COURS N°3 PYTHON

L2 MATHÉMATIQUES

Matrices et vecteurs

Vue pour impression

Auteur : Antoine Laurent

POURQUOI NUMPY ?

- Numpy est un module Python pour le calcul scientifique
- Il est écrit en C
- Il permet de faire des opérations rapides sur les vecteurs et les matrices

Importer un module en utilisant un alias

```
>>> import numpy as np
>>> import numpy.random as npr
```

TABLEAUX (ARRAYS)

- Dans numpy, on travaille avec des **arrays**

```
>>> lst = [3, 2, 4, 1]
>>> a = np.array(lst) # convertir un liste en tableau numpy
>>> a
array([3, 2, 4, 1])
```

- Tous les éléments sont du même type

```
>>> a.dtype.name
'int64'
```

LES VECTEURS

- Les vecteurs sont des **arrays** de dimension 1

```
>>> np.zeros(5)
array([ 0., 0., 0., 0., 0.])
>>> np.ones(4)
array([ 1., 1., 1., 1.])
>>> npr.randn(2)  #normal distribution, moyenne 0 variance 1
array([-0.09454536,  0.0338529 ])
>>> np.linspace(0,1,5) #5 valeurs uniformes entre 0 et 1
array([ 0. , 0.25, 0.5 , 0.75, 1. ])
```

LES MATRICES

- Les matrices sont des **arrays** de dimension 2

```
>>> np.zeros((2,5))
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> npr.randn(3, 3)
array([[ 0.48763503, -2.03086611,  1.24233348],
       [ 0.38491276,  0.54755702,  0.40538603],
          [-0.86832433, -1.58760882,  0.21661244]])
```

LES MATRICES

- Matrices spéciales

```
>>> np.ones((3,5), np.int) #par défaut le type est numpy.float64
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
>>> np.zeros((3,5), np.bool)
array([[False, False, False, False, False],
       [False, False, False, False, False],
       [False, False, False, False, False]], dtype=bool)
>>> np.eye(3) #matrice identité
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

LES MATRICES

- Matrices spéciales

```
>>> x = np.arange(9).reshape((3,3))
>>> x
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.diag(x)
array([0, 4, 8])
>>> np.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

ATTRIBUTS D'UN TABLEAU (1/2)

- Les tableaux numpy sont des objets qui possèdent des attributs (propriétés de ces objets)

```
>>> a = np.array( [1, 2, 3])
>>> a.size #nombre d'éléments d'un tableau
3

>>> a.dtype #type commun des éléments du tab
dtype('int64')

>>> a.ndim # nombre de dimensions du tableau
1

>>> a.shape # donne le tuple du format du tableau (lignes, colonnes, ...)
(3,)
```

ATTRIBUTS D'UN TABLEAU (2/2)

- Les tableaux numpy sont des objets qui possèdent des attributs (propriétés de ces objets)

