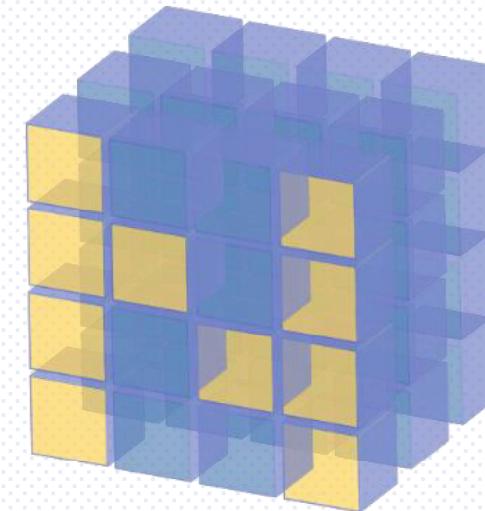


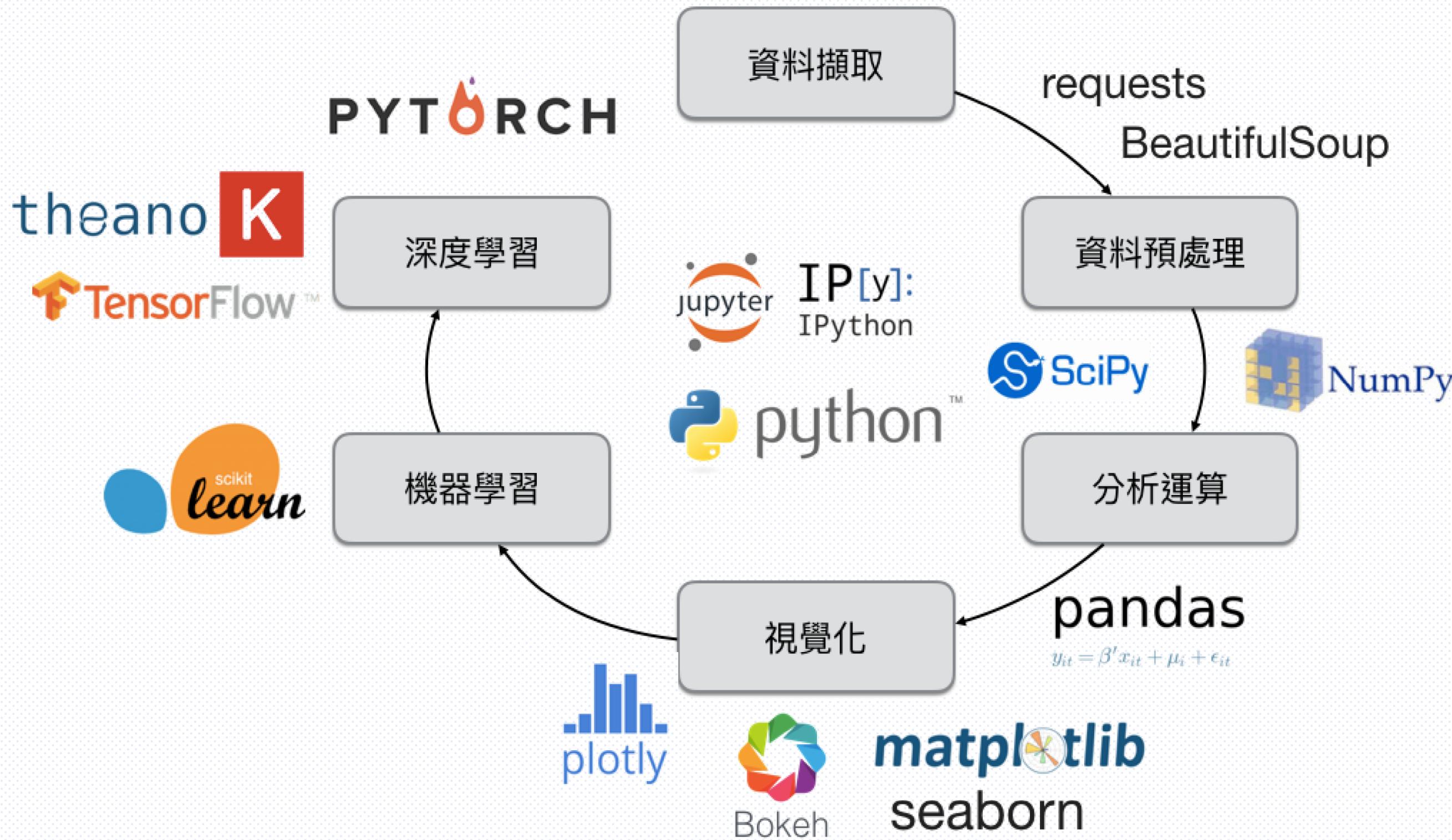
Numpy 套件

Data Analysis Package



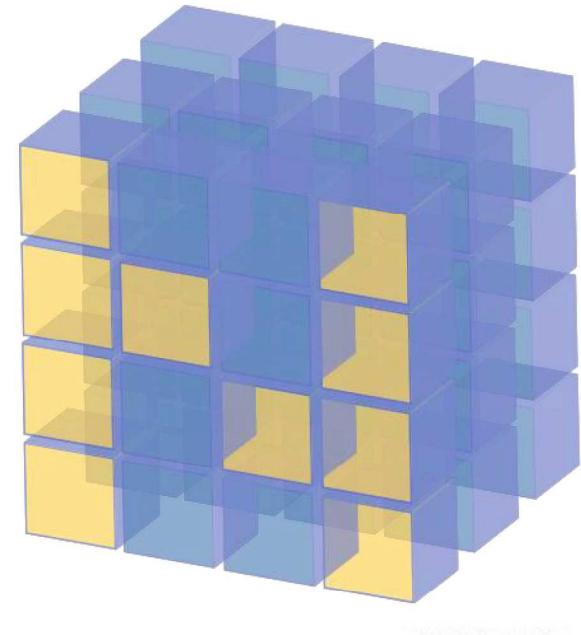
NumPy

Python Data Analysis



NumPy 簡介

- ◆ 對於資料科學運算的程式語言，必須具備「有效率處理大量資料」及「高速進行陣列運算」兩特點。
- ◆ 陣列有多個資料，在Python以串列(list)或數組(tuple)呈現，串列與數組可藉由巢狀構造來表現多維陣列。
- ◆ 不過串列與數組在處理大量資料時，其處理速度過慢。
