Le module `numpy` : création d'un tableau

Le module `numpy` est un outil performant pour la manipulation de tableaux à plusieurs dimensions. Il ajoute en effet la classe `array` qui a des similarités avec les listes mais tous les éléments sont obligatoirement du même type : des entiers, des flotants ou des booléens.

Nous commençons par charger le module `numpy` et nous lui donnons un alias pour raccourcir son appel (l'alias `np` n'est pas obligatoire mais dans la pratique, tout le monde l'utilise).

In [1]:
```
help(numpy)
```

```
---------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
<ipython-input-1-4c3aefd14e91> in <module>
----> 1 help(numpy)

NameError: name 'numpy' is not defined
```

In [1]:
```
import numpy as np
help(np.array)
```

```
Help on built-in function array in module numpy:

array(...)
    array(object, dtype=None, *, copy=True, order='K', subok=False
, ndmin=0)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an obje
ct whose
        __array__ method returns an array, or any (nested) sequenc
e.
    dtype : data-type, optional
        The desired data-type for the array.  If not given, then t
he type will
        be determined as the minimum type required to hold the obj
ects in the
        sequence.
    copy : bool, optional
        If true (default), then the object is copied.  Otherwise,
a copy will
```

```
        only be made if __array__ returns a copy, if obj is a nest
ed sequence,
        or if a copy is needed to satisfy any of the other require
ments
        (`dtype`, `order`, etc.).
    order : {'K', 'A', 'C', 'F'}, optional
        Specify the memory layout of the array. If object is not a
n array, the
        newly created array will be in C order (row major) unless
'F' is
        specified, in which case it will be in Fortran order (colu
mn major).
        If object is an array the following holds.

        ===== ========= ===========================================
=========
        order  no copy                     copy=True
        ===== ========= ===========================================
=========
        'K'   unchanged F & C order preserved, otherwise most simi
lar order
        'A'   unchanged F order if input is F and not C, otherwise
C order
        'C'   C order   C order
        'F'   F order   F order
        ===== ========= ===========================================
=========

        When ``copy=False`` and a copy is made for other reasons,
the result is
        the same as if ``copy=True``, with some exceptions for `A`
, see the
        Notes section. The default order is 'K'.
    subok : bool, optional
        If True, then sub-classes will be passed-through, otherwis
e
        the returned array will be forced to be a base-class array
(default).
    ndmin : int, optional
        Specifies the minimum number of dimensions that the result
ing
        array should have.  Ones will be pre-pended to the shape a
s
        needed to meet this requirement.

    Returns
    -------
    out : ndarray
        An array object satisfying the specified requirements.

    See Also
    --------
    empty_like : Return an empty array with shape and type of inpu
```

```
t.
    ones_like : Return an array of ones with shape and type of inp
ut.
    zeros_like : Return an array of zeros with shape and type of i
nput.
    full_like : Return a new array with shape of input filled with
value.
    empty : Return a new uninitialized array.
    ones : Return a new array setting values to one.
    zeros : Return a new array setting values to zero.
    full : Return a new array of given shape filled with value.


    Notes
    -----
    When order is 'A' and `object` is an array in neither 'C' nor
'F' order,
    and a copy is forced by a change in dtype, then the order of t
he result is
    not necessarily 'C' as expected. This is likely a bug.

    Examples
    --------
    >>> np.array([1, 2, 3])
    array([1, 2, 3])

    Upcasting:

    >>> np.array([1, 2, 3.0])
    array([ 1.,  2.,  3.])

    More than one dimension:

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

    Minimum dimensions 2:

    >>> np.array([1, 2, 3], ndmin=2)
    array([[1, 2, 3]])

    Type provided:

    >>> np.array([1, 2, 3], dtype=complex)
    array([ 1.+0.j,  2.+0.j,  3.+0.j])

    Data-type consisting of more than one element:

    >>> x = np.array([(1,2),(3,4)],dtype=[('a','<i4'),('b','<i4')]
)
    >>> x['a']
    array([1, 3])
```

```
Creating an array from sub-classes:

>>> np.array(np.mat('1 2; 3 4'))
array([[1, 2],
       [3, 4]])

>>> np.array(np.mat('1 2; 3 4'), subok=True)
matrix([[1, 2],
        [3, 4]])
```

In [5]:
```
############## INTERDIT #################
# from math import *
# from numpy import *

# print(exp(1))
# print(sin(1))
```

```
2.718281828459045
0.8414709848078965
```

Notons au passage que l'importation par une commande du type `from numpy import *` est strictement interdit dans notre cours : il y a un grand risque de collision de fonctions et c'est à éviter à tout prix.

Création d'un tableau à partir d'une liste

Cela permet en particulier de remplir à la main des petits tableaux ou bien d'utiliser de manière astucieuses la compréhension de liste. On utilise pour cela la commande `array` qui transforme une liste en un tableau.

Voici quelques exemples :

In [4]:
```
# création d'un tableau d'entiers
A = np.array([1, 2, 3])
print(type(A), A)

# création d'un tableau de réels
A = np.array([1., 2, 3])
print(A)
```

```
<class 'numpy.ndarray'> [1 2 3]
[1. 2. 3.]
```

In [ ]:
```
# tableau à deux dimensions
A = np.array(
    [
        [1, 2, 3],
        [2, 3, 4]
    ]
)
print(A)
```

In [5]:
```
# Avec compréhension de liste
A = np.array([k**2/2 for k in range(-3, 4)])
print(A)
```

```
[4.5 2.  0.5 0.  0.5 2.  4.5]
```

In [ ]:
```
# Un exemple en dimension 2 pour faire une matrice
N = 10
B = np.array([
    [k*l for l in range(N+1)] for k in range(N+1)
])
print(B)
```

In [10]:
```
A = np.array(["toto", 1, 2., lambda x: x])
print(A.dtype)
```

```
object
```

Création d'un tableau à partir de commandes toutes prêtes

Il existe un grand nombre de tableaux connus que l'on peut créer à l'aide de commandes. Par exemple des tableaux vides, remplis de 0 ou de 1, des matrices diagonales, des nombres équirépartis entre 2 valeurs... Les commandes à connaitre sont les suivantes : (mais ce ne seront peut-être pas les seules que nous utiliserons !)

- `np.empty` , `np.zeros` , `np.ones` , `np.random.rand` ;
- `np.eye` , `np.identity` , `np.diag` , `np.triu` , `np.tril` ;
- `np.arange` , `np.linspace` .

In [15]:
```
nx, ny = 5, 4

# tableaux à 1 dimension
print(np.empty((nx,)))
print(np.zeros((nx,)))
# print(np.zeros(nx))
print(np.ones((nx,)))

# tableaux à 2 dimensions
print(np.empty((nx, ny)))
print(np.zeros((nx, ny)))
print(np.ones((nx, ny)))

# on peut ajouter des dimensions...
print(np.zeros((2, 2, 2)))

# pour faire des tableaux alétoires
print(np.random.rand(5))
print(np.random.rand(5, 5))
```

```
[0.30562184 0.35299477 0.78219896 0.09416814 0.61034607]
[0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[[0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]]]
[0.05394773 0.45685637 0.87944123 0.1295037  0.09739161]
[[0.32416458 0.46433385 0.00491584 0.78712345 0.96229485]
 [0.778651   0.94578333 0.1156074  0.02456486 0.35228066]
 [0.72125883 0.49003829 0.09871025 0.52285621 0.73976038]
 [0.82892019 0.00608217 0.91171069 0.08710232 0.49922933]
 [0.76973098 0.20620891 0.00658322 0.45979274 0.16894842]]
```

In [14]:
```
help(np.zeros)
```

```
Help on built-in function zeros in module numpy:

zeros(...)
    zeros(shape, dtype=float, order='C')

    Return a new array of given shape and type, filled with zeros.

    Parameters
    ----------
    shape : int or tuple of ints
        Shape of the new array, e.g., ``(2, 3)`` or ``2``.
    dtype : data-type, optional
        The desired data-type for the array, e.g., `numpy.int8`.
Default is
        `numpy.float64`.
    order : {'C', 'F'}, optional, default: 'C'
        Whether to store multi-dimensional data in row-major
        (C-style) or column-major (Fortran-style) order in
        memory.

    Returns
    -------
    out : ndarray
        Array of zeros with the given shape, dtype, and order.

    See Also
    --------
    zeros_like : Return an array of zeros with shape and type of i
nput.
    empty : Return a new uninitialized array.
    ones : Return a new array setting values to one.
    full : Return a new array of given shape filled with value.

    Examples
    --------
    >>> np.zeros(5)
    array([ 0.,  0.,  0.,  0.,  0.])

    >>> np.zeros((5,), dtype=int)
    array([0, 0, 0, 0, 0])

    >>> np.zeros((2, 1))
    array([[ 0.],
           [ 0.]])

    >>> s = (2,2)
    >>> np.zeros(s)
    array([[ 0.,  0.],
           [ 0.,  0.]])

    >>> np.zeros((2,), dtype=[('x', 'i4'), ('y', 'i4')]) # custom
dtype
    array([(0, 0), (0, 0)],
          dtype=[('x', '<i4'), ('y', '<i4')])
```

In [16]:
```
help(np.eye)
```

```
Help on function eye in module numpy:

eye(N, M=None, k=0, dtype=<class 'float'>, order='C')
    Return a 2-D array with ones on the diagonal and zeros elsewhe
re.

    Parameters
    ----------
    N : int
      Number of rows in the output.
    M : int, optional
      Number of columns in the output. If None, defaults to `N`.
    k : int, optional
      Index of the diagonal: 0 (the default) refers to the main di
agonal,
      a positive value refers to an upper diagonal, and a negative
value
      to a lower diagonal.
    dtype : data-type, optional
      Data-type of the returned array.
    order : {'C', 'F'}, optional
        Whether the output should be stored in row-major (C-style)
or
        column-major (Fortran-style) order in memory.

        .. versionadded:: 1.14.0

    Returns
    -------
    I : ndarray of shape (N,M)
      An array where all elements are equal to zero, except for th
e `k`-th
      diagonal, whose values are equal to one.

    See Also
    --------
    identity : (almost) equivalent function
    diag : diagonal 2-D array from a 1-D array specified by the us
er.

    Examples
    --------
    >>> np.eye(2, dtype=int)
    array([[1, 0],
           [0, 1]])
    >>> np.eye(3, k=1)
    array([[0.,  1.,  0.],
           [0.,  0.,  1.],
           [0.,  0.,  0.]])
```

In [22]:
```
N = 5
print(np.eye(N, N, k=-1))
A = 2*np.eye(N, N, k=0) - np.eye(N, N, k=-1) - np.eye(N, N, k=1)
print(A)
```

```
[[0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]]
[[ 2. -1.  0.  0.  0.]
 [-1.  2. -1.  0.  0.]
 [ 0. -1.  2. -1.  0.]
 [ 0.  0. -1.  2. -1.]
 [ 0.  0.  0. -1.  2.]]
```

In [ ]:
```
help(np.diag)
```

In [ ]:
```
x = np.random.rand(5)
print(x)
A = np.diag(x) + np.diag(x[1:], k=1) + np.diag(x[1:], k=-1)
print(A)
```

In [ ]:
```
# On peut aussi récupérer la partie triangulaire supérieure
# (ou la partie triangulaire inférieure) d'une matrice

A = np.random.rand(5, 5)
print(np.triu(A))
print(np.triu(A, k=-1))
```

In [28]:
```
# points équi-répartis entre a et b par pas de dx : np.arange(a, b, dx)
print(np.arange(10))
print(np.arange(2, 10))
print(np.arange(2, 10, 0.5))

# N points équi-répartis entre a et b : np.linspace(a, b, N)
print(np.linspace(0, 1, 11))
print(np.linspace(0, 1, 10, endpoint=False))
```

```
[0 1 2 3 4 5 6 7 8 9]
[2 3 4 5 6 7 8 9]
[2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5 9.  9.5]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

In [ ]:
```

```

Création d'un tableau à partir d'une fonction

Il est aussi possible de créer un tableau à partir d'une fonction sur les indices du tableau. Par exemple, pour un tableau de dimension 1

$$x_i = \phi(i), \quad 0 \leq i \leq N-1,$$

ou en dimension 3

$$A_{i,j} = \phi(i,j), \quad 0 \leq i,j \leq N-1.$$

On utilise pour le cela la commande `np.fromfunction` qui prend en argument une fonction (attention, par défaut les indices `i` , `j` , ... passés en argument à la fonction sont des `float`) et la forme du tableau sous la forme d'un tuple.

In [34]:
```
print(np.fromfunction(lambda i: 3*i+1, (10,)))
hilbert = lambda i, j: 1/(i+j+1)
print(np.fromfunction(hilbert, (50, 50)))
```

```
[ 1.  4.  7. 10. 13. 16. 19. 22. 25. 28.]
[[1.         0.5        0.33333333 ... 0.02083333 0.02040816 0.02
]
 [0.5        0.33333333 0.25       ... 0.02040816 0.02       0.019
60784]
 [0.33333333 0.25       0.2        ... 0.02       0.01960784 0.019
23077]
 ...
 [0.02083333 0.02040816 0.02       ... 0.01052632 0.01041667 0.010
30928]
 [0.02040816 0.02       0.01960784 ... 0.01041667 0.01030928 0.010
20408]
 [0.02       0.01960784 0.01923077 ... 0.01030928 0.01020408 0.010
10101]]
```

In [37]:
```
x = np.arange(10)
# print(np.fromfunction(lambda i, j: x[i+j % x.size], (x.size, x.si
ze)))
print(np.fromfunction(lambda i, j: x[(i+j) % x.size], (x.size, x.si
ze), dtype=int))
```

```
[[0 1 2 3 4 5 6 7 8 9]
 [1 2 3 4 5 6 7 8 9 0]
 [2 3 4 5 6 7 8 9 0 1]
 [3 4 5 6 7 8 9 0 1 2]
 [4 5 6 7 8 9 0 1 2 3]
 [5 6 7 8 9 0 1 2 3 4]
 [6 7 8 9 0 1 2 3 4 5]
 [7 8 9 0 1 2 3 4 5 6]
 [8 9 0 1 2 3 4 5 6 7]
 [9 0 1 2 3 4 5 6 7 8]]
```

In [36]:
```
help(np.fromfunction)
```

```
Help on function fromfunction in module numpy:

fromfunction(function, shape, *, dtype=<class 'float'>, **kwargs)
    Construct an array by executing a function over each coordinat
e.

    The resulting array therefore has a value ``fn(x, y, z)`` at
    coordinate ``(x, y, z)``.

    Parameters
    ----------
    function : callable
        The function is called with N parameters, where N is the r
ank of
        `shape`.  Each parameter represents the coordinates of the
array
        varying along a specific axis.  For example, if `shape`
        were ``(2, 2)``, then the parameters would be
        ``array([[0, 0], [1, 1]])`` and ``array([[0, 1], [0, 1]])`
`
    shape : (N,) tuple of ints
        Shape of the output array, which also determines the shape
of
        the coordinate arrays passed to `function`.
    dtype : data-type, optional
        Data-type of the coordinate arrays passed to `function`.
        By default, `dtype` is float.

    Returns
    -------
    fromfunction : any
        The result of the call to `function` is passed back direct
ly.
        Therefore the shape of `fromfunction` is completely determ
ined by
        `function`.  If `function` returns a scalar value, the sha
pe of
        `fromfunction` would not match the `shape` parameter.

    See Also
    --------
    indices, meshgrid

    Notes
    -----
    Keywords other than `dtype` are passed to `function`.

    Examples
    --------
    >>> np.fromfunction(lambda i, j: i == j, (3, 3), dtype=int)
    array([[ True, False, False],
           [False,  True, False],
           [False, False,  True]])
```

```
>>> np.fromfunction(lambda i, j: i + j, (3, 3), dtype=int)
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

In [ ]: