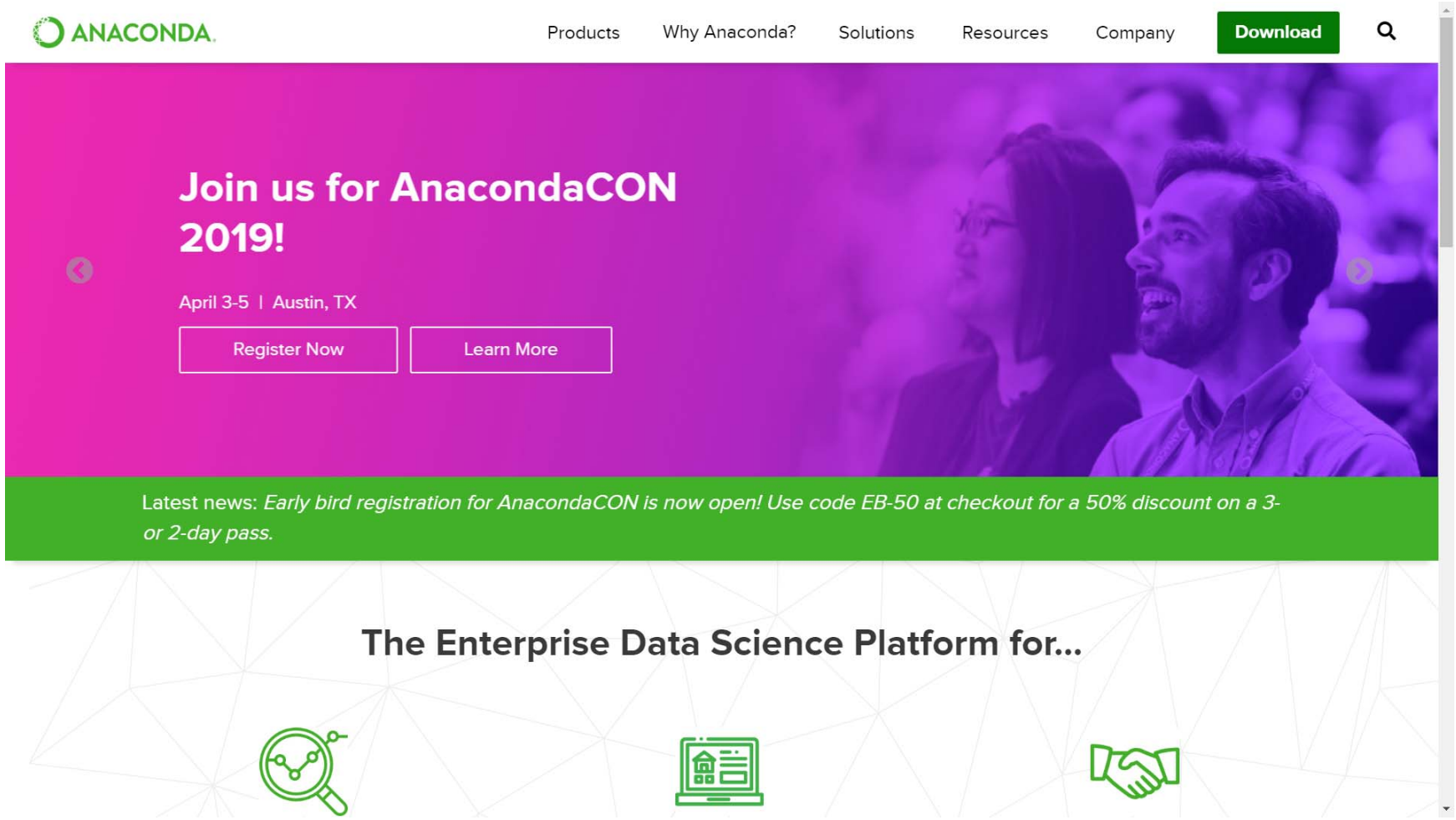
0

Python
Install Anaconda

Go to Anaconda webpage and click Download



Download Python 3.7 version and install it

- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews

Windows | macOS | Linux

Anaconda 2018.12 for macOS Installer

Python 3.7 version

Download

64-Bit Graphical Installer (652.7 MB)
64-Bit Command Line Installer (557 MB)

Python 2.7 version

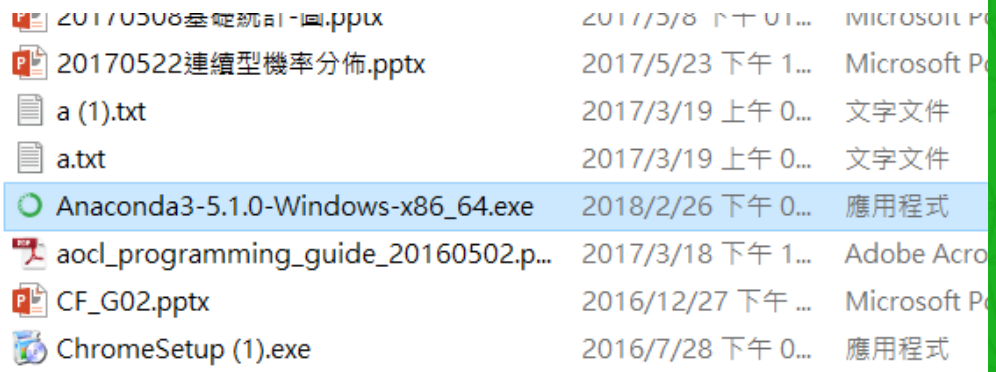
Download

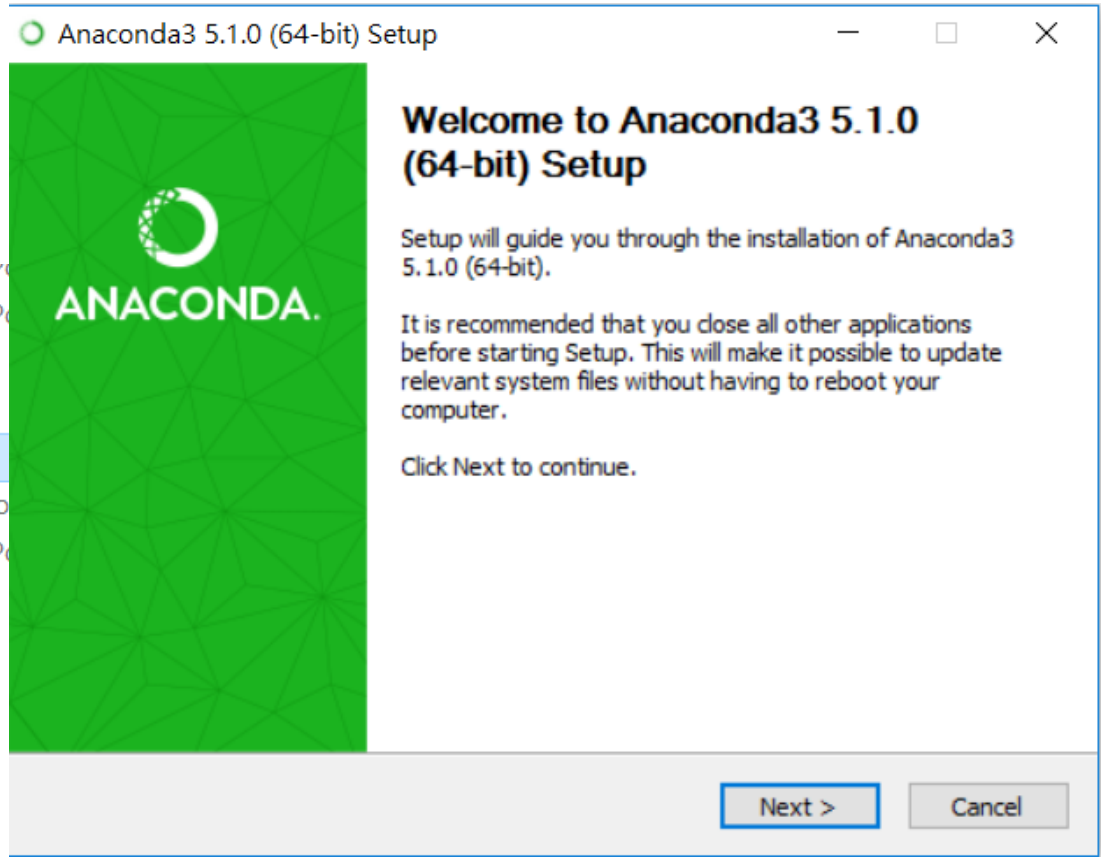
64-Bit Graphical Installer (640.7 MB)
64-Bit Command Line Installer (547 MB)

Get Started with Anaconda Distribution

Anaconda 安裝

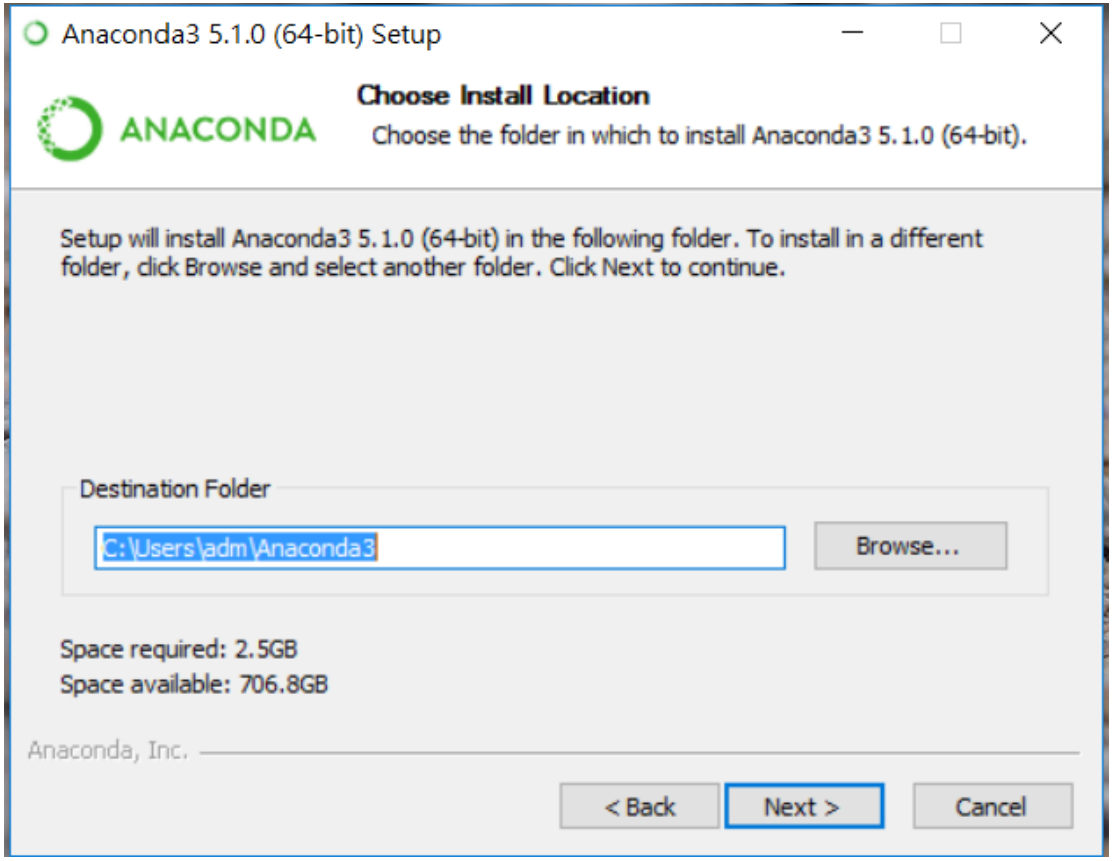
- 點擊安裝檔





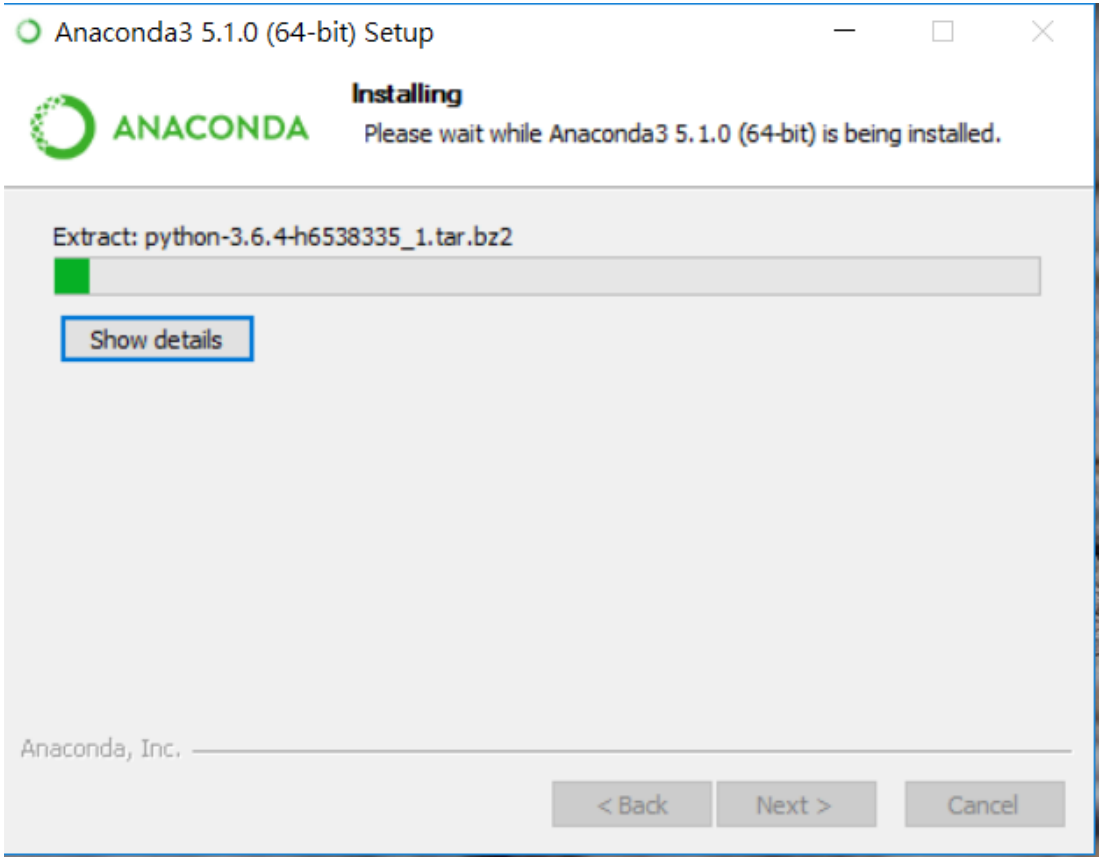
Anaconda 安裝

- 設定安裝路徑

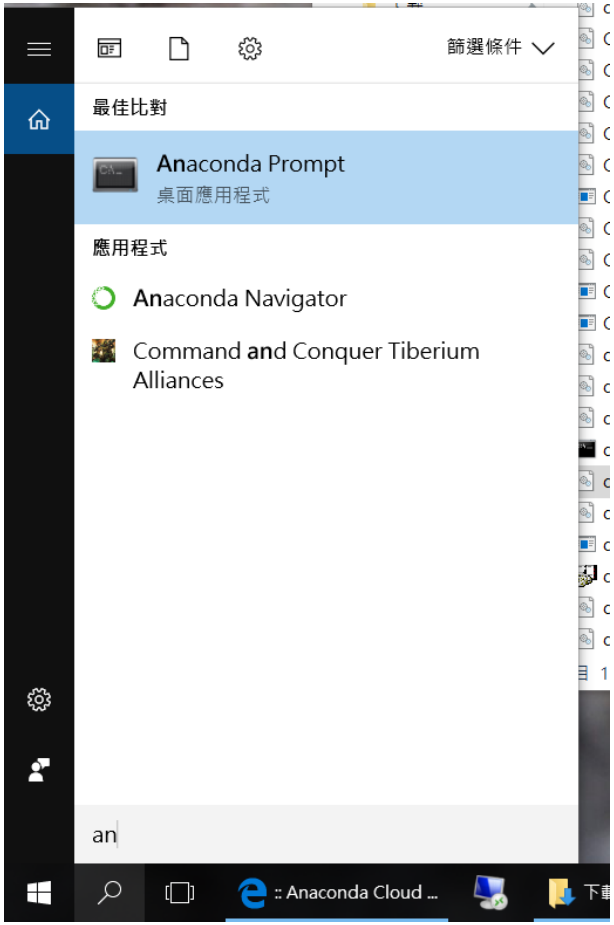


Anaconda 安裝

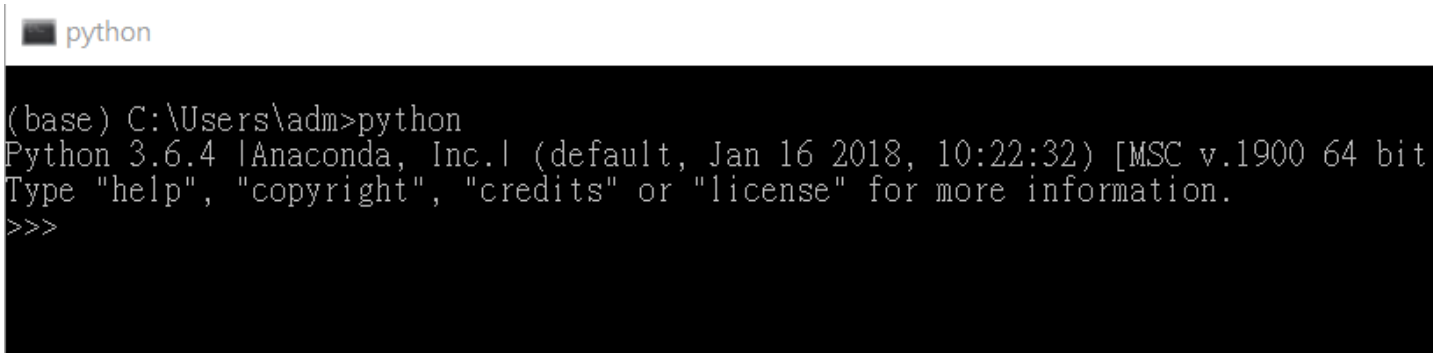
- 等待安裝完畢



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 檔名.py

hello.py

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

python程式撰寫(利用終端機)

cd 到檔案存放位置

Anaconda Prompt

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

(base) D:\>
```

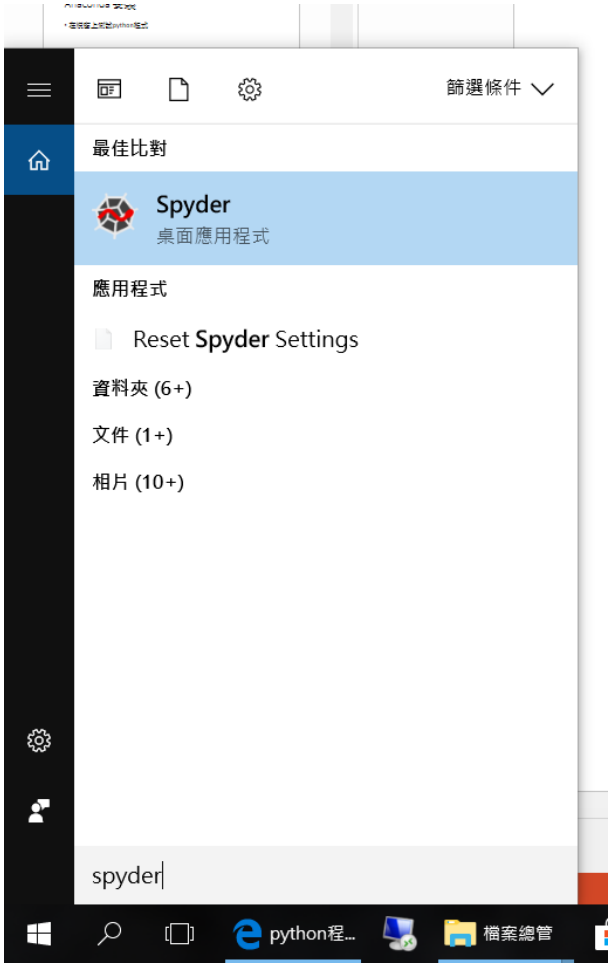
執行檔案

python 檔名.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
```

