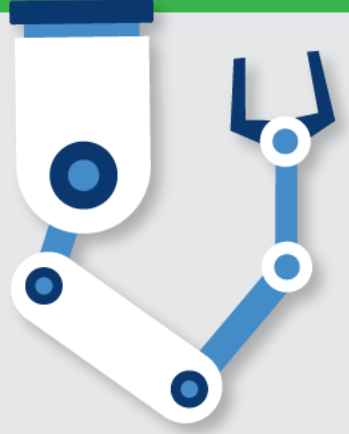
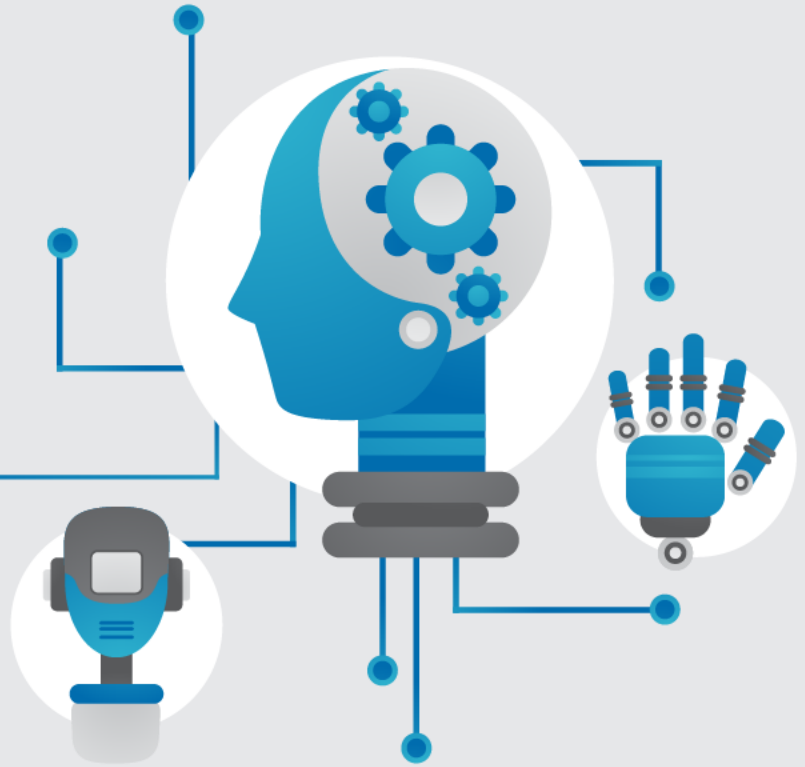


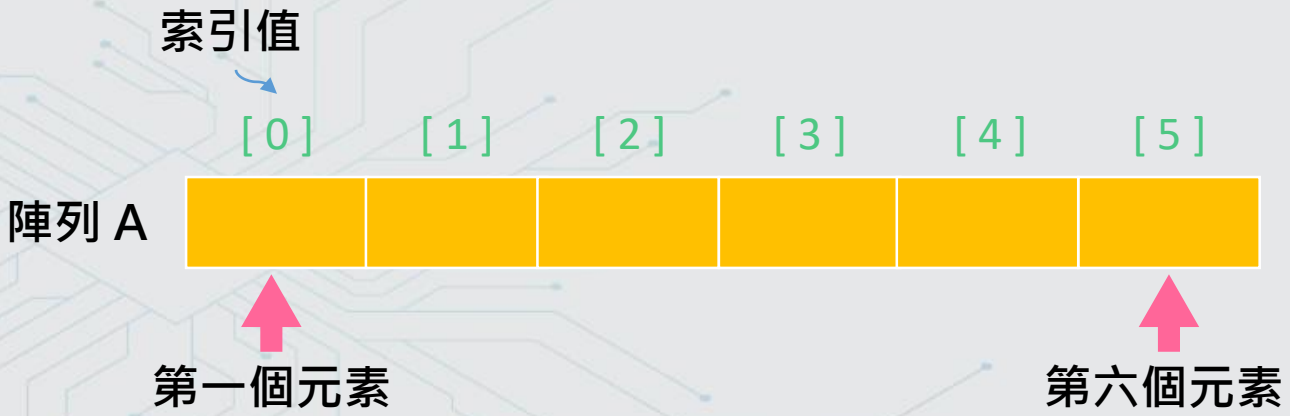
資料處理工具(I)

Numpy



陣列

- NumPy是python用來處理陣列的套件
- 陣列（Arrays）類似Python清單（Lists），但陣列元素的資料型態必須是相同的，不同於清單可不同。
- 陣列是程式語言的一種基本資料結構，屬於循序性的資料結構。



陣列

- **多維陣列**（multi-dimensional array）是指二維以上維度的陣列（含二維），屬於一維陣列的擴充。

功課表

	一	二	三	四	五
1		2		2	
2	1	4	1	4	1
3	5		5		5
4					
5	3		3		3
6					

課程名稱	課程代碼
計算機概論	1
離散數學	2
資料結構	3
資料庫理論	4
上機實習	5

NumPy安裝與使用

Python安裝套件

- 一般安裝完python預設都已經裝好numpy，需要改變版本如下

  C:\> pip install numpy==[版本]

