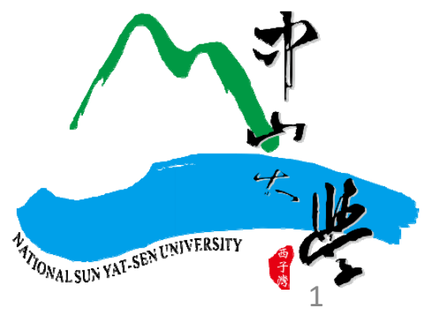
Python Programming

國立中山大學
資訊工程系
張雲南



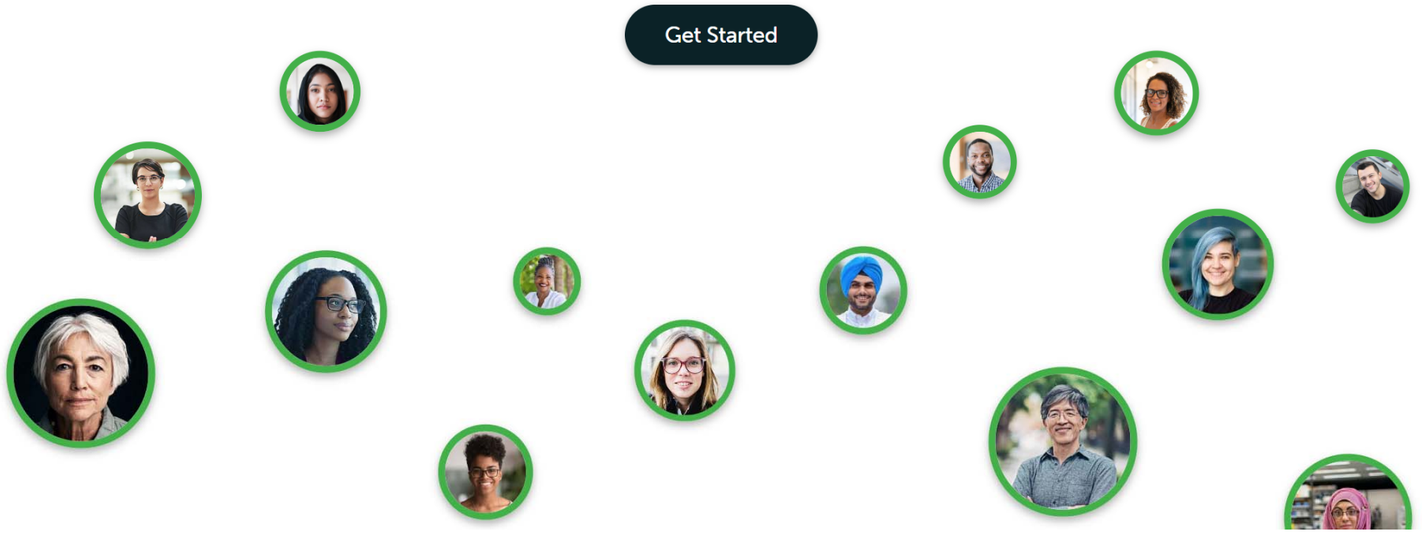
0

Python Development Environment

Go to Anaconda webpage and click Download

Data science technology for
human sensemaking.

A movement that brings together millions of data science practitioners,
data-driven enterprises, and the open source community.

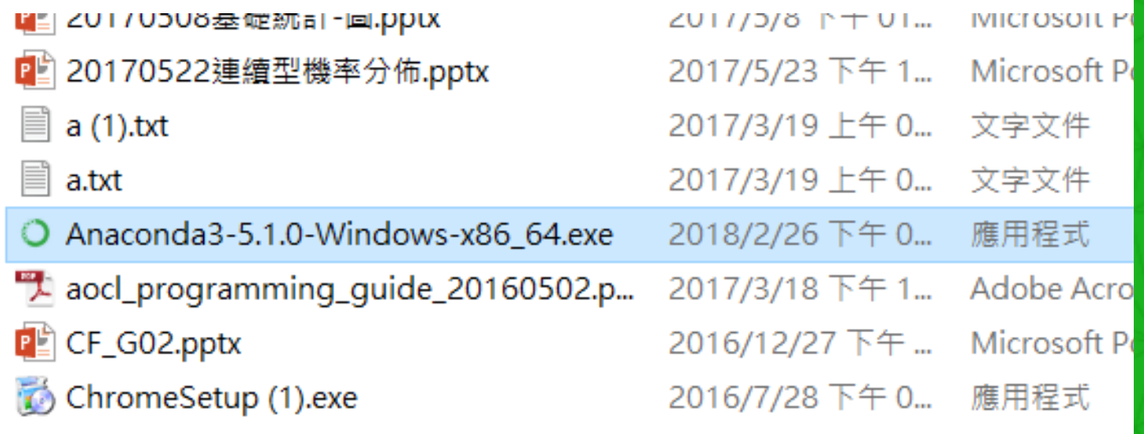


Download Python 3.7 version and install it

https://www.anaconda.com/products/individual

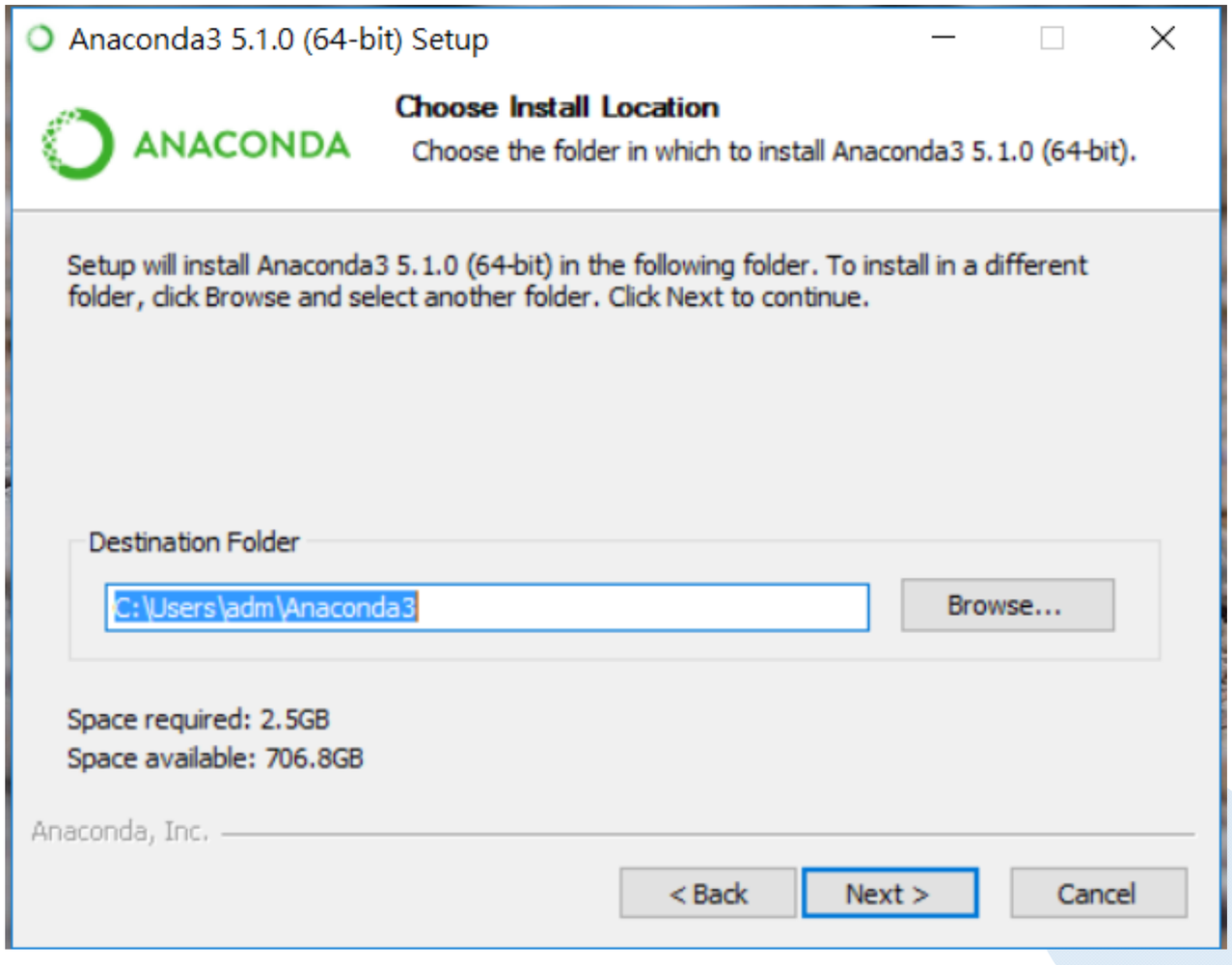
Anaconda 安裝

- 點擊安裝檔



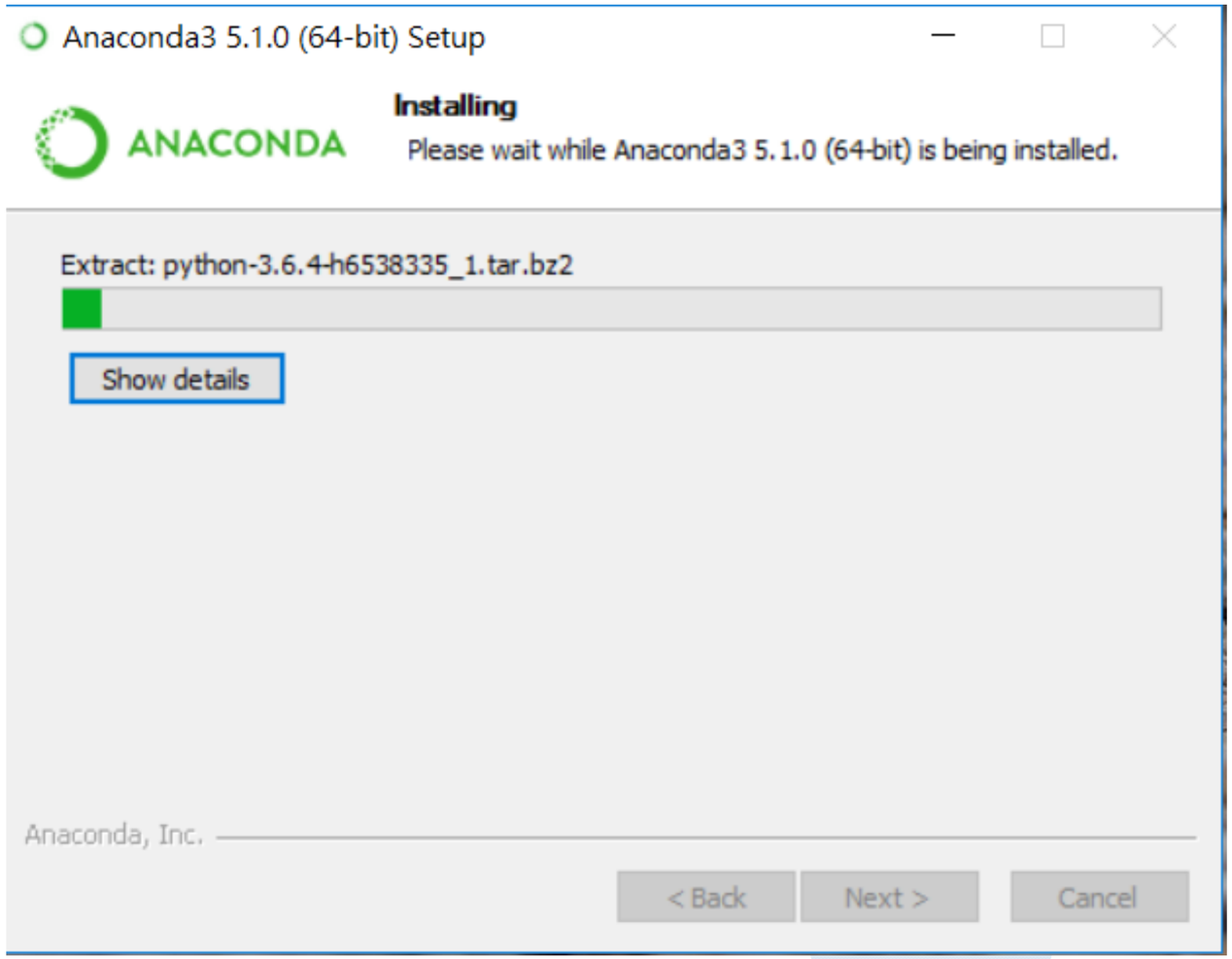
Anaconda 安裝

- 設定安裝路徑

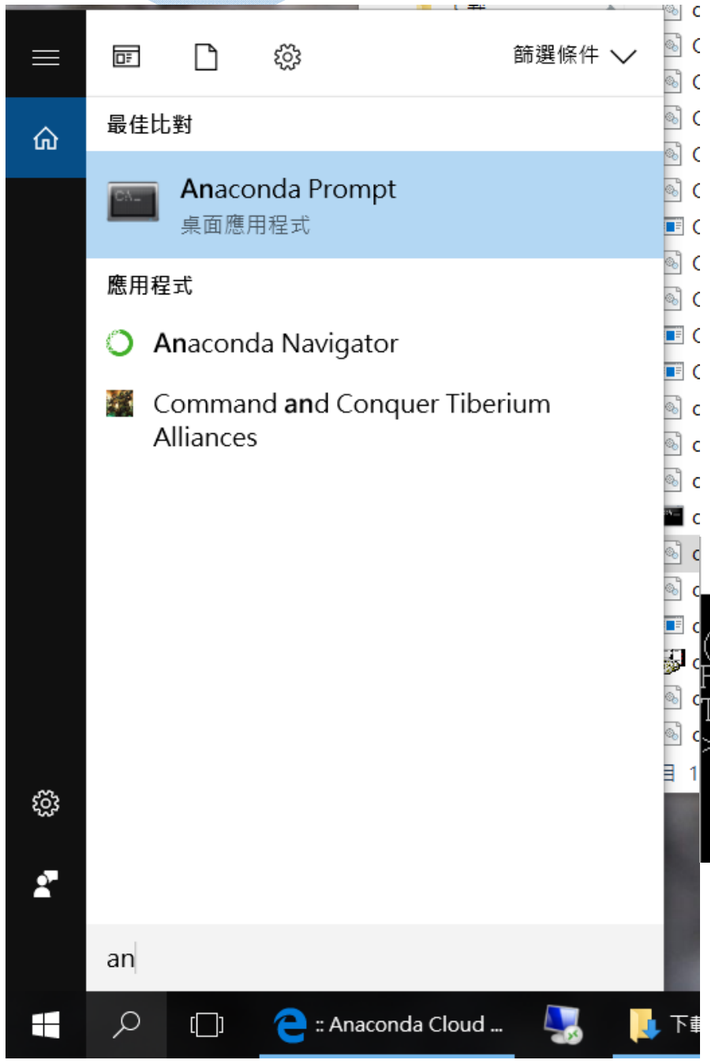


Anaconda 安裝

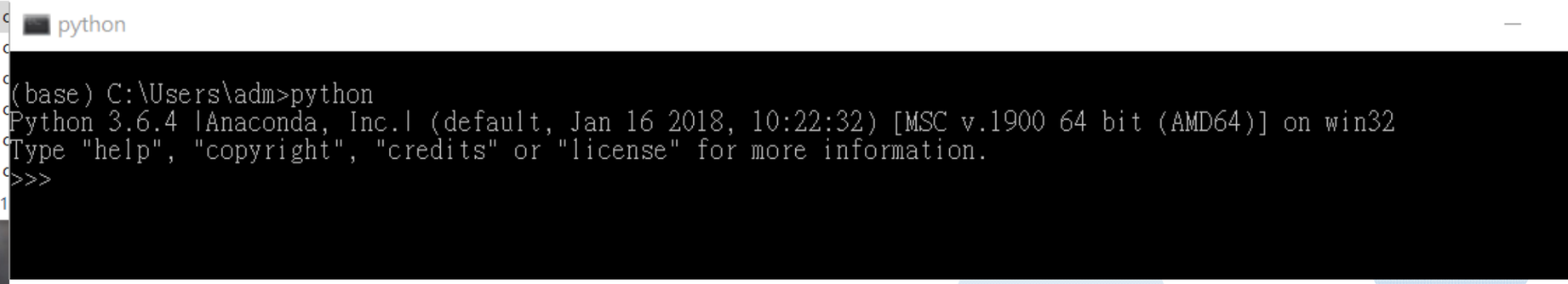
- 等待安裝完畢



Anaconda 安裝



- 搜尋anaconda prompt
- 在終端機輸入python，按下enter



Anaconda 安裝

- 在視窗上測試python程式

```
(base) C:\Users\adm>python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print ('hello')
hello
>>> a=10
>>> b=100
>>> a+b
110
>>>
```

Python程式撰寫(利用終端機)

- Python程式附檔名為.py
- 建立 ***filename.py***

hello.py

```
1  for i in range (0,10):
2      print('hello world',i)
```

Python程式撰寫 (使用終端機)

- cd 到檔案存放位置

```
Anaconda Prompt

(base) C:\Users\adm>D:

(base) D:\>
```

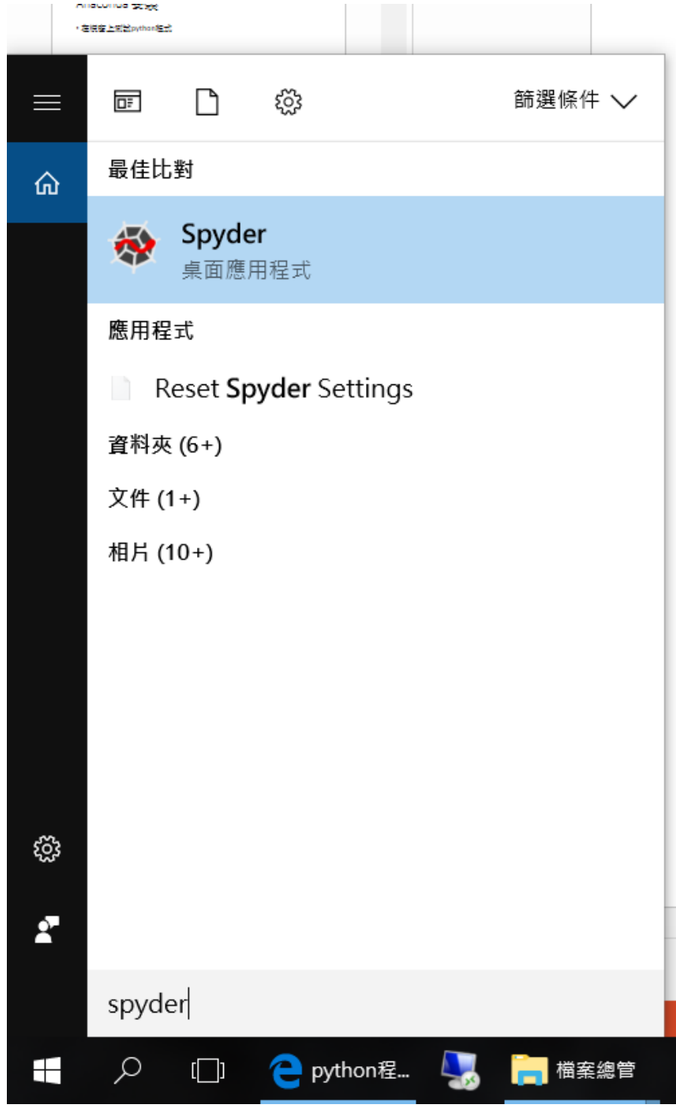
- 執行檔案:

  python ***filename.py***

```
(base) D:\>python hello.py
hello world 0
hello world 1
hello world 2
hello world 3
hello world 4
hello world 5
hello world 6
hello world 7
hello world 8
hello world 9
```