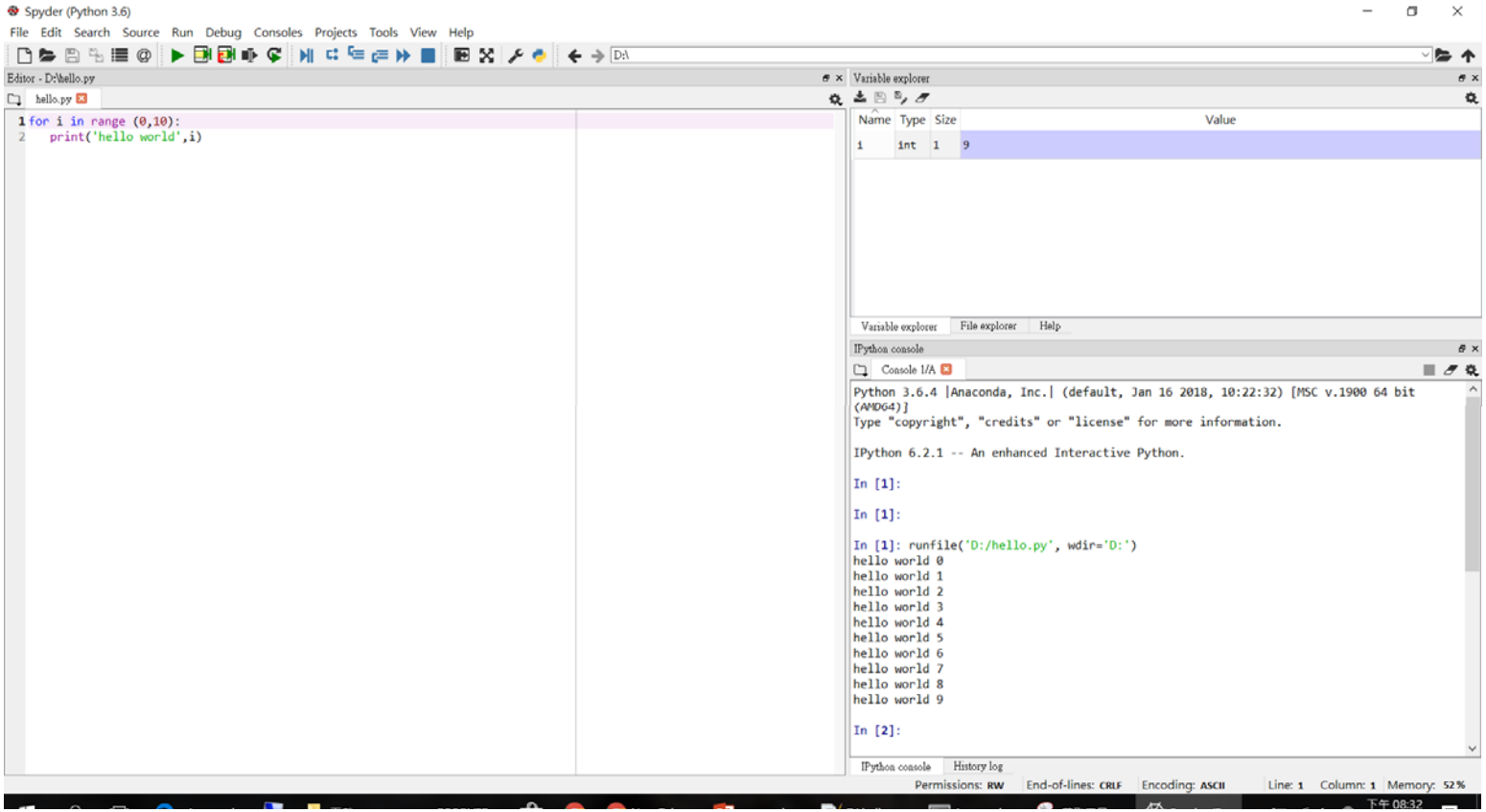
python程式撰寫(利用spyder)

- 搜尋spyder



python程式撰寫(利用spyder)

- 拖曳檔案至視窗，按下執行鍵(綠色三角形)



python程式撰寫(jupyter notebook)

- 在anaconda prompt輸入ipython notebook

Anaconda Prompt - ipython notebook

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

(base) D:\>jupyder notebook
'jupyder' 不是內部或外部命令、可執行的程式或批次檔。

(base) D:\>ipython notebook
[TerminalIPythonApp] WARNING | Subcommand `ipython notebook` is deprecated and will be removed in future versions.
[TerminalIPythonApp] WARNING | You likely want to use `jupyter notebook` in the future
[I 20:35:18.049 NotebookApp] JupyterLab beta preview extension loaded from C:\Users\adm\Anaconda3\lib\site-packages\jupy
terlab
[I 20:35:18.049 NotebookApp] JupyterLab application directory is C:\Users\adm\Anaconda3\share\jupyter\lab
[W 20:35:18.049 NotebookApp] Error loading server extension jupyterlab
    Traceback (most recent call last):
      File "C:\Users\adm\Anaconda3\lib\site-packages\jupyterlab\commands.py", line 321, in __init__
        self._run(['node', 'node-version-check.js'], cwd=HERE, quiet=True)
      File "C:\Users\adm\Anaconda3\lib\site-packages\jupyterlab\commands.py", line 1165, in _run
        proc = Process(cmd, **kwargs)
      File "C:\Users\adm\Anaconda3\lib\site-packages\jupyterlab\process.py", line 73, in __init__
        self.proc = self._create_process(cwd=cwd, env=env)
```

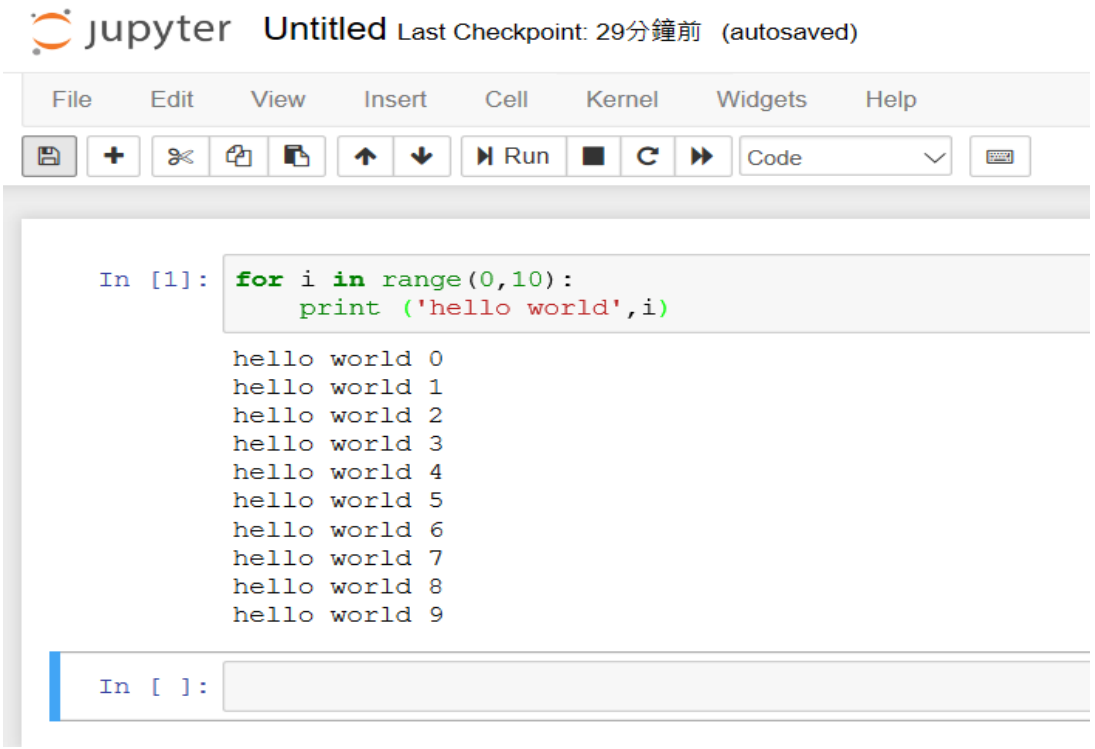
python程式撰寫(jupyter notebook)

新增檔案

New -> Python3

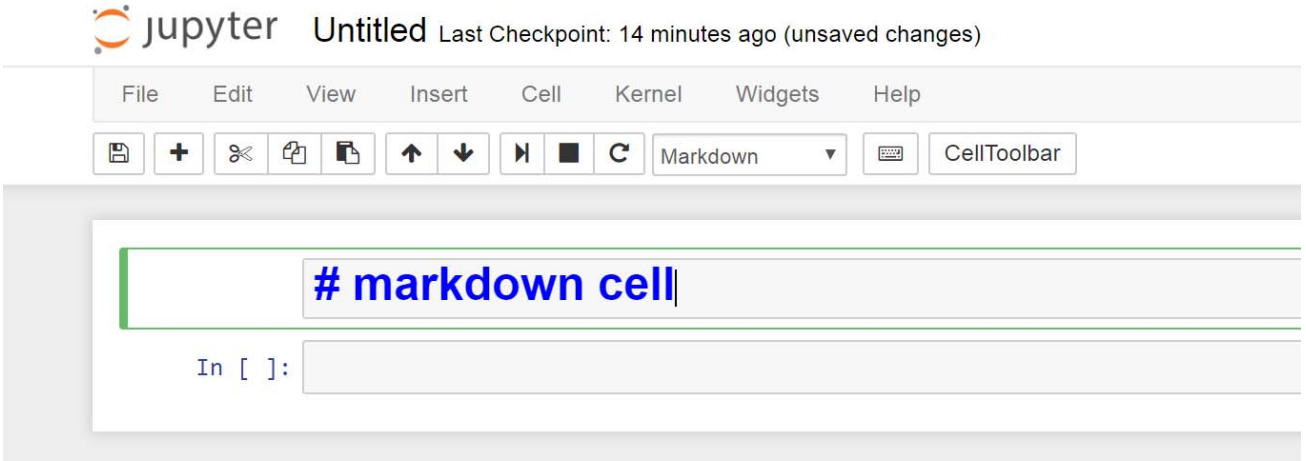
python程式撰寫(jupyter notebook)

- 在cell中撰寫程式，點擊run執行



Markdown syntax

- 產生cell，將其性質從Code改為markdown



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

Some Markdown examples

- # first-level heading
- ## second-level heading
- $Latex a^b=c$
- http://example.com/
- *literal asterisks*
- Use the `printf()` function.
- *single asterisks*
- _single underscores_
- **double asterisks**
- __double underscores__

```
This is a normal paragraph:

    This is a code block.
```

```
* Item 1
* Item 2
+ Item 3
-  Item 4
```

```
*   A list item with a blockquote:

    > This is a blockquote
    > inside a list item.
```

1

Python Basic

Quick look of Python v.s C++/C

Python:

```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("product sum =", psum[i])
print("total sum=", np.sum(psum));
```

C/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("product sum =%d \n", psum[i]);
    }
   print("total sum=%d \n", np_sum(psum));
}
```

Variable

- **In C++:**

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

- **In Python:**
  - **You don’t need to declare the data type**

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

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

Multiple assignment

- **In Python:**
  - If you want to assign several variables with the **same** value, you can use the sentence like

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

  - Then A, B, C will all be assigned to 20.

