ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Langage Python

Lecture6: Numpy

Abderrahim MESBAH
a.mesbah@um5r.ac.ma

Plan

- Création des tableaux
- Attributs des tableaux
- Opérations sur les tableaux
- Indexation et découpage
- Algèbre matricielle
- Calcul matricielle
- Polynômes
- Calculs statistiques
- Autres

Modules et Packages

Modules

- Les modules sont des programmes Python qui contiennent des fonctions que l'on est amené à réutiliser souvent (on les appelle aussi bibliothèques ou librairies).
- Vous pouvez accéder aux modules disponibles sur le site de Python:

  https://docs.python.org/3/py-modindex.html

- Importation de modules:
  - L'instruction **import** donne accès à toutes les fonctions du module.

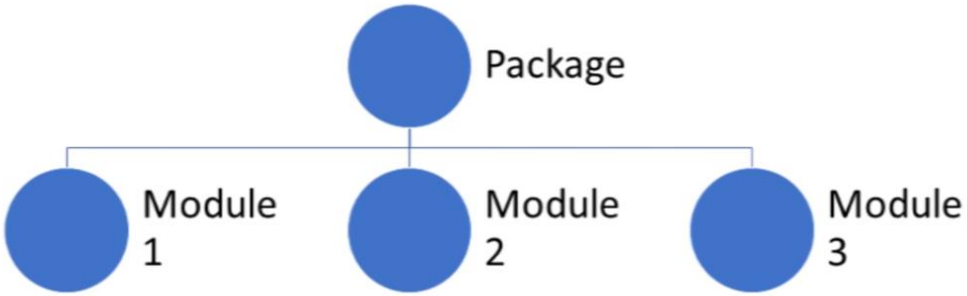
```
import random
a = random . randint (0, 10)
print (a)
```

```
from random import randint
a = randint (0, 10)
print (a)
```

```
import random as rd
a = rd. choice (['Chien', 'Chat', 'Cheval'])
print (a)
```

Packages

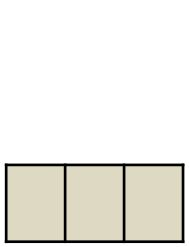
- Un package est un module contenant d'autres modules. Les modules d'un package peuvent être des sous-packages, ce qui donne une structure arborescente.



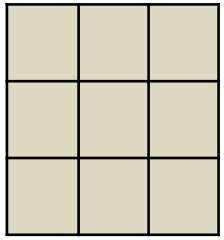
Création des tableaux

Définition

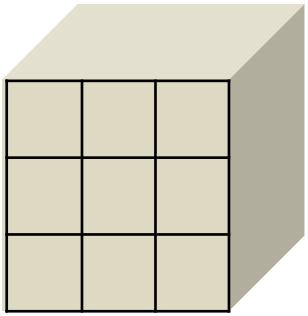
- **Numpy, Scipy et Matplotlib fournissent des fonctionnalités de type MATLAB en python.**
- **Caractéristiques Numpy:**
  - Tableaux multidimentionnels typés (matrices) : ND ARRAY
  - Calculs numériques rapides (mathématiques matricielles)
  - Fonctions mathématiques de haut niveau



1D array

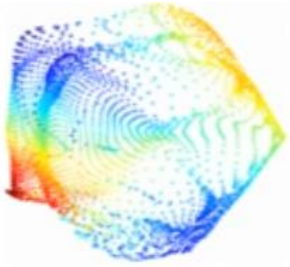


2D array



3D array

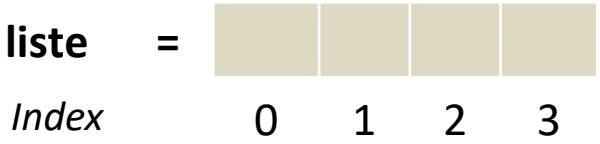
.....

