Python
Using the Numpy Pacakge

馬誠佑

2025/05/02

numpy

- 為許多科學計算套件的基礎，可讓使用者容易建立向量（vector）、矩陣（matrix）並進行高效大量的資料運算。
- Command line安裝： pip install numpy
- numpy 可產生一或多維的向量（矩陣），並在運算時有非常優異的效能。

ndarray vs. list

- ndarray 是由許多相同資料型別的元素所組成（若為物件陣列則例外），list 元素可以是不同資料型別（data type）
- ndarray建立時大小是固定的，若更改大小則需新創建ndarray，list則可以是動態調整大小。
- numpy的矩陣運算效能是list所無法比擬的。

Numpy (ndarra

```python
import numpy as np

a = np.array([1, 2, 3, 4, 5])
b = np.array((6.23, 7.22, 8.76, 9.52, 10.0))
print(a)
print(b)
print(type(a))
print(type(b))
print('a.dtype = ', a.dtype)
print('b.dtype = ', b.dtype)
a[3] = 120
b[0] = 100.0
print(a)
print(b)
```

✓ 0.0s

```
[1 2 3 4 5]
[ 6.23  7.22  8.76  9.52 10.  ]
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
a.dtype =  int64
b.dtype =  float64
[  1   2   3 120   5]
[100.     7.22   8.76   9.52  10.  ]
```

```python
import numpy as np

a = np.array([[1, 2, 3, 4, 5],[6, 7, 8, 9, 10]])
print('a.ndim = ',a.ndim)
print('a.shape = ',a.shape)
print('a.size = ',a.size)
print('a.dtype = ',a.dtype)
print('a.itemsize = ',a.itemsize)
print('a.data = ',a.data)
```

✓ 0.0s

```
a.ndim =  2
a.shape =  (2, 5)
a.size =  10
a.dtype =  int64
a.itemsize =  8
a.data =  <memory at 0x108025790>
```

產生一維陣列

```python
import numpy as np

a = np.array([6,5,4,3,2])
b = np.arange(10)
c = np.arange(0, 10, 1.5, dtype = np.float64)
d = np.zeros(5)
e = np.ones(5)
print(a)
print(b)
print(c)
print(d)
print(e)
```

✓ 0.0s

```
[6 5 4 3 2]
[0 1 2 3 4 5 6 7 8 9]
[0.  1.5 3.  4.5 6.  7.5 9. ]
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
```

linspace

```
import numpy as np

lin = np.linspace(0,5,9)
print(lin)
lin2 = np.linspace(0,9,6)
print(lin2)
```

✓ 0.0s

```
[0.    0.625 1.25  1.875 2.5   3.125 3.75  4.375 5.   ]
[0.  1.8 3.6 5.4 7.2 9. ]
```

產生N維陣列

```python
import numpy as np

aa = np.array([[1, 2, 3, 4, 5],[6, 7, 8, 9, 10]])
bb = np.zeros([3,3])
ccc = np.ones([3,3,3])
print('aa=>\n',aa)
print('bb=>\n',bb)
print('ccc=>\n',ccc)
print('aa.ndim = ',aa.ndim)
print('aa.shape = ',aa.shape)
print('bb.ndim = ',bb.ndim)
print('bb.shape = ',bb.shape)
print('ccc.ndim = ',ccc.ndim)
print('ccc.shape = ',ccc.shape)
```

```
aa=>
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
bb=>
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
ccc=>
 [[[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. 1. 1.]]]
aa.ndim =  2
aa.shape =  (2, 5)
bb.ndim =  2
bb.shape =  (3, 3)
ccc.ndim =  3
ccc.shape =  (3, 3, 3)
```

numpy.full()

```
    a = np.full((3,3,3), 100)
    print('a =\n',a)
```

✓ 0.0s

```
a =
 [[[100 100 100]
  [100 100 100]
  [100 100 100]]

 [[100 100 100]
  [100 100 100]
  [100 100 100]]

 [[100 100 100]
  [100 100 100]
  [100 100 100]]]
```

Indexing & slicing & masking

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

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

✓ 0.0s

```
3
6
6
[5 4]
[[5 6]
 [8 9]]
```

```
mask = (aa % 2 == 0)
print(aa[mask])
mask2 = (aa > 5)
print(aa[mask2])
mask3 = (aa % 2 == 0) & (aa > 5)
print(aa[mask3])
```

✓ 0.0s

```
[2 4 6 8]
[6 7 8 9]
[6 8]
```

```
a = np.array([6,5,4,3,2])
fancy_index = np.array([0, 2, 4])
print(a[fancy_index])
```

✓ 0.0s

```
[6 4 2]
```

矩陣運算

```python
a = np.array([1,2,3,4])
b = np.array([5,6,7,8])
aa = np.array([[1,2,3], [4,5,6]])
bb = np.array([[7,8,9], [10,11,12]])

print('a+b = ',a + b)
print('aa+bb = ',aa + bb)
print('b-a = ',b-a)
print('bb-aa =', bb-aa)
print('a*b = ',a*b) # element-wise product
print('aa*bb = ',aa*bb) # element-wise product
print('b/a = ',b/a)
print('bb/aa = ',bb/aa)
print('np.dot(aa,bb.T) = ',np.dot(aa,bb.T))
```

```
a+b =  [ 6  8 10 12]
aa+bb =  [[ 8 10 12]
 [14 16 18]]
b-a =  [4 4 4 4]
bb-aa = [[6 6 6]
 [6 6 6]]
a*b =  [ 5 12 21 32]
aa*bb =  [[ 7 16 27]
 [40 55 72]]
b/a =  [5.         3.         2.33333333 2.        ]
bb/aa =  [[7.  4.  3. ]
 [2.5 2.2 2. ]]
np.dot(aa,bb.T) =  [[ 50  68]
 [122 167]]
```

Inner/outer product

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

print('np.inner(aa,bb) =\n',np.inner(aa,bb))
print('np.outer(aa,bb) =\n',np.outer(aa,bb))
```

✓ 0.0s

```
np.inner(aa,bb) =
 [[ 50  68]
 [122 167]]
np.outer(aa,bb) =
 [[ 7  8  9 10 11 12]
 [14 16 18 20 22 24]
 [21 24 27 30 33 36]
 [28 32 36 40 44 48]
 [35 40 45 50 55 60]
 [42 48 54 60 66 72]]
```

Reshape, concatenate, vstac
hstack

```
a = np.array([1,2,3,4])
b = np.array([5,6,7,8])
aa = np.array([[1,2],
               [3,4]])
bb = np.array([[5,6],
               [7,8]])
print('np.concatenate((a,b)) =\n',np.concatenate((a,b)))
print('np.vstack((a,b)) =\n',np.vstack((a,b)))
print('np.hstack((a,b)) =\n',np.hstack((a,b)))
print('np.concatenate((aa,bb)) =\n',np.concatenate((aa,bb)))
print('np.vstack((aa,bb)) =\n',np.vstack((aa,bb)))
print('np.hstack((aa,bb)) =\n',np.hstack((aa,bb)))
cc = np.array([[1,2,3,4,5,6],[7,8,9,10,11,12]])
print('cc.reshape(3,4) =\n',cc.reshape(3,4))
print('cc.reshape(2,3,2) =\n',cc.reshape(2,3,2))
```

```
np.concatenate((a,b)) =
 [1 2 3 4 5 6 7 8]
np.vstack((a,b)) =
 [[1 2 3 4]
 [5 6 7 8]]
np.hstack((a,b)) =
 [1 2 3 4 5 6 7 8]
np.concatenate((aa,bb)) =
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
np.vstack((aa,bb)) =
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
np.hstack((aa,bb)) =
 [[1 2 5 6]
 [3 4 7 8]]
cc.reshape(3,4) =
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
cc.reshape(2,3,2) =
 [[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

np.array_split(ary, indices_or_sections, axis=0)

```
a = np.array([1,2,3,4,5,6,7,8,9,10])
splited2 = np.array_split(a, 2)
print('splited2 =\n',splited2)
splited3 = np.array_split(a, 3)
print('splited3 =\n',splited3)
splited_len3 = np.array_split(a, [3])
print('splited[3] =\n',splited_len3)
splited325 = np.array_split(a, [3,5,6])
print('splited325 =\n',splited325)
```

✓ 0.0s

```
splited2 =
 [array([1, 2, 3, 4, 5]), array([ 6,  7,  8,  9, 10])]
splited3 =
 [array([1, 2, 3, 4]), array([5, 6, 7]), array([ 8,  9, 10])]
splited[3] =
 [array([1, 2, 3]), array([ 4,  5,  6,  7,  8,  9, 10])]
splited325 =
 [array([1, 2, 3]), array([4, 5]), array([6]), array([ 7,  8,  9, 10])]
```

```
aa = np.array([[1,2,3,4],
               [5,6,7,8]])
# axis = 0 水平方向切一刀分割
print('np.array_split(aa, 2, axis = 0) =\n',
      np.array_split(aa, 2,0))
# axis = 1 垂直方向切一刀分割
print('np.array_split(aa, 2, axis = 1) =\n',
      np.array_split(aa, 2,1))
```

✓ 0.0s

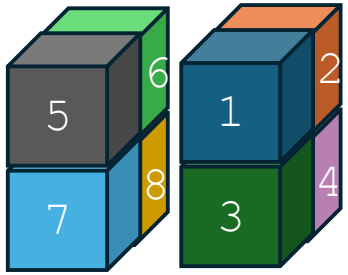
```
np.array_split(aa, 2, axis = 0) =
 [array([[1, 2, 3, 4]]), array([[5, 6, 7, 8]])]
np.array_split(aa, 2, axis = 1) =
 [array([[1, 2],
        [5, 6]]), array([[3, 4],
        [7, 8]])]
```

np.hsplit, np.vs

```
aa = np.array([[1,2,3,4],
               [5,6,7,8]])
print('np.hsplit(aa, 2) =\n',np.hsplit(aa, 2))
print('np.vsplit(aa, 2) =\n',np.vsplit(aa, 2))
```

✓ 0.0s

```
np.hsplit(aa, 2) =
 [array([[1, 2],
        [5, 6]]), array([[3, 4],
        [7, 8]])]
np.vsplit(aa, 2) =
 [array([[1, 2, 3, 4]]), array([[5, 6, 7, 8]])]
```




```
aaa = np.array([1,2,3,4,5,6,7,8]).reshape(2,2,2)
print('aaa=\n',aaa)
print('np.hsplit(aaa, 2) =\n',np.hsplit(aaa, 2))
print('np.vsplit(aaa, 2) =\n',np.vsplit(aaa, 2))
```

✓ 0.0s

```
aaa=
 [[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
np.hsplit(aaa, 2) =
 [array([[[1, 2]],

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

        [[7, 8]]])]
np.vsplit(aaa, 2) =
 [array([[[1, 2],
         [3, 4]]]), array([[[5, 6],
         [7, 8]]])]
```

numpy.resize(arr, shape)

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

print ('a:')
print (a)
print ('a.shape:')
print (a.shape)
b = np.resize(a, (3,2))

print ('b:')
print (b)
print ('b.shape:')
print (b.shape)

print ('c:')
c = np.resize(a,(3,3))
print (c)
```

```
a:
[[1 2 3]
 [4 5 6]]
a.shape:
(2, 3)
b:
[[1 2]
 [3 4]
 [5 6]]
b.shape:
(3, 2)
c:
[[1 2 3]
 [4 5 6]
 [1 2 3]]
```

如果新array大小大於原始大小，則多出的位置為原始array中的元素的副本。

統計計算

```python
data = np.array([1, 5, 18, -3, 9, 12, 15])
#最大值
print('最大值：',data.max())
#最大值的index
print('最大值的index:',data.argmax())
#最小值
print('最小值:',data.min())
#最小值的index
print('最小值的index:', data.argmin())
#sum
print('總和：',data.sum())
# 平均值
mean = np.mean(data)
print('平均值:', mean)
# 中位數
median = np.median(data)
print('中位數:', median)
# 標準差
std_dev = np.std(data)
print('標準差:', std_dev)
```

```
最大值： 18
最大值的index: 2
最小值: -3
最小值的index: 3
總和： 57
平均值: 8.142857142857142
中位數: 9.0
標準差: 7.0189248551647125
```

```python
import numpy as np

a = np.array([[1,2,3],[4,5,6]])
print(a.max())              # 获取整个矩阵的最大值 结果： 6
print(a.min())              # 结果： 1

# 可以指定关键字参数axis来获得行最大（小）值或列最大（小）值
# axis=0 行方向最大（小）值，即获得每列的最大（小）值
# axis=1 列方向最大（小）值，即获得每行的最大（小）值
# 例如

print(a.max(axis=0))
# 结果为 [4 5 6]

print(a.max(axis=1))
# 结果为 [3 6]

# 要想获得最大最小值元素所在的位置，可以通过argmax函数来获得
print(a.argmax(axis=1))
# 结果为 [2 2]
```

```python
import numpy as np

a = np.array([[1,2,3],[4,5,6]])
print(a.var())              # 结果 2.91666666667

print(a.var(axis=0))        # 结果 [ 2.25   2.25   2.25]
print(a.var(axis=1))        # 结果 [ 0.66666667   0.66666667]
```

```python
import numpy as np

a = np.array([[1,2,3],[4,5,6]])
print(a.std())              # 结果 1.70782512766

print(a.std(axis=0))        # 结果 [ 1.5   1.5   1.5]
print(a.std(axis=1))        # 结果 [ 0.81649658   0.81649658]
```

numpy.view() & numpy.copy()

```
a = np.array([1, 2, 3, 4, 5, 6])
b = a
b[3] = 100
print('a after we changed b:',a)
c = a[2:5]
c[1] = 200
print('a after we changed c:',a)
b.shape = (2,3)
print('a after we changed b\'s shape:',a)
d = a.view()
d.shape = (3,2)
print('d.shape:',d)
print('a after we changed d\'s shape:',a)
d[1,1] = 300
print('a after we changed d:',a)
```

✓ 0.0s

```
a after we changed b: [  1   2   3 100   5   6]
a after we changed c: [  1   2   3 200   5   6]
a after we changed b's shape: [[  1   2   3]
 [200   5   6]]
d.shape: [[  1   2]
 [  3 200]
 [  5   6]]
a after we changed d's shape: [[  1   2   3]
 [200   5   6]]
a after we changed d: [[  1   2   3]
 [300   5   6]]
```

```
a = np.array([1, 2, 3, 4, 5, 6])
b = a.copy()
b[3] = 100
print('b after we changed b:',b)
print('a after we changed b:',a)
b.shape = (2,3)
print('b after we changed b\'s shape:',b)
print('a after we changed b\'s shape:',a)
```

✓ 0.0s

```
b after we changed b: [  1   2   3 100   5   6]
a after we changed b: [1 2 3 4 5 6]
b after we changed b's shape: [[  1   2   3]
 [100   5   6]]
a after we changed b's shape: [1 2 3 4 5 6]
```

numpy.flatten & numpy.ravel

- ndarray.flatten(order='C') order：'C' -- 按行，'F' -- 按列，'A' – 原順序，'K' -- 元素在記憶體中的出現順序。

```
import numpy as np

a = np.arange(9).reshape(3,3)
print('a=\n',a)
print('a.flatten() =\n',a.flatten())
print('a.flatten(order = "F") =\n',a.flatten(order = 'F'))
```

✓ 0.0s

```
a=
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
a.flatten() =
 [0 1 2 3 4 5 6 7 8]
a.flatten(order = "F") =
 [0 3 6 1 4 7 2 5 8]
```

Broadcast

- 廣播（Broadcast）是 numpy 對不同形狀（shape）的陣列進行數值計算的方式， 對陣列的算術運算通常在相應的元素上進行。
- 如果兩個陣列 a 和 b 形狀相同，即滿足 a.shape == b.shape，那麼 a*b 的結果就是 a 與 b 陣組對應位相乘。 這要求維數相同，且各維度的長度相同。
- 當運算中的 2 個陣列的形狀不同時，numpy 將自動觸發廣播機制。

```python
import numpy as np

a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print ('a*b:', c)
a = np.array([1,2,3,4])
b = np.array([[0,0,0,0],
              [10,10,10,10],
              [20,20,20,20],
              [30,30,30,30]])
c = a * b
print ('a*b:\n',c)
k = np.array([
              [10,10,10,10]
              ])
print('b+k =\n',b+k)

d = a + b
print('a+b =\n', d)
e = np.array([1])
f = a + e
print('a+e =\n',f)
```

```
a*b: [ 10  40  90 160]
a*b:
 [[  0   0   0   0]
 [ 10  20  30  40]
 [ 20  40  60  80]
 [ 30  60  90 120]]
b+k =
 [[10 10 10 10]
 [20 20 20 20]
 [30 30 30 30]
 [40 40 40 40]]
a+b =
 [[ 1  2  3  4]
 [11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
a+e =
 [2 3 4 5]
```

- **廣播的規則：**
  - **讓所有輸入陣列都向其中形狀最長的陣列看齊，形狀中不足的部分都通過在前面加 *1* 補齊。**
  - **輸出陣列的形狀是輸入數位形狀的各個維度上的最大值。**
  - **如果輸入陣列的某個維度和輸出數位的對應維度的長度相同或者其長度為 *1* 時，這個數位能夠用來計算，否則出錯。**
  - **當輸入陣列的某個維度的長度為 *1* 時，沿著此維度運算時都用此維度上的第一組值。**
- **簡單理解：對兩個陣列，分別比較他們的每一個維度（若其中一個陣列沒有當前維度則忽略），滿足：**
  - **陣列擁有相同形狀。**
  - **當前維度的值相等。**
  - **當前維度的值有一個是 *1*。**
  - **若條件不滿足，拋出「*ValueError：frames are not aligned*」異常。**

Numpy documentation

- NumPy Documentation
- NumPy 教程 | 菜鸟教程 (runoob.com)

HW8

1. 請練習numpy package (40分)
   a. 請產生一個 3 x 3的ndarray 命名為A, 其中元素的值為1~30 的隨機整數，並將其印出。
   b. 請生成一個3 x 3的ndarray命名為B, 其中元素的值全為2，並將其印出。
   c. 請生成一個 2 x 6的ndarray 命名為C, 其中元素的值依序為1~12，並將其印出。
   d. 請計算 A * B並印出結果
   e. 請計算A dot C並印出結果
   f. 請計算A的mean並印出結果
   g. 請找出A的最大值並印出結果
   h. 請計算A的標準差並印出結果
   i. 請印出A的flatten按列順序
   j. 請印出A依axis為 0的最大值

2. 練習numpy package (40分)

a. 產生一個長度為10的一維ndarray 並命名為A，其中元素值依序為3~12並將其印出

b. 產生一個長度為10的一維ndarray並命名為B，其中元素值依序為10~1並將其印出

c. 請將A變更為2x5的矩陣並將其印出

d. 請將B變更為5X2的矩陣並將其印出

e. 請將A與B 依照水平方向concatenate起來並將其印出（需改變B的形狀但不可動到B）並將其印出

f. 請將A與B 依照垂直方向concatenate起來並將其印出 （需改變A的形狀但不可動到A） 並將其印出

g. 請將e的結果以垂直分割成2個ndarray 並將其印出(不可動到e)

h. 請將f的結果以水平分割成2個ndarray 並將其印出(不可動到f)

i. 請將A按行拉直並將其印出

j. 請將B按列拉直並將其印出

3. 請隨機生成一長度為100的ndarray, 其中元素的值為1~100的隨機值 （20分）

a. 請印出該ndarray的中位數
b. 請印出該ndarray的變異數
c. 請印出最大值的index
d. 請印出最小值的index
e. 請印出 >35 且 <=90的元素之index