NumPy

NumPy是專為科技運算設計的第三方套件，相較於Python的標準列表型，能更有效率地處理多維陣列的資料，所以也成為Python的科技運算基礎套件

NumPy的概要

NumPy是Python的第三方套件，能有效率地處理陣列與矩陣的運算。

NumPy包含陣列類型的ndarry與矩陣類型的matrix，這些陣列或矩陣的元素必須是相同的資料類型，可指定 int 16 或 float 32 這類 NumPy 專用的數值類型。

NumPy內建了專用的運算函數與方法，可高速完成陣列與矩陣的計算。

利用NumPy處理資料

1. 一維陣列

```
#使用as關鍵字設定為as np，就可利用np呼叫numpy
import numpy as np
```

```
#先處理一維陣列。
#將Python的list傳遞給array函數，就能建立ndarry物件。將3個元素的一維陣列放入變數a
a = np.array([1,2,3])
```

```
#確認以array為字首的輸出結果
a
```

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

```
#利用print函數輸出a。使用print函數就不會輸出array，而是改以空白為間隔字元
print(a)
```

```
[1 2 3]
```

```
#利用type函數確認a這個物件。可以確定a是NumPy的陣列ndarray物件
type(a)
```

```
numpy.ndarray
```

```
#確認是一維陣列與3個元素
a.shape
```

```
(3,)
```

2. 二維陣列

```
#與一維陣列時一樣，使用array函數。這次讓Python的list轉換成巢狀結構的二重list，再利用二重list建立二維的ndarray物件，再將物件帶入變數b
b = np.array([[1,2,3],[4,5,6]])
b
```

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

```
#(2,3)的輸出結果可解讀成2*3矩陣
b.shape
```

```
(2, 3)
```

3. 翻轉

```
#一開始要先建立6個元素的一維陣列，再代入變數c1
c1 = np.array([0,1,2,3,4,5])
c1
```

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

```
#確認c1儲存了NumPy的一維陣列。接著使用reshape方法將這個陣列轉換成2*3矩陣的陣列。
#第一列依序放了3個元素，第二列則放了剩下的3個元素。使用reshape方法時，重點在於元素數量。若是c1.reshape((3,4))這種元素數量不一致的情況會顯示錯誤訊息
c2 = c1.reshape((2,3))
c2
```

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

```
#使用ravel方法讓陣列恢復成一維陣列
c3 = c2.ravel()
c3
```

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

```
#flatten方法
#傳回copy
c4 = c2.flatten()
c4
```

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

- ravel方法與flatten方法的差異在於傳回結果的方法。ravel傳回的是參照，但flatten傳回的是複製

4. 資料類型 (dtype)

Numpy 陣列的元素可利用dtype屬性確認資料類型。

Numpy的元素必須只有一種資料類型，該類型也是NumPy傳統的類型。一開始先確認一維陣列a的元素是什麼資料類型。

```
#NumPy陣列a是以[1,2,3]與Python的int類型的資料建立。若在建立陣列之際未宣告類型，將自動宣告為np.int64
a.dtype
```

```
dtype('int64')
```

```
#宣告np.int16這種類型，建立NumPy陣列
d = np.array([1,2],dtype=np.int16)
d
```

```
array([1, 2], dtype=int16)
```

```
#利用dtype屬性確認資料類型
d.dtype
```

```
dtype('int16')
```

```
#NumPy陣列除了可儲存整數，也可儲存浮點數與布林值
d.astype(np.float16)
```

```
array([1., 2.], dtype=float16)
```

5. 索引與切片

NumPy陣列與Python的標準list一樣，都可以使用索引與切片取得元素

```
#先確認前面建立的一維陣列
a
```

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

```
#與Python標準list一樣，指定索引值0，就能取得開頭的資料
a[0]
```

```
1
```

```
#與Python標準list一樣，可利用[1:]指定切片範圍
a[1:]
```

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

```
#也可以如Python的標準list一樣使用負數的索引
a[-1]
```

```
3
```

```
#先確認之前建立的陣列 b
b
```

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

```
#指定0，因此會以一維陣列的方式取得第一列的元素。若以逗號指定兩個值，就能取得列與行的索引值所對應的值。
b[0]
```