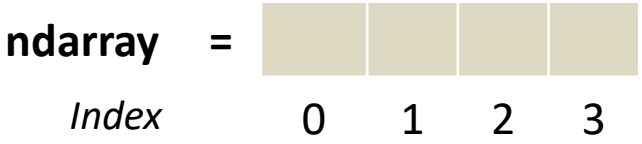


ND array

Création des tableaux

- **1D ARRAY.**



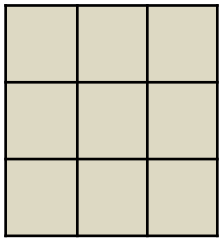


ndarray est beaucoup plus ***rapide*** et meilleur aux ***calculs scientifiques***

Création des tableaux

- **2D ARRAY.**

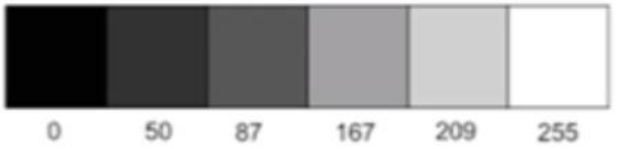
	A	B	C	D	
1	sepal.length	sepal.width	petal.length	petal.width	variet
2	5.1	3.5	1.4	0.2	Setosa
3	4.9	3	1.4	0.2	Setosa
4	4.7	3.2	1.3	0.2	Setosa
5	4.6	3.1	1.5	0.2	Setosa
6	5	3.6	1.4	0.2	Setosa
7	5.4	3.9	1.7	0.4	Setosa
8	4.6	3.4	1.4	0.3	Setosa
9	5	3.4	1.5	0.2	Setosa
10	4.4	2.9	1.4	0.2	Setosa
11	4.9	3.1	1.5	0.1	Setosa
12	5.4	3.7	1.5	0.2	Setosa
13	4.8	3.4	1.6	0.2	Setosa
14	4.8	3	1.4	0.1	Setosa
15	4.3	3	1.1	0.1	Setosa
16	5.8	4	1.2	0.2	Setosa
17	5.7	4.4	1.5	0.4	Setosa
18	5.4	3.9	1.3	0.4	Setosa
19	5.1	3.5	1.4	0.3	Setosa
20	5.7	3.8	1.7	0.3	Setosa
21	5.1	3.8	1.5	0.3	Setosa
22	5.4	3.4	1.7	0.2	Setosa
23	5.1	3.7	1.5	0.4	Setosa

