Institute of Electronics
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

AI Training Course Series

Introduction to Python, Numpy and Pytorch

Lecture 0-2

Architecture Development & Algorithm Refinement for Intelligent Computing

Student: Meng Hsun Hsieh, Jye-En Wu
Advisor: Juinn-Dar Huang, Ph.D.

June 15, 2024

Outline

- Jupyter Notebook
- Introduction to Python
- Introduction to Numpy
- Introduction to Pytorch
- References

Jupyter Notebook on Server

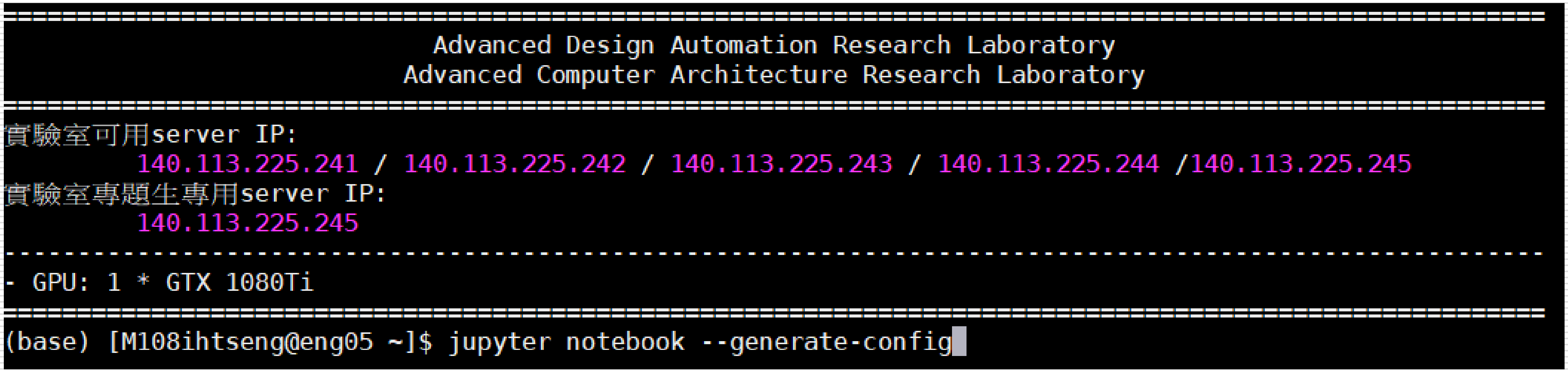
Jupyter Notebook

- Jupyter Notebook:
  - 基於Web的交互式計算環境
  - 利用直譯式語言的特性，容易做到資料視覺化及逐步執行

- Jupyter Notebook on Lab server:
  - 可利用實驗室server的GPU (240, 244和245才有)
  - 可直接使用server上的dataset (ImageNet, COCO)

Step 1

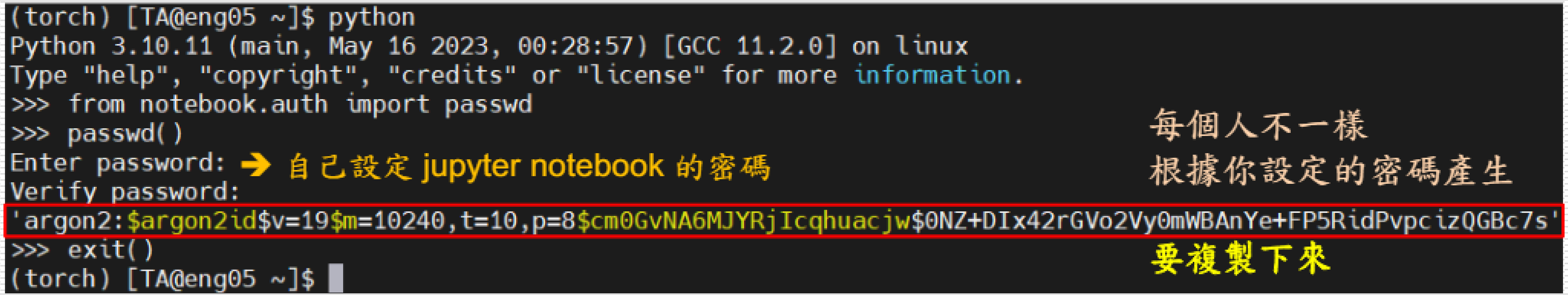
- 登入server (140.113.225.245)
- 在command打:
  - jupyter notebook --generate-config



```
============================================================================================
                    Advanced Design Automation Research Laboratory
                  Advanced Computer Architecture Research Laboratory
============================================================================================
實驗室可用server IP:
        140.113.225.241 / 140.113.225.242 / 140.113.225.243 / 140.113.225.244 /140.113.225.245
實驗室專題生專用server IP:
        140.113.225.245
--------------------------------------------------------------------------------------------
- GPU: 1 * GTX 1080Ti
============================================================================================
(base) [M108ihtseng@eng05 ~]$ jupyter notebook --generate-config
```

Step 2 (1/2)

- 進入conda環境
- 按照下圖輸入
  - ***$ python***
  - ***>>> from notebook.auth import passwd***
  - ***>>> passwd()***



```
(torch) [TA@eng05 ~]$ python
Python 3.10.11 (main, May 16 2023, 00:28:57) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from notebook.auth import passwd
>>> passwd()
Enter password:
Verify password:
'argon2:$argon2id$v=19$m=10240,t=10,p=8$cm0GvNA6MJYRjIcqhuacjw$0NZ+DIx42rGVo2Vy0mWBAnYe+FP5RidPvpcizQGBc7s'
>>> exit()
(torch) [TA@eng05 ~]$
```

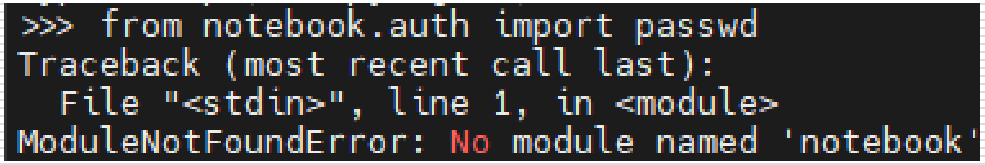
➔ 自己設定 jupyter notebook 的密碼

每個人不一樣
根據你設定的密碼產生

要複製下來

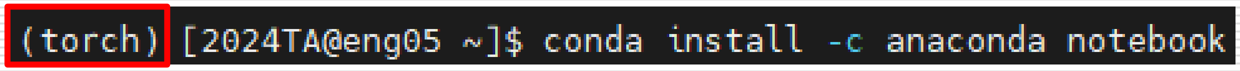
Step 2 (2/2)

- 如果出現下圖中的Error

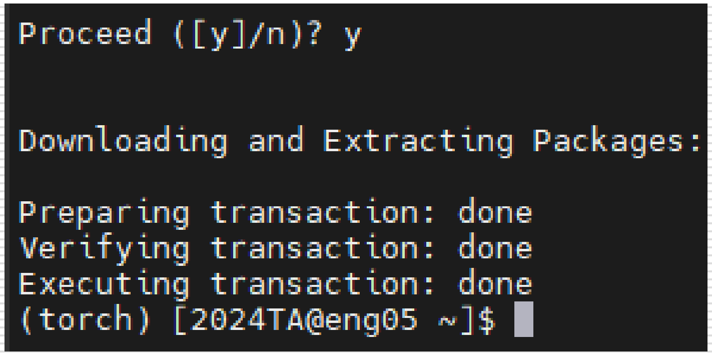


```
>>> from notebook.auth import passwd
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'notebook'
```

- 在conda環境下輸入以下指令，再進行上一頁的步驟
  - ***$ conda install -c anaconda notebook***



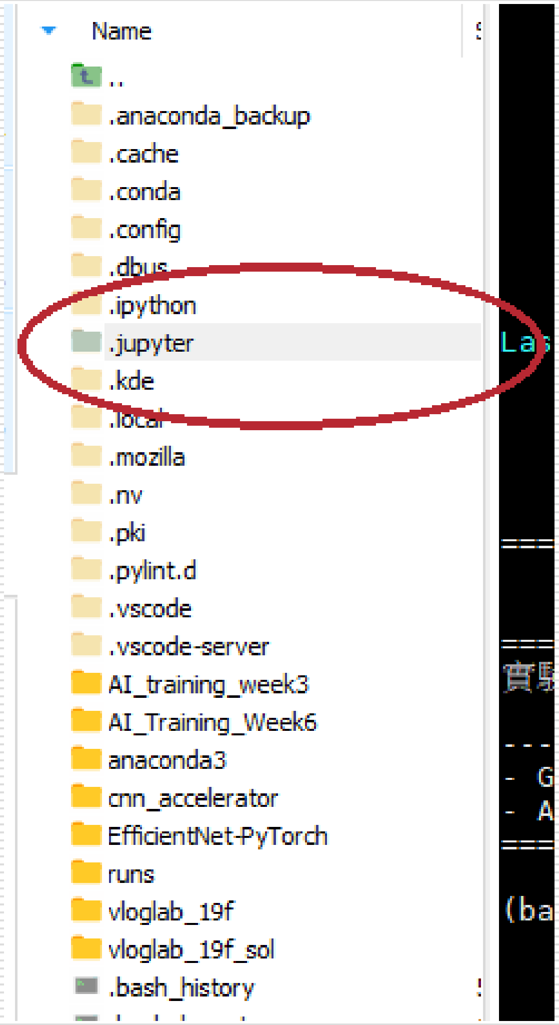
```
(torch) [2024TA@eng05 ~]$ conda install -c anaconda notebook
```



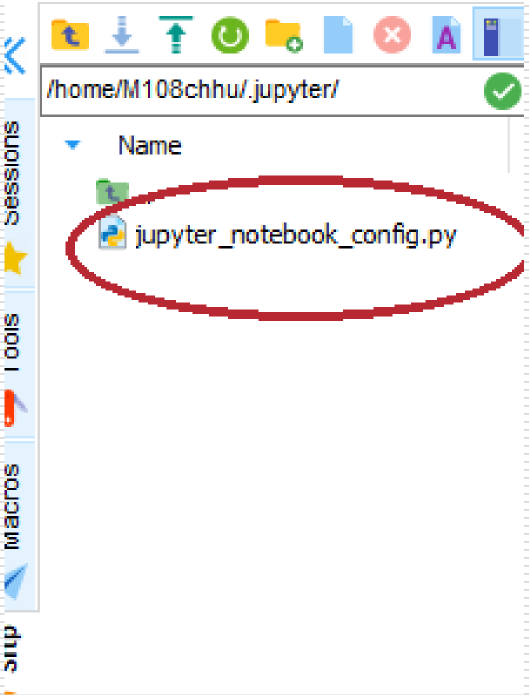
```
Proceed ([y]/n)? y


Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(torch) [2024TA@eng05 ~]$
```

Step 3 (1/2)

- 打開jupyter_notebook_config.py



Name
..
.anaconda_backup
.cache
.conda
.config
.dbus
.ipython
.jupyter
.kde
.local
.mozilla
.nv
.pki
.pylint.d
.vscode
.vscode-server
AI_training_week3
AI_Training_Week6
anaconda3
cnn_accelerator
EfficientNet-PyTorch
runs
vloglab_19f
vloglab_19f_sol
.bash_history

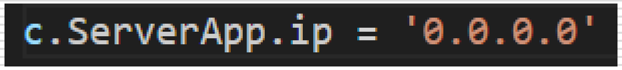


/home/M108chhu/.jupyter/

Name
jupyter_notebook_config.py

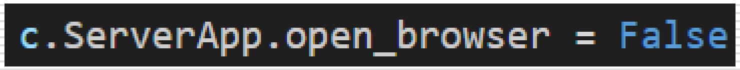
Step 3 (2/2)

- 修改以下四行並儲存 (利用Ctrl+F搜尋)

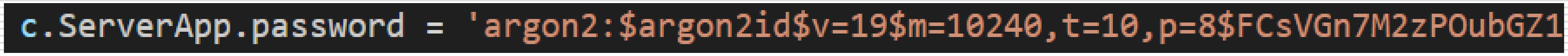


```
c.ServerApp.ip = '0.0.0.0'
```

※ 四行都要記得取消註解
※ 四行前面都不要留空格

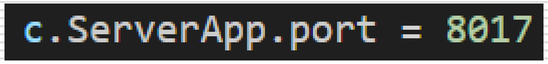


```
c.ServerApp.open_browser = False
```



```
c.ServerApp.password = 'argon2:$argon2id$v=19$m=10240,t=10,p=8$FCsVGn7M2zPOubGZ1
```

你剛剛複製的東西

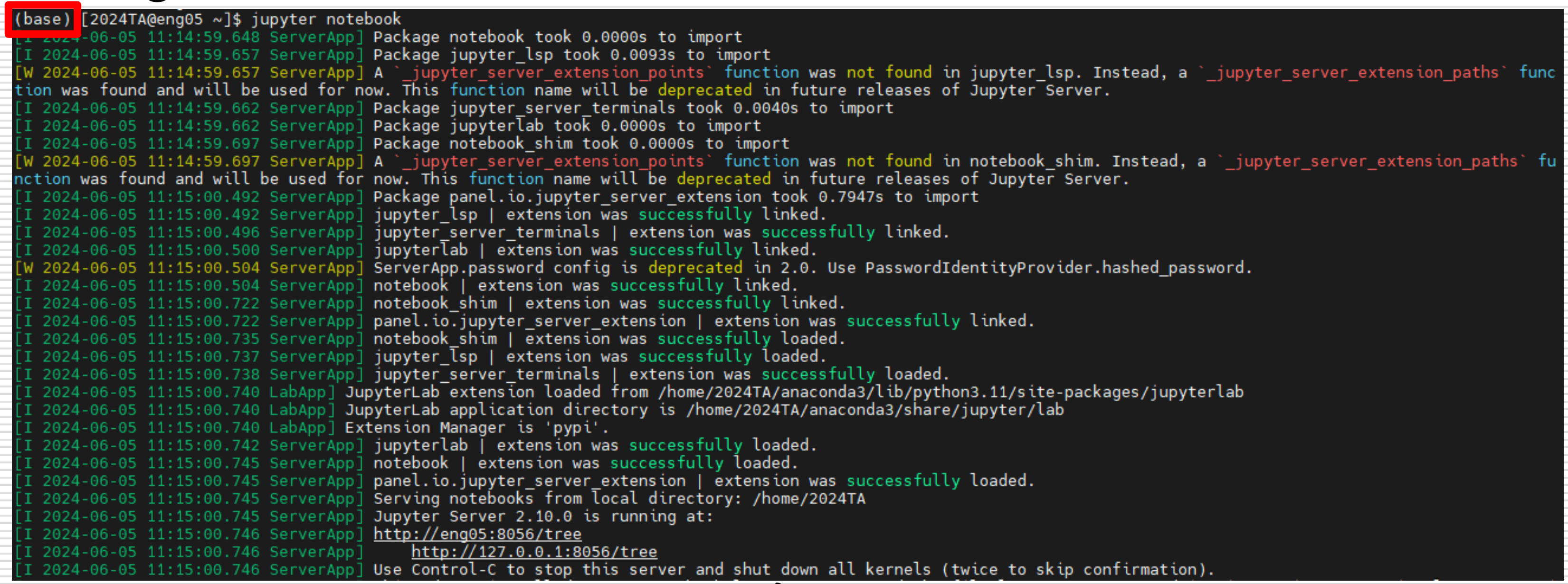


```
c.ServerApp.port = 8017
```

8000—8099挑一個
盡量不要跟別人衝到

Step 4

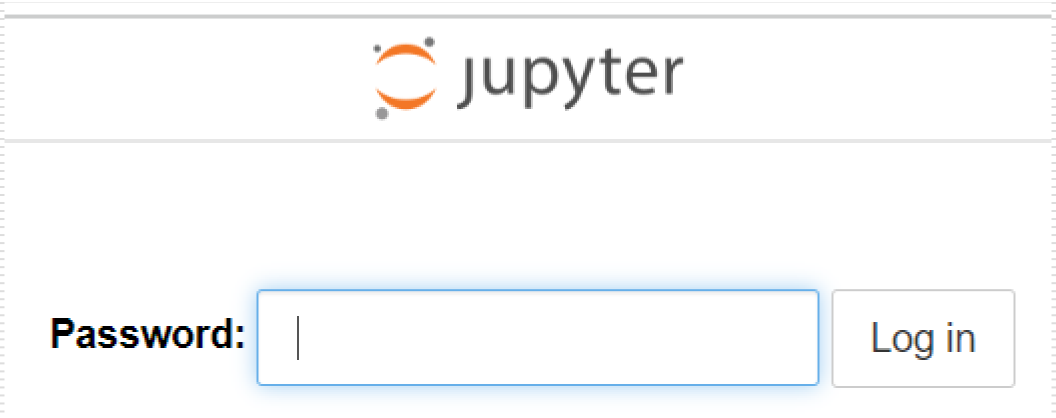
- 回到terminal，在base環境打 jupyter notebook
- 打開瀏覽器，網址輸入140.113.225.245:[你挑的port]
  - E.g. 140.113.225.245:8017



```
(base) [2024TA@eng05 ~]$ jupyter notebook
[I 2024-06-05 11:14:59.648 ServerApp] Package notebook took 0.0000s to import
[I 2024-06-05 11:14:59.657 ServerApp] Package jupyter_lsp took 0.0093s to import
[W 2024-06-05 11:14:59.657 ServerApp] A `_jupyter_server_extension_points` function was not found in jupyter_lsp. Instead, a `_jupyter_server_extension_paths` func
tion was found and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
[I 2024-06-05 11:14:59.662 ServerApp] Package jupyter_server_terminals took 0.0040s to import
[I 2024-06-05 11:14:59.662 ServerApp] Package jupyterlab took 0.0000s to import
[I 2024-06-05 11:14:59.697 ServerApp] Package notebook_shim took 0.0000s to import
[W 2024-06-05 11:14:59.697 ServerApp] A `_jupyter_server_extension_points` function was not found in notebook_shim. Instead, a `_jupyter_server_extension_paths` fu
nction was found and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
[I 2024-06-05 11:15:00.492 ServerApp] Package panel.io.jupyter_server_extension took 0.7947s to import
[I 2024-06-05 11:15:00.492 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-06-05 11:15:00.496 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-06-05 11:15:00.500 ServerApp] jupyterlab | extension was successfully linked.
[W 2024-06-05 11:15:00.504 ServerApp] ServerApp.password config is deprecated in 2.0. Use PasswordIdentityProvider.hashed_password.
[I 2024-06-05 11:15:00.504 ServerApp] notebook | extension was successfully linked.
[I 2024-06-05 11:15:00.722 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-06-05 11:15:00.722 ServerApp] panel.io.jupyter_server_extension | extension was successfully linked.
[I 2024-06-05 11:15:00.735 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-06-05 11:15:00.737 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-06-05 11:15:00.738 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-06-05 11:15:00.740 LabApp] JupyterLab extension loaded from /home/2024TA/anaconda3/lib/python3.11/site-packages/jupyterlab
[I 2024-06-05 11:15:00.740 LabApp] JupyterLab application directory is /home/2024TA/anaconda3/share/jupyter/lab
[I 2024-06-05 11:15:00.740 LabApp] Extension Manager is 'pypi'.
[I 2024-06-05 11:15:00.742 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-06-05 11:15:00.745 ServerApp] notebook | extension was successfully loaded.
[I 2024-06-05 11:15:00.745 ServerApp] panel.io.jupyter_server_extension | extension was successfully loaded.
[I 2024-06-05 11:15:00.745 ServerApp] Serving notebooks from local directory: /home/2024TA
[I 2024-06-05 11:15:00.745 ServerApp] Jupyter Server 2.10.0 is running at:
[I 2024-06-05 11:15:00.746 ServerApp] http://eng05:8056/tree
[I 2024-06-05 11:15:00.746 ServerApp]     http://127.0.0.1:8056/tree
[I 2024-06-05 11:15:00.746 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

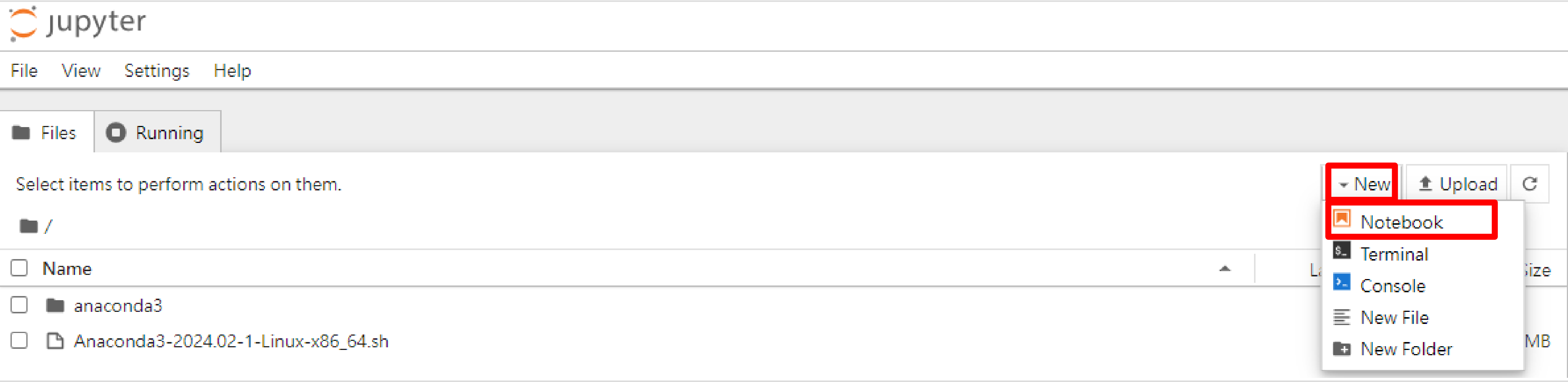
- 在用jupyter notebook的時候不要在terminal按到ctrl+c (不要不小心關掉)
- 設定一次就可以了，下次不需要重新設定config

Jupyter Notebook (1/4)



輸入你設定的密碼

Jupyter Notebook (2/4)



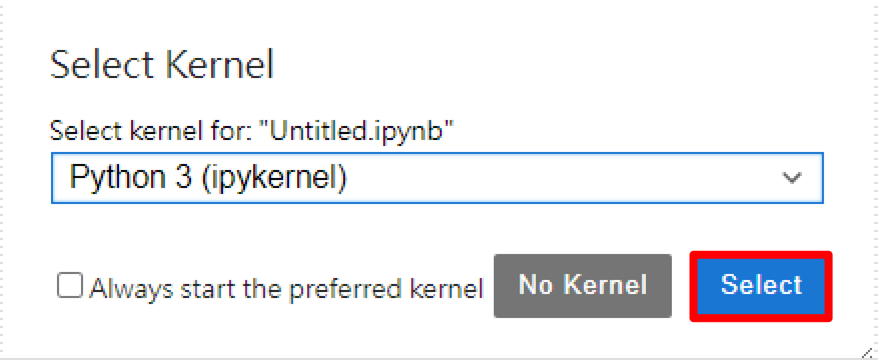
Select items to perform actions on them.

/

anaconda3

Anaconda3-2024.02-1-Linux-x86_64.sh

MB

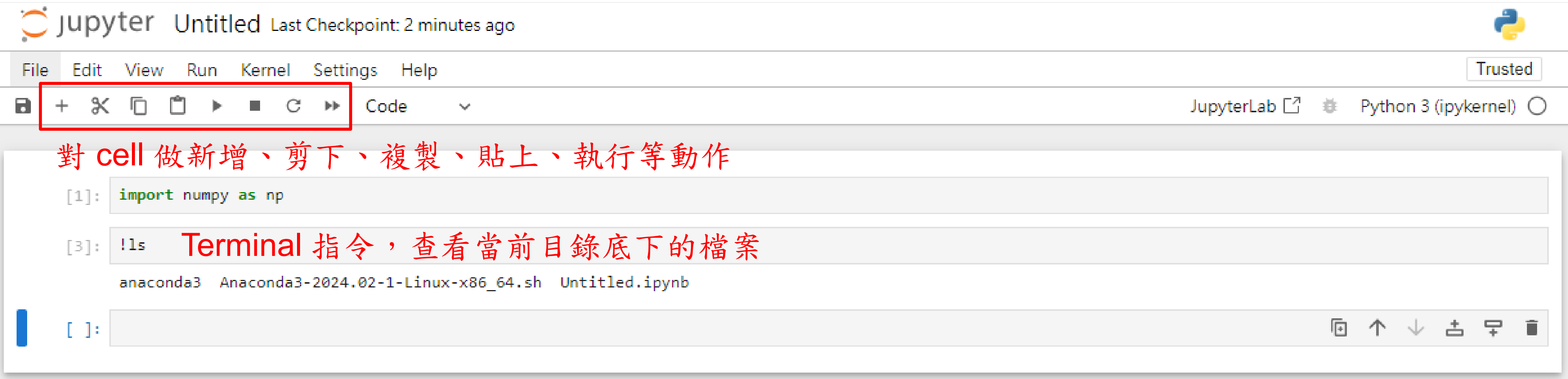


Select Kernel

Select kernel for: "Untitled.ipynb"

Python 3 (ipykernel)

Jupyter Notebook (3/4)

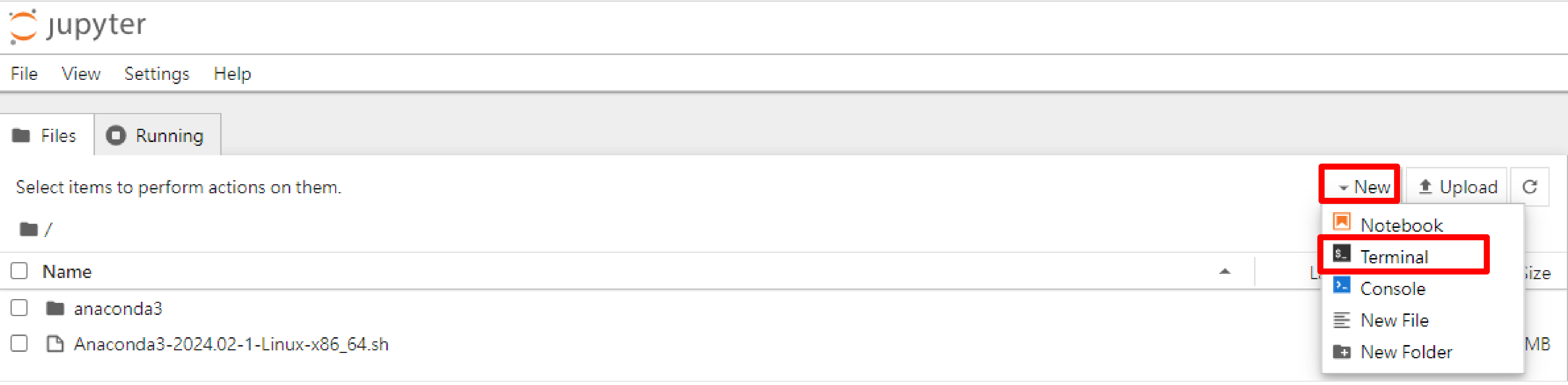

jupyter Untitled Last Checkpoint: 2 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
對 cell 做新增、剪下、複製、貼上、執行等動作
Terminal 指令，查看當前目錄底下的檔案

```
[1]: import numpy as np

[3]: !ls
anaconda3  Anaconda3-2024.02-1-Linux-x86_64.sh  Untitled.ipynb

[ ]:
```

- 執行cell: shift+enter
- Terminal指令：指令前面加！

Jupyter Notebook (4/4)

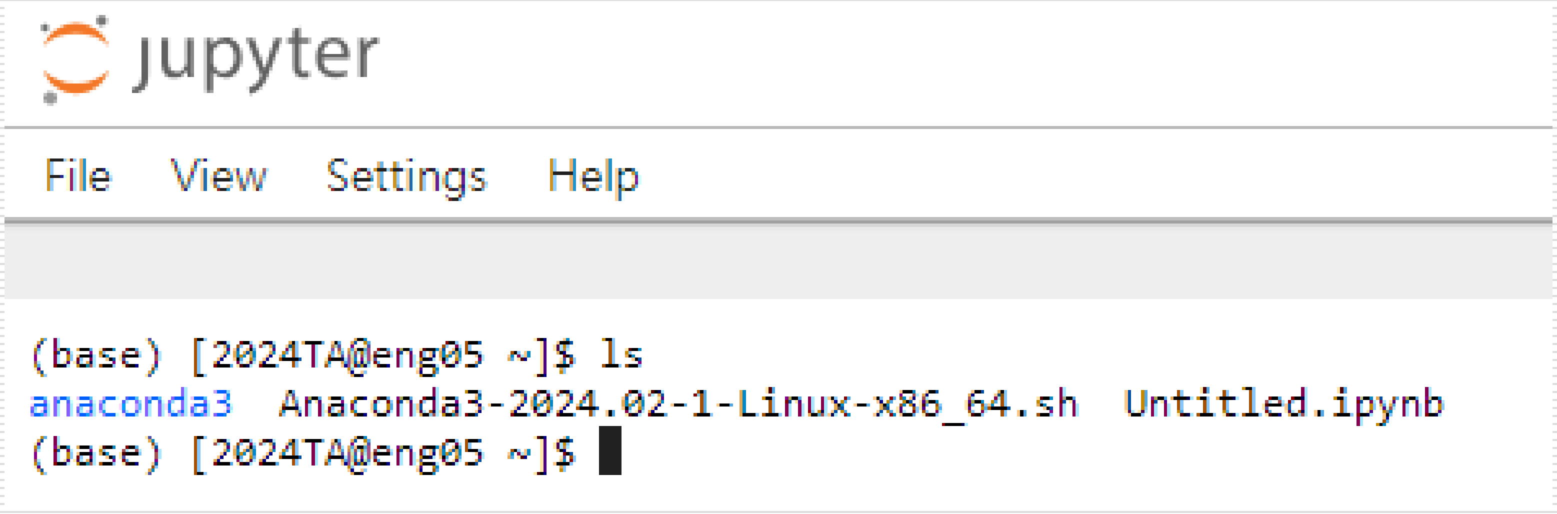
- Terminal on Jupyter



Select items to perform actions on them.

/

Name
anaconda3
Anaconda3-2024.02-1-Linux-x86_64.sh