  - If you want to assign several variables with **different** values, you can use the sentence like

```
A , B , C = 10, 15, 20
```

  - Then A will be assigned to 10, B to 15, C to 20.

String Variable

- String variable can be enclosed in any type 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**, while ' ' are for **character**.

Type of the variable

- However, “**type**” is still exist in Python.
- If you don’t know what the type of the certain variable, you can use

```
type(variable)
```

to know what’s the type of that variable.

- **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 the variable

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

Variable explorer

Name	Type	Size	
A	int	1	1
B	float	1	2.3
C	bool	1	True
D	str	1	Hello World

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

Assignment = in Python

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

- **We'll talk about the concept of Mutable/Immutable in chapter 3.**

Semicolon

- **In Python:** it doesn't necessarily to add **semicolon** (**;**) at the end of each line, but add it is still fine!

- However, if there're **two codes in one line**, then you **must separate them with semicolon**

- **Ex**

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

=

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

Comments

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

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

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

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

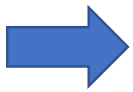
Print

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

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

  - "**sep**" and "**end**" are not necessarily
    - **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
I BANG have BANG an BANG apple
Total: Apple Pen!
```

Print

- However, you can use **print** in the way that similar to **C** Language!

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

- **EX**

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

→

Zhang's score was 60.

Print

- The third way print function supported is…

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

- **EX**

```
Name = "Zhang"
Score = 60
print ("{}'s score was {}." .format (Name, Score))
```

→

Zhang's score was 60.

Input

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

```
variable = input ("Your prompt")
```

- **EX**

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

- Be careful, the input function in Python will return a "**string**" type variable. You have to do typecasting if you want it to be the else type.
- **Ex**

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

2

Python
If, For, While

if elif else

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

- **Python** uses **elif** to replace "**else if**" in **C/C++.**
- **Python** uses **indentation** to indicate a block of code! So **indentation is necessary**. Also, colon "**:**" is necessary.
- **Expression don't** need to add **parentheses**. But add them were fine.
- **Ex**

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

and not or

- In **Python:** We use “and”, “not”, “or” to replace “&&”, “!”, “||” from **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**
//	Divide and get quotient	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 have something like **X++** or **++X**. Yet it still has assignment operators like X**+=**2, X***=**2, X**%=**2…

- **Question**
  - If **X = 5**, then what will X be after doing **X/=2**

Ans

X = 2.5

Range

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

- This function will create a list from **start**(**not necessary**) to **end-1** with **interval**(**not necessary**)
- If there's only one number filled in the range function, then it will start from **0** to **end-1**, with 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 = range(-2, 3, 2)**

  Ans

  A = [-2, 0, 2]

- Create a list from 5 to -1

  Ans

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

For Loop

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

- Again, **Python** uses **indentation** to indicate a block of code!
- **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 the same as **C/C++**

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
7 is a prime number
8 is not a prime number
9 is a prime number
9 is not a prime number
```

For Loop with else

- **In Python:** We can do a handy trick like something showed in the 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, than 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 was end **normally**(without break), then it will execute things in else

While Loop

```
while expression:
    #code
```

- The usage is the same as **C/C++!**
- Expression don't need to add parentheses just like "**if**" function in **Python**.
- There's no "**Do … While**" in **Python**.

3

Python
List, Tuple, Dict, Set
, Mutable/Unmutable

Overview

- In **Python:** there're 3 main data structures. They're "**List**", "**Tuple**", and "**Dict**"
- "**List []**" is like "**vector** (dynamic array)" in **C++**.
- "**Tuple ()**" is like an **constant array** in **C.** The data in the tuple can't be modified after assignment.
- "**Dict {}**" is like "**map**" in **C++.** Each element contain "**Key**" and its corresponding "**Value**".

- In **C**: the type of the element in the array **must be the same**
- In **Python**: we **can** have **different** type of variables in list, tuple, and dict.

List []

- We can create a list like...

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

- We can also have some list, tuple, dict in the list

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

- If we want to get the **1st** element in B: `D = B[ 1 ]` → **D = 'Hello World'**
- The position starts from **0**, just like how we done in **C/C++.**
- In **Python:** if the index is **less than 0**, than it would count **in backward**.

  `E = B[ -1 ]` → **E = 'H'**

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

  `F = B[ 2 : 4 ]` → **F = [3.0, True]**

List: append() & insert()

- After using **append**(**A**), item **A** will be added **at the end** of the original list.
- **Ex**

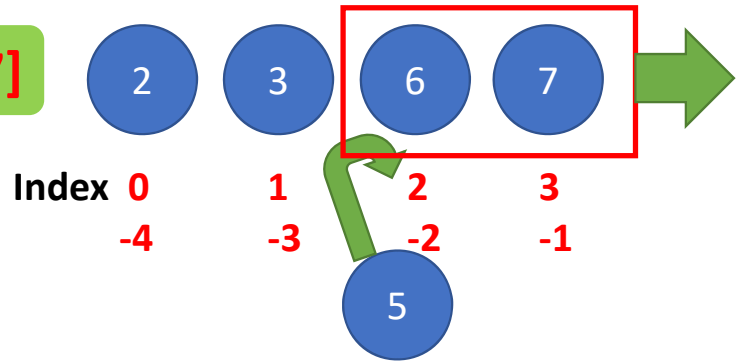
```
A = [2, 3, 4]
A.append (5)
```

→ **A = [2, 3, 4, 5]**

- We could use **insert**(**B**,**A**) if we want to add item **A** at the index **B**.
- **Ex**

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

→ **A = [2, 3, 5, 6, 7]**



Questions

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

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

  Ans: **B = [3, 4]** Because the last one(-1th) **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 function

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

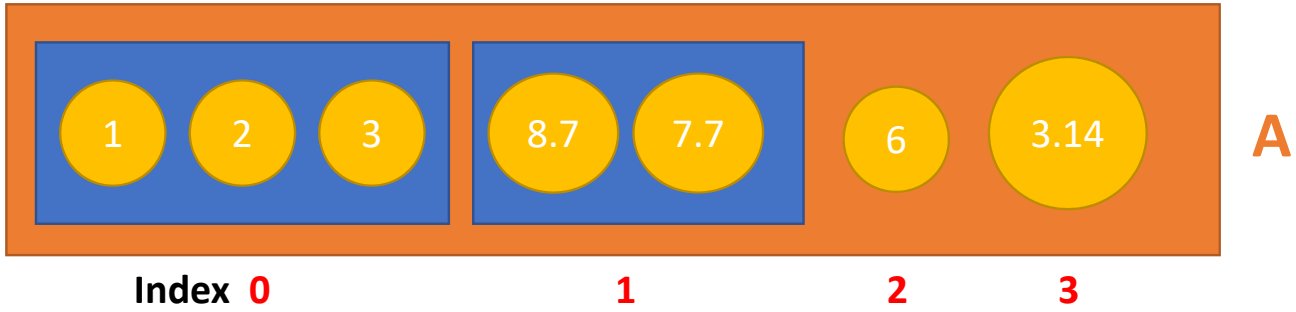
X = A.index(3)	Get the first index of 3 in A	X = 1
X = A.count(3)	Get count of how many times *3* in A	X = 2
X = A.pop()	Delete the last element and return value	X = 3
A.remove(3)	Remove the first element 3 in A.	A = [1, 2, 5, 4, 6, 3]
X = len(A)	Get the count of element in A	X = 7
A.reverse()	Reverse 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 we have A = [[1,2,3], [8.7, 7.7], 6, 3.14]

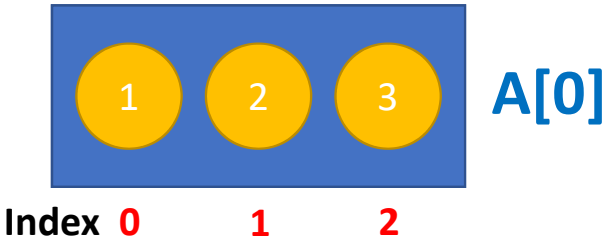
- **X = len(A)**, X = ??

  Ans: **X = 4**



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

  Ans: **Y = 3**



Tuple ()

- We **can't modify** the tuple in **Python**.
- **Ex**
  - If we have A = (1,2,3,4), then we **can't** use A.**append**(5) or others function to **add or delete** the item in tuple.

- We can convert between tuple and 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 assignment to one variable

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

A = 1, 2, 3, 4 → **A = (1, 2, 3, 4)**

- **Question:**

  Can we do this? A, B= 1, 2, 3, 4

  Ans: No

Iterate List and Tuple

- **Python can iterate item in list and tuple directly.**
- **Ex:**

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

```
1
2
3
4
```

- **You can also iterate the 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 list.

```
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 combine the elements in the input objects one by one, and return with **tuple**.
- **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 size of the inputs **aren't the same**, then it would only combine **minimal size** of times.

```
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 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
```

- You can add 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 there are two element with the same Key name, the **newer one** will **replace** the **first one**.
  - **Ex**

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

→ **50**

- The indexes in dict are **random**, so we can't use index to get the value.
  - **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 function

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

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

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

Set

- Set in Python is similar to the concept of set in mathematic.
- Three way to create a set object:

```
s1=set() #create a 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'}
```

- Notice that, although there’re two ‘P’ in s2 and s3, but since it’s “set”, it will only has one ‘P’, which means ‘P’ is in the set

Add/Remove element to set

- You can add/remove element 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

- You can compare two sets with comparison operators.
- Guess what the answer?

