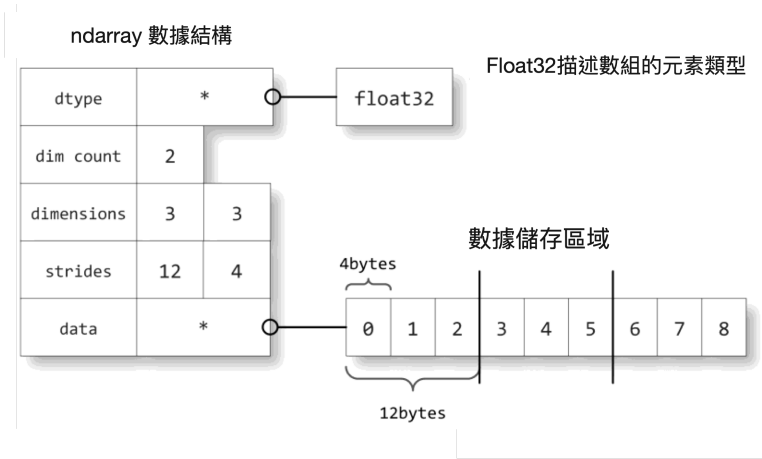
Numpy 陣列的初始化

簡報閱讀 範例與作業 問題討論 學習心得(完成)

重要知識點

- 能夠使用不同的方法初始化一個陣列
- 知道固定大小對於陣列的意義
- 了解不同的亂數陣列有什麼差異
- 如何操作結構化陣列 (Structured Array

NumPy 提供了一個同類型元素的多維容器型態，稱為是數組或陣列。陣列的全名是 N-dimensional Array，習慣簡寫為 NdArray 或 Array。



建立陣列的四種方式

一般來說，在 NumPy 當中建立/初始化陣列有四種方法：

1. 從內建型態作轉換
2. 從固定大小的初始值開始
3. 從固定大小的序列值開始
4. 從固定大小的亂數值開始

從內建型態作轉換

Python 中內建的容器型態有列表、元祖、字典和集合，可以直接利用 np.array() 方法做型態轉換：

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

補充：陣列與列表本質上的差異請參考「Numpy 陣列的定義與屬性」課程

np.array(...) 更多用法

來看一下關於 np.array(...) 參數比較細節的用法：

numpy.array(object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0, like=None)

- object：必填，任何 array_like 物件
- dtype：指定轉成陣列後的元素型態
- copy：預設為 True，是否產生一個新的物件
- order：指定元素在記憶體中的儲存方式

補充：更多細節用法，請參考 官方文件。

從內建型態作轉換

會自動轉換成範圍比較大的型態：

```
np.array([1, 2, 3.0])
# array([ 1.,  2.,  3.])
```

也可以指定成想要的型態：

補充：陣列與列表本質上的差異請參考「Numpy 陣列的定義與屬性」課程

使用之後會發現，字典型態雖然可以成功被轉成陣列，不過好像不符合我們的預期。

```
np.array({0: 123, 1: 456})
# array({0: 123, 1: 456})
```

正確的寫法應該寫轉成有序的 List 再作轉換：

```
np.array(list({0: 123, 1: 456}.items()))
# array([[  0 123]  [  1 456]])
```

從固定大小的初始值開始

第二種方法可以先建立一個固定大小的初始值，例如由 0、1 或特定值所組成的陣列：

```
np.zeros((2, 3))
# 建立由 0 組成的 2x3 陣列
np.ones((2, 3))
# 建立由 1 組成的 2x3 陣列
np.full((2, 3), 9)
# 建立由 9 組成的 2x3 陣列
```

np.zeros 和 np.empty

一個類似的方法是 np.empty(...)：

```
np.zeros((2, 3))
#
np.empty((2, 3))
#
```

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

[[0. 0. 2.1e-31 6.7e-31]
 [2.2e-31 2.1e-31 2.3e-312 6.7e-31]
 [2.2e-31 2.1e-31 2.46e-31 2.4e-31]]

補充：e-31 是自然數的 -31 次方意思，代表很小很小的數。

建立 NumPy array (陣列)

- 呼叫 zeros()、ones() 函式，可以依照傳入的形狀引數，建立元素全為 0、全為 1 的陣列。
- 使用 empty() 函式則是不需要給定起始值，但是可以建立給定形狀的陣列，元素值則會隨機給定。

從固定大小的序列值開始

這個方法可以產生一個特定的序列值，有三種不同的序列：

3. 等比序列

```
np.arange( 10, 30, 5 )
# array([10, 15, 20, 25])
np.linspace( 0, 2, 3 )
# array([0. 1. 2.])
np.logspace( 0, 2, 3 )
# array([1. 10. 100.])
```

從固定大小的亂數值開始

另外一種常見的情境是產生一組固定大小的亂數：

```
from numpy.random import default_rng
rng = default_rng()

normal = rng.standard_normal((3,2))
random = rng.random((3,2))
integers = rng.integers(0, 10, size=(3,2))
```

[[-0.50235868 -0.82384404]
 [-0.13563574 -1.38596654]
 [-1.27113178 1.32275548]]

[[0.36101818 0.40320228]
 [0.39802392 0.99634935]
 [0.05937934 0.03889768]]

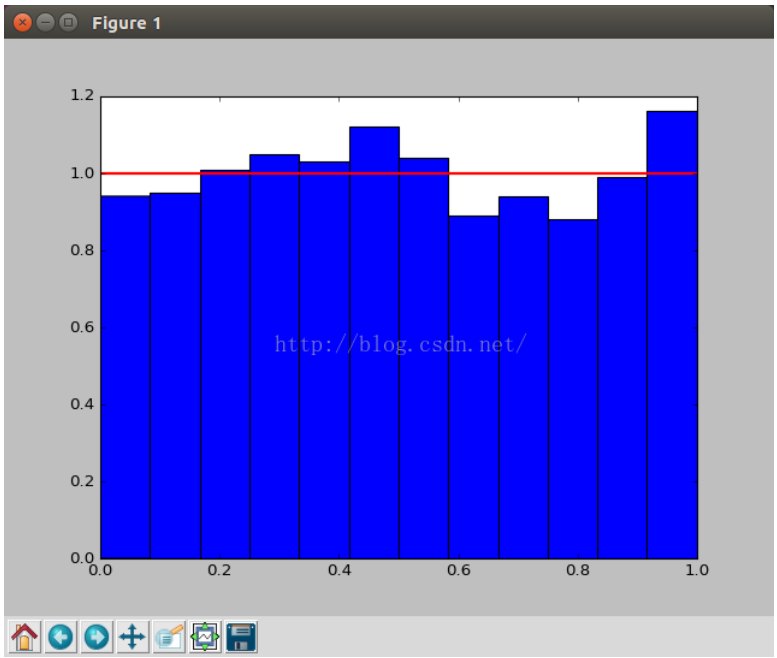
[[5 7]
 [4 1]
 [1 9]]

補充：更多不同的分佈跟用法，請參考文件。

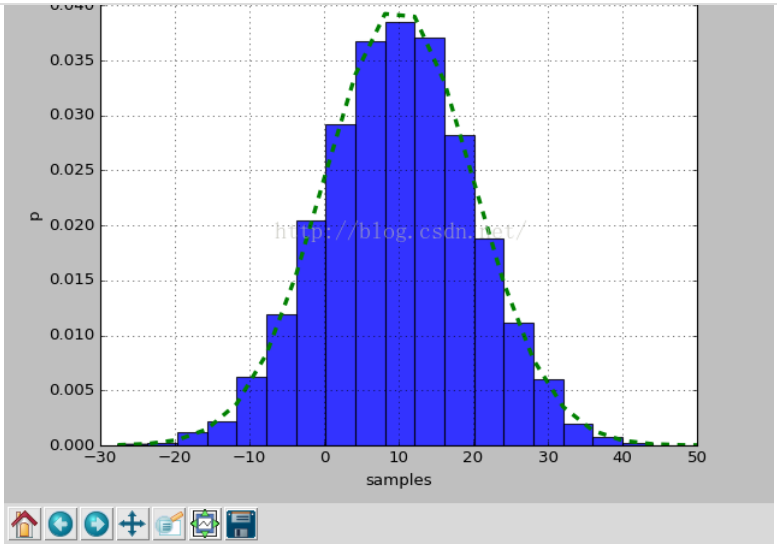
從文件來上看，可以發現 np.random 其實還有許多種不同的亂數方法，他們彼此間究竟有什麼差別呢？

仔細觀察之後，會發現這些隨機亂數最大的差別在於「分佈」。也就是當亂數資料大量之後，會呈現不同的分佈趨勢。此外，在 NumPy 當中，也有提供各式各項的分佈亂數提供使用。

random 分佈：



normal 分佈：



為什麼習慣建立一個「固定大小」的陣列？

根據初始化陣列的四種方法：

1. 從內建型態作轉換
2. 從固定大小的初始值開始
3. 從固定大小的序列值開始
4. 從固定大小的亂數值開始

我們為什麼習慣從固定大小的陣列開始初始化呢？主要原因是因為陣列的儲存特性，陣列的元素會配置在連續的記憶體位置。每次改動到大小對於記憶體的更動負擔是比較大的，因此希望在一開始就練利一組固定的尺寸避免頻繁改動記憶體。

NumPy 結構化陣列 (Structured Arrays)

- 建立結構化陣列可透過 dictionary 型別的資料建立 np.dtype 物件，並指定 dtype 給陣列。

皆可。在範例中我們混用了 3 種型別的表示方式

```
1  dt = np.dtype({'names':('Name', 'num1', 'num2'
2  b = np.genfromtxt("structured.txt", delimiter=
```

```
dt = np.dtype({'names':('Name', 'num1', 'num2', 'True'), 'formats':((np.str_, 5), np.int32, int, 'U3')})

b = np.genfromtxt("structured.txt", delimiter=',', dtype=dt)
b

array([('Jay', 1, 2, 'Yes'), ('James', 3, 4, 'No'), ('Joe', 5, 6, 'Yes')],
      dtype=[('Name', '<U5'), ('num1', '<i4'), ('num2', '<i8'), ('True', '<U3')])
```

- 建立陣列後，可以用索引的方式存取元素資料。

```
b[0]
```

```
('Jay', 1, 2, 'Yes')
```

- 也可以用 Column 名稱，取得 Column 所有元素值。

```
b['Name']
```

```
array(['Jay', 'James', 'Joe'], dtype='<U5')
```

```
1  b['Name']
```

- 取得單筆資料的欄位值。

```
b[1]['True']
```

```
'No'
```

- 也可以進行邏輯操作，取得對應的結果。

```
b[b['num2'] >= 3]['Name']
```

- 新建立一個結構化陣列，方式跟建立陣列非常類似。
- 下例使用 zeros() 初始化陣列，並指定 dtype。

```
c = np.zeros(3, dtype=dt)
c
```

```
array([('', 0, 0, ''), ('', 0, 0, ''), ('', 0, 0, '')],
      dtype=[('Name', '<U5'), ('num1', '<i4'), ('num2', '<i8'), ('True', '<U3')])
```

```
c = np.zeros(3, dtype=dt)
```

- 將清單資料餵入結構化陣列中。

```
name = ['Chloe', 'Charlotte', 'Clara']
num_1 = [11, 12, 13]
num_2 = [14, 15, 16]
check = ['Y', 'Y', 'N']
```

```
c['Name'] = name
c['num1'] = num_1
c['num2'] = num_2
c['True'] = check
```

```
print(c)
```

```
[('Chloe', 11, 14, 'Y') ('Charl', 12, 15, 'Y') ('Clara', 13, 16, 'N')]
```

```
name = ['Chloe', 'Charlotte', 'Clara']
num_1 = [11, 12, 13]
num_2 = [14, 15, 16]
check = ['Y', 'Y', 'N']

c['Name'] = name
c['num1'] = num_1
c['num2'] = num_2
c['True'] = check

```

RecordArray 與 Structured Array 非常類似，但是提供更多的屬性可以用來存取結構化陣列。不過 RecordArray 雖然方便但是在效能上會比原來的陣列差。使用方法如下：

```
c_rec = c.view(np.recarray)
c_rec
```

```
rec.array([('Chloe', 11, 14, 'Y'), ('Charl', 12, 15, 'Y'),
           ('Clara', 13, 16, 'N')],
          dtype=[('Name', '<U5'), ('num1', '<i4'), ('num2', '<i8'), ('True', '<U3')])
```

```
1  c_rec = c.view(np.recarray)
```

原先我們是透過索引或是名稱存取元素值，但是 RecordArray 可以使用屬性的方式來取得。

```
c_rec.Name
```

```
array(['Chloe', 'Charl', 'Clara'], dtype='<U5')
```

知識點回顧

- 能夠使用不同的方法初始化一個陣列
- 知道固定大小對於陣列的意義
- 了解不同的亂數陣列有什麼差異
- 如何操作結構化陣列 (Structured Arrays)

參考資料

Python 中的 list 與 NumPy 中 array 的區別 及相互轉換
網站：blog.csdn.net/

完整說明 Pytho 中的列表與 NumPy 中的陣列的本質差異與轉換方法。

a為python的list類型

將a轉化為numpy的array:

```
np.array(a)
```

得到類型：array([3.234, 34. , 3.777, 6.33])

將a轉化為python的list

```
a.tolist()
```

下一步：閱讀範例與完成作業