TRƯỜNG ĐẠI HỌC NGOẠI NGỮ - TIN HỌC TP. HỒ CHÍ MINH
HO CHI MINH CITY UNIVERSITY OF FOREIGN LANGUAGES - INFORMATION TECHNOLOGY

Python cho Máy học

Biên soạn: **ThS. Vũ Đình Ái***(aivd@huflit.edu.vn)*

Cập nhật: tháng 09/2023

Nội dung

- Giới thiệu python
- Các thư viện Numpy, Pandas, Matplotlib,
- Bài tập

Giới thiệu Python

- Python là một ngôn ngữ lập trình bậc cao.
- Những dòng Code của Python thường được cho là gần giống với mã giả, và có thể thực hiện những ý tưởng trong một vài dòng Code và rất dễ đọc.
- Ví dụ về thuật toán sắp xếp Quicksort trong Python:

```python
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([3,6,8,10,1,2,1]))
```

Python

```
[1, 1, 2, 3, 6, 8, 10]
```

Giới thiệu Python

- Kiểu dữ liệu
- Containers
  - Lists (Kiểu danh sách)
  - Dictionaries (Kiểu từ điển)
  - Sets (Kiểu set)
  - Tuples (Kiểu Tuple)
- Functions (hàm)
- Classes (lớp)

Giới thiệu Python

- **Kiểu dữ liệu :** Cũng giống như các ngôn ngữ khác, Python có các kiểu dữ liệu cơ bản bao gồm: integers, floats, booleans, và strings.
  - **Numbers (Kiểu số)** : Kiểu integers (số nguyên) / floats (số thực) hoạt động giống như với các ngôn ngữ khác. (**Chú ý:** không giống như các ngôn ngữ khác, Python không có các phép toán x++ hay x--)
  - **Booleans** : Python thực hiện tất cả các phép toán thông thường trong đại số Boolean.(Chú ý: có thể sử dụng các từ tiếng Anh thay cho các ký tự `&&`, `||`, v.v...)
  - **Strings**

Giới thiệu Python

```python
### Kiểu số nguyên (integers)
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x) # Prints "3"
print(x + 1) # Thực hiện phép cộng; prints "4"
print(x - 1) # Thực nhiên phép trừ; prints "2"
print(x * 2) # Thực hiện phép nhân; prints "6"
print(x ** 2) # Thực hiện phép mũ; prints "9"
x += 1
print(x) # Prints "4"
x *= 2
print(x) # Prints "8”
```

Python

```
<class 'int’>
3
4
2
6
9
4
8
```

Giới thiệu Python

```python
### Kiểu số thực (floats)
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25”
```

Python

```
<class 'float’>
2.5 3.5 5.0 6.25
```

```python
# Kiểu booleans
t = True
f = False
print(type(t))
```

Python

```
<class 'bool'>
```

```python
print(t and f) # Logical AND; prints "False"
print(t or f) # Logical OR; prints "True"
print(not t) # Logical NOT; prints "False"
print(t != f) # Logical XOR; prints "True”
```

Python

```
False
True
False
True
```

Giới thiệu Python

```python
hello = 'hello' # Chuỗi ký tự sử dụng dấu nháy ' '
world = "world" # Hoặc sử dụng dấu nháy " "
print(hello) # Prints "hello"
print(len(hello)) # lấy độ dài của chuỗi; prints "5"
hw = hello + ' ' + world # Kết nối chuỗi
print(hw) # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12) # định dạng chuỗi kiểu sprintf
print(hw12) # prints "hello world 12”
```

Python

```
hello
5
hello world
hello world 12
```

```python
s = "hello"
# Viết hoa chữ cái đầu tiên
print(s.capitalize())
# Viết hoa toàn bộ chữ cái
print(s.upper())
# căn giữa
print(s.center(30))
# loại bỏ dấu cách thừa
print(' world '.strip())
```

Python

```
Hello
HELLO
          hello
world
```

Giới thiệu Python

- **Containers:** Trong Python có các kiểu chứa các dữ liệu như:
  - Lists
  - Dictionaries
  - Sets
  - Tuples.

