7. Numpy

本章小結

- 介紹
- 數據類型、運算子、比較
- Reshape, flatten, T, concatenate, split, vsplit
- Append, insert, delete, unique
- Rounding decimal
- Random, randint, rand, uniform, ndenumerate
- Scalar, vector and matrix
- Broadcasting
- Universal Function
- Filter, Conditional Statement with np.where

什麼是 NumPy

- NumPy ： Python 數值處理
- Python 計算開源專案
- 廣泛用於 python 程式設計
- 擴展到 Pandas、SciPy、Matplotlib、scikit-learn、scikit-image 和大多數其他數據科學和科學 Python 包

安裝

- pip install numpy

檢查安裝的版本

- numpy.__version__

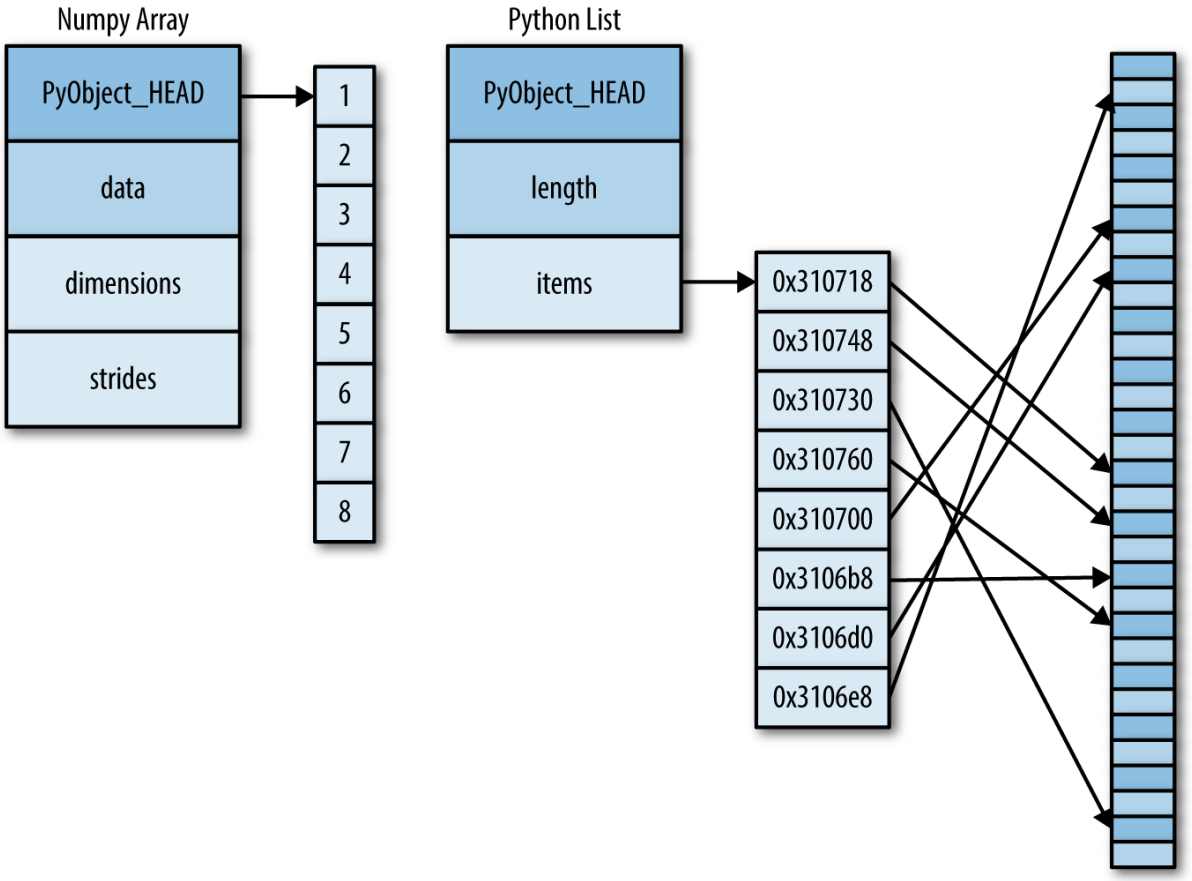
創建簡單的一維陣列

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

```
Out[3]:
array([1, 2, 3])
```

```
In [4]:
type(arr)
```

```
Out[4]:
numpy.ndarray
```



Numpy array

```
In [4]:
arr = np.array([1, 2, 3])
print(arr)
type(arr)
```

```
[1 2 3]
```

```
Out[4]:
```

```
numpy.ndarray
```

Python list

```
In [3]:
python_list = [1,2,3]
print(python_list)
type(python_list)
```

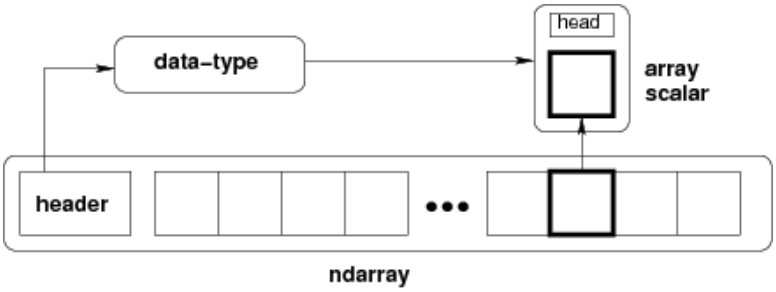
```
[1, 2, 3]
```

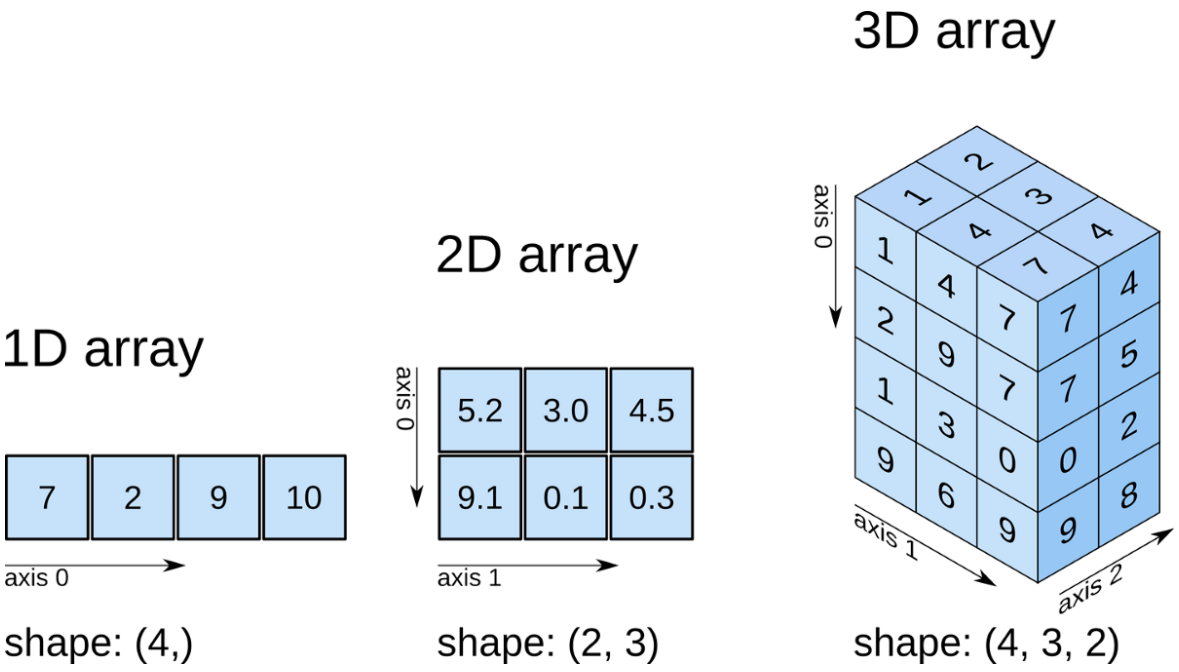
```
Out[3]:
```

```
list
```

Numpy 陣列物件

NumPy 提供了一個 N 維數位型態，即 ndarray，它描述了 同類型.這些專案可以是索引例如，使用 N 個整數。





創建二維陣列

In [5]:

```
arr_2D = np.array([[1,2],[3,4],[5,6],[7,8]])
print(arr_2D)
arr_2D.shape
```

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

Out[5]:

```
(4, 2)
```

訪問陣列值 Access the array value

In [6]:

```
arr_2D = np.array([[1,2],[3,4],[5,6],[7,8]])
print(arr_2D[1])    # [3 4]
print(arr_2D[3][0]) # 7
print(arr_2D[:2])
```

```
[3 4]
7
[[1 2]
 [3 4]]
```

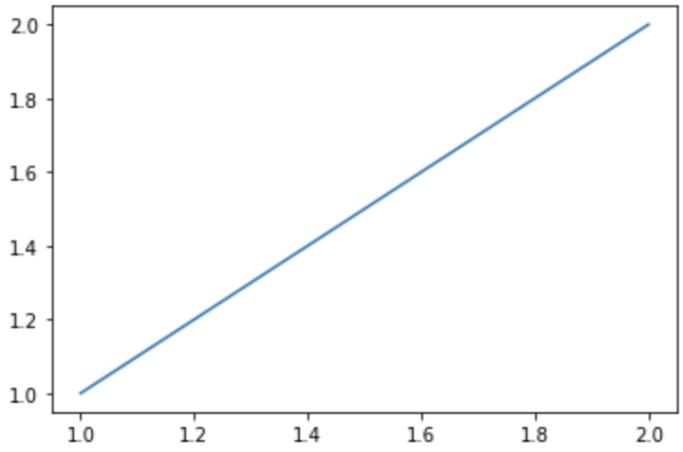
對陣列進行切片 Slicing the array

```
In [8]: arr_2D = np.array([[1,2],[3,4],[5,6],[7,8]])
        print(arr_2D[:,0])
        print(arr_2D[:,1])

        import matplotlib.pyplot as plt
        plt.plot(arr_2D[0][:], arr_2D[:][0])
```

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

```
Out[8]: [<matplotlib.lines.Line2D at 0x116f92700>]
```



重塑現有陣列 Reshape an existing array

```
In [8]: data = np.array([1,2,3,4,5,6])
        data.shape
```

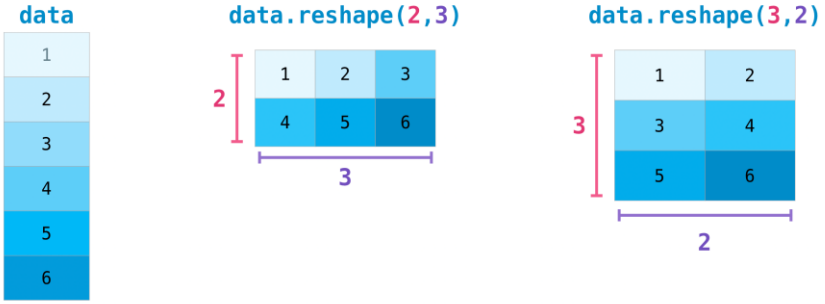
```
Out[8]: (6,)
```

```
In [9]: data.reshape(2,3)
```

```
Out[9]: array([[1, 2, 3],
               [4, 5, 6]])
```

```
In [10]: data.reshape(3,2)
```

```
Out[10]: array([[1, 2],
                [3, 4],
                [5, 6]])
```



創建 3D 陣列

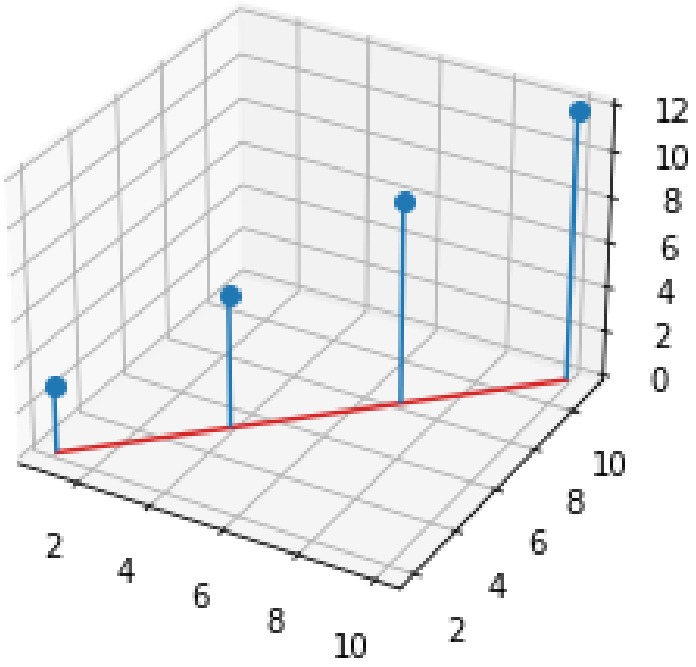
```
arr_3D = np.array([[[1,2,3],
                    [4,5,6]],
                   [[7,8,9],
                    [10,11,12]],
                   [[1,2,3],
                    [4,5,6]],
                  ])
arr_3D.shape
```

```
(3, 2, 3)
```

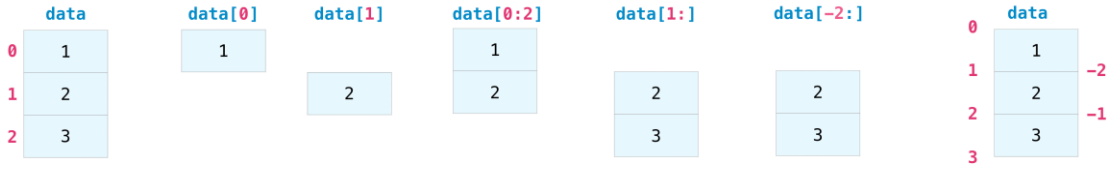
```
arr_3D = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9],
                   [10,11,12]])
```

```
X = arr_3D[:,0]  # 1,4,7,10
Y = arr_3D[:,1]  # 2,5,8,11
Z = arr_3D[:,2]  # 3,6,9,12

from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
fig, ax = plt.subplots(subplot_kw=dict(projection='3d'))
ax.stem(X, Y, Z)
plt.show()
```



一維陣列索引和切片

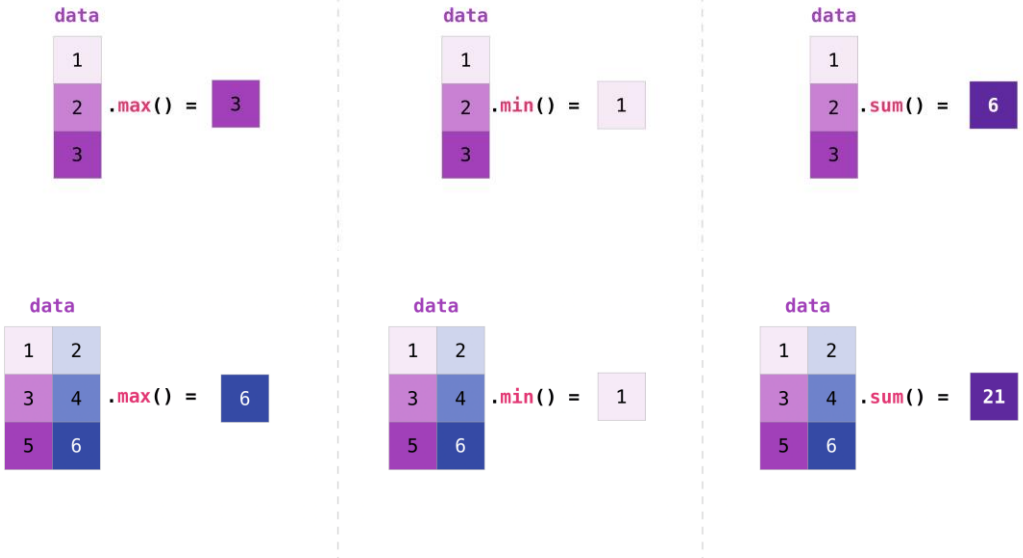


In [4]:

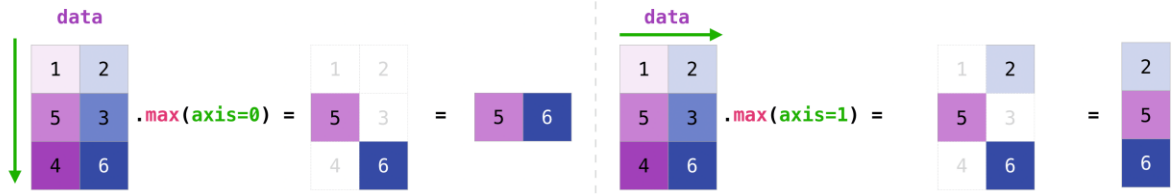
```
arr2 = np.array([1, 2, 3])
print(arr2[0])
print(arr2[1])
print(arr2[-1])
print(arr2[:])
print(arr2[1:])
print(arr2[:2])
```

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

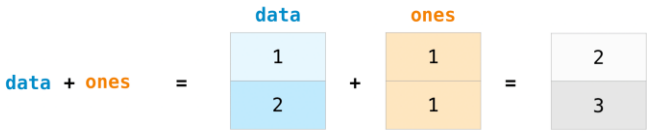
數位運算 – 最大值、最小值、總和

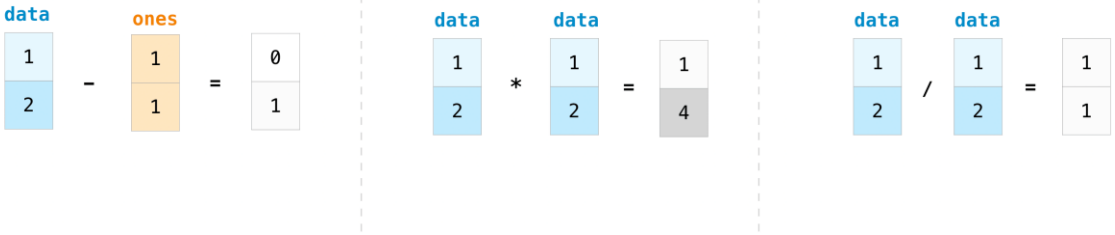


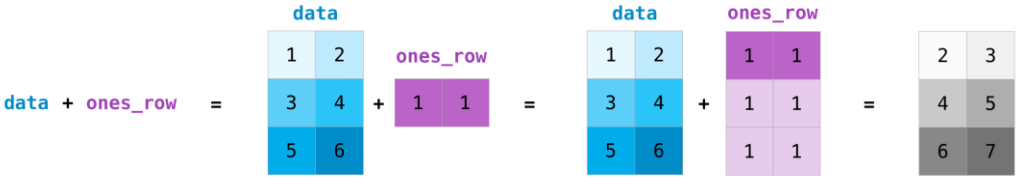
數位運算 – 最大值、最小值、軸總和值 Array Operation – max, min, sum in axis



陣列操作 Array Operation







標準 Numpy 數值資料類型

Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C `long`; normally either `int64` or `int32`)
`intc`	Identical to C `int` (normally `int32` or `int64`)
`intp`	Integer used for indexing (same as C `ssize_t`; normally either `int32` or `int64`)
`int8`	Byte (–128 to 127)
`int16`	Integer (–32768 to 32767)
`int32`	Integer (–2147483648 to 2147483647)
`int64`	Integer (–9223372036854775808 to 9223372036854775807)
`uint8`	Unsigned integer (0 to 255)
`uint16`	Unsigned integer (0 to 65535)
`uint32`	Unsigned integer (0 to 4294967295)
`uint64`	Unsigned integer (0 to 18446744073709551615)
`float_`	Shorthand for `float64`
`float16`	Half-precision float: sign bit, 5 bits exponent, 10 bits mantissa
`float32`	Single-precision float: sign bit, 8 bits exponent, 23 bits mantissa
`float64`	Double-precision float: sign bit, 11 bits exponent, 52 bits mantissa
`complex_`	Shorthand for `complex128`
`complex64`	Complex number, represented by two 32-bit floats
`complex128`	Complex number, represented by two 64-bit floats

NumPy 陣列上的計算可以非常快，也可以非常慢。讓它變得快速的關鍵是使用*矢量化*運算，通常通過 Numpy 的*通用函數*（ufuncs） 實現。

通用函數（或 ufunc 簡稱）是一個函數，用於 ndarrays 以逐個元素的方式，支援陣列廣播,類型鑄造，以及其他一些標準功能。

常用函數

- np.linsspace
- np.排列
- np.random.randint
- np.zeros
- np.ones 的

```
In [31]: # generate evenly spaced numbers over a specified interval
         np.linspace(0,2,9) # (smallest number, largest number, total number)

Out[31]: array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])

In [32]: # generate arrays with regularly incrementing values
         np.arange(10)

Out[32]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [33]: # np.arange(start, stop, interval)
         np.arange(9, 10, 0.1)

Out[33]: array([9. , 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9])

In [34]: # generate random integer (smallest integer, largest integer, total number)
         np.random.randint(1, 100, size=10)

Out[34]: array([87, 68, 82, 97, 87, 53, 47, 30, 34, 91])

In [35]: # create array of all zero
         np.zeros((2,3))

Out[35]: array([[0., 0., 0.],
                [0., 0., 0.]])

In [38]: # create array of all one
         np.ones((3,2))

Out[38]: array([[1., 1.],
                [1., 1.],
                [1., 1.]])
```

算術運算子和函數 Arithmetic operators and functions

Operator	Equivalent ufunc	Description
+	np.add	Addition (e.g., 1 + 1 = 2)
-	np.subtract	Subtraction (e.g., 3 - 2 = 1)
-	np.negative	Unary negation (e.g., -2)
*	np.multiply	Multiplication (e.g., 2 * 3 = 6)
/	np.divide	Division (e.g., 3 / 2 = 1.5)
//	np.floor_divide	Floor division (e.g., 3 // 2 = 1)
**	np.power	Exponentiation (e.g., 2 ** 3 = 8)
%	np.mod	Modulus/remainder (e.g., 9 % 4 = 1)

在 iPython 中函數

- //
- Mod
- **
- np.sqrt()
- np.log10()
- np.log()
- np.exp()
- np.around()
- np.ceil()
- np.floor()
- np.rint()

```
In [37]: # floor divide
         7//2
Out[37]: 3

In [38]: # modulas, remainder
         np.mod(7,2)
Out[38]: 1

In [39]: # power
         2**3
Out[39]: 8

In [40]: # square root
         np.sqrt(9)
Out[40]: 3.0

In [41]: # 10 logarithm of the input
         np.log10(1000)
Out[41]: 3.0

In [42]: # natural logarithm, element-wise.
         np.log(2.718281828459045)
Out[42]: 1.0

In [43]: # exponential of all elements in the input
         np.exp(1)
Out[43]: 2.718281828459045

In [44]: np.around(3.14159265359, decimals=2, out=None)
Out[44]: 3.14

In [45]: # Return the ceiling of the input, element-wise
         np.ceil([3.1, 3.9, 3])
Out[45]: array([4., 4., 3.])

In [46]: # Return the floor of the input, element-wise
         np.floor([3.9, 3.2, 3])
Out[46]: array([3., 3., 3.])
```

常見聚合函數 Common Aggregate Functions

Function Name	NaN-safe Version	Description
np.sum	np.nansum	Compute sum of elements
np.prod	np.nanprod	Compute product of elements
np.mean	np.nanmean	Compute median of elements
np.std	np.nanstd	Compute standard deviation
np.var	np.nanvar	Compute variance
np.min	np.nanmin	Find minimum value
np.max	np.nanmax	Find maximum value
np.argmin	np.nanargmin	Find index of minimum value
np.argmax	np.nanargmax	Find index of maximum value
np.median	np.nanmedian	Compute median of elements
np.percentile	np.nanpercentile	Compute rank-based statistics of elements
np.any	N/A	Evaluate whether any elements are true
np.all	N/A	Evaluate whether all elements are true

```
In [37]: arr = np.arange(0,10,0.5)
         arr

Out[37]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. ,
                6.5, 7. , 7.5, 8. , 8.5, 9. , 9.5])

In [38]: arr.size

Out[38]: 20

In [39]: arr.sum()

Out[39]: 95.0

In [40]: arr.min()

Out[40]: 0.0

In [41]: arr.max()

Out[41]: 9.5

In [42]: arr.mean()

Out[42]: 4.75

In [43]: np.median(arr)

Out[43]: 4.75

In [44]: # Standard Deviation
         np.std(arr)

Out[44]: 2.883140648667699
```

陣列中的數據類型結構

Character	Description	Example
'b'	Byte	np.dtype('b')
'i'	Signed integer	np.dtype('i4') == np.int32
'u'	Unsigned integer	np.dtype('u1') == np.uint8
'f'	Floating point	np.dtype('f8') == np.int64
'c'	Complex floating point	np.dtype('c16') == np.complex128
'S','a'	string	np.dtype('S5')
'U'	Unicode string	np.dtype('U') == np.str_
'V'	Raw data (void)	np.dtype('V') == np.void

```
# create an empty array with settings
client = np.array([('Alice','F',30,3500.5),
    ('Ben','M',35,2800.3),],dtype=[('name', 'U10'), ('gender', 'U1'), ('age', 'i4'), ('order_amount', 'f16')])
client
```

```
array([('Alice', 'F', 30, 3500.5), ('Ben', 'M', 35, 2800.3)],
      dtype=[('name', '<U10'), ('gender', '<U1'), ('age', '<i4'), ('order_amount', '<f16')])
```

```
client[0]
```

```
('Alice', 'F', 30, 3500.5)
```

```
type(client)
```

```
numpy.ndarray
```

```
new_client = client.copy()
new_client
```

```
array([('Alice', 'F', 30, 3500.5), ('Ben', 'M', 35, 2800.3)],
      dtype=[('name', '<U10'), ('gender', '<U1'), ('age', '<i4'), ('order_amount', '<f16')])
```

```
new_client[0]
```

```
('Alice', 'F', 30, 3500.5)
```

```
new_client[0][2]=18
```

```
new_client[0]
```

```
('Alice', 'F', 18, 3500.5)
```

比較運算子 Comparison Operator

Operator	Equivalent ufunc
==	np.equal
!=	np.not_equal
<	np.less
<=	np.less_equal
>	np.greater
>=	np.greater_equal

```
In [63]: arr = np.array([1, 2, 3, 4, 5])
         arr < 3
Out[63]: array([ True,  True, False, False, False])

In [64]: arr >= 3
Out[64]: array([False, False,  True,  True,  True])

In [65]: arr != 3
Out[65]: array([ True,  True, False,  True,  True])

In [66]: arr == 3
Out[66]: array([False, False,  True, False, False])

In [67]: len(arr)
Out[67]: 5

In [68]: np.count_nonzero(arr>=2)
Out[68]: 4
```

陣列操作 Array Manipulation

In [72]:
```
arr = np.zeros([1,2])
arr
```

Out[72]:
```
array([[0., 0.]])
```

In [73]:
```
# Append values to the end of an array
# numpy.append(arr, values, axis=None)
arr = np.append(arr, [[2,2]], axis=0)
arr
```

Out[73]:
```
array([[0., 0.],
       [2., 2.]])
```

In [74]:
```
arr = arr.flatten()
arr
```

Out[74]:
```
array([0., 0., 2., 2.])
```

In [75]:
```
arr = arr.reshape([2,2])
arr
```

Out[75]:
```
array([[0., 0.],
       [2., 2.]])
```

In [76]:
```
# Insert values along the given axis before the given indices.
# numpy.insert(arr, obj, values, axis=None) , axis=0 means insert by row
arr = np.insert(arr, [1], [1,1], axis=0)
arr
```

Out[76]:
```
array([[0., 0.],
       [1., 1.],
       [2., 2.]])
```

In [77]:
```
# axis=0 means insert by column
arr = np.insert(arr, [1], [3], axis=1)
arr
```

Out[77]:
```
array([[0., 3., 0.],
       [1., 3., 1.],
       [2., 3., 2.]])
```

In [78]:
```
# Join a sequence of arrays along an existing axis.
# numpy.concatenate((a1, a2, ...), axis=0, out=None, dtype=None, casting="same_kind")
arr2 = np.array([[3,3,3]])
arr3 = np.concatenate((arr,arr2), axis=0)
arr3
```

Out[78]:
```
array([[0., 3., 0.],
       [1., 3., 1.],
       [2., 3., 2.],
       [3., 3., 3.]])
```

NumPy 陣列轉換 Array conversion

In [69]:
```
# Python list to np.array
arr = np.array([1,2,3,4,5])
type(arr)
```

Out[69]:
```
numpy.ndarray
```

In [70]:
```
# np.array to Python list
p_list = arr.tolist()
type(p_list)
```

Out[70]:
```
list
```

In [71]:
```
# Tuple to np.array
arr1 = np.array((1,2,3,4,5))
type(arr1)
```

Out[71]:
```
numpy.ndarray
```

In [72]:
```
p_tuple = tuple(arr1)
type(p_tuple)
```

Out[72]:
```
tuple
```

標量、向量和矩陣 Scalar, Vector and Matrix

標量 scalar 是一個量級或數值。

- 示例：品質、速度、距離、時間、能量、密度、體積、溫度、距離

向量 vector 是一個大小和一個方向。

- 例如：位移、加速度、力、動量、重量、光速、引力場、電流

```
import numpy as np
# Scalar
s = 1
```

```
# Vector
v1 = np.array([1,4,5])  # horizontal
v2 = np.array([[1],
               [-5]])   # vetical vector
print("v1 shape: ", v1.shape)
print("v2 shape: ", v2.shape)
```

```
v1 shape:  (3,)
v2 shape:  (2, 1)
```

矩陣是一種二維數據結構，其中數字排列成行和列。Python 中沒有矩陣的數據類型。它由嵌套清單或陣列定義。

```
# Matrice
m = np.array([[1, 4, 5],
              [-5, 8, 9]])
m
```

```
array([[ 1,  4,  5],
       [-5,  8,  9]])
```

```
m.shape
```

```
(2, 3)
```

np.transpose

np.T 返回一個軸轉置的陣列。與 np.transpose 相同。

```
np.arange(1,7).reshape(3,2)
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
np.arange(1,7).reshape(3,2).T
```

```
array([[1, 3, 5],
       [2, 4, 6]])
```

np.flatten（ ）

返回摺疊成一維的陣列的副本。

```
1 np.arange(1,7).reshape(3,2)
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
1 np.arange(1,7).reshape(3,2).flatten()
```

```
array([1, 2, 3, 4, 5, 6])
```

np.concatenate numpy 陣列

numpy.concatenate(*(a1, a2, ...), axis=0*)。沿現有軸連接一系列陣列。

```
1 a = np.array([[1, 2], [3, 4]])
2 b = np.array([[5, 6]])
3 np.concatenate((a, b), axis=0)
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
1 np.concatenate((a, b.T), axis=1)
```

```
array([[1, 2, 5],
       [3, 4, 6]])
```

```
1 np.concatenate((a, b), axis=None)
```

```
array([1, 2, 3, 4, 5, 6])
```

np.split 陣列分拆

np.split（*ary*， *indices_or_sections*， *axis=0*）。將一個陣列拆分為多個子陣列，作為陣列的檢視。

```
x = np.arange(9)
x
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
arr1, arr2, arr3 = np.split(x, 3)
arr1
```

```
array([0, 1, 2])
```

```
np.split(x, [0,6])
```

```
[array([], dtype=int64), array([0, 1, 2, 3, 4, 5]), array([6, 7, 8])]
```

```
np.split(x, [0,6])[1]
```

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

```
np.split(x, [0,6])[2]
```

```
array([6, 7, 8])
```

np.append 陣列添加

np.append（*arr*， *values*， *axis=None*）。將值追加到陣列的末尾。

```
x = np.arange(1,7).reshape(2,3)
x
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
x = np.append(x, [[7,8,9]], axis=0)
x
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
np.append(x, [[3.5],[6.5],[9.5]], axis=1)
```

```
array([[1. , 2. , 3. , 3.5],
       [4. , 5. , 6. , 6.5],
       [7. , 8. , 9. , 9.5]])
```

np.insert 陣列插入

np.insert（*arr*， *obj*， *values*， *axis=None*）。在給定索引之前沿給定軸插入值。

```
a = np.array([[1, 1], [2, 2], [3, 3]])
a
```

```
array([[1, 1],
       [2, 2],
       [3, 3]])
```

```
np.insert(a, [1], [[9],[9],[9]], axis=1)
```

```
array([[1, 9, 1],
       [2, 9, 2],
       [3, 9, 3]])
```

```
np.insert(a, 1, 5)
```

```
array([1, 5, 1, 2, 2, 3, 3])
```