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

>>> a.size # nombre d'éléments ?
```

```
6
```

```
>>> a.shape # tuple (lignes, colonnes) ?
```

```
(2,3)
```

```
>>> a.ndim # nombre de dimensions ?
```

```
2
```

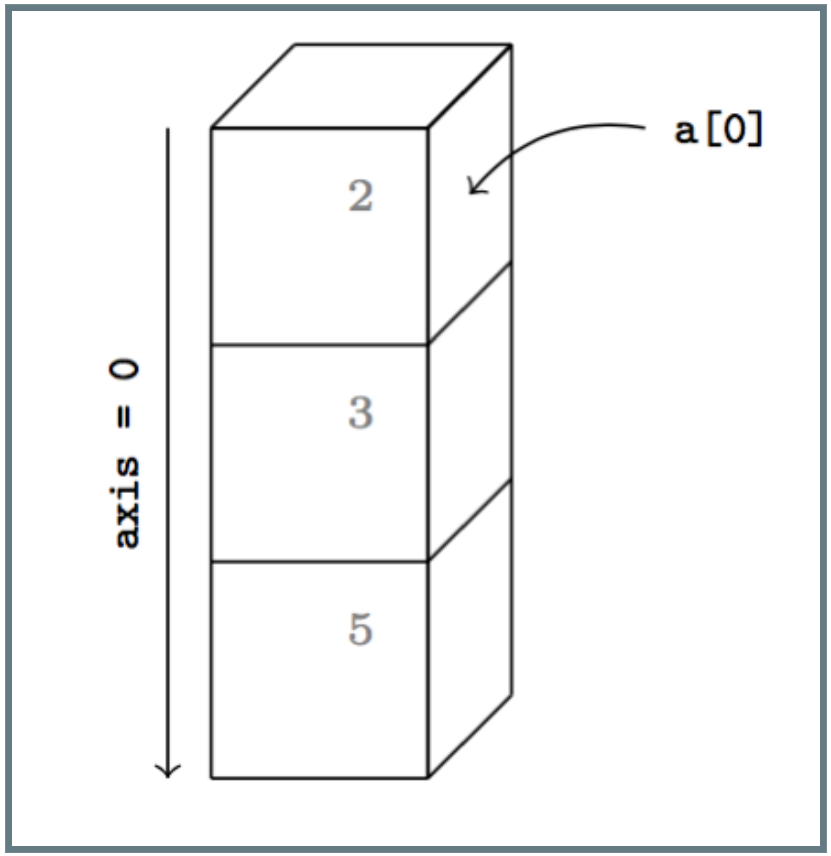
ARRAY SHAPE

- La taille des tableaux (array shape) peut être lue et modifiée

```
>>> z = np.arange(9)
>>> z
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> z.shape
(9,)
>>> z.reshape(3,3) # ou z.shape = (3,3)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> z.reshape(3,3).shape
(3, 3)
```

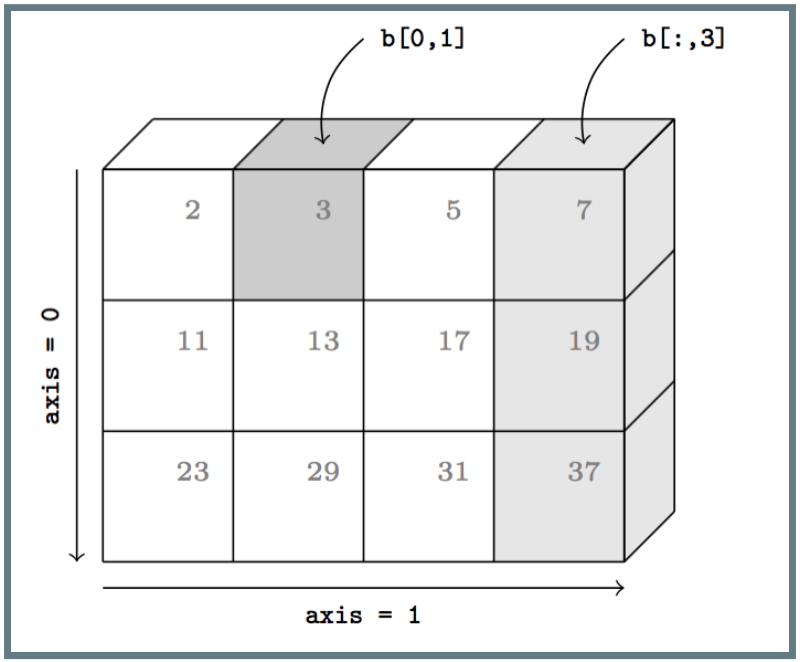
INDEX ET DÉCOUPAGE, SLICING

```
>>> a = np.array([2,3,5])
>>> a[0]
2
```



INDEX ET DÉCOUPAGE, SLICING

```
>>> b = np.array([ [2,3,5,7], [11, 13, 17, 19], [23, 29, 31, 37] ])
>>> b[0,1]
3
>>> b[:,3] #toutes les lignes, colonne 3
array([ 7, 19, 37])
```



INDEX ET DÉCOUPAGE, SLICING