Giới thiệu Python

- **Lists** :List trong Python là một dạng dữ liệu cho phép lưu trữ nhiều kiểu dữ liệu khác nhau, và chúng ta có thể thay đổi kích thước của nó:

```python
xs = [3, 1, 2] # Tạo một list
print(xs, xs[2]) # Prints "[3, 1, 2] 2"
print(xs[-1]) # Các số âm thể hiện việc truy cập ngược từ cuối; prints "2"
xs[2] = 'foo' # List có thể chứa các phần tử khác nhau
print(xs) # Prints "[3, 1, 'foo']"
xs.append('bar') # Thêm phần tử mới vào list
print(xs) # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop() # Xoá phần tử khỏi list
print(x, xs) # Prints "bar [3, 1, 'foo']”
```

Python

```
[3, 1, 2] 2
2
[3, 1, 'foo']
[3, 1, 'foo', 'bar']
bar [3, 1, 'foo']
```

Giới thiệu Python

- **Slicing (cắt List):** ngoài việc truy cập từng phần tử trong List, Python cung cấp cú pháp ngắn gọn để truy cập nhiều phần tử (sublists).
- **Chú ý:** việc cắt List này không ảnh hưởng đến List ban đầu.

```python
nums = list(range(5)) # tạo một list từ 0 đến 4
print(nums) # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4]) # lấy phần tử có vị trí 2 đến 4 (exclusive); prints "[2, 3]"
print(nums[2:]) # lấy phần tử có vị trí 2 đến hết; prints "[2, 3, 4]"
print(nums[:2]) # lấy phần tử từ đầu đến phần tử có vị trí 2 (exclusive);
prints "[0, 1]"
print(nums[:]) # lấy toàn bộ list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1]) # chỉ số có thể là số âm; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9] # thay đổi nhiều phần tử cùng lúc
print(nums) # Prints "[0, 1, 8, 9, 4]”
```

Python

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

Giới thiệu Python

- **Vòng lặp:**truy cập các phần tử của List bằng vòng lặp.

```python
animals = ['cat', 'dog', 'monkey']
for animal in animals:
print(animal)
```

Python

```
cat
dog
monkey
```

- truy cập vào chỉ mục của từng phần tử của List, sử dụng hàm `enumerate`:

```python
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
print('#%d: %s' % (idx + 1, animal))
```

Python

```
#1: cat
#2: dog
#3: monkey
```

Giới thiệu Python

- **List comprehensions:**trong quá trình lập trình, chúng ta thường xuyên muốn thay đổi giá trị của các phần tử đồng loạt.
- Ví dụ: nhân giá trị của các phần tử bên trong List với 2.
  - Cách 1: chúng ta có thể sử dụng vòng lặp:

```python
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
squares.append(x ** 2)
print(squares)
```

Python

```
[0, 1, 4, 9, 16]
```

  - Cách 2: sử dụng List Comprehensions

```python
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)
```

Python

```
[0, 1, 4, 9, 16]
```

Giới thiệu Python

- có thể sử dụng thêm các điều kiện khi dùng List Comprehension:

```python
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)
```

Python

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

Giới thiệu Python

- **Dictionaries**: Dictionary trong Python là một tập hợp các cặp (key, value) không có thứ tự. Nó là một container mà chứa dữ liệu, được bao quanh bởi các dấu ngoặc móc đơn {}.

```python
d = {'cat': 'cute', 'dog': 'furry'} # tạo một Dict
print(d['cat']) # Lấy giá trị có key = cat; prints "cute”
```

Python

```
cute
```

```python
print('cat' in d) # kiểm tra cat có trong dict ?
```

Python

```
True
```

```python
d['fish'] = 'wet' # thêm một cặp key, value trong Dicts
print(d['fish']) # Prints "wet”
```

Python

```
wet
```

Giới thiệu Python

```python
# Sẽ bị lỗi nếu `key` không tồn tại trong Dict
print(d['monkey’])
```