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

```
#取第二列第一行的值
b[1,0]
```

```
4
```

```
#利用切片範圍指定列或行。[:,2]，所取得列方向所有元素，以及行方向第三個元素
b[:,2]
```

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

```
b[1,:]
```

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

```
#列或行也能個別指定範圍
b[0,1:]
```

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

```
#即使索引值不連續，也能取得列或行的元素
b[:,[0,2]]
```

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

6. 重新代入資料

```
#先確認一維陣列a
a
```

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

```
#將索引值[2]的3換成4
a[2] = 4
a
```

```
array([1, 2, 4])
```

```
#先確認二維陣列b
b
```

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

```
#要變更二維陣列的資料時，可指定列與行的索引值
b[1,2]=7
b
```

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

```
#針對所有的列，變更同一行的值
b[:,2] = 8
b
```

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

7. 深複製 (copy)

```
#先將陣列a代入a1
a1 = a
a1
```

```
array([1, 2, 4])
```

```
#變更陣列a1的資料
a1[1] = 5
a1
```

```
array([1, 5, 4])
```

```
#a1的確變更了。接著確認陣列a的資料
a
```

```
array([1, 5, 4])
```

- 陣列 a 變得與陣列 a1一樣了。雖然沒直接替換陣列 a 的資料，但是 a1=a 的操作產生了參照陣列 a 的 a1物件，所以當 a1的資料被置換，被參照的陣列a也跟著改變了

```
#以copy方法複製資料。陣列a與a2儲存相同的資料
a2 = a.copy()
a2
```

```
array([1, 5, 4])
```

```
#變更a2的資料
a2[0] = 6
a2
```

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

```
#確認複製來源的陣列a
a
```

```
array([1, 5, 4])
```

- a 的資料完全沒變

看看在翻轉中的 ravel方法與 flatten方法有什麼不同

```
#先確認c2內容
c2
```

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

```
#將ravel方法的執行結果放入c3，並將flatten的執行結果放入c4之後，再變更c3與c4的部分元素
c3 = c2.ravel()
c4 = c2.flatten()
c3[0] = 6
c4[1] = 7
```

```
c3
```

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

```
c4
```

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

```
c2
```

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

- 由此可知，ravel方法是參照，flatten方法是複製
- Python 內建的list是以複製的方式傳遞切片結果，但是NumPy卻是以參照的方式傳遞切片結果

```
#以Python的list而言
py_list1 = [0,1]
py_list2 = py_list1[:]
py_list2[0] = 2
print(py_list1)
print(py_list2)
```

```
[0, 1]
[2, 1]
```

```
#以NumPy的adarray示範
np_array1 = np.array([0,1])
np_array2 = np_array1[:]
np_array2[0] = 2
print(np_array1)
print(np_array2)
```

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

- 廣義來說，複製也包含傳遞參照的意思，但如果要明確地將參照與複製分開來看，參照應該稱為淺複製 (Shallow Copy)，其他情況則稱為深複製 (Deep Copy)

8. 傳回數列(arange)

Python內建的range函數可建立數列，而NumPy也有類似的函數。使用arange函數可建立NumPy陣列(ndarray)

```
np.arange(10)
```

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

- 這次將參數設定為一個整數(10)，所以輸出0至9，共10個整數的陣列。
- 假設參數設定為兩個整數，arrange函數也能與Python內建的range函數產生相同的結果

```
np.arange(1,11)
```

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

- 將輸出第一個參數為起始值，第二個參數為結尾前一個數值的陣列

```
#指定三個參數也能得到與Python內建的range函數一樣的結果，也將輸出ndarray物件
np.arange(1,11,2)
```

```
array([1, 3, 5, 7, 9])
```

9. 亂數

Python 內建的random模組可產生亂數，而NumPy也內建了能快速產生大量亂數的函數

np.random.random函數一接到列與行的元組，就會產生二維陣列，此二維陣列將儲存大於等於0，小於1的亂數

```
#此功能很適合用來建立具有介於0於1之間的元素的矩陣
#使用此項功能產生的亂數，可產生每次都不一樣的資料
f = np.random.random((3,2))
f
```

```
array([[0.82290783, 0.99332532],
       [0.74778527, 0.12455118],
       [0.53855524, 0.95137081]])
```

```
#亂數值是固定的，就能得到固定的結果。將亂數種子的值指定為123
np.random.seed(123)
np.random.random((3,2))
```

```
array([[0.69646919, 0.28613933],
       [0.22685145, 0.55131477],
       [0.71946897, 0.42310646]])
```