```
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)
```

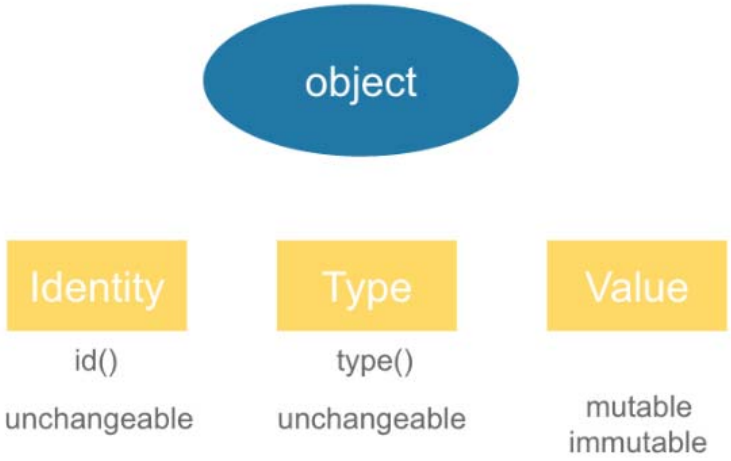
```
True
False
False
True
False
False
```

Other useful function

```
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: All the elements which only one side (set1 or set2) have.	print(B ^ C) #{'Truck', 'Airplane'}

Mutable/Immutable



- In Python, every object contains one identity(address), type, and value. Some objects of value can't be changed, others can 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 see that **lists** are **mutable**, so after we changed the value, its address didn't changed. Which means we **can** change the value of the same object of list!

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++**. As example, `a = 1` will **create a object with value 1**, then let **object a points to that Object with value 1**. So both objects'(1 and 2) value **won't** be changed!

Mutable/Immutable

- **Case3**

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

a[0] = 1

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

```
4317018016

# error with a[0] = 1

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

- As you can see, immutable objects aren't mean that "they can't changed at all"! If they contain some mutable objects, then the values of those mutable objects still can be changed.

Pitfall

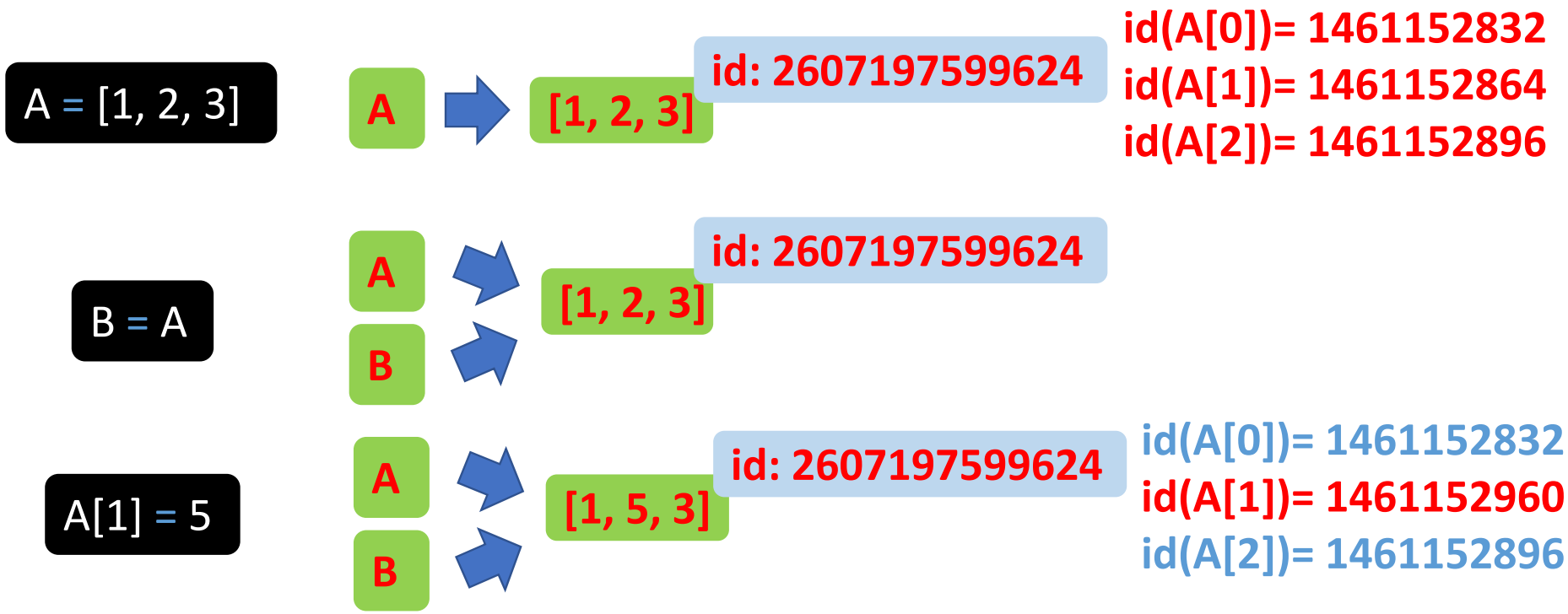
- **Guess what's the value of B?**

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

→ B = **[1, 5, 3]**

- **As we mentioned earlier, in Python** `B = A` **will create object B points to A. And A is points to the object with [1, 2, 3]. And since A is a list which is mutable. So after modifying elements in A, the objects A was originally pointing to will be modified and A won't point to any other new object.**

Pitfall


A = [1, 2, 3]
A
[1, 2, 3]
id: 2607197599624
id(A[0])= 1461152832
id(A[1])= 1461152864
id(A[2])= 1461152896
B = A
A
B
[1, 2, 3]
id: 2607197599624
A[1] = 5
A
B
[1, 5, 3]
id: 2607197599624
id(A[0])= 1461152832
id(A[1])= 1461152960
id(A[2])= 1461152896

Pitfall

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

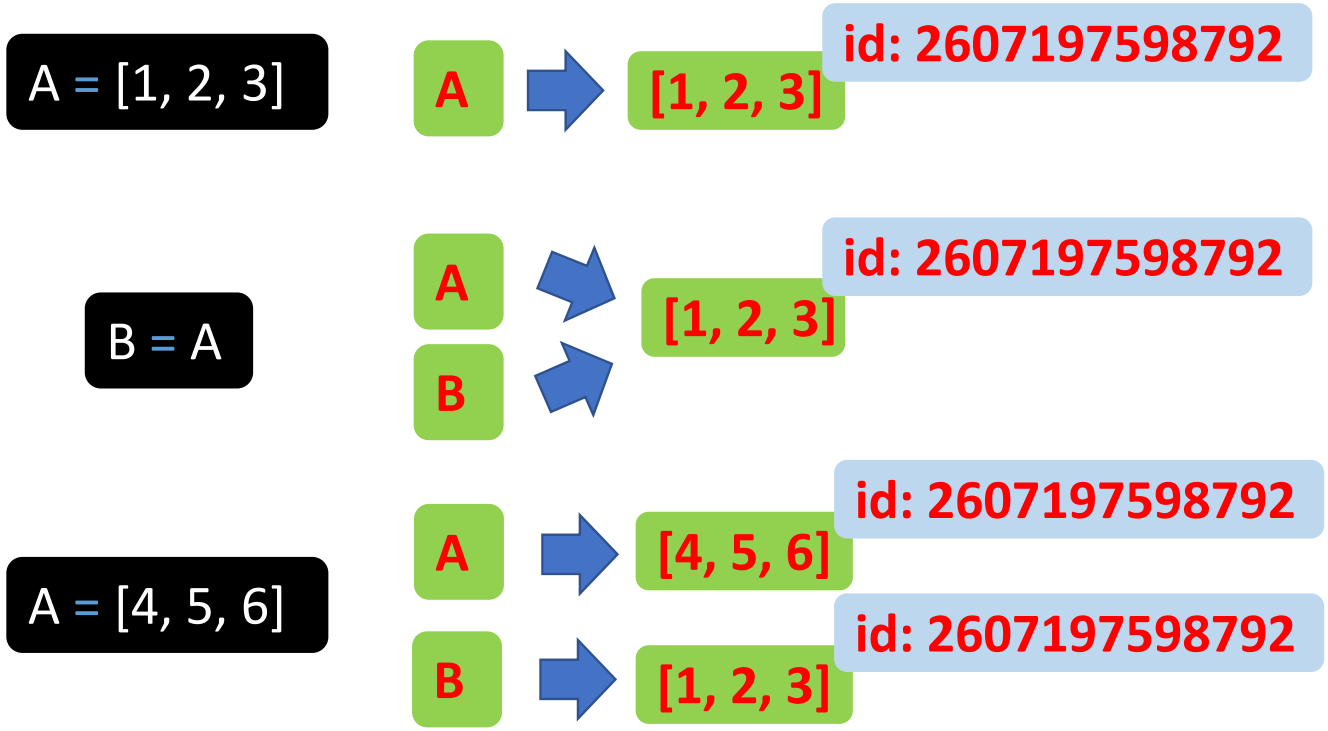
```
A = [1, 2, 3]
B = A
A = [4, 5, 6]
```

→

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

- **Since A = [4, 5, 6] is making A point to the new object with value [4, 5, 6]. So the address of A will be changed, but the original object is still there, so B's value won't be changed.**

Pitfall


A = [1, 2, 3]
A
[1, 2, 3]
id: 2607197598792
B = A
A
B
[1, 2, 3]
id: 2607197598792
A = [4, 5, 6]
A
[4, 5, 6]
id: 2607197598792
B
[1, 2, 3]
id: 2607197598792

List: copy()

- Then what can we do when we don't want B be changed when we modified 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]**
id A, B, C, D, E were all different

4

Python Function, Class, Import

Function

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

- In **Python:** function names **don't** need to add **return type name**. Neither do parameters.

- In **Python:** function 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
```

Function parameter with default value

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

- Just like **C/C++**, you can add some default value for the parameter, but you have to put them to the **rightmost** of the parameter list. Or else you will get error message.
- **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 with unknown amount of parameter

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

- Sometimes, you don't know how many input the function will get, then you can add * in front of the parameter name. Python will create a list for that 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
```

Some useful build-in function

		Example	Answer
abs(x)	Return the absolute value of a number	X= abs(-2)	X= 2
int (x) float(x)	Return a 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 *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'>

Class

- **Python** has class just like **C++**.
- Every function in class must add "**self**" to be **the first parameter**.
- Using class variable must add "**self.**" in front of the variable name.
- **Private** variables need to add "**__**"(**double underscore**) in front of the variable name.

Class

- **Ex**