```
>>> A
array([[0, 1, 2],
       [3, 4, 5]])
>>> A[1,2]
5

>>> A[1,:] # ligne 1, toutes les colonnes
array([3, 4, 5])

>>> A[:, 0] # toutes les lignes, colonne 0
array([0, 3])

>>> A[:,[0,2]] # toutes les lignes, colonnes 0 et 2
array([[0, 2],
       [3, 5]])
```

INDEX ET DÉCOUPAGE, MASQUE (1)

- Si **a** est un tableau **numpy** et **b** un tableau de booléens de même format, alors a[b] met en relation un à un les éléments de **a** avec ceux de **b**
- Seuls les éléments de **a** associés à True sont conservés

```
>>> a = np.arange(10) # equivalent : np.array(range(10))
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.shape = (2, 5) # equivalent : a=a.reshape(2,5)
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])

>>> b = np.array( [[True, False, False, True, False],
    [False, True, False, False, True]])
>>> a[b]
array([0, 3, 6, 9])
```

INDEX ET DÉCOUPAGE, MASQUE (2)

```
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> c = a%3 == 0
>>> c
array([[ True, False, False,  True, False],
       [False,  True, False, False,  True]], dtype=bool)

>>> a[c] # eq. a[a%3==0]
array([0, 3, 6, 9])
```

INDEX ET DÉCOUPAGE, MASQUE (3)

```
>>> A
array([[0, 1, 2],
       [3, 4, 5]])

>>> mask = A.min(axis=0) > 1 # tres utile

>>> mask
array([False, False,  True], dtype=bool)

>>> A[:, mask] #toutes les lignes, True
array([[2],
       [5]])
```

ECRITURE DANS UN TABLEAU

- Les objets **numpy** sont mutables, on peut modifier les valeurs qu'ils contiennent sans redéfinir l'objet
- On peut modifier simultanément plusieurs valeurs en utilisant les mêmes syntaxes que pour le **slicing**

```
>>> a = np.arange(10).reshape(2,5)
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a[0,0] = 7
array([[7, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a[0,:] = [8, 7, 6, 5, 4]
array([[8, 7, 6, 5, 4],
       [5, 6, 7, 8, 9]])
>>> a[ a%2 == 0 ] += 1
array([[9, 7, 7, 5, 5],
       [5, 7, 7, 9, 9]])
```

COPIES OU RÉFÉRENCE ?

- En python, si a est un objets, b = a ne va pas recopier l'objet a dans b mais faire de b une nouvelle référence vers l'objet a

```
>>> a = np.arange(10).reshape(2,5)
>>> b = a
>>> id(a), id(b) # adresse mémoire de l'objet sur lequel pointent a et b
(4515994640, 4515994640)
>>> a[0,0] = 1
>>> a
array([[1, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> b
array([[1, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

COPIES OU RÉFÉRENCE ?

- Attention ...

```
>>> a = np.arange(10).reshape(2,5)
>>> b = a[:,:]
>>> id(a), id(b)
(4515906464, 4515909104)  #semble bien ... et pourtant
>>> b.fill(1)
>>> b
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
>>> a
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

COPIES OU RÉFÉRENCE ?

- Meme chose avec un array (pas numpy array)

```
>>> a = [i for i in range(10)]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> b = a[:]
>>> id(a), id(b)
(4517303496, 4517393928)
>>> b[3:6]=[-1,-1,-1]
>>> b
[0, 1, 2, -1, -1, -1, 6, 7, 8, 9]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Cette fois le tableau a été copié ...

COPIES OU RÉFÉRENCE ?