- np.random.rand函數與np.random.random一樣能產生介於0-1範圍的亂數陣列，random函數會傳遞列與行的元組，但rand函數卻會以兩個參數傳遞形狀

```
np.random.seed(123)
np.random.rand(4,2)
```

```
array([[0.69646919, 0.28613933],
       [0.22685145, 0.55131477],
       [0.71946897, 0.42310646],
       [0.9807642 , 0.68482974]])
```

- 產生隨意整數的np.random.randint函數

```
#指定在大於等於1，小於10的整數之中選一個輸出，輸出結果為3
np.random.seed(123)
np.random.randint(1,10)
```

```
3
```

```
np.random.seed(123)

#np.random.randint函數可利用第一個參數指定大於等於的數值，並以第二個整數指定小於的數值，而第三個參數則是以元組傳遞的列與行建立的二維陣列，亂數就是在此二維陣列產生
np.random.randint(1,10,(3,3))
```

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

```
np.random.seed(123)

#np.random.uniform函數可利用第一個參數指定大於等於的數值，並以第二個整數指定小於的數值，第三個參數則是以元組傳遞的列與行建立的二維陣列，隨機的小數點就是在此陣列產生
#大於等於0.0，小於5.0的範圍內，建立2*3矩陣的二維陣列
np.random.uniform(0.0,5.0,size=(2,3))
```

```
array([[3.48234593, 1.43069667, 1.13425727],
       [2.75657385, 3.59734485, 2.1155323 ]])
```

```
np.random.seed(123)

#第一個參數與第二個參數都可省略，省略時，第一個參數將自動指定為0.0，第二個參數則自動指定為1.0。
#np.random.uniform函數與np.random.randint的差異在於傳回值ndarray的元素會是小數點的數值
#因沒有指定數值的範圍，所以將採用預設值，在大於等於0.0，小於1.0的範圍內，產生4*3矩陣的二維陣列
np.random.uniform(size=(4,3))
```

```
array([[0.69646919, 0.28613933, 0.22685145],
       [0.55131477, 0.71946897, 0.42310646],
       [0.9807642 , 0.68482974, 0.4809319 ],
       [0.39211752, 0.34317802, 0.72904971]])
```

- 如何利用np.random.randn輸出符合標準常態分配的亂數。

- np.random.randn與np.random.rand一樣，都會將形狀傳遞至參數，亂數也將符合標準常態分佈，以平均值0、變異數1的分配輸出

```
np.random.seed(123)
np.random.randn(4,2)
```

```
array([[-1.0856306 ,  0.99734545],
       [ 0.2829785 , -1.50629471],
       [-0.57860025,  1.65143654],
       [-2.42667924, -0.42891263]])
```

10. 建立元素相同的數列

```
#將整數的參數傳入zeros函數，就能取得以參數指定的元素數量0.0的陣列
np.zeros(3)
```

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

```
#傳遞二個元素的元組，依照指定的列行數量建立二維陣列
np.zeros((2,3))
```

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

```
#將ones函數的參數指定為整數，取得以參數指定的元素數量1.0陣列
np.ones(2)
```

```
array([1., 1.])
```

```
#建立二維陣列
np.ones((3,4))
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

11. 單位矩陣

```
#eye函數可指定單位矩陣位於對角線的元素，藉此建立單位矩陣
np.eye(3)
```

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

12. 以指定值填滿陣列

```
#利用full函數將100這個數值放入具有3個元素的陣列
np.full(3,100)
```

```
array([100, 100, 100])
```

```
#指定列與行。這次使用的是NumPy的常數值，也是代表圓周率的np.pi
np.full((2,4), np.pi)
```

```
array([[3.14159265, 3.14159265, 3.14159265, 3.14159265],
       [3.14159265, 3.14159265, 3.14159265, 3.14159265]])
```

- 填補 NumPy缺損值的特殊數值np.nan
- naa 是 Not a Number的縮寫，可宣告元素並非數值，也被分類為float資料類型。
- NumPy的ndarray只能存納資料類型相同的資料，而且為方便計算，Python的None或空白字串是無法計算的，所以另外設計np.nan此特殊的常數

