心理與神經資訊學
(Psychoinformatics & Neuroinformatics)

課號: Psy5261

識別碼: 227U9340

教室:彷彿在雲端

時間: 二789


本週就要繳作業了喔
!

More on “import”

- **正規法:**

import random ⇦幫助大家了解函數來源
random.random()

- **取暱稱:**

import random as rnd
rnd.random()

試import this
試import antigravity

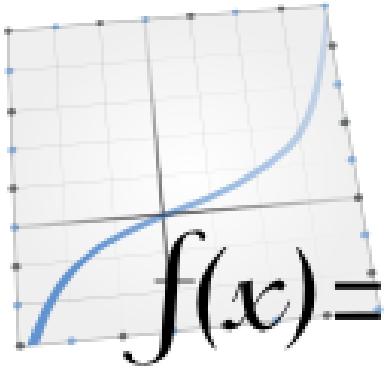
- **懶人法:**

from random import *
random() ⇦別的模組可能有一樣名稱的函數

基本資料分析
(NumPy & Pandas)

自建函數

Try: (注意縮排用來告訴**Python**從屬關係)




```
import math
def adjust_score(old):
  new=math.sqrt(old)*10
  return new

a=adjust_score(0)
b=adjust_score(60)
print(a,b)
```

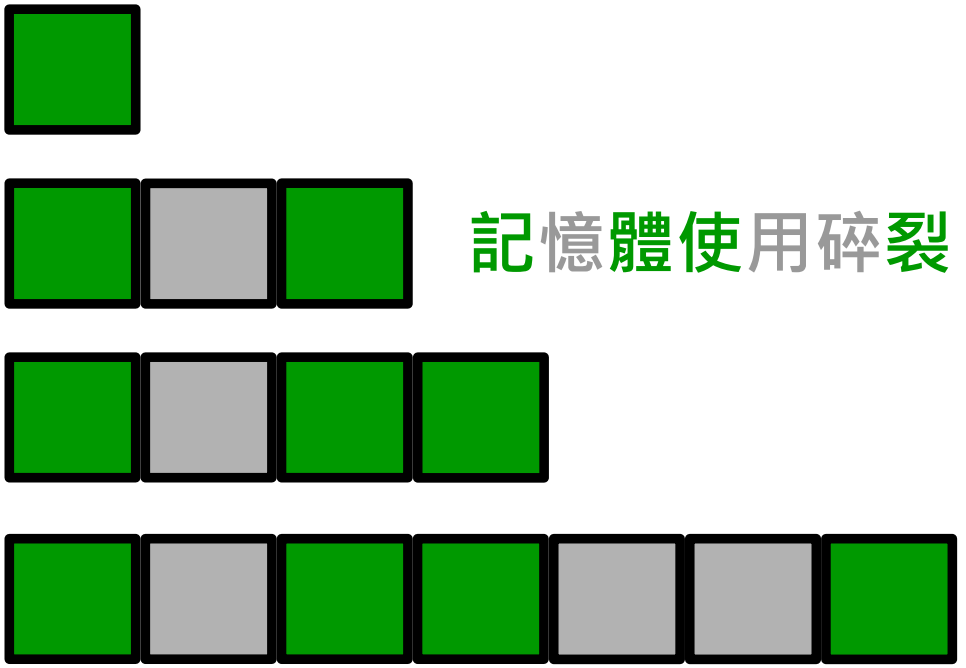
處理多個數據的需求

```
print(adjust_score(range(0,101,10)))
```

TypeError: a float is required

解法1a:利用迴圈

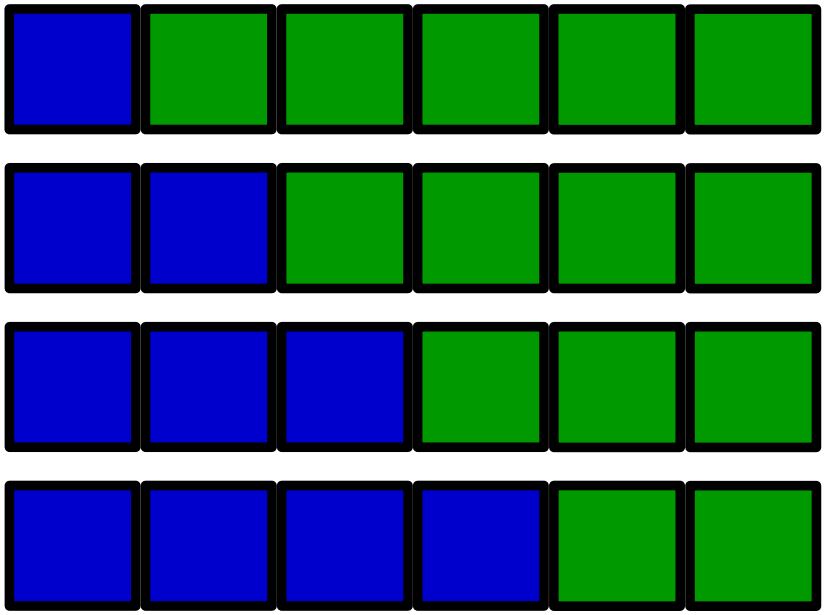
```
scores=[]
for i in range(0,101,10):
 scores.append(adjust_score(i))
 #scores=scores+[adjust_score(i)]
print(scores)
```



記憶體使用碎裂

解法1b:利用迴圈

```
old=range(0,101,10)
N=len(old)
scores=[0.]*N
for i in range(N):
 scores[i]=adjust_score(i)
print(scores)
```



序列計算vs.平行計算

調 調 調 調 調 調 調 調 調 調

0 10 20 30 40 50 60 70 80 90 100

(只有1條生產線)

調 調 調 調 調 調 調 調 調 調 調

0 10 20 30 40 50 60 70 80 90 100

(共有11條生產線)

解法2:利用內建函數map

```
import math
def adjust_score(old):
 new=math.sqrt(old)*10
 return new

print(list(map(adjust_score,range(0,101,10))))
```

套上去

multiprocessing的map才是真的平行計算
以後講Big Data的時候會再看到類似觀念

資料科學家真實案例

某臺大畢業生去Facebook應徵時

請解釋何謂
MapReduce?

解法3: 利用NumPy (1行!)

```
import numpy as np
(np.arange(0,101,10)**0.5)*10 #但仍是單核計算
```

NumPy的arange和內建的range有何不同?

```
a=range(0,101,10)
b=np.arange(0,101,10)
a+1
b+1
a+a
b+b
a*3
b*3
```

向量加法(c,d)+(e,f)=(c+e,d+f)
因此element-wise的平行運算又稱向量化(vectorization)

另一個例子:亂數

若要產生100個亂數

用random.random做100次:

```
import random
r=[]
for i in range(100):
   r.append(random.random())
```

用numpy.random.rand做1次:

```
import numpy as np
r=np.random.rand(100)
```

List vs. NumPy Array (1/2)

List很自由，可以亂塞資料

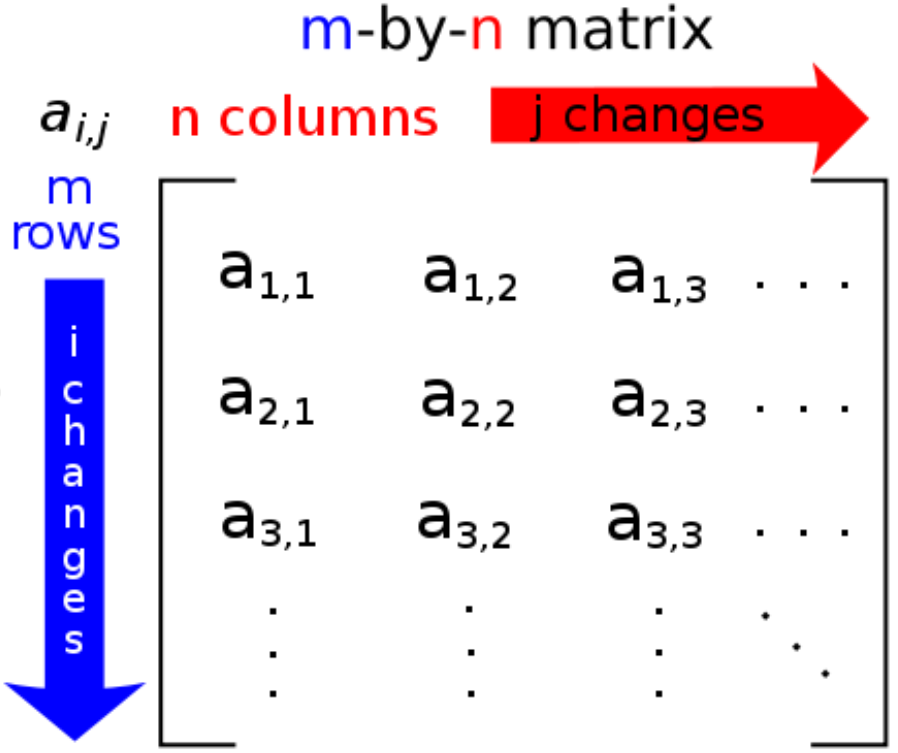
```
a=[[5566,'never dies'],5,[[['R',range(3)],'doll'],6]]
a[0] #[5566, 'never dies']
a[1] #5
a[0][0] #5566
a[2][0][0][1] #range(0,3)
```

