NumPy 陣列的算術運算

簡報閱讀 範例與作業 問題討論 學習心得(完成)

重要知識點

- 正確的使用 NumPy 中的算術運算及
  式操作
- 了解 NumPy 陣列與數學矩陣的關係
- 知道 NumPy 與 SciPy 差異為何

算術運算是指一般數字間的加、減、乘、除或次方等等的運算，在 NumPy 當中也有提供。NumPy 的算術運算可以使用運算子或是函式來實現：

算術運算	運算子	函式
加法	+	np.add(...)
減法	-	np.subtract(a,b)
乘法	*	np.multiply(a,b)
除法	/	np.divide(a,b)
次方	**	np.power(a,b)

NumPy 陣列運算 - 四則運算

運算的時候，陣列的形狀 (shape) 必須相同，或是遵循廣播 (broadcasting) 規則，才能正確進行 element-wise 運算。規則如下：

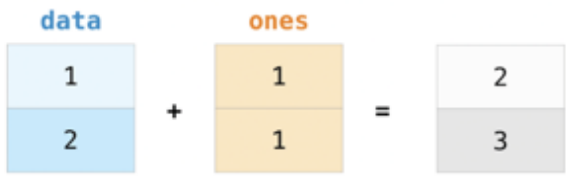
- 兩個陣列形狀完全相同
- 比較兩個陣列的維度，如果維度的形狀相同的話，可以進行廣播
- 比較兩個陣列的維度，其中一個維度為1的話，可以進行廣播

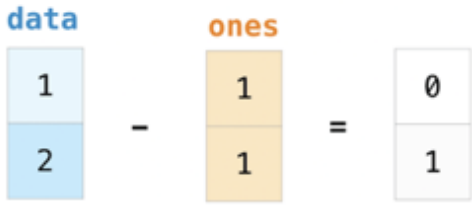
相同大小的陣列運算

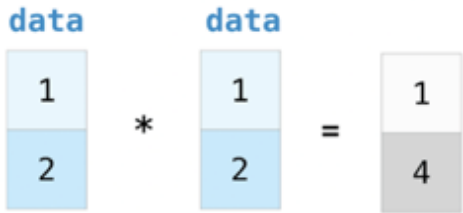
相同大小的陣列運算會符合「對齊」運算的特性：

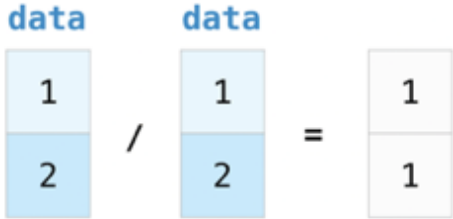
```
a = np.array( [20,30,40,50] )
b = np.arange( 4 )

print(a + b)
# [20 31 42 53]
print(a * b)
# [ 0 30 80 150]
```









常術與陣列運算會符合「廣播（Broadcast）」運算的特性，廣播特並會將常數補齊成多維的陣列。

```
import numpy as np

a = np.array( [20,30,40,50] )

print(a - 2)
# [18 28 38 48]
print(a / 10)
# [2. 3. 4. 5.]
```

不同大小的陣列運算

不同大小的陣列運算也會符合「廣播」運算的特性：

```
import numpy as np

data = np.array([[1, 2], [3, 4], [5, 6]])
ones_row = np.array([[1, 1]])

print(data + ones_row)
# array([[2, 3],
#        [4, 5],
#        [6, 7]])
```

陣列運算與容器運算的差異

陣列有「對齊」跟「廣播」兩種重要的運算特性，這個其實就是我們講的向量運算特性。換句話說，在向量的運算中，是以「整組」為單位在進行運算，這是和 Python 容器最大的差異之一。

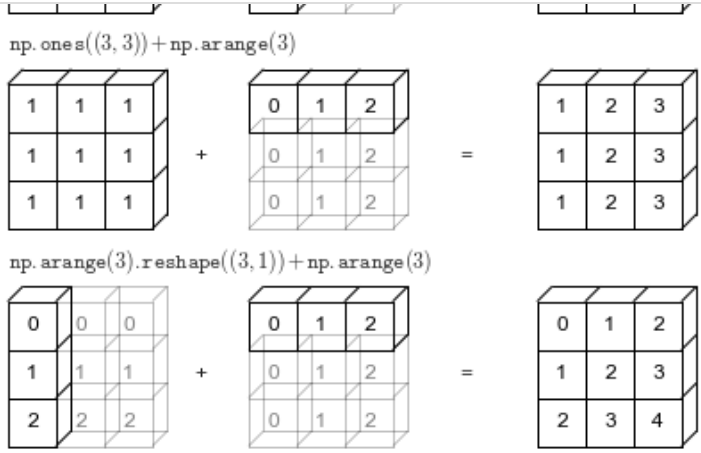
以「所有元素 +1」這個例子來看，兩種截然不同的做法：

```
a = np.array( [20,30,40,50] )
print(a + 1)
# array([21 31 41 51])
```

```
a = [20, 30, 40, 50]
b = []
for i in a:
  b.append(i+1)
print(b)
# [21, 31, 41, 51]
```