- ◆ Python本身的功能與標準函式庫上有所不足，而Numpy便可以克服這個缺點，高速地進行陣列運算。
- ◆ 藉由瞭解NumPy的基本知識，對於接下來的Pandas，Matplotlib也有幫助。



NumPy

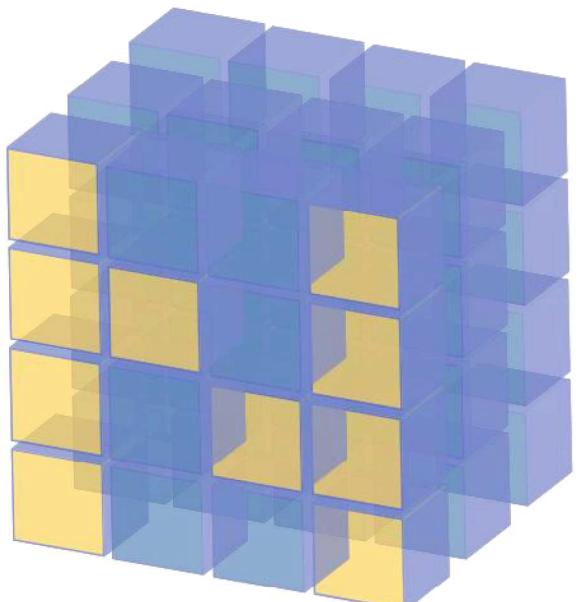


NumPy 簡介

- ◆ Numpy 是因應科學運算而誕生的函式庫，它使用獨特的多維陣列(ndarray)的資料結構做為運算單位， ndarray 可以說是NumPy高速運算的核心。以下是Numpy官網上對ndarray的定義：

An ndarray is a (usually fixed-size) multidimensional container of items of the same type and size.

