

Entrée-sortie et indexation avec Numpy

Entrée-sortie
Indexation

Programmation Python Partie 2 : Entrée-Sortie et indexation avec NumPy

Fichiers - Entrée/Sortie

Fichiers séparés par des virgules (CSV)

Un format fichier classique est le format CSV (comma-separated values), ou bien TSV (tab-separated values). Pour lire de tels fichiers utilisez `numpy.genfromtxt`.

Par exemple:

```
In [61]: !cat data.csv
```

```
1,2,3,4,5  
6,7,8,9,10  
1,3,3,4,6  
1,2,3,4,20
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie

Indexation

```
In [62]: data = np.genfromtxt('data.csv', delimiter=',')  
print(data)
```

```
[[ 1.  2.  3.  4.  5.]  
 [ 6.  7.  8.  9. 10.]  
 [ 1.  3.  3.  4.  6.]  
 [ 1.  2.  3.  4. 20.]]
```

```
In [63]: print(data.shape)
```

```
(4, 5)
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

A l'aide de `numpy.savetxt` on peut enregistrer un *array* numpy dans un fichier txt:

```
In [64]: M = random.standard_normal((3, 3))  
print(M)
```

```
[[-2.3700268  -0.24567366 -2.33387944]  
 [ 0.07599637  2.04665479  0.99520716]  
 [ 1.89936908 -0.48671053  0.55652171]]
```

```
In [65]: np.savetxt("random-data.csv", M,  
                    fmt='% 2.3f', delimiter=',')  
!cat random-data.csv
```

```
-2.370, -0.246, -2.334  
0.076,  2.047,  0.995  
1.899, -0.487,  0.557
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie

Indexation

Format de fichier Numpy natif

Pour sauvegarder et recharger des tableaux numpy : `np.save` et `np.load`:

```
In [66]: np.save("random-matrix.npy", M)
```

```
In [67]: np.load("random-matrix.npy")
```

```
Out [67]: array([[ -2.3700268 , -0.24567366, -2.33387944],  
                 [  0.07599637,  2.04665479,  0.99520716],  
                 [  1.89936908, -0.48671053,  0.55652171]])
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

Indexation avancée

```
In [68]: v = np.array([1, 3, 2, 4])
```

```
In [69]: M = np.array([[1, 3], [2, 4]])
```

Accès à un élément

```
In [70]: # v ndarray à 1 dimension -> un indice  
print(v[0])
```

1

```
In [71]: # M ndarray à 2 dimensions -> deux indices  
print(M[0, 1])
```

3

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

Accès à plus d'un élément

```
In [72]: print(M[0]) # La première ligne
```

```
[1 3]
```

On peut aussi utiliser :

```
In [73]: print(M[1, :]) # 2 ème ligne (indice 1)
```

```
[2 4]
```

```
In [74]: print(M[:, 1]) # 2 ème colonne (indice 1)
```

```
[3 4]
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

Assignement de plus d'un élément

```
In [75]: print(M)
```

```
[[1 3]
 [2 4]]
```

```
In [76]: M[1, :] = -1 # assigne d'un élément à toute une ligne
print(M)
```

```
[[ 1  3]
 [-1 -1]]
```

```
In [77]: M[:, 1] = [5, 6] # assignement de plusieurs éléments
print(M)
```

```
[[ 1  5]
 [-1  6]]
```


Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

Slicing

Slicing fait référence à la syntaxe `M[start:stop:step]` pour extraire une partie d'un *array*:

```
In [78]: A = np.array([1, 2, 3, 4, 5])  
         print(A)
```

```
[1 2 3 4 5]
```

```
In [79]: print(A[1:3])
```

```
[2 3]
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

On peut omettre n'importe lequel des arguments dans
`M[start:stop:step]:`

```
In [80]: print(A[:])
```

```
[1 2 3 4 5]
```

```
In [81]: print(A[:,2]) # step = 2
```

```
[1 3 5]
```

```
In [82]: print(A[:3]) # les trois premiers éléments
```

```
[1 2 3]
```

```
In [83]: print(A[3:]) # à partir de l'indice 3
```

```
[4 5]
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

On peut utiliser des indices négatifs :

```
In [84]: A = np.array([1, 2, 3, 4, 5])
```

```
In [85]: print(A[-1]) # le dernier élément
```

5

```
In [86]: print(A[-3:]) # les 3 derniers éléments
```

[3 4 5]

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

slicing sur les tableaux multi-dimensionnels

```
In [87]: A = np.array([[n + m * 10 for n in range(4)]  
                        for m in range(4)])  
A
```

```
Out[87]: array([[ 0,  1,  2,  3],  
                [10, 11, 12, 13],  
                [20, 21, 22, 23],  
                [30, 31, 32, 33]])
```

```
In [88]: # un bloc du tableau  
print(A[1:3, 1:3])
```

```
[[11 12]  
 [21 22]]
```

```
In [89]: print(A[:,2, ::2]) # 1 ligne sur 2 et 1 colonne sur 2
```

```
[[ 0 2]  
 [20 22]]
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

Indexation avancée (*fancy indexing*)

Utilisation de listes ou de tableaux pour accéder à plusieurs éléments.

```
In [90]: print(A)
```

```
[[ 0  1  2  3]
 [10 11 12 13]
 [20 21 22 23]
 [30 31 32 33]]
```

```
In [91]: indices = [1, 3]
print(A[indices])  # lignes d'indices 1 et 3
```

```
[[10 11 12 13]
 [30 31 32 33]]
```

```
In [92]: indices = [1, 3]
print(A[:, indices])  # colonnes d'indices 1 et 3
```

```
[[ 1  3]
 [11 13]
 [21 23]
 [31 33]]
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

```
In [93]: # éléments d'indices [0,1] et [2,3]  
print(A[[0, 2], [1, 3]])
```

```
[ 1 23]
```

```
In [94]: # sous-tableau lignes [0,1] et colonnes [2,3]  
print A[np.ix_([0, 2], [1, 3])]
```

```
[[ 1  3]  
 [21 23]]
```

Entrée-sortie et indexation avec Numpy

Entrée-sortie Indexation

Indexation avec masques booléens

```
In [95]: v = np.arange(5)
         print(v)
```

```
[0 1 2 3 4]
```

```
In [96]: mask = np.array([True, False, True, False, False])
         print(v[mask])
         print(v[[0, 2]])
```

```
[0 2]
```

```
[0 2]
```

```
In [97]: print(v > 2)
         print(v[v > 2])
```

```
[False False False  True  True]
```

```
[3 4]
```