Python

```
---------------------------------------------------------------------------
-- KeyError                                  Traceback (most recent call
last) <ipython-input-20-b78aff1f0d89> in <module>
          1 # Sẽ bị lỗi nếu `key` không tồn tại trong Dict
---->     2 print(d['monkey’])
KeyError: 'monkey'
```

```python
# Thay vào đó chúng ta sẽ sử dụng lệnh `get`
print(d.get('monkey', 'N/A')) # Lấy giá trị có key = 'monkey' không có in
ra 'N/A'; prints "N/A"
print(d.get('fish', 'N/A')) # Lấy giá trị có key = 'fish'không có in ra
'N/A'; prints "wet”
```

Python

```
N/A
wet
```

Giới thiệu Python

- **Vòng lặp**:chúng ta có thể dễ dàng in các giá trị trong Dict bằng vòng lặp.

```python
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
legs = d[animal]
print('A %s has %d legs' % (animal, legs))
```

Python

```
A person has 2 legs
A cat has 4 legs
A spider has 8 legs
```

- **Dictionary comprehensions**: Cũng giống như List Comprehensions.

```python
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)
```

Python

```
{0: 0, 2: 4, 4: 16}
```

Giới thiệu Python

- **Sets** là một kiểu chứa dữ liệu đặc biệt, nó chỉ chứa các giá trị riêng biệt không trùng lặp nhau.

```python
animals = {'cat', 'dog'}
print('cat' in animals) # kiểm tra phần tử trong set; prints "True"
print('fish' in animals) # prints "False"
animals.add('fish') # thêm phần tử vào set
print('fish' in animals) # Prints "True"
print(len(animals)) # số lượng bên trong set; prints "3"
print(animals)
```

Python

```
True
False
True
3
{'fish', 'dog', 'cat'}
```

Giới thiệu Python

```python
animals.add('cat') # thêm phần tử vào set, nhưng `cat` đã có nên set không thay đổi
print(len(animals)) # Prints "3"
print(animals)
```

Python

```
3
{'fish', 'dog', 'cat'}
```

```python
animals.remove('cat') # xoá phần tử trong set
print(len(animals)) # Prints "2"
print(animals)
```

Python

```
2
{'fish', 'dog'}
```

Giới thiệu Python

- **Vòng lặp**: Bạn cũng có thể truy cập các phần tử của Set bằng vòng lặp.

```python
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
print('#%d: %s' % (idx + 1, animal))
```

```
#1: fish
#2: dog
#3: cat
```

- **Set comprehensions:** giống như Lists và Dictionaries, chúng ta cũng có thể tạo Set bằng phương thức Set Comprehensions:

```python
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)
```

```
{0, 1, 2, 3, 4, 5}
```

Giới thiệu Python

- **Tuples** trong Python là một kiểu dữ liệu dùng để lưu trữ các đối tượng không thay đổi về sau (giống như hằng số).

```python
day = ('monday', 'tuesday', 'wednesday', 'thursday')
print(type(day)) # Lấy kiểu dữ liệu
print(day) # in toàn bộ tuples
print(day[1]) # truy cập vào phần tử index = 1
```

Python

```
<class 'tuple'>
('monday', 'tuesday', 'wednesday', 'thursday')
tuesday
```

```python
# Thực hiện thay đổi giá trị Tuples sẽ bị lỗi
day[1] = 2
```

Python

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
last) <ipython-input-30-d5a2c290c4ce> in <module>
         1 # Thực hiện thay đổi giá trị Tuples sẽ bị lỗi
---->    2 day[1] = 2
TypeError: 'tuple' object does not support item assignment
```

Giới thiệu Python

- **Functions** :Hàm trong Python được định nghĩa bằng từ khoá `def`. Ví dụ:

```python
def sign(x):
    if x > 0:
        return 'positive’
    elif x < 0:
        return 'negative’
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
```

Python

```
negative
zero
positive
```

Giới thiệu Python

- thêm đối số trong hàm như sau:

```python
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True) # Prints "HELLO, FRED!”
```

Python

```
Hello, Bob
HELLO, FRED!
```

Giới thiệu Python

- **Class**: Cú pháp để sử dụng Class trong Python rất đơn giản:

```python
class Greeter(object):