```
np.nan
```

```
nan
```

```
np.array([1,2,np.nan])
```

```
array([ 1.,  2., nan])
```

13. 在指定範圍內建立間距相等的資料

```
#使用linspace函數可建立5個元素以相等間距在0-1的範圍內配置的陣列
np.linspace(0,1,5)
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

- linspace函數與使用arange函數，執行np.arange(0.0 , 1.1 , 02.5)的結果一樣，但使用linspace函數比較方便

```
#建立0-π，20等分的資料
np.linspace(0,np.pi,21)
```

```
array([0.        , 0.15707963, 0.31415927, 0.4712389 , 0.62831853,
       0.78539816, 0.9424778 , 1.09955743, 1.25663706, 1.41371669,
       1.57079633, 1.72787596, 1.88495559, 2.04203522, 2.19911486,
       2.35619449, 2.51327412, 2.67035376, 2.82743339, 2.98451302,
       3.14159265])
```

14. 元素之間的差距

np.diff可傳回元素之間的差距

```
#傳回每個元素之間的差距
l = np.array([2,2,6,1,3])
np.diff(l)
```

```
array([ 0,  4, -5,  2])
```

15. 連結

```
#先確認之前建立的NumPy陣列a與a1的內容
print(a)
print(a1)
```

```
[1 5 4]
[1 5 4]
```

```
#利用concatenate函數連結這兩個陣列
np.concatenate([a,a1])
```

```
array([1, 5, 4, 1, 5, 4])
```

```
#確認先前建立的二維陣列
b
```

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

```
#以下列的命令建立二維陣列
b1 = np.array([[10], [20]])
b1
```

```
array([[10],
       [20]])
```

```
#使用concatenate函數連結
#由於欄位(行方向)會增加，所以指定axis=1
np.concatenate([b,b1], axis=1)
```

```
array([[ 1,  2,  8, 10],
       [ 4,  5,  8, 20]])
```

```
#使用hstack函數可得到相同的效果
np.hstack([b,b1])
```

```
array([[ 1,  2,  8, 10],
       [ 4,  5,  8, 20]])
```

```
#再次建立一微陣列b2
b2 = np.array([30,60,45])
b2
```

```
array([30, 60, 45])
```

```
#使用vstack函數沿著增加列的方向連結
b3 = np.vstack([b,b2])
b3
```

```
array([[ 1,  2,  8],
       [ 4,  5,  8],
       [30, 60, 45]])
```

16. 分割

分割二維陣列的方法

```
#使用hsplit函數分割，建立2個二維陣列。
#範例是將第二個參數指定為[2]，所以第一個陣列或是2行，剩下的1行會自行成為一個陣列
first, second = np.hsplit(b3,[2])
first
```

```
array([[ 1,  2],
       [ 4,  5],
       [30, 60]])
```

```
second
```

```
array([[ 8],
       [ 8],
       [45]])
```

```
#使用vsplit函數沿著列方向分割
first1, second1 = np.vsplit(b3,[2])
first1
```

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

```
second1
```

```
array([[30, 60, 45]])
```

17. 轉置

二維陣列的列與行互換位置稱為轉置

```
#確認前面建立的陣列b
b
```

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

```
#2*3矩陣轉換成3*2矩陣
b.T
```

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

18. 增加維度

```
#將a陣列轉換成二維陣列。
#在指定列方向的切片指定np.newaxis，藉此增加維度
a[np.newaxis,:]
```

```
array([[1, 5, 4]])
```

```
#在指定行方向的切片指定np.newaxis，藉此追加列
a[:, np.newaxis]
```

```
array([[1],
       [5],
       [4]])
```

- 要增減維度也可以使用reshape，不過使用reshape方法得指定元素的數量，使用np.newaxis則不需要另外指定元素的數量

19. 建立網格資料

meshgrid函數會在繪製與平面上的點對應的等高線或是聚類熱圖時使用。從x座標、y座標的陣列取出元素，再組合這些元素，藉此產生所有點的座標資料

```
m = np.arange(0,4)
m
```

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

```
n = np.arange(4,7)
n
```

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

```
#在列方向與行方向產生m與n的網格資料
xx, yy = np.meshgrid(m,n)
xx
```