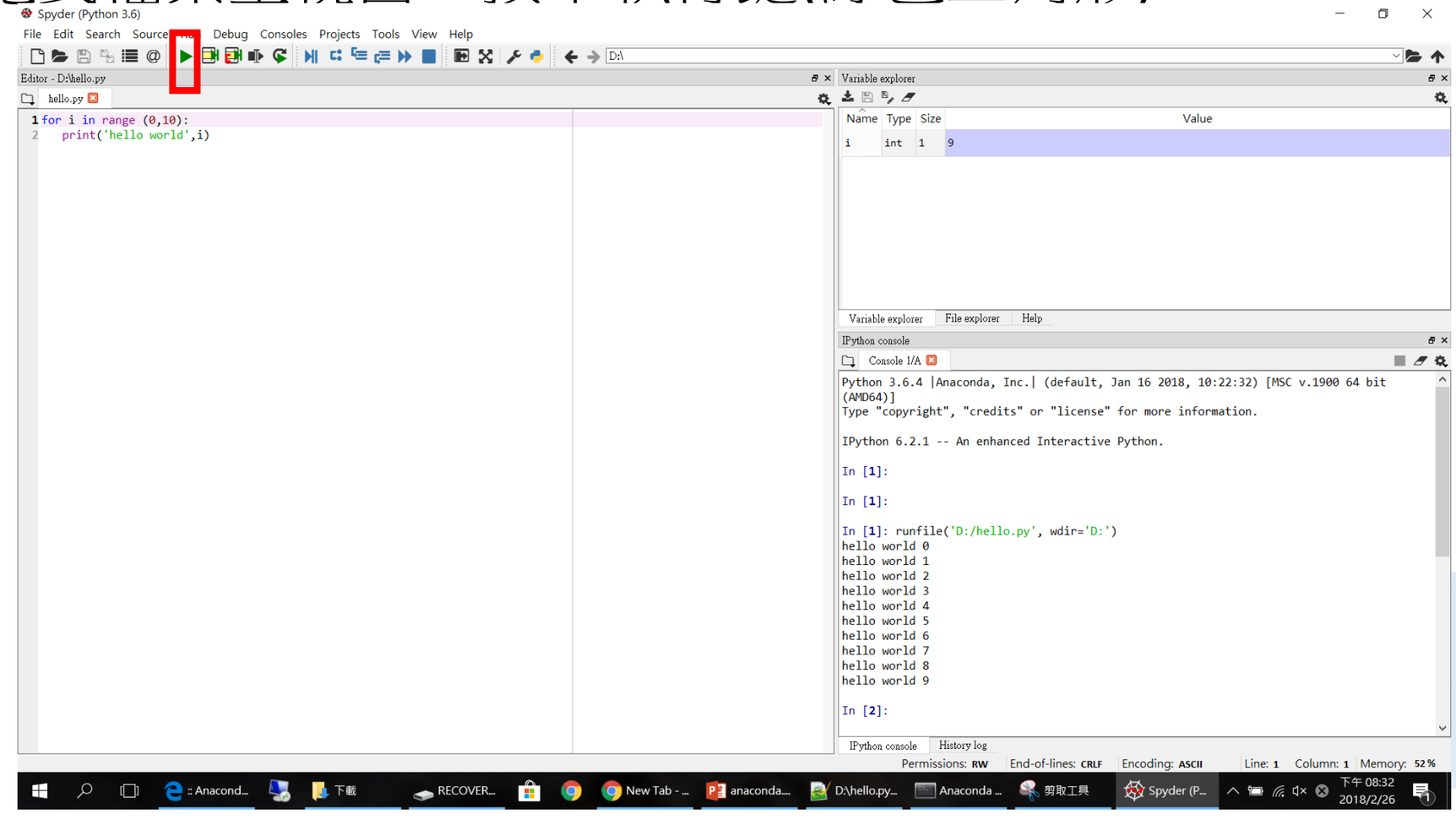
Python程式撰寫 (使用spyder)

- 搜尋spyder



Python程式撰寫 (使用spyder)

- 拖曳檔案至視窗，按下執行鍵(綠色三角形)



jupyter notebook

Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

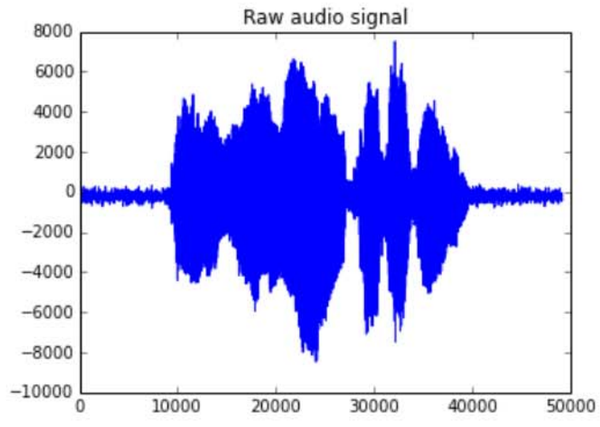
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

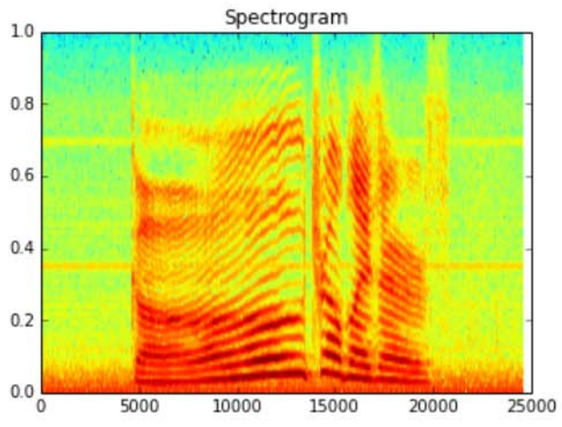
We begin by loading a datafile using SciPy's audio file support:

In [1]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

In [2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```





Python程式撰寫（jupyter notebook)

- 在anaconda prompt輸入jupyter notebook

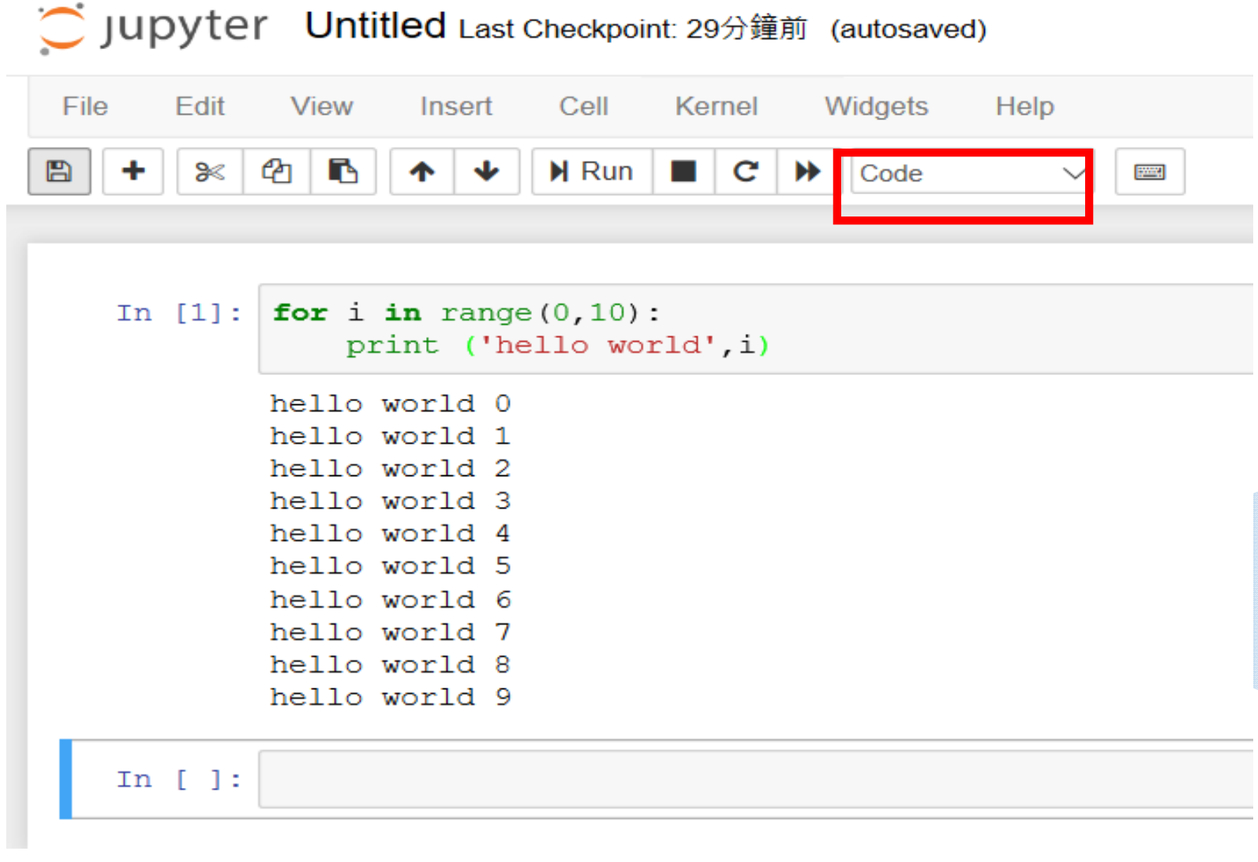
```
Nicolede-MacBook-Pro:~ nicole$ jupyter notebook
[I 14:46:34.851 NotebookApp] Serving notebooks from local directory: /Users/nicole
[I 14:46:34.851 NotebookApp] The Jupyter Notebook is running at:
[I 14:46:34.851 NotebookApp] http://localhost:8888/?token=dccc832d52d3f568f9846f648af439034fee4adc05778a6b
[I 14:46:34.851 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 14:46:34.854 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=dccc832d52d3f568f9846f648af439034fee4adc05778a6b
[I 14:46:35.118 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

Python程式撰寫 (jupyter notebook)

- 新增檔案
- New -> Python3

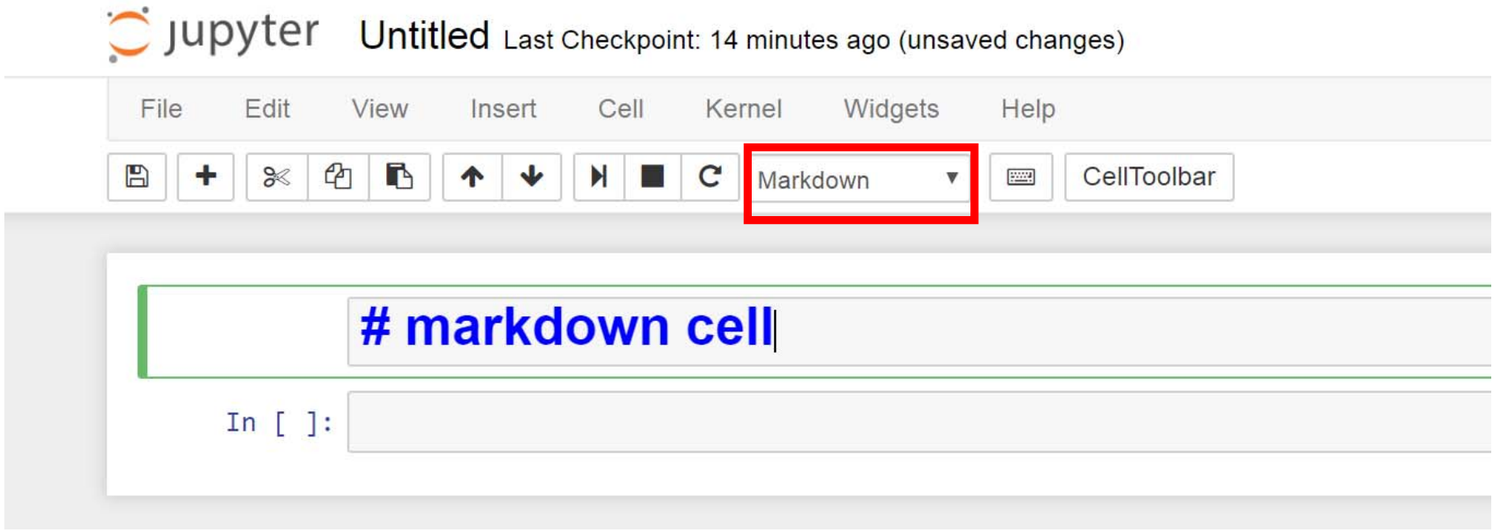
Python程式撰寫 (jupyter notebook)

- 在cell中撰寫程式，點擊run執行該cell
- 每個cell預設為”code” 模式



Markdown syntax

- 可將該cell切換為”Markdown”模式，用來做筆記



- You can refer to the markdown syntax from the related websites:
  - https://help.github.com/articles/getting-started-with-writing-and-formatting-on-github/
  - https://markdown.tw/

Some Markdown examples

File Edit View Insert Cell Kernel Widgets Help

Run Markdown

```
# 大標題

## 小標題
正體

*斜體*

**粗體**

***粗體又斜體***

## 分項說明
* 分項說明1
* 分項說明2
* 分項說明3

## 連結
[中山網路大學](http://cu.nsysu.edu.tw)

## 數學式
用LaTex語法

$f(x) = 3x^2 + 5x +7$
```

RUN

File Edit View Insert Cell Kernel Widgets Help

Run Markdown

大標題

小標題

正體

斜體

粗體

粗體又斜體

分項說明

- 分項說明1
- 分項說明2
- 分項說明3

連結

中山網路大學

數學式

用LaTex語法

$$f(x) = 3x^2 + 5x + 7$$

1

Python Basic Syntax

Recommend reference site: https://docs.python.org/3/

Quick comparison of Python vs C/C++

Python:

```
import numpy as np

string="9*9 multiplication table\n"
print(string)
psum=[0 for i in range(10)]
for i in range(1,10):
    print("++++++++++++++")
    for j in range(1,10):
        psum[i]+=i*j
        print(i, "x", j, "=", i*j)

print("total sum=", np.sum(psum));
```

C/C++:

```
#include <stdio.h>
int np_sum(){……}
int main(){
    int i, j, k;
    int psum[9];
    char *string = "9x9 multiplication table\n\n";
    printf("%s",string);
    for(i=1; i<10; i++) {
     print("++++++++++++++\n")
      psum[i]=0;
     for (j=1; j<10; j++) {
       psum[i]+= i*j;
       printf ("%d x %d =%d \n", i, j, i*j);
      }

    }
   print("total sum=%d \n", np_sum(psum));
}
```

Variable

- **In C++:**

```
int a = 5;  float b = 3.14159;
string c = "Hello World!";
```

變數存著資料!!!

- **In Python:**
  - **Don’t need to declare the data type. (It will be inferred automatically)**

```
A = 3.14159
B = 'Hello World!'
C = True
```

```
A = B = C = 20
```

有資料才有變數!!!

Multiple assignment

In Python:

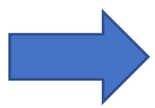
- Use syntax `A = B = C = 20` to assign variables to the **same** value (data object).
  - Then A, B, and C are all assigned to 20.
- Use the syntax `A , B , C = 10, 15, 20` to assign variables with **different** values.
  - Then A is 10, B is 15, and C is 20.

How about `A = 10, 15, 20` `*A, B = 10, 15, 20`

String Variable

- String variables can be enclosed in any types of the **quotation mark.**

```
A = 'Hello World!'
B = "Hello World!"
```



Both of them are OK!

- But in **C/C++** " " and ' ' are different.

  " " are for **string**, and ' ' are for **character**.

Type of variables

- No type declaration, but "**type**" still exists in Python.
- To know the type of the certain variable, use **type(variable)**
  - In fact, *type(variable)* is used to find what type of data the variable refers to.

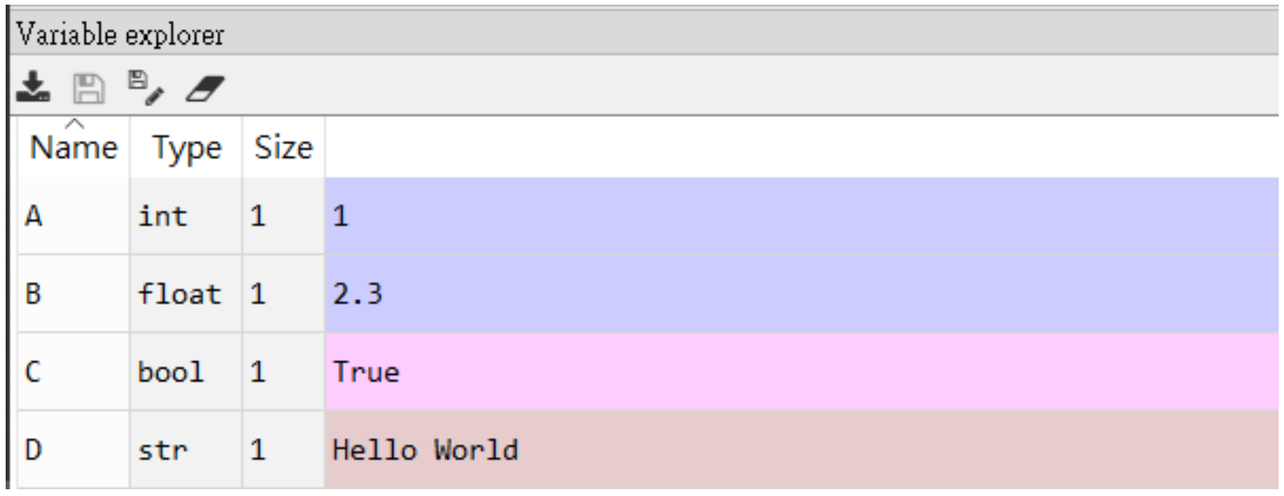
- **Ex**

```
A = 1
B = 2.3
C = True
D = "Hello World"
print(type(A), type(B), type(C), type(D))
```

<class 'int'> <class 'float'> <class 'bool'> <class 'str'>

Type of variables

- In **Spyder IDE**, we can get the type of each variable easily with **variable explorer**.




```
1 A=1
2 B=2.3
3 C=True
4 D="Hello World"
5 print(type(A),type(B),type(C),type(D))
```

Assignment = in Python

- **In Python**, the operator = is not the same as = in **C/C++**.
  - It assigns the reference of an object instead of a value.
- For example,

```
a = 1; b = 2
a, b = b, a
```

a = 1 **creates an object with the value of 1**, and let **"a" points to that object.**

- ***id(varA)*** returns the ***id*** of the object the variable ***varA*** refers to.
- ***A=B*** should be explained as let A refer to the object variable B refers to.

```
a = 1
b = 1
print(id(a), id(b))
```

140731748163984 140731748163984

Semicolon

- **In Python:** it's not necessary to add **semicolon** (**;**) at the end of each line, but adding it is fine!
- If **two statements are in same line**, we need to **separate them by ";".**

- **Ex**

```
A = 6
B = 3.14159
```

=

```
A = 6 ; B = 3.14159
```

Comments

- **In C/C++:** We use **//** to comment **one line** and **/* … */** to comment **multiple lines**.
- **Ex**

```
int a = 5;  //this is a comment
/* School: NSYSU
      Name: YUN NAN CHANG
      Job: Professor  */
```

- **In Python:** We use **#** or **""" … """** or **''' … '''** correspondingly.
- **Ex**

```
a = 5 #this is a comment
""" School: NSYSU
      Name: YUN-NAN CHANG
      Job: Professor  """
```

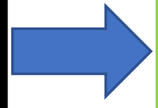
Print

- The basic form of the **print** function in Python is:

```
print (Item1, Item2, Item3...., sep = "separate character", end = "end character")
```

  - "**sep**" and "**end**" aren't necessary
    - **sep**: In default = " "(**white space**)
    - **end**: In default = "**\n**"
- **EX**

```
print ('PPAP')
print ('I', 'have', 'a', 'pen')
print ('I', 'have', 'an', 'apple', sep = 'BANG')
print ('Total', end = ':')
print ('Apple Pen!')
```



```
PPAP
I have a pen
IBANGhaveBANGanBANGapple
Total: Apple Pen!
```

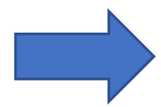
Print

- Can also use **print** in the similar way to **C** Language!

```
print (Item1, Item2,… % (variable list))
```

- **EX**

```
Name = “Chang"
Score = 60
print ("%s's score was %d." % (Name, Score))
```



Chang's score was 60.

Print

- The third way that *print* function supports is :

```
print ("{} … {}… " .format (variable list))
```

- **EX**

```
Name = “Chang"
Score = 60
print ("{}'s score was {}." .format (Name, Score))
print ("{1}'s score was {0}." .format (Name, Score))
```

→ **Chang's score was 60.**
60’s score was Chang.

Don’t put , here.

```
print (f"{Name}'s score was {Score}.")
```

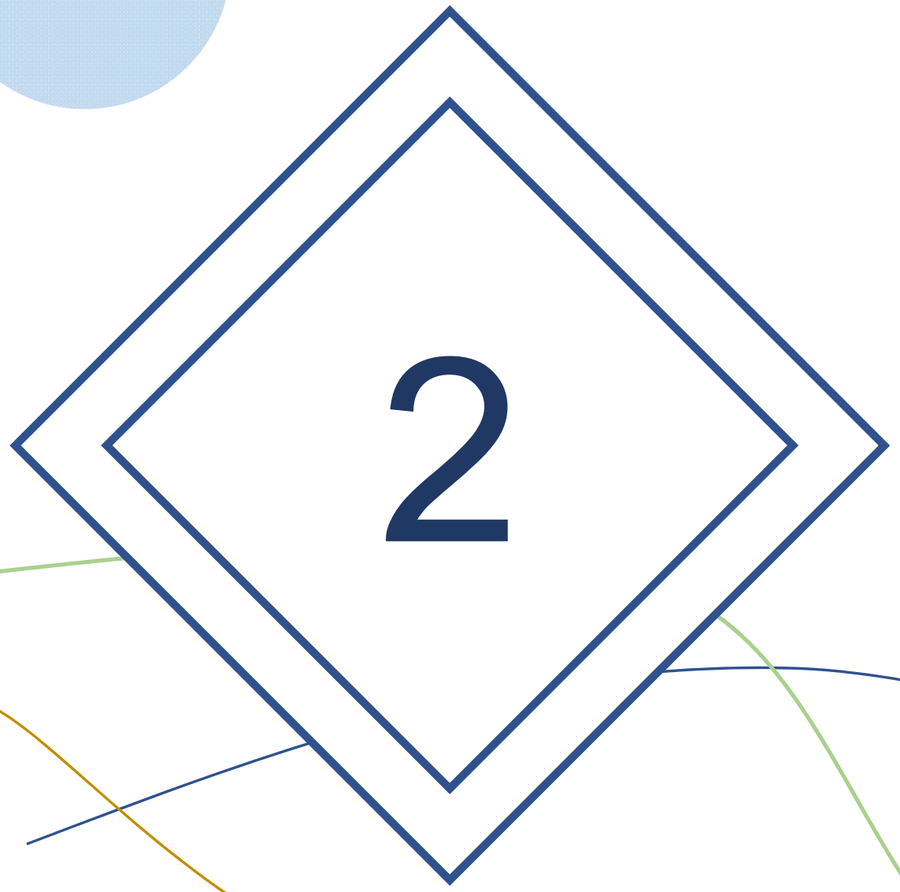
Input

- **In C:** use "**scanf**" to read the input from keyboard.
- **In C++:** use "**cin**"
- **In Python:** use "**input**" function
- **EX**

```
name = input ("What's your name?")
```

- Be careful, the "**input**" in Python return a "**string**" type variable. Remember to typecast !
- **Ex**

```
number = int (input ("What's your score?"))
```



Flow Control
If, For, While

If elif else

```
if expression:
    #code
elif expression:
    #code
else:
    #code
```

- **Python** uses **elif** instead of "**else if**" in **C/C++.**
- **Python** uses ***indentation*** to indicate a block of codes! So **indentation is necessary**. Also, colon "**:**" is required.
- **Expression** doesn't need to add **parentheses**, but adding them is fine.
  - **Ex**

```
if (score == 100) :
    print("Excellent!")
elif score <= 60 :
    print("Failed!")
else:
    print("Pass!")
```

and not or

- In **Python:** Use “**and**”, “**not**”, “**or**” instead of “&&”, “!”, “||” in **C/C++**.
- Ex

```
if score == 100 :
    print("Excellent! ")
elif (score <= 60) and (score >= 0):
    print("Failed!")
elif (score >= 60) and (score < 85):
    print("Great!")
elif (score >= 85) and (score < 100):
    print("Awesome!")
else:
    print(“Cheating!")
```

Operator

Arithmetic Operators

		Example	Answer
+ - *	Addition, Subtraction, Multiplication	X = 2*5	X = 10
/	Division	X = 100/3	X = **33.3**
//	Get Quotient of division operation	X = 100//3	X = **33**
%	Modulus	X = 100%3	X = 1
**	Exponent	X = 2**3	X = 8

Comparison Operators

		Example (A = 3; B = 5)	Answer
== != <>	Is equal, not equal, not equal	A != B	True
> <	Greater than, Less than	A > B	False
>= <=	Greater or equal to, Less or equal to	A >= B	False

Assignment Operator

- Unlike **C/C++, Python** doesn't support operators like **x++** or **--x**.
- But still have assignment operators like x**+=**2, x***=**2, x**%=**2…

- **Question**
  - If **x = 5**, what will X be after running **x/=2** ?

Ans

x = 2.5

- *Again, after the value of variable has been changed, the variable refers to a new data object.*

```
x=2
print(id(x), x)
x+=2
print(id(x), x)
```

```
140707458752944 2
140707458753008 4
```

For Loop

```
for variable in list:
    #code
```

For iteration!!

- Again, **Python** uses ***indentation*** to indicate a block of codes!
- **Ex**

```
x = range(6)
for i in x:
    print(i)
```

=

```
for i in range(6):
    print(i)
```

```
0
1
2
3
4
5
```

- The usage of "**break**" and "**continue**" are same as **C/C++**

Range

```
range ([start], end, [interval])
```

- This function creates a list from **start** (**optional; default 0**) to **end - 1** with an **interval** (**optional; default 1**)
  - If only one number is filled in the range function, it start from **0** to **end - 1**, with an interval = **1**
- **Ex**

```
x = range(6)
y = range(-2, 3)
z = range(-2, 6, 2)
```

```
x = [0, 1, 2, 3, 4, 5]
y = [-2, -1, 0, 1, 2]
z = [-2, 0, 2, 4]
```

Questions

- What will **A** become after **A = list(range(-2, 3, 2))**

Ans

A = [-2, 0, 2]

- Create a list from 5 to -1.

Ans

A = list(range(5, -2, -1))

for loop with else

- **What if we do:**

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, "is not a prime number")
            break
        else:
            print(n, "is a prime number")
```

```
3 is a prime number
4 is not a prime number
5 is a prime number
5 is a prime number
5 is a prime number
6 is not a prime number
7 is a prime number
7 is a prime number
7 is a prime number
7 is a prime number
8 is not a prime number
9 is a prime number
9 is not a prime number
```

Why doesn't it show "***2 is a prime number***''?

for-else usage

- **In Python:** We can do a handy trick like something shown in this example.

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, "is not a prime number")
            break
    else:
        # loop fell through without finding a factor
        print(n, "is a prime number")
```

◆ ***Because if n is 2, x is range(2,2), and it won't do anything.***

```
2 is a prime number
3 is a prime number
4 is not a prime number
5 is a prime number
6 is not a prime number
7 is a prime number
8 is not a prime number
9 is not a prime number
```

- **Guess what's the result?**
- If the ***for loop*** ends **normally**(without break), it will execute statements in else

While Loop

```
while expression:
    #code
```

- same as **C/C++ !**

- Expression doesn’t need to add parentheses just like “**if**” function in **Python**.

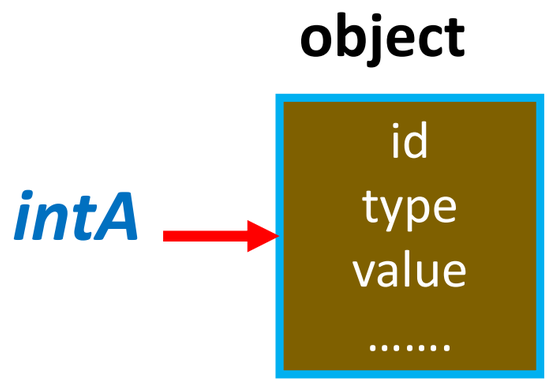
- There’s no “**Do … While**” in **Python**.

3

Python Container

Python built-in data types

- Numeric data
  - int
    - intA =10
  - Float
    - floatA = 3.14
  - Bool
    - boolA = **True**; boolB = **False**
- String/character
  - stringA = ‘Hello World!’
- Container
  - List, tuple, dict, set



Containers (of data)

- In **Python:** there're four main data structures (**containers**):

  "**List**", "**Tuple**", "**Dict**", and "**Set**".

- "**List []**" is like a "**vector** (dynamic array)" in **C++.**
- "**Tuple ()**" is like a **constant array** in **C.**
  - The data in the tuple can't be modified after assignment.
- "**Dict {}**" is like "**map**" in **C++.**
  - Each element contains a "**Key**" and its corresponding "**Value**".
- In **C**: the type of all elements in the array **must be the same.**
- In **Python**: we **can** have **different** types of variables in a list, tuple, and dict.

List []

- Create a list like...

```
A = []
B = [2, 'Hello World', 3.0, True, 'H']
```

- List, tuple or dict can also be the element in a list.

```
C = [ [2, 'Hello World'], (3.0, True), {"Hello": 1, "World": 2} ]
```

- Get the first element in B:

```
D = B[ 1 ]
```

→ **D = 'Hello World'**

- The index starts from **0**, just like in **C/C++.**
- In **Python:** if the index is **less than 0**, it counts **from backward**.

```
E = B[ -1 ]
```

→ **E = 'H'**

- The index can be **A:B**, then it will get the items from A^{th} to $(B-1)^{th}$.

```
F = B[ 2 : 4 ]
```

→ **F = [3.0, True]**

List methods: append() & insert()

- Each container object provides many useful methods for use.
- Use **append**(**A**) to add an item **A at the end** of the original list.
- **Ex**

```
A = [2, 3, 4]
A.append (5)
```

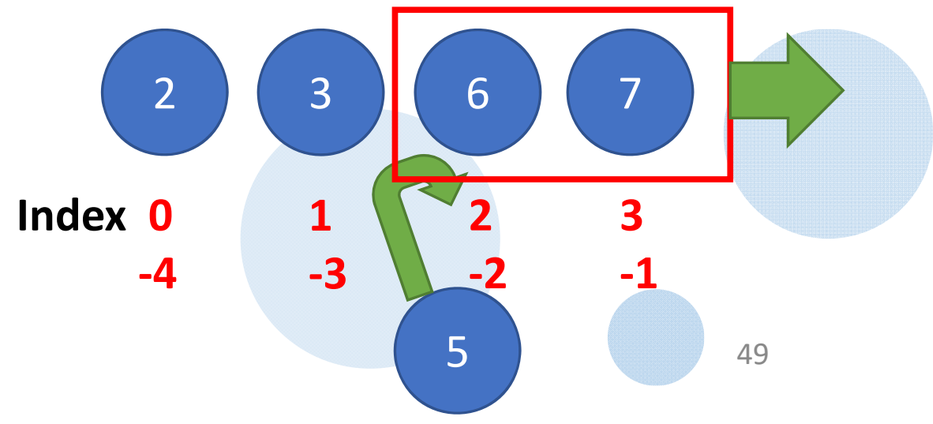
→ **A = [2, 3, 4, 5]**

- Use **insert**(**i**,**A**) to add an item **A** at the position **i**.
- **Ex**

```
A = [2, 3, 6, 7]
A.insert (2, 5)
```

→ **A = [2, 3, 5, 6, 7]**

- **extend(A)** is different from **append(A)**



Questions

If we have **A = [2,3,4,5]**

- **B = A[1:-1]** , **B = ??**

  Ans: **B = [3, 4]** because the last one($(-1)^{th}$) **is not included**!

- Can we use **A.insert(-1,6)** ? If your answer was yes, **A = ??**

  Ans: **Yes**, **A = [2, 3, 4, 6, 5]**

Other useful functions

A = [1, 3, 2, 5, 4, 6, 3]

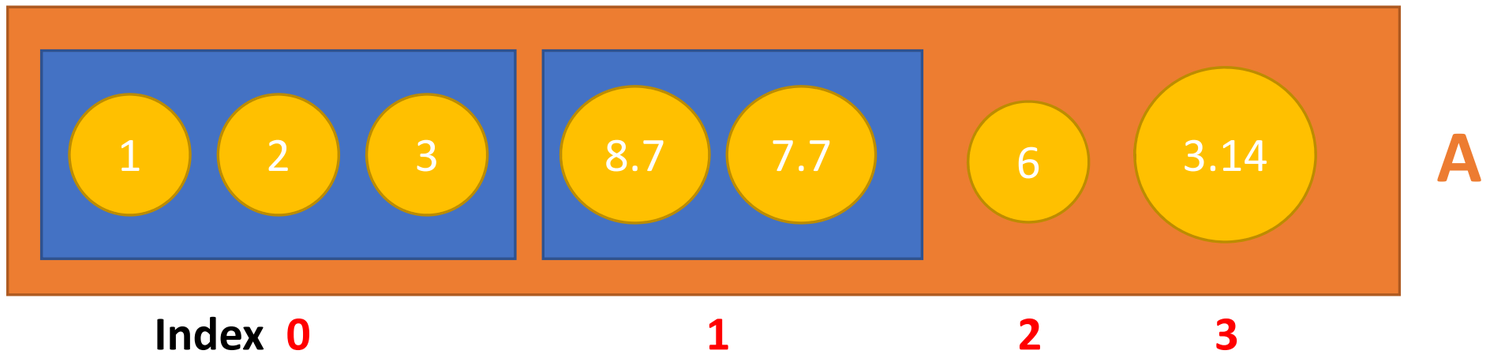
X = A.index(3)	Get the first index of element 3 in A	X = 1
X = A.count(3)	Get the number of elements *3* in A	X = 2
X = A.pop()	Delete and return the last element	X = 3
A.remove(3)	Remove the first element 3 in A.	A = [1, 2, 5, 4, 6, 3]
X = len(A)	Get the count of elements in A	X = 7
A.reverse()	Reverse A	A = [3, 6, 4, 5, 2, 3, 1]
A.sort()	Sort A from the smallest to the largest	A = [1, 2, 3, 3, 4, 5, 6]
del A	Delete A	

Questions

If A = **[**[1,2,3], [8.7, 7.7], 6, 3.14**]**

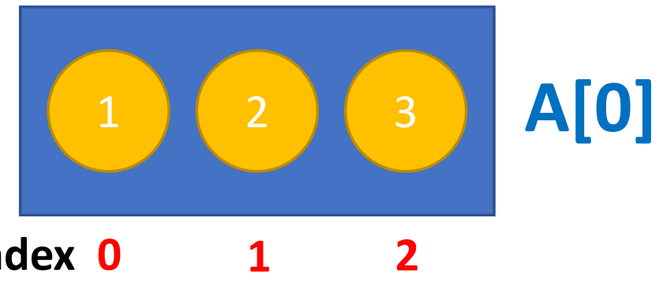
- **X = len(A)**, X = ??

Ans: **X = 4**



- **Y = len(A[0])**, Y = ??

Ans: **Y = 3**



Tuple ()

- Tuple variables (objects) in **Python** cannot be ***modified***.
- **Ex**
  - If we have A = (1,2,3,4), we **can't** use A.**append**(5) or other functions to **add or delete** the original items in a tuple.

- We can convert a tuple to a list.
- **Ex**

```
A = (1, 2, 3, 4)
B = list(A)
B.append(5)
```

→ **B = [1, 2, 3, 4, 5]**

```
C = [1, 2, 3, 4]
D = tuple(C)
D.append(5)
```

→ **Error: Tuple can't be modified!**

Multiple assignments to one variable

- If we assign multiple objects to one variable, it will create a tuple and assign it to that variable.
- **Ex**

A = 1, 2, 3, 4 → **A = (1, 2, 3, 4)**

- **Question:**

  Can we do this?

  Ans: No

A, B= 1, 2, 3, 4

Iterate List and Tuple

- Python can ***iterate*** a list and a tuple
- **Ex:**

```
A = [5, 6, 7, 8]
for i in A:
    print(i)
```

```
5
6
7
8
```

- Or walk through a list by index.
- **Ex:**

```
A = [1, 2, 3, 4]
for i in range(len(A)):
    print(A[i])
```

```
1
2
3
4
```

Multiple Iteration

- You can use double *for loop* to iterate two lists.

```
A = [1, 2, 3, 4]
B = [5, 6]
for x in A:
    for y in B:
        print(x, y)
```

```
1 5
1 6
2 5
2 6
3 5
3 6
4 5
4 6
```

- This can be shortened to one line.

```
A = [1, 2, 3, 4]
B = [5, 6]
for x, y in [(x,y) for x in A for y in B]:
    print(x, y)
```

zip()

zip(*iterables)

- zip() can pair input objects one by one, and return **tuple** data.
- **Ex**

```
A = [1, 2, 3, 4]
B = [5, 6, 7, 8]
for i in zip(A,B):
    print(i)
```

```
(1, 5)
(2, 6)
(3, 7)
(4, 8)
```

- If the length of two objects **aren't the same**, it will only combine part of the inputs according to the **minimal length** of the objects.

```
A = [1, 2, 3, 4]
B = [5, 6]
for i in zip(A, B):
    print(i)
```

```
(1, 5)
(2, 6)
```

enumerate()

```
enumerate(sequence, [start = 0])
```

- **enumerate()** can add a counter to an ***iterable*** object just like **enum** in **C/C++.**
- **Ex**

```
A = [3, 7, 1, 8]
for i, x in enumerate(A):
    print(i, x)
```

```
0 3
1 7
2 1
3 8
```

- Add a starting number to enumerate() function.
- **Ex**

```
A = [3, 7, 1, 8]
for i, x in enumerate(A, 10):
    print(i, x)
```

```
10 3
11 7
12 1
13 8
```

Dict {}

- **Dict** (**Dictionary**) in **Python**:

```
{Key1: value1, Key2: value2… }
```

  - **Ex**

```
A = {"Pencil": 15, "Paper": 10, "Glue": 20}
print( A["Pencil"] )
```

→ 15

- **Keys** in dict **must be unique**. If the ***Key*** name is duplicated, the **new one** will **replace** the **old one**.

```
A = {"Pencil": 15, "Paper": 10, "Glue": 20, "Pencil": 50}
print( A["Pencil"] )
```

→ 50

- We can’t index a value in dict .
  - **Ex**

```
print( A[0] )
```

→ Error

Dict {}

- How to **add** a new element or update the value in the dict?
  - **Ex**

```
A = {"Pencil": 15, "Paper": 10, "Glue": 20}
```

```
A["Ruler"] = 60
```

→

```
A = {"Pencil": 15, "Paper": 10, "Glue": 20, "Ruler": 60}
```

```
A["Glue"] = 10
```

→

```
A = {"Pencil": 15, "Paper": 10, "Glue": 10, "Ruler": 60}
```

- How to **delete**/**clear** the element?
  - **Ex**

```
del A["Pencil"]
```

→

```
A = {"Paper": 10, "Glue": 10, "Ruler": 60}
```

```
A.clear()
```

→

```
A = {}
```

Other useful functions

```
A = {"Pencil": 15, "Paper": 10 }
```

X = A.**items**()	Get all the ***(Key-Value)*** items in A	X = [("Pencil": 15), ("Paper": 10)]
X = A.**keys**()	Get all the ***Key***s in A	X = ["Pencil", "Paper"]
X = A.**values**()	Get all the ***Value***s in A	X = [15, 10]
X = "Pencil" **in** A	Check if “Pencil” is in A	X = True

dict.**get**(**Key**, [**return_value**])	if key is not found, it will return ***return_value*** (None in default), and **won’t** add this element to the dict	Y = A.get("Pencil") #Y=15
		X = A.get("Pencil", 30) #X=15
		Z = A.get("Ruler")
		#Z = None, and A is not changed
		R = A.get("Ruler", 30)
		#R = 30, and A is not changed

dict.**setdefault**(**Key**, [**return_value**])	if key is not found, it will return ***return_value*** (None in default), and **will** add this new element to the dict	S = A.setdefault("Ruler", 30) #S = 30, and A={"Pencil": 15, "Paper": 10, "Ruler"=30}

Set (集合)

- **Set** in Python is similar to the concept of set in mathematics.
- Three ways to create a set object:

```
s1=set() #create an empty set
s2=set('APPLE')
s3={'A','P','P','L','E’} #create a set object with {}
print(s1, s2, s3, sep = '\n')
```

```
set()
{'A', 'L', 'P', 'E'}
{'A', 'L', 'P', 'E'}
```

- Although there are two **'P'** when creating s2 and s3, it will only retain one **'P'** since they’re “set”.

Add/Remove elements of a set

- Add/remove elements by using

setname.add(something)
setname.update(multiplethings)
setname.remove(something)

- Ex

```
s = {'A','P','P','L','E'}
print(s)
s.add('P')
print(s)
#s.add('P','Q')
s.update('P','Q')
print(s)
s.remove('P')
print(s)
#s.remove('P')
```

```
{'A', 'L', 'P', 'E'}
{'A', 'L', 'P', 'E’}
#TypeError!!
{'L', 'P', 'E', 'Q', 'A'}
{'L', 'E', 'Q', 'A’}
#KeyError!!
```

Set Comparison

- Compare two sets by using comparison operators.
- Guess what the answer is?

```
s1=set()
s2=set('APPLE')
s3=set('APDLE')
s4=set('APBLE')
s5={'A','P','L','E'}
print(s2==s5)
print(s2!=s5)
print(s2>s3)
print(s3!=s4)
print(s3<s4)
print(s3>s4)
```



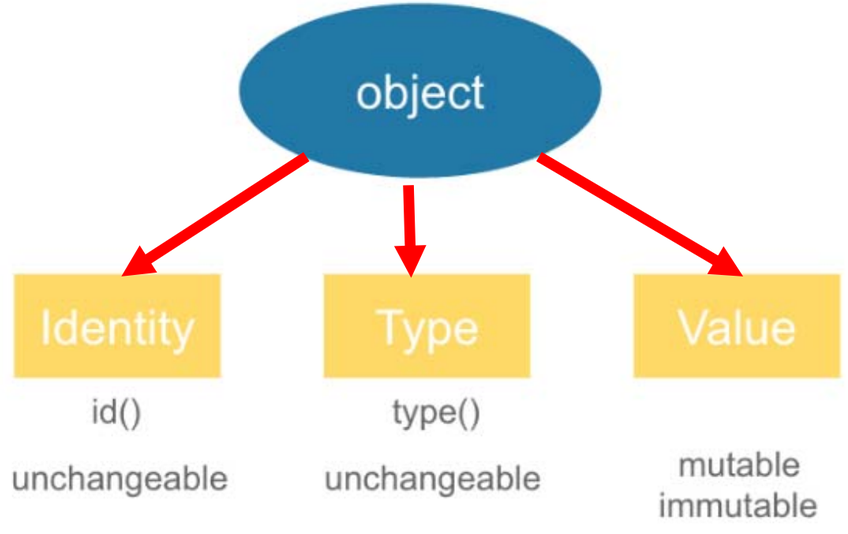
```
True
False
False
True
False
False
```

Other useful functions

```
A = {"Car", Airplane", "Truck"}
B = {"Car", "Airplane"}
C = {"Car", "Truck"}
```

element in set1	Check whether ***element*** is in set1	print("Car" in A) #True print(B in A) #False, because there isn't any set in A.
set1.issubset(set2)	Check whether set1 is the subset of set2	print(A.issubset(B)) #False print(B.issubset(A)) #True
set1.issuperset(set2)	Check whether set1 is the superset of set2 *Superset: All elements in set2 are also in set1*	print(A.issuperset(B)) #True print(B.issuperset(A)) #False
set1.union(set2) set1 | set2		print(B | C) #{'Truck', 'Car', 'Airplane'}
set1.intersection(set2) set1 & set2		print(B & C) #{'Car'}
set1.difference(set2) set1 - set2		print(B - C) #{'Airplane'}
set1.symmetric_difference(set2) set1 ^ set2	Symmetric difference: Find the elements which only exist in one of the sets (set1 or set2).	print(B ^ C) #{'Truck', 'Airplane'}

Mutable/Immutable



- In Python, every object contains one ***identity(address)***, ***type***, and ***value***. Some objects' value can't be changed.

- **Immutable objects**
  - **Numeric types**: int, float, complex
  - **string**
  - **tuple**
  - **frozen set**

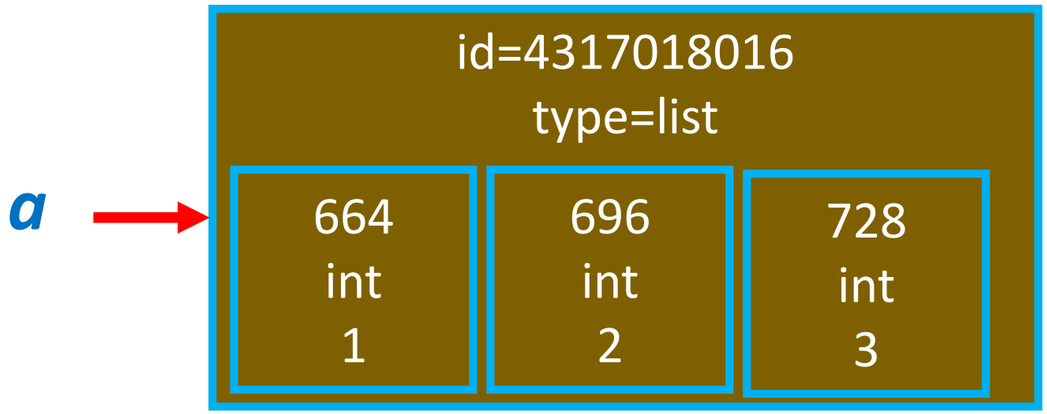
- **Mutable objects**
  - **list**
  - **dict**
  - **set**
  - **byte array**

Mutable/Immutable

Case1

```
a = [1, 2, 3]
print( id(a) )

a[1] = 10
print( id(a) )
print( a )
```

```
4317018016
4317018016
[1, 10, 3]
```



- It's easy to find that **lists** are **mutable**. After changing the value, its **id** doesn't change, which means we **can** change the value of the list objects.

Mutable/Immutable

- **Case2**

```
a = 1
print( id(a) )

a = 2
print( id(a) )
```

```
140207626202088
140207626202064
```

- **In Python**, the operator **=** is not the same as **=** in **C/C++**.
- For example, `a = 1` **creates an object with a value of 1**, and **a points to that object**.
  - So both objects'(1 and 2) value **won't** be changed!

Mutable/Immutable

◈**Case3**

```
a = (1, 2, 3)
print( id(a) )

a[2] = 1
```

```
4317018016

# error with a[2] = 1
```

```
a = (1, 2, [3])
print( id(a) )
a[2][0] = 100
print( id(a) )
print( a )
```

```
4317018016
4317018016
(1, 2, [100])
```

◈An immutable object means “their elements can’t be changed ”.

◈BUT if the element itself is mutable, it can be changed

Pitfall

- **Guess what the value of B is.**

```
A = [1, 2, 3]
B = A
A[1] = 5
```

B = [1, 5, 3]

- As we mentioned earlier, in Python `B = A` let variable B refer to the same object which A also refers to.
- After modifying the objet A refers to, *the value of B* is changed too.

Pitfall

A = [1, 2, 3]

A → [1, 2, 3]

id(A)= 2607197599624

id(A[0])= 1461152832
id(A[1])= 1461152864
id(A[2])= 1461152896

B = A

A → [1, 2, 3]

B → [1, 2, 3]

id(A)= 2607197599624
id(B)= 2607197599624

A[1] = 5

A → [1, 5, 3]

B → [1, 5, 3]

id(A[0])= 1461152832
id(A[1])= 1461152960
id(A[2])= 1461152896

Pitfall

- **Then how about this? What will B become?**

```
A = [1, 2, 3]
B = A
A = [4, 5, 6]
```

B = **[1, 2, 3]**

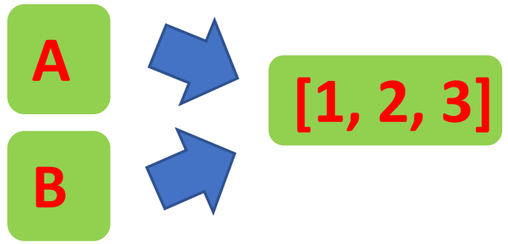
- "A = [4, 5, 6]" lets A point to a new object [4, 5, 6].
- The original object [1,2,3], which B points to, is still there.

Pitfall

```
A = [1, 2, 3]
```

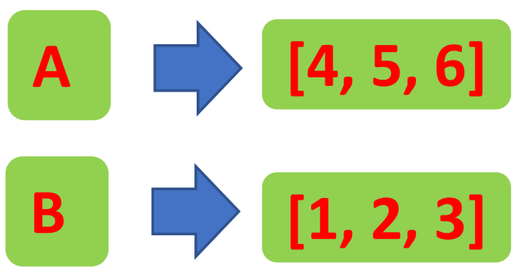
id(A)= 2607197598792

```
B = A
```



id(A)= 2607197598792
id(B)= 2607197598792

```
A = [4, 5, 6]
```



id(A)= 2607197598856

id(B)= 2607197598792

List: copy()

- What can we do if we don't want B to be changed after we modify the elements in A?

➢1. **B = list(A)**

➢2. **B = A [:]**

➢3. **B = [x for x in A]**

➢4. **B = A.copy()**

```
A = [1, 2, 3]
B = list(A)
C = A[:]
D = [x for x in A]
E = A.copy()

A[1] = 5
```

A= **[1,5,3]**
B = C = D = E = **[1, 2, 3]**
ids of A, B, C, D, E are all different

4

Function & Class

Function

```
def function_name (parameter1, parameter2,…):
        #code
        [return variables1, variables2,… ]
```

- In **Python:** function declaration doesn't necessarily have to **return an object**. Parameters are also optional.
- In **Python:** functions can return **multiple variables**.
- **Ex**

```
def find_max(a, b):
    return a if a > b else b
```

```
def division (a, b):
    return a//b, a%b #quotient and remainder
```

- In **Python:** functions are also objects.

- *Whether an object (ex:find-max) is callable can be tested by using **callable(object)**.*

Function parameters with default values

```
def function_name (parameter1, …, parameterx = value1, … = valuen):
        #code
        [return variables1, variables2,… ]
```

- As **C/C++**,we can assign default values to the parameters
- Remember putting the parameters with default values to the right.
- **Ex**

```
def printScore (name, engScore = 60, mathScore = 60):
        print(name, "'s English score was: ", engScore, "  Math score was: ", mathScore)

printScore("A", 60, 60)
printScore("A", 60)
printScore(name = "A", mathScore = 60)
```

```
A 's English score was:  60   Math score was:  60
A 's English score was:  60   Math score was:  60
A 's English score was:  60   Math score was:  60
```

Function argument passing in Python

- In Python, arguments are passed by sharing (call by ***object reference***)
- A new variable will be created inside the function which refers to the passed data object.
  - What's the value of listV after calling the function?

```
def test1(listA):
    print(id(listA))
    listA.append(1)
    listA = [2]
    print(id(listA))

listV=[]
test1(listV)
print(id(listV), '\n', listV)
```

```
2725265682568
2725265672328
2725265682568
[1]
```

More argument passing examples

```
def test2(listA):
    print(id(listA))
    listA[0] = [2]
    print(id(listA))


listV=[1]
test2(listV)
print(id(listV), '\n', listV)
```

```
2725265673288
2725265673288
2725265673288
 [2]
```

- *listA and listV still refer to the same data object.*

```
def test3(varA):
    print(id(varA))
    varA = varA +1
    print(id(varA))


varV=1
test3(varV)
print(id(varV), '\n', varV)
```

```
140731748163984
140731748164016
140731748163984
 1
```

- *varA and varV refer to different data objects after value assignment inside the function.*

Function return

- In Python, function will return the reference of the returned object.

```
def test4(listA):
    listA[0]=2
    return listA

listV=[1]
print('listV:', id(listV), '\n', listV)
listU=test4(listV)
print('listV:',id(listV), '\n', listV)
print('listU:',id(listU), '\n', listU)
```

```
listV: 2725266274888
 [1]
listV: 2725266274888
 [2]
listU: 2725266274888
 [2]
```

Function with unknown amount of parameters

```
def function_name (*parameter):
        #code
        [return variables1, variables2,... ]
```

- Add * in front of the parameter if we are not sure how many arguments will we receive.
  - Python actually creates a **tuple** for *parameter.

- Ex

```
def add (*number):
        total = 0
        for i in number:
            total += i
        return total
A = add(1)
B = add(1, 2)
C = add(1, 2, 3, 4, 5)
```

```
A = 1
B = 3
C = 15
```

Magic asterisks (*) in Python

- Put an asterisk as a prefix for a variable so it can pack all these elements that are not explicitly denoted by the specified variables.

```
list_a = ['a', 'b', 'c', 'd']
v1, *v2, v3 = list_a
print(v1, v2, v3, sep='|')
```

```
a|['b', 'c']|d
```

- Can pass arguments with unspecified length as a tuple to a function by declaring *def func*(*args)
  - *func('a','b','c')*
- Can pass a dictionary to a function by declaring *def func*(**kwargs)

```
def show(**element):
    for key in element:
        print(key)
show(dog=1, elephant=2, cat=3)
```

```
dog
elephant
cat
```

lambda function

- There are multiple ways to define a function.

```
def plus1 (x):
        return x+1
f1 = plus1

f2 = lambda x: x+1

def plusn (n):
        return lambda x: x+n

f3 = plusn(1)
```

```
print(plus1(10))
print(f1(10))
print(f2(10))
print(f3(10))
```

```
11
11
11
11
```

Python Scope

- Local Scope：A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.
  - Function Inside Function: the variable used inside a function will not be available outside the function, but available for any function inside the function:
- Global Scope: A variable created in the main body of the Python code is a global variable and belongs to the global scope.
  - Global variables are available from within any scope, global and local.
  - If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.
  - The global keyword makes the variable global.
- If the variable name used inside the function is the same as the variable used outside the function, they will be treated as different variable objects.

```
var=10
print(id(var))

def scope_test(var):
    print(id(var))

def scope_test2():
    print(id(var))

scope_test(30)
scope_test(40)
scope_test2()
print(id(var))
```

```
140728131822256
140728131822896
140728131823216
140728131822256
140728131822256
```

Some useful built-in functions

		Example	Answer
abs(x)	Return the absolute value of a number	X= abs(-2)	X= 2
int (x) float(x)	Return an integer/floating point number constructed from a number or string *x*.	X= float(“3.14159\n“)	X= 3.14159
hex(x) oct(x)	Convert x to the hexadecimal/octal string	X= hex(255)	X= "0x22"
round(x, [*n]*)	Return a *number* rounded to *ndigits* precision after the decimal point.	X= round(3.14159)	X= 3
sorted(list1, [reverse=False])	Sort list1 from the smallest to the biggest. If reverse=True, it will reverse the answer	X= sorted([4,2,10,8,6])	X= [2, 4, 6, 8, 10]
type (x)	Get the data type of x	X=type(3.14159)	X= <class 'float'>

map() ‘s usage

- Can be used to avoid the use of loop such that the computation can be accelerated.
- ***map()*** itself will return *a map object* which has to be converted to a list.

```
def num_func(x):
    return x**2/2

data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list(map(num_func, data))
```

```
[0.5, 2.0, 4.5, 8.0, 12.5, 18.0, 24.5, 32.0, 40.5, 50.0]
```

```
[num_func(x) for x in data]
```

filter() ‘s usage

- Can also avoid the use of loop such that the computation can be accelerated.
- ***filter()*** itself will return *an filter object* which has to be converted to a list.

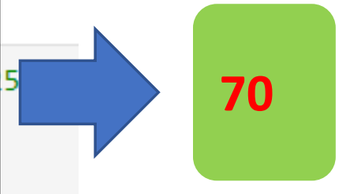
```
def more_than_10(x):
    return x > 15

data = [3, 17, 32, 12, 54, 3, 2, 1]
list(filter(more_than_10, data))
```

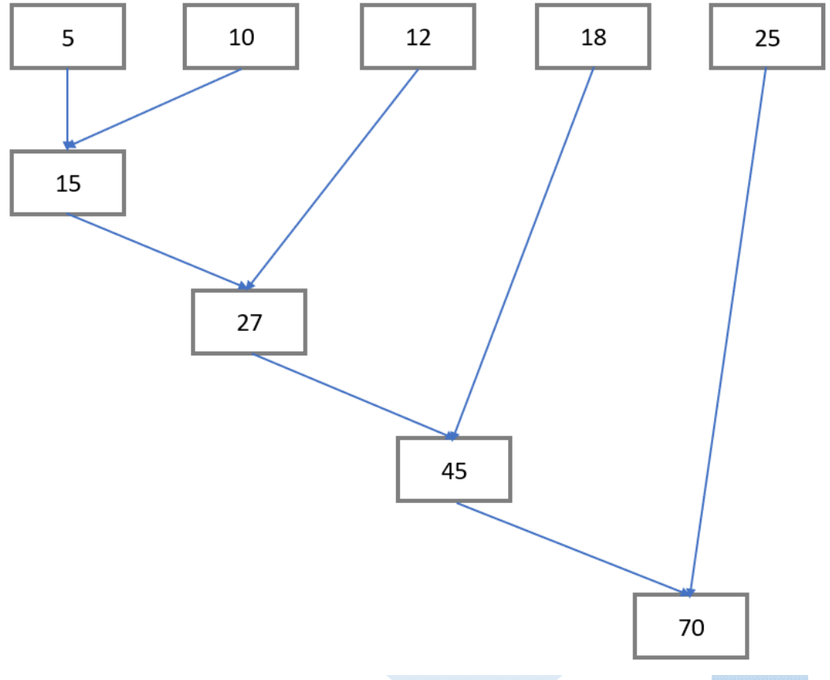
[17, 32, 54]

reduce() ‘s usage

- The list will be iteratively processed can be reduced into a single value.
- It is not set by default; instead, it has to be imported.

```
from functools import reduce

def add_nums(a, b):
    return a + b

data = [5, 10, 12, 18, 25]
reduce(add_nums, data)
```





Class

- **Python** supports ***class,*** just like **C++.**

- Can declare some variables/members or functions/methods as its ***attributes***.

- Every function in an class must add “**self**” as **the first parameter**.

- Using class variables must add “**self.**” in front of the variable name.

- Put “**__**”(**double underscore**) in front of the **Private** variables.

Define the class

- class A():
  - Or simply class A
  - A is the name of this class.

```
class A():
    def __init__(self):
        self.a = 0
        self.b = 10

    def set_a(self, a):
        self.a = a

    def set_b(self, b):
        self.b = b

    def add_a(self, x):
        self.a = self.a + x
        return self.a

    def print(self):
        print("self.a = ", self.a, "\nself.b = ", self.b)
```

Constructor: __init__

- When the class A object is built, the construction function ***__init__*** is invoked to do the initialization job.

- The class members self.a can be accessed by the member functions of the same class.

```
class A():

    def __init__(self):
        self.a = 0
        self.b = 10

    def set_a(self, a):
        self.a = a

    def add_a(self, x):
        self.a = self.a + x
        return self.a

    def print(self):
        print("self.a = ", self.a)
```

Definition of member functions

- Provide some methods to manipulate the object.
- Remember to put ***self*** when defining the member functions
  - Ex: def print(self)
- The calling method of the member functions:
  - class_a = A()
  - class_a.print()
- *A custom class (ex:A) is also callable.*

```
class A():
    def __init__(self):
        self.a = 0
        self.b = 10

    def set_a(self, a):
        self.a = a

    def add_a(self, x):
        self.a = self.a + x
        return self.a

    def print(self):
        print("self.a = ", self.a, "\nself.b = ", self.b)
```

Class example

Build the class

Use the class

```
class A():
#Constructor definition
    def __init__(self):
        self.a = 0

    def print(self):
        print("self.a = ", self.a)

    def set_a(self, a):
        self.a = a

    def add_a(self, x):
        self.a = self.a + x
        return self.a
```

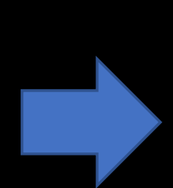
```
if __name__ == '__main__':
    print("class_a = A()")
    class_a = A()
    class_a.print()

    print("\nclass_a.set_a(100)")
    class_a.set_a(100)
    class_a.print()

    print("\nclass_a.add_a(20)")
    class_a.add_a(20)
    class_a.print()
```

Call the print() member function

Class

- **Ex**

```
class Shape():  # () is not required when there is no class inheritance
    def __init__ (self, width, length): # class initialization
        self.__width = width # private variable width
        self.__length = length # private variable length

    def area (self):
        return self.__width * self.__length

A = Shape(20,20) # create a class shape named "A"
print(A.area() )
```



400

Class inheritance

- **In Python**: To inherit the **father's** class by putting the name **in the bracket** behind **child's** name.

```
class DerivedClassName(Base1, Base2, Base3):
        #code
```

- **Python** supports multiple class inheritance just like **C++**!
- ***super(DerivedClassName).methodName(args)*** can be used to call the parent class method.

Class inheritance

```
class Shape():
    def __init__ (self, width, length):
        self.__width = width
        self.__length = length

    def area (self):
        return self.__width * self.__length
```

Inheriting class ”Shape”

```
class Circle(Shape):
    def __init__ (self, radius):
        self.__radius = radius

    def area (self):
        return (pi**2)*self. __radius
```

```
A = Circle (5)
print(A.area() )
```

49.34802200544679

Object class

- All python's classes inherit from **object class**.
- Some inherited methods can be overwritten.
  - ***__init__, __str__, __eq__***
  - *Can run dir(class_name) to check the default methods.*

```
class A:
    def __init__(self, i=0):
        self.i=i

    def __str__(self):
        print("This is class A")
        return 'class A '+str(self.i)
```

```
X = A(9)
print(X)
Z = str(X) + "+Z"
print(Z)
```

```
This is class A
class A 9
This is class A
class A 9 +Z
```

__call__() method

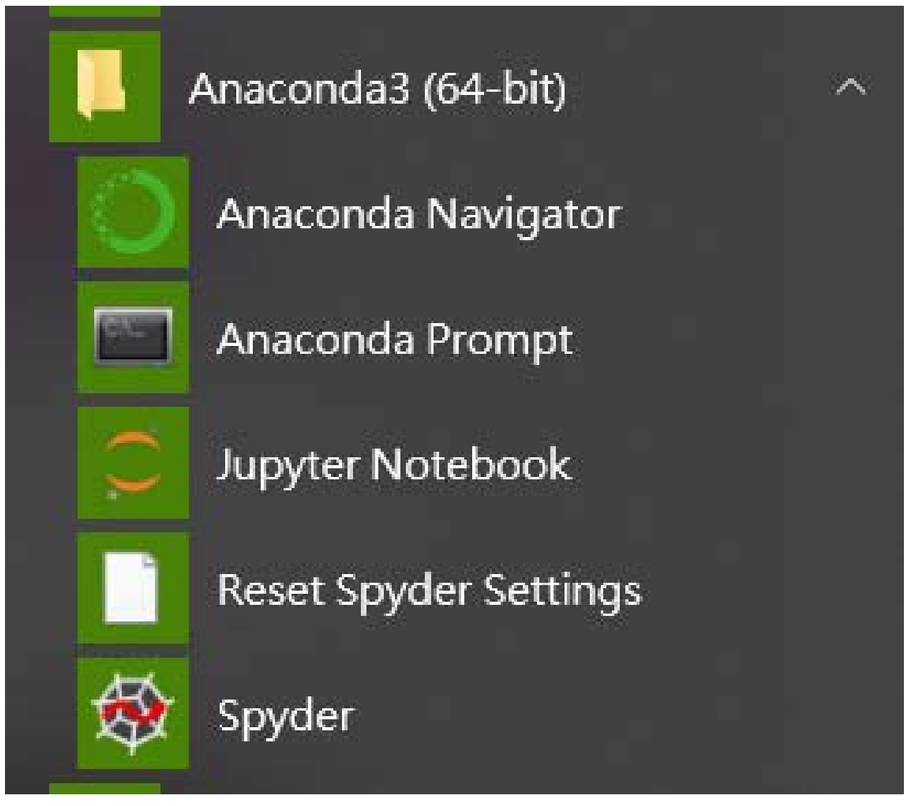
- The ***__call__()*** method enables Python programmers to write classes where the instances behave like functions.
- It can make the object ***callable***.

```
class fun():

    def hello(self):
        print("Hello ！ ")

    def __call__(self, name):
        print("Hello!", name)
```

```
f = fun()
f.hello()
f('John')
```

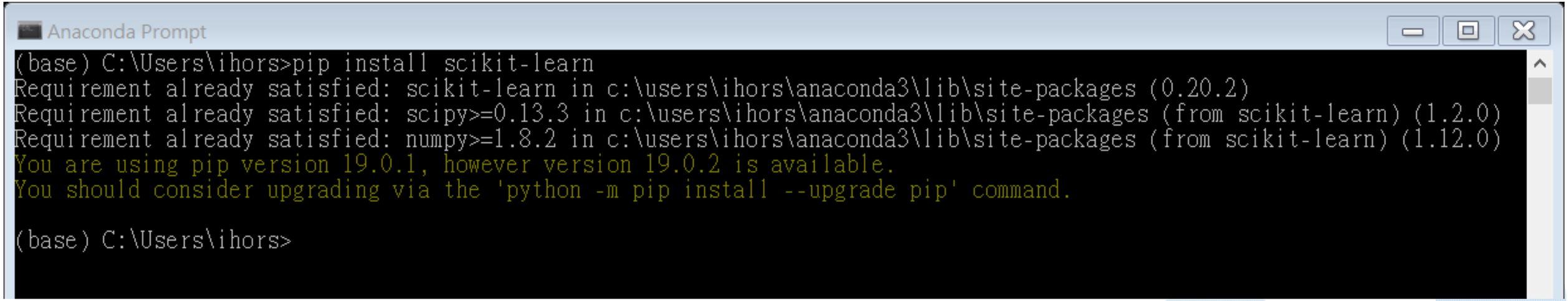
```
Hello ！
Hello!
John
```

Install Package – Open Anaconda prompt



Install packages

- pip for python2.x
- pip3 for python3.x



Module import

File: A.py

```
class Class_A:
    def print(self):
        print("class A")

    def print_hello(self):
        print("class A say hello")
```

```
import A
```

```
a = A.Class_A()
a.print()
```

```
import A as AAAAA
```

```
a = AAAAA.Class_A()
a.print()
```

```
from A import Class_A
```

```
a = Class_A()
a.print()
```

Import

- `import module_name` to import module. Just like *#include* in C
  - **Ex**

```
import random
print (random.randint(0, 10)) # pick an integer from 0~10
```

- `from module import [name1, name2, ...]` to import certain functions
  - **Ex**

```
from math import pi
print (pi)        #needn't add model's name "math"
```

3.141592653589793…

Import

- If the module name is **too long** we can give it an alias

- **Ex**

```
import pandas as pd
readData = pd.read_csv('NSYSU.csv')
```

Different ways to import NumPy

```
import numpy
```

```
numpy.array([1, 2, 3])
```

```
import numpy as np
```

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

```
from numpy import array
```

```
array([1, 2, 3])
```

dir

- **dir(object)** returns a list of valid attributes(variables, methods…) of the object.

```
print(dir())
#in Spyder IDE
```

→

```
['In', 'Out', '_', '__', …  … , 'exit', 'get_ipython', 'quit']
```

```
class Shape():
    def __init__ (self, width, length):
        self.__width = width
        self.__length = length
    def area (self):
        return self.__width * self.__length

print(dir(Shape(3,4)))
```

→

```
['_Shape__length', '_Shape__width', … …
'__weakref__', 'area']
```

dir

```
import numpy
print(dir(numpy))
```

['ALLOW_THREADS', 'BUFSIZE', 'CLIP', 'ComplexWarning',
'DataSource', 'ERR_CALL', 'ERR_DEFAULT', 'ERR_IGNORE', 'ERR_LOG',
'ERR_PRINT', 'ERR_RAISE', 'ERR_WARN',
'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO', 'FPE_INVALID',
'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf', 'Infinity',
'MAXDIMS', 'MAY_SHARE_BOUNDS', 'MAY_SHARE_EXACT',
'MachAr', 'ModuleDeprecationWarning', 'NAN', 'NINF', 'NZERO',
'NaN', 'PINF', 'PZERO', 'PackageLoader', 'RAISE', 'RankWarning',
'SHIFT_DIVIDEBYZERO', 'SHIFT_INVALID', 'SHIFT_OVERFLOW',
'SHIFT_UNDERFLOW', 'ScalarType', 'Tester', 'TooHardError', 'True_',
'UFUNC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME',
'VisibleDeprecationWarning', 'WRAP', '_NoValue',
'__NUMPY_SETUP__', '__all__', '__builtins__', '__cached__',
'__config__', '__doc__', '__file__', '__git_revision__', '__loader__',
'__name__', '__package__', '__path__', '__spec__', '__version__',
'_distributor_init', '_globals', '_import_tools', '_mat', 'abs',
'absolute', 'absolute_import', 'add', 'add_docstring', 'add_newdoc',
'add_newdoc_ufunc', 'add_newdocs', 'alen', 'all', 'allclose', 'alltrue',
'alterdot', 'amax', 'amin', 'angle', 'any', 'append', 'apply_along_axis',
'apply_over_axes', 'arange', 'arccos', ……]

5

Numerical computing NumPy

Overview

- **NumPy** is a powerful module which has been very frequently used in Data Science.
  - ***import numpy as np***

- The core functionality of NumPy is **“ndarray”**, which is a data structure **like “C array”**.

- NumPy supports **parallel processing**, so it is faster to use “ndarray” than “list” when processing multi-dimensional data.

Creating a numpy array

- Numpy array can be created by converting a list by using

```
np.array([number_array])
```

- The elements of the list should be in **same data type.**
- Ex:
  - a = np.array(1,2,3,4) #**False**
  - a = np.array([1,2,3,4]) #**Correct**
- The type of the array can also be explicitly specified at creation time:

```
c = np.array([ [1,2], [3,4] ], dtype = complex )
print(c)
```

```
c = [[ 1.+0.j  2.+0.j]
     [ 3.+0.j  4.+0.j]]
```

Indexing and Slicing ndarray

- Use **[ ] after the array name** to access the element of array, just like C

```
b
```

```
array([[4, 5, 6],
       [7, 8, 9]])
```

```
b[1]
```

get row1

```
array([7, 8, 9])
```

```
b[1][1]
```

get row1 and then get index1 element in row1

```
8
```

```
b[1,1]
```

get the element at row1, column1

```
8
```

```
b[1,:2]
```

get row1's column0 and column1

```
array([7, 8])
```

```
b[:,2]
```

get the whole row, but only column2

```
array([6, 9])
```

- *We often refer the first dimension as **row** while the second dimension as **column**.*

Operation of numpy arrays

- In the past, if you want to add two different **lists** into a new one, it won't be easy!
- But if you're using **numpy array**, you can simply use **a+b** to get the answer you want.
- Since the operators **+, -, *, /** have been overloaded.

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

c = [7, 9, 11, 13, 15]
d = [6, 14, 24, 36, 50]

3 key attributes of ndarray

- **dtype** : data type of element of ndarray
- **ndim** : dimension of ndarray
- **shape** : shape of ndarray
  - Check the # of elements starting from the outmost bracket.

```
import numpy as np
na = np.array([[[],[]],[[],[]],[[],[]]])
```

na.shape=??

```
b
```

```
array([[4, 5, 6],
       [7, 8, 9]])
```

```
b.dtype
```

data type is int64

```
dtype('int64')
```

```
b.ndim
```

2 dimension array

```
2
```

```
b.shape
```

2 rows, 3 columns

```
(2, 3)
```

Change the shape of a ndarray

- **np.reshape(*ndarray, shape, order*)**
  - **shape: a tuple**
  - **order: 'c' or 'f'**
- **'c'** order (***default***) for (2,2,2) shape
  - (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1) *內層拿起*
- **'f'** order for (2,2,2) shape
  - (0,0,0), (1,0,0), (0,1,0), (1,1,0), (0,0,1), (1,0,1), (0,1,1), (1,1,1) *外層拿起*

a = np.array([[1,2,3,4],[5,6,7,8]])
np.reshape(a, (2,2,-1),'f') ?

```
b
```

```
array([[4, 5, 6],
       [7, 8, 9]])
```

the original shape is 2 row 3 col

```
np.reshape(b, (3,2))
```

```
array([[4, 5],
       [6, 7],
       [8, 9]])
```

changed to 3 col 2 row

```
np.reshape(b, (3,-1))
```

```
array([[4, 5],
       [6, 7],
       [8, 9]])
```

calculate the shape of this dimension automatically if the value is negative

```
np.reshape(b, (-200))
```

```
array([4, 5, 6, 7, 8, 9])
```

- ***Convert a ndarray into one-dimension: np_var.ravel() & np_flattern()***

Playing with ndarray

- **ndarray.astype(*'data_type'*)**

```
b.dtype
```

```
dtype('int64')
```

change the type to float

```
b = b.astype('float')
```

```
b.dtype
```

```
dtype('float64')
```

```
b = np.array([[4,5,6],[7,8,9]], dtype = 'float')
```

or just specify the type while creating

```
b.dtype
```

```
dtype('float64')
```

- *Choose proper data type to prevent wrong result or overflow.*

```
nn=np.array(10)
for i in range(10):
    nn = nn **2
    print(i, nn)
```

```
0 100
1 10000
2 100000000
3 1874919424
4 0
5 0
6 0
7 0
8 0
9 0
```

```
nn=np.array(10).astype(float)
for i in range(10):
    nn = nn **2
    print(i, nn)
```

```
0 100.0
1 10000.0
2 100000000.0
3 1e+16
4 1e+32
5 1.0000000000000002e+64
6 1.0000000000000003e+128
7 1.0000000000000005e+256
8 inf
9 inf
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: RuntimeWarning:
  This is separate from the ipykernel package so we can avoid doing imports until
```

Understand the dimension of ndarray

- **Concept of "axis"**
- It is important to know this for further data processing learning.

```
c = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
```

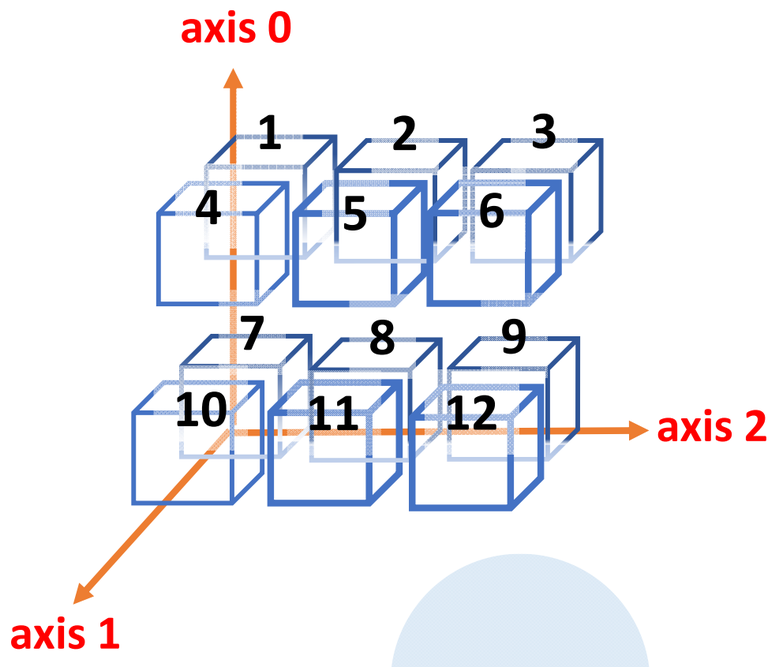
```
c
```

```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

```
c.shape
```

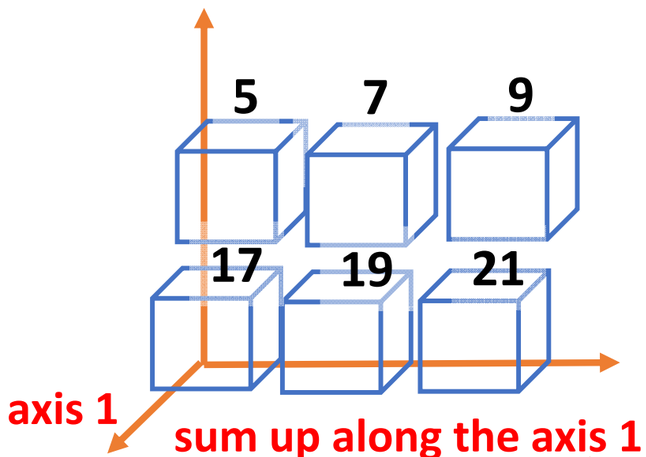
```
(2, 2, 3)
```

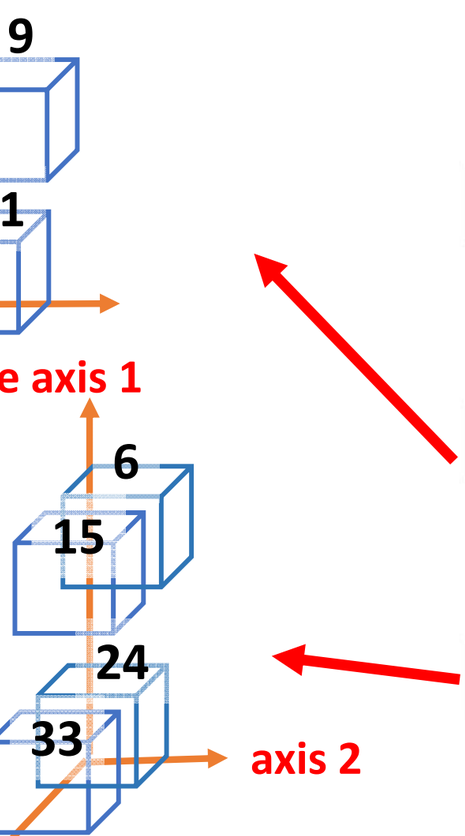
→ **(axis0, axis1, axis2)**



Playing with ndarray

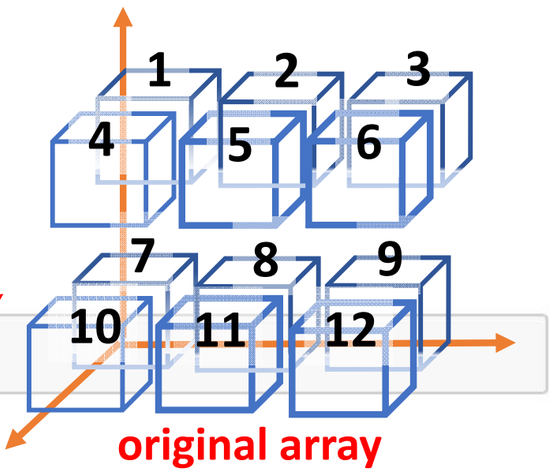
- **ndarray.sum(axis = *value*)**






```
c
```

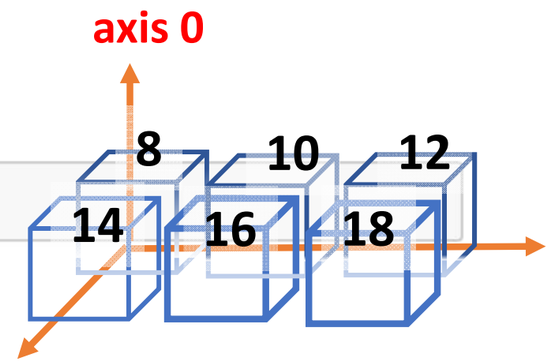
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```




```
c.sum(axis = 0)
```

```
array([[ 8, 10, 12],
       [14, 16, 18]])
```

axis 0




```
c.sum(axis = 1)
```

```
array([[ 5,  7,  9],
       [17, 19, 21]])
```

```
c.sum(axis = 2)
```

```
array([[ 6, 15],
       [24, 33]])
```

Playing with ndarray

- **np.expand_dims(*array, axis*)**
- Expand dimension at a specific axis

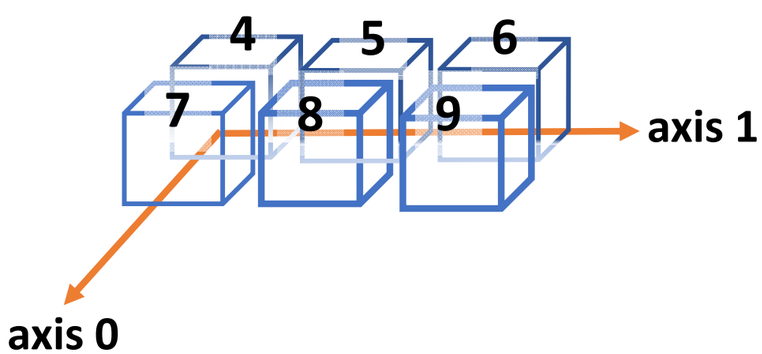
```
b
```

```
array([[4, 5, 6],
       [7, 8, 9]])
```

```
b.shape
```

```
(2, 2)
```

(axis0, axis1)



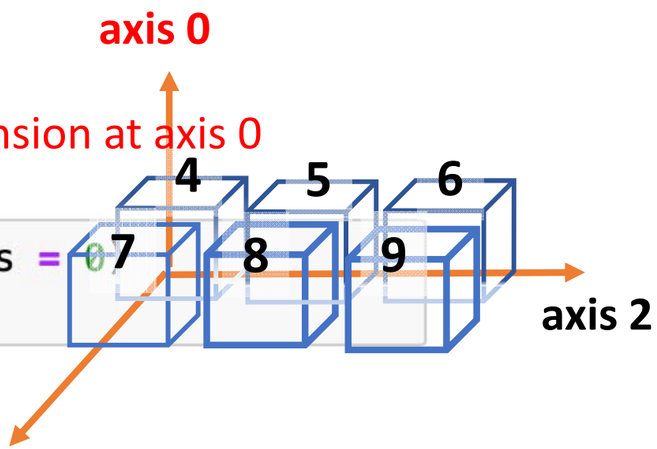
To expand a new dimension at axis 0

```
b_new = np.expand_dims(b, axis = 0)
b_new.shape
```

```
(1, 2, 3)
```

```
b_new
```

```
array([[[4, 5, 6],
        [7, 8, 9]]])
```



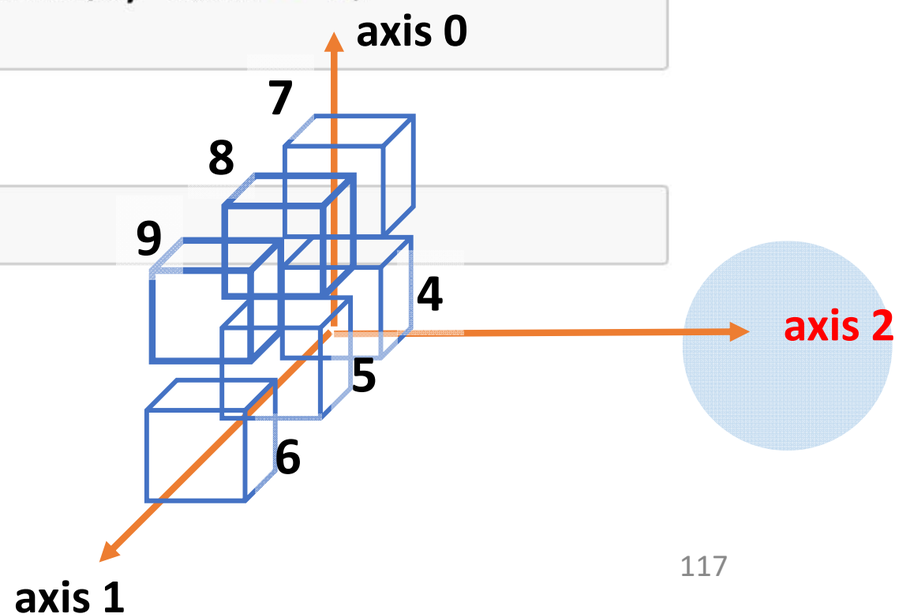
To expand dimension at axis 2

```
b_new = np.expand_dims(b, axis = 2)
b_new.shape
```

```
(2, 3, 1)
```

```
b_new
```

```
array([[[4],
        [5],
        [6]],

       [[7],
        [8],
        [9]]])
```



Playing with ndarray

- **np.newaxis**
- Also used for expanding dimension

```
np.newaxis == None
```

```
True
```

np.newaxis is exactly the same with the use of None

```
b = np.array([[4,5,6],[7,8,9]])
```

```
b
```

```
array([[4, 5, 6],
       [7, 8, 9]])
```

```
b_new = b[:, np.newaxis, :]
b_new.shape
```

```
(2, 1, 3)
```

To expand a new axis at axis 1

```
b_new
```

```
array([[[4, 5, 6]],
       [[7, 8, 9]]])
```

```
b[0]
```

slice row 0

```
array([4, 5, 6])
```

```
b[0][np.newaxis,:]
```

To expand a new axis at axis 0

```
array([[4, 5, 6]])
```

```
b[0][:, np.newaxis]
```

To expand a new axis at axis 1

```
array([[4],
       [5],
       [6]])
```

Mask

- Use Boolean array to mask the ndarray

```
c
```

```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

```
mask = (c%2 == 0)
mask
```

Create a mask targeted for even-number elements

```
array([[[False,  True, False],
        [ True, False,  True]],

       [[False,  True, False],
        [ True, False,  True]]])
```

```
c[mask]
```

get even elements

```
array([ 2,  4,  6,  8, 10, 12])
```

Can be coded in a more compact form.

```
c[ (c>5) & (c<10) ]
```

```
array([6, 7, 8, 9])
```

Other useful NumPy functions

- **np.zeros(*(shape)*)**
- **np.ones(*(shape)*)**
- **np.random.rand(*shape*)**
- **np.random.randn(*shape*)**
- ***np.append(arr, values, axis=None)***

```
np.zeros((4,2))
```

```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

```
np.ones((5,))
```

```
array([1., 1., 1., 1., 1.])
```

```
np.random.rand(2,3)
```

generate random numbers in the interval [0, 1)

```
array([[0.40924054, 0.10378049, 0.22750873],
       [0.78833649, 0.99529632, 0.98145778]])
```

```
np.random.randn(3,2)
```

generate normal distribution random numbers

```
array([[-0.88193913,  1.77671569],
       [ 1.1634023 ,  0.18809235],
       [ 0.24666233, -0.35417192]])
```

Some useful functions

```
import numpy as np
X = np.array ( [1, 2, 3, 4], [5, 6, 7, 8] )
```

		Example	Answer
np.zeros() np.ones()	Create an array filled with zero/one with type 'float'	X= np.zeros((3,4), dtype='int')	[[0. 0. 0. 0.] [0. 0. 0. 0.] [0. 0. 0. 0.]]
np.full((Size), Num)	Create an array filled with ***Num***.	X= np.full((2,2), 5)	[[5 5] [5 5]]
np.arange([A],B,[C])	Similar to range(A,B,[C]) function. C=1 in default	X= np.arange(10, 30, 5)	[10 15 20 25]
np.linspace(A,B,C)	C number from A to B	X= np.linspace(0, 1, 5)	[0. 0.25 0.5 0.75 1.]
ndarray. reshape( )	Reshape the numpy array	Y= X.reshape(2,2,2)	[[[1 2] [3 4]] [[5 6] [7 8]]]
ndarray.astype( )	Type casting	Y= X.astype('float')	[[1. 2. 3. 4.] [5. 6. 7. 8.]]

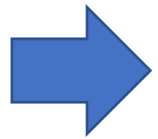
- *Some functions return a **copy** while some returns a **view** of array.*

Tensor 張量

- A tensor of ***rank 1*** is a vector, which is a one-dimensional array such as ***[a,b,c]*** .
  - 一維陣列 (# of dimension)
  - 三維向量 (size of vectors)
- A tensor of rank 2 is a vector of vectors, or a matrix, or a two-dimensional array, ***[[a,b],[c,d]]***.
  - $rank - i \equiv i - dimension$
- A tensor of rank 3 is a vector of vectors of vectors, so something with three nestings, ***[[[a,b],[c,d]],[[e,f],[g,h]]]*** .
- The "***dimension***" here is the (tensorial) rank, or the number of inputs you need to locate an entry.

Tensorflow

```
import tensorflow as tf

t1 = tf.constant([1,2,3])
t2 = tf.constant([4,5,6])

t1+t2
```



```
<tf.Tensor: id=3, shape=(3,), dtype=int32, numpy=array([5, 7, 9])>
```

```
lin_tensor = tf.linspace(5., 9., 5)
print(lin_tensor )
```

```
<tf.Tensor([5. 6. 7. 8. 9.], shape=(5,), dtype=float32)
```

```
one_tensor = tf.ones([2, 3])
print(one_tensor )
```

```
tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]], shape=(2, 3), dtype=float32)
```

6

Python Plotting Matplotlib & Seaborn

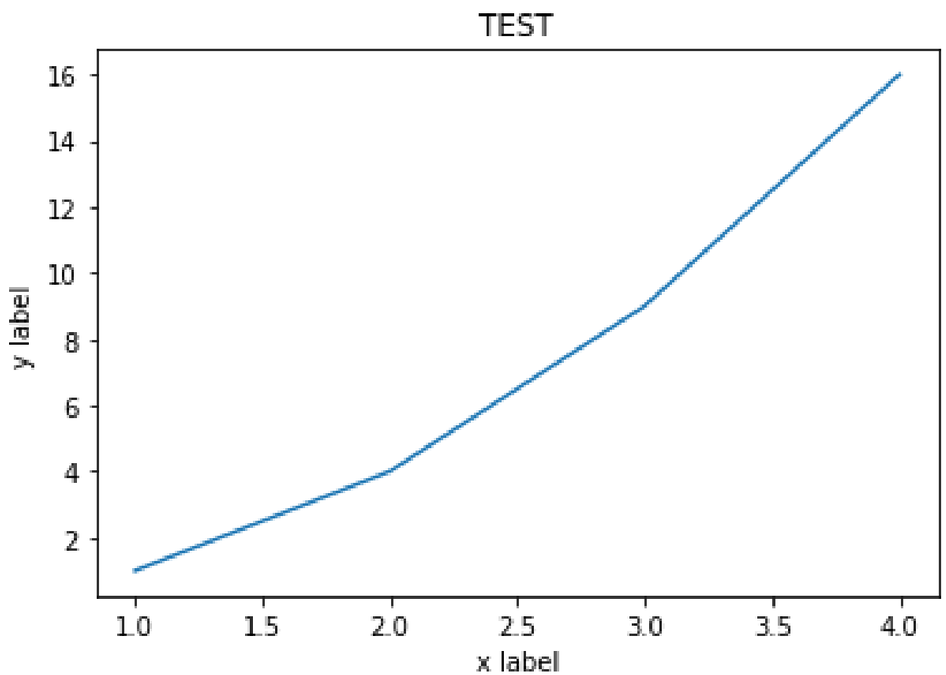
Overview

- **Matplotlib, Seaborn** are powerful and popular modules for plotting!
- Matplotlib is the basic one, while Seaborn can create more beautiful and detailed plots.

- Recommend using `import matplotlib.pyplot as plt` `import seaborn as sns`
- In Jupyter IDE, if you can add `%matplotlib inline` , you wouldn't need to type `plt.show()` to show the plot result every time you create a new plot.

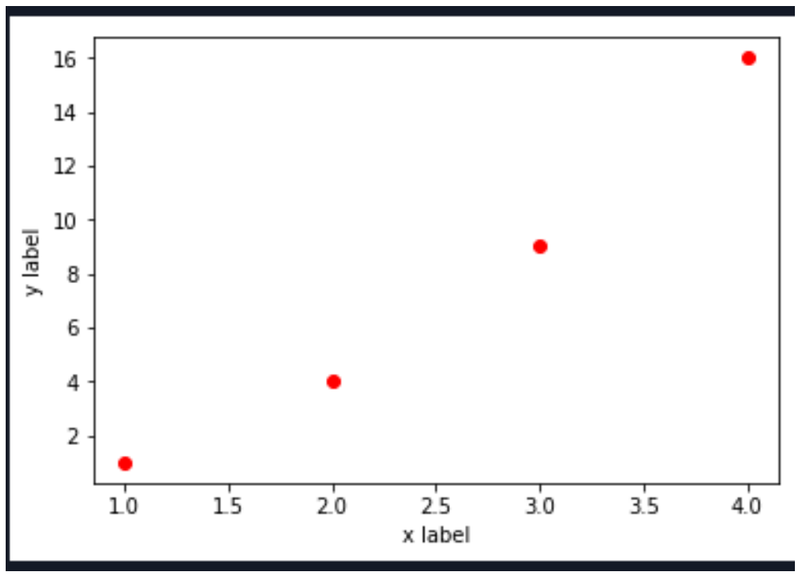
Line Chart

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.ylabel('y label')
plt.xlabel('x label')
plt.title('TEST')
#plt.show()
```



Dot Chart

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.ylabel('y label')
plt.xlabel('x label')
#plt.show()
```

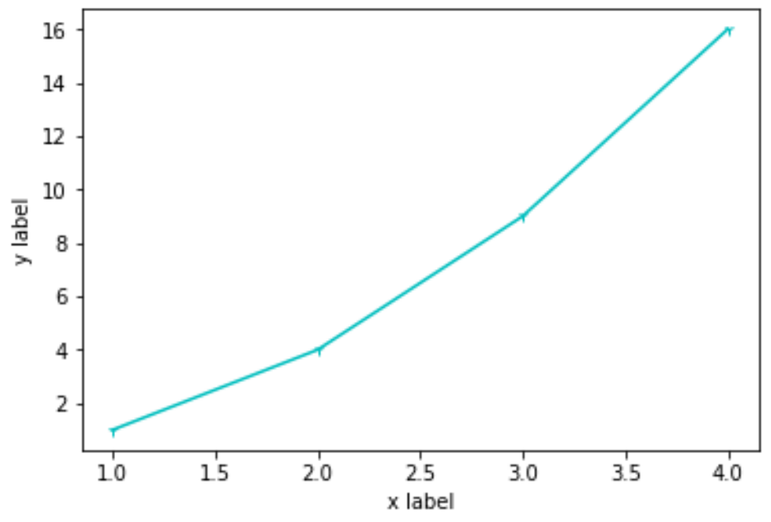


color and markers

means ‘red’ and ‘o’

Line Chart with dot

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], marker = '1' , color = 'c')
plt.ylabel('y label')
plt.xlabel('x labe')
#plt.show()
```



Supported colors and markers

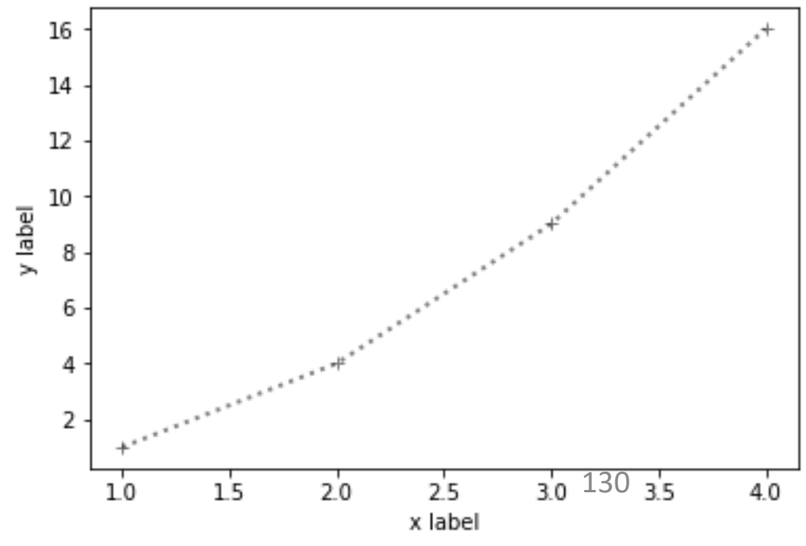
'b'	blue	'm'	magenta
'g'	green	'y'	yellow
'r'	red	'k'	black
'c'	cyan	'w'	white

'.'	point marker	'1'	tri_down marker	'h'	hexagon1 marker	'_'	hline marker
','	pixel marker	'2'	tri_up marker	'H'	hexagon2 marker		
'o'	circle marker	'3'	tri_left marker	'+'	plus marker		
'v'	triangle_down marker	'4'	tri_right marker	'x'	x marker		
'^'	triangle_up marker	's'	square marker	'D'	diamond marker		
'<'	triangle_left marker	'p'	pentagon marker	'd'	thin_diamond marker		
'>'	triangle_right marker	'*'	star marker	'|'	vline marker		

More common customizable options

alpha	Float number range in 0 - 1	Transparency of the line
linewidth or lw		Line width
linestyle or ls	'-' or 'solid'	solid line
	'--' or 'dashed'	dashed line
	'-.' or 'dashdot'	dash-dotted line
	':' or 'dotted'	dotted line

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], marker = '+', color = 'k' ,
lw = 2, ls = ':', alpha = 0.5)
plt.ylabel('y label')
plt.xlabel('x label')
#plt.show()
```

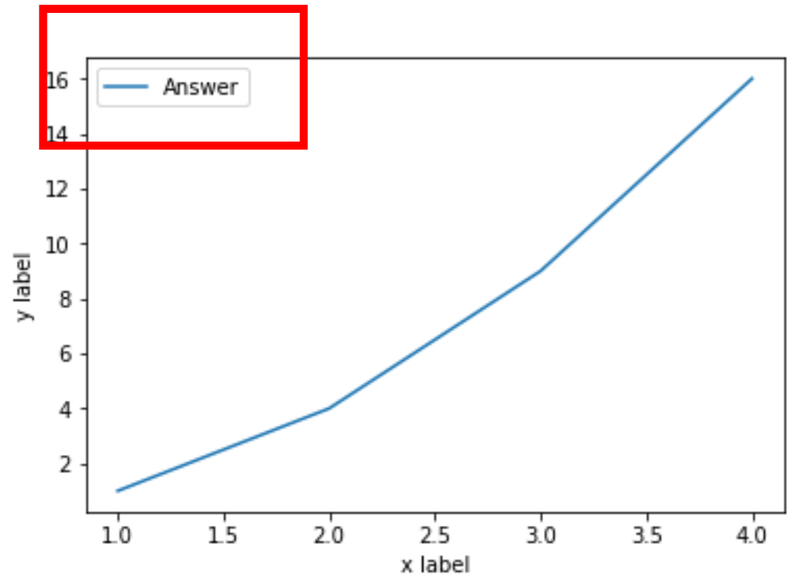


Plot with label

plt.**legend**([**Location String**])
Location String: '**best**' (default), '**upper right**', '**upper left**', '**lower left**', '**lower right**', '**right**', '**center left**', '**center right**', '**lower center**', '**upper center**', '**center**'

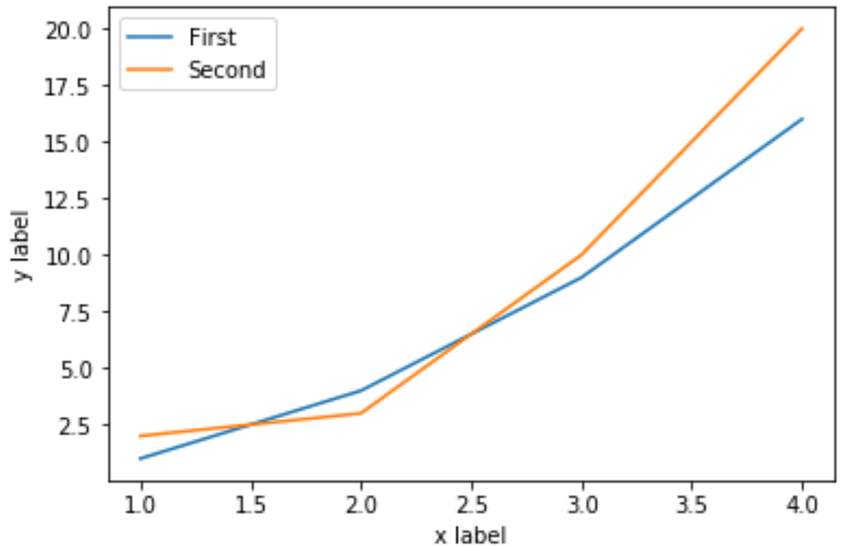
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], label = 'Answer')
plt.ylabel('y label')
plt.xlabel('x label')
plt.legend()
#plt.show()
```

To add Label



Draw multiple lines in one plot

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], label = ‘First')
plt.plot([1, 2, 3, 4], [2, 3, 10, 20], label = 'Second')
plt.ylabel('y label')
plt.xlabel('x label’)
plt.legend()
#plt.show()
```

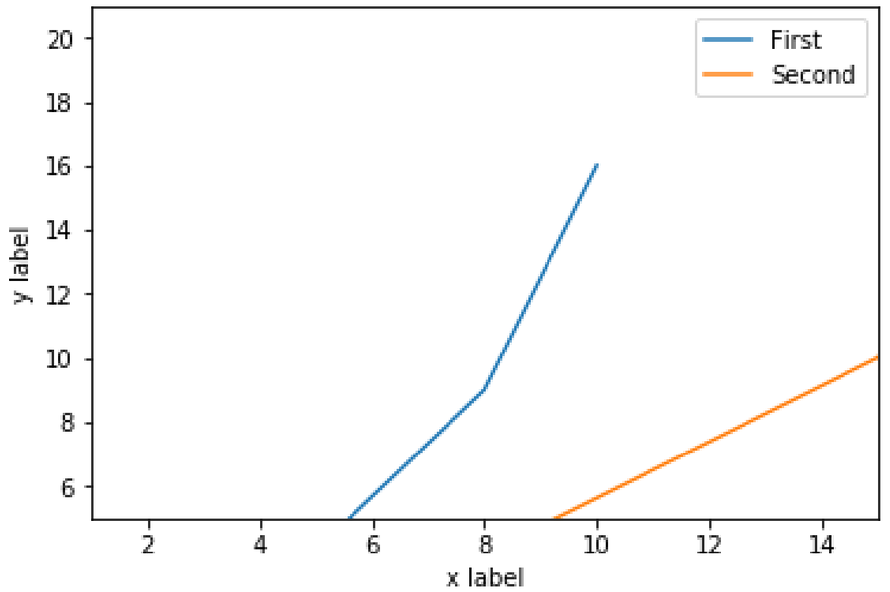


plt.xlim(), plt.ylim()

plt.xlim(start[, end])
plt.ylim(start[, end])

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 5, 8, 10], [1, 4, 9, 16], label = ‘First’)
plt.plot([1, 7, 15, 20], [2, 3, 10, 20], label = 'Second')
plt.ylabel('y label')
plt.xlabel('x label’)
plt.xlim(1,15)
plt.ylim(5)
plt.legend()
#plt.show()
```

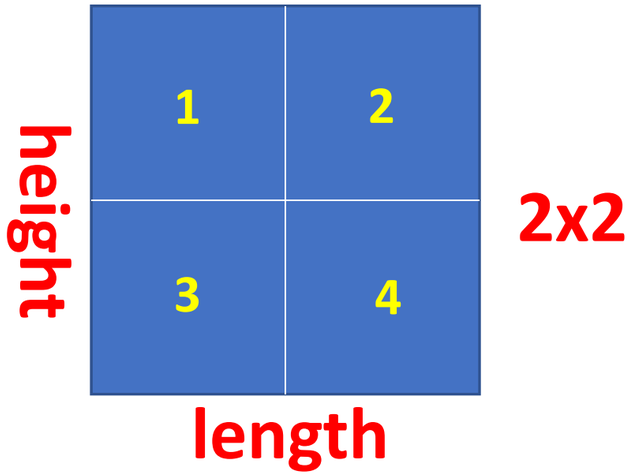
x axis show [1,15]
y axis show [5,...



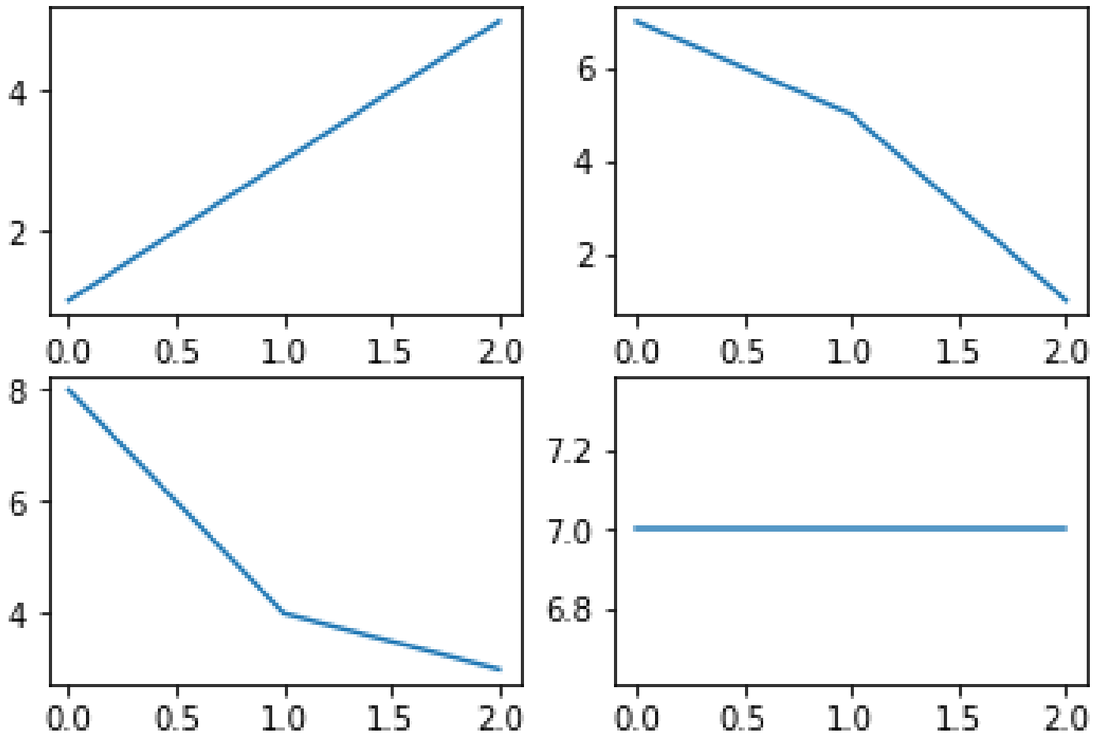
Subplot

- Can be shorten as

plt.**subplot**(height length **index**)

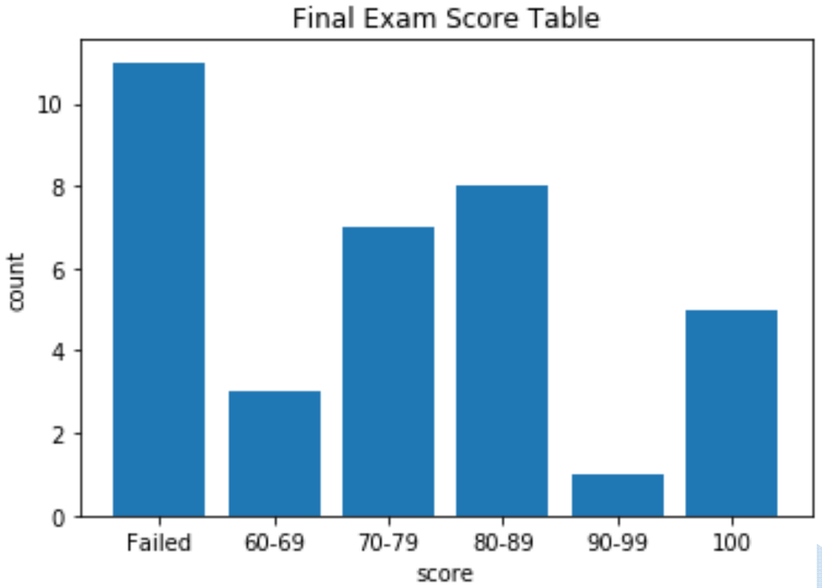



```
import matplotlib.pyplot as plt
%matplotlib inline
plt.subplot(2, 2, 1)
plt.plot([0, 1, 2], [1, 3, 5])
plt.subplot(2, 2, 2)
plt.plot([0, 1, 2], [7, 5, 1])
plt.subplot(2 ,2, 3)
plt.plot([0, 1, 2], [8, 4, 3])
plt.subplot(2, 2, 4)
plt.plot([0, 1, 2], [7 ,7, 7])

# plt.show()
```



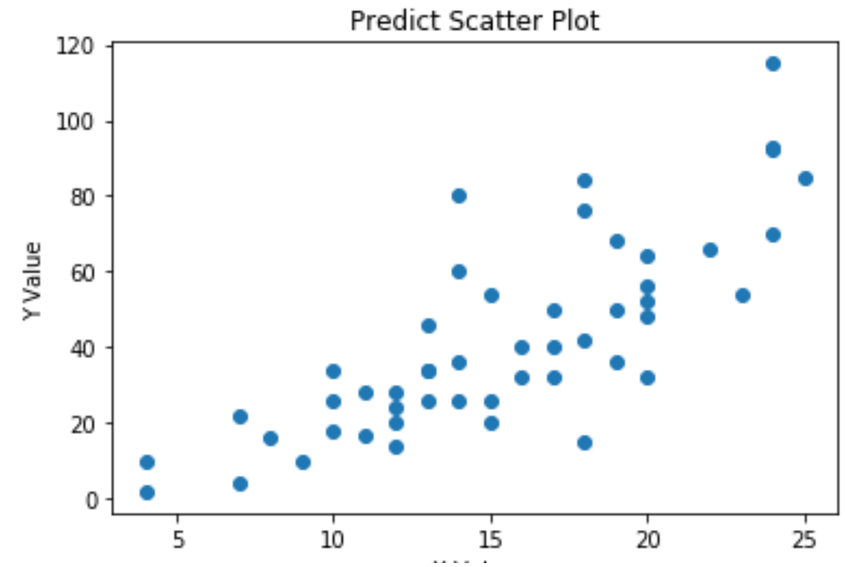
Bar Plot

```
import matplotlib.pyplot as plt
%matplotlib inline
X1 = ['Failed', '60-69', '70-79', '80-89', '90-99', '100']
Y1 = [11, 3, 7, 8, 1, 5]
plt.ylabel('count')
plt.xlabel('score')
plt.bar(X1, Y1)
plt.title('Final Exam Score Table')
#plt.show()
```



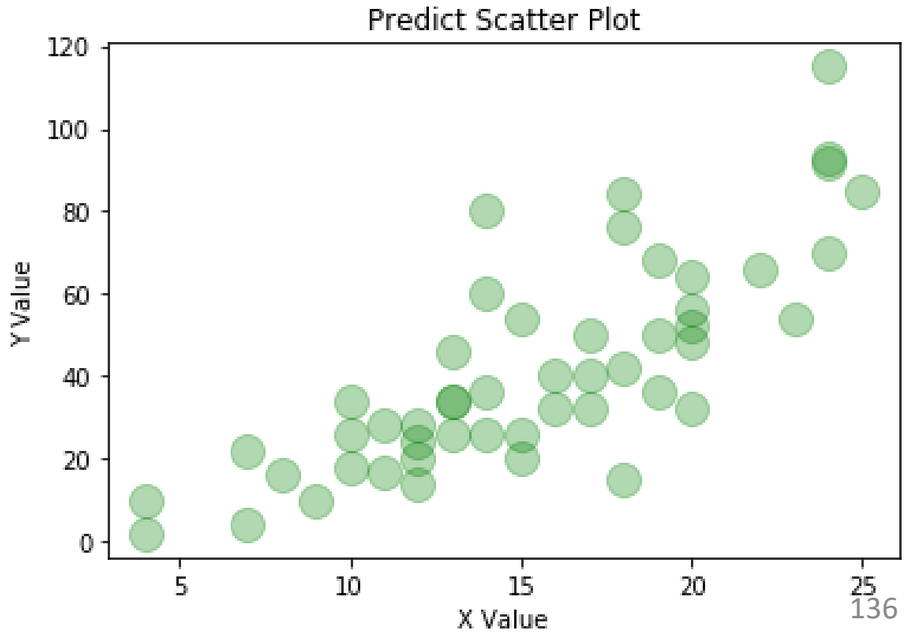
Scatter Plot

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.ylabel('Y Value')
plt.xlabel('X Value')
plt.scatter(x, y)
plt.title('Predict Scatter Plot’)
#plt.show()
```



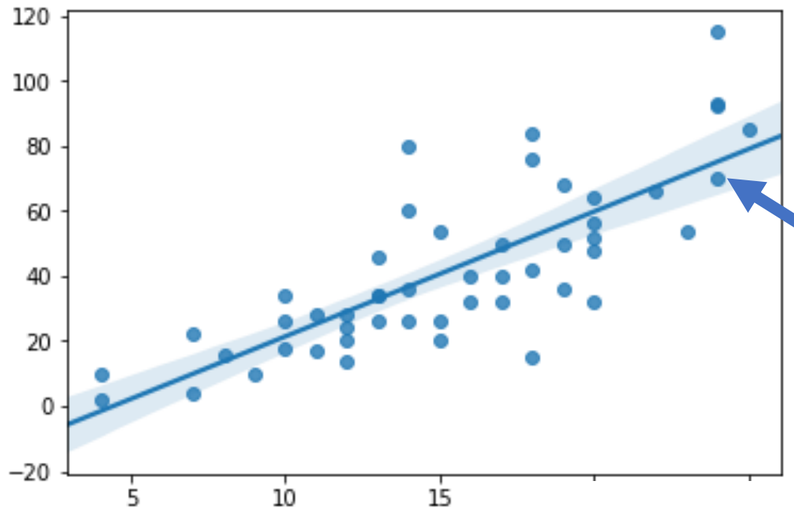

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.ylabel('Y Value')
plt.xlabel('X Value')
plt.scatter(x, y, color = "g", alpha = 0.3, s = 200)
plt.title('Predict Scatter Plot’)
#plt.show()
```

size



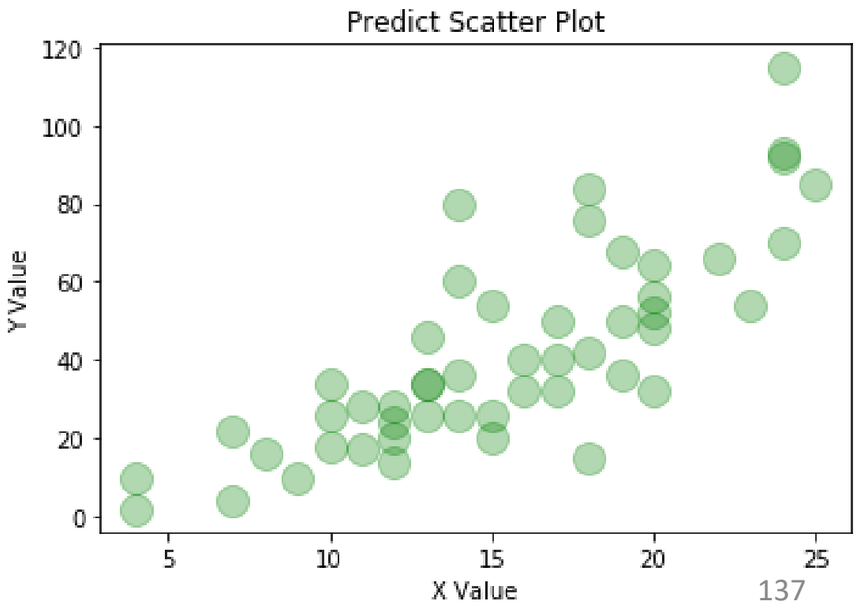
Scatter Plot with Seaborn – regplot()

```
import seaborn as sns
sns.regplot(x, y)
```



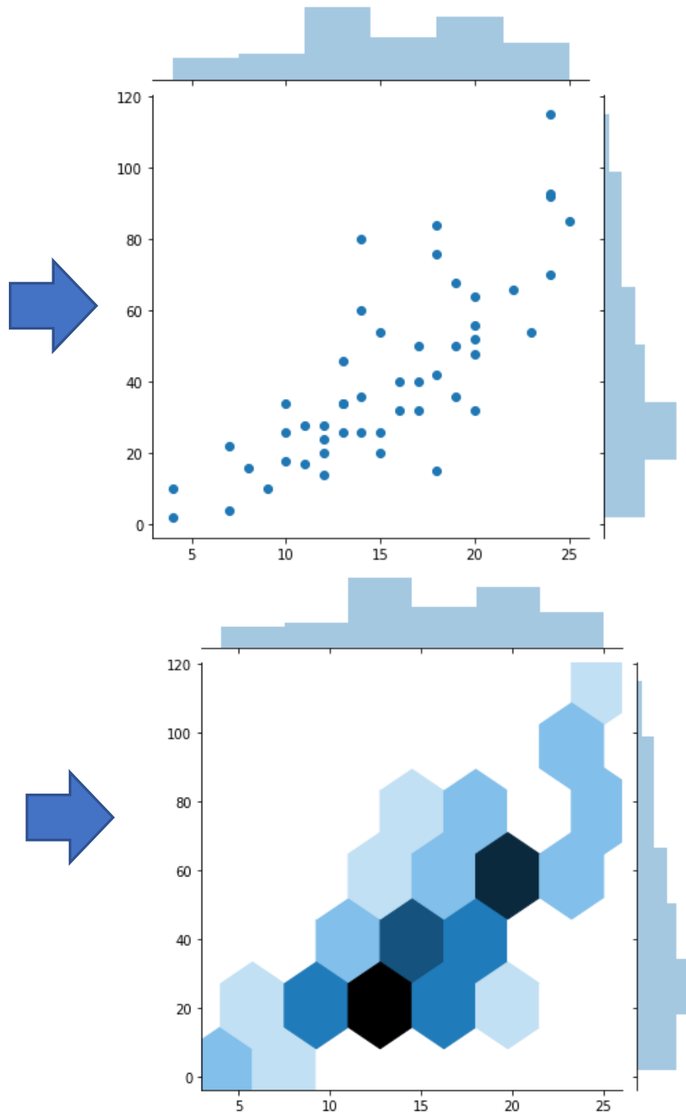
With regression line in default

```
import seaborn as sns
plt = sns.regplot(x, y,
scatter_kws={"color":"g","alpha":0.3,"s":200},
 fit_reg = False)
plt.set_title('Predict Scatter Plot')
plt.set_ylabel('Y Value')
plt.set_xlabel('X Value')
```

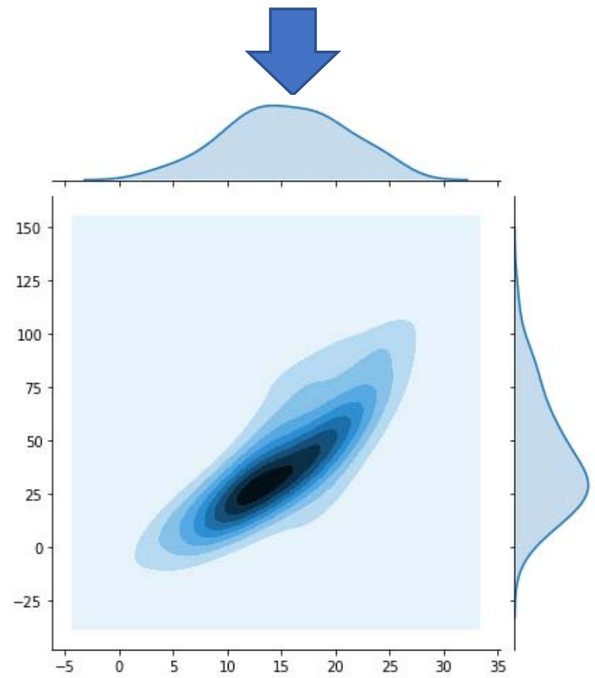


Marginal plot with Seaborn – jointplot()

```
sns.jointplot(x, y)
#sns.jointplot(x, y, kind = 'scatter')
```



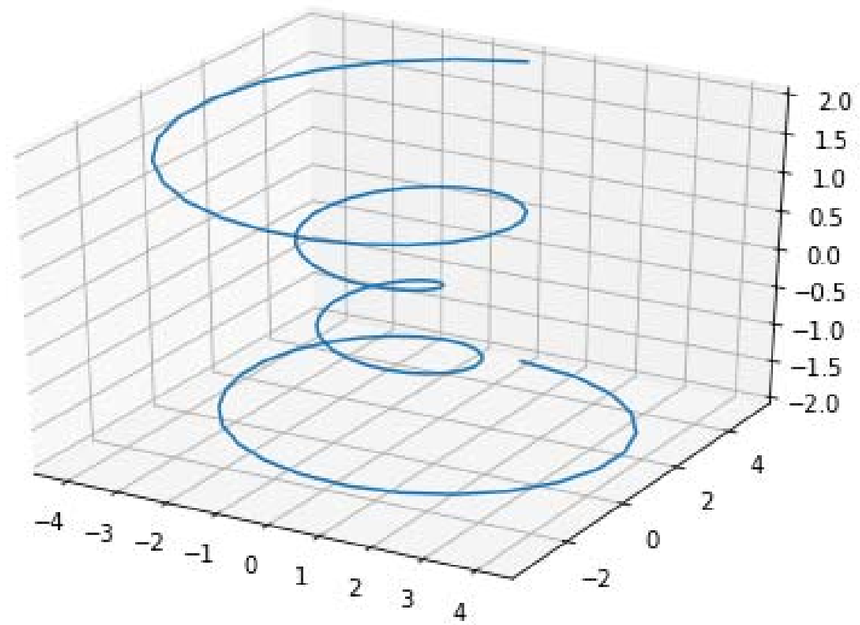
```
sns.jointplot(x, y, kind = 'kde')
```



```
sns.jointplot(x, y, kind = 'hex')
```

3D Plot with Axes3D and Matplotlib

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# create data
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
#create 3D model
fig = plt.figure()
ax = Axes3D(fig)
# Draw
ax.plot(x, y, z)
#plt.show()
```



7

Python File I/O, os, shutil, glob

open()

- Use **open(file_name[, mode][, encode])** to open a file
- Built-in function. No need to import any package.
- File operation modes:

r	*Read* mode. The cursor will start at the beginning (default).
r+	*Read and write* mode. The cursor will start at the beginning (default).
w	*Write* mode. If the file doesn't exist, it will create a new file. If the file exists, the cursor will be at the beginning, and the file will be overwritten.
w+	*Read and write* mode. If the file doesn't exist, it will create a new file. If the file exists, the cursor will start at the beginning, and the file will be overwritten.
a	*Write* mode. If the file doesn't exist, it will create a new file. If the file exists, the cursor will move to the end of the file in order to append data after it.
a+	*Read and write* mode. If the file doesn't exist, it will create a new file. If the file exists, the cursor will move to the end of the file in order to append data after it.

open()

- The file can be opened in the binary format by adding "***b***" at the end of each mode.
  - Ex: rb, rb+, wb, wb+, ab, ab+

- Encoding: There are different encoding formats used in Windows.
  - Ex : **cp950(Default in Chinese Windows), UTF-8…**
  - Ex :

```
f = open('test.txt', 'r' , encoding = 'UTF-8')
```

File I/O

read([size])	Read specific (default is all) length of characters from the file.
readline()	Read specific (default is one) lines from the current cursor position. A newline character (\n) is put at the end of the string.
readlines()	Read all lines from the file and put them into a list.
next()	Move the cursor to the next line.
write(str)	Write a string ***str*** to the file. The data ***str*** must be a string.
seek(offset, [whence])	Move the cursor to a new position by *offset* characters relative to the position specified by *whence*. The optional *whence* field defines: 0: relative to the beginning of file (default) 1: the current cursor position 2: relative to the end of the file
tell()	Return the current cursor position.
close()	Close the file. After closing the file, you can't operate the file anymore until you reopen it again.

open()

```
f=open('test.txt','r',encoding='UTF-8’)
print(f.tell())
print(f.read(7))
f.seek(0)
print(f.readline())
print(f.tell())
print(f.readline())
print(f.readline())
print(f.readline())
f.seek(0)
x = f.readlines()
print(x)
f.close()
```

test.txt

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

```
0
Hello W
Hello World!

14
Hello

World

!
['Hello World!\n', 'Hello\n', 'World\n', '!']
```

with open(...) as file_name:

- File objects can be manipulated using the ***with*** statement to provide much cleaner syntax and exceptions handling.
- Any files opened will be closed automatically after ***#code*** has been completely executed.
- Ex :

```
with open(...) as file_name:
    #code
```

```
with open('test.txt','r',encoding='UTF-8') as f:
    for line in f:
        print(line, end = '')
```

Looping over a file object

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

os module

- The ***os*** module provides a portable way of using operating system dependent functionality.

- It can let us create/delete path, delete specific file, or even run the shell commands easily.

- Run **import os** before using it.

os.path

- The module ***os.path*** provides some methods to check the existence of file/path, fetch file/path information, and operate the file paths.

abspath()	Get the absolute pathname.
basename()	Get the base name of the pathname.
dirname()	Get the directory name of the pathname. The use of *dirname(__file__)* can obtain the current directory name. (But cannot work in Jupyter Notebook. Use *os.path.abspath('')* instead.)
exists()	Check whether the file exists or not.
getsize()	Get the path size in bytes.
split()	Split the pathname *path* into a tuple ***(head, tail)*** where tail is the last pathname component and head is everything leading up to that.
splitdrive()	Split the pathname *path* into a tuple ***(drive, tail)*** where head is the driver name, and tail is the rest of pathname.
join()	Merge one or more path components together.

```
import os
cur_path=os.path.dirname(__file__)
print(cur_path)
filename=os.path.abspath("test.txt")
if(os.path.exists(filename)):
        print(os.path.basename(filename))
        print(os.path.dirname(filename))
        print(os.path.abspath(filename))
        fullpath,fname = os.path.split(filename)
        print(fullpath)
        print(filename)
        driver,fpath = os.path.splitdrive(filename)
        print(driver)
        print(fpath)
fullpath = os.path.join(fullpath+"\\"+fname)
print(fullpath)
```

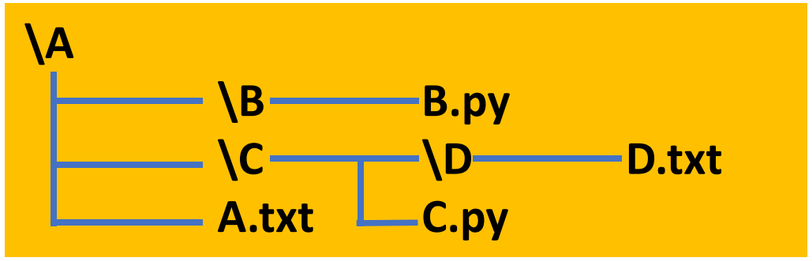
- ***__file__*** *represents the current working directory, but cannot work under the jupyter notebook.*
- *Current working directory can be obtained by* ***os.getcwd()*** *or* ***os.path.abspath('')****.*

```
C:/Users/nsysu/Desktop
test.txt
C:\Users\nsysu\Desktop
C:\Users\nsysu\Desktop\test.txt
C:\Users\nsysu\Desktop
C:\Users\nsysu\Desktop\test.txt
C:
\Users\nsysu\Desktop\test.txt
C:\Users\nsysu\Desktop\test.txt
```

os.walk()

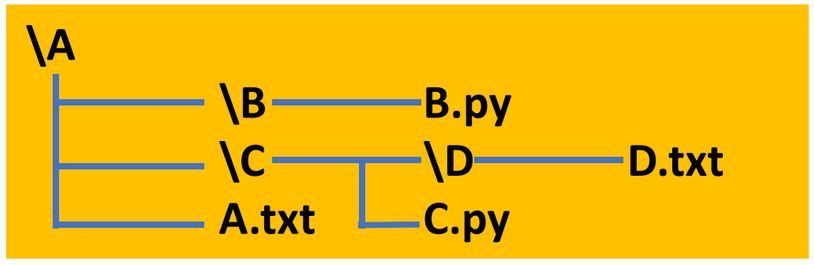
- ***os.walk()*** allows us to search a specific directory, and its sub-directory.
- It will return a (or multiple) tuple(s) with 3 elements consisting of ***(dirpath, dirnames, filenames)***. This function can be run recursively.

- **Ex: if we have a folder whose structure is:**



os.walk()

File structure




```
import os

filename = os.path.abspath("A")
s= os.walk(filename)
for dirpath,dirnames,filenames in s:
        print(dirpath)
        print(dirnames)
        print(filenames,'\n')
```

```
C:\Users\ihors\Desktop\A
['B', 'C']
['A.txt']

C:\Users\ihors\Desktop\A\B
[]
['B.py']

C:\Users\ihors\Desktop\A\C
['D']
['c.py']

C:\Users\ihors\Desktop\A\C\D
[]
['d.txt']
```

os.remove()

- Used to remove a specific file.
- Use ***os.path.exists()*** to check whether the file exists.
  - Ex :

```
import os
file = "test.txt"
if os.path.exists(file):
        os.remove(file)
        print("Remove successfully.")
else:
        print("Failed to remove file.")
```

os.mkdir()

- Used to create a specific directory.
- Use ***os.path.exists()*** to check whether a directory exists or not.

```
import os
dir = "NEW"
if not os.path.exists(dir):
        os.mkdir(dir)
        print("Create successfully.")
else:
        print(dir , "has already existed.")
```

os.system()

- os.system() allows us to **run a *shell command.***

```
import os
cur_path = os.path.dirname(__file__)
os.system("mkdir new") #create a folder “new”
os.system("copy test.txt new\copytest.txt")  #create test.txt to new\copytest.txt
file = os.path.join(cur_path, "new", "copytest.txt")
os.system("notepad "+ file) #use notepad to open copytest.txt
```

os.system()

```python
import os
cur_path = os.path.dirname(__file__)
print(cur_path)
os.system("mkdir new")
os.system("copy test.txt new\copytest.txt")
file = os.path.join(cur_path,"\\new\\copytest.txt")
os.system("notepad "+ file
```

C:\WINDOWS\system32\cmd.exe

copytest.txt - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

Hello World!
Hello
World
!