小數進位 Rounding decimal

在 NumPy 中，主要有五種方法可以四捨五入小數：

- truncation
- fix (same as truncation)
- rounding
- floor
- ceil

```
arr = np.array([-1.99, -1.13, 0.56, 1.28, 2.10, 2.65])
```

```
# truncate decimal
np.trunc(arr)
```

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

```
# Round to nearest integer towards zero.
np.fix(arr)
```

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

```
# round off to n decimal point
np.round(arr, 1)
```

```
array([-1.9, -1.1,  0.5,  1.2,  2.1,  2.6,  3.2,  3.8])
```

```
np.ceil(arr)
```

```
array([-1., -1.,  1.,  2.,  3.,  3.,  4.,  4.])
```

```
np.floor(arr)
```

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

隨機數 Random Numbers

隨機數並不意味著每次都有不同的數位。隨機意味著無法從邏輯上預測的東西。隨機是數據科學和機器學習中的關鍵話題。

應用實例：

- 密碼學
- 加密貨幣錢包
- 類比
- 機器學習
- 科學研究

random.**randint**(*low*, *high=None*, *size=None*, *dtype=int*)

- 從低（*含*）*到*高（*不含*）返回隨機整數。
- 從指定 dtype 的「離散均勻」分佈中返回隨機整數，該分佈位於「半開」區間 [*low*， *high*]。如果 *high* 為 None（預設值），則結果來自 [0， *low*]。

```
from numpy import random
# Generate 10 ints between 0 and 99, inclusive
random.randint(100 , size=(10))
```

```
array([44,  2, 49, 22, 32, 39, 58, 65, 33, 72])
```

```
#Generate a 2 x 4 array of ints between 1 and 8, inclusive
np.random.randint(1,9, size=(2, 4))
```

```
array([[1, 8, 4, 7],
       [1, 3, 2, 5]])
```

random.**rand**(*d0*, *d1*, ..., *dn*)Random values in a given shape.

創建一個給定形狀的陣列，並用來自 [0， 1] 上均勻分佈的隨機樣本填充它。

```
random.rand(5)
```

```
array([0.8011264 , 0.25988628, 0.90796268, 0.7302317 , 0.26139519])
```

```
random.rand(4,5)
```

```
array([[0.3452997 , 0.84728647, 0.85974315, 0.70047975, 0.69632204],
       [0.07339232, 0.08965751, 0.93790906, 0.02398837, 0.56169819],
       [0.71365956, 0.62741451, 0.29302985, 0.65682291, 0.10798099],
       [0.31750426, 0.52681618, 0.37137185, 0.88634499, 0.13717501]])
```

random.choice

random.**choice**(*a*, *size=None*, *replace=True*, *p=None*). 從給定的一維陣列生成隨機樣本

```
# choice of 1-29, pick 10 random, repeated number allowed
random.choice(np.arange(1,30), 10, replace=True)
```

```
array([19,  5, 20, 11,  1, 26, 10,  5,  1, 25])
```

```
random.choice(["A","B","C","D"], 2, replace=False)
```

```
array(['C', 'D'], dtype='<U1')
```

```
# pick 6 random mark 6 number
mark6 = random.choice(np.arange(1,50), 6, replace=False)
sorted(mark6)
```

```
[10, 11, 14, 18, 22, 44]
```

random.uniform

隨機。**uniform**（*low=0.0*， *high=1.0*， *size=None*）。從均勻分佈中抽取樣本。

樣本均勻分佈在半開區間 [低、高]（包括低，但不包括高）。

```
# random number range -1 to +1 , size: 20
s = np.random.uniform(-1,1,20)
s
```

```
array([-0.68894514,  0.08310177, -0.85816642, -0.41840843,  0.3839412 ,
       -0.88159189, -0.03724022, -0.44483262,  0.20934449,  0.04664035,
       -0.3505514 ,  0.23609872, -0.98573827, -0.54794272, -0.442871  ,
        0.82176829,  0.02372383,  0.13043628,  0.38517956,  0.01710117])
```

ndenumerate 陣列

麻木。**ndenumerate**（*arr*）。多維索引反覆運算器。

返回一個反覆運算器，生成數位座標和值對。

```
a = np.array([[1, 2], [3, 4]])
for index, x in np.ndenumerate(a):
    print(index, x)
```

```
(0, 0) 1
(0, 1) 2
(1, 0) 3
(1, 1) 4
```

Broadcasting

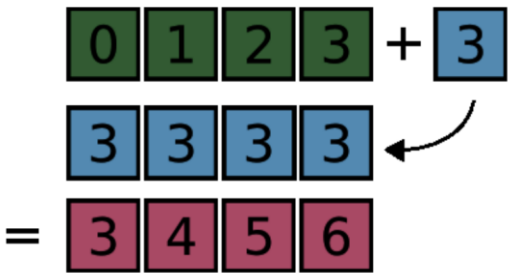
術語 broadcasting 描述了 NumPy 在算術運算期間如何處理具有不同形狀的陣列。