# Constructor
    def __init__(self, name):
        self.name = name # Cho giá trị của name vào self.name

# Instance method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred') # Khởi tạo Class g với đầu vào `Fred`
g.greet() # Gọi hàm greet() khi loud = False; prints "Hello, Fred"
g.greet(loud=True) # Gọi hàm greet() khi loud = True; prints "HELLO, FRED!”
```

Python

```
Hello, Fred
HELLO, FRED!
```

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, ...]
dictionary = { key1:value1, key2:value2, ...}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

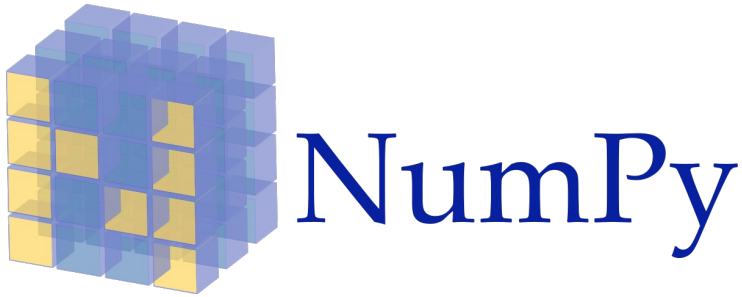
Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: x,y stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

Thư viện numpy

- Numpy là một thư viện của Python hỗ trợ việc tính toán trên mảng nhiều chiều. Một mảng Numpy là một tập hợp các giá trị cùng kiểu dữ liệu và được đánh số bằng các số nguyên dương.
- Numpy là Module quan trọng cho việc sử lý dữ liệu và có thể chuyển đổi qua kiểu dữ liệu Tensor trong Tensorflow và Pytorch.
- Nội dung
  - Tạo mảng Numpy (ndarray)
  - Kiểu dữ liệu (Datatypes)
  - Array Indexing
  - Phép toán trên mảng
  - Broadcasting

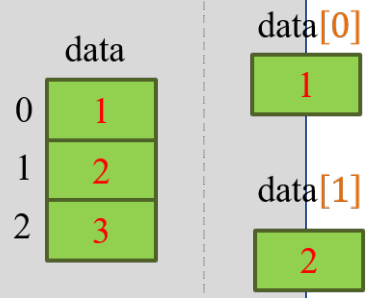


Thư viện numpy

- **Tạo mảng Numpy (ndarray)**
  - Tạo ndarray từ List

```python
# Tạo ndarray từ list

import numpy as np

# tạo list
l = list(range(1, 4))

# tạo ndarray
data = np.array(l)

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

Python




```
[1 2 3]
1 2
```

Thư viện numpy

- Sử dụng thuộc tính `shape` và hàm `type()` cho mảng Numpy

```python
# Tạo ndarray từ list

import numpy as np

# tạo list
l = list(range(1, 4))

# tạo ndarray
data = np.array(l)

print(data)
print(type(data))
print(type(data[0]))
print(data.shape)
```

Python

```
[1 2 3]
<class 'numpy.ndarray'>
<class 'numpy.int64'> (3,)
```

Thư viện numpy

- Thay đổi `shape` của mảng sử dụng `reshape`

```python
# thay đổi shape của một mảng
import numpy as np

arr1 = np.arange(12)
print(arr1.shape)

arr2 = arr1.reshape((3, 4))
print(arr2.shape)
```

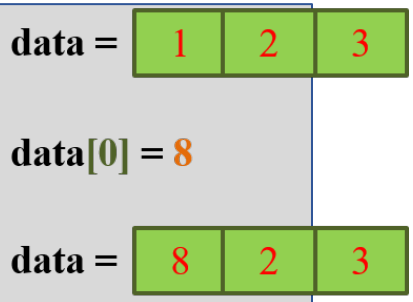
Python

```
(12,)
(3, 4)
```

- Thay đổi giá trị của một phần tử

```python
# thay đổi giá trị phần tử
import numpy as np

l = list(range(1, 4))
data = np.array(l)
print(data)

data[0] = 8
print(data)
```

Python



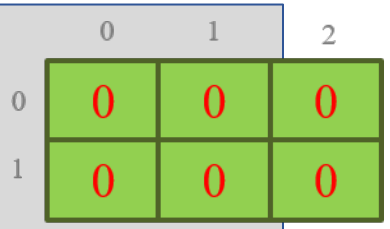

```
[1 2 3]
[8 2 3]
```

Thư viện numpy

- Tạo ndarray với hàm `zeros()`

```python
# tạo một numpy array với tất cả các phần tử là 0
import numpy as np

# shape: 2 dòng, 3 cột
arr = np.zeros((2, 3))
print(arr)
```

Python

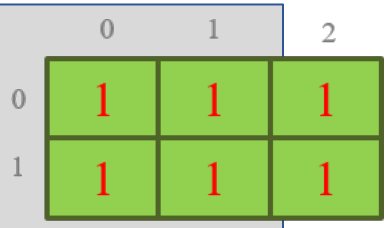



```
[[0. 0. 0.]
 [0. 0. 0.]]
```

- Tạo ndarray với hàm `ones()`

```python
# tạo một numpy array với tất cả phần tử là 1
import numpy as np

# numpy.ones(shape, dtype=None, order='C')
# shape: 2 dòng, 3 cột
arr = np.ones((2, 3))
print(arr)
```

Python




```
[[1. 1. 1.]
 [1. 1. 1.]]
```

Thư viện numpy

- Tạo ndarray với hàm `full()`

```python
# tạo một numpy array với tất cả phần tử là hằng số K
import numpy as np

# numpy.full(shape, fill_value, dtype=None, order='C')
# shape: 2 dòng, 3 cột - với K = 9
arr = np.full((2, 3), 9)
print(arr)
```

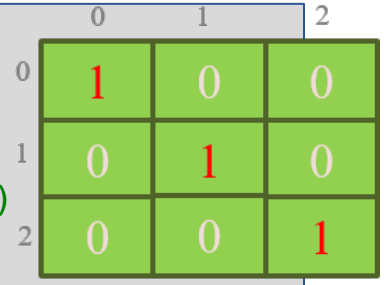
Python


```
[[9 9 9]
 [9 9 9]]
```

- Tạo ma trận đường chéo

```python
# tạo một numpy array với đường chéo là số 1
# số 0 được điền vào những ô phần tử còn lại
import numpy as np

# numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')
# shape: 2 dòng, 3 cột
arr = np.eye(3)
print(arr)
```

Python




```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Thư viện numpy

- Tạo một numpy array với giá trị ngẫu nhiên

```python
# tạo một numpy array với giá trị ngẫu nhiên
import numpy as np

# numpy.random.random(size=None)
# shape: 2 dòng, 3 cột; với phần tử có giá trị ngẫu nhiên

arr = np.random.random((2,3))
print(arr)
```

Python

```
[[0.55136636 0.90697043 0.80751447]
 [0.50399661 0.53732573 0.94443   ]]
```

- Điều kiện cho mảng numpy

```python
import numpy as np

arr = np.arange(10)
print(arr)

out = np.where(arr < 5, arr, 10*arr)
print(out)
```

Python

```
[0 1 2 3 4 5 6 7 8 9]
[ 0 1 2 3 4 50 60 70 80 90]
```

Thư viện numpy

- Chuyển mảng về một chiều

```python
import numpy as np

arr = np.array([[1, 2], [3, 4]])
out = arr.flatten()

print(arr)
print(out)
```

Python

```
[[1 2]
 [3 4]]
[1 2 3 4]
```

Thư viện numpy

- **Kiểu dữ liệu (Datatypes)** :Mảng numpy chứa các phần tử cùng kiểu dữ liệu. Numpy cung cấp một tập hợp các kiểu dữ liệu mà chúng có thể sử dụng để xây dựng các mảng.

```python
import numpy as np

# int32
arr1 = np.array([1, 2])
print(arr1.dtype)

# float64
arr2 = np.array([1.0, 2.0])
print(arr2.dtype)

# int64
arr3 = np.array([1, 2], dtype = np.int64)
print(arr3.dtype)
```

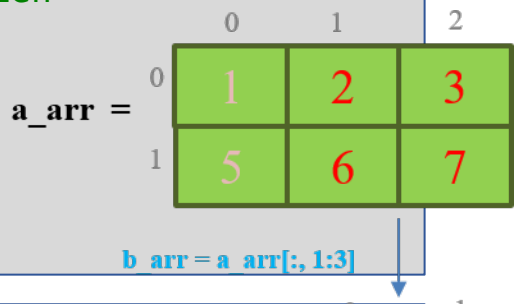
Python

```
int64
float64
int64
```

Thư viện numpy

- **Array Indexing :**Numpy cung cấp một số cách để truy xuất phần tử trong mảng. Truy xuất phần tử dùng kỹ thuật slicing tương tự như danh sách (list) trong Python.
  - **Ví dụ 1:** Lấy các phần tử từ mảng 2 chiều như sau:

```
import numpy as np
# Khởi tạo numpy array có shape = (2, 3) có giá trị như sau:
# [[ 1 2 3]
# [ 4 6 7]]
a_arr = np.array([[1,2,3],[5,6,7]])
print('a_arr: \n', a_arr)
# Sử dụng slicing để tạo mảng b bằng cách lấy 2 hàng đầu tiên
# và cột 1, 2. Như vậy b sẽ có shape = (2, 3):
# [[2 3]
# [6 7]]
b_arr = a_arr[:, 1:3]
print('b_arr: \n', b_arr)
```

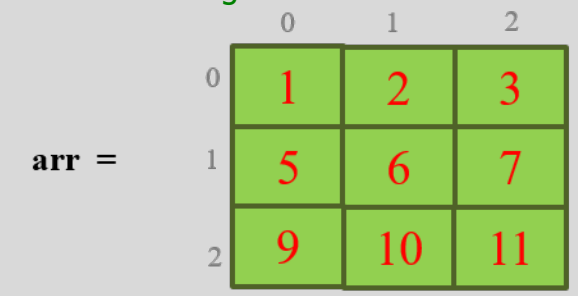


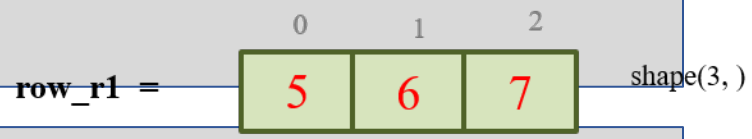

```
a_arr:
[[1 2 3]
 [5 6 7]]
b_arr:
[[2 3]
 [6 7]]
```

Thư viện numpy

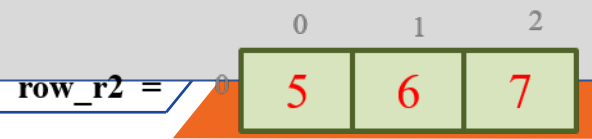
- **Ví dụ 2:** Lấy một dòng dữ liệu

```
import numpy as np

# Tạo một numpy array có shape (3, 4) với giá trị như sau:
# [[ 1 2 3 ]
# [ 5 6 7 ]
# [ 9 10 11 ]]
arr = np.array([[1, 2, 3], [5, 6, 7], [9, 10, 11]])

# Hai cách truy cập dữ liệu ở hàng giữa của mảng
# Cách 1: Dùng kết hợp chỉ số và slice -> được array mới có số chiều thấp
hơn
# Cách 2: Nếu chỉ dùng slice ta sẽ có array mới có cùng số chiều với array
gốc

# Cách 1: số chiều giảm
row_r1 = arr[1, :]

# Cách 2: số chiều được giữ nguyên
row_r2 = arr[1:2, :]

print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)
```






```
[5 6 7] (3,)
[[5 6 7]] (1, 3)
```



Thư viện numpy

- Ví dụ 3: Lấy một cột dữ liệu

```
import numpy as np
# Tạo một numpy array có shape (3, 4) với giá trị như sau:
arr = np.array([[1, 2, 3], [5, 6, 7], [9, 10, 11]])
#[[ 1 2 3 ]
# [ 5 6 7 ]
# [ 9 10 11 ]]
# Hai cách truy cập dữ liệu ở cột giữa của mảng
# Cách 1: Dùng kết hợp chỉ số và slice
#-> được array mới có số chiều thấp hơn (số chiều giảm)
col_r1 = arr[:, 1]
# Cách 2: Nếu chỉ dùng slice ta sẽ có array mới có
#cùng số chiều với array gốc(số chiều được giữ nguyên)
col_r2 = arr[:, 1:2]

print(col_r1, col_r1.shape)
print(col_r2, col_r2.shape)
```

arr =

	0	1	2
0	1	2	3
1	5	6	7
2	9	10	11

col_r1 =

0	1	2
2	6	10

col_r2 =

	0
0	2
1	6
2	10

(3,1)

```
[ 2 6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
```

Thư viện numpy

- **Boolean indexing:** Cho phép chúng ta chọn ra các phần tử tùy ý của một mảng. Kiểu truy xuất này thường được sử dụng để chọn ra các phần tử thỏa mãn điều kiện nào đó.
- **Ví dụ 4:** Tìm các vị trí thoả mãn điều kiện

```
import numpy as np

a_arr = np.array([[1, 2], [3, 4], [5, 6]])
print(a_arr)

# Tìm các phần tử lớn hơn 2
# Trả về 1 mảng Boolean có số chiều như mảng a_arr
# và giá trị tại mỗi phần từ là True nếu phần tử của a tại đó > 2,
# False cho trường hợp ngược lại

bool_idx = (a_arr > 2)
print(bool_idx)
```

arr =

	0	1
0	1	2
1	3	4
2	5	6

bool_idx = (a_arr >2)

	0	1
0	F	F
1	T	T
2	T	T

```
[[1 2]
 [3 4]
 [5 6]]
[[False False]
 [ True True]
 [ True True]]
```

Thư viện numpy

- **Ví dụ 5**: Tìm các vị trí thoả mãn điều kiện và lấy phần tử tương ứng

```
import numpy as np

a_arr = np.array([[1, 2], [3, 4], [5, 6]])
print(a_arr)

# Tìm các phần tử lớn hơn 2
# Trả về 1 mảng Boolean có số chiều như mảng a_arr
# và giá trị tại mỗi phần từ là True nếu phần tử của a tại đó > 2,
# False cho trường hợp ngược lại

bool_idx = (a_arr > 2)
print('bool_idx: \n', bool_idx)

# Chúng ta sẽ sử dụng boolean array indexing để xây dựng mảng 1 chiều
# Bao gồm các phần tử tương ứng với giá trị True của bool_idx

# Ví dụ ở đây in ra các giá trị của a_arr >2, sử dụng array bool_idx đã
tạo
out = a_arr[bool_idx]
print('\nMethod 1:\n', out)

# một cách ngắn gọn hơn
print('\nMethod 2:\n', a_arr[a_arr > 2])
```

Thư viện numpy

```
[[1 2]
 [3 4]
 [5 6]]
bool_idx:
 [[False False]
  [ True True]
  [ True True]]

Method 1:
 [3 4 5 6]

Method 2:
 [3 4 5 6]
```

arr =

	0	1
0	1	2
1	3	4
2	5	6

Cách 1:

(a_arr[bool_idx])

	0	1
0	F	F
1	T	T
2	T	T

0	1	2	3
3	4	5	6

Cách 2:

a_arr[a_arr > 2]

0	1	2	3
3	4	5	6

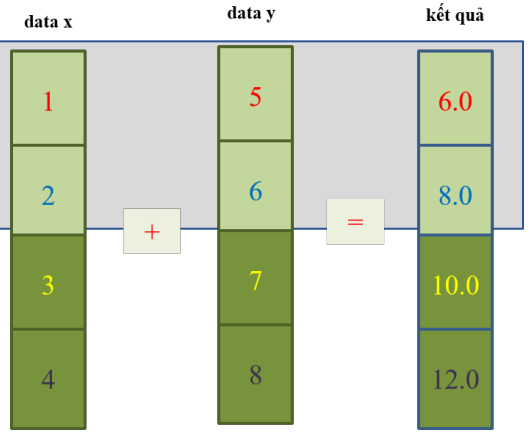
Thư viện numpy

- **Phép toán trên mảng**
  - Phép cộng giữa 2 mảng

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Tổng của 2 mảng, cả 2 cách đều cho cùng một kết quả
# