88]: T1 / T2
Out[88]: array([ 0.25,  0.4 ,  0.5 ])

In [89]: T1 * T2
Out[89]: array([ 4, 10, 18])

In [90]: T1 ** T2
Out[90]: array([  1,  32, 729])
```

Opérations sur le type array

Si les deux tableaux ne sont pas de même taille, on obtient une erreur:

```
In [93]: T1 = np.array([1, 2, 3])

In [94]: T2 = np.array([4, 5, 6, 7])

In [95]: T1 + T2
---------------------------------------
ValueError
```

Opérations sur le type array

On peut aussi faire des opérations avec une constante:

```
In [13]: T = np.array([1.5, -2.3, 3.14])

In [14]: 2*T
Out[14]: array([ 3.  , -4.6 ,  6.28])

In [15]: T - 1
Out[15]: array([ 0.5 , -3.3 ,  2.14])

In [16]: T**3
Out[16]: array([  3.375  , -12.167  ,  30.959144])
```

Vitesse de array vs list

Temps d'exécution pour multiplier par 2 chaque élément:

```
L_test = list(range(10**6))
T_test = np.array(L_test)

def double_list(L):
    L2 = []
    for i in range(len(L)):
        L2.append(2*L[i])
    return L2

def double_array(T):
    return 2*T
```

```
In [105]: %timeit(double_list(L_test))
10 loops, best of 3: 134 ms per loop

In [106]: %timeit(double_array(T_test))
1000 loops, best of 3: 1.14 ms per loop
```

Opérations sur le type array

On peut aussi directement appliquer une fonction à un tableau, auquel cas la fonction s'applique sur tous les éléments:

```
In [113]: T = np.array([1, 2, 3])

In [114]: np.exp(T)
Out[114]: array([  2.71828183,   7.3890561 ,  20.08553692])

In [115]: np.arctan(T)
Out[115]: array([ 0.78539816,  1.10714872,  1.24904577])
```

Listes de dimension ≥ 2

On peut créer des listes de listes (par exemple des matrices).

Si `L` est une liste de listes alors `L[i][j]` est le $j$ème élément de la $i$ème liste.

```
In [17]: L = [[0, 1], [2, 3, 4]]

In [18]: L[0][1]
Out[18]: 1

In [19]: L[1][0]
Out[19]: 2
```

Tableaux de dimension ≥ 2

On peut aussi créer une matrice comme tableau de tableaux.

Création de la matrice $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$:

```
In [26]: M = np.array([ [0, 1], [2, 3] ])

In [27]: M
Out[27]:
array([[0, 1],
       [2, 3]])

In [28]: M[1][1]
Out[28]: 3
```

Tableaux de dimension ≥ 2

On peut aussi créer une matrice comme tableau de tableaux.

Création de la matrice $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$:

```
In [26]: M = np.array([ [0, 1], [2, 3] ])

In [27]: M
Out[27]:
array([[0, 1],
       [2, 3]])

In [28]: M[1][1]
Out[28]: 3
```

Élément ligne `i`, colonne `j` de `M`:

Tableaux de dimension ≥ 2

On peut aussi créer une matrice comme tableau de tableaux.

Création de la matrice $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$:

```
In [26]: M = np.array([ [0, 1], [2, 3] ])

In [27]: M
Out[27]:
array([[0, 1],
       [2, 3]])

In [28]: M[1][1]
Out[28]: 3
```

Élément ligne `i`, colonne `j` de `M`: `M[i][j]`

Nombre de lignes de `M`:

Tableaux de dimension ≥ 2

On peut aussi créer une matrice comme tableau de tableaux.

Création de la matrice $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$:

```
In [26]: M = np.array([ [0, 1], [2, 3] ])

In [27]: M
Out[27]:
array([[0, 1],
       [2, 3]])

In [28]: M[1][1]
Out[28]: 3
```

Élément ligne `i`, colonne `j` de `M`: `M[i][j]`

Nombre de lignes de `M`: `len(M)`

Nombre de colonnes de `M`:

Tableaux de dimension ≥ 2

On peut aussi créer une matrice comme tableau de tableaux.

Création de la matrice $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$:

```
In [26]: M = np.array([ [0, 1], [2, 3] ])

In [27]: M
Out[27]:
array([[0, 1],
       [2, 3]])

In [28]: M[1][1]
Out[28]: 3
```

Élément ligne `i`, colonne `j` de `M`: `M[i][j]`

Nombre de lignes de `M`: `len(M)`

Nombre de colonnes de `M`: `len(M[0])`

Initialisation d'un tableau

Pour initialiser un tableau à 2 dimensions (c'est à dire une matrice) de taille $n \times p$, on peut utiliser `np.zeros( (n, p) )`:

```
In [13]: np.zeros( (3, 5) )
Out[13]:
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

Initialisation d'un tableau

Pour initialiser un tableau à 2 dimensions (c'est à dire une matrice) de taille $n \times p$, on peut utiliser `np.zeros( (n, p) )`:

```
In [13]: np.zeros( (3, 5) )
Out[13]:
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

Question

Un entier n étant défini, créer une matrice identité de taille $n \times n$.

Initialisation d'un tableau

Pour initialiser un tableau à 2 dimensions (c'est à dire une matrice) de taille $n \times p$, on peut utiliser `np.zeros( (n, p) )`:

```
In [13]: np.zeros( (3, 5) )
Out[13]:
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

Question

Un entier n étant défini, créer une matrice identité de taille $n \times n$.

Question

Écrire une fonction ayant une matrice en argument et renvoyant sa transposée.

Opérations sur les tableaux de dimension ≥ 2

On peut faire les même opérations que sur les tableaux, terme à terme:

```
In [52]: M = np.array([ [0, 1], [2, 3] ])

In [53]: M * M
Out[53]:
array([[0, 1],
       [4, 9]])

In [54]: M + 1
Out[54]:
array([[1, 2],
       [3, 4]])
```

Attention: `M * M` ne correspond pas à la multiplication des matrices!

Opérations sur les tableaux de dimension ≥ 2

On peut faire les même opérations que sur les tableaux, terme à terme:

```
In [52]: M = np.array([ [0, 1], [2, 3] ])

In [53]: M * M
Out[53]:
array([[0, 1],
       [4, 9]])

In [54]: M + 1
Out[54]:
array([[1, 2],
       [3, 4]])
```

Attention: `M * M` ne correspond pas à la multiplication des matrices!

Question

Écrire une fonction `produit` ayant deux matrices en argument et renvoyant leur produit.

Opérations sur les tableaux de dimension ≥ 2

La multiplication des matrices peut aussi être obtenue avec `np.dot`:

```
In [56]: M = np.array([ [0, 1], [2, 3] ])

In [57]: np.dot(M, M)
Out[57]:
array([[ 2,  3],
       [ 6, 11]])
```

Ce qui correspond à:

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^2 = \begin{pmatrix} 2 & 3 \\ 6 & 11 \end{pmatrix}$$

Opérations sur les tableaux de dimension ≥ 2

La multiplication des matrices peut aussi être obtenue avec `np.dot`:

```
In [56]: M = np.array([ [0, 1], [2, 3] ])

In [57]: np.dot(M, M)
Out[57]:
array([[ 2,  3],
       [ 6, 11]])
```

Ce qui correspond à:

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^2 = \begin{pmatrix} 2 & 3 \\ 6 & 11 \end{pmatrix}$$

Question

Écrire une fonction `puissance(M, k)` renvoyant M^k.