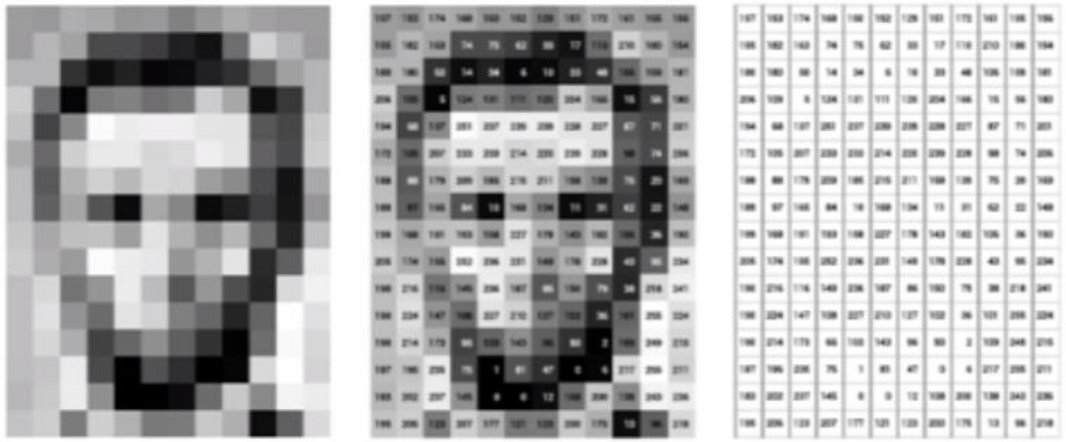


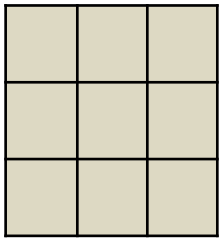
2D array

Création des tableaux

- **2D ARRAY.**



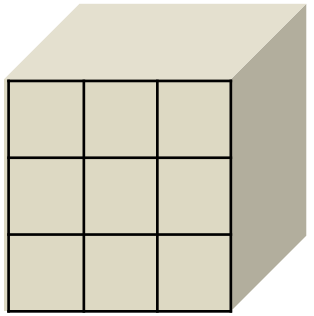




2D array

Création des tableaux

- **3D ARRAY.**



3D array

Création des tableaux

- **Propriétés de base.**

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
print (a.ndim, a.shape, a.dtype)
```

  - Les tableaux numpy peuvent avoir un nombre quelconque de dimensions.
  - les tableaux numpy ont un type associé : np.uint8, np.int64, np.float32, np.float64
  - Chaque élément du tableau a le même type. On ne pourra pas stocker dans un tableau numpy des données de types hétérogènes comme les listes.

Création des tableaux

- **np.zeros((n,m,l,…k)): tableau de 0 de dimension n x m x…k**

```
import numpy as np
a=np.zeros((2,3),dtype=np.float32)
print(a)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

- **np.ones((n,m,l,…k)): tableau de 1 de dimension n x m x…k**

```
import numpy as np
a=np.ones((2,3),dtype=np.float32)
print(a)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

Création des tableaux

- **ny.arange([start,]stop, [step,], dtype=None): tableau à valeurs séquentielles de start à stop avec un pas step.**

```
import numpy as np
a = np.arange(start=1, stop=10, step=3)
print (a)
```

```
[1 4 7]
```

```
import numpy as np
a = np.arange(1,5)
print (a)
```

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

```
import numpy as np
a = np.arange(1,5,0.5)
print (a)
```

```
[1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```

Création des tableaux

- **np.arange([start,]stop, [step,], dtype=None)**

```
>>> np.arange(start=0, stop=10, step=1)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.arange(0, 10, 1)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.arange(start=0, stop=10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

 >>> np.arange(0, 10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>>  np.arange(5)
array([0, 1, 2, 3, 4])

>>> np.arange(5.0)
array([0., 1., 2., 3., 4.])
```

Création des tableaux

- **np.arange([start,]stop, [step,], dtype=None)**

```
>>> np.arange(-5, -1)
array([-5, -4, -3, -2])

>>> np.arange(-8, -2, 2)
array([-8, -6, -4])

>>> np.arange(-5, 6, 4)
array([-5, -1, 3])

>>> np.arange(5, 1, -1)
array([5, 4, 3, 2])

>>> np.arange(7, 0, -3)
array([7, 4, 1])
```

Création des tableaux

- **np.random.randint(start, stop, size=n): tableau a valeurs aléatoires entre start et stop de dimension n**

```
import numpy as np
a = np.random.randint(0, 100, size=4)
b = np.random.randint(100, size=(2,3))
c = np.random.randint(3, 10, size=10)
d = np.random.randint(3, 10)
print(a)
print(b)
print(c)
print(d)
```

```
[ 0 62 64 47]
[[75 91 17]
 [13 82  8]]
[9 9 7 7 3 5 8 4 8 7]
9
```

Création des tableaux

- **np.random.randn(n): tableau à n valeurs aléatoires échantillonnés à partir d'une distribution normale standard.**

```
import numpy as np

a = np.random.randn(4)
b = np.random.randn(3, 4)
print(a)
print(b)
```

```
[ 1.4671901   1.27630519 -0.5857987  -1.06770789] [13 82  8]]

[[-0.08933539 -1.58143111 -0.21976241  0.07736355]
 [ 1.4844477   0.1284122  -0.14764632 -0.46930852]
 [ 0.35431424 -2.53793377 -0.2300015   0.92261706]]
```

Attributs des tableaux

Attributs des tableaux

- **array. shape: taille d'un tableau**

```
>>> a = np.arange(6).reshape((2, 3))
>>> a
 array([[0, 1, 2], [3, 4, 5]])
>>> a.shape
(2, 3)
```

- **array. ndim: dimension d'un tableau**

```
import numpy as np
a = np.arange(36)
b=a.reshape((6, 6))
c=a.reshape((3, 3,4))
print(a.ndim)
print(b.ndim)
print(c.ndim)
```

```
1
2
3
```

Attributs des tableaux

- **array. dtype: type d'un tableau**

```
import numpy as np
a = np.arange(3)
b=np.arange(3,dtype=np.float32)
print(a.dtype)
print(b.dtype)
```

```
Int32
float32
```

Attributs des tableaux

- **array. dtype: type d'un tableau**

np.int8: 8-bit signed integer (from -128 to 127)

np.uint8: 8-bit unsigned integer (from 0 to 255)

np.int16: 16-bit signed integer (from -32768 to 32767)

np.uint16: 16-bit unsigned integer (from 0 to 65535)

*np.int32: 32-bit signed integer (from -2**31 to 2**31-1)*

*np.uint32: 32-bit unsigned integer (from 0 to 2**32-1)*

*np.int64: 64-bit signed integer (from -2**63 to 2**63-1)*

*np.uint64: 64-bit unsigned integer (from 0 to 2**64-1)*

Opérations sur les tableaux

Opérations sur les tableaux

- **np.reshape(shape): Redimensionnement d'un tableau**

```
>>> a = np.arange(6).reshape((2, 3))
>>> a
 array([[0, 1, 2], [3, 4, 5]])
>>> a.shape
(2, 3)
>>> a.ndim
2
```

```
>>> a = np.arange(6)
array([0, 1, 2, 3, 4, 5])
>>> a=a. reshape((2, 3))
 array([[0, 1, 2], [3, 4, 5]])
>>> a.shape
(2, 3)
>>> a.ndim
2
```

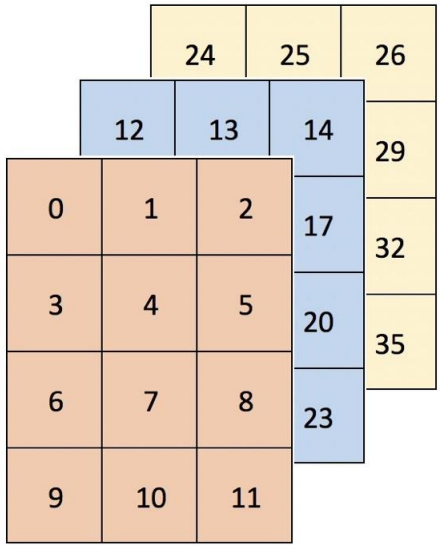
Opérations sur les tableaux

- **np.reshape(shape): Redimensionnement d'un tableau**

```
>>> import numpy as np
>>> arr = np.arange(36).reshape(3, 4, 3)
>>> arr
[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]
  [ 9 10 11]]

 [[12 13 14]
  [15 16 17]
  [18 19 20]
  [21 22 23]]

 [[24 25 26]
  [27 28 29]
  [30 31 32]
  [33 34 35]]]
```



Opérations sur les tableaux

- **np.reshape(shape): Redimensionnement d'un tableau**

```
>>> import numpy as np
>>> arr = np.arange(36).reshape(?, ?, ?)
>>> arr
[[[ 0  1  2  3  4  5]
  [ 6  7  8  9 10 11]
  [12 13 14 15 16 17]]

 [[18 19 20 21 22 23]
  [24 25 26 27 28 29]
  [30 31 32 33 34 35]]]
```

Opérations sur les tableaux

- **np.add(), np.subtract(), np.multiply(), np.divide()**

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)
print ('First array:')
print (a)

print ('Second array:')
b = np.array([10,10,10])
print (b)

print ('Add the two arrays:')
print (np.add(a,b) )

print ('Subtract the two arrays:')
print (np.subtract(a,b))

print ('Multiply the two arrays:')
print (np.multiply(a,b))

print ('Divide the two arrays:')
print (np.divide(a,b))
```

Opérations sur les tableaux

- **np.add(), np.subtract(), np.multiply(), np.divide()**

```
First array:[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
Second array:
[10 10 10]
Add the two arrays:
[[10. 11. 12.] [13. 14. 15.]
 [16. 17. 18.]]
Subtract the two arrays:
[[-10.  -9.  -8.]
 [ -7.  -6.  -5.]
 [ -4.  -3.  -2.]]
Multiply the two arrays:
[[ 0. 10. 20.]
 [30. 40. 50.]
 [60. 70. 80.]]
Divide the two arrays:
[[0.  0.1 0.2]
 [0.3 0.4 0.5]
 [0.6 0.7 0.8]]
```

Indexation et découpage

Indexation et découpage

- **A[i]: indexation**