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

atically after writing a left
You can activate this behavior in

Read our tutorial

shutil

- ***shutil*** module allows us to **copy, delete, or move the file or directory.**
- Run **import shutil** before using it.

```
import os, shutil
destfile = os.path.join(os.path.abspath(''), "new", "copytest.txt")
shutil.copy("test.txt", destfile)
```

copy(src, dst)	Copy file “src” to file “dst”
copytree(src, dst)	Recursively copy an entire directory tree rooted at ”src” to another location “dst”
rmtree(dir)	Delete an entire directory tree “dir”
move(src, dst)	Recursively move a directory “src” to another location “dst”

glob

- ***glob*** module allow us to **search for a specific directory or file** easily.
- Run `import glob` before using it.
- `glob.glob(pathname[,recursive = False])` returns a list which matches *pathname*
  - Pathname can support the use of some special characters like:
    - '?' can represent one character
    - '*' can represent multiples words
    - [0-9] can represent one number
    - [a-z] or [A-Z] can represent one alphabet (not case-sensitive).

glob

```
import glob
finding = glob.glob("a.py") + glob.glob("?.txt") + glob.glob("?.py") + glob.glob("[0-9].*")
for file in finding:
        print(file)
```

```
a.py
a.txt
1.txt
1.py
12345.py
1.pdf
2.gif
```

8

Python Exception

Exception

- Just like C++, Python provides an exception handling mechanism as well.

```
try:
    #code
except exptiontype1:
    #code
except exptiontype2:
    #code
else:
    #code
finally:
    #code
```

```
try:
    a=int(input("Input dividend: "))
    b=int(input("Input divisor: "))
    c=a/b
    print(a,"/",b,"= ",end="")
except ZeroDivisionError:
    print("Division by 0!")
except ValueError:
    print("Please input a valid integer numbers!")
else:
    print(c)
```

An example of using Exception

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

Why use break here?

Exception

- The format of "*except*" is similar to *if, elif*, and *else*. The one with no exception type name is for those exceptions not specified before.
- "*else*" will be executed when no exceptions occur.
- "*finally*" will be executed definitely. It would be put at the end of the *try…except* block.
- Can put multiple exception types in one exception line.
  - Ex :

```
except ZeroDivisionError, ValueError:
    print("ZeroDivisionError or ValueError occured!")
```

Built-in Exceptions

ArithmeticError	KeyboardInterrupt	TabError
AssertionError	LookupError	SystemError
AttributeError	MemoryError	SystemExit
Exception	NameError	TypeError
EOFError	NotImplementedError	UnboundLocalError
FloatingPointError	OSError	UnicodeError
GeneratorExit	OverflowError	UnicodeEncodeError
ImportError	ReferenceError	UnicodeDecodeError
IndentationError	RuntimeError	UnicodeTranslateError
IndexError	StopIteration	ValueError
KeyError	SyntaxError	ZeroDivisionError

Exception as

```
except exptiontype as name:
    #code
```

- When an exception occurs, it may have an associated value, also known as the exception's *argument*. The presence and type of the argument depend on the exception type.
- Use "***as name***" to make the variable *name* bound to an exception instance stored in *name*.args which stores some information about the occurred exception.
- Ex:

```
except ZeroDivisionError as errormessage:
    print(type(errormessage))
    print(errormessage)
```

```
<class 'ZeroDivisionError'>
division by zero
```

Pass

- Can simply put **pass** if nothing needs to be done when the specific exception occurs.
- Ex :

```
a, b = 5, 0
try:
    c=a/b
    print(a,"/",b,"= ",c, end="")
except ZeroDivisionError:
    pass
else:
    print(c)
```

No output in this example

Raise exception

raise exceptiontype([, args] [, traceback])

- Can use **raise** function to manually **raise exception.**
- Ex : *raise RuntimeError(), raise Exception()* …..

```
try:
    raise RuntimeError('errorargs', 'errorargstraceback‘)
except Exception as errormessage:
    print(type(errormessage))
    print(errormessage.args)
    print(errormessage)
```

```
<class 'RuntimeError'>
('errorargs', 'errorargstraceback')
('errorargs', 'errorargstraceback')
```

Defining Clean-up Actions

- A ***finally*** clause is always executed before leaving the try statement.
- Can be used to define clean-up actions.
- Ex:

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

```
try:
    raise …….
finally:
    print(‘goodbye!’)
```

The with statement allows objects like files to be used in a way that ensures they are always cleaned up promptly and correctly.

```
for line in open("myfile.txt"):
    print(line, end="")
```

This code will leave the file open for an indeterminate amount of time after this part of the code has finished executing.

The use of *with….as* in general cases

```
class Sample:
    def __init__(self, name):
        print(“Hello!“, name)
    def __enter__(self):
        print("In __enter__()")
        return ‘NSYSU'
    def __exit__(self, type, value, trace):
        print("In __exit__()")
        return True  #optional

test = Sample (‘John’)
with test as t:
    # 5/0
    print(“Test:", t)
```

```
Hello!  John
In __enter__()
Test:  NSYSU
In __exit__()
```

9

Data Analysis Pandas

Overview

- **Pandas** is a powerful module for **data analyzing and processing**. For example, it supports read in SQL or spreadsheet data, etc.
- Furthermore, it provides data filtration, reshape, merging , insertion, etc.
- It supports **DataFrame** objects for data manipulation with integrated indexing.
- Conventionally use `import pandas as pd` before using pandas module.

Create DataFrame by hand

- We can create a DataFrame from a "***dictionary***" data.

```
dict1 = {'A':[1,2,3,4],'B':[5,6,7,8]}
df = pd.DataFrame(dict1)
```

```
df
```

	A	B
0	1	5
1	2	6
2	3	7
3	4	8

- *But very often we get the **DataFrame** from files, like excel, csv, etc.*

Data in ‘text.xlsx

- Suppose we want to fetch data from the ***test.xlsx*** file.

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66

Panda file import functions

pd.read_table('file_name')	Read a general delimited file into DataFrame
pd.read_csv('file_name')	Read CSV (comma-separated) file into DataFrame
pd.read_sql('file_name')	Read SQL query or database table into a DataFrame.
pd.read_excel('file_name')	Read an Excel table into a pandas DataFrame
pd.read_json('file_name')	Convert a JSON string to pandas object
pd.read_html('file_name')	Read HTML tables into a list of DataFrame objects.
pd.read_pickle('file_name')	Read Pickle file into a panda DataFrame

- Ex:

```
import pandas as pd
tables = pd.read_excel('test.xlsx')
```

Import data from excel

- Ex :

```
In [1]: import pandas as pd

In [2]: df = pd.read_excel('test.xlsx')
        print(type(df))

        <class 'pandas.core.frame.DataFrame'>

In [3]: df

Out[3]:
```

Import excel file “test.xlsx”

A DataFrame class object

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66

Jupyter can show DataFrame’s content nicely.

DataFrame.head()

- Use `df.head(lines_number)` to get first ***lines_number*** rows of data.

- Ex :

```
In [4]: df.head(3)
```

Out[4]:

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90

Some Important Attributes and Functions

		Answer
df.**shape**	Return a tuple representing the dimensionality of the DataFrame	(6, 3) #6 rows, 3 columns
df.**columns**	Return the column labels of the DataFrame.	Index(['Math', 'English', 'Computer'], dtype='object')
df.**index**	Return the index (row labels) of the DataFrame.	Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')
df.**info()**	Print a concise summary of a DataFrame.	-
df.**describe()**	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.	-

http://pandas.pydata.org/pandasdocs/stable/reference/frame.html

Select a single column

- Use **df[column_name]** to get ***one*** specific column of the DataFrame.
  - Ex:

```
math = df['Math']
print(type(math))
```

```
<class 'pandas.core.series.Series'>
```

```
math
```

```
A    90
B    75
C    80
D    80
E    77
F    22
Name: Math, dtype: int64
```

A column of DataFrame is called **Series.**

- **df[column_name] = [….]** can also be used to append a new column.

Select multiple columns

- Use **df[[name1, name2,…]]** to select one or multiple columns of the DataFrame.
  - A list of columns' name (**[… , …,]**) has to be used.
  - Cannot use *df[name1:name2]* or *df[name1, name2]*

- Ex :

```
df[['Math','Computer']]
```

A list of columns' name.

	Math	Computer
A	90	85
B	75	65
C	80	90
D	80	33
E	77	53
F	22	66

DataFrame is composed of one or more Series

Select a specific or multiple rows (by name)

- Use `df.loc[row_name]` and `df.loc[ [row_name] ]` to get a specific and multiple rows of the DataFrame respectively according to the given **row name.**

- Ex :

```
df.loc['A']
```

```
Math        90
English     90
Computer    85
Name: A, dtype: int64
```

It's a Series.

```
df.loc[['B','D']]
```

	Math	English	Computer
B	75	77	65
D	80	55	33

Get multiple rows by a list of row names.

Select specific range of rows

- Can use `df.loc[name1:name2]` to get the specific row range of DataFrame **by rows' name**.
- Ex :

Original DataFrame

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66

```
df.loc['B':'E']
```

Select row 'B' to 'E' (including 'E')

	Math	English	Computer
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53

```
df.loc['B':'E':2]
```

Select row 'B' to 'E' with a stride of 2.

	Math	English	Computer
B	75	77	65
D	80	55	33

Select specific row and column (by name)

- Use **df.loc[row_name, column_name]** to get the specific rows and columns of the DataFrame by name.
  - Use a list ***[x, y,.....]*** or a range ***x:y***

```
df.loc[['B','D'], ['Math','Computer']]
```

	Math	Computer
B	75	65
D	80	33

Row

Column

```
df.loc['B':'E', 'Math':'English']
```

	Math	English
B	75	77
C	80	97
D	80	55
E	77	20

Row

Column

Select a specific or multiple rows (by index)

- Use **df.iloc[row_index]** to get the specific rows by ***index*** of the DataFrame
  - iloc represents “integer-location”

```
df.iloc[0]
```

select row 0

```
Math        90
English     90
Computer    85
Name: A, dtype: int64
```

```
df.iloc[0:3]
```

select row 0, 1, 2

Doesn’t include row 3 !

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90

Compared with df.loc which will include end label ‘C‘

```
df.loc['A':'C']
```

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90

Select specific range of rows (by index)

- Use **df[start : end]** to get the specific row range of DataFrame by index, same as ***df.iloc[start : end]***

Ex:

row 0~1 (*not including row 1*)

```
df[0:1]
```

	Math	English	Computer
A	90	90	85

```
df[3:6]
```

row 3~6 (*not including row 6*)

	Math	English	Computer
D	80	55	33
E	77	20	53
F	22	80	66

Ex:

can't select one single row by index

```
df[0]
```

```
---------------------------------------------------------------------------
KeyError
Traceback (most recent call last)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3077                 try:
-> 3078                     return self._engine.get_loc(key)
```

Select specific rows and columns using indices

- Use **df.iloc[row_index, column_index]** to get the specific row and column of the DataFrame by index

df.iloc uses index while df.loc uses label !

```
df.iloc[0:2,0]
```

Select row 0~2, and column 0

```
A    90
B    75
Name: Math, dtype: int64
```

```
df.loc['A':'B','Math']
```

```
A    90
B    75
Name: Math, dtype: int64
```

```
df.iloc[[0,2,3],[1]]
```

	English
A	90
C	97
D	55

Select rows 0, 2, 3, and column 1

```
df.loc[['A','C','D'],['English']]
```

	English
A	90
C	97
D	55

Modify data

- Use the same way of row or column selection for data modification.

```
df.iloc[0] = np.zeros(3)
df.loc['C','English'] = 0
df['Math']['B'] = 0
```

by index
by name
Select column and then select row

```
df
```

	Math	English	Computer
A	0.0	0.0	0.0
B	0.0	77.0	65.0
C	80.0	0.0	90.0
D	80.0	55.0	33.0
E	77.0	20.0	53.0
F	22.0	80.0	66.0

Append a new data row / column

- Use `df.loc[new_row_name] = value` to append a new row.
- Use `df[new_col_name] = value` to append new column.

```
df.loc['Z'] = [100,90,80]
```

```
df
```

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66
Z	100	90	80

```
df['Physics'] = 60
```

```
df
```

	Math	English	Computer	Physics
A	90	90	85	60
B	75	77	65	60
C	80	97	90	60
D	80	55	33	60
E	77	20	53	60
F	22	80	66	60

Delete data

```
df.drop(row_or_column_name, axis = 0, inplace = False)
```

drop row 'A'

```
df.drop('A', inplace = True)
```

"True" will modify df itself.

```
df
```

	Math	English	Computer
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66

```
df.drop(['Math','Computer'], axis = 1)
```

Drop 'Math' and 'Computer' columns

	English
B	77
C	97
D	55
E	20
F	80

"True" will return a copy of DataFrame while the original DataFrame will not be modified.

DataFrame mask

- Similar to the mask used in ndarray.

```
df['English'] > 60
```

```
A     True
B     True
C     True
D    False
E    False
F     True
Name: English, dtype: bool
```

Create a mask for ‘English > 60’

original data

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66

```
df[ df['English'] > 60]
```

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
F	22	80	66

Get the rows which meet the criteria of ‘English > 60’

```
df[ (df['English'] > 60) | (df['Math'] >= 80) ]
```

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
F	22	80	66

Drop Missing Data

original data

	Math	English	Computer
A	90.0	90	85.0
B	75.0	77	65.0
C	NaN	97	90.0
D	80.0	55	33.0
E	77.0	20	53.0
F	22.0	80	NaN

- When there are some missing data,
  - Use **dropna(axis = 0, inplace = ….)** to **delete missed rows/column.**

```
df.dropna()
```
default axis = 0

	Math	English	Computer
A	90.0	90	85.0
B	75.0	77	65.0
D	80.0	55	33.0
E	77.0	20	53.0

drop row ‘C’ and row ’F’

```
df.dropna(axis = 1)
```

	English
A	90
B	77
C	97
D	55
E	20
F	80

drop columns of ‘Math’ and ‘Computer’

Fill Missing Data

- When there are some missing data,
  - Use **fillna(value, inplace =…)** to **fill value in missing data.**

	Math	English	Computer
A	90.0	90	85.0
B	75.0	77	65.0
C	NaN	97	90.0
D	80.0	55	33.0
E	77.0	20	53.0
F	22.0	80	NaN

original data

```
df.fillna(60)
```

	Math	English	Computer
A	90.0	90	85.0
B	75.0	77	65.0
C	60.0	97	90.0
D	80.0	55	33.0
E	77.0	20	53.0
F	22.0	80	60.0

Data sorting in DataFrame

- Use `name.sort_index(inplace = False)` lets you sort the rows based on the index.

```
df.sort_index(inplace = True)
df
```

	Math	English	Computer
E	77	20	53
D	80	55	33
B	75	77	65
F	22	80	66
A	90	90	85
C	80	97	90

original data

sort by index

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66

Substitute data of Series

- Use **df.map(arg, na_action = None)** for substituting data value of a Series. ***'arg'*** can be a 'function' or a 'dictionary'.

map by function

```
def func(x):
    if x >= 60:
        return 'Pass'
    else:
        return 'Failed'
```

```
df['English'] = df['English'].map(func)
df
```

Pass every value of series to 'func' as parameter.
The return value will replace the original value.

	Math	English	Computer
A	90	Pass	85
B	75	Pass	65
C	80	Pass	90
D	80	Failed	33
E	77	Failed	53
F	22	Pass	66

map by dictionary

```
mapdict = {80:'eighty',90:'ninty'}
```

```
df['Math'] = df['Math'].map(mapdict)
df
```

Values that are **not found in the dictionary** will be converted to **NaN.**

	Math	English	Computer
A	ninty	Pass	85
B	NaN	Pass	65
C	eighty	Pass	90
D	eighty	Failed	33
E	NaN	Failed	53
F	NaN	Pass	66

Substitute data of DataFrame

- Use **df.apply(func, axis = 0)** to apply a function along the specified axis of the DataFrame.

```
def average(x):
    return int(sum(x) / x.shape[0])
```

Pass **every column of DataFrame** as the parameter to 'average'.

```
df.loc['Average'] = df.apply(average, axis = 0)
df
```

	Math	English	Computer
A	90	90	85
B	75	77	65
C	80	97	90
D	80	55	33
E	77	20	53
F	22	80	66
Average	70	69	65

axis 0

Put the function return value at the new row.

Pass **every row of DataFrame** as the parameter to 'average'.

```
df['Average'] = df.apply(average, axis = 1)
df
```

axis 1

	Math	English	Computer	Average
A	90	90	85	88
B	75	77	65	72
C	80	97	90	89
D	80	55	33	56
E	77	20	53	50
F	22	80	66	56

Put the return value at the new column

10

Image Processing Pillow

Open Image

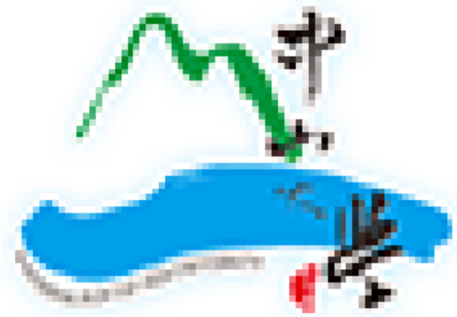
- **Pillow** is a powerful module for **photo/image processing**!

- Most of the works can be done with just "Image" package,
  - Use **from PIL import Image** before using PIL module.

- When demonstrating the images which have been processed, we can use both Pillow and Matplotlib simultaneously.

plot

```
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image

img = Image.open('logo.png')
plt.imshow(img)
plt.axis('off')
```



In default, the axis is **on.**
or you can set it “**on**”

```
plt.axis('on')
```

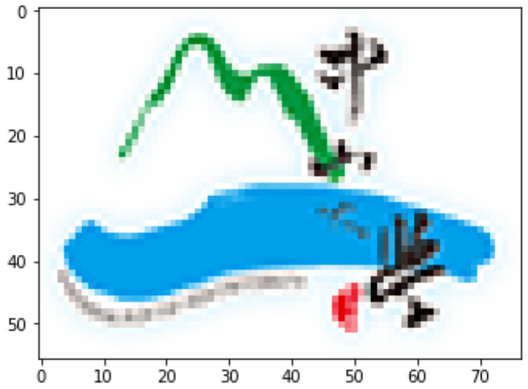


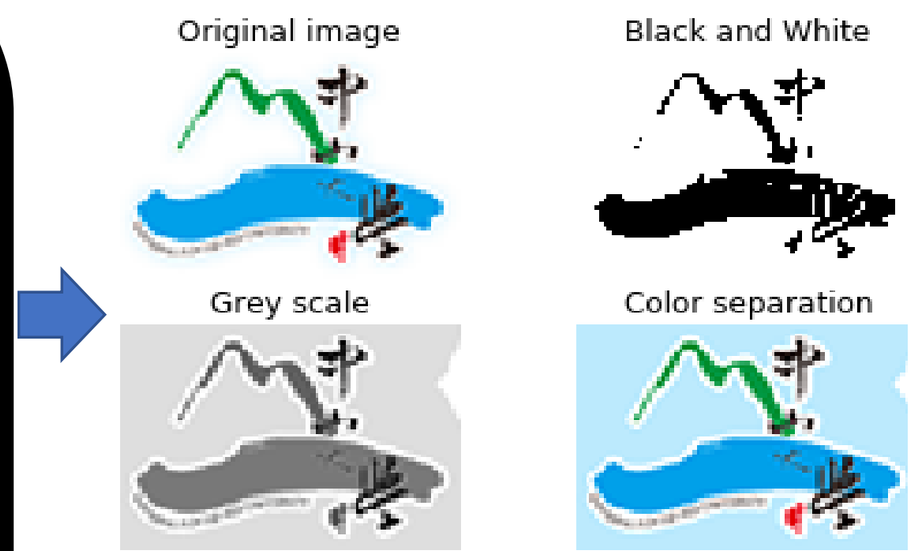
Image.convert()

- Convert the color format of the photo by using `img = img.convert('Type')`
- Popular types
  - ‘1’ : dither = Image.NONE : black and white
  - ‘L’ : greyscale
  - ‘RBG‘ : 2x8-bit pixels, true color
  - ‘RGBA‘ : 4x8-bit pixels, true color with transparency mask
  - ‘CMYK’ : 4x8-bit pixels, color separation

```
img = Image.open('logo.png')
img = img.convert('L')
plt.imshow(img)
plt.axis('off’)
```

Image subplots

```
img = Image.open('logo.png')
plt.subplot(2,2,1)
plt.imshow(img)
plt.axis('off')
plt.title('Original image')

plt.subplot(2,2,2)
img2=img.convert('1', dither = Image.NONE)
plt.imshow(img2)
plt.axis('off')
plt.title('Black and White')
plt.subplot(2,2,3)
img3=img.convert('L')
plt.imshow(img3)
plt.axis('off')
plt.title('Grey scale')

plt.subplot(2,2,3)
img3=img.convert('L')
plt.imshow(img3)
plt.axis('off')
plt.title('Grey scale')

plt.subplot(2,2,4)
img4=img.convert('CMYK')
plt.imshow(img4)
plt.axis('off')
plt.title('Color separation')
```



Useful image functions

- Image.convert(mode=*None*, matrix=*None*, dither=*None*, palette=*0*, colors=*256*)
- Image.copy()
- Image.crop(box=*None*)
- Image.filter(filter)
- Image.getpixel(xy)
- Image.histogram(mask=*None*, extrema=*None*)
- Image.resize(size, resample=*3*, box=*None*, reducing_gap=*None*)
- Image.rotate(angle, resample=*0*, expand=*0*, center=*None*, translate=*None*, fillcolor=*None*)
- Image.show(title=*None*, command=*None*)
- Image.save(fp, format=*None*, **params)

Image.load()

```
pixdata = img.load()
width, height = img.size
for x in range(width):
    for y in range(height):
        pixdata[x, y]=…….
```

➢ Coordinates of image pixels (width=W, height=H)

[0,0]	[1,0]	[2,0]		(W-1,0)
[0,1]				
[0,2]				
[0,H-1]				[W-1,H-1]

◆Can also use ***np.array(img)*** to take out the image pixels into a numpy data.

- ➢ Ex, pixel_np = np.array(img)
- ➢ However, $pixdata[x, y] \leftrightarrow pixel\_np[y, x]$