Tip:想成樹狀結構就不會昏頭了

List vs. NumPy Array (2/2)

通常Array內所有元素皆為數字以方便計算
結構上較List方正(2維平面, 3維方塊, etc.)

```
a=np.array([range(3),np.random.rand(3)])
a.dtype #dtype('float64')
a.T # transpose:矩陣轉置
a[0][2] #2.0
a[0,2] #2.0
a[0,:] #array([ 0.,  1.,  2.])
a[0,1:3] #array([ 1.,  2.])
np.mean(a)
np.mean(a,0)
np.mean(a,1)
```



實驗設計

Randomized design:

```
import numpy as np
trials=np.random.randint(0,3,15)
```

Counterbalanced design:

```
import numpy as np
trials=np.array(list(range(3))*5) #3條件各5次
trials=np.random.permutation(trials)
```

```
for t in trials:
  print(t)
  if t==0:
    print("I got you!")
```

資料分析(1/2)

實驗條件	正確與否	反應時間
1	1	-1 (timed out)
0	1	0.444112
1	0	-1 (timed out)
1	0	2.597051
2	1	1.927228
...	...	...

```
import numpy as np
data=np.loadtxt('exp_subj0.txt') # 匯入資料
valid=(data[:,2]>0) #尋找RT>0的valid trials
data=data[valid,:] #平均正確率 & 平均反應時間
print(np.mean(data[:,1]),np.mean(data[:,2]))
```

資料分析(2/2)

資料應該要分組別分析

```
Ngroups=np.unique(data[:,0]).size #3
groups=[0]*Ngroups #[0, 0, 0]
for i in range(Ngroups):
  selector=(data[:,0]==i) #判斷組別為0, 1, or 2
  groups[i]=data[selector,:] #用List來存Array!
print(groups[0]) #印出第0組來看看
```

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64      Signed 64-bit integer types
>>> np.float32    Standard double-precision floating point
>>> np.complex    Complex numbers represented by 128 floats
>>> np.bool       Boolean type storing TRUE and FALSE values
>>> np.object     Python object type
>>> np.string_    Fixed-length string type
>>> np.unicode_   Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape            Array dimensions
>>> len(a)             Length of array
>>> b.ndim             Number of array dimensions
>>> e.size             Number of array elements
>>> b.dtype            Data type of array elements
>>> b.dtype.name       Name of data type
>>> b.astype(int)      Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                          Array-wise sum
>>> a.min()                          Array-wise minimum value
>>> b.max(axis=0)                    Maximum value of an array row
>>> b.cumsum(axis=1)                 Cumulative sum of the elements
>>> a.mean()                         Mean
>>> b.median()                       Median
>>> a.corrcoef()                     Correlation coefficient
>>> np.std(b)                        Standard deviation
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()              Sort an array
>>> c.sort(axis=0)        Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                  Select the element at the 2nd index
3
>>> b[1,2]                Select the element at row 1 column 2
6.0                       (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                Select items at index 0 and 1
array([1, 2])
>>> b[0:2,1]              Select items at rows 0 and 1 in column 1
array([ 2.,  5.])
>>> b[:1]                 Select all items at row 0
array([[1.5, 2., 3.]])    (equivalent to b[0:1, :])
>>> c[1,...]              Same as [1,:,:]
array([[[ 3.,  2.,  1.],
        [ 4.,  5.,  6.]]])
>>> a[ : :-1]             Reversed array a
array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                Select elements from a less than 2
array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]   Select elements (1,0),(0,1),(1,2) and (0,0)
array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]   Select a subset of the matrix's rows
array([[ 4. ,5. , 6. , 4. ],       and columns
       [ 1.5, 2. , 3. , 1.5],
       [ 4. , 5. , 6. , 4. ],
       [ 1.5, 2. , 3. , 1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)      Permute array dimensions
>>> i.T                      Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                Flatten the array
>>> g.reshape(3,-2)          Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))          Return a new array with shape (2,6)
>>> np.append(h,g)           Append items to an array
>>> np.insert(a, 1, 5)       Insert items in an array
>>> np.delete(a,[1])         Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)   Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))               Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                     Stack arrays vertically (row-wise)
>>> np.hstack((e,f))               Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))         Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                     Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)           Split the array horizontally at the 3rd
[array([1]),array([2]),array([3])]   index
>>> np.vsplit(c,2)           Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

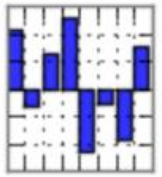
DataCamp
Learn Python for Data Science Interactively

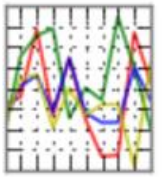
模仿R的Pandas

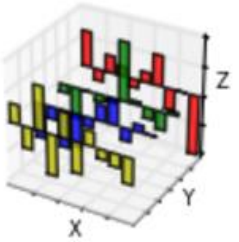
DataFrame便於整理/分析混合型資料&時間序列

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$







Python Data Analysis Library

VERSIONS

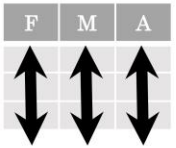
```
import pandas as pd
df=pd.read_table('exp_subj0.txt',sep=' ')
df.describe() # ~ R's summary
```

Data Wrangling with pandas Cheat Sheet

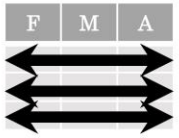
http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



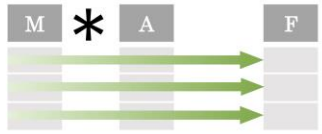
&



Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v'])))
```

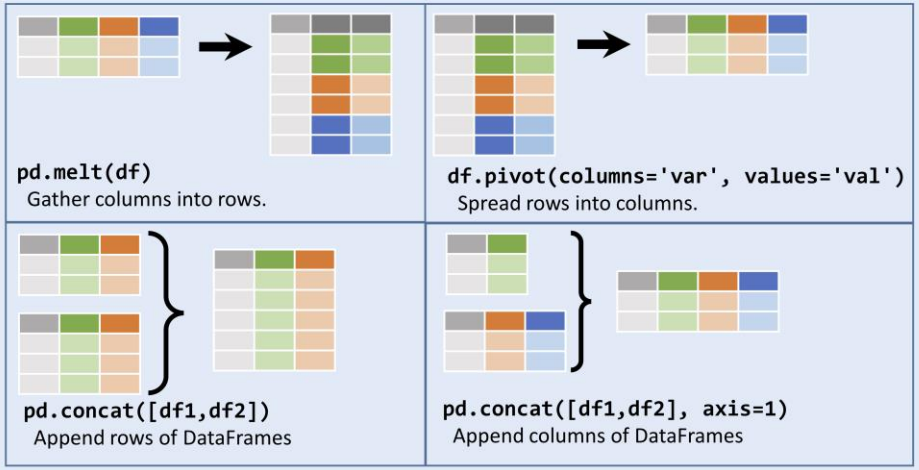
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set



pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

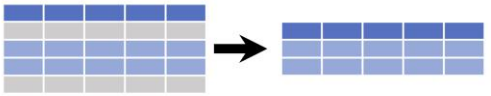
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)

df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

本週作業

用pandas分析power poses實驗資料

Game Over