```
class Shape():  # ()is not necessary when you aren't using 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() )
```

Class inheritance

- **In Python**: you can put **father's** class name **in the bracket** behind **child's** name to inherite.

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

- **Python** supports multiple class inheritance just like **C++**!

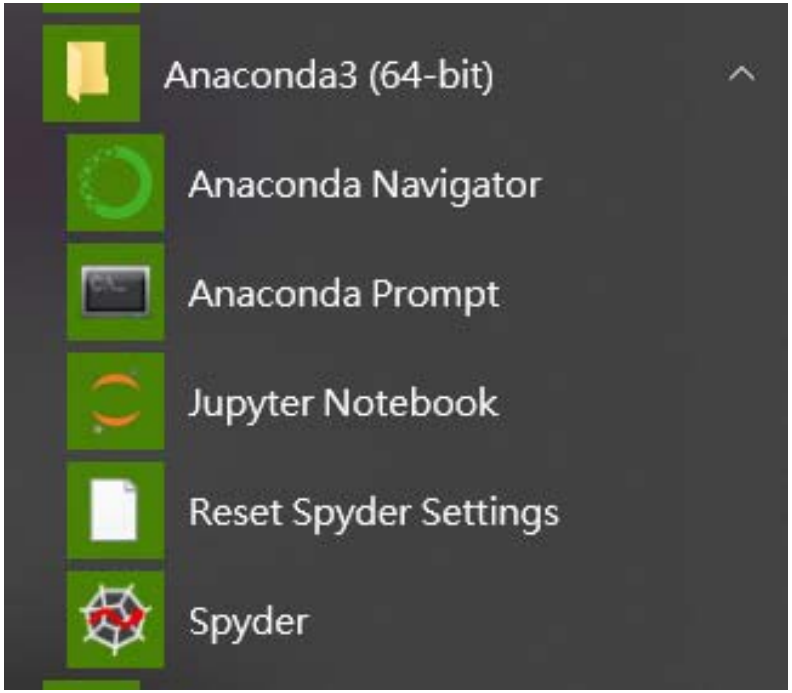
Class inheritance

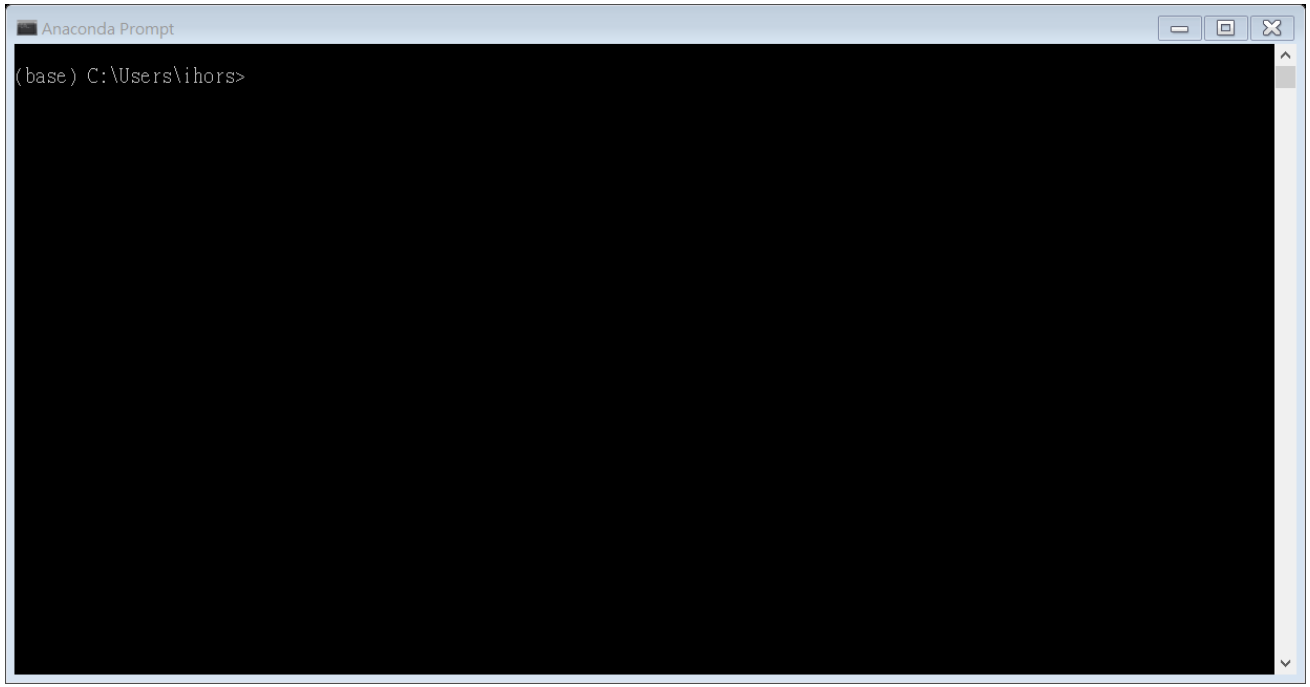
- **Ex**

```
from math import pi

class Circle(Shape):
    def __init__ (self, radius): # class initialization
        self.__radius = radius # private variable radius

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

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

49.34802200544679

Install Package – Open Anaconda prompt





Typing pip install The_package_you_want

Anaconda Prompt

```
(base) C:\Users\ihors>pip install scikit-learn
Requirement already satisfied: scikit-learn in c:\users\ihors\anaconda3\lib\site-packages (0.20.2)
Requirement already satisfied: scipy>=0.13.3 in c:\users\ihors\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: numpy>=1.8.2 in c:\users\ihors\anaconda3\lib\site-packages (from scikit-learn) (1.12.0)
You are using pip version 19.0.1, however version 19.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

(base) C:\Users\ihors>
```

Import

- **In Python**, there're so many powerful modules created by others. Then you can use `import module_name` to import them. Just like "**#include**" in **C**.
- **Ex**

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

- if you just want to import **some functions** within the module…

```
from module import [name1, name2, ...]
```

- If you use this method, and want to use the function, then you don't need to add module name
- **Ex**

```
from math import pi
print (pi) #You don't need to use  "math.pi "
```

→ **3.141592653589793…**

Import

- If the module name is **too long** or you **often need to use it**, then you can use…

```
import module as new_name
```

- **Ex**

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

dir

- **dir(object)** can return return a list of valid attributes(variables, methods…) of the object. You don't need to fill in the parameter object
- **Ex**

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

```
['In', 'Out', '_', '__', '___', '__builtin__', '__builtins__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1', '_ih',
'_ii', '_iii', '_oh', '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', '__class__', '__delattr__',
'__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__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

Python Numpy

Overview

- **Numpy** is a powerful module which is similar to the regular Python "**list**" data structure.

- A NumPy array is a multidimensional array of objects **with the same type**.

- NumPy supports **parallel processing**, so it could be much faster if you're processing **multiple big dimension arrays at once**.

Some useful variables to get memory layout

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

		Example	Answer
ndarray.**shape**	dimensions information about the array.	X.shape	(2, 3) #2*3 array
ndarray.**ndim**	the dimensions of the array.	X.ndim	2
ndarray.size	the total number of elements of the array.	X.size	6
ndarray.**dtype**	type of the elements in the array. numpy.int32, numpy.int16, and numpy.float64 are some examples.	X.dtype	int32
ndarray.**itemsize**	the size in bytes of each element of the array.	X.itemsize	4
ndarray.**item**	Get the value at specific position	X.item((0, 0)) X.item(1,2)	0 5

Changing the value of Numpy array

```
X[1,1] = 7
```

```
X = [[ 1  2  3 ]
     [ 4  7  6]]
```

Creating a numpy array

- The basic one, you can use `np.array([number_array])` to create a numpy array.
- a = np.array(1,2,3,4) #**False**
- a = np.array([1,2,3,4]) #**Correct**

- The type of the array can be explicitly specified at creation time:
- **Ex**

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

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

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 number	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.]]

Operate with numpy array

- In before, if you want to add two different **list** to a new list. It won't be easy!

- But if you're using **numpy array**, you can simply use **a+b** to get the answer you want.

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

c = [7 9 11 13 15]

6

Python
Matplotlib, Seaborn

Overview

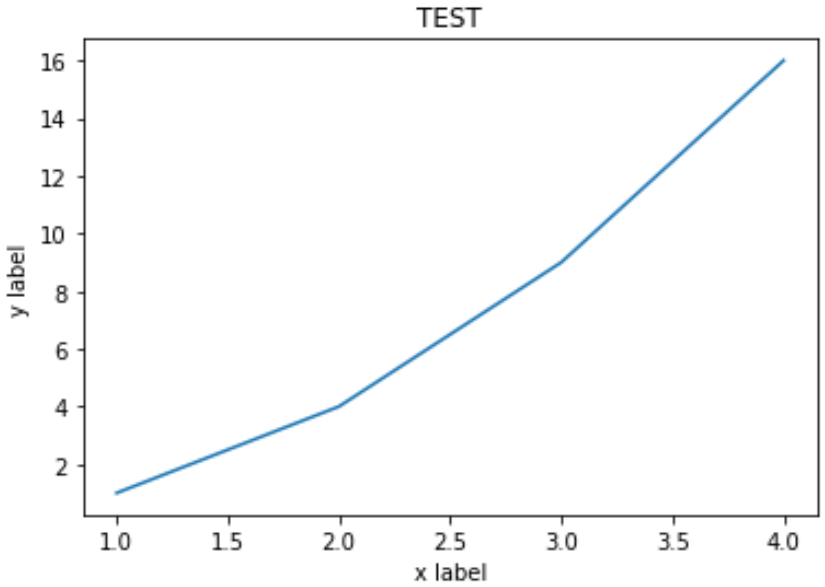
- **Matplotlib, Seaborn** are all powerful module for plotting! Matplotlib is the basic one, while Seaborn can create more beautiful and detailed plot.

- Recommend using `import matplotlib.pyplot as plt` `import seaborn as sns`

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

Line Chart

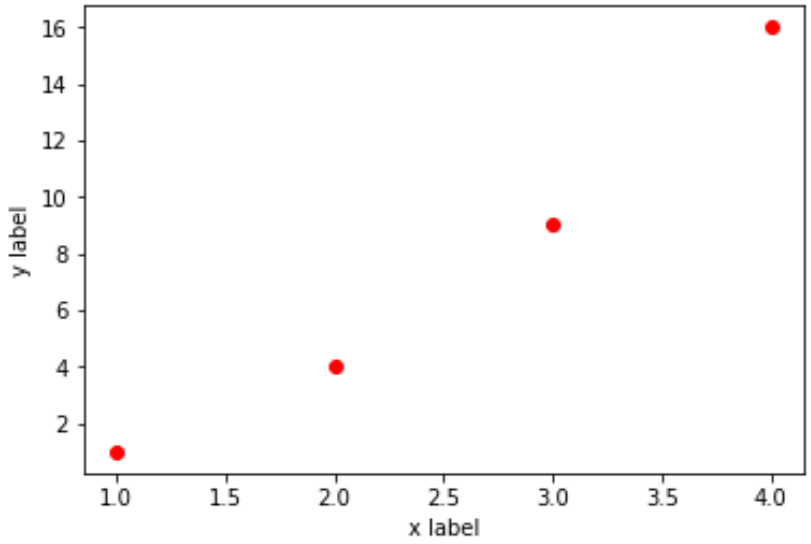
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.ylabel('y label')
plt.xlabel('x label')
plt.title('TEST')
#plt.show()
```



Dot Chart

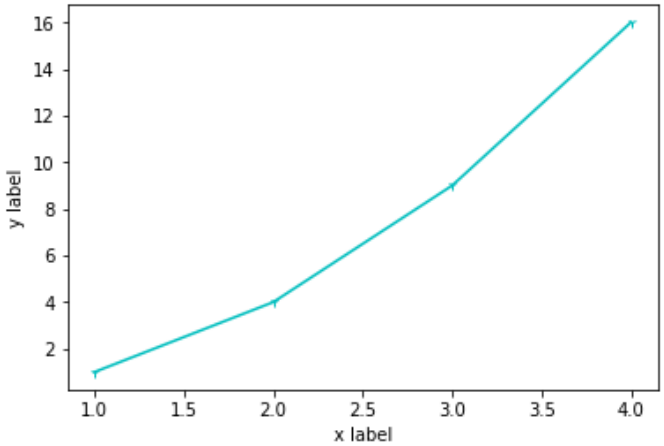
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.ylabel('y label')
plt.xlabel('x label')
#plt.show()
```

color and markers



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 label')
#plt.show()
```



Supported colors and markers

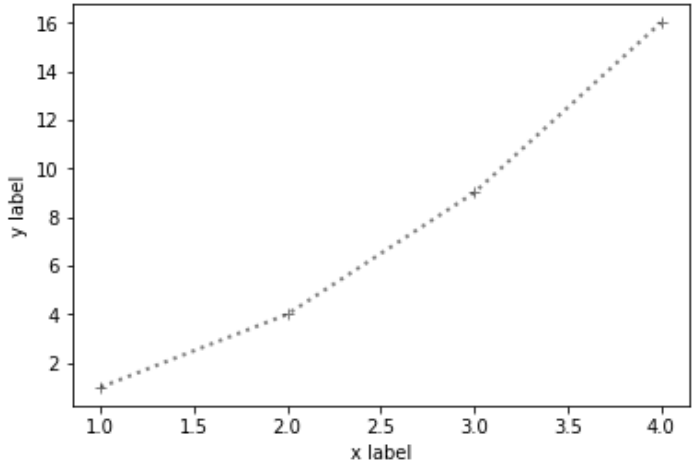
'b'	blue	'm'	magenta
'g'	green	'y'	yellow
'r'	red	'k'	black
'c'	cyan	'w'	white

'.'	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 option

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

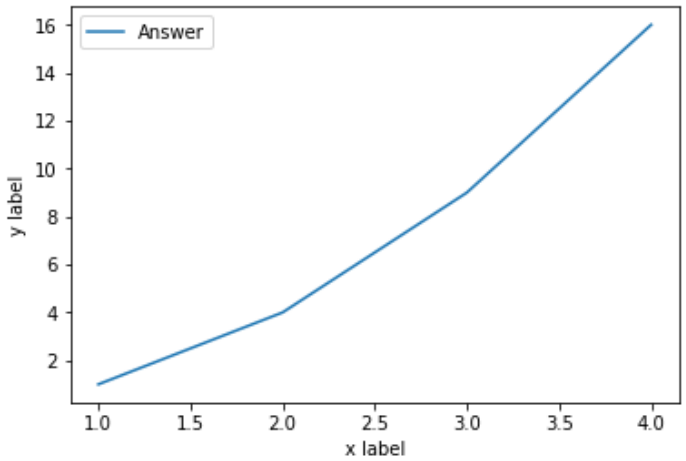
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], marker = '+', color = 'k' ,
lw = 2, ls = ':', alpha = 0.5)
plt.ylabel('y label')
plt.xlabel('x label')
#plt.show()
```



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'

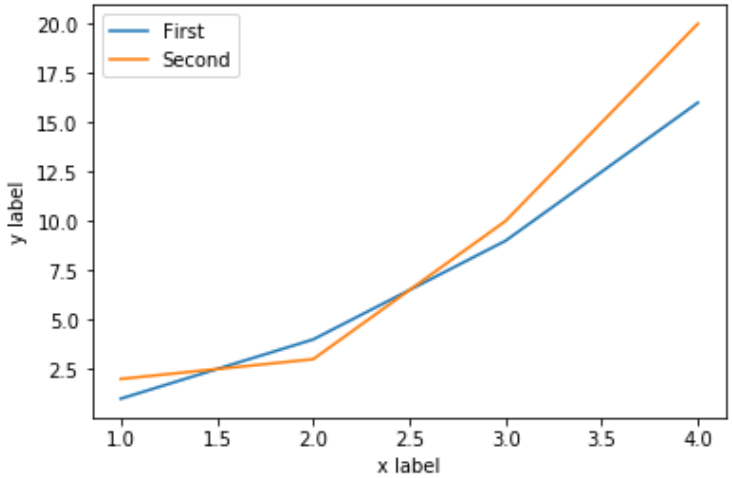
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], label = 'Answer')
plt.ylabel('y label')
plt.xlabel('x label')
plt.legend()
#plt.show()
```



If we want to add labe, then we need to add "plt.legend()"

Multiple line in one plot

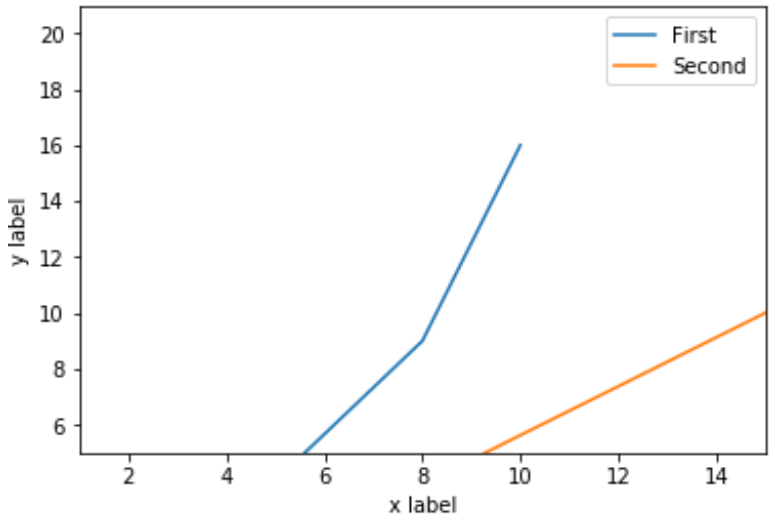
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], label = ‘First')
plt.plot([1, 2, 3, 4], [2, 3, 10, 20], label = 'Second')
plt.ylabel('y label')
plt.xlabel('x label’)
plt.legend()
#plt.show()
```



plt.xlim(), plt.ylim()

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

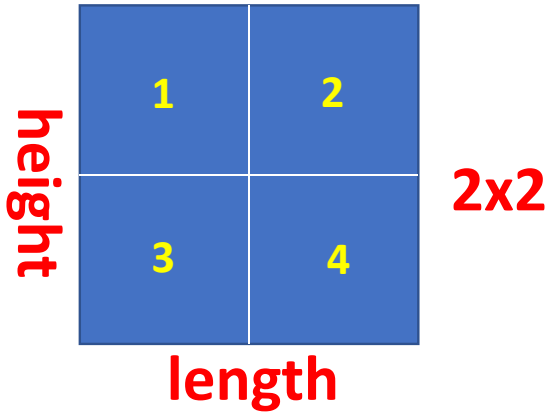
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1, 5, 8, 10], [1, 4, 9, 16], label = ‘First’)
plt.plot([1, 7, 15, 20], [2, 3, 10, 20], label = 'Second')
plt.ylabel('y label')
plt.xlabel('x label’)
plt.xlim(1,15)
plt.ylim(5)
plt.legend()
#plt.show()
```



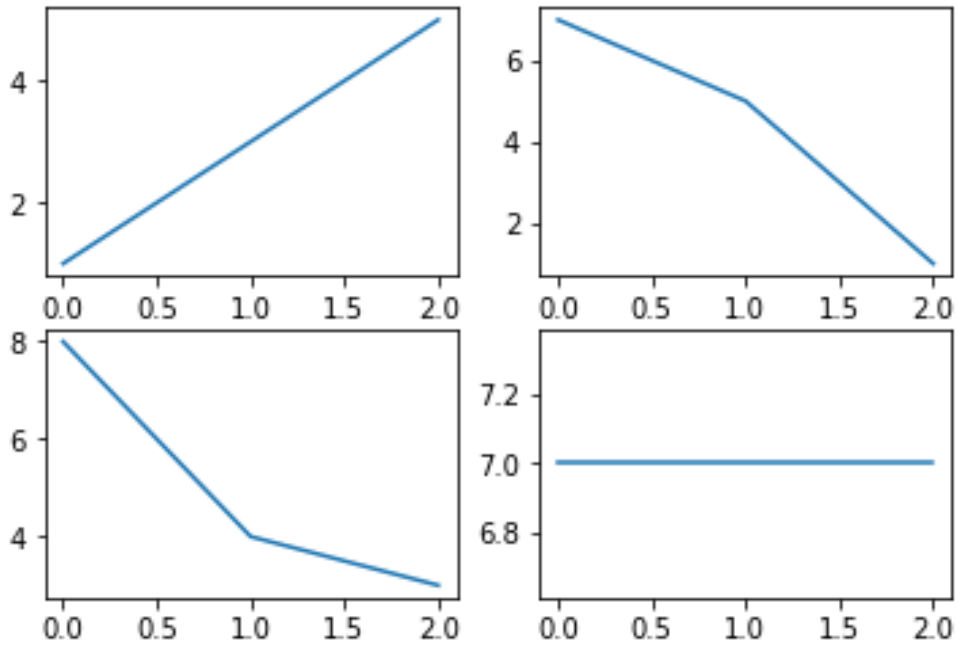
Subplot

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

- Can be shorten as plt.**subplot**(height length **index**)