- Références (vues) partielles ...

```
>>> a = np.zeros((2,5),dtype=np.int64)
>>> a
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
>>> b = a[1, 1:3]
>>> b
array([0, 0])
>>> b[:]=[1,1]
>>> a
array([[0, 0, 0, 0, 0],
       [0, 1, 1, 0, 0]])
```

- b est un vue partielle de a ...
- pour copier un objet numpy, on utilise **copy**

COPIES OU RÉFÉRENCE ?

- Utilisation de copy

```
>>> a = np.zeros((2,5),dtype=np.int64)
>>> a
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
>>> b = a[1, 1:3].copy()
>>> b
array([0, 0])
>>> b[:]=[1,1]
>>> a
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
>>> c = a.copy()
```

OPÉRATION MATHÉMATIQUES (1/3)

- Les opérateurs arithmétiques s'appliquent sur chaque élément

```
>>> a = np.arange(6).reshape(2, 3)
array([[0, 1, 2],
       [3, 4, 5]])
>>> b = np.ones((2,3))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> a+b
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
>>> a * b
array([[0, 1, 2],
       [3, 4, 5]])
>>> a < 3
array([[ True,  True,  True],
       [False, False, False]], dtype=bool)
```

OPÉRATIONS MATHÉMATIQUES (2/3)

- Très pratique !

```
>>> x = np.arange(4).reshape(2, 2)
array([[0, 1],
       [2, 3]])
>>> np.sin(x) * np.exp(-x ** 2 / 2)
array([[ 0.        ,  0.51037795],
       [ 0.12306002,  0.0015677 ]])
```

OPÉRATIONS MATHÉMATIQUES (3/3)

- On peut définir nos propres fonctions

```
>>> def f(x):
      k, s = 0, 0
      while k<x:
        k += 1
        s += k
      return s

>>> f(3) #3+2+1
6

>>> x = np.arange(1,10,2) #array de 1 à 10 avec pas de 2
array([1, 3, 5, 7, 9])

>>> vf = np.vectorize(f)
>>> vf(x)
array([ 1,  6, 15, 28, 45])
```

MÉTHODES

- Par défaut s'appliquent sur le tableau, comme s'il s'agissait d'une liste de nombres

```
>>> a = np.array([[0, 9, 2], [3, 4, 1]])
array([[0, 9, 2],
       [3, 4, 1]])
>>> a.min(), a.argmax(), a.sum()  #argmax indice element le plus grand
(0, 1, 19)
```

- On peut aussi spécifier un **axe**

```
>>> a.sum(axis=0) #colonne
array([ 3, 13,  3])
>>> a.sum(axis=1) #ligne
array([11,  8])
>>> a.sort(axis=0) #va modifier a !!
array([[0, 4, 1],
       [3, 9, 2]])
```

ACCOLER DES MATRICES

```
>>> a = (np.arange(9)+1).reshape(3,3)
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> b = np.array([[0,1,0],[1,0,0],[0,0,1]])
>>> np.concatenate((a,b),axis=0)
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9],
       [0, 1, 0],
       [1, 0, 0],
       [0, 0, 1]])
>>> np.concatenate((a,b),axis=1)
array([[1, 2, 3, 0, 1, 0],
       [4, 5, 6, 1, 0, 0],
       [7, 8, 9, 0, 0, 1]])
```

ACCOLER DES MATRICES

```
>>> v1 = np.array([1,2,3])
>>> v2 = np.array([4,5,6])
>>> v3 = np.array([1,0,0])
>>> np.column_stack((v1,v2,v3))
array([[1, 4, 1],
       [2, 5, 0],
       [3, 6, 0]])
```

OPÉRATIONS SUR LES MATRICES

```
>>> A = np.arange(6).reshape(2, 3)
array([[0, 1, 2],
       [3, 4, 5]])
>>> x = 2 * np.ones(3)
array([ 2.,  2.,  2.])
>>> A.dot(x)  #Ax (0*2+1*2+2*2 3*2+4*2+5*2)
array([  6.,  24.])
>>> A.T #ou A.transpose()
array([[0, 3],
       [1, 4],
       [2, 5]])
>>> A.dot(A.T) #AA'
array([[ 5, 14],
       [14, 50]])
```