- ◆ 也就是說，Numpy是由相同型別(type)和相同大小(size)的元素構成的多維容器。ndarray 只能儲存具有相同型別的元素，此外各維的元素數必須固定。
- ◆ NumPy的底層是用C或Fortran語言撰寫的，在處理NumPy陣列時，這些運算可以用經過最佳化的NumPy函式來處理，它們會將繁重的工作委託給底層，做到高速運算。



NumPy

NumPy 簡介

- ◆ Numpy是Python的一個擴充程式庫，代表 "Numeric Python"，是由多維陣列物件和用於處理陣列的函式集合組成，支援高階大量的維度陣列與矩陣運算，此外也針對陣列運算提供大量的數學函式庫。
- ◆ 對於陣列運算，Numpy提供了大量的數學函式庫。因此，只要是針對陣列或矩陣運算的演算法，就能使用Numpy處理，其執行效率幾乎都可以與編譯過的等效C語言程式碼一樣快速。
- ◆ Numpy的核心功能是"ndarray" (即n-dimensional array，多維陣列) 的資料結構，是一個多維度、同性質、固定大小的陣列物件。[1 D ARRAY:](#)

C	O	D	I	N	G	E	E	K
0	1	2	3	4	5	6	7	8

single row of elements

[2 D ARRAY:](#)