```
(base) [2024TA@eng05 ~]$ ls
anaconda3  Anaconda3-2024.02-1-Linux-x86_64.sh  Untitled.ipynb
(base) [2024TA@eng05 ~]$ 
```

Introduction to Python

Python 特性

- 強調代碼的可讀性和簡潔的語法 (注重空格縮排)
- 直譯式語言(一行一行動態執行)
- 能夠自動管理記憶體使用
- 容易整合其他底層語言 (glue language)
- 完備的標準庫和套件
- 應用廣 (讀寫檔案 、機器學習 、網站開發)

語法差異

C++

需要分號結尾

用大括號決定語意

C++ for loop

```
for(int i-0; i<100; i++){
    cout << i;
}
```

C++ if else

```
if (x>10){
    ......
}else if (x>0){
    ......
}else{
    ......
}
```

Python

不須分號結尾

用縮排決定語意

Python for loop

```
for i in range(100):
    print(i)
```

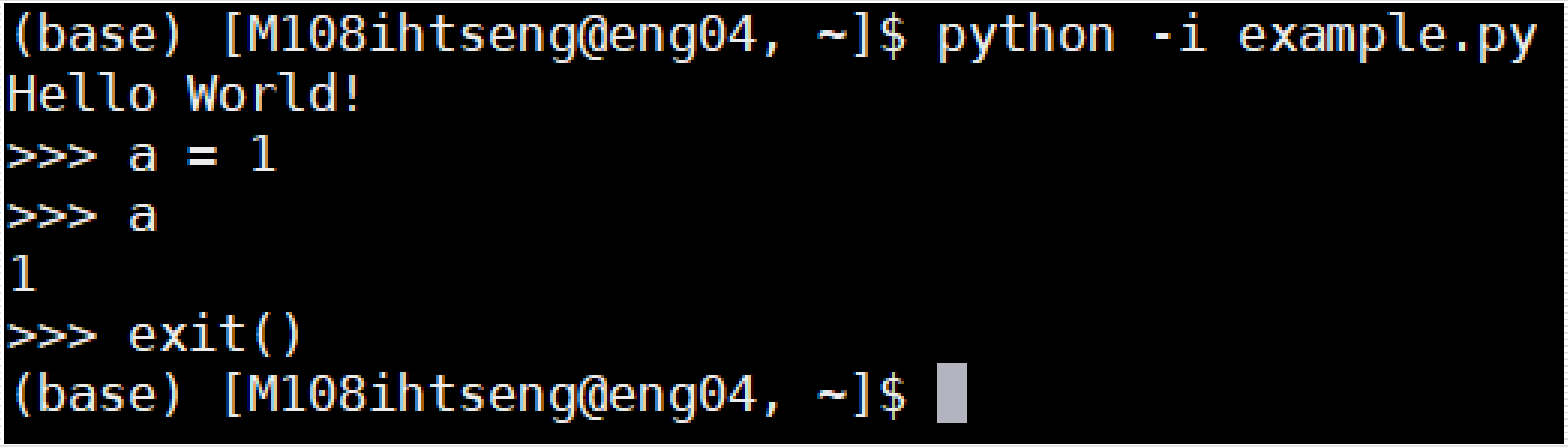
May cause indentation error

Python if else

```
if x>10:
    ......
elif x>0:
    ......
else:
    ......
```

執行 .py File

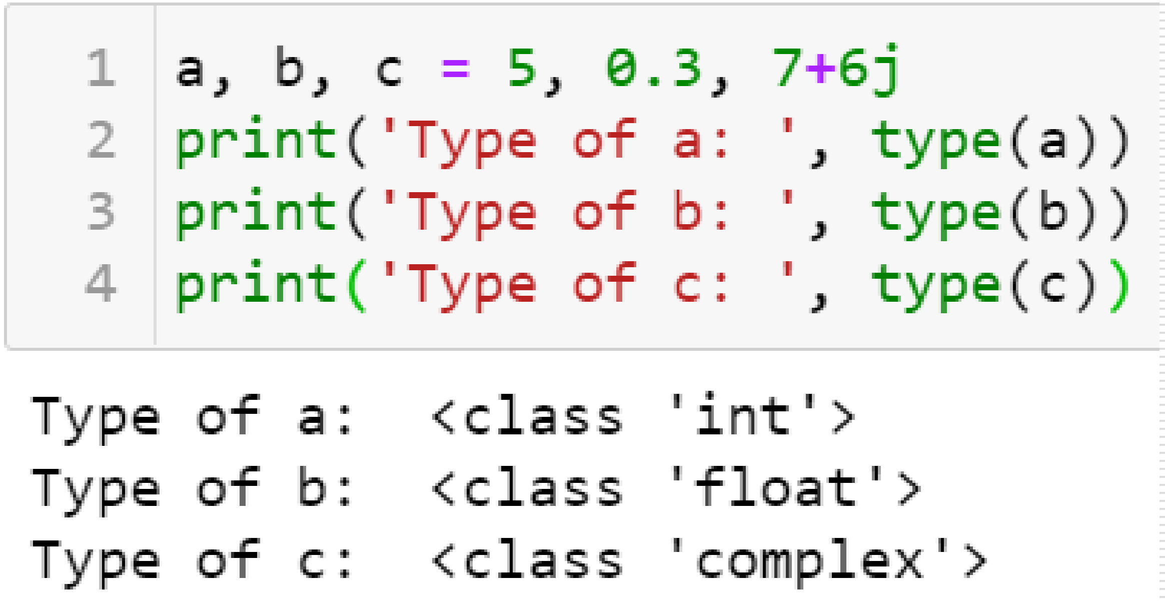
- python example.py
- python -i example.py
  - 執行完程式後留在python控制台
  - 互動式窗口，debug用



```
(base) [M108ihtseng@eng04, ~]$ python -i example.py
Hello World!
>>> a = 1
>>> a
1
>>> exit()
(base) [M108ihtseng@eng04, ~]$ 
```

Variables (1/4)

- 不須事先宣告，直接用”=”賦值
- Number
  - Python3支持 int, float, complex



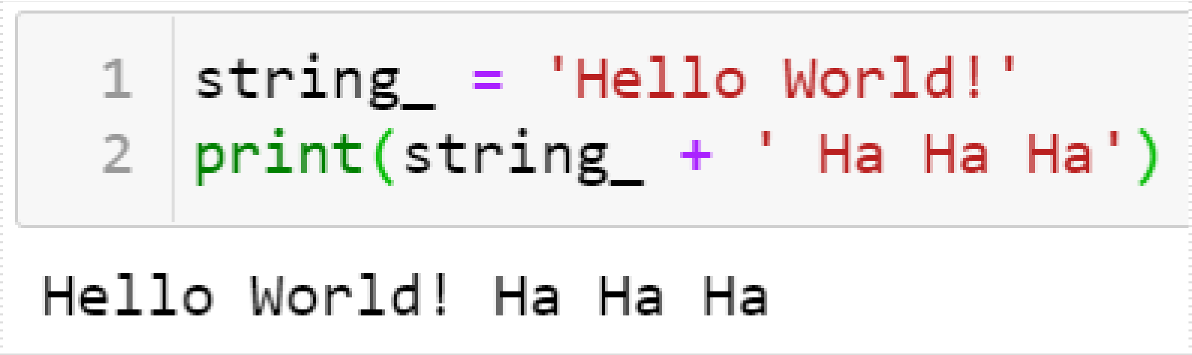
```python
a, b, c = 5, 0.3, 7+6j
print('Type of a: ', type(a))
print('Type of b: ', type(b))
print('Type of c: ', type(c))
```

```
Type of a:  <class 'int'>
Type of b:  <class 'float'>
Type of c:  <class 'complex'>
```

  - 數值運算：＋(加), -(減), *(乘), /(除,得到浮點數), // (除,得到整數), %(取餘數), **(次方)

Variables (2/4)

- String



```
string_ = 'Hello World!'
print(string_ + ' Ha Ha Ha')
```

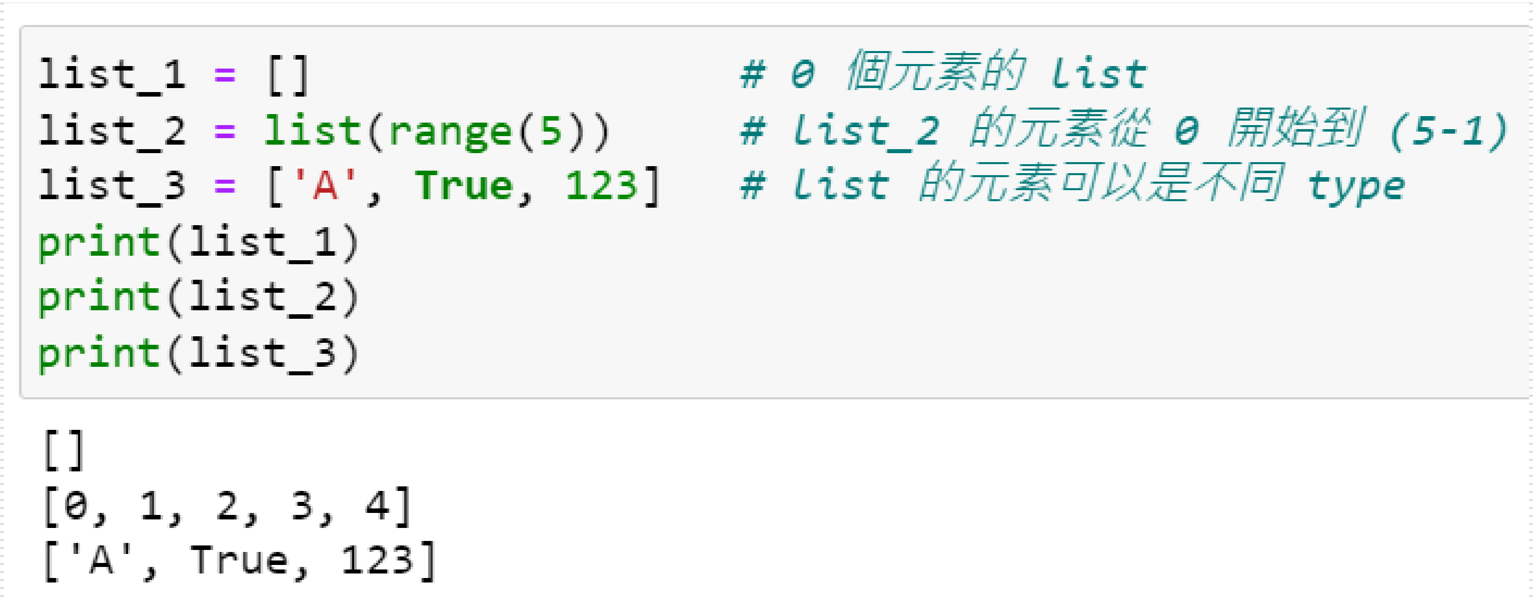
```
Hello World! Ha Ha Ha
```

- Boolean
  - True, False

List (1/4)

- List
  - 由 0 或多個元素組成

- Assignment

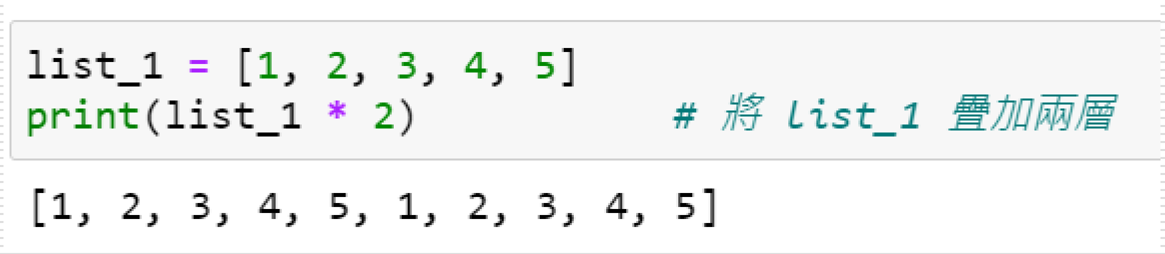


```
list_1 = []                 # 0 個元素的 list
list_2 = list(range(5))     # list_2 的元素從 0 開始到 (5-1)
list_3 = ['A', True, 123]   # list 的元素可以是不同 type
print(list_1)
print(list_2)
print(list_3)
```

```
[]
[0, 1, 2, 3, 4]
['A', True, 123]
```

List (2/4)

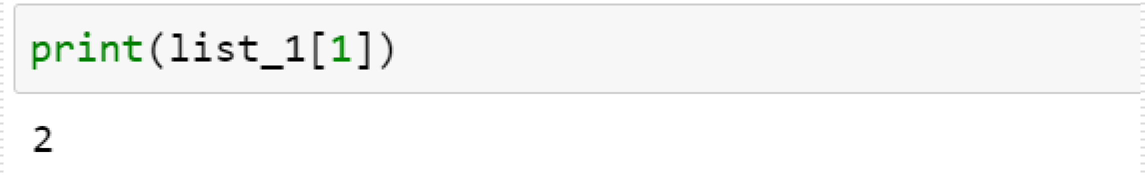
- 串列重複



```
list_1 = [1, 2, 3, 4, 5]
print(list_1 * 2)             # 將 list_1 疊加兩層
```

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

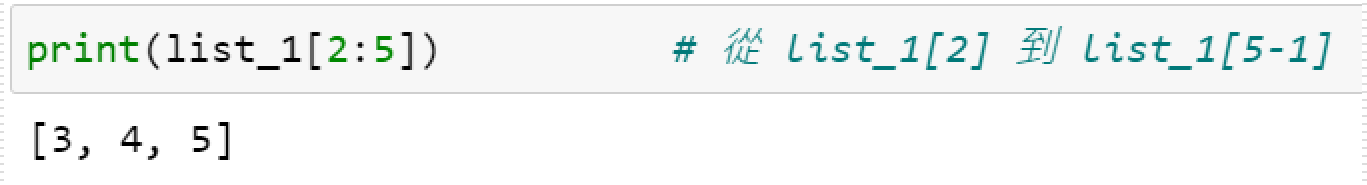
- 取得元素值



```
print(list_1[1])
```

```
2
```

- 切片

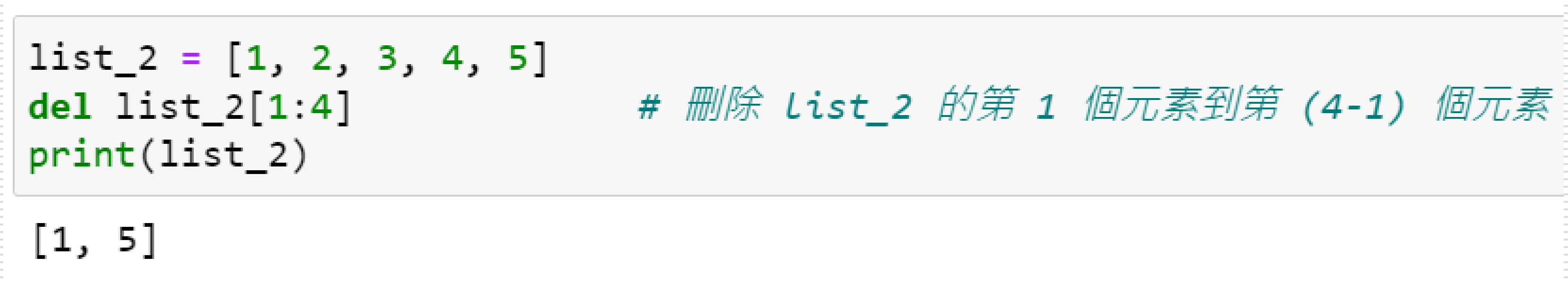


```
print(list_1[2:5])            # 從 list_1[2] 到 list_1[5-1]
```

```
[3, 4, 5]
```

List (3/4)

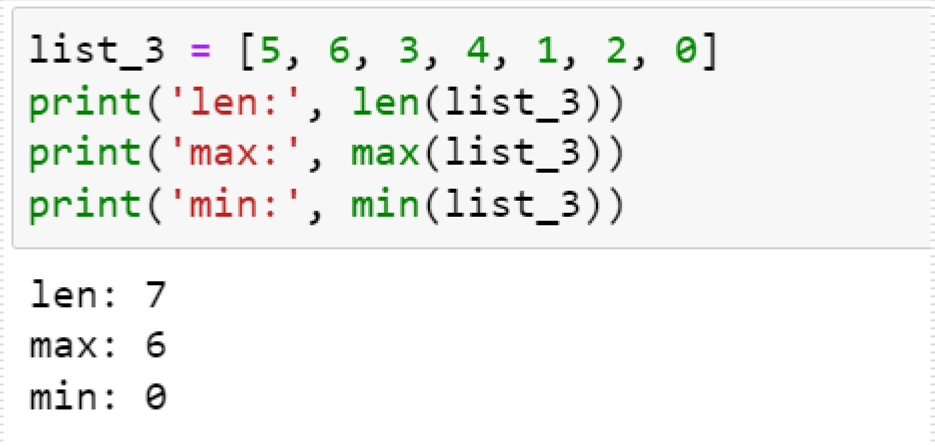
- 刪除元素



```
list_2 = [1, 2, 3, 4, 5]
del list_2[1:4]           # 刪除 list_2 的第 1 個元素到第 (4-1) 個元素
print(list_2)
```

```
[1, 5]
```

- List 資訊

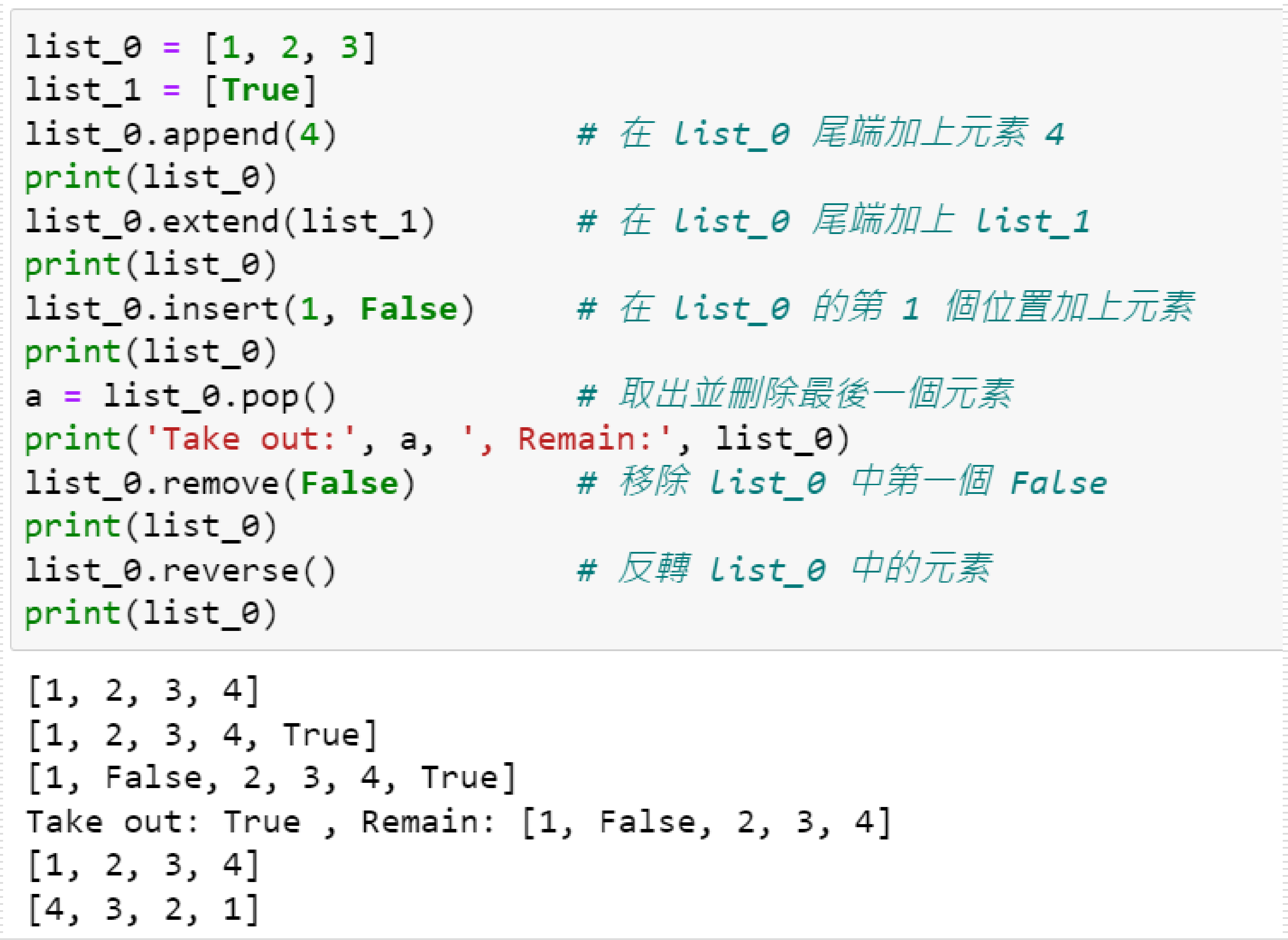


```
list_3 = [5, 6, 3, 4, 1, 2, 0]
print('len:', len(list_3))
print('max:', max(list_3))
print('min:', min(list_3))
```

```
len: 7
max: 6
min: 0
```

List (4/4)

- List 函數

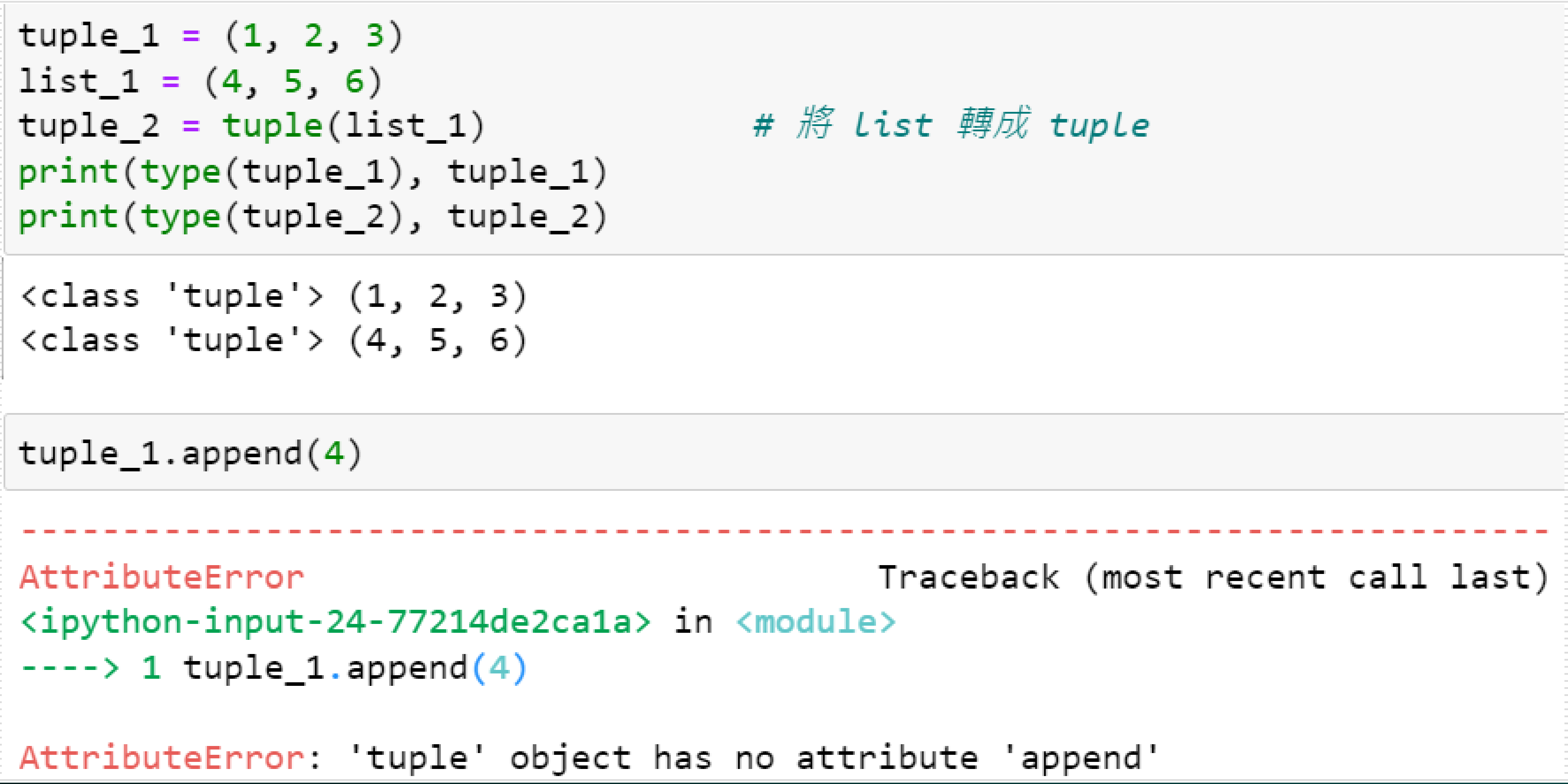


```
list_0 = [1, 2, 3]
list_1 = [True]
list_0.append(4)            # 在 list_0 尾端加上元素 4
print(list_0)
list_0.extend(list_1)       # 在 list_0 尾端加上 list_1
print(list_0)
list_0.insert(1, False)     # 在 list_0 的第 1 個位置加上元素
print(list_0)
a = list_0.pop()            # 取出並刪除最後一個元素
print('Take out:', a, ', Remain:', list_0)
list_0.remove(False)        # 移除 list_0 中第一個 False
print(list_0)
list_0.reverse()            # 反轉 list_0 中的元素
print(list_0)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, True]
[1, False, 2, 3, 4, True]
Take out: True , Remain: [1, False, 2, 3, 4]
[1, 2, 3, 4]
[4, 3, 2, 1]
```

Tuple

- Tuple
  - 類似list，但宣告後不能修改
  - 執行速度比 list 快，儲存的資料沒有被修改的風險



```
tuple_1 = (1, 2, 3)
list_1 = (4, 5, 6)
tuple_2 = tuple(list_1)                 # 將 list 轉成 tuple
print(type(tuple_1), tuple_1)
print(type(tuple_2), tuple_2)
```

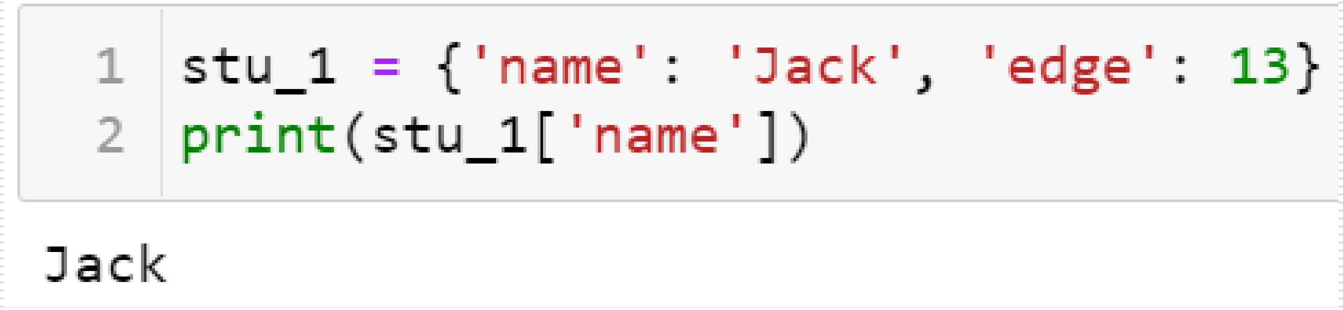
```
<class 'tuple'> (1, 2, 3)
<class 'tuple'> (4, 5, 6)
```

```
tuple_1.append(4)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-24-77214de2ca1a> in <module>
----> 1 tuple_1.append(4)

AttributeError: 'tuple' object has no attribute 'append'
```

Dictionary

- Dictionary
  - list, tuple 都是用數字取得元素值 (ex. list_1[0])，dictionary 可以用 key 取值
  - 語法：{key1: value1, key2: value2, …}



```
stu_1 = {'name': 'Jack', 'edge': 13}
print(stu_1['name'])
```

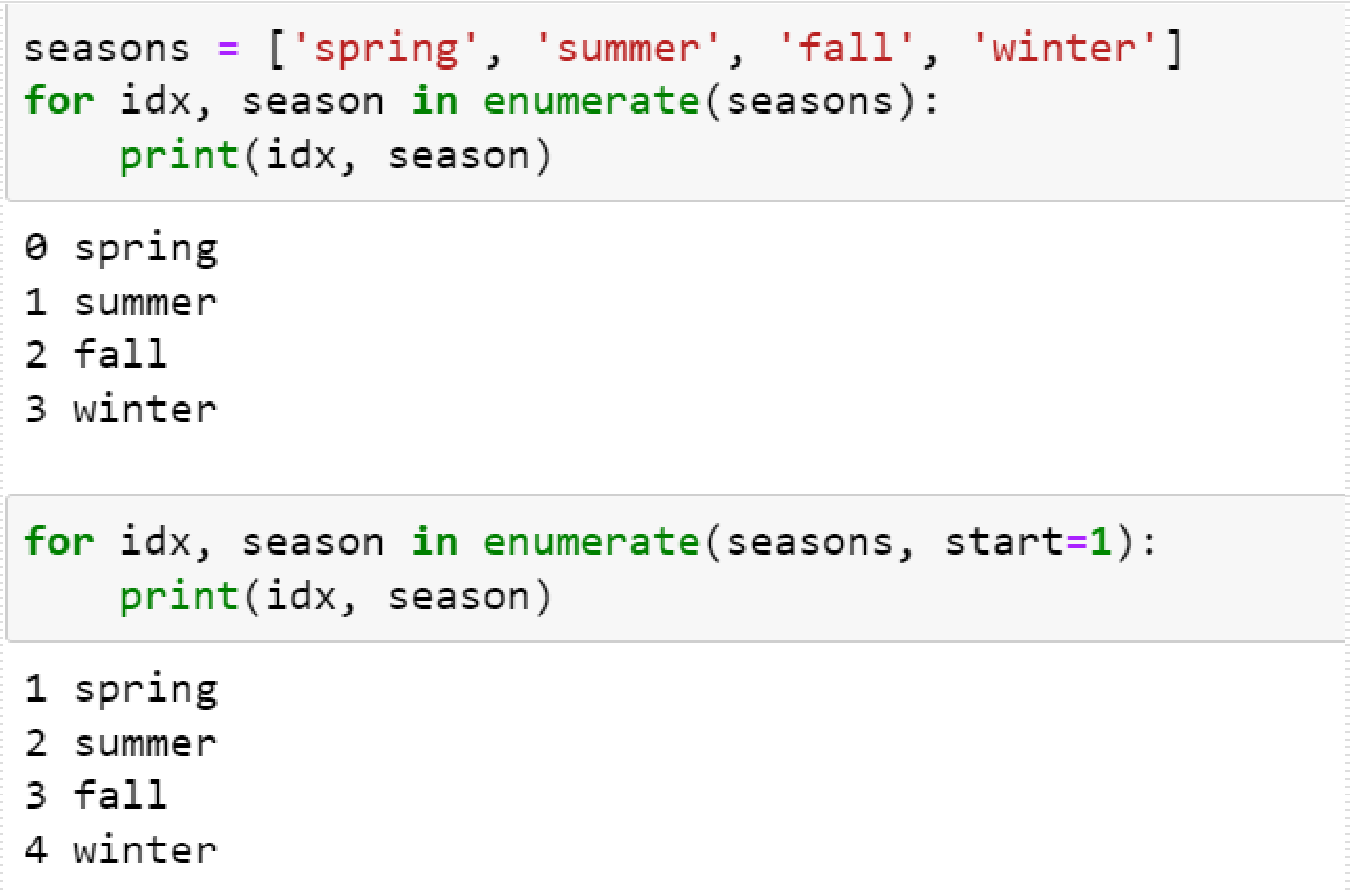
```
Jack
```

Range

- 用來創建整數列表
- range(結束值)
  - range(5) => [0,1,2,3,4]
  - range(0) => [ ]
- range(起始值, 結束值)
  - range(1, 5) => [1,2,3,4]
- range(起始值, 結束值, 間隔)
  - range(0,10,2) => [0, 2, 4, 6, 8]
  - range(10,1,-2) => [10, 8, 6, 4, 2]

Enumerate

- Enumerate 用於將數據列表組合為一個索引序列，可以同時列出數據和數據索引，通常用於 for loop



```python
seasons = ['spring', 'summer', 'fall', 'winter']
for idx, season in enumerate(seasons):
    print(idx, season)
```

```
0 spring
1 summer
2 fall
3 winter
```

```python
for idx, season in enumerate(seasons, start=1):
    print(idx, season)
```

```
1 spring
2 summer
3 fall
4 winter
```

Control Flow

- if 條件1 :

      …

  elif 條件2 :

      …

  else:

      …

- for x in range(5):

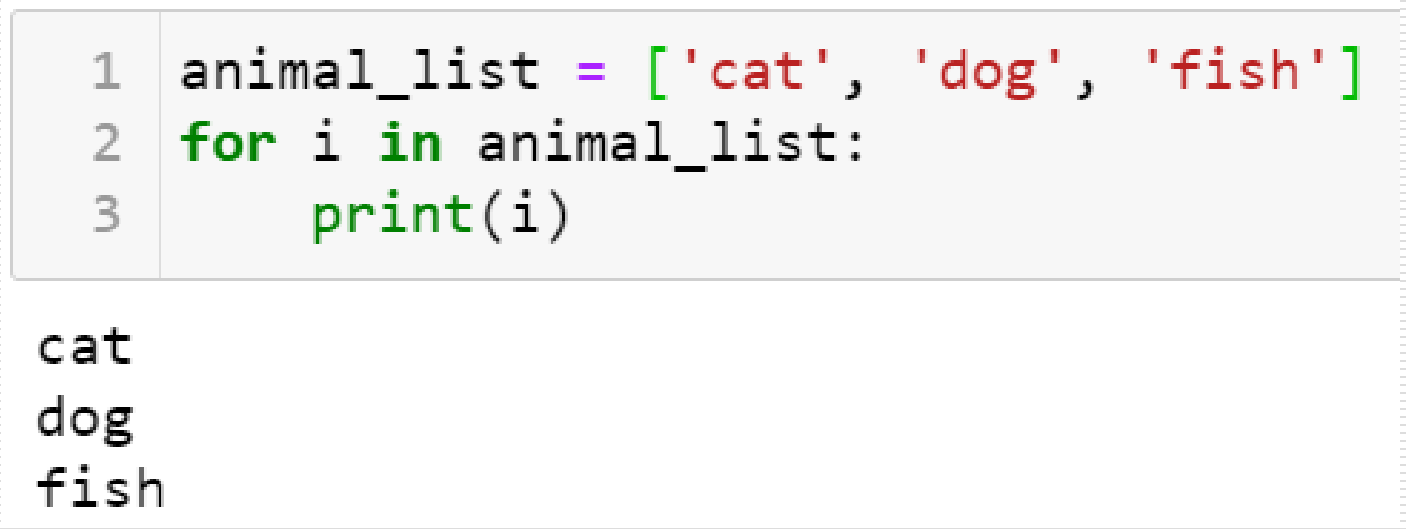
      …

- while:

      …

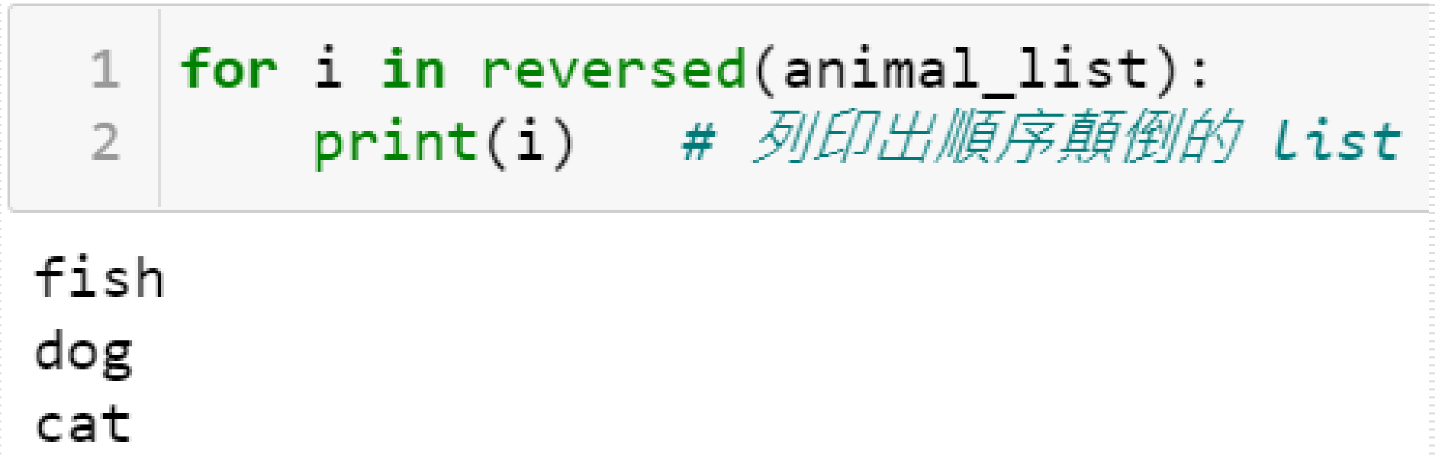
- break
- continue

For Loop (1/2)



```python
animal_list = ['cat', 'dog', 'fish']
for i in animal_list:
    print(i)
```

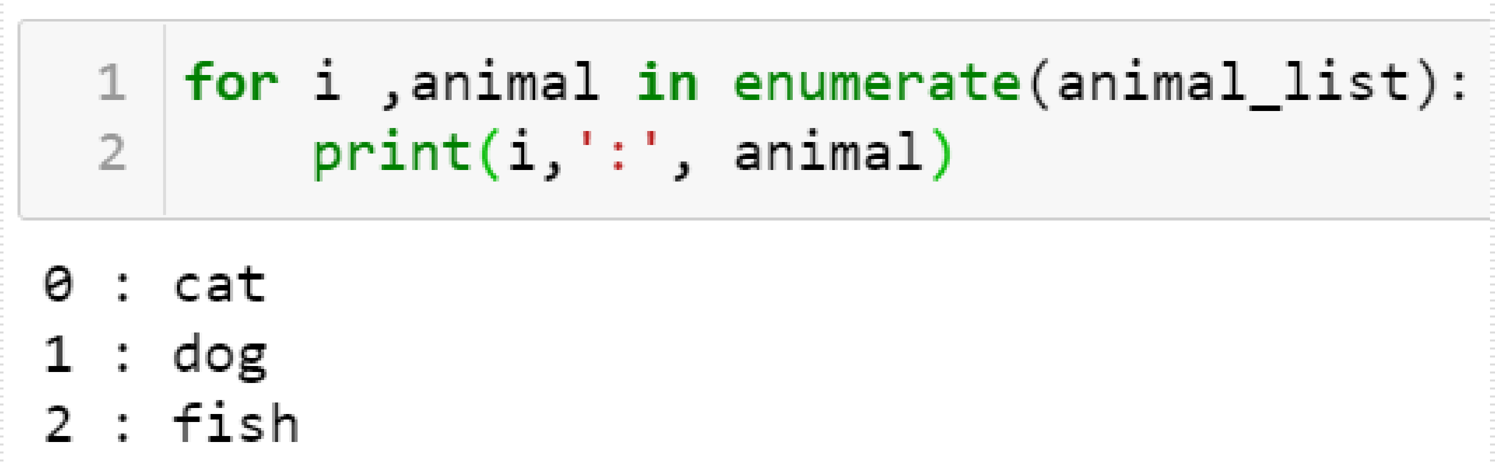
```
cat
dog
fish
```



```python
for i in reversed(animal_list):
    print(i)    # 列印出順序顛倒的 List
```

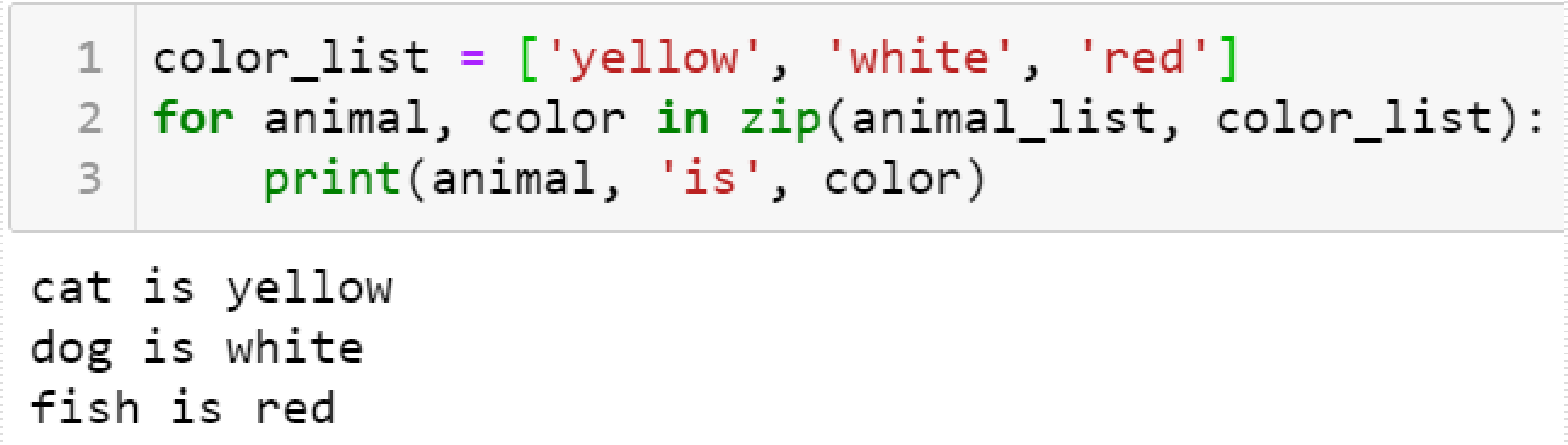
```
fish
dog
cat
```

For Loop (2/2)



```python
for i ,animal in enumerate(animal_list):
    print(i,':', animal)
```

```
0 : cat
1 : dog
2 : fish
```

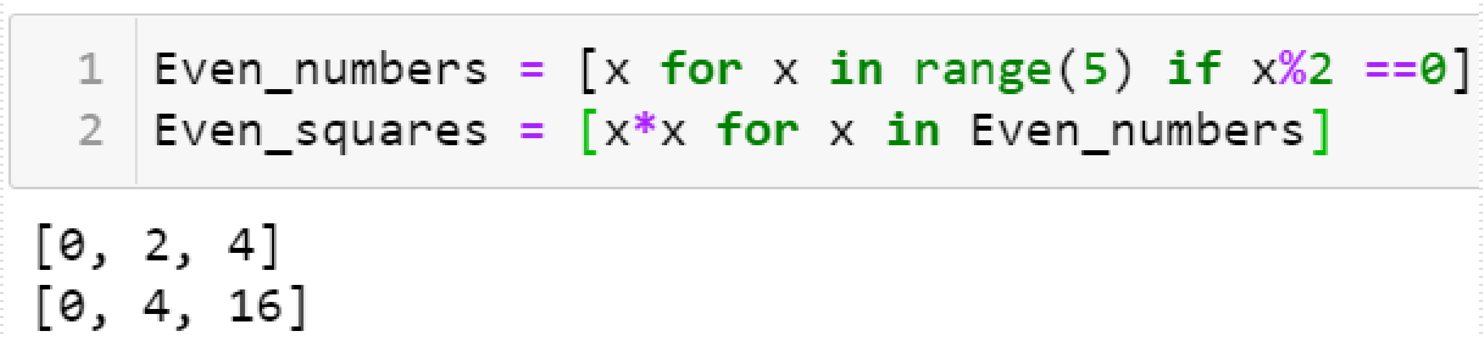


```python
color_list = ['yellow', 'white', 'red']
for animal, color in zip(animal_list, color_list):
    print(animal, 'is', color)
```

```
cat is yellow
dog is white
fish is red
```

Advanced Usage of List

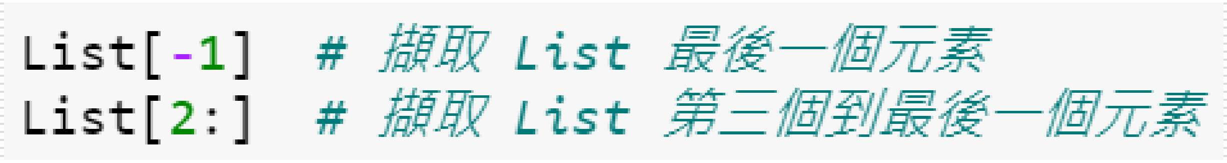
- 搭配 for loop



```
Even_numbers = [x for x in range(5) if x%2 ==0]
Even_squares = [x*x for x in Even_numbers]
```

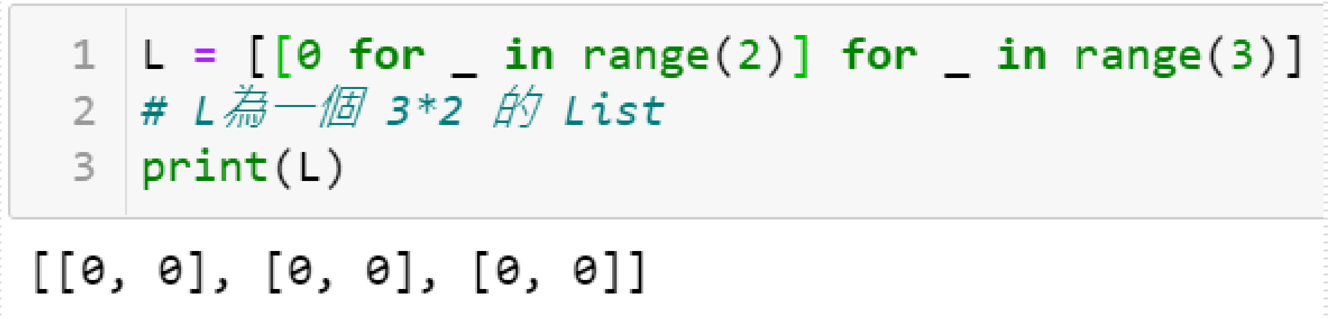
```
[0, 2, 4]
[0, 4, 16]
```

- Indexing



```
List[-1]  # 擷取 List 最後一個元素
List[2:]  # 擷取 List 第三個到最後一個元素
```

- 創建多維 list



```
L = [[0 for _ in range(2)] for _ in range(3)]
# L為一個 3*2 的 List
print(L)
```

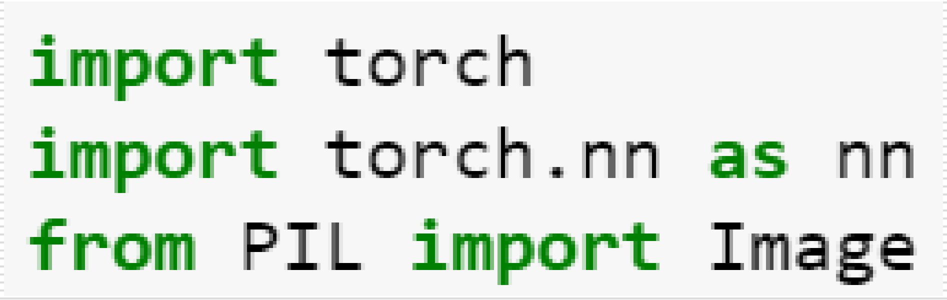
```
[[0, 0], [0, 0], [0, 0]]
```

Import Module

- import module1
  - module1.func()

- from module1 import func
  - func()

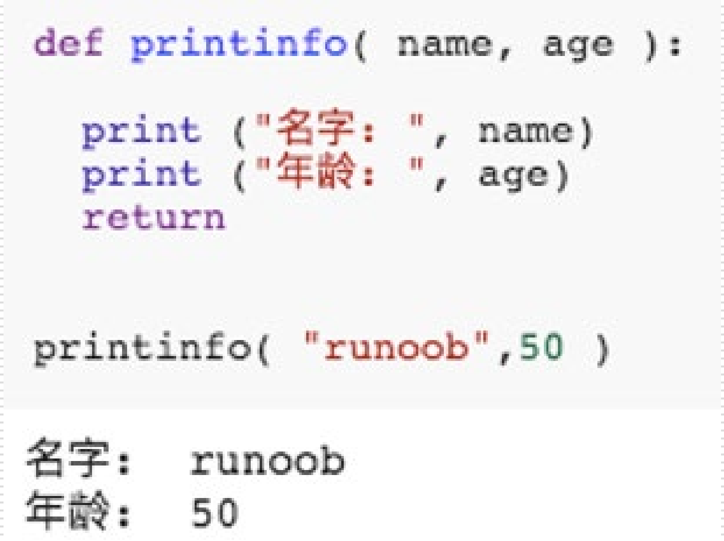
- import module1 as m1
  - m1.func()



```
import torch
import torch.nn as nn
from PIL import Image
```

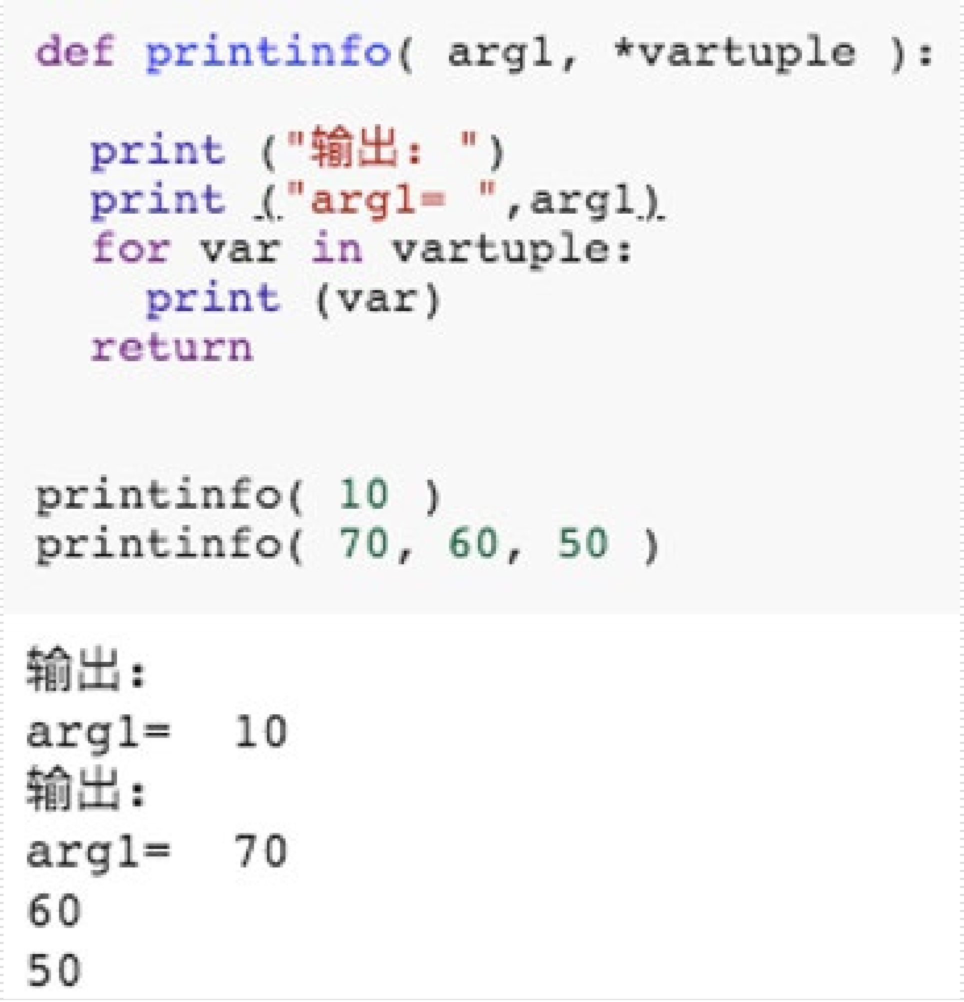
Def (1/2)

- def func_name (parameters, ...):



```
def printinfo( name, age ):

   print ("名字: ", name)
   print ("年龄: ", age)
   return


printinfo( "runoob",50 )
```

```
名字:   runoob
年龄:   50
```

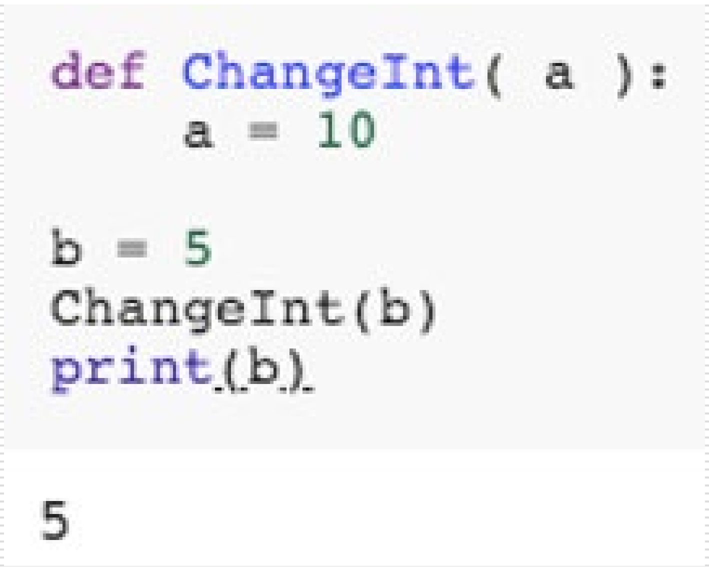


```
def printinfo( arg1, *vartuple ):

   print ("输出: ")
   print ("arg1= ",arg1)
   for var in vartuple:
      print (var)
   return


printinfo( 10 )
printinfo( 70, 60, 50 )
```

```
输出:
arg1=  10
输出:
arg1=  70
60
50
```

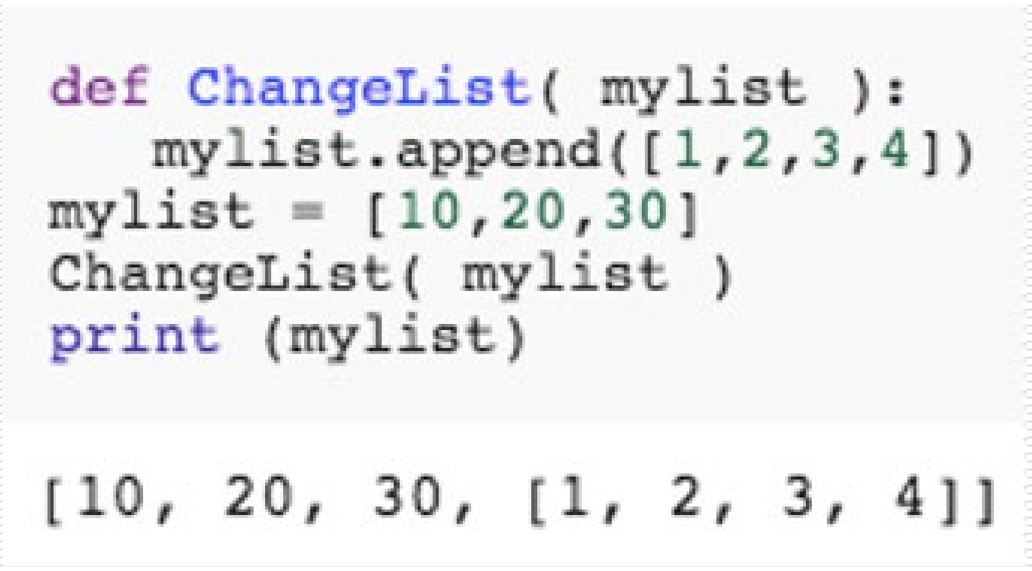
Def (2/2)

- 參數傳遞
  - 不可變(類似pass by value): number、string、tuple
  - 可變(類似pass by reference): list、dictionary