SCLICING AVANCÉ

- Dans l'instruction [début:fin:pas], 2 arguments peuvent être omis. début vaut 0 par défaut, pas vaut 1 par défaut

```
>>> a = np.arange(10)*2
[ 0  2  4  6  8 10 12 14 16 18]
>>> a[2:9] # début:fin+1 [:pas]
[ 4  6  8 10 12 14 16]
>>> a[2:9:3] # début:fin+1 [:pas]
[ 4 10 16]
>>> a[:5] #de 0 à 4
[0 2 4 6 8]
```

SCLICING AVANCÉ

```
>>> a=np.eye(5,5)
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
>>> a[:3, :4]=4 #lignes => de 0 à 2 et colonnes de 0 à 3 = 4
[[ 4.  4.  4.  4.  0.]
 [ 4.  4.  4.  4.  0.]
 [ 4.  4.  4.  4.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
```

SCLICING AVANCÉ

```
>>> a[::3,::4]=5 #en partant de 0, toutes les 3 lignes un 5
                 #en partant de 0, toutes les 4 colonnes un 5
[[ 5.  4.  4.  4.  5.]
 [ 4.  4.  4.  4.  0.]
 [ 4.  4.  4.  4.  0.]
 [ 5.  0.  0.  1.  5.]
 [ 0.  0.  0.  0.  1.]]
>>> a[:,3]=6 #sur toutes les lignes, colonne 3 => 6
[[ 5.  4.  4.  6.  5.]
 [ 4.  4.  4.  6.  0.]
 [ 4.  4.  4.  6.  0.]
 [ 5.  0.  0.  6.  5.]
 [ 0.  0.  0.  6.  1.]]
```

AUTRE MÉTHODE D'EXTRACTION :

- a[b] où a est un vecteur et b un tableau d'entiers correspondant aux indices à extraire
- a[b,c] où a est une matrice, b contient la liste des indices de lignes et c contient la liste des indices de colonnes

```
>>> a = np.array([2,4,6,8],float)
[ 2.  4.  6.  8.]
>>> b=np.array([0,0,1,3,2,1],int)
[0 0 1 3 2 1]
>>> a[b]
[ 2.  2.  4.  8.  6.  4.]
```

AUTRE MÉTHODE D'EXTRACTION :

- a[b] où a est un vecteur et b un tableau d'entiers correspondant aux indices à extraire
- a[b,c] où a est une matrice, b contient la liste des indices de lignes et c contient la liste des indices de colonnes

```
>>> a=a.reshape(2,2)
[[ 2.  4.]
 [ 6.  8.]]
>>> b=np.array([0,0,1,1,0],int)
[0 0 1 1 0]
>>> c=np.array([0,1,1,1,1],int)
[0 1 1 1 1]
>>> a[b,c]
[ 2.  4.  8.  8.  4.]
```

NUMPY TAKE

- np.take permet d'extraire des lignes ou des colonne

```
>>> a=np.arange(16).reshape(4,4)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
>>> a.take([0,3],axis=1) #colonnes 0 et 3
[[ 0  3]
 [ 4  7]
 [ 8 11]
 [12 15]]
>>> a.take([0,3],axis=0) #lignes 0 et 3
[[ 0  1  2  3]
 [12 13 14 15]]
```

NUMPY TAKE

- np.take permet d'extraire des lignes ou des colonne

```
>>> a=np.arange(16).reshape(4,4)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
>>> a.take([0,3,2,2,2],axis=0)
[[ 0  1  2  3]
 [12 13 14 15]
 [ 8  9 10 11]
 [ 8  9 10 11]
 [ 8  9 10 11]]
```