		col 0	col 1	col 2
row 0	i \ j	0	1	2
	0	A	A	A
	1	B	B	B
2	C	C	C	

column

array elements

rows

體驗 NumPy 的運算速度

- ◆ 底下產生1000000個數值，分別以ndarray(程式中的a變數)及Python的list(程式中的b變數)存放。
- ◆ 分別比較「將所有數值加總」需要的時間，由下面的結果可以清楚看到兩者處理速度的落差，如果資料量更多，結構更複雜，用NumPy更可看出高速運算的優勢。

```
import time
import numpy as np

def calculatetime():
    a=np.random.rand(1000000)
    b=list(a)
    start_time=time.time()
    for _ in range(100):
        sum1=np.sum(a)

    print("Using NumPy\t%f sec" % (time.time()-start_time) )
    start_time=time.time()
    for _ in range(100):
        sum2=sum(b)

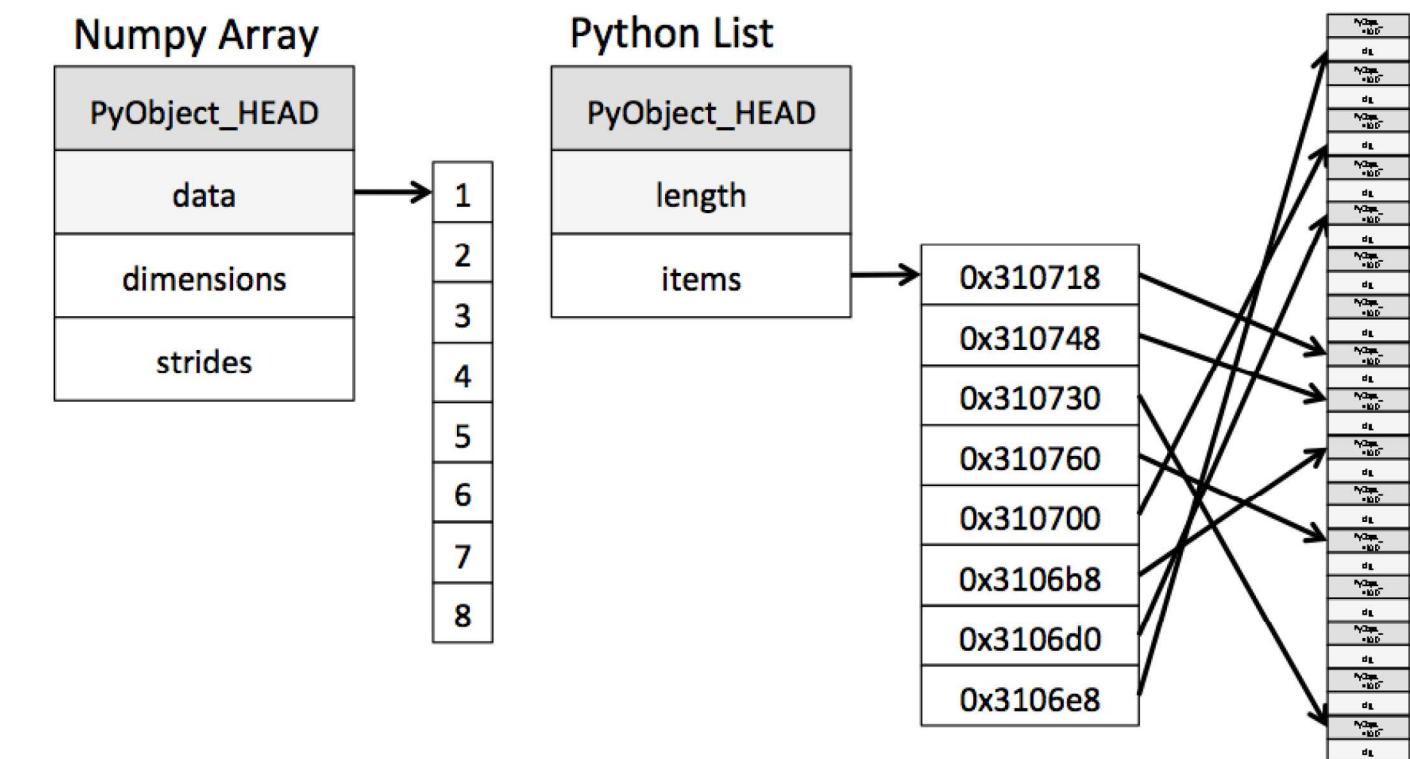
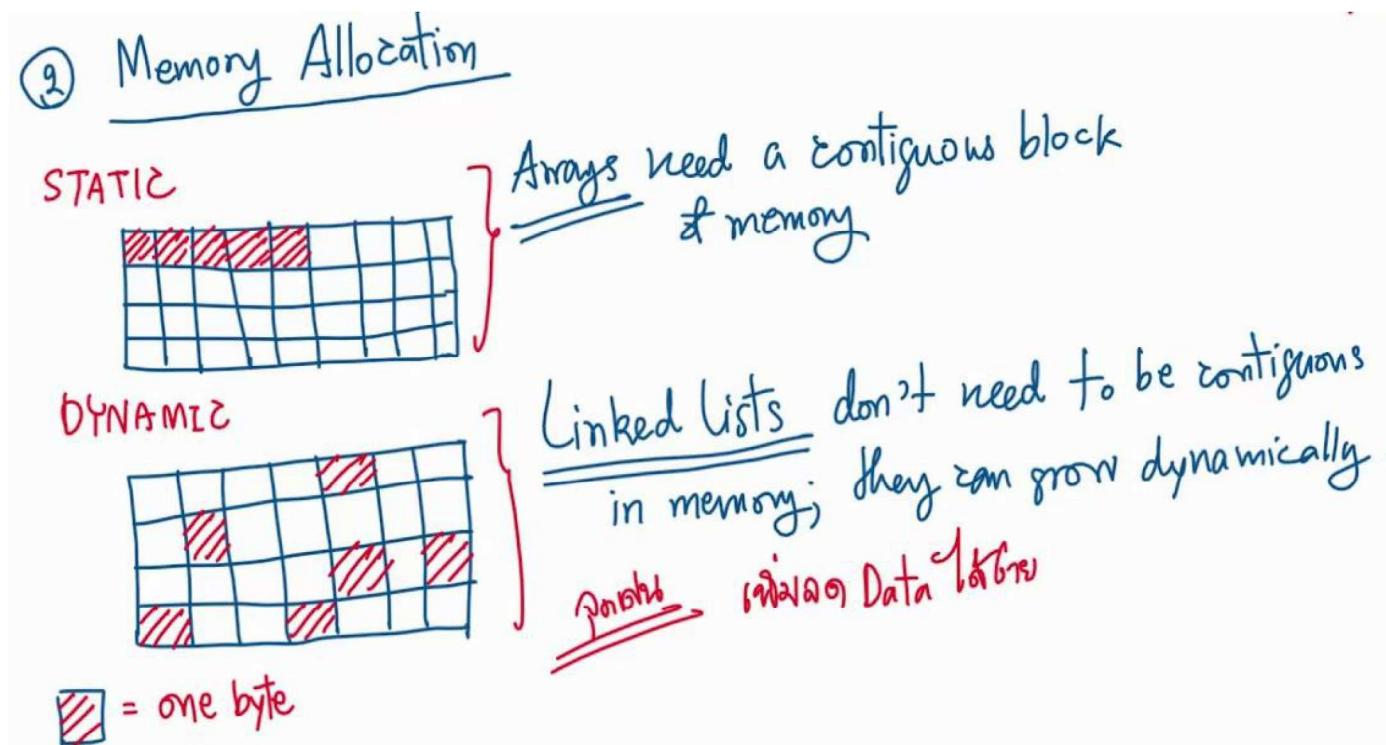
    print("Not using NumPy\t %f sec" % (time.time()-start_time))

calculatetime()
```

```
Using NumPy      0.078131 sec
Not using NumPy  4.615678 sec
```