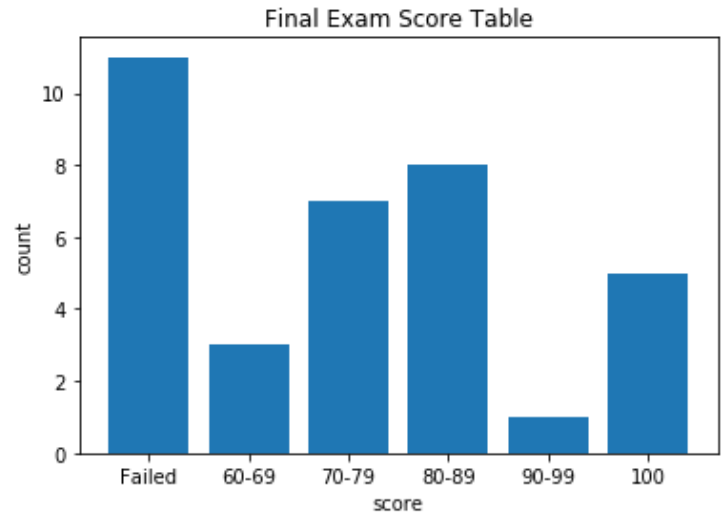



```
import matplotlib.pyplot as plt
%matplotlib inline
plt.subplot(2, 2, 1)
plt.plot([0, 1, 2], [1, 3, 5])
plt.subplot(2, 2, 2)
plt.plot([0, 1, 2], [7, 5, 1])
plt.subplot(223)
plt.plot([0, 1, 2], [8, 4, 3])
plt.subplot(224)
plt.plot([0, 1, 2], [7 ,7, 7])

# plt.show()
```



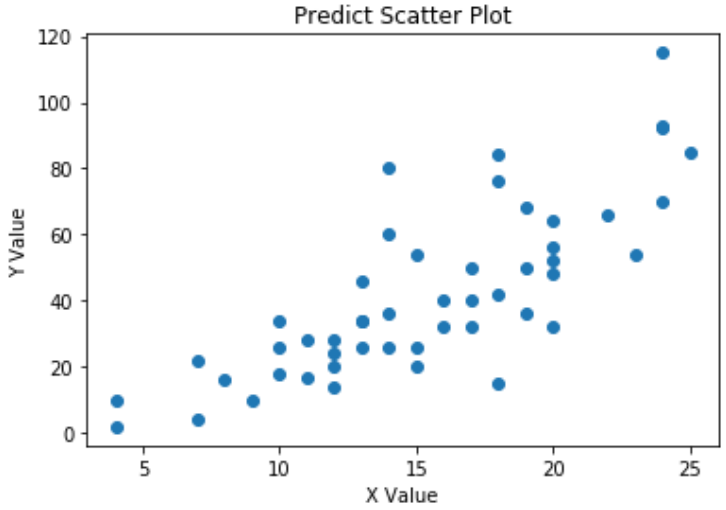
Bar Plot

```
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()
```



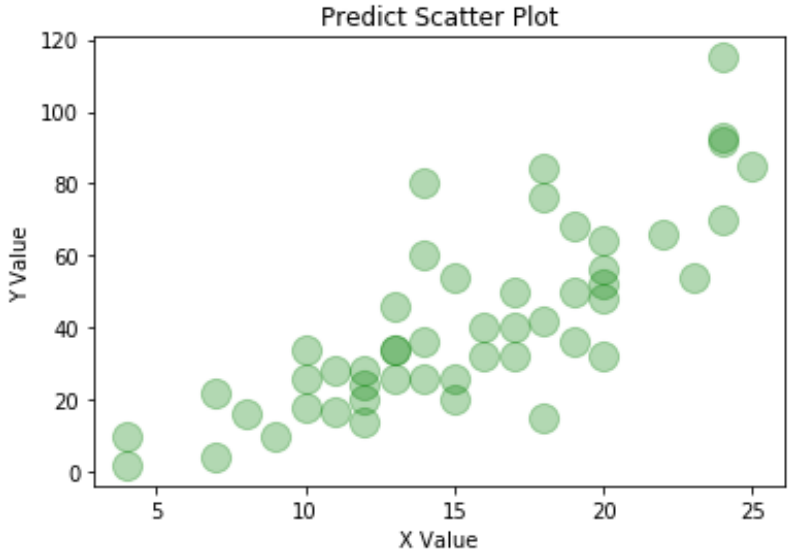
Scatter Plot

```
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()
```

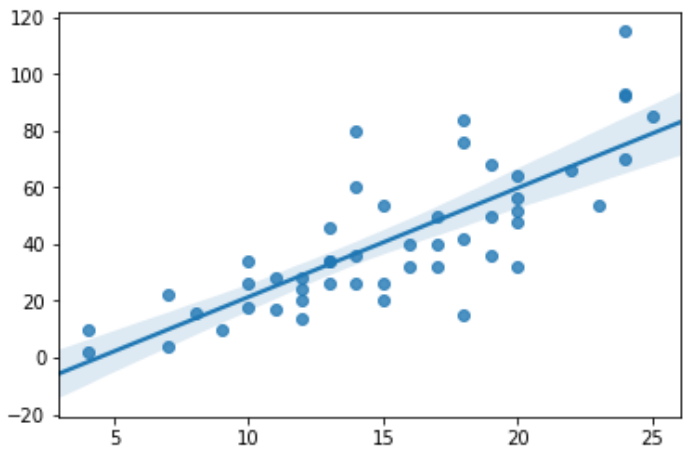
size



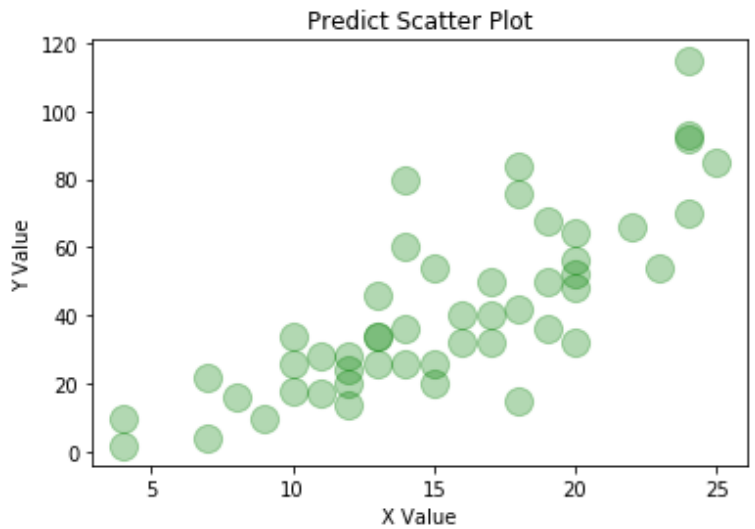
Scatter Plot with Seaborn – regplot()

With regression line in default

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

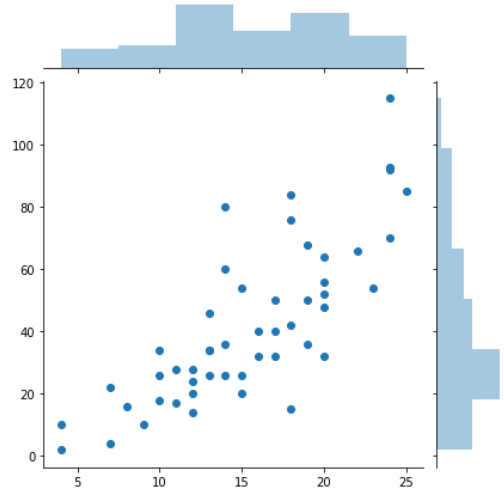



```
import seaborn as sns
plt = sns.regplot(x, y,
scatter_kws={"color":"g","alpha":0.3,"s":200},
 fit_reg = False)
plt.set_title('Predict Scatter Plot')
plt.set_ylabel('Y Value')
plt.set_xlabel('X Value')
```

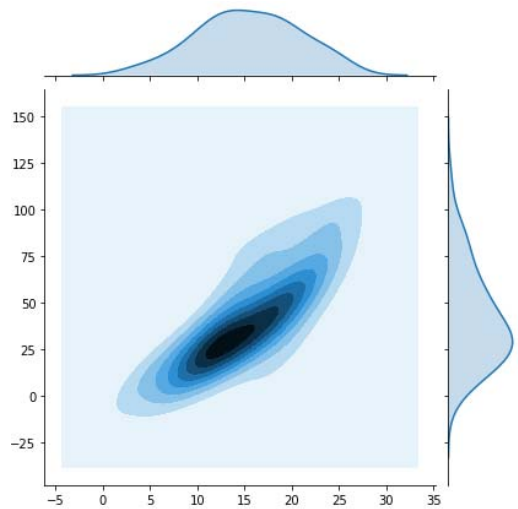


Marginal plot with Seaborn – jointplot()

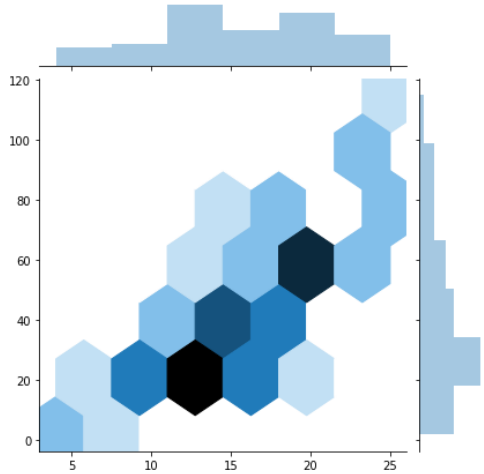
```
import seaborn as sns
sns.jointplot(x, y)
#or sns.jointplot(x, y, kind = 'scatter')
```




```
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()
```