Python 程式匯入套件

- import numpy as np

建立陣列

指令

- np.array([element], [dtype])
- np.zeros([shape] , [dtype])
- np.full([shape], [value])

參數

- element：陣列元素（list or tuple）
- dtype [optional]：指定資料型態（type or string）
- shape：陣列的維度資訊（tuple）
- value：元素值（int, float, string...）

回傳

- 陣列 (np.array)

建立陣列 - 範例

- 範例程式

```
a = np.array([[1, 2], [3, 4]], dtype=  'float32')
```

```
print(a)
```

```
In [77]: print(a)
[[1. 2.]
 [3. 4.]]
```

```
print(a[1, 0])
```

```
In [79]: print(a[1, 0])
3.0
```

建立陣列 - 範例

- 範例程式

```
b=np.zeros((3,3))
print(b)
```

```
In [92]: b=np.zeros((3,3))
    ...: print(b)
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
c =np.full((4,4), 255, dtype='uint8')
print(c)
```

```
In [93]: c=np.full((4,4), 255, dtype='uint8')
    ...: print(c)
[[255 255 255 255]
 [255 255 255 255]
 [255 255 255 255]
 [255 255 255 255]]
```

陣列元素取得

指令

- np.array[[index]]
- np.array[[slice]]
- np.array[[list or tuple]]
- np.array[[boolean mask]]

參數

- index：索引（整數）
- slice：取出特定範圍，格式為[start:end:step]
- list or tuple：多個索引
- boolean mask：True為要抓出來的元素

回傳

- 對應元素或陣列 (int, np.array)

陣列元素取得 - 範例

› 範例程式

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

```
print(a)
```

```
In [91]: a
Out[91]:
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

```
print(a[0,1], a[0,-1])
```

```
In [96]: print(a[0,1], a[0,-1])
2 5
```

```
print(a[:,4::-2])
```

```
In [97]: print(a[:,4::-2])
[[ 5  3  1]
 [10  8  6]]
```

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

```
In [98]: print(a[:, [-1,-2,-3]])
[[ 5  4  3]
 [10  9  8]]
```

```
# 等於a[0,2] a[1,3]兩個元素
```

```
print(a[ ([0,1],[2,3]) ])
```

```
In [99]: print(a[ ([0,1],[2,3]) ])
[3 9]
```

```
print(a<7)
```

```
In [100]: print(a<7)
[[ True  True  True  True  True]
 [ True False False False False]]
```

```
print(a[a<7])
```

```
In [101]: print(a[a<7])
[1 2 3 4 5 6]
```

陣列物件屬性

- **指令**　[np.array].[attr]
- **參數**
  - np.array：陣列物件
  - attr：屬性
- **回傳**　對應元素或陣列（int, np.array）
- **說明**
  - 常用屬性
    - ✓ shape
    - ✓ dtype
    - ✓ size

陣列物件屬性 - 範例

- 範例程式

```
a = np.array([[1, 2], [3, 4]], dtype='float32')
```

```
print(a.shape)
```

```
In [137]: print(a.shape)
(2, 2)
```

```
print(a.dtype)
```

```
In [138]: print(a.dtype)
float32
```

```
print(a.size)
```

```
In [139]: print(a.size)
4
```

陣列物件方法

- **指令**　[np.array].[method]
- **參數**
  - np.array：陣列物件
  - method：方法
- **回傳陣列**　（numpy.array）
- **說明**
  - 常用方法
    - ✓ copy ： clone array物件
    - ✓ reshape ：重新設定陣列維度資訊
    - ✓ astype ：轉型
    - ✓ clip ：限制陣列元素範圍

陣列物件方法範例 - copy

範例程式

```
img=cv2.imread('lena.jpg')
print(img[10,10])
# 保存原本的影像陣列
origin=img
# 更動該像素顏色
img[10,10]=np.array([10,10,10])
# 因為沒使用copy，連保存的也一起變了
print(img[10,10)
print(origin[10,10])
```

```
In [185]: print(img[10,10])
[111 133 228]
```

```
In [187]: print(img[10,10])
[10 10 10]
```

```
In [188]: print(origin[10,10])
[10 10 10]
```

陣列物件方法範例 - copy

範例程式

```
img=cv2.imread('lena.jpg')
print(img[10,10])
```

```
In [190]: print(img[10,10])
[111 133 228]
```

```
# 保存原本的影像陣列
origin=img.copy()
# 更動該像素顏色
img[10,10]=np.array([10,10,10])
print(img[10,10])
```

```
In [192]: print(img[10,10])
[10 10 10]
```

```
print(origin[10,10])
```

```
In [193]: print(origin[10,10])
[111 133 228]
```

陣列物件方法範例 - reshape

範例程式

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test)= mnist.load_data()
print(x_train.shape, x_test.shape)
```

```
In [210]: print(x_train.shape, x_test.shape)
(60000, 28, 28) (10000, 28, 28)
```

```
# -1代表根據其他維度資訊自動計算(只能有一個-1)
x_train=x_train.reshape(len(x_train),-1)
x_test=x_test.reshape(len(x_test),-1)
print(x_train.shape, x_test.shape)
```

```
In [212]: print(x_train.shape, x_test.shape)
(60000, 784) (10000, 784)
```

陣列物件方法範例 - astype

範例程式

```
img=cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
print(img[8,507], img.dtype)
```

```
In [2]: print(img[8,507], img.dtype)
49 uint8
```

```
# 降低亮度
img-=50
# 因為數值已小於uint8的最小值，所以循環至最大值
print(img[8,507])
```

```
In [4]: print(img[8,507])
255
```

陣列物件方法範例 - astype

範例程式

```
img=cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
print(img[8,507], img.dtype)
# 改成有正負的16位元整數
```

```
In [13]: print(img[8,507], img.dtype)
49 uint8
```

```
img=img.astype('int16')
img-=50
# 預想的值
print(img[8,507])
```

```
In [15]: print(img[8,507])
-1
```

陣列物件方法範例 - clip

範例程式

```
img=cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
# 改成有正負的16位元整數
img=img.astype('int16')
img-=50
# 切到0 ~ 255範圍內
img=img.clip(0, 255)
# 轉回來，因為opencv規定該型態才能存成影像
img=img.astype('uint8')
```

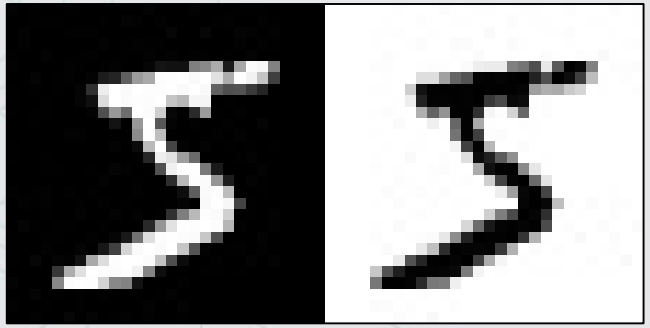
NumPy方法

- **指令**　[np.[method]]
- **參數**
  - method：方法
- **回傳**　陣列（numpy.array）
- **說明**
  - 常用方法
    - ✓ concatenate，將兩個陣列根據軸接在一起（該軸長度需相等）
    - ✓ unique，返回所有不重複的元素
    - ✓ where，返回符合條件的索引值
    - ✓ save、load，將陣列存成npy檔，方便之後載入使用

NumPy方法範例 - concatenate

- 範例程式

```
(x_train, y_train), (x_test, y_test)=mnist.load_data()
img=x_train[0]
rev=255-img
cv2.imwrite('show.jpg', np.concatenate((img,rev), axis=1))
```



NumPy方法範例 - unique

- 範例程式

(x_train, y_train), (x_test, y_test)=mnist.load_data()

print(np.unique(y_train))

```
In [65]: print(np.unique(y_train))
[0 1 2 3 4 5 6 7 8 9]
```

NumPy方法範例 - where

- 範例程式

```
(x_train, y_train), (x_test, y_test)=mnist.load_data()
img=x_train[0]
img=cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
print(np.where(img==0))
```

```
In [132]: print(np.where(img==0))
(array([ 0,  0,  0, ..., 27, 27, 27], dtype=int64),
array([ 0,  0,  0, ..., 27, 27, 27], dtype=int64),
array([0, 1, 2, ..., 0, 1, 2], dtype=int64))
```

NumPy方法範例 - save、load

- 範例程式

```
x=[]
for file in os.listdir('cifar10_train'):
    x.append(cv2.imread('cifar10_train/'+file))
x=np.array(x)
x=x.astype('float32')/255.0
np.save('cifar10_x_normalize', x)
```

```
x=np.load('cifar10_x_normalize')
```

當資料量較大時
讀取及處理需要很多時間
處理好可先將陣列存成檔案
之後load即可使用

NumPy random模組

- **指令**　np.random.[method]

- **參數**
  - method：方法

- **說明**
  - 常用方法
    - ✓ shuffle：隨機打亂

NumPy random模組範例 - shuffle

範例程式

```
print(y_train[0])
```

```
In [159]: print(y_train[0])
5
```

```
# 將兩個陣列包成index對應的list，[ [img1, label1], [img2, label2], ..... ]
xy=list(zip(x_train, y_train))
# 隨機打亂
np.random.shuffle(xy)
# zip(*)為復原指令，等於x_train=np.array([ i[0] for i in xy ])
x_train, y_train=zip(*xy)
print(y_train[0])
```

```
In [161]: print(y_train[0])
6
```