NumPy 為何快速

- ◆ Numpy之所以能高速處理大量資料，是由於將多維陣列物件 ndarray 作為基本的資料儲存格式，ndarray 儲存的「資料」、以及稱為「metadata」(維度、資料型別等)的資訊組成。
- ◆ Python裡如「list」這樣的物件，各個元素分散存於記憶體的各處，因此在存取資料時便會發生間接成本(花費額外時間)
- ◆ 而在ndarray中，資料緊密地配置於系統的記憶體(RAM)的一個區域，因此能高速實作陣列運算，並能將資料有效率地讀出。

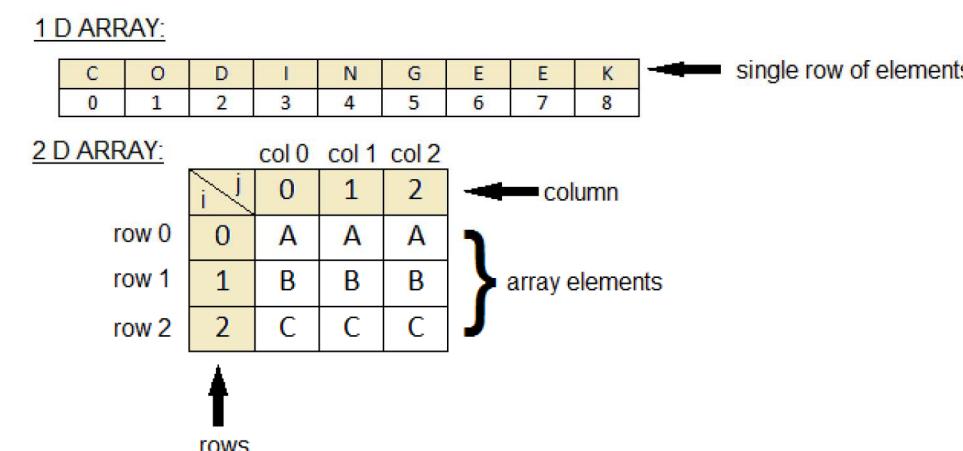


ndarray的產生

- ◆ NumPy為了更有效率地處理大量資料，使用了名為 ndarray 的物件，能表現一維或多維陣列。
- ◆ 我們可以使用 Numpy 中的 array() 函式，以串列或數組的資料產生 ndarray。
- ◆ 以下的例子裡，X與Y是完全相同的。

```
import numpy as np
X=np.array(([1,2,3],[4,5,6])) # 數組(tuple)轉成ndarray
Y=np.array([(1,2,3),(4,5,6)]) # 串列(list)轉成ndarray
print(X)
print(Y)
```

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



Numpy 的 ndarray

- ◆ 一維串列轉成一維陣列。

```
import numpy as np  
data1=[1,2,3,4,5]  
array1=np.array(data1)  
print(array1)
```

```
[1 2 3 4 5]
```

- ◆ 二維串列轉成二維陣列(矩陣)。

```
import numpy as np  
data2=[[1,2,3,4,5],[6,7,8,9,10]]  
array2=np.array(data2)  
print(array2)
```

```
[[ 1  2  3  4  5]  
 [ 6  7  8  9 10]]
```

- ◆ 在括號內加上dtype=資料類型，可以設定我們需要的資料類型。

```
import numpy as np  
data=[1,2,3,4,5]  
array1=np.array(data)  
array2=np.array(data, dtype=float)  
array3=np.array(data, dtype=bool)  
print(array1)  
print(array2)  
print(array3)
```

```
[1 2 3 4 5]  
[1. 2. 3. 4. 5.]  
[ True  True  True  True  True]
```

建立ndarray的函式

- ◆ 我們可以使用array()、ones()、zeros()、arange()、linspace()等函式來建立ndarray型態。
- ◆ ones()及zeros()函式以串列或數組所設定列與欄的個數產生元素為0或1的ndarray，資料型態預設為float64。
- ◆ arange() 函式類似於range()函式；dtype 用於指定資料型別，若不指定將根據start或stop的數值型別進行設定。

arange([start,] stop[, step,] [, dtype])

- ◆ 接下來可使用linspace()函式，用21個數字將0~100均分。

```
1 import numpy as np
2 print(np.zeros((3,5)))
3 print()
4 print(np.ones((3,2)))
5 print()
6 print(np.arange(3,20,2))
7 print()
8 print(np.linspace(0,100, 21))
```

```
[[0.  0.  0.  0.  0.]
 [0.  0.  0.  0.  0.]
 [0.  0.  0.  0.  0.]]

[[1.  1.]
 [1.  1.]
 [1.  1.]]

[ 3  5  7  9 11 13 15 17 19]

[  0.    5.   10.   15.   20.   25.   30.   35.   40.   45.   50.   55.   60.   65.
  70.   75.   80.   85.   90.   95.  100.]
```

ndarray 陣列物件相關屬性

◆ 以下介紹ndarray常用的屬性(attributes)，要注意的是這些屬性均要是ndarray的型態才可以使用。

屬性	說明
ndim	顯示ndarray有幾維(dimension)，不過來說是axis(軸)的數量
shape	用tuple顯示ndarray的形狀
T	回傳轉置後的陣列，例如shape為(2,3)的二維陣列轉置後會變成shape(3,2)，簡單說就是行變列，列變行 (transform)
data	顯示陣列資料從何處理始，會傳回(memory at 0x00020ba5654678>這樣的記憶體位置
dtype	顯示ndarray中元素的資料型別
flat	可將ndarray轉換成一維陣列
imag	顯示ndarray的虛數部分(imaginary part)
real	顯示ndarray的實數部分(real part)
size	顯示ndarray的總元素數量
itemsize	以byte為單位元元，顯示每個元素的記憶體使用量。舉例來說若a陣列的dtype是int32，則a.itemsize=4 (32bits/8)
nbytes	以byte為單位，顯示ndarray所有元素的記憶體使用量
strides	步長，用tuple顯示往各維移動一個相輿元素時，需要的bytes數。每移一步需要的bytes數就是上面的「陣列物件.itemsize」的值

ndarray 陣列物件相關屬性

- ◆ 針對屬性的用法，如底下這樣寫就可以查看物件的屬性。

物件名稱.屬性名稱

- ◆ 要注意的是執行屬性後 ndarray 的原始資料「**不會**」改變，例如用.T 會傳回陣列轉置的結果，不過原始資料不會改變。底下是一些屬性的例子

```
1 import numpy as np
2
3 a= np.array([[3,6,9],[2,4,6]])
4
5 print(a.ndim)
6 print(a.shape)
7 print(type(a))
8 print(a.dtype)
9 print(a.data)
10 print(a.T)
11 print(a.size)
12 print(a.itemsize)
13 print(a.nbytes)
```

```
2
(2, 3)
<class 'numpy.ndarray'>
int32
<memory at 0x0000021B04468708>
[[3 2]
 [6 4]
 [9 6]]
6
4
24
```

NumPy 聚合函式 (Aggregations)

函式名稱	說明
<code>np.sum()</code>	計算元素之和
<code>np.prod()</code>	計算元素的乘積
<code>np.mean()</code>	計算元素的平均
<code>np.median()</code>	計算元素的中位元元數
<code>np.min()</code>	找到最小值
<code>np.max()</code>	找到最大值
<code>np.argmin()</code>	查找最小值索引
<code>np.argmax()</code>	查找最大值索引
<code>np.any()</code>	評估是否有任何元素為真
<code>np.std()</code>	計算標準差
<code>np.var()</code>	計算(均)方差， <code>np.std()</code> 的平方等於 <code>np.var()</code>
<code>np.all()</code>	評估所有元素是否都為真
<code>np.percentile()</code>	計算元素基於排名的統計數據(第%分位的數值)

Numpy - 數學&統計方法

```
In [1]: import numpy as np  
x = np.random.randn(5, 4) # 建立一個5 X 4的隨機資料  
x
```

```
Out[1]: array([[ 1.44817216,  1.34442998, -0.13616892,  1.15668132],  
[-1.27028608, -1.89138058,  1.68052131, -0.7858721 ],  
[-0.45023249,  1.09773632, -1.39559407,  0.5843488 ],  
[-0.92650082, -0.97816234,  1.88492239, -0.68833393],  
[-0.16152264,  1.21541026, -0.73811865,  0.68241415]])
```

```
In [2]: x.mean() # 取平均值
```

```
Out[2]: 0.0836232035641529
```

```
In [3]: x.sum() # 取總和
```

```
Out[3]: 1.6724640712830579
```

```
In [4]: x.min() # 取最大值
```

```
Out[4]: -1.891380576450109
```

```
In [5]: x.max() # 取最小值
```

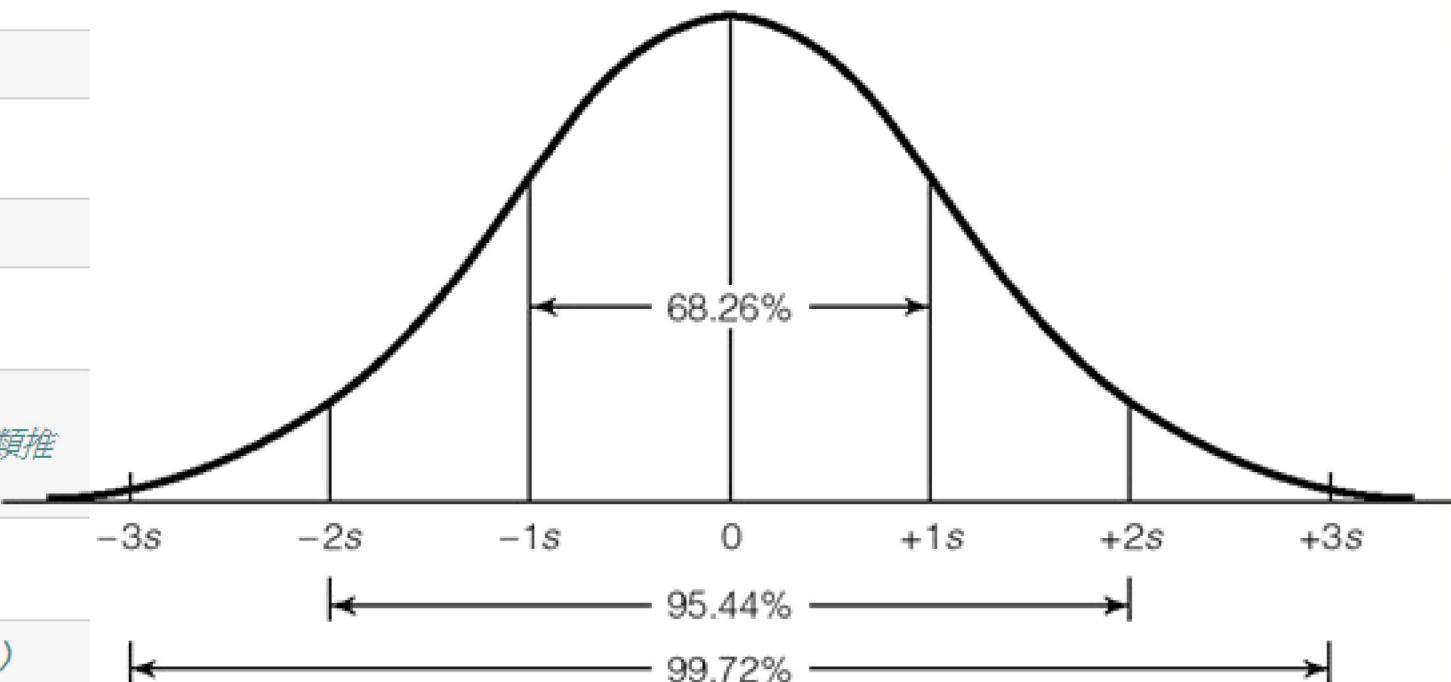
```
Out[5]: 1.8849223895215022
```

```
In [6]: x = np.array([[2,3,4], [5,6,7]])  
#回傳累積的數值 第一個index為2 = 2，第二個index為2+3 = 5，第三個index為2+3+4 = 9，依此類推  
x.cumsum()
```

```
Out[6]: array([ 2,  5,  9, 14, 20, 27], dtype=int32)
```

```
In [7]: np.std(x) # 取標準差，標準公式 : std = sqrt(mean(abs(x - x.mean())**2))
```

```
Out[7]: 1.707825127659933
```



Numpy 的索引

```
In [1]: a=[i for i in range(1,6)]  
  
In [2]: b=[i for i in range(6,12)]  
  
In [3]: x=[a,b]  
  
In [4]: print(x)  
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10, 11]]  
  
In [5]: y=x[1][1:]  
  
In [6]: print(y)  
[7, 8, 9, 10, 11]  
  
In [7]: y[1]=200  
  
In [8]: print(y)  
[7, 200, 9, 10, 11]  
  
In [9]: print(x)  
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10, 11]]
```

```
In [1]: import numpy as np  
  
In [2]: x=np.arange(12).reshape(2,6)  
  
In [3]: print(x)  
[[ 0  1  2  3  4  5]  
 [ 6  7  8  9 10 11]]  
  
In [4]: y=x[1][1:]  
  
In [5]: print(y)  
[ 7  8  9 10 11]  
  
In [6]: y[1]=200  
  
In [7]: print(y)  
[ 7 200   9 10 11]  
  
In [8]: print(x)  
[[ 0  1  2  3  4  5]  
 [ 6  7 200   9 10 11]]
```

Numpy 的索引

```
>>> a[0, 3:5]
```

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

```
>>> a[4:, 4:]
```

```
array([[44, 55],  
       [54, 55]])
```

```
>>> a[:, 2]
```

```
a([2, 12, 22, 32, 42, 52])
```

```
>>> a[2::2, ::2]
```

```
array([[20, 22, 24],  
       [40, 42, 44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Numpy 的索引

- ◆ 整數陣列索引也可使用多個陣列來進行指定。

```
import numpy as np
x=np.arange(12).reshape(3,4)
print(x)
y=x[[0,1,2],[2,1,0]]
print(y)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[2 5 8]
```

- ◆ 若想將格子狀裡特定列與特定行交叉點位置的元素取出時，可使用np.ix_函式。

```
import numpy as np
x=np.arange(12).reshape(3,4)
z=x[np.ix_([0,2],[1,3])]
print(z)
```

```
[[ 1  3]
 [ 9 11]]
```

Numpy 的切割(Slicing)

- ◆ 我們可以只改變局部的資料，這就是所謂的切割(Slicing)。

```
In [1]: import numpy as np
```

```
In [2]: arr = np.arange(10)
```

```
In [3]: slice = arr[5:8]
```

```
In [4]: arr[5:8]=7
```

```
In [5]: slice[1]=87
```

```
In [6]: slice
```

```
Out[6]: array([ 7, 87, 7])
```

```
In [7]: arr
```

```
Out[7]: array([ 0, 1, 2, 3, 4, 7, 87, 7, 8, 9])
```

Q & A