```
import numpy as np
a = np.array([1, 2,3, 7])
print(a[0])
print(a[2])
```

```
1
3
```

Indexation et découpage

- **A[i:j]: découpage**

```
import numpy as np
a = np.array([[1, 2, 3],
[4, 5, 6],
[-7, 8, 9]])

print(a[2,:])
print(a[1:3,:])
```

```
[-7  8  9]

[[ 4  5  6]
 [-7  8  9]]
```

Algèbre matricielle

Algèbre matricielle

- **np.matmul(a,b)/ np.dot(a,b)/array.dot(array)/ operateur ***

```
import numpy as np
a = np.array([[1, 1],[2, 1]])
b = np.array([[4, 1],[2, 2]])
c=np.matmul(a, b)
d=a.dot(b)
e=np.dot(a,b)
f=a*b
print(c)
print(d)
print(e)
print(f)
```

```
[[ 6  3]
 [10  4]]

[[ 6  3]
 [10  4]]

[[ 6  3]
 [10  4]]

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

Algèbre matricielle

- **np.transpose(a)/ array.T**

```
import numpy as np
a = np.array([[1, 1],[2, 1]])
b=a.T
c=np.transpose(a)
print(b)
print(c)
```

```
[[1 2]
 [1 1]]

[[1 2]
 [1 1]]
```

Algèbre matricielle

- **np.linalg.inv(array): Calcul de l'inverse d'une matrice**

```
import numpy as np
a = np.array([[1, 2, 3],[4, 5, 6],[-7, 8, 9]])

b=np.linalg.inv(a)
print(b)
```

```
[[-0.07142857  0.14285714 -0.07142857]
 [-1.85714286  0.71428571  0.14285714]
 [ 1.5952381  -0.52380952 -0.07142857]]
```

Algèbre matricielle

- **np.linalg.det(array): Calcul du déterminant d'une matrice**

```
import numpy as np
a = np.array([[1, 2, 3],[4, 5, 6],[-7, 8, 9]])
b=np.linalg.det(a)
print(b)
```

```
41.999999999999986
```

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$= a(ei - fh) - b(di - fg) + c(dh - eg)$$

$$= aei - afh - bdi + bfg + cdh - ceg$$

ENSIAS

Polynômes

Polynômes

- La classe Polynomial du module numpy.polynomial.polynomial permet de travailler avec des polynômes.

```
from numpy.polynomial import Polynomial
```

Polynômes

- **Création d'un polynôme**

  Exemple: $\mathbf{X^3+2X-3}$

```
from numpy.polynomial import Polynomial
p=Polynomial([-3,2,0,1])
```

- **Calcul d'une valeur de la fonction polynôme associée**

```
from numpy.polynomial import Polynomial
p=Polynomial([-3,2,0,1])
print(p(0))
print(p(3))
```

```
-3.0
9.0
```

Polynômes

- **Attribut coef d'un polynôme: donne accès aux coefficients ordonnés par degré croissant**

Exemple: X^3+2X-3

```
from numpy.polynomial import Polynomial
p=Polynomial([-3,2,0,1])
p.coef
```

```
array([-3.,  2.,  0.,  1.])
```

- **Méthode degree(): Renvoie le degré d'un polynôme**

```
p.degree()
```

```
3
```

Polynômes

- **Méthode integ() : renvoie l'intègral d'un polynôme**

```
pi1=p.integ()
print(pi1.coef)

pi2=p.integ(1, 2) # intégrer une fois avec la
constante 2
print(pi2.coef)
pi3=p.integ(2, [1, 2]) # intégrer deux fois avec
deux constantes 1 et 2
print(pi3.coef)
```

```
array([ 0. , -3. , 1. , 0. , 0.25])
[ 2.  -3.   1.   0.   0.25]
[ 2.        1.       -1.5       0.33333333  0.        0.05     ]
```

ENSIAS

Calculs statistiques

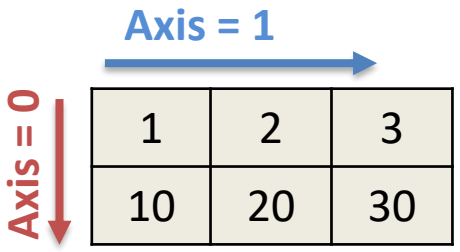
Calculs statistiques

- **np.sum(array, axis): Somme des éléments d'un tableau**

```
>>> arr = np.array([[1, 2, 3],  [10, 20, 30]])

>>> arr.sum(axis=0)
array([11, 22, 33])

>>> arr.sum(axis=1)
array([ 6, 60])
```



- **np.mean(array, axis): moyenne des éléments d'un tableau**

```
>>> arr = np.array([[1, 2, 3],  [10, 20, 30]])

>>> arr.mean(axis=0)
array([5.5, 11., 16.5])

>>> arr.mean(axis=1)
array([ 2, 20])
```

Calculs statistiques

- **np.var(array): Calcul de la variance d'une matrice**

```
import numpy as np
a = np.array([1, 2, 3, 4, 5, 6, -7, 8, 9])

b=np.var(a)
print(b)
```

```
19.80246913580247
```

- **np.std(array, axis = None) : Calcul l'écart type des données (éléments de array) le long de l'axe spécifié (le cas échéant).**

```
import numpy as np
a = [[2, 2, 2, 2, 2],
     [15, 6, 27, 8, 2],
     [23, 2, 54, 1, 2, ],
     [11, 44, 34, 7, 2]]
```

Calculs statistiques

- **np.std(array, axis = None) : Calcul l'écart type des données (éléments de array) le long de l'axe spécifié (le cas échéant).**

```
import numpy as np
arr = [[2, 2, 2, 2, 2],
       [15, 6, 27, 8, 2],
       [23, 2, 54, 1, 2, ],
       [11, 44, 34, 7, 2]]

# std of the flattened array
print("\nstd of arr, axis = None : ", np.std(arr))

# std along the axis = 0
print("\nstd of arr, axis = 0 : ", np.std(arr, axis = 0))

# std along the axis = 1
print("\nstd of arr, axis = 1 : ", np.std(arr, axis = 1))
```

```
std of arr, axis = None :  15.3668474320532
std of arr, axis = 0 :  [ 7.56224173 17.68473918 18.59267329  3.04138127  0. ]
std of arr, axis = 1 :  [ 0.         8.7772433  20.53874388 16.40243884]
```

Autres

Autres

- **np.power(a ,b)**

```
import numpy as np
a = np.array([10,100,1000])
print ('Our array is:')
print (a)

print ('Applying power function:')
print(np.power(a,2))

print ('Second array:')
b = np.array([1,2,3])
print (b)

print ('Applying power function again:')
print (np.power(a,b))
```

```
Our array is:
[  10  100 1000]

Applying power function:
[    100   10000 1000000]

Second array:
[1 2 3]

Applying power function again:
[       10      10000 1000000000]
```

Autres

- **np.around(array)**

```
import numpy as np
a = np.array([1.0,5.55, 123, 0.567, 25.532])
print('Original array:')
print (a)
print ('After rounding:')
print(np.around(a) )
print (np.around(a, decimals = 1) )
```

```
Original array:
[  1.     5.55  123.     0.567  25.532]
After rounding:
[  1.   6. 123.   1.  26.]
[  1.    5.6 123.    0.6  25.5]
```

Autres

- **np.floor(array)**

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])
print ('The given array:')
print (a)
print ('The modified array:')
print (np.floor(a))
```

```
The given array:
[-1.7  1.5 -0.2  0.6 10. ]
The modified array:
[-2.  1. -1.  0. 10.]
```

Autres

- **np.ceil(array)**

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])
print ('The given array:')
print (a)
print ('The modified array:')
print (np.ceil(a))
```

```
The given array:
[-1.7  1.5 -0.2  0.6 10. ]
The modified array:
[-1.  2. -0.  1. 10.]
```

Autres

- np.sin()
- np.cos()
- np.tanh()