廣播其實很複雜

在前面的例子，我們都是從一維或常數的方法示範廣播特性。不過其實多維陣列間的運算也是[illegible]的特性，建議直接從圖片著手：



補充：更多廣播的細節，可以詳閱**參考資料**。

陣列與矩陣

講到這邊，你可能會思考「為什麼 Python 需要有 NumPy 這個套件呢？他幫我們解決的什麼問題嗎？」

NumPy 是為了彌補 Python 在數學運算上的不足而誕生，原有的 Python 僅有「程式邏輯」的特性，對於數學運算的支援並沒有特別的優勢。而 NumPy 則是把數學中基本的運算單位「向量」和「矩陣」這種類型在 Python 中實現，一維陣列就是數學中的向量、二維陣列可以用來實現數學當中的矩陣。

用於陣列的矩陣運算

在 NumPy 中的二維陣列支援了許多的矩陣

```
a = np.array([[1.0, 2.0], [3.0, 4.0]])
# [[1. 2.]
#  [3. 4.]]
y = np.array([[5.], [7.]])
# [[5.]]
#  [7.]]
```

```
a.transpose()
# array([[1., 3.],
#        [2., 4.]])
np.linalg.inv(a)
# array([[-2. ,  1. ],
#        [ 1.5, -0.5]])
np.trace(u)
# 2.0
np.linalg.solve(a, y)
# array([[-3.],
#        [ 4.]])
```

補充：矩陣運算與線性代數

在舊版本的 NumPy 當中，甚至有一種 np.matrix() 的結構特別用來代表矩陣。不過後來被整併到 array() 當中，並且把大部分的方法都收入在 Linear algebra (numpy.linalg) 當中。

NumPy 陣列運算 – 次方 np.power()

次方的運算，跟四則運算一樣，也要遵循上
則，才能成功進行運算。語法如下表：

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

NumPy 陣列運算 – 平方根 np.sqrt()

基本語法: np.sqrt(a)，對陣列進行 element-wise 的平方根。

範例：

```
[18]: np.sqrt(4)
[18]: 2.0
[19]: np.sqrt(a)
[19]: array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798])
```

```
1  np.sqrt(4)
2  np.sqrt(a)
```

NumPy 陣列運算 – 歐拉數 (Euler's number) 及指數函式 np.exp()

NumPy 提供歐拉常數 *e* **(np.e)**，以及指數函式 **np.exp()**，表示 e^X。

範例：

```
[20]: np.e
[20]: 2.718281828459045
[21]: np.exp(1)
[21]: 2.718281828459045
[22]: np.exp(np.arange(5))
[22]: array([ 1.        ,  2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

```
3  np.exp(np.arange(5))
```

NumPy 陣列運算 – 對數函式

log 函式如下表：

函式	
np.log(x)	底數為e
np.log2(x)	底數為2
np.log10(x)	底數為10
np.log1p(x)	底數為e，計算log(1+x)

若要使用其他底數，可以用下列的方法 (以底數 3 為例)。

```
[24]: np.log(9)/np.log(3)

[24]: 2.0
```

```
1  np.log(9)/np.log(3)
```

若是 log(負數) 則會產生 nan 常數，NaN / NAN 為 nan (not a number) 的別名。

```
np.log([-1, 1, 2])

c:\python\python36_64\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid
tered in log
  """Entry point for launching an IPython kernel.

array([       nan, 0.        , 0.69314718])
```

NumPy 陣列運算 – 取近似值

函式	常用語法	說明
round(), around()	ndarray.round(decimals=0) numpy.round(a, decimals=0) numpy.around(a, decimals=0)	在 Rounding 的方法部分，與 Python 同樣採用 IEEE 754 規範，四捨、五取最近偶數、六入，而非我們一般講的四捨五入。 round 與 around 用法及結果相同
rint()	numpy.rint(a)	Round至最近的整數
trunc()	numpy.trunc(a)	無條件捨去小數點
floor()	numpy.floor(a)	向下取整數
ceil()	numpy.ceil(a)	向上取整數
fix()	numpy.ceil(a)	向0的方向取整數

NumPy 陣列運算 – 取絕對值：np.abs(), np.absolute(), np.fabs()

- np.abs() 是 np.absolute() 的簡寫，兩者完全相同；np.fabs() 的差異在於無法處理複數 (Complex)。
- 如果傳入複數至 fabs() 的話則會產生錯誤。

NumPy 陣列運算 – 點積 (dot product)

- 進行點積運算須注意形狀必須注意形狀 (shape)。若是兩個向量的點積，兩個向量的元素數目也須相同，或其中一個數目為 1 (廣播)。
- 若是兩個多維陣列 (矩陣) 的點積，則中間兩個大小要相同才能進行點積，例如：(2,3)·(3,4)→ 成為 (2,4)

- 如果形狀不符合則無法進行點積。
- 點積運算示意圖：

$$A \cdot B = \begin{bmatrix} a_{11}*b_{11}+a_{12}*b_{21}+a_{13}*b_{31} & a_{11}*b_{12}+a_{12}*b_{22}+a_{13}*b_{32} & a_{11}*b_{13}+a_{12}*b_{23}+a_{13}*b_{33} & a_{11}*b_{14}+a_{12}*b_{24}+a_{13}*b_{34} \\ a_{21}*b_{11}+a_{22}*b_{21}+a_{23}*b_{31} & a_{21}*b_{12}+a_{22}*b_{22}+a_{23}*b_{32} & a_{21}*b_{13}+a_{22}*b_{23}+a_{23}*b_{33} & a_{21}*b_{14}+a_{22}*b_{24}+a_{23}*b_{34} \end{bmatrix}$$

SciPy 與 NumPy 的同與異

很多人可能有聽過 SciPy 這個套件，跟 NumPy 很常一起被提及。在早期 NumPy 主要提供了「向量」的資料結構，方便在 Python 有更好的運算運算特性。而 SciPy 主要基於 NumPy 的向量結構，提供了一系列的科學運算方法，包含統計、三角函數、線性代數、傅立葉轉換圖像等較高階的科學運算。

知識點回顧

- 正確的使用 NumPy 中的算術運算及相關函式操作
- 了解 NumPy 陣列與數學矩陣的關係
- 知道 NumPy 與 SciPy 差異為何

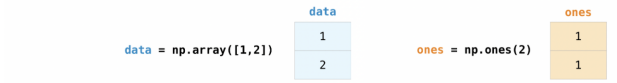
參考資料

Bsic array operations

網站：numpy

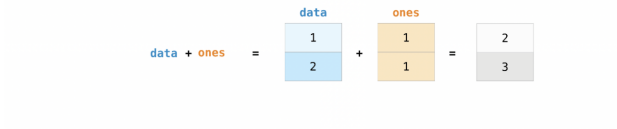
來自 NumPy 的官方網站的教學，透過大量的示意圖解釋陣列的運算特性。

Once you've created your arrays, you can start to work with them. Let's say, for example, that you've created two arrays, one called "data" and one called "ones"



You can add the arrays together with the plus sign.

```
>>> data = np.array([1, 2])
>>> ones = np.ones(2, dtype=int)
>>> data + ones
array([2, 3])
```



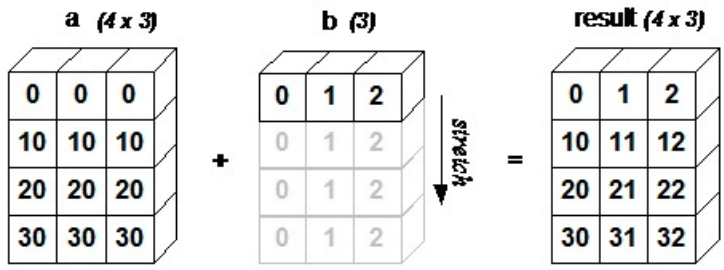
You can, of course, do more than just addition!

```
>>> data - ones
array([0, 1])
>>> data * data
array([1, 4])
>>> data / data
array([1., 1.])
```

numpy的廣播(Broadcasting)

網站：[知乎]

廣播是 NumPy 中的運算特性之一，而購自動將範圍小的常數補齊成範圍較大的陣列。但除此之外，背後也有更多複雜的機制。



下一步：閱讀範例與完成作業