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

```
yy
```

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

- 第一個傳回值xx的部分，第一個參數的m依照第二個參數n的陣列長度沿著列方向複製，第二個傳回值yy的部分，則是第二個參數的n依照第一個參數m的陣列長度沿著行方向複製

NumPy的各項功能

```
import numpy as np

a = np.arange(3)
b = np.arange(-3,3).reshape((2,3))
c = np.arange(1,7).reshape((2,3))
d = np.arange(6).reshape((3,2))
e = np.linspace(-1,1,10)

print("a:",a)
print("b:",b)
print("c:",c)
print("d:",d)
print("e",e)
```

```
a: [0 1 2]
b: [[-3 -2 -1]
 [ 0  1  2]]
c: [[1 2 3]
 [4 5 6]]
d: [[0 1]
 [2 3]
 [4 5]]
e [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
  0.33333333  0.55555556  0.77777778  1.        ]
```

```
print("a:",a.shape)
print("b:",b.shape)
print("c:",c.shape)
print("d:",d.shape)
print("e:",e.shape)
```

```
a: (3,)
b: (2, 3)
c: (2, 3)
d: (3, 2)
e: (10,)
```

1. Universal Functions

Universal Functions是NumPy的超級工具之一，可一口氣轉換陣列的資料

```
li = [[-3,-2,-1],
    [0,1,2]]
new = []
for i , j in enumerate(li):
   new.append([])
   for k in j:
      new[i].append(abs(k))
new
```

```
[[3, 2, 1], [0, 1, 2]]
```

```
#使用NumPy輸出絕對值得情況
np.abs(b)
```

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

- np.abs函數可取得內部元素的計算結果

```
#sin函數
np.sin(e)
```

```
array([-0.84147098, -0.70169788, -0.52741539, -0.3271947 , -0.11088263,
        0.11088263,  0.3271947 ,  0.52741539,  0.70169788,  0.84147098])
```

```
#cos函數
np.cos(e)
```

```
array([0.54030231, 0.71247462, 0.84960756, 0.94495695, 0.99383351,
       0.99383351, 0.94495695, 0.84960756, 0.71247462, 0.54030231])
```

```
#以log函數計算以納皮爾數為底的自然對數
np.log(a)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
array([      -inf, 0.        , 0.69314718])
```

- -inf代表負無限大的意思。

```
#常用對數(log的底為10)可使用log10函數計算
np.log(c)
```

```
array([[0.        , 0.69314718, 1.09861229],
       [1.38629436, 1.60943791, 1.79175947]])
```

```
#確認自然對數的底e。exp函數的意思是ex
np.exp(a)
```

```
array([1.        , 2.71828183, 7.3890561 ])
```

2. Broadcast

Broadcast與Universal Functions 一樣，都能直接運算陣列的內部資料，是NumPy非常好用的功能。

利用加總陣列內部的純量，介紹Broadcast這項功能

```
#先確認a的內容
a
```

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

```
#在這個陣列的元素加10
a+10
```

```
array([10, 11, 12])
```

```
#確認陣列b的內容
b
```

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

```
#一維陣列a加二維陣列b
a + b
```

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

```
#將a轉換3*1的矩陣，再代入變數a1
a1 = a[:, np.newaxis]
a1
```

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

```
#讓a與a1相加
a+a1
```

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

```
#二維陣列c的各個元素減掉c元素平均值
#先確認c的內容
c
```

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

```
#建立元素扣掉平均值之後的陣列
c - np.mean(c)
```

```
array([[-2.5, -1.5, -0.5],
       [ 0.5,  1.5,  2.5]])
```

```
#確認陣列與純量的乘積與乘方運算
b * 2
```

```
array([[-6, -4, -2],
       [ 0,  2,  4]])
```

```
#各元素乘以三次方
b ** 3
```

```
array([[-27,  -8,  -1],
       [  0,   1,   8]])
```

```
#減法運算
b - a
```

```
array([[-3, -3, -3],
       [ 0,  0,  0]])
```

```
#乘法運算
a * b
```

```
array([[ 0, -2, -2],
       [ 0,  1,  4]])
```

```
#除法運算
a / c
```

```
array([[0.        , 0.5       , 0.66666667],
       [0.        , 0.2       , 0.33333333]])
```

```
#如果陣列的元素為0，一執行除法，元素之中就會出現下列代表無限大的inf(也會輸出Runtime Warning)
c / a
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWarning: divide by zero encountered in true_divide

array([[inf, 2. , 1.5],
       [inf, 5. , 3. ]])
```

```
#為了避免輸出Runtime Warning，介紹加入極小值的技巧，也就是先將1e-6 (10-6)這個非常小的數值存入除數的陣列再執行除法
c / (a+1e-6)
```

```
array([[1.00000000e+06, 1.99999800e+00, 1.49999925e+00],
       [4.00000000e+06, 4.99999500e+00, 2.99999850e+00]])
```

- 以0之外的數值除之的元素，計算結果幾乎都是相同的數值，但以0除之的元素會輸出非常大的數值。此種技巧常用於不輸出無限大的inf，卻需要逼近某個數值的情況使用

3. 點積

- 計算二維陣列b與一維陣列a的點積
- 點積可利用dot函數計算

```
np.dot(b,a)
```

```
array([-4,  5])
```

```
#Python3.5之後可利用@運算子計算
b @ a
```

```
array([-4,  5])
```

```
#二維陣列之間的點積
#2*3矩陣與3*2矩陣點積，輸出2*2矩陣
b @ d
```

```
array([[ -8, -14],
       [ 10,  13]])
```

```
#二維陣列之間的點積
#3*2矩陣與2*3矩陣的點積，傳回3*3矩陣
d @ b
```

```
array([[  0,   1,   2],
       [ -6,  -1,   4],
       [-12,  -3,   6]])
```

4. 判定、邏輯值

利用運算子比較陣列與純量，會以形狀相同的陣列輸出比較結果的布林值(True/False)

```
a > 1
```

```
array([False, False,  True])
```

```
b > 0
```

```
array([[False, False, False],
       [False,  True,  True]])
```

- 不管是一維陣列還是二維陣列，各元素與純量的比較結果，將以陣列方式傳回
- 也可利用布林值的陣列計算符合條件的元素有幾個

```
#先計算True的數量
np.count_nonzero(b>0)
```

```
2
```

```
#利用np.any輸出元素中是否含有True的結構
#b>0的結果是由2個True與4個False組成。元素之中的True超過一個以上，所以才輸出True這個結果
np.any(b>0)
```

```
True
```

```
#利用np.all確認所有元素是否為True
#由於元素之中有False，所以輸出False結果
np.all(b>0)
```

```
False
```

```
#介紹以上述的布林值陣列判斷元素是否符合條件，再以新陣列的方式輸出符合條件的元素
#只輸出b>0為True的元素
b[b>0]
```

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

```
#上述都是讓陣列與純量比較，其實陣列也能互相比較
#此比較是讓形狀相同的陣列互相比較，會逐一比較陣列的元素
b == c
```

```
array([[False, False, False],
       [False, False, False]])
```

```
#一維陣列與二維陣列相互比較
a == b
```

```
array([[False, False, False],
       [ True,  True,  True]])
```

- 在Broadcast說明，陣列的形狀不一致時，會依照Broadcast的規則調整形狀再比較

```
#多個陣列互相比較，再以位元運算輸出結果
(b == c) | (a == b)
```

```
array([[False, False, False],
       [ True,  True,  True]])
```

```
#可利用上述結果根據多個陣列的條件，從陣列取得元素
b[ (b == c) | (a == b)]
```

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

- 到目前為止都是將注意力放在元素
- 下面介紹確認陣列是否以相同元素組成的方法

```
#確認陣列是否以相同元素組成的方法
#np.allclose並非判斷所有的元素是否相同，而是判斷誤差的範圍
np.allclose(b,c)
```

```
False
```

```
#利用atol參數指定為絕對誤差
#將誤差定為10，所有元素也在誤差範圍之內，所以回傳True
#此種判斷方式在遇到需要忽略浮點數計算誤差的情況時，是非常方便的功能
np.allclose(b,c,atol=10)
```

```
True
```

5. 函數與方法

到目前都是利用NumPy的函數計算元素的平均值或總和

```
#計算a的元素總和會使用np.sum函數
np.sum(a)
```

```
3
```

```
#同樣的計算也能利用陣列的方法達成
a.sum()
```

```
3
```