```
def ChangeInt( a ):
    a = 10

b = 5
ChangeInt(b)
print(b)
```

```
5
```

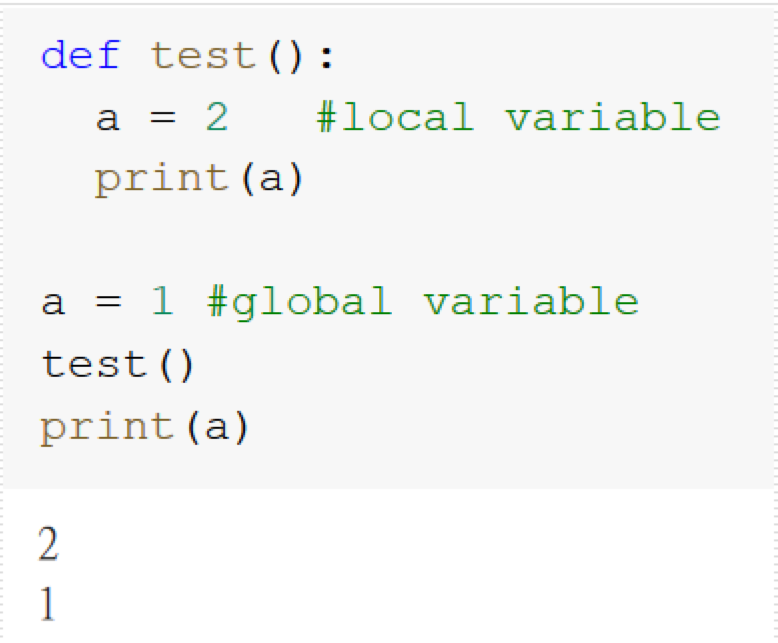


```
def ChangeList( mylist ):
    mylist.append([1,2,3,4])
mylist = [10,20,30]
ChangeList( mylist )
print (mylist)
```

```
[10, 20, 30, [1, 2, 3, 4]]
```

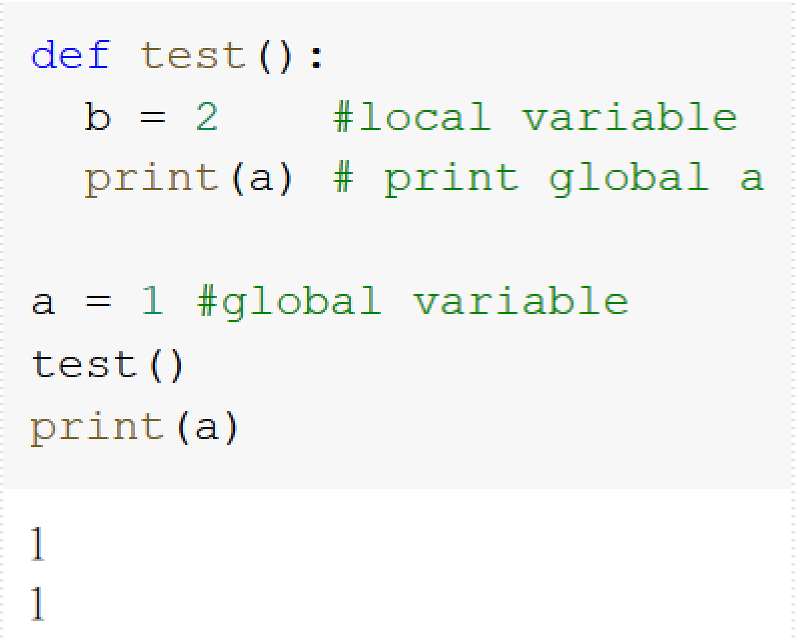
Global/Local Variables (1/2)

- 在函數內建立的變數為local variable, 在函式外建立的變數為global variable
- 若只是使用global variable，則直接用就好
- Local variable的遮蔽效果:



```
def test():
    a = 2    #local variable
    print(a)

a = 1 #global variable
test()
print(a)
```

```
2
1
```

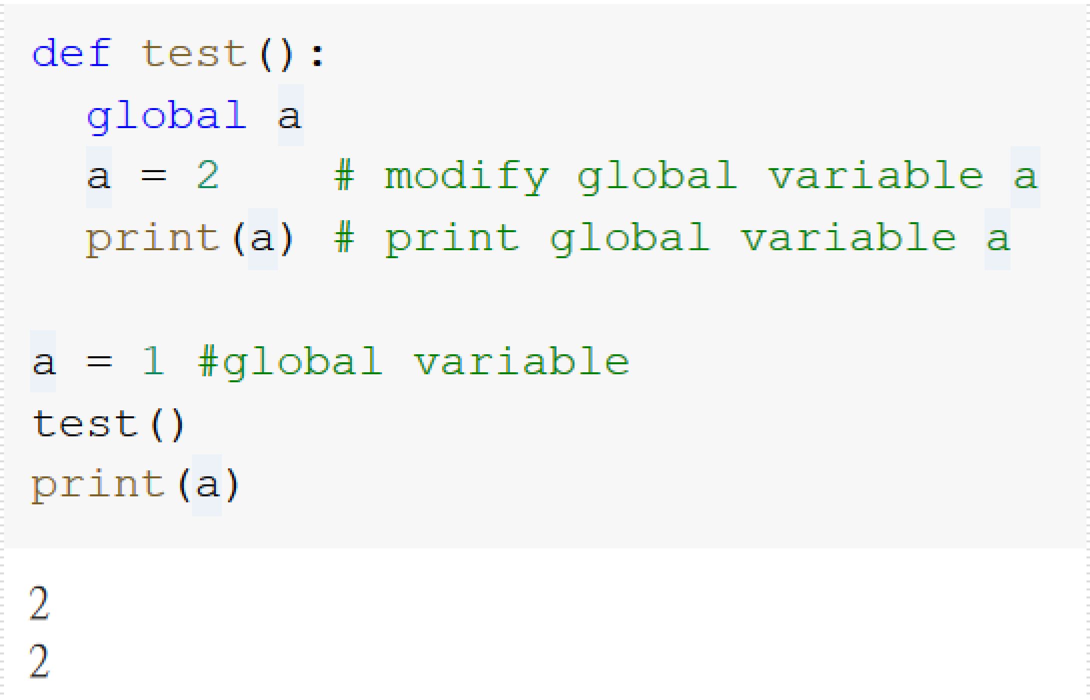


```
def test():
    b = 2     #local variable
    print(a) # print global a

a = 1 #global variable
test()
print(a)
```

```
1
1
```

Global/Local Variables (2/2)

- 如果要在函式裡修改global variable的值，要用關鍵字 global

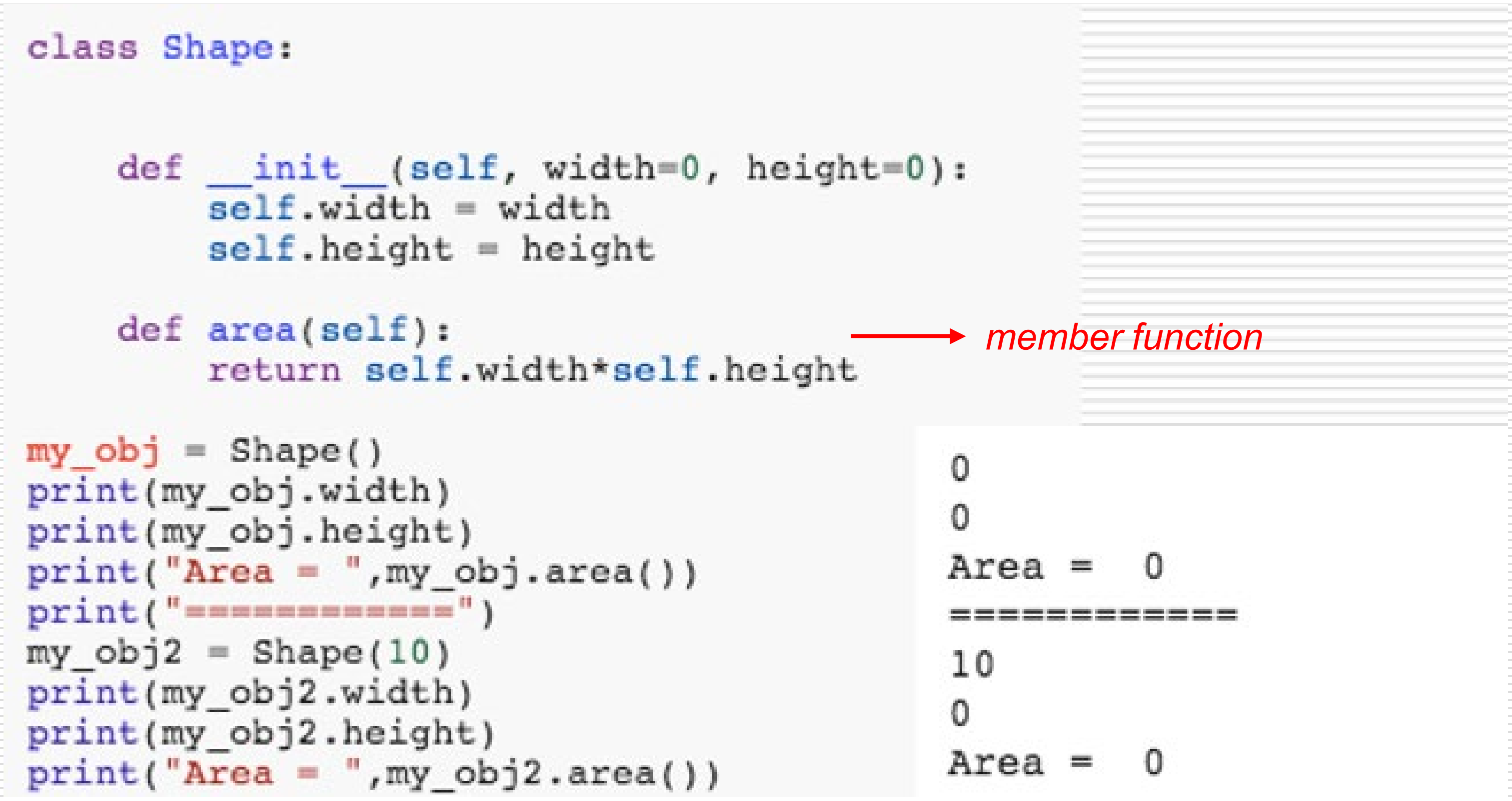


```python
def test():
    global a
    a = 2     # modify global variable a
    print(a) # print global variable a

a = 1 #global variable
test()
print(a)
```

```
2
2
```

Class

- Class class_name:
  - def__init__(self,parameter,…): 建構子 constructor
    - 用來宣告和初始化 class 中的變數
    - self.parameter = … 宣告這個變數的屬性



```python
class Shape:


    def __init__(self, width=0, height=0):
        self.width = width
        self.height = height

    def area(self):
        return self.width*self.height
```

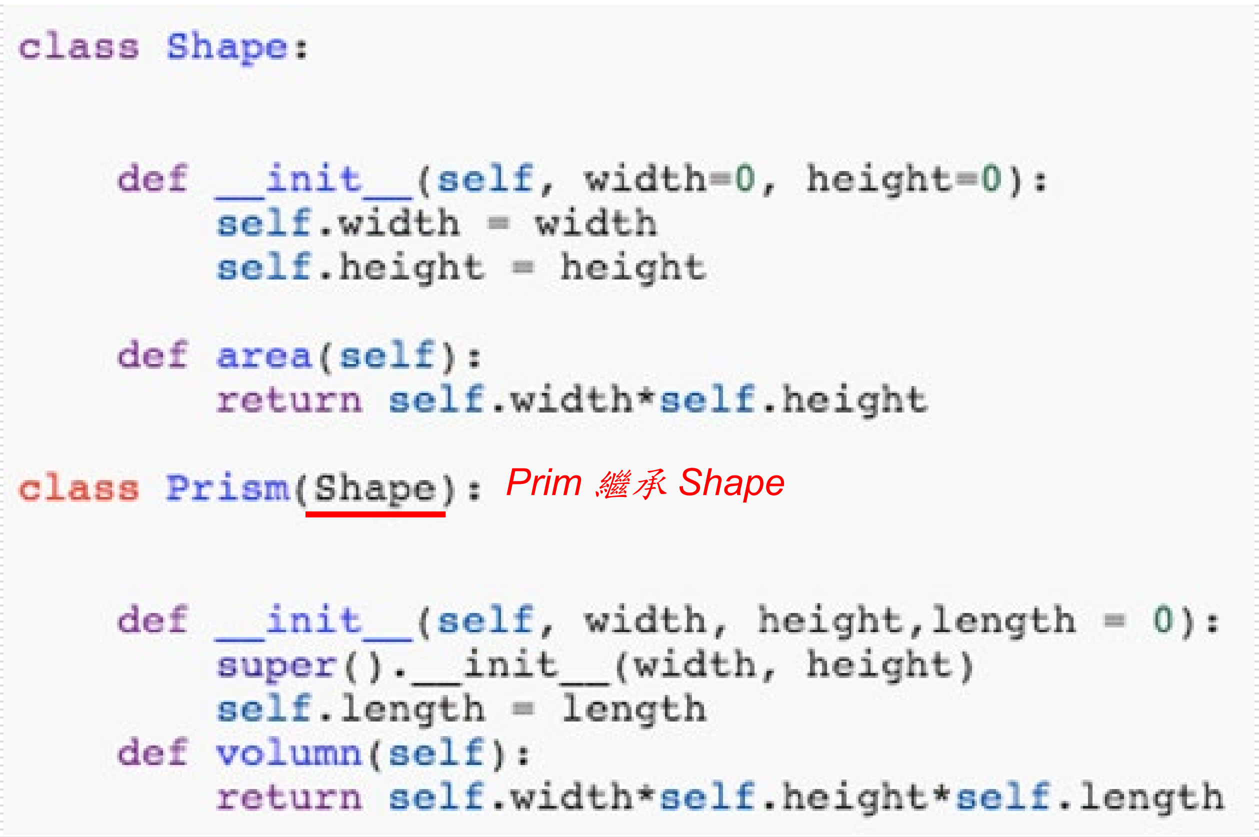
member function

```python
my_obj = Shape()
print(my_obj.width)
print(my_obj.height)
print("Area = ",my_obj.area())
print("============")
my_obj2 = Shape(10)
print(my_obj2.width)
print(my_obj2.height)
print("Area = ",my_obj2.area())
```

```
0
0
Area =  0
============
10
0
Area =  0
```

Class Inheritance (1/3)

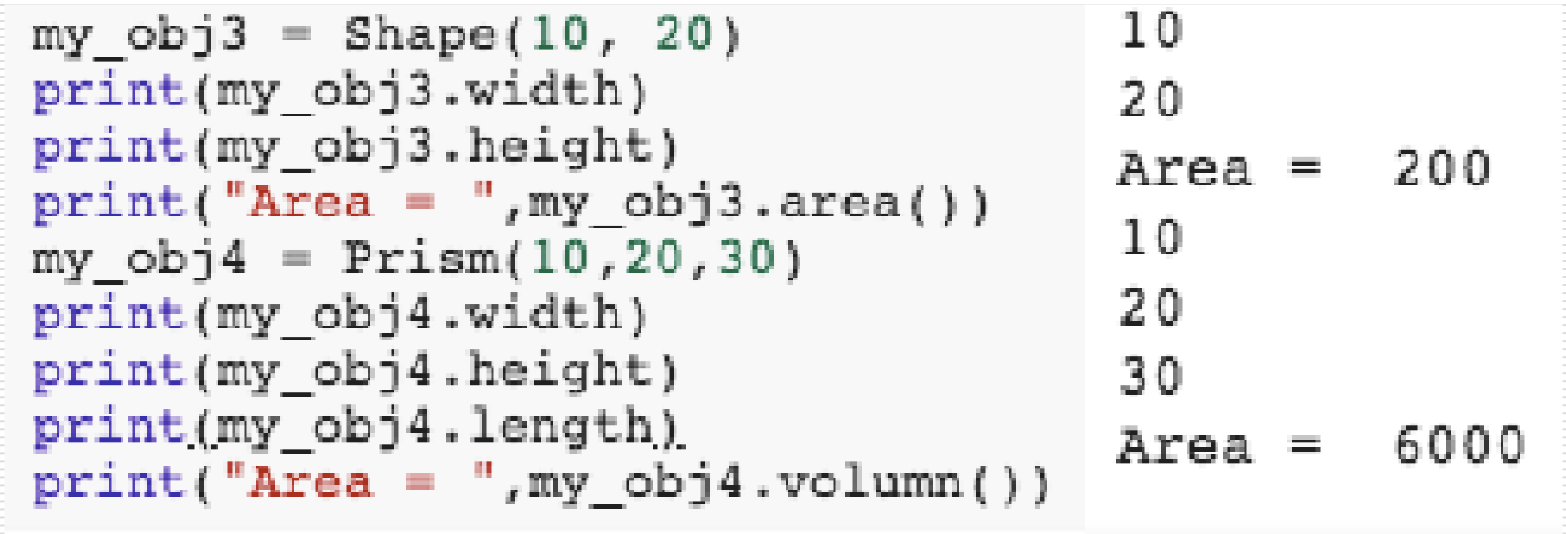
- Class class_name(父類別):
  - def __init__(self,parameter,…):
    - super().__init__(父類別的parameter)



```
class Shape:

    def __init__(self, width=0, height=0):
        self.width = width
        self.height = height

    def area(self):
        return self.width*self.height

class Prism(Shape):

    def __init__(self, width, height,length = 0):
        super().__init__(width, height)
        self.length = length
    def volumn(self):
        return self.width*self.height*self.length
```

Prim 繼承 Shape

Class Inheritance (2/3)



```python
my_obj3 = Shape(10, 20)
print(my_obj3.width)
print(my_obj3.height)
print("Area = ",my_obj3.area())
my_obj4 = Prism(10,20,30)
print(my_obj4.width)
print(my_obj4.height)
print(my_obj4.length)
print("Area = ",my_obj4.volumn())
```

```
10
20
Area =  200
10
20
30
Area =  6000
```

Class Inheritance (3/3)

```
import torch.nn as nn
import torch.nn.function as F
                     submodule

class Model(nn.Module):  # 繼承 nn.module
    def __init__(self):  # Define what modules you need
        super(Model, self).__init__()
        self.fc1 = nn.Linear(2,3)

    def forward(self, x):  # Define how to pass data
        out = self.fc1(x)
        out = F.relu(out)
        return out
```

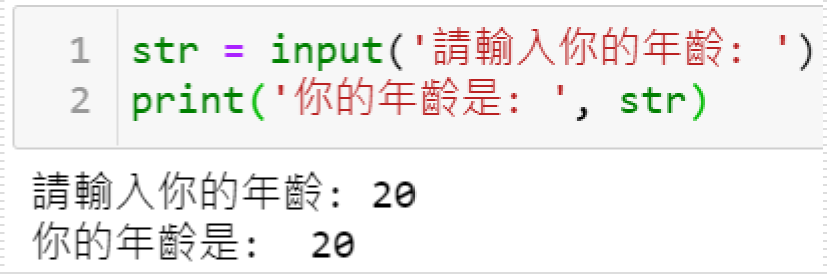
Time

- import time
- ticks = time.time() → 從1970年1月1日午夜到現在經過幾秒
- Ex :

```
ticks1 = time.time()
......
一些計算過程
......
ticks2 = time.time()
second = ticks2 - ticks1
```
→計算中間的指令需要幾秒

File I/O

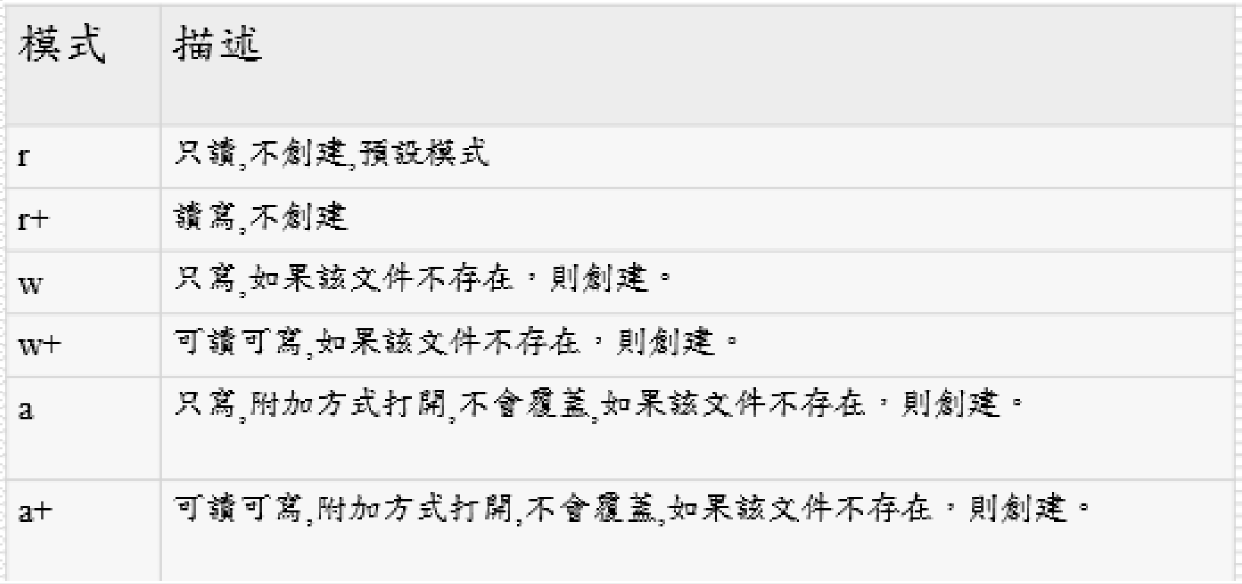
- input()



```
str = input('請輸入你的年齡: ')
print('你的年齡是: ', str)
```

```
請輸入你的年齡: 20
你的年齡是:  20
```

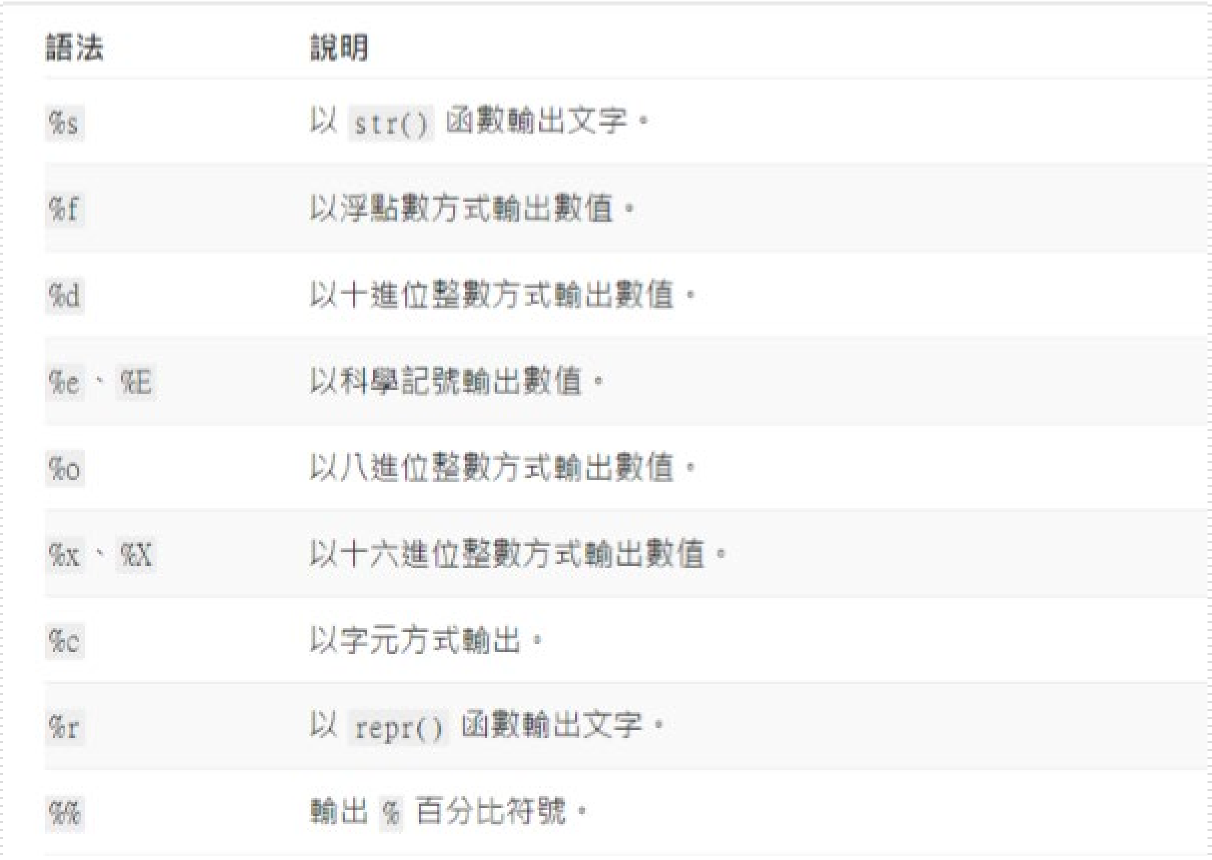
- file = open(filename, mode)



模式	描述
r	只讀,不創建,預設模式
r+	讀寫,不創建
w	只寫,如果該文件不存在，則創建。
w+	可讀可寫,如果該文件不存在，則創建。
a	只寫,附加方式打開,不會覆蓋,如果該文件不存在，則創建。
a+	可讀可寫,附加方式打開,不會覆蓋,如果該文件不存在，則創建。

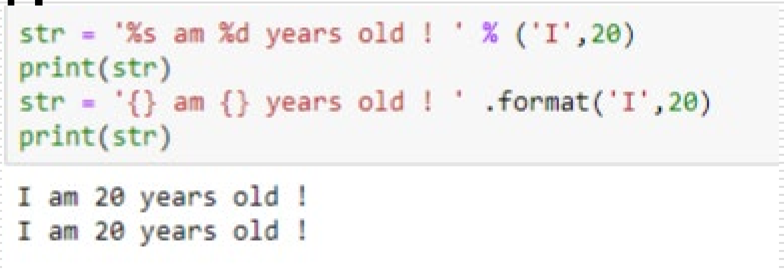
Print

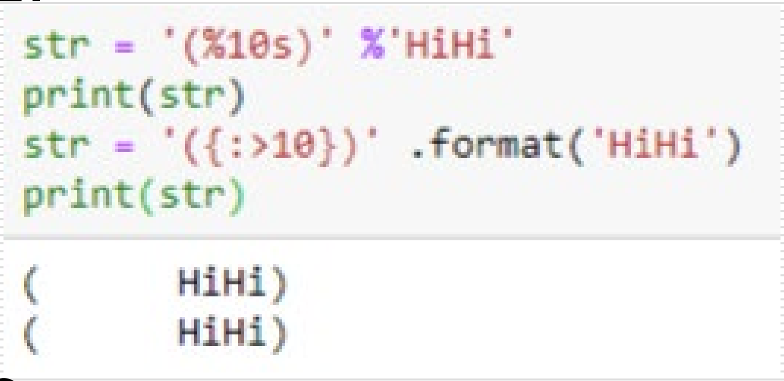
- % 開頭
- .format()

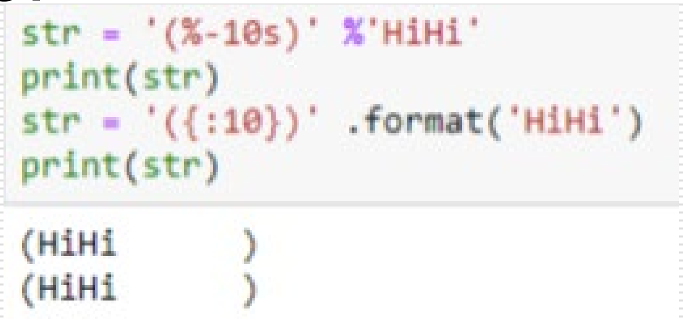


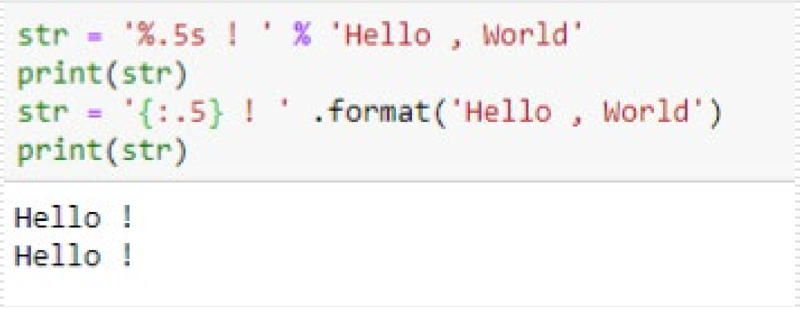
語法	說明
%s	以 str() 函數輸出文字。
%f	以浮點數方式輸出數值。
%d	以十進位整數方式輸出數值。
%e、%E	以科學記號輸出數值。
%o	以八進位整數方式輸出數值。
%x、%X	以十六進位整數方式輸出數值。
%c	以字元方式輸出。
%r	以 repr() 函數輸出文字。
%%	輸出 % 百分比符號。

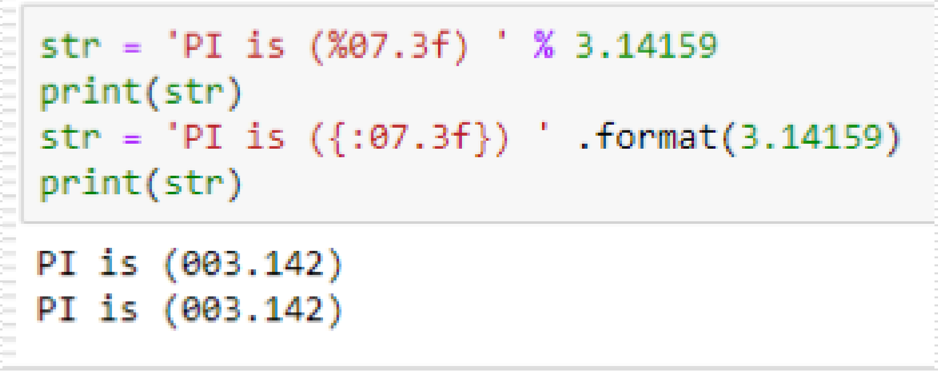
% vs. .format()

1.