NUMPY PUT

- np.put permet de remplacer des valeurs dans un numpy array

```
>>> a=np.array([0, 1, 2, 3, 4, 5],float)
[ 0.  1.  2.  3.  4.  5.]
>>> b=np.array([9,8,7],float)
[ 9.  8.  7.]
>>> a.put([0,1,3], b)
[ 9.  8.  2.  7.  4.  5.] #contenu de b en 0, 1, 3
```

NUMPY PUTMASK

- np.putmask permet de remplacer des valeurs dans un numpy array en fonction d'un masque

```
>>> x=np.arange(6, dtype=float).reshape(2,3)
[[ 0. 1. 2.]
 [ 3. 4. 5.]]
 >>> np.putmask(x, x>2, x**2)
 [[ 0. 1. 2.]
  [ 9. 16. 25.]]
```

LE TYPE MAT

- Il permet de construire des numpy array de dimension 2 avec des "raccourcis"
- On peut calculer le produit matriciel directement avec le symbole "*" entre deux matrices

```
>>> a=np.mat('[1 2 4 ; 3 4 5]')
[[1 2 4]
 [3 4 5]]
>>> b=np.mat('[2. ; 4 ; 6]')
[[ 2.]
 [ 4.]
 [ 6.]]
>>> a*b
[[ 34.]
 [ 52.]]
>>> a = np.array([1,2,4])
>>> np.mat(a) # convertir en matrice
matrix([[1, 2, 4]])
```

FLIP LEFT/RIGHT

```
>>> a = np.arange(9).reshape(3,3)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> np.fliplr(a)
array([[2, 1, 0],
       [5, 4, 3],
       [8, 7, 6]])
>>> np.eye(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> np.fliplr(np.eye(3))
array([[ 0.,  0.,  1.],
       [ 0.,  1.,  0.],
       [ 1.,  0.,  0.]])
```

STATISTIQUES

```
>>> A
array([[0, 1, 2],
       [3, 4, 5]])

>>> A.mean()
2.5

>>> A.std(axis=0) #deviation standard / ecart type
array([ 1.5,  1.5,  1.5])

>>>  np.median(A, axis=1) #elem median chaque ligne
array([ 1.,  4.])
```

TIRAGES ALÉATOIRES (1/2)

```
>>> npr.seed(42) #utile si on veut reproduire, meme sur autre machine

>>> npr.rand(3, 2) # random in [0,1]
array([[ 0.37454012,  0.95071431],
       [ 0.73199394,  0.59865848],
       [ 0.15601864,  0.15599452]])

>>> npr.randint(0, 10, size=(2,5)) #tirage uniforme d'entiers dans [0,10[
array([[4, 5, 3, 3, 6],
       [7, 5, 4, 7, 0]])
```

TIRAGES ALÉATOIRES (2/3)

```
>>> npr.binomial(10, 0.3, (2,5)) #simule (2,5) valeurs d'une variable
                                 #aléatoire suivant une binomiale(10,0.3)

>>> npr.geometric(0.3, 5)
array([2, 3, 1, 1, 5])

>>> npr.poisson(4.3, 5)
array([1, 7, 5, 4, 5])

>>> dir(npr)
```

TIRAGES ALÉATOIRES (3/3)

```
>>> letters = np.array(['a', 'b', 'c', 'd'])
>>> npr.shuffle(letters)
>>> letters
array(['c', 'd', 'a', 'b'],
      dtype='<U1') #little-endian Unicode avec 1 caractere
>>> npr.choice(letters, 2) #tirage aléatoire de 2 elements
array(['a', 'd'],
      dtype='<U1')
```

LECTURE DES VALEURS DANS UN FICHIER

- Si des données sont stockées dans un fichier texte avec délimiteur, on peut utiliser la fonction genfromtxt

```
#Exemple : data.csv (toutes les lignes doivent avoir le meme nb de col)
1,2,3,4,5
6,0,0,7,8
0,0,9,10,11
```

```
>>> m = np.genfromtxt("data.csv", delimiter=",")
>>> m
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   0.,   0.,   7.,   8.],
       [  0.,   0.,   9.,  10.,  11.]])
```