在兩個陣列上操作時，NumPy 會按元素比較它們的形狀。它從尾隨（即最右邊）的維度開始，然後向左工作。當

一. 它們是相等的，或者

二. 其中之一是 1。

Broadcasting – 1D



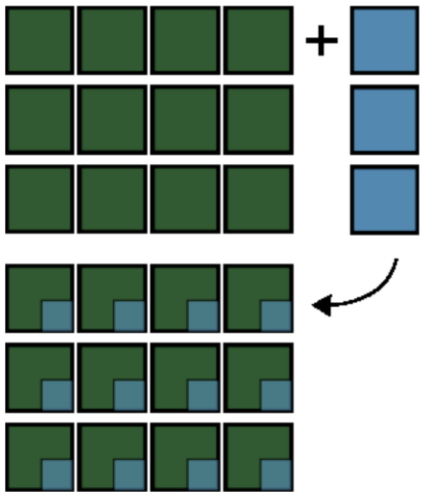

```
x = np.array([0,1,2,3])
x + np.array([3,3,3,3])
```

```
array([3, 4, 5, 6])
```

```
x = np.array([0,1,2,3])
x+3
```

```
array([3, 4, 5, 6])
```

Broadcasting – 2D



```
a = np.arange(12).reshape((3,4))
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

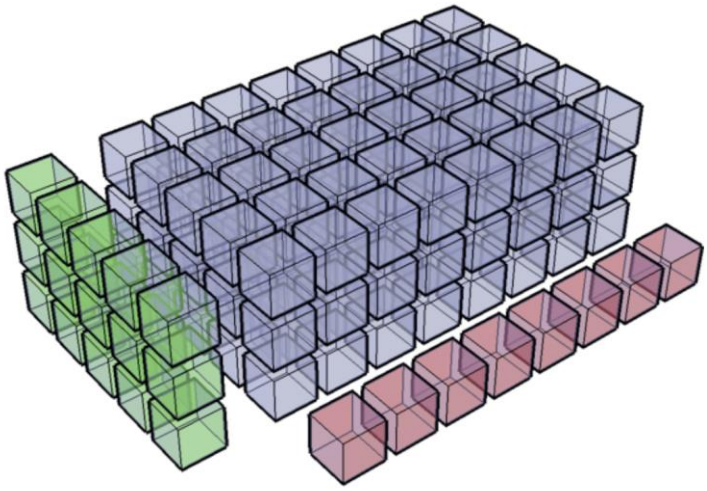
```
b = np.array([1 , 2 , 3])[:, None]
b
```

```
array([[1],
       [2],
       [3]])
```

```
a + b
```

```
array([[ 1,  2,  3,  4],
       [ 6,  7,  8,  9],
       [11, 12, 13, 14]])
```

Broadcasting – 3D



```
x = np.zeros((3,5))
x
```

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

```
y = np.zeros(8)
y
```

```
array([0., 0., 0., 0., 0., 0., 0., 0.])
```

省略號... 表示盡可能多的 ： 根據需要

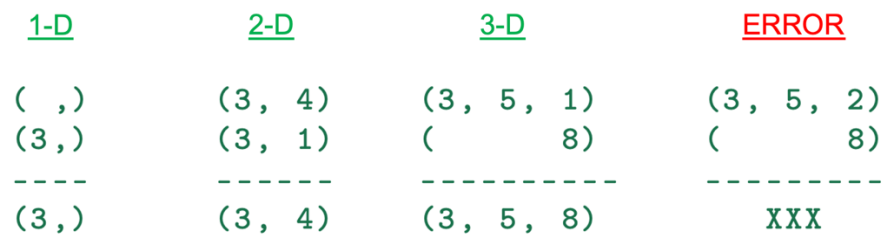
```
x[...,None] + y     # shape(3,5,8)
```

```
array([[[0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0.]]])
```

Broadcasting 比較維度，從最後一個開始。當任一維度為 1 或 None 時，或者如果維度相等時，則匹配：



Broadcasting 代碼簡單，但會消耗記憶體。明智地使用它。較快的演算法應為 np.add、np.subtract、np.multiply、np.divide 等算術。

創建你自己的 ufunc

創建一個 ufunc 來計算元素 numpy 陣列內矩形的面積。

```
1  def area_rect(x,y):
2      return x*y
3
4  # create an ufunc with 2 input 1 output
5  funcarea = np.frompyfunc(area_rect, 2, 1)
6
7  arr1 = [1,2,3,4]
8  arr2 = [2,3,4,5]
9  funcarea(arr1, arr2)
```

```
array([2, 6, 12, 20], dtype=object)
```

```
1  type(funcarea)
```

```
numpy.ufunc
```

使用條件過濾陣列

```
1 arr = np.array(np.arange(1,10))
2 arr
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
1 # filter with even number condition
2 arr[arr % 2 == 0]
```

```
array([2, 4, 6, 8])
```

```
1 arr[arr > 5]
```

```
array([6, 7, 8, 9])
```

幾乎每個 NumPy 函數都是元素式的(element-wise)。這些功能有助於加快數據操作和性能。

NumPy 函數專為數位處理而設計。

在官方網站上瞭解更多資訊： numpy.org

參考

官方網站：

https://numpy.org/doc/stable/index.html

參考教材：

Python Data Science Handbook, Jake VanderPlas, O'Reilly

Python for Data Analysis, 2nd edition, Wes McKinney, O'Reilly

更多練習：

https://www.machinelearningplus.com/