```
str = '%s am %d years old ! ' % ('I',20)
print(str)
str = '{} am {} years old ! ' .format('I',20)
print(str)
```
```
I am 20 years old !
I am 20 years old !
```

2.

```
str = '(%10s)' %'HiHi'
print(str)
str = '({:>10})' .format('HiHi')
print(str)
```
```
(      HiHi)
(      HiHi)
```

3.

```
str = '(%-10s)' %'HiHi'
print(str)
str = '({:10})' .format('HiHi')
print(str)
```
```
(HiHi      )
(HiHi      )
```

4.

```
str = '%.5s ! ' % 'Hello , World'
print(str)
str = '{:.5} ! ' .format('Hello , World')
print(str)
```
```
Hello !
Hello !
```

5.

```
str = 'PI is (%07.3f) ' % 3.14159
print(str)
str = 'PI is ({:07.3f}) ' .format(3.14159)
print(str)
```
```
PI is (003.142)
PI is (003.142)
```

Parser (1/5)

- import argparse

```
ArgumentParser(prog=None, usage=None, description=None, epilog=None)
```

  - prog:
    - program的名字，default 是檔名。
  - usage:
    - 字串，主要是告知使用者說應該怎麼使用你寫的program
    - 保持None的話就會自動根據你設的參數產生相對應的說明字串。
  - description:
    - 字串，簡短說明程式資訊，會在所有參數說明的**前面**。
  - epilog:
    - 字串，補充程式資訊，會在所有參數說明的**後面**。

Parser (2/5)

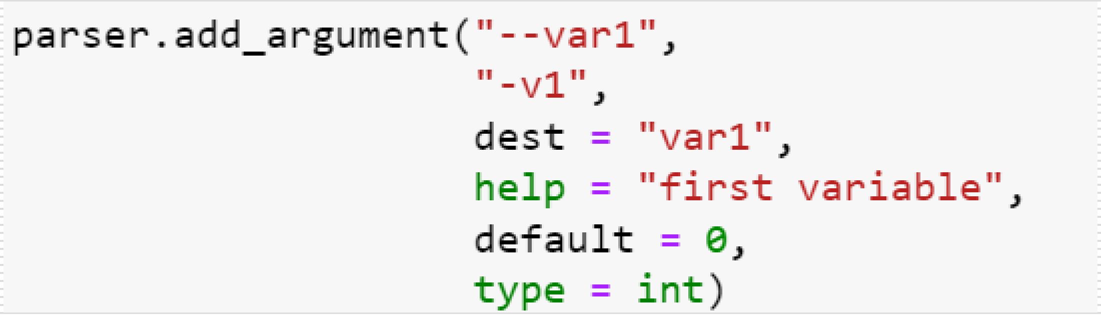
- Example

```python
from argparse import ArgumentParser
parser=ArgumentParser(prog="prog",usage="Tutorial",description="input var1",epilog="type: int")
parser.print_help()
```

```
usage: Tutorial  ───→ usage

input var1  ───→ description

optional arguments:
  -h, --help  show this help message and exit  ───→ 參數說明，自動產生

type: int  ───→ epilog
```

Parser (3/5)

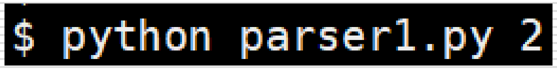
- Parser.add_argument()　增加參數
  - Name or flags
    - 參數的名稱，可以用縮寫，但要有全名。例如:--target, -t
  - dest
    - 當parse_args()解析完後的參數名稱。
  - default
    - 預設的參數值
  - type
    - 參數值的型態
  - help
    - 參數的說明
  - required
    - 參數值是否必須



```
parser.add_argument("--var1",
                    "-v1",
                    dest = "var1",
                    help = "first variable",
                    default = 0,
                    type = int)
```

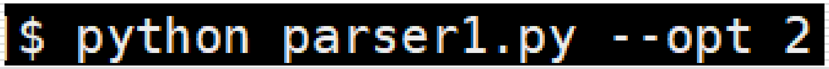
Parser (4/5)

- Positional argument:
  - 依照輸入順序放進你宣告的引數變數中
  - 沒有前綴”-”
  - parser.add_argument(“pos”, help = “pos_arg”)



```
$ python parser1.py 2
```

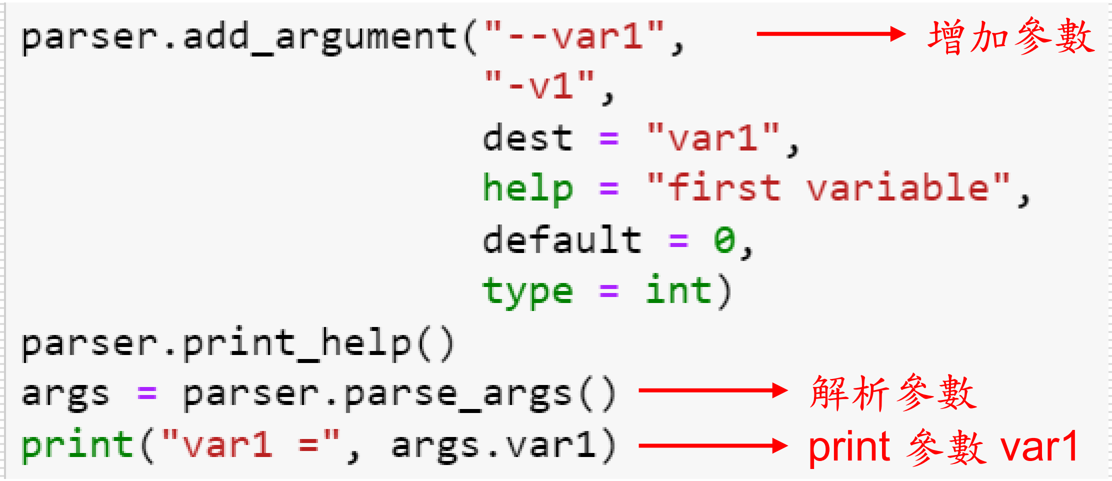
- Optional argument:
  - 有前綴”-”
  - parser.add_argument(“-o”, “--opt”, help = “opt_arg”)



```
$ python parser1.py --opt 2
```

- args = parser.parse_args() ➡ 解析添加的參數
- a = args.opt ➡ 將參數提出來使用

Parser (5/5)

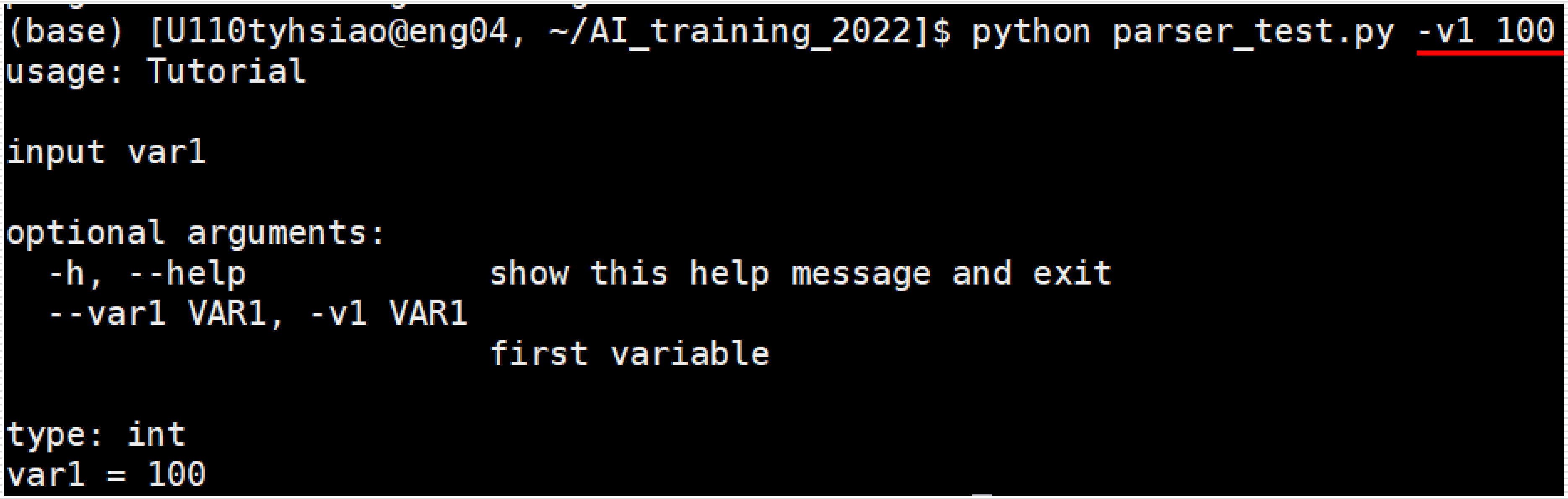


```
parser.add_argument("--var1",
                    "-v1",
                    dest = "var1",
                    help = "first variable",
                    default = 0,
                    type = int)
parser.print_help()
args = parser.parse_args()
print("var1 =", args.var1)
```

→ 增加參數

→ 解析參數

→ print 參數 var1



```
(base) [U110tyhsiao@eng04, ~/AI_training_2022]$ python parser_test.py -v1 100
usage: Tutorial

input var1

optional arguments:
  -h, --help            show this help message and exit
  --var1 VAR1, -v1 VAR1
                        first variable

type: int
var1 = 100
```

Introduction to Numpy

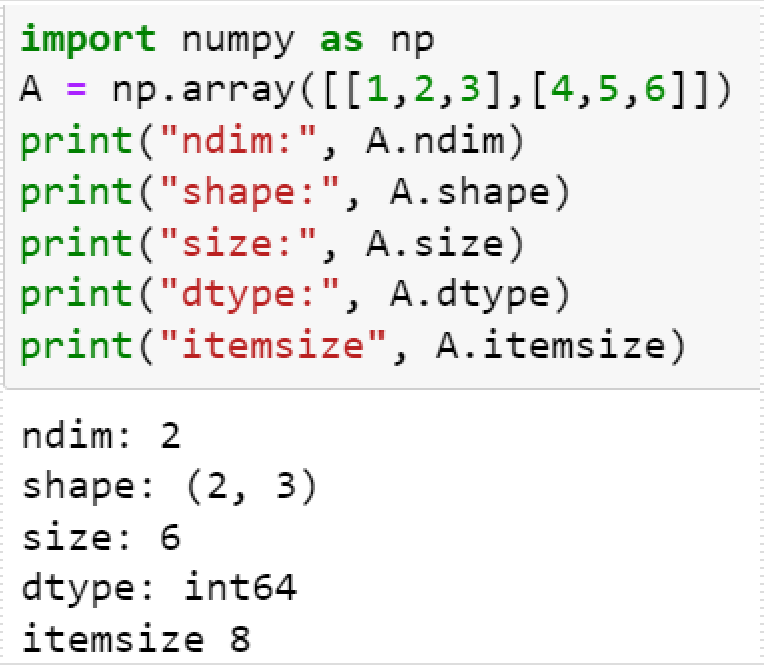
Numpy Array vs. List

- Numpy Array:
  - homogeneous
  - 高效能的多維陣列(multi-dimensional array)數學函式庫
    - 需要import
  - 平行處理、科學運算較快

- List:
  - homogeneous or heterogeneous
  - python內建資料型別

Ndarray Attribute

- Import numpy as np
- A.ndim ➡ array A 的維度
- A.shape ➡ 每個維度的大小
- A.size ➡ array A 的總元素量
- A.dtype ➡ 元素的型態
- A.itemsize ➡ 每個元素的大小是多少個byte

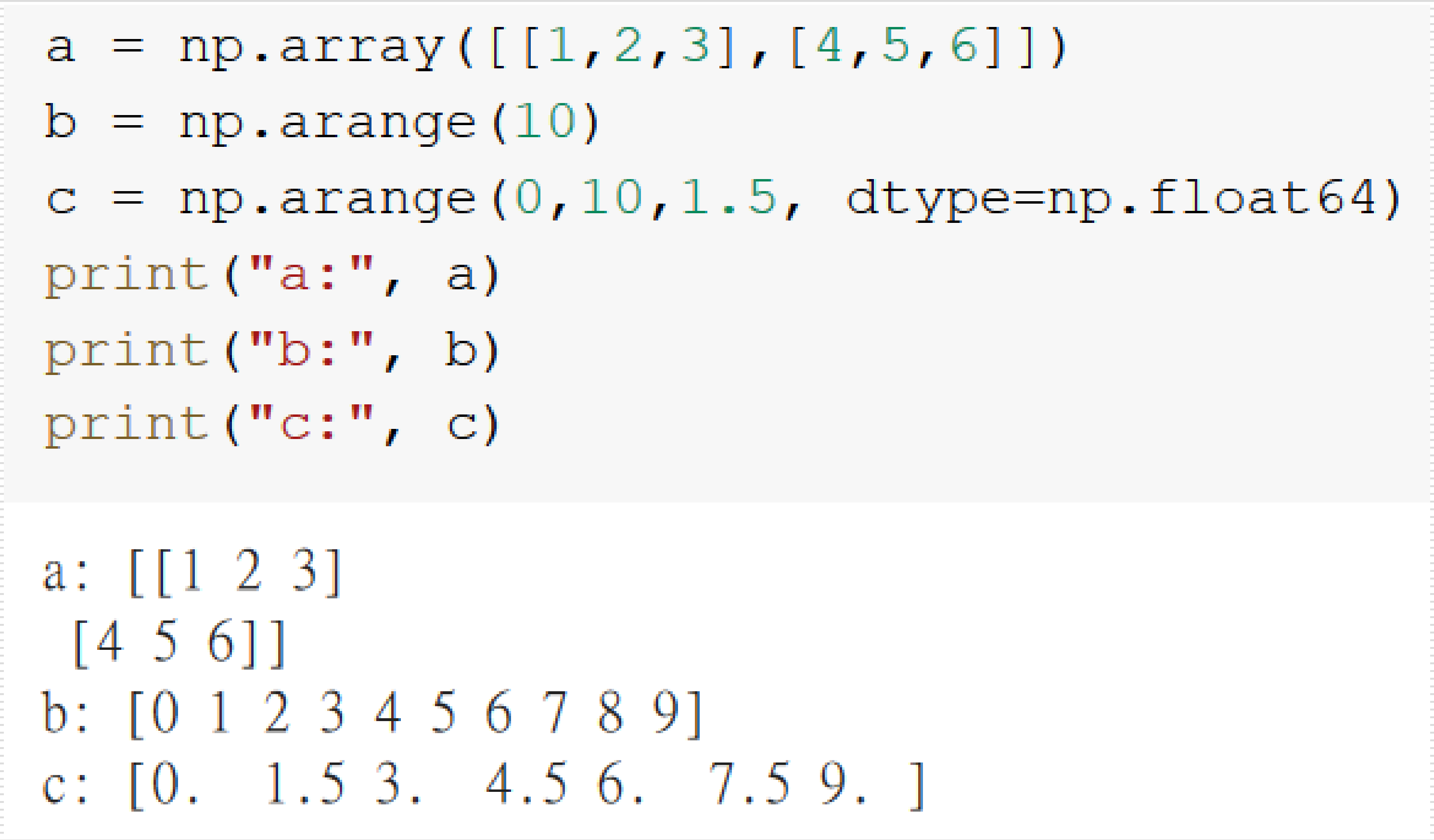


```python
import numpy as np
A = np.array([[1,2,3],[4,5,6]])
print("ndim:", A.ndim)
print("shape:", A.shape)
print("size:", A.size)
print("dtype:", A.dtype)
print("itemsize", A.itemsize)
```

```
ndim: 2
shape: (2, 3)
size: 6
dtype: int64
itemsize 8
```

Generate an Ndarray (1/2)

- np.array
- np.arange(起始值,結束值,間隔)

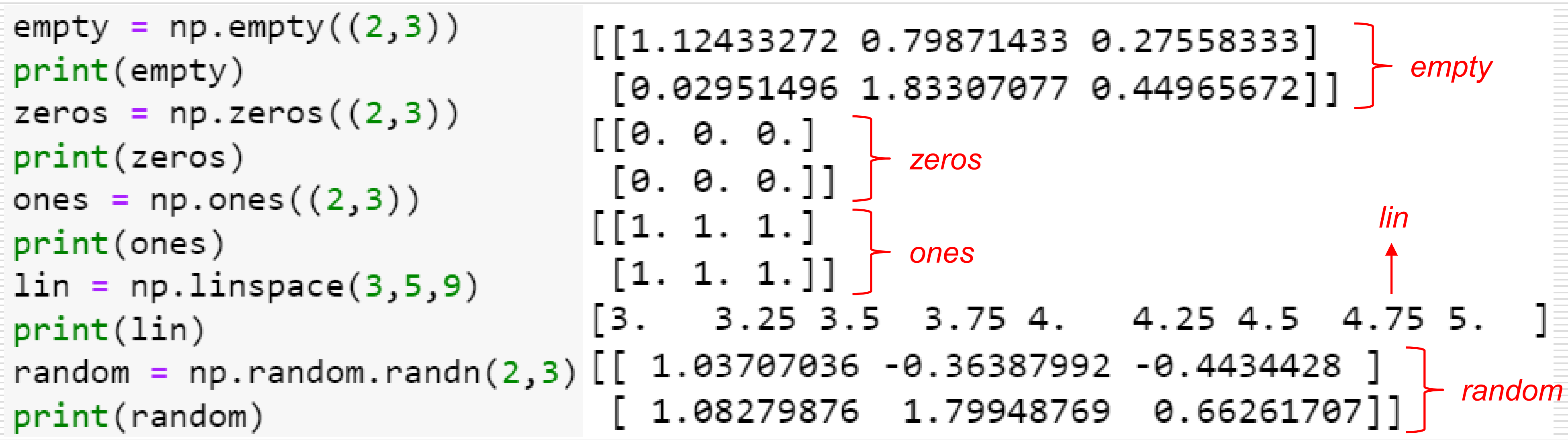


```python
a = np.array([[1,2,3],[4,5,6]])
b = np.arange(10)
c = np.arange(0,10,1.5, dtype=np.float64)
print("a:", a)
print("b:", b)
print("c:", c)
```

```
a: [[1 2 3]
 [4 5 6]]
b: [0 1 2 3 4 5 6 7 8 9]
c: [0.  1.5 3.  4.5 6.  7.5 9. ]
```

Generate an Ndarray (2/2)

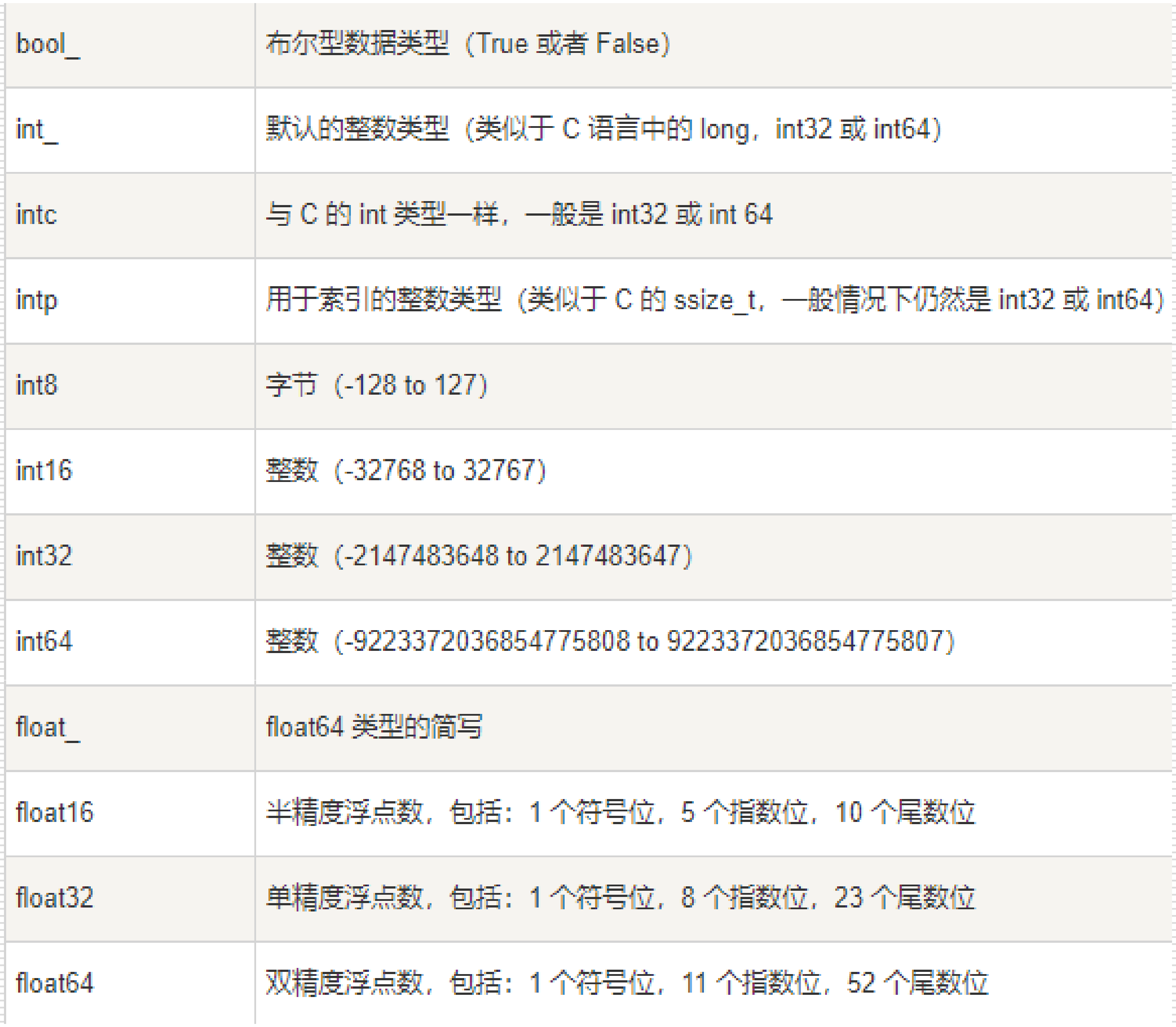
- np.empty ((a,b)) ➡沒有初始值的 a*b array
- np.zeros ((a,b)) ➡所有元素都為 0 的 a*b array
- np.ones ((a,b)) ➡所有元素都為 1 的 a*b array
- np.linspace(起始值,結束值,中間要產生多少個元素)
- np.random.randn(2,3) ➡ 常態分佈的 2*3 array



```
empty = np.empty((2,3))
print(empty)
zeros = np.zeros((2,3))
print(zeros)
ones = np.ones((2,3))
print(ones)
lin = np.linspace(3,5,9)
print(lin)
random = np.random.randn(2,3)
print(random)
```

```
[[1.12433272 0.79871433 0.27558333]
 [0.02951496 1.83307077 0.44965672]]
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
[3.   3.25 3.5  3.75 4.   4.25 4.5  4.75 5.  ]
[[ 1.03707036 -0.36387992 -0.4434428 ]
 [ 1.08279876  1.79948769  0.66261707]]
```

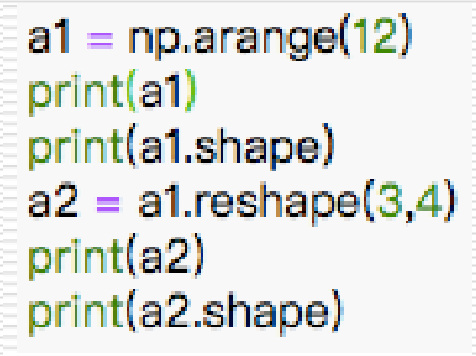
Common dtype



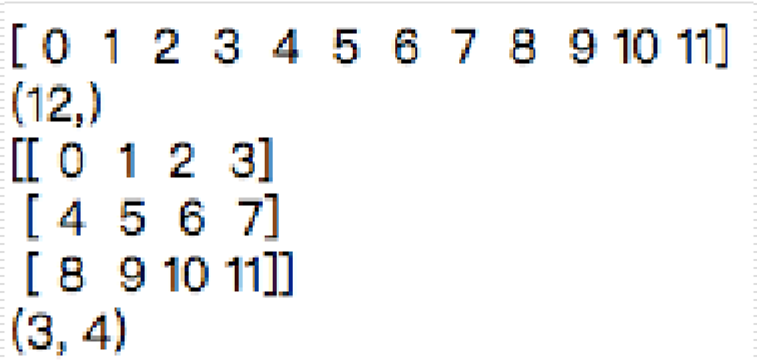
bool_	布尔型数据类型（True 或者 False）
int_	默认的整数类型（类似于 C 语言中的 long，int32 或 int64）
intc	与 C 的 int 类型一样，一般是 int32 或 int 64
intp	用于索引的整数类型（类似于 C 的 ssize_t，一般情况下仍然是 int32 或 int64）
int8	字节（-128 to 127）
int16	整数（-32768 to 32767）
int32	整数（-2147483648 to 2147483647）
int64	整数（-9223372036854775808 to 9223372036854775807）
float_	float64 类型的简写
float16	半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
float32	单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
float64	双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位

Array vs. List

- np.array(list_name) ➡ list 轉numpy array
- list(array_name) vs. array_name.tolist()
- reshape ➡ 改變array的形狀



```
a1 = np.arange(12)
print(a1)
print(a1.shape)
a2 = a1.reshape(3,4)
print(a2)
print(a2.shape)
```

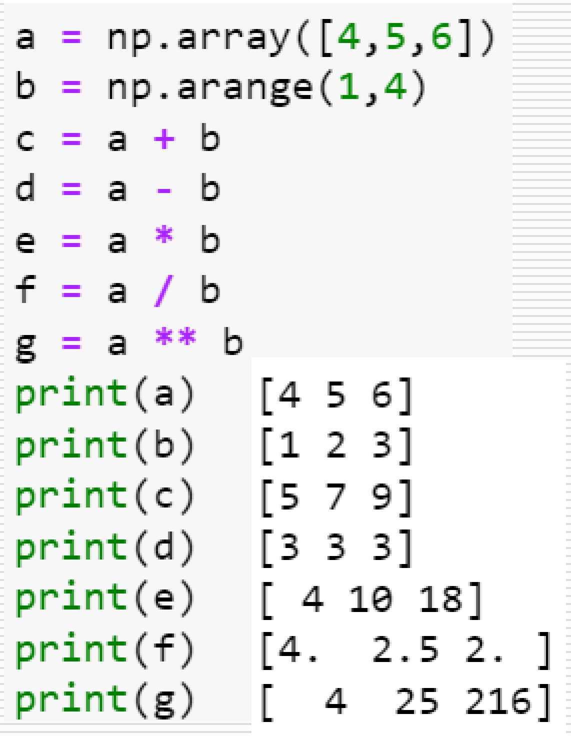


```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
(12,)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
(3, 4)
```

- dtype
  – A = np.array([1,2,3],dtype=np.float64)
- astype ➡ 轉換資料型別
  – B=A.astype(np.float64)

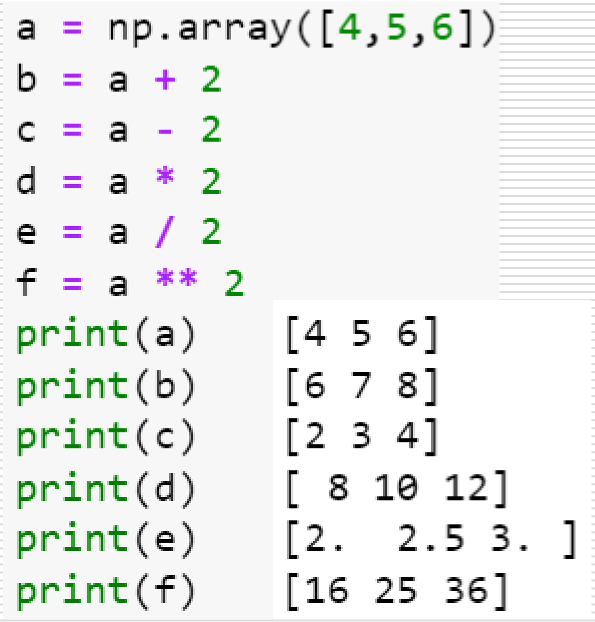
Basic Operations

- +, -, *, /, **
  - 一定要同維度的兩個array ➡ 同位置的元素做運算
  - array跟特定純量做運算 ➡ 純量跟每個元素做運算



```
a = np.array([4,5,6])
b = np.arange(1,4)
c = a + b
d = a - b
e = a * b
f = a / b
g = a ** b
print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
```

```
[4 5 6]
[1 2 3]
[5 7 9]
[3 3 3]
[ 4 10 18]
[4.  2.5 2. ]
[  4  25 216]
```



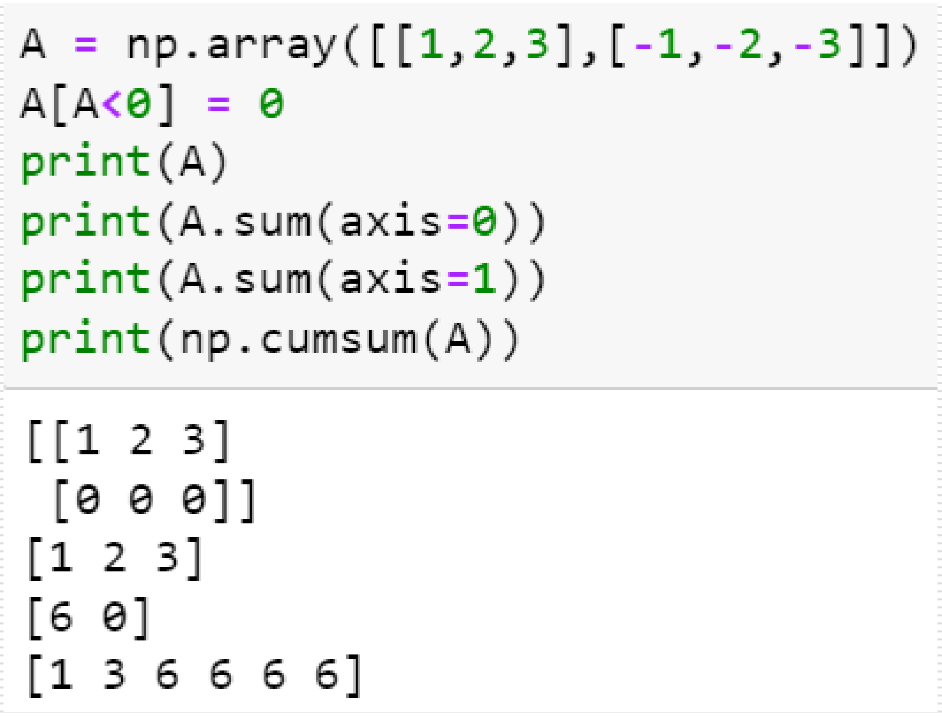
```
a = np.array([4,5,6])
b = a + 2
c = a - 2
d = a * 2
e = a / 2
f = a ** 2
print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
```

```
[4 5 6]
[6 7 8]
[2 3 4]
[ 8 10 12]
[2.  2.5 3. ]
[16 25 36]
```

- 矩陣相乘：A.dot(B)

Common Usage

- A[A<=0] = 0
- np.sum(A)
  - A.sum(axis=0) $\downarrow$
  - A.sum(axis=1) $\longrightarrow$
- np.cumsum()
  - 前面元素的累加
- np.max()
- np.min()
- np.mean()
- B=np.exp(A) ➡ e的指數次方
- B=np.sqrt(A)



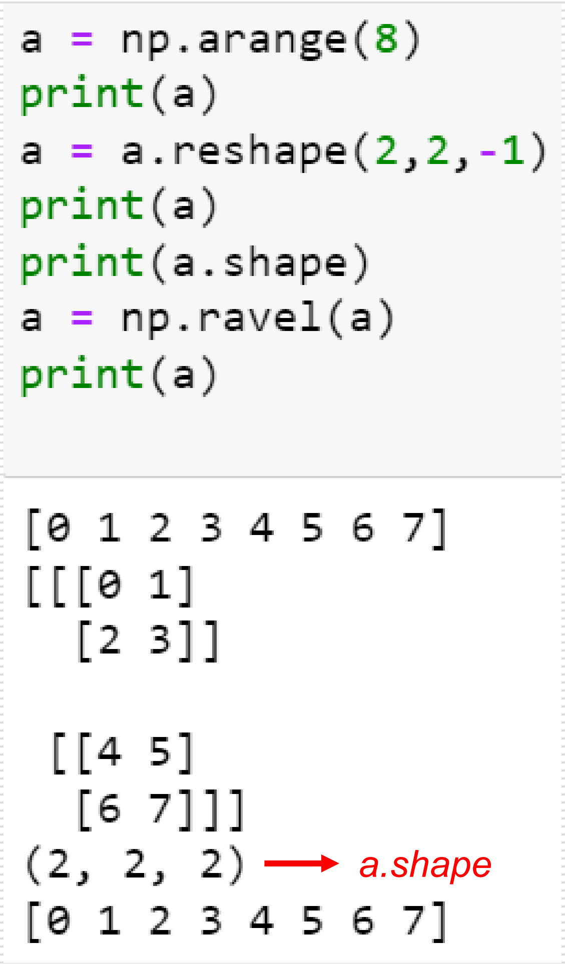
```python
A = np.array([[1,2,3],[-1,-2,-3]])
A[A<0] = 0
print(A)
print(A.sum(axis=0))
print(A.sum(axis=1))
print(np.cumsum(A))
```

```
[[1 2 3]
 [0 0 0]]
[1 2 3]
[6 0]
[1 3 6 6 6 6]
```

Shape of Array (1/2)

- np.reshape(a,b,-1) ➡ -1表示自動計算維度
- np.ravel() ➡ flattened
- A.T ➡ A的轉置矩陣
- np.vstack((A,B)) ➡ 縱向疊加
- np.hstack((A,B)) ➡ 橫向疊加
- np.vsplit(A,num) ➡ 縱向拆分
- np.hsplit(A,num) ➡ 橫向拆分
- np.concatenate((a,b), axis=dim) ➡ 沿dim維度疊加

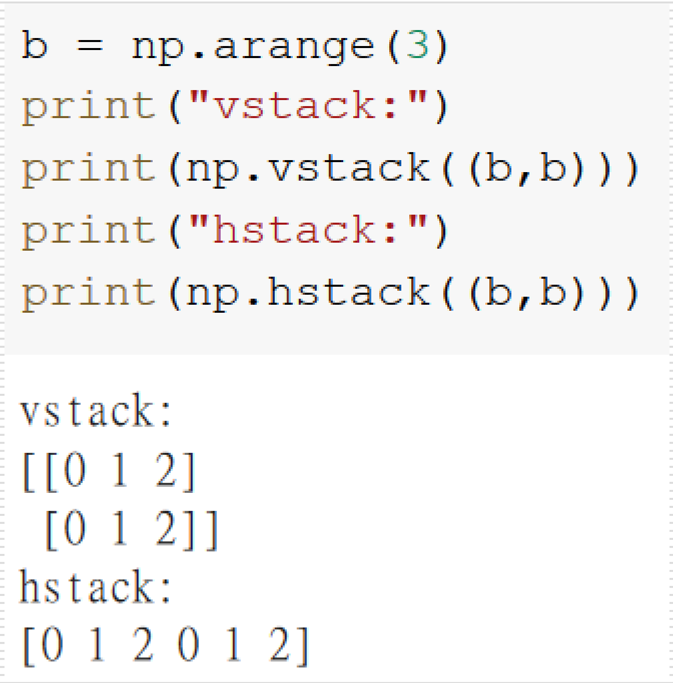
Shape of Array (2/2)



```
a = np.arange(8)
print(a)
a = a.reshape(2,2,-1)
print(a)
print(a.shape)
a = np.ravel(a)
print(a)
```

```
[0 1 2 3 4 5 6 7]
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
(2, 2, 2)
[0 1 2 3 4 5 6 7]
```

$\longrightarrow$ *a.shape*

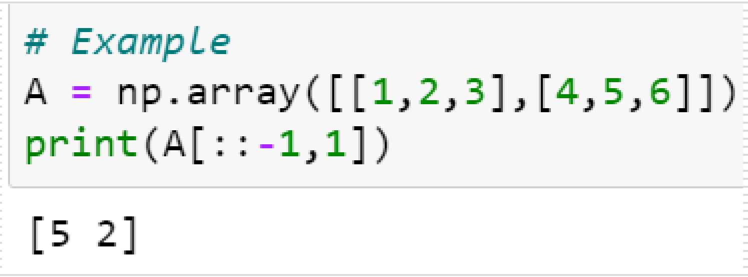


```
b = np.arange(3)
print("vstack:")
print(np.vstack((b,b)))
print("hstack:")
print(np.hstack((b,b)))
```

```
vstack:
[[0 1 2]
 [0 1 2]]
hstack:
[0 1 2 0 1 2]
```

Indexing & Slicing (1/2)

- A[第0個維度的index/slicing,第1個維度...]
  - A[a:b]
    - a:起始索引值, b:結束索引值+1
    - a的位置不填: default 0，b的位置不填: default size+1
  - A[a:b:c]
    - a: 起始索引
    - b: 結束索引+1
    - c: 選取資料間隔,以索引值可以被此值整除的元素,預設為1
  - A[:]
    - 選取所有此維度的元素
  - A[::-1]
    - 此維度的元素順序顛倒



```
# Example
A = np.array([[1,2,3],[4,5,6]])
print(A[::-1,1])
```

```
[5 2]
```

- Ex. A[::-1,1] ➡ A的第1個column且順序顛倒

Introduction to Pytorch

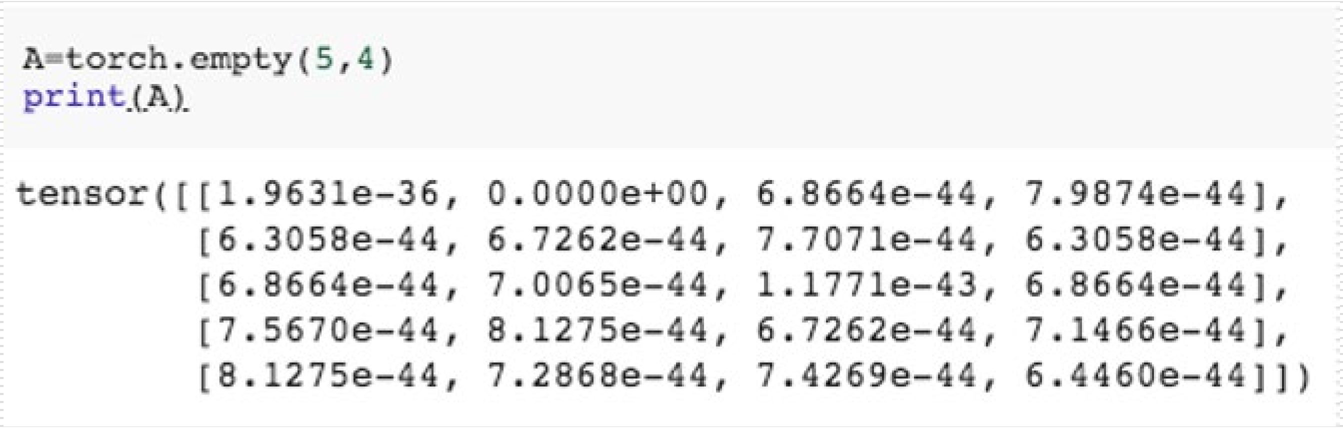
Pytorch & Tensor

- Pytorch 是 Facebook 於 2017 年所開源的深度學習框架，因其語法簡潔、直觀的特性深受歡迎，已成為目前深度學習熱門框架之一。

- Tensor
  - 多維度的矩陣
  - Pytorch 的基本元素
  - 用法與 numpy 類似
  - Pytorch 可以在 GPU 上執行，numpy 只能在 CPU 上執行

Generate a Tensor (1/2)

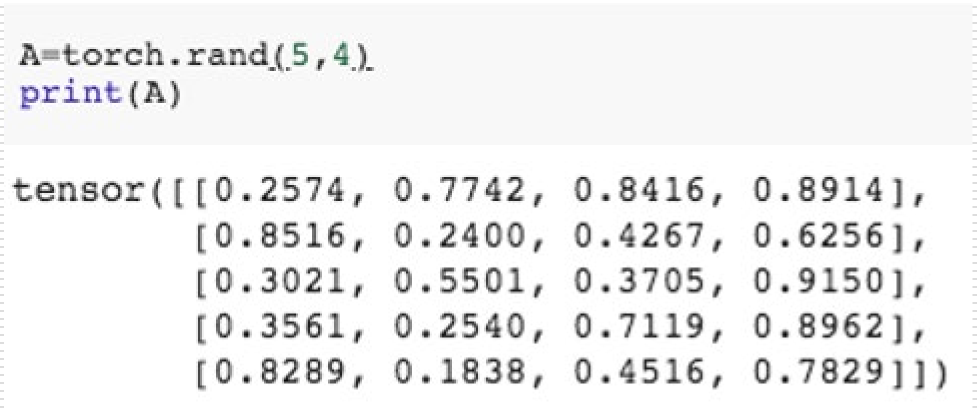
- import torch
- A = torch.empty(a,b)



```
A=torch.empty(5,4)
print(A)
```

```
tensor([[1.9631e-36, 0.0000e+00, 6.8664e-44, 7.9874e-44],
        [6.3058e-44, 6.7262e-44, 7.7071e-44, 6.3058e-44],
        [6.8664e-44, 7.0065e-44, 1.1771e-43, 6.8664e-44],
        [7.5670e-44, 8.1275e-44, 6.7262e-44, 7.1466e-44],
        [8.1275e-44, 7.2868e-44, 7.4269e-44, 6.4460e-44]])
```

- A = torch.rand(a,b)
  – 均勻分布
- A = torch.randn(a,b)
  – 常態分布

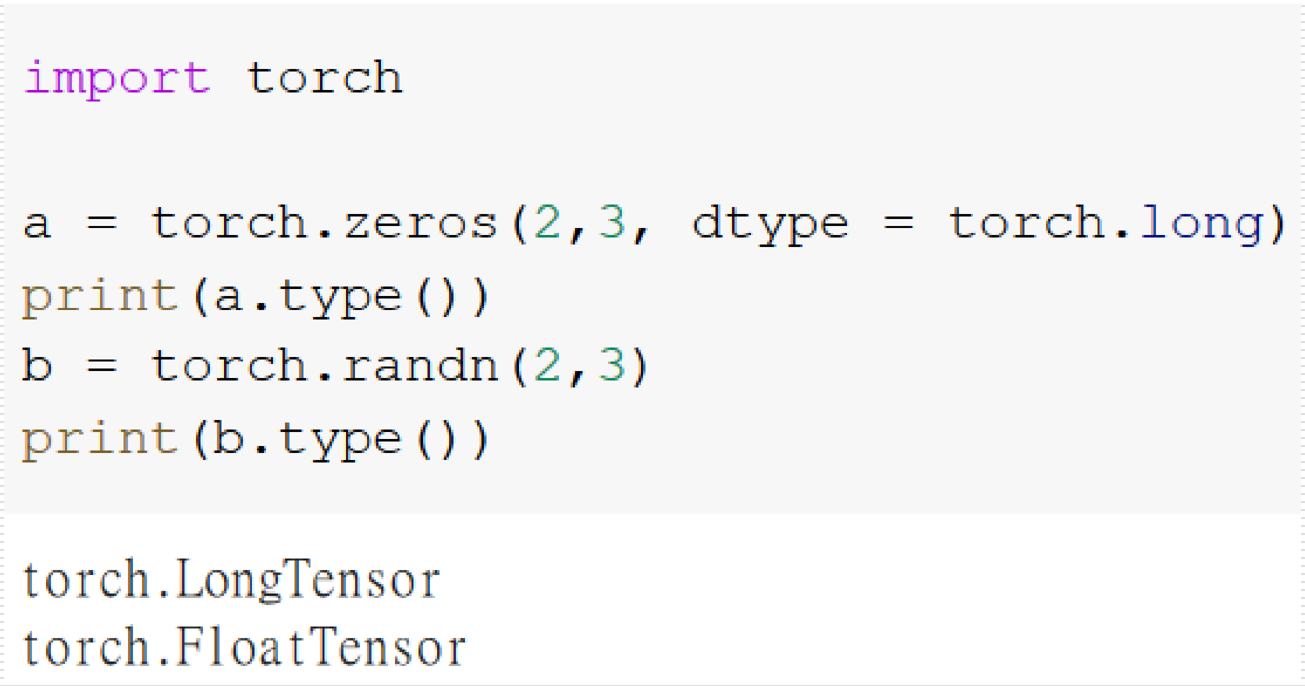


```
A=torch.rand(5,4)
print(A)
```

```
tensor([[0.2574, 0.7742, 0.8416, 0.8914],
        [0.8516, 0.2400, 0.4267, 0.6256],
        [0.3021, 0.5501, 0.3705, 0.9150],
        [0.3561, 0.2540, 0.7119, 0.8962],
        [0.8289, 0.1838, 0.4516, 0.7829]])
```

Generate a Tensor (2/2)

- A = torch.zeros(a,b, dtype=torch.long)
- A = torch.ones(a,b)
- A = torch.tensor([2.2,3.3])
- A.type()



```python
import torch

a = torch.zeros(2,3, dtype = torch.long)
print(a.type())
b = torch.randn(2,3)
print(b.type())
```

```
torch.LongTensor
torch.FloatTensor
```

Torch.dtype

Data type	dtype	Tensor types
32-bit floating point	torch.float32 or torch.float	torch.*.FloatTensor
64-bit floating point	torch.float64 or torch.double	torch.*.DoubleTensor
16-bit floating point	torch.float16 or torch.half	torch.*.HalfTensor
8-bit integer (unsigned)	torch.uint8	torch.*.ByteTensor
8-bit integer (signed)	torch.int8	torch.*.CharTensor
16-bit integer (signed)	torch.int16 or torch.short	torch.*.ShortTensor
32-bit integer (signed)	torch.int32 or torch.int	torch.*.IntTensor
64-bit integer (signed)	torch.int64 or torch.long	torch.*.LongTensor

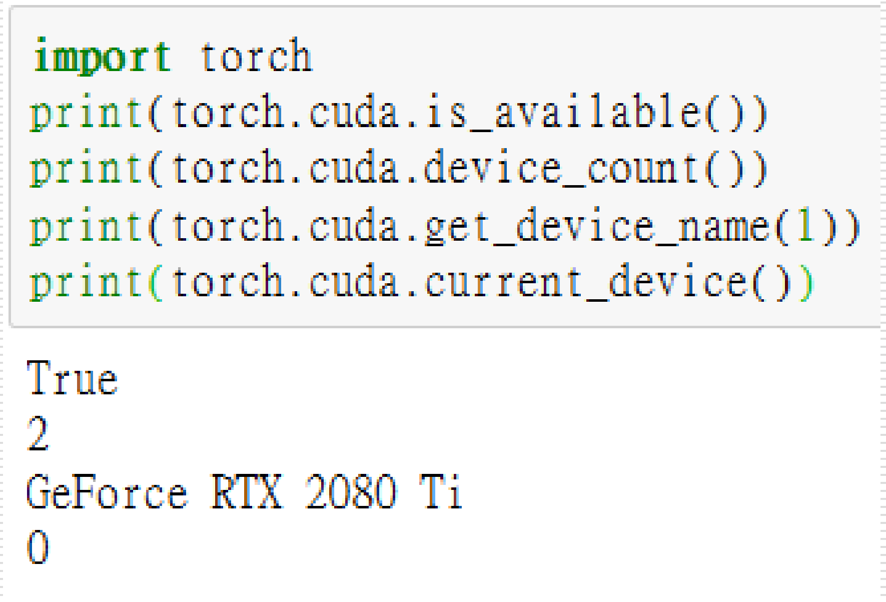
CUDA

- **CUDA**（**C**ompute **U**nified **D**evice **A**rchitecture）是由 NVIDIA 所推出的一種整合技術，是該公司對於 GPGPU 的正式名稱。

- torch.cuda
  - 實現了與CPU張量相同的功能，但使用GPU進行計算

Torch.cuda

- torch.cuda.is_available
- torch.cuda.device_count()
- torch.cuda.get_device_name(0)
- torch.cuda.set_device(device)
- torch.cuda.current_device()

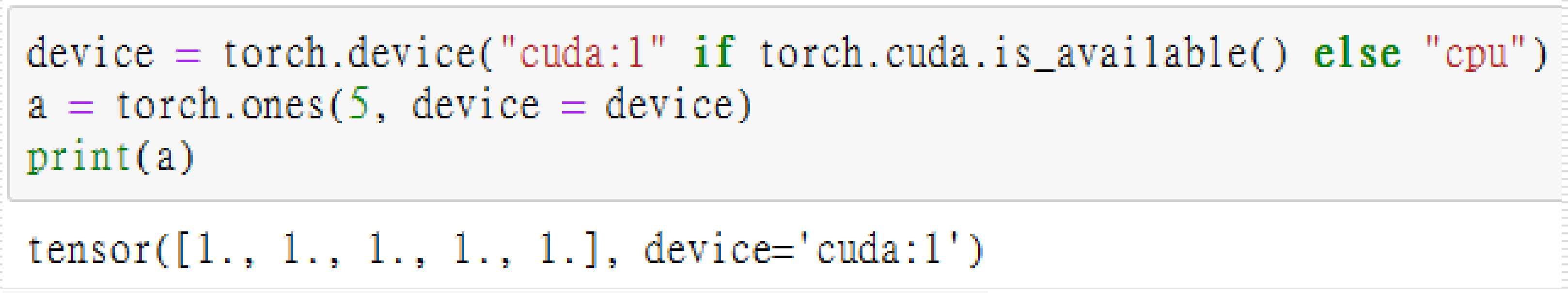


```
import torch
print(torch.cuda.is_available())
print(torch.cuda.device_count())
print(torch.cuda.get_device_name(1))
print(torch.cuda.current_device())
```

```
True
2
GeForce RTX 2080 Ti
0
```

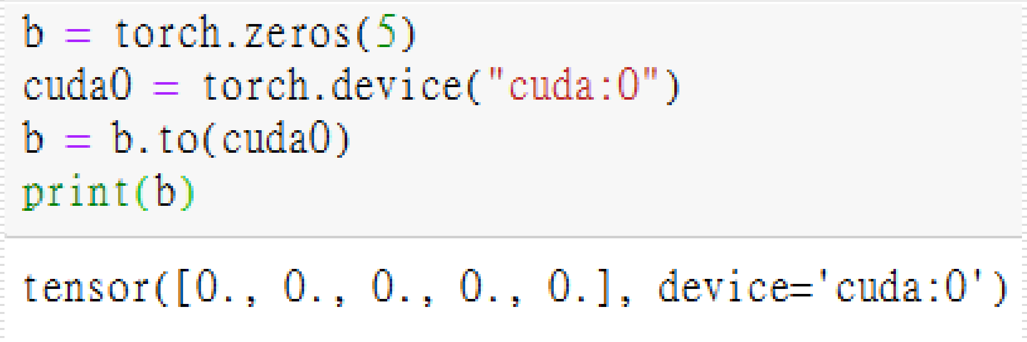
Torch.device

- torch.device(‘cpu’)
- torch.device(‘cuda:0’)
- 將 tensor 搬到 CUDA 上:
  - tensor.to(device)
  - tensor.cuda()



```
device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")
a = torch.ones(5, device = device)
print(a)
```

```
tensor([1., 1., 1., 1., 1.], device='cuda:1')
```



```
b = torch.zeros(5)
cuda0 = torch.device("cuda:0")
b = b.to(cuda0)
print(b)
```

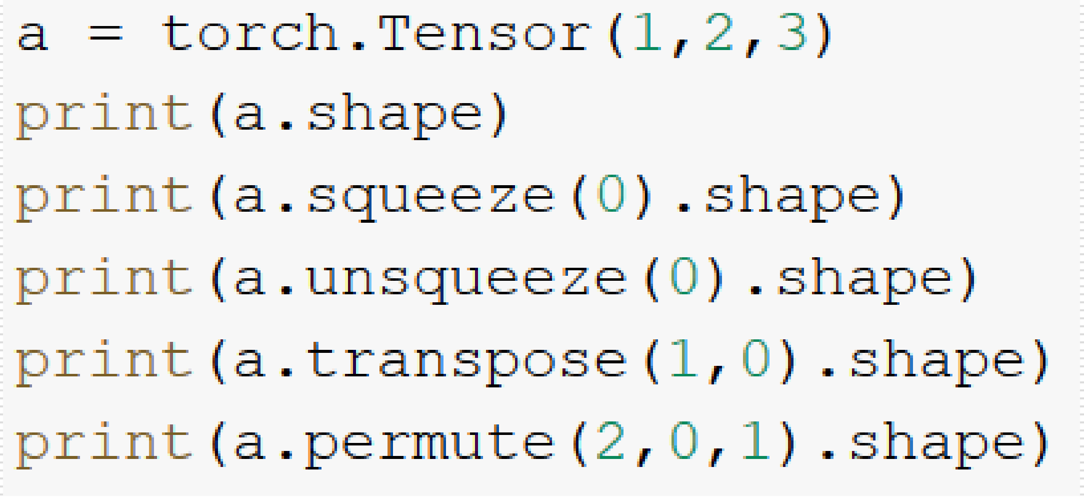
```
tensor([0., 0., 0., 0., 0.], device='cuda:0')
```

Tensor to Numpy & Numpy to Tensor

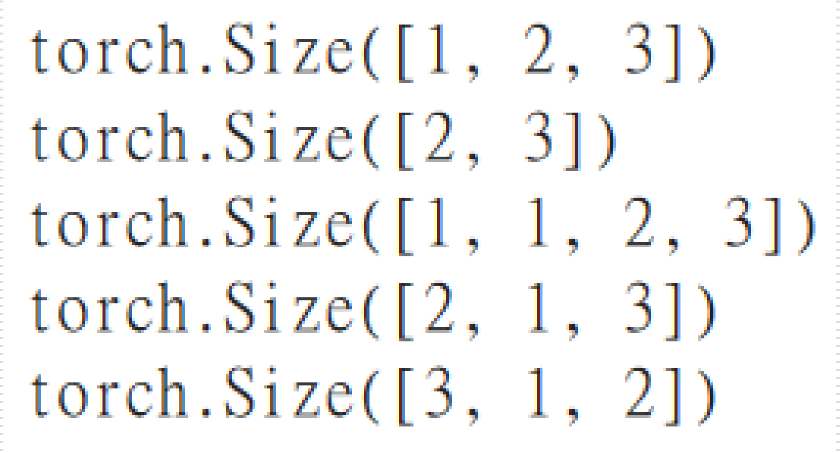
- Tensor to Numpy:
  - A = B.numpy()
- Numpy to Tensor:
  - B = torch.tensor(A)
  - B = torch.from_numpy(A)
- Device 轉換
  - cpu(), cuda()
- 類型轉換
  - double(), float()

Shape Manipulation

- X.squeeze(d) ➡ 壓縮第d維
- X.unsqueeze(d) ➡ 將第d維的維度設成1(多出一個維度)
- X.transpose(d1,d2) ➡ 兩個維度交換
- X.permute(d1,d2,d3,…) ➡ 多個維度交換



```
a = torch.Tensor(1,2,3)
print(a.shape)
print(a.squeeze(0).shape)
print(a.unsqueeze(0).shape)
print(a.transpose(1,0).shape)
print(a.permute(2,0,1).shape)
```

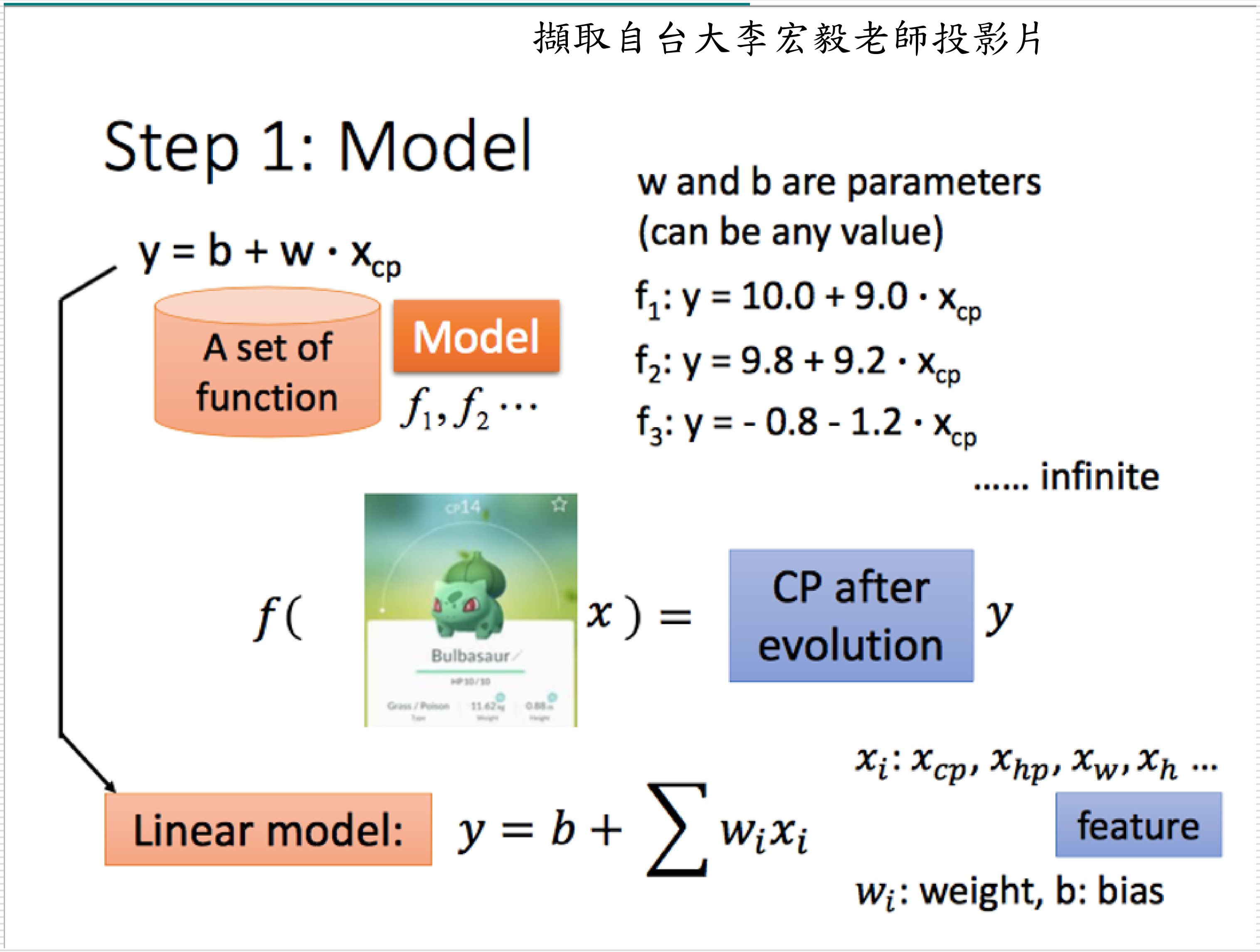


```
torch.Size([1, 2, 3])
torch.Size([2, 3])
torch.Size([1, 1, 2, 3])
torch.Size([2, 1, 3])
torch.Size([3, 1, 2])
```

Common Usage

- A.size()
- torch.add(a, b), torch.mul(a,b) ….
- torch.mm(a, b)
- torch.abs(a)
- A.view (-1,a)
- torch.cat((A,B), dim) ➡ 相當於np.concatenate
- A.sum()
- A.data (回傳相同的tensor，但requires_grad = False)

Linear Regression (1/7)



擷取自台大李宏毅老師投影片

Step 1: Model

$y = b + w \cdot x_{cp}$

A set of function

Model

$f_1, f_2 \cdots$

w and b are parameters
(can be any value)

f_1: $y = 10.0 + 9.0 \cdot x_{cp}$

f_2: $y = 9.8 + 9.2 \cdot x_{cp}$

f_3: $y = -0.8 - 1.2 \cdot x_{cp}$

…… infinite

$f($ $x) =$ CP after evolution y

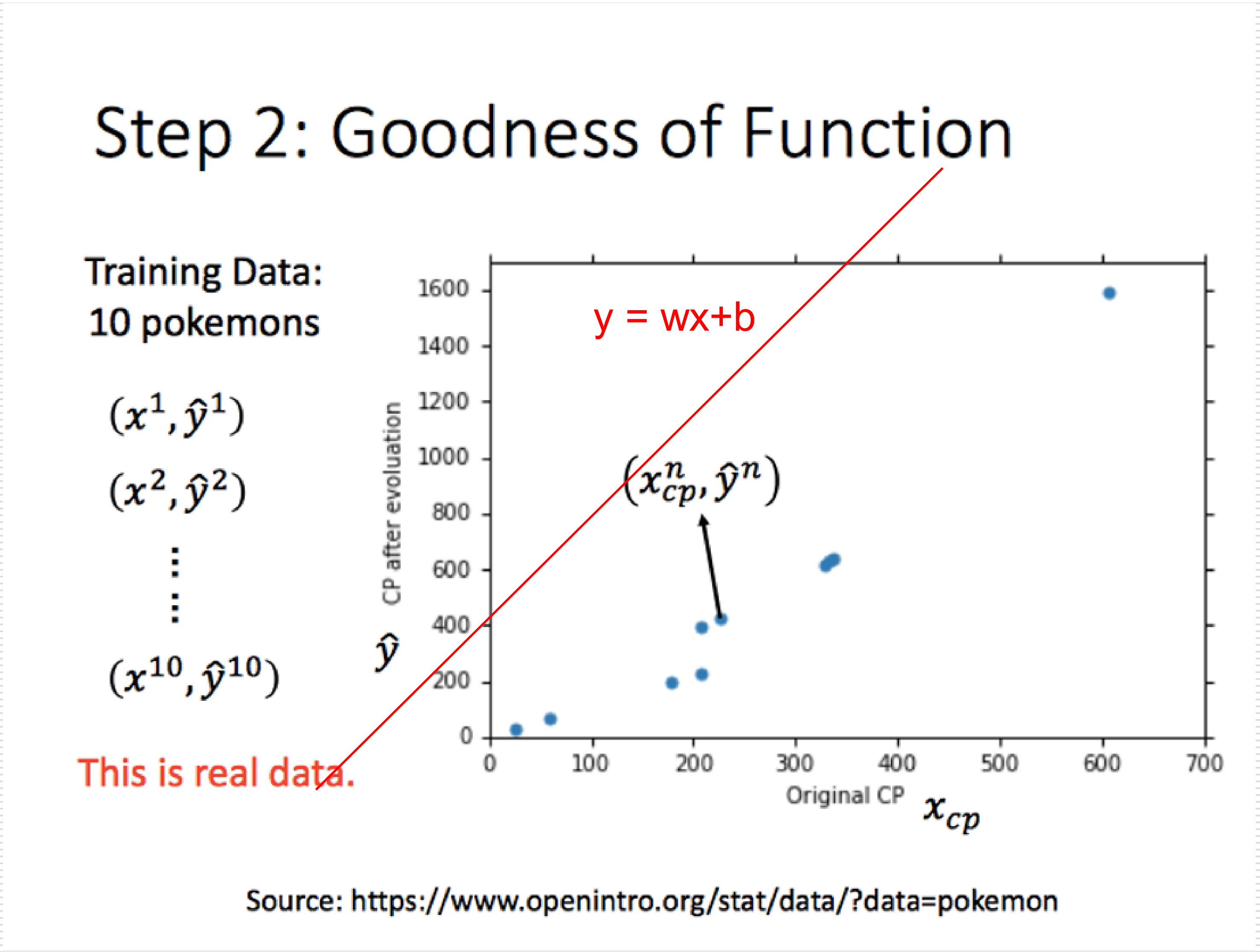
Linear model: $y = b + \sum w_i x_i$

x_i: x_{cp}, x_{hp}, x_w, x_h …

feature

w_i: weight, b: bias

Linear Regression (2/7)



Step 2: Goodness of Function

Training Data:
10 pokemons

$(x^1, \hat{y}^1)$

$(x^2, \hat{y}^2)$

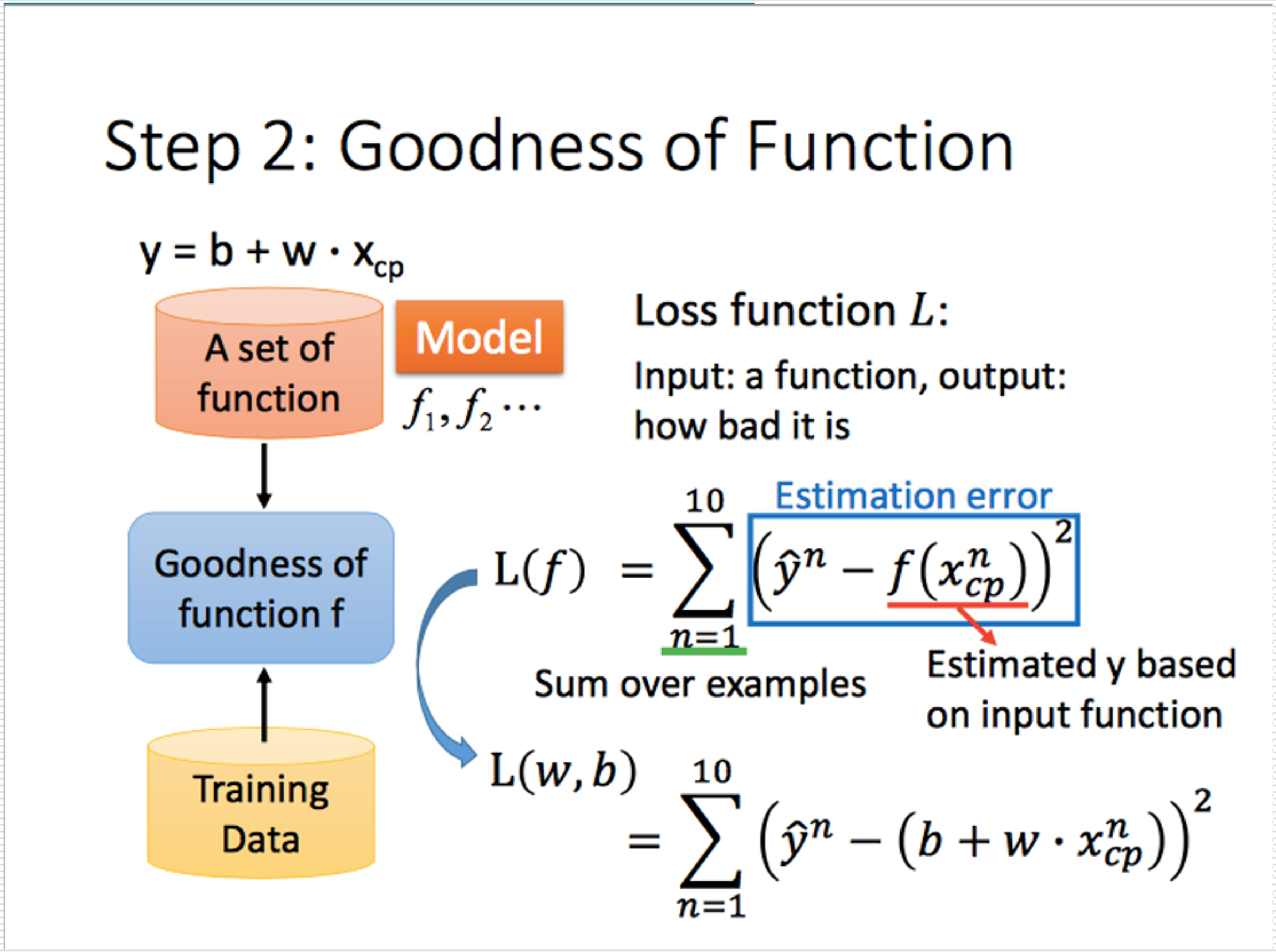
⋮

$(x^{10}, \hat{y}^{10})$

This is real data.

Source: https://www.openintro.org/stat/data/?data=pokemon

Linear Regression (3/7)



Step 2: Goodness of Function

Loss function L:

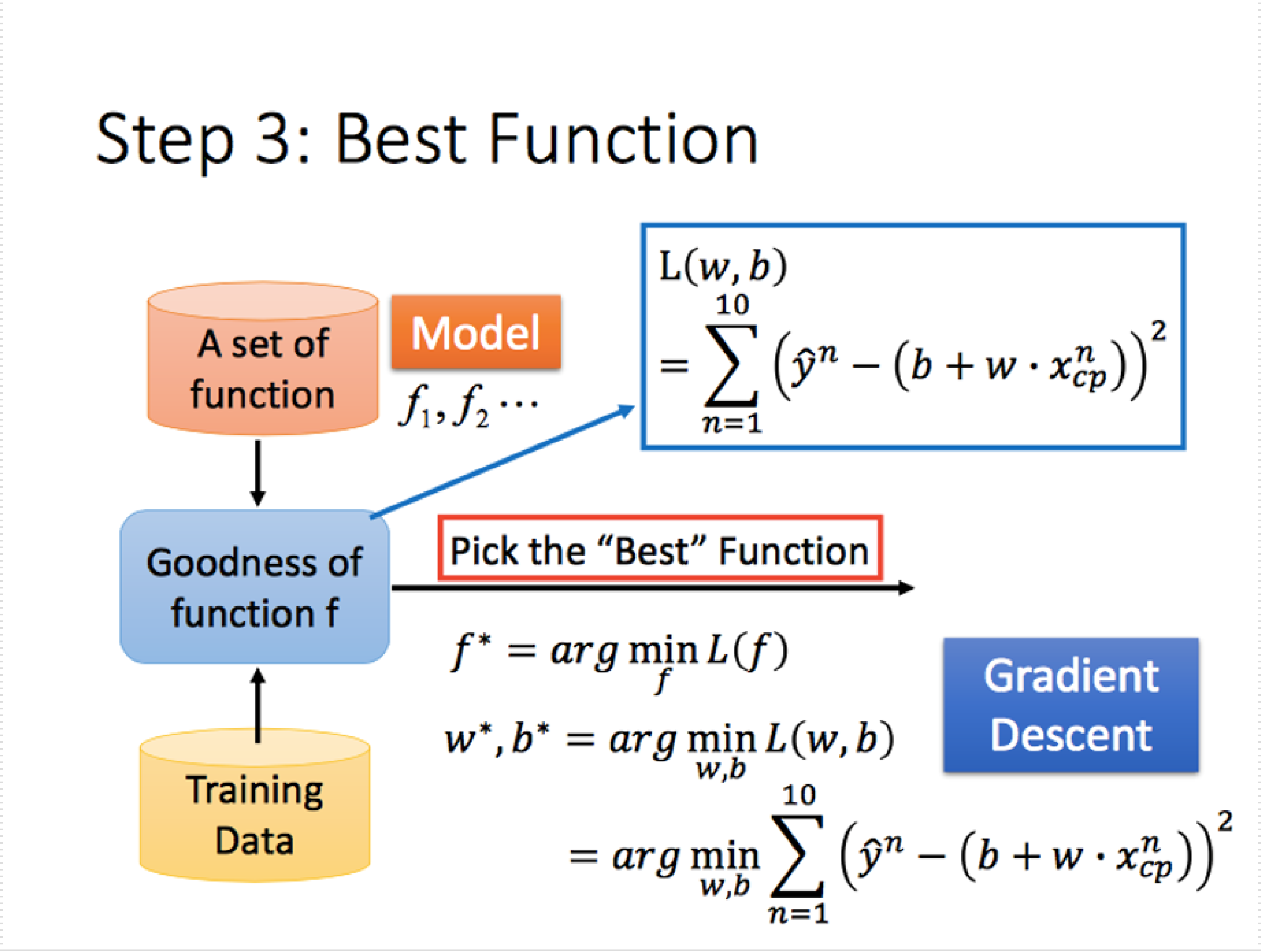
Input: a function, output: how bad it is

$$L(f) = \underbrace{\sum_{n=1}^{10}}_{\text{Sum over examples}} \overbrace{\boxed{\left(\hat{y}^n - \underline{f\left(x_{cp}^n\right)}\right)^2}}^{\text{Estimation error}}$$

Estimated y based on input function

$$L(w,b) = \sum_{n=1}^{10} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

Linear Regression (4/7)



Step 3: Best Function

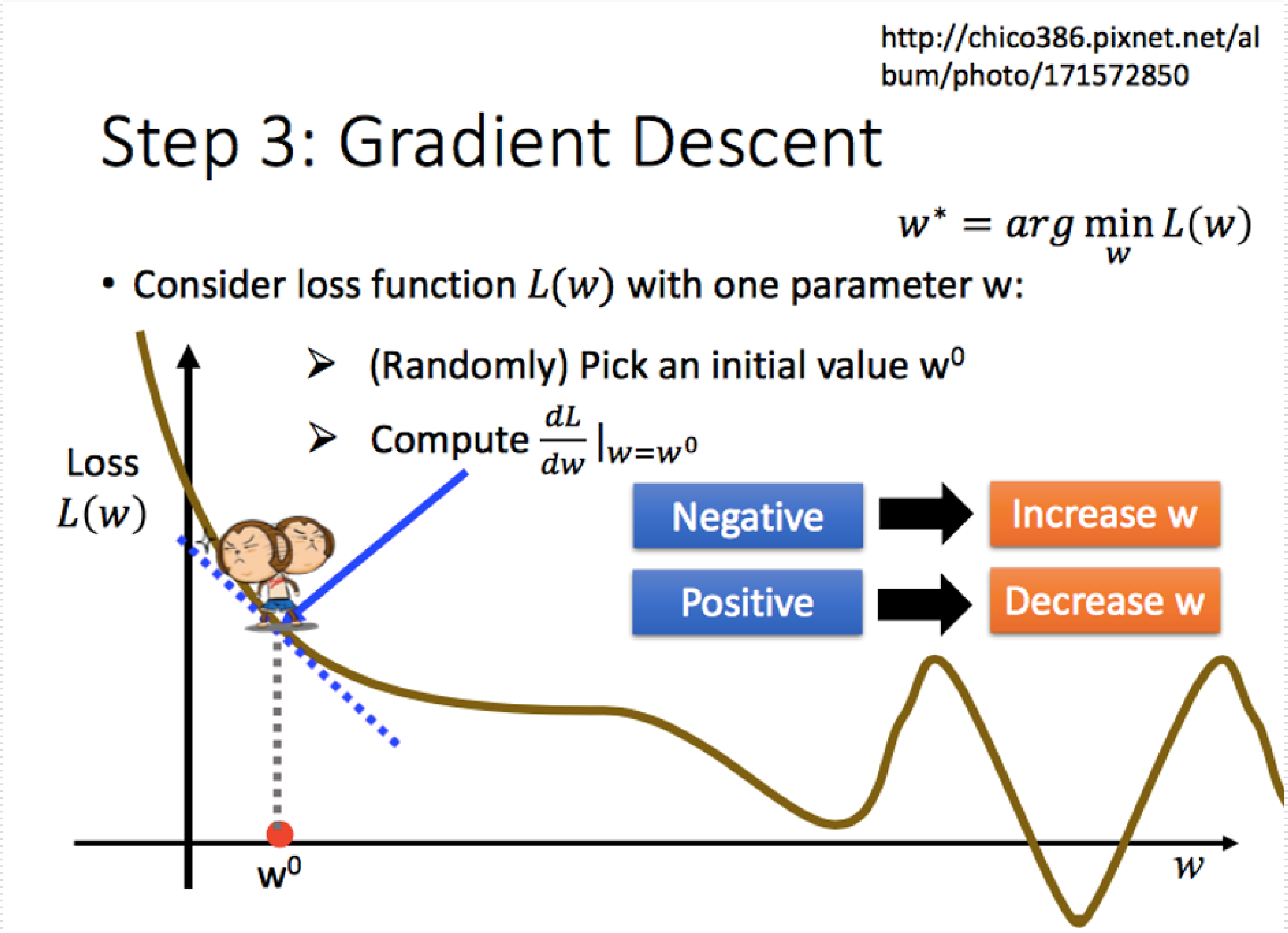
$$L(w,b) = \sum_{n=1}^{10} \left(\hat{y}^n - (b + w \cdot x_{cp}^n)\right)^2$$

$$f^* = arg \min_f L(f)$$

$$w^*, b^* = arg \min_{w,b} L(w,b)$$

$$= arg \min_{w,b} \sum_{n=1}^{10} \left(\hat{y}^n - (b + w \cdot x_{cp}^n)\right)^2$$

Linear Regression (5/7)



http://chico386.pixnet.net/album/photo/171572850

Step 3: Gradient Descent

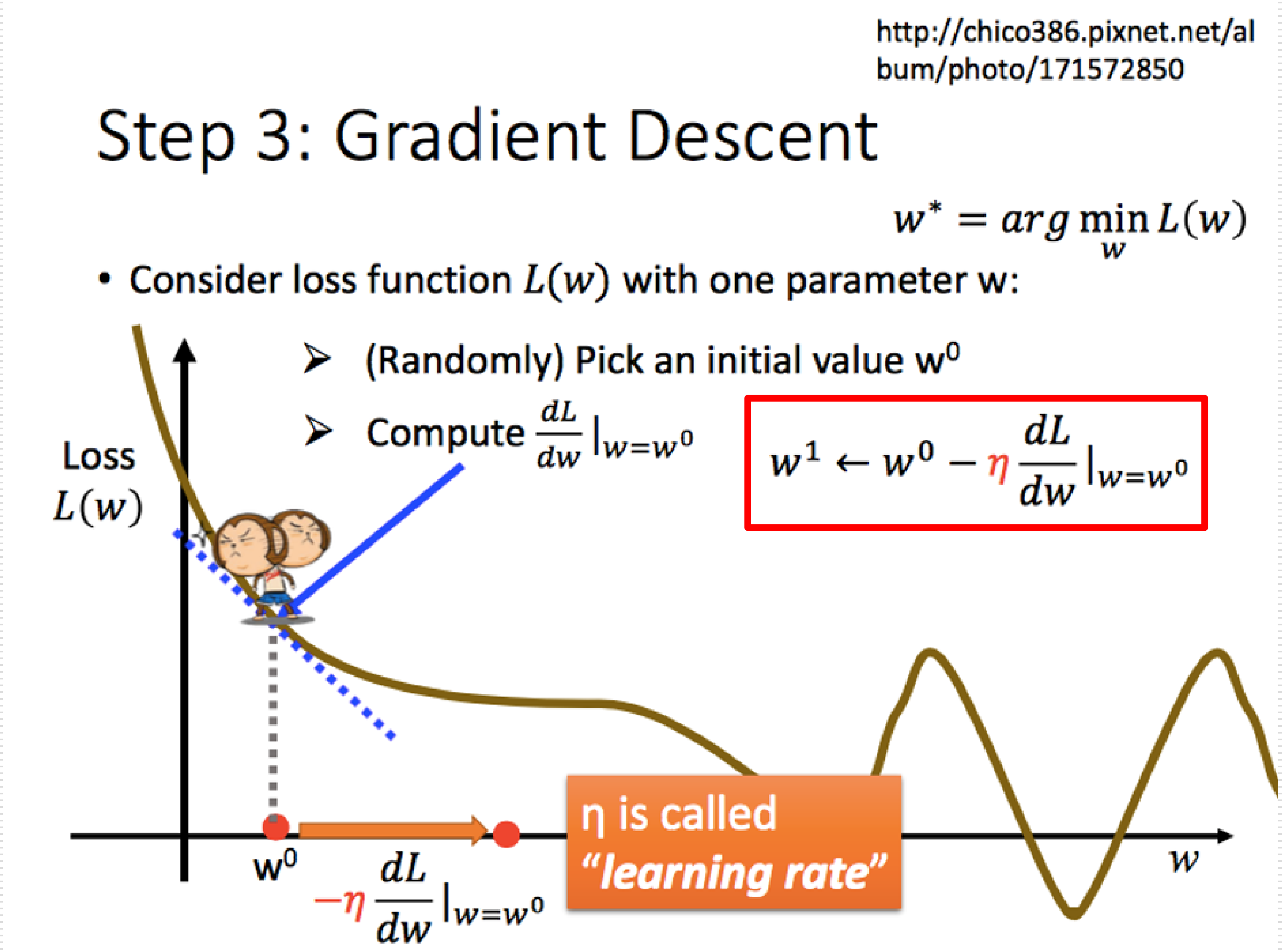
$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:
  - ➢ (Randomly) Pick an initial value w^0
  - ➢ Compute $\frac{dL}{dw}|_{w=w^0}$

Negative → Increase w

Positive → Decrease w

Linear Regression (6/7)



http://chico386.pixnet.net/album/photo/171572850

Step 3: Gradient Descent

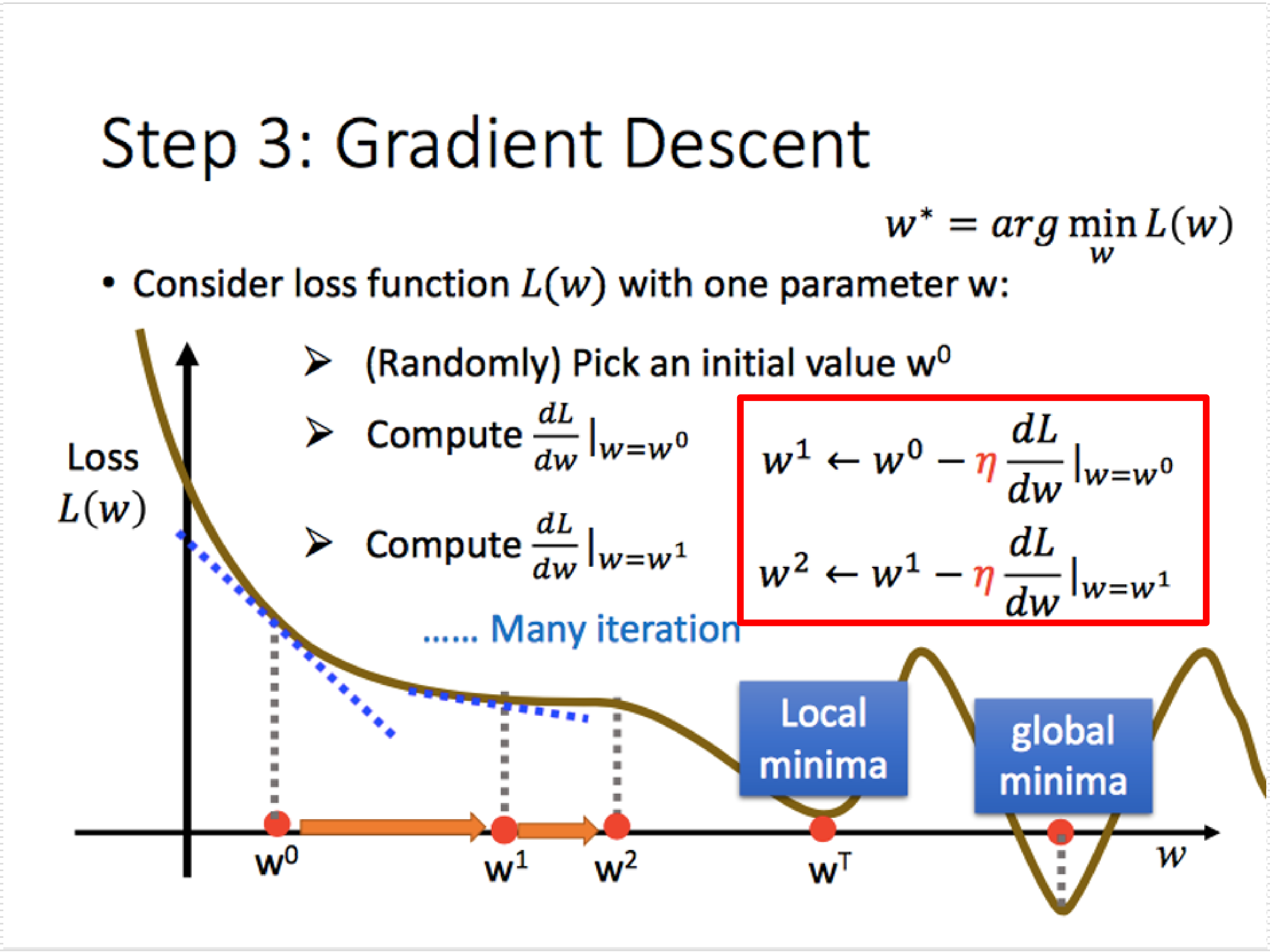
$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:
  - (Randomly) Pick an initial value w^0
  - Compute $\frac{dL}{dw}|_{w=w^0}$

$$\boxed{w^1 \leftarrow w^0 - \eta \frac{dL}{dw}|_{w=w^0}}$$

η is called *"learning rate"*

Linear Regression (7/7)



Step 3: Gradient Descent

$$w^* = arg \min_w L(w)$$

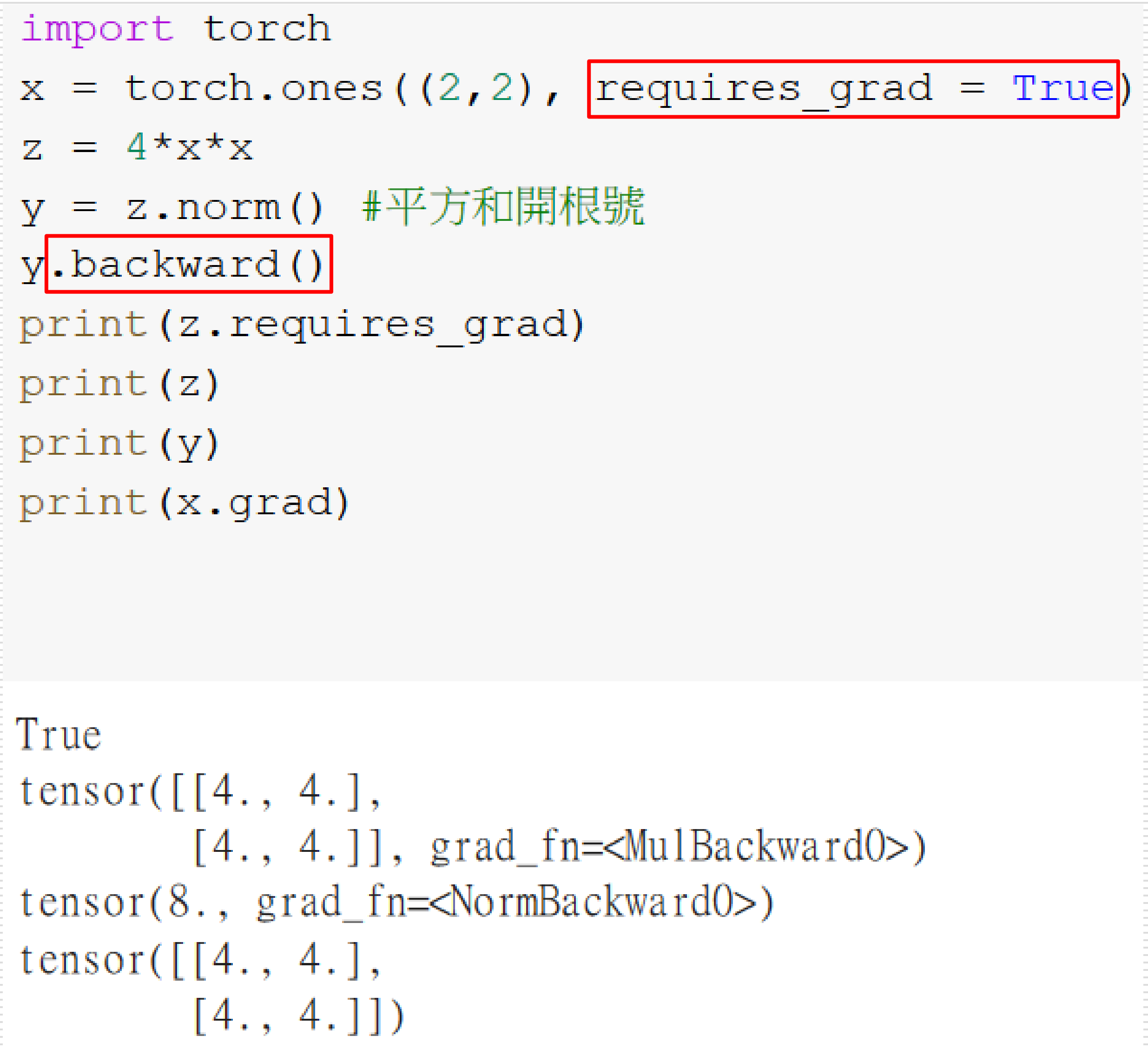
- Consider loss function $L(w)$ with one parameter w:
  - ➢ (Randomly) Pick an initial value w^0
  - ➢ Compute $\frac{dL}{dw}|_{w=w^0}$
  - ➢ Compute $\frac{dL}{dw}|_{w=w^1}$

$$w^1 \leftarrow w^0 - \eta \frac{dL}{dw}|_{w=w^0}$$

$$w^2 \leftarrow w^1 - \eta \frac{dL}{dw}|_{w=w^1}$$

…… Many iteration

Autograd: Automatic Differentiation (1/3)



```
import torch
x = torch.ones((2,2), requires_grad = True)
z = 4*x*x
y = z.norm() #平方和開根號
y.backward()
print(z.requires_grad)
print(z)
print(y)
print(x.grad)
```

```
True
tensor([[4., 4.],
        [4., 4.]], grad_fn=<MulBackward0>)
tensor(8., grad_fn=<NormBackward0>)
tensor([[4., 4.],
        [4., 4.]])
```

Autograd: Automatic Differentiation (2/3)

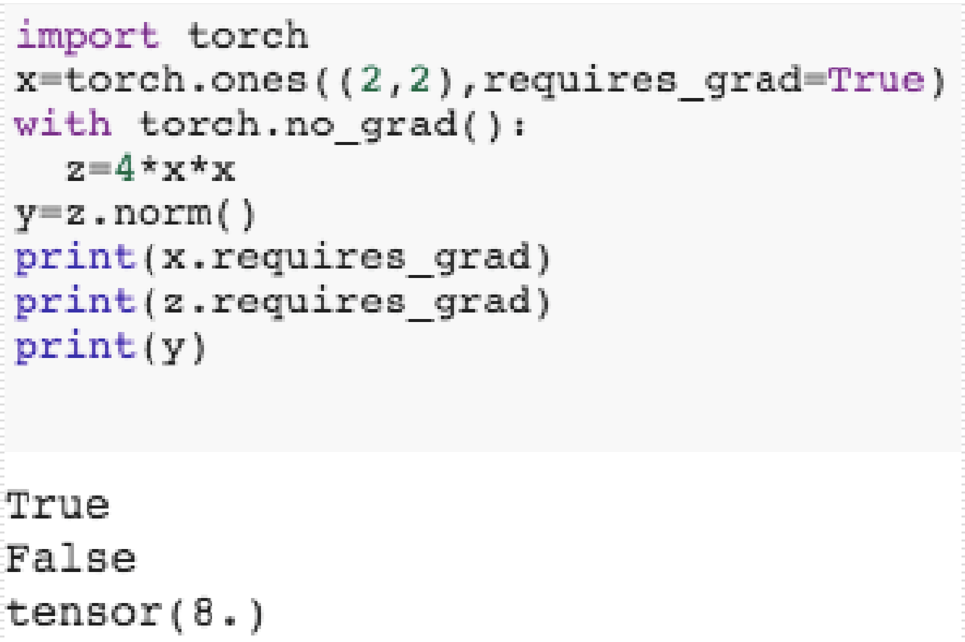
- backward():
  - 如果呼叫 backward() 的 tensor 不是 scalar 會報錯

```
import torch
x = torch.ones((2,2), requires_grad = True)
z = 4*x*x
z.backward()
```

```
RuntimeError: grad can be implicitly created only for scalar outputs
```

Autograd: Automatic Differentiation (3/3)

- with torch.autograd.no_grad() ➡ 此參數之後的參數不需要再算gradient ,可降低內存、加速計算
- A.grad.data.zero_() ➡ 將 gradient 歸零
  - 在 nn.Module裡面,被包裝成.zero_grad()



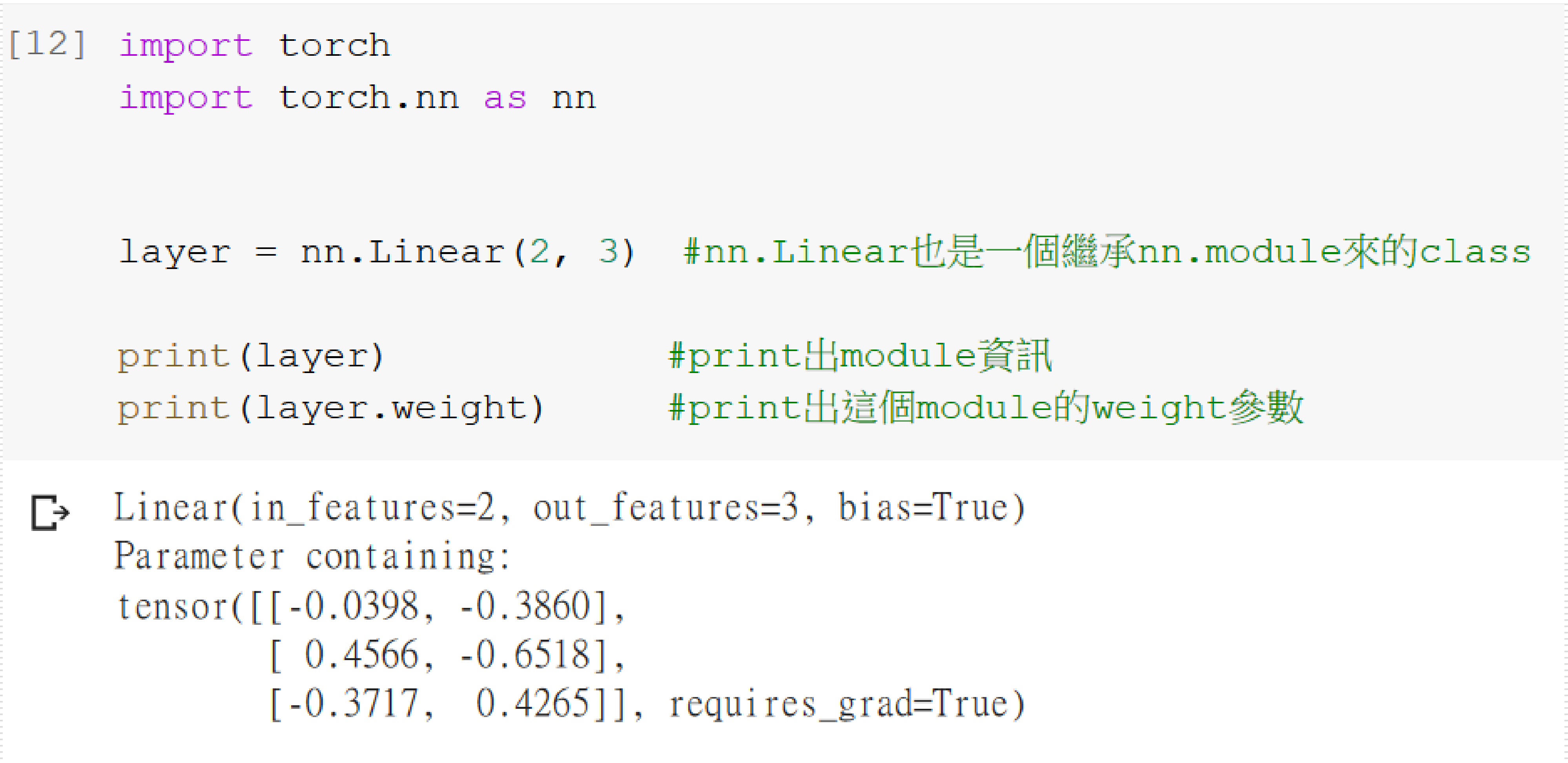
```
import torch
x=torch.ones((2,2),requires_grad=True)
with torch.no_grad():
    z=4*x*x
y=z.norm()
print(x.requires_grad)
print(z.requires_grad)
print(y)
```

```
True
False
tensor(8.)
```

Torch.nn

- import torch.nn as nn
  - pytorch 針對類神經網路包成一包模組
- torch.nn 提供的function：
  - Convolution
  - Pooling
  - Linear
  - Dropout
  - Activation function
  - Loss function
  - …

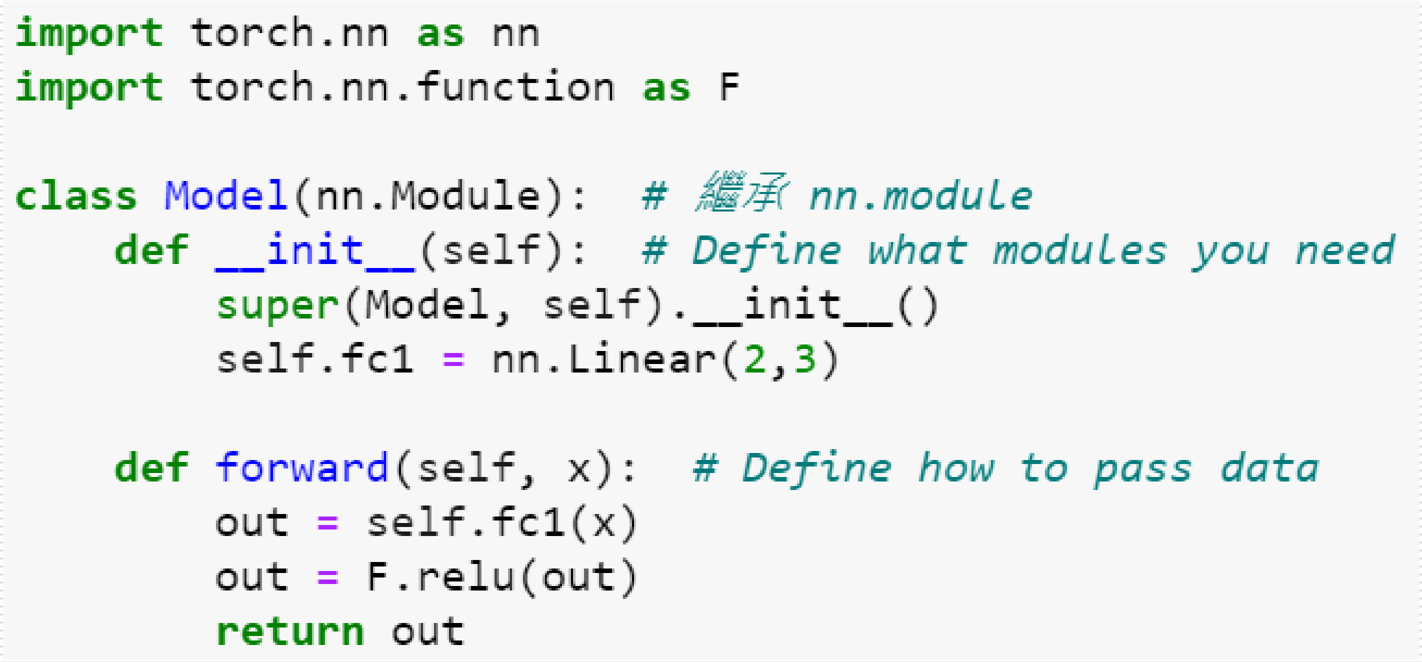
Torch.nn



```
[12] import torch
     import torch.nn as nn



     layer = nn.Linear(2, 3)  #nn.Linear也是一個繼承nn.module來的class

     print(layer)             #print出module資訊
     print(layer.weight)      #print出這個module的weight參數
```

```
Linear(in_features=2, out_features=3, bias=True)
Parameter containing:
tensor([[-0.0398, -0.3860],
        [ 0.4566, -0.6518],
        [-0.3717,  0.4265]], requires_grad=True)
```

Torch.nn.Module



```
import torch.nn as nn
import torch.nn.function as F

class Model(nn.Module):  # 繼承 nn.module
    def __init__(self):  # Define what modules you need
        super(Model, self).__init__()
        self.fc1 = nn.Linear(2,3)

    def forward(self, x):  # Define how to pass data
        out = self.fc1(x)
        out = F.relu(out)
        return out
```

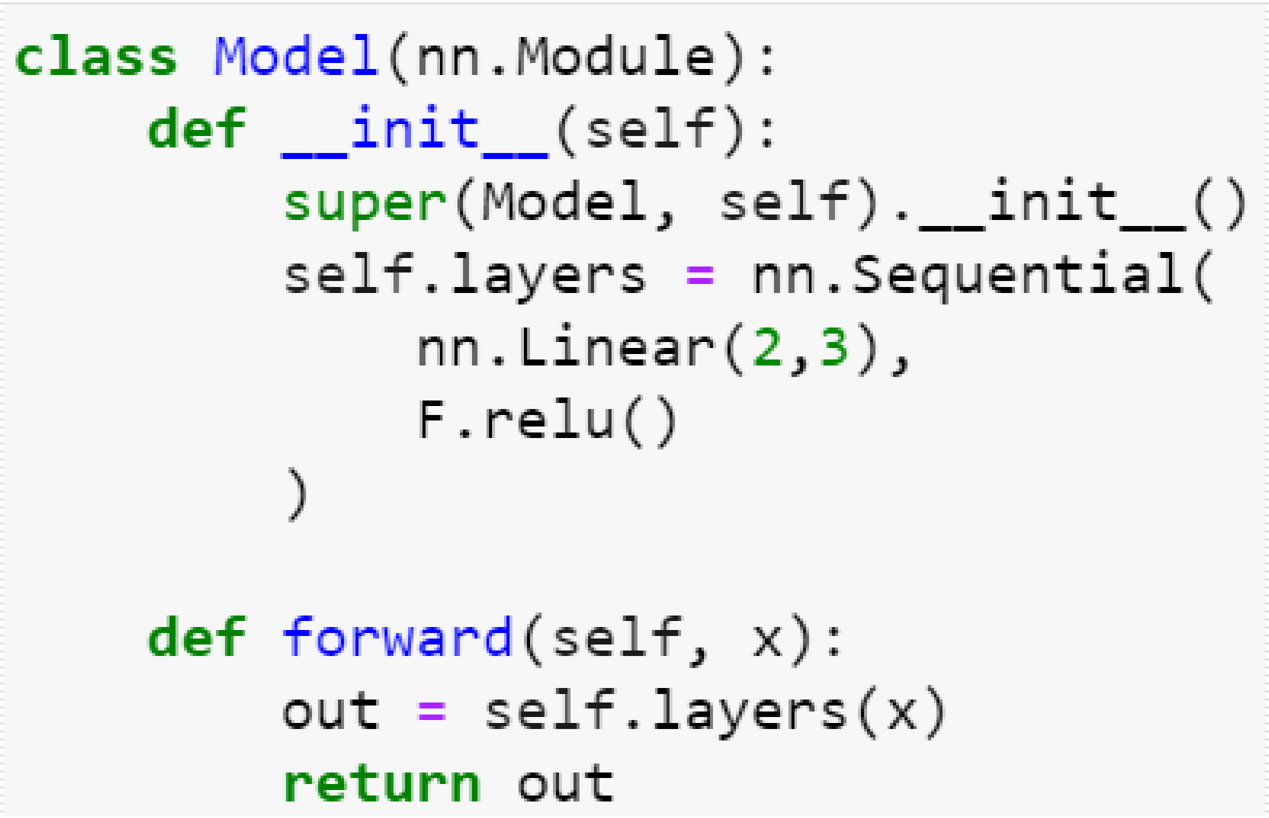
- 核心功能：
  - add_module(name,module) ➡ 將子模塊加到當前的模塊
  - forward(*input) ➡ 前向傳播
  - state_dict() ➡ 保存module的參數資訊
  - load_state_dict() ➡ 用來加載模型參數

Torch.nn.Module

- 查看模塊
  - parameters() ➡ "Model" 的所有參數
  - modules() ➡ "Model"這個 module 和 "Model" 裡的所有子 module
- 設置模式:
  - train() ➡ 將module設置為training mode
  - zero_grad() ➡ 將module所有梯度設為0

Torch.nn.Sequential

- modules依順序添加到容器中



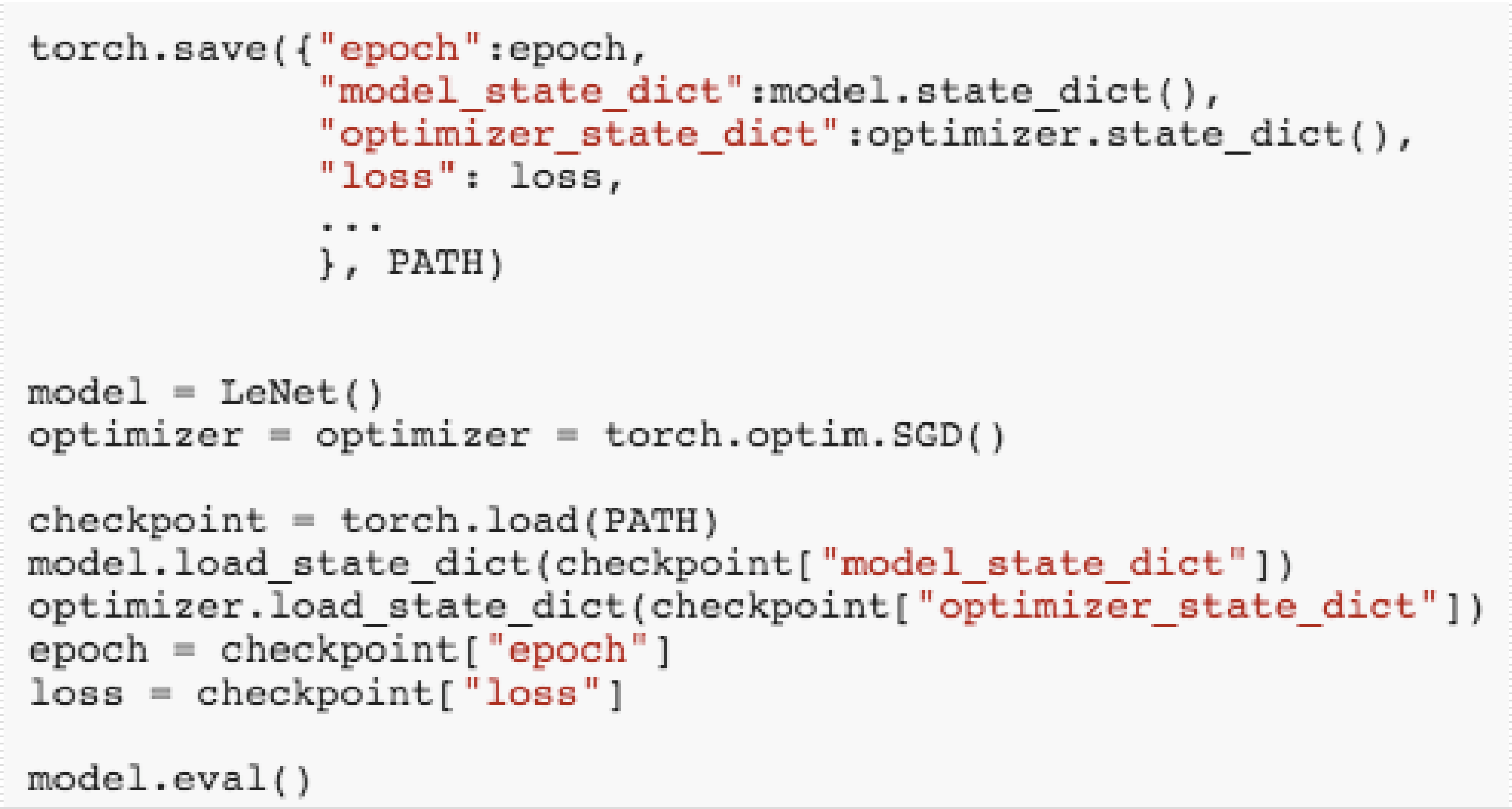
```python
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(2,3),
            F.relu()
        )

    def forward(self, x):
        out = self.layers(x)
        return out
```

Saving & Loading a Model (1/2)

- Save entire model
  - torch.save(model, PATH)
  - 將 model 儲存到 PATH 的位置
- Load entire model
  - model = torch.load(PATH)

Saving & Loading a Model (2/2)

- Saving & Loading a General Checkpoint for Inference and/or Resuming Training

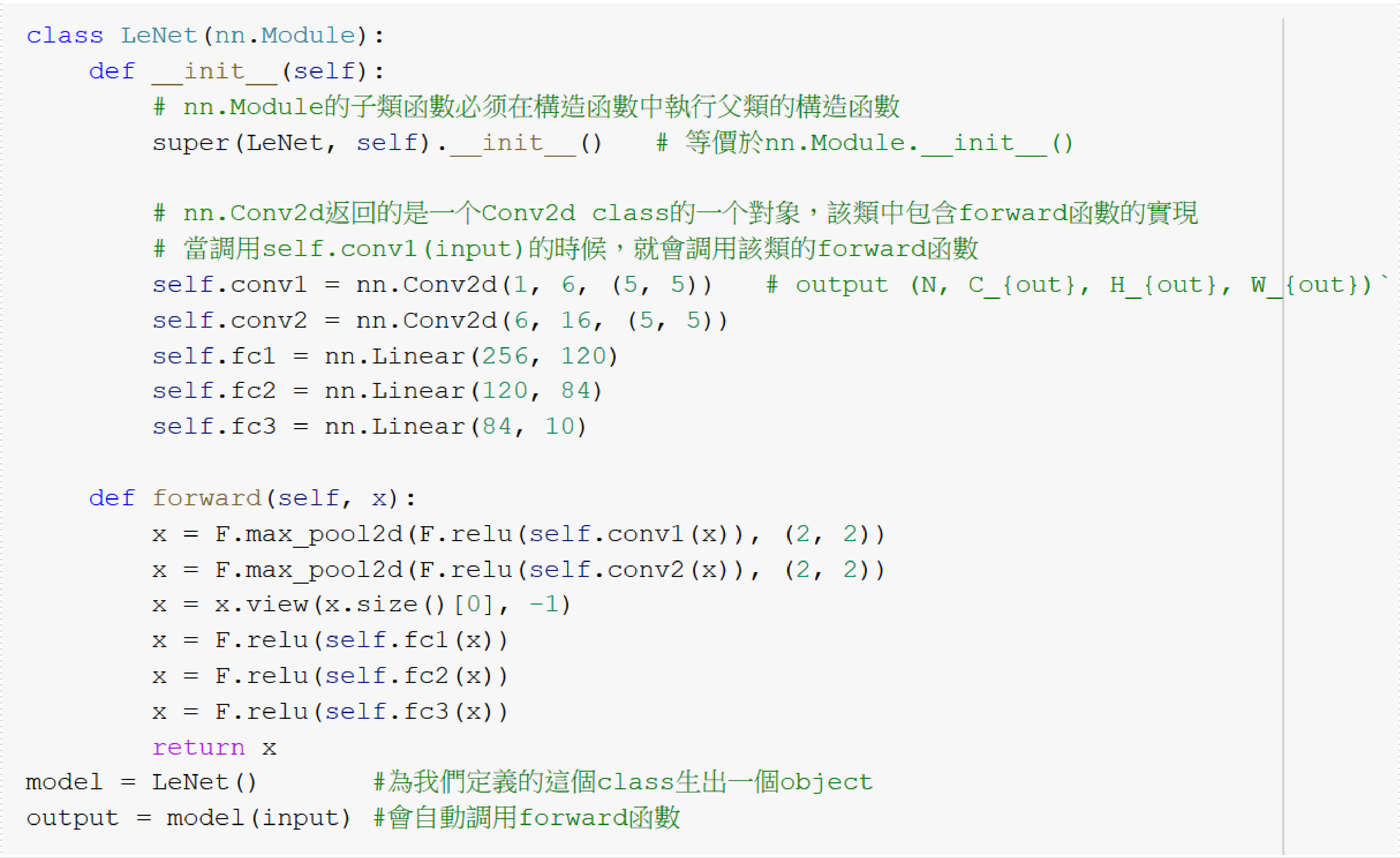


```
torch.save({"epoch":epoch,
            "model_state_dict":model.state_dict(),
            "optimizer_state_dict":optimizer.state_dict(),
            "loss": loss,
            ...
            }, PATH)


model = LeNet()
optimizer = optimizer = torch.optim.SGD()

checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint["model_state_dict"])
optimizer.load_state_dict(checkpoint["optimizer_state_dict"])
epoch = checkpoint["epoch"]
loss = checkpoint["loss"]

model.eval()
```

Torch.nn.functional

- import torch.nn.functional as F
- torch.nn vs. torch.nn.functional
  - torch.nn.functional提供的是純函數
  - torch.nn提供的是包括完整的nn.Module(包含參數資訊)
- 需要維持參數狀態的，主要是convolution layer 和 linear layer，所以用torch.nn所提供的module
- 而在計算時，relu、dropout、pooling不需要保存狀態的可以直接使用torch.nn.function

Example Code



```
class LeNet(nn.Module):
    def __init__(self):
        # nn.Module的子類函數必须在構造函數中執行父類的構造函數
        super(LeNet, self).__init__()   # 等價於nn.Module.__init__()

        # nn.Conv2d返回的是一个Conv2d class的一个對象，該類中包含forward函數的實現
        # 當調用self.conv1(input)的時候，就會調用該類的forward函數
        self.conv1 = nn.Conv2d(1, 6, (5, 5))   # output (N, C_{out}, H_{out}, W_{out})`
        self.conv2 = nn.Conv2d(6, 16, (5, 5))
        self.fc1 = nn.Linear(256, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = x.view(x.size()[0], -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        return x
model = LeNet()        #為我們定義的這個class生出一個object
output = model(input) #會自動調用forward函數
```

References

- 台大李宏毅老師Youtube
  - https://www.youtube.com/channel/UC2ggjtuuWvxrHHHiaDH1dlQ
- 莫凡pytorch
  - https://morvanzhou.github.io/tutorials/machine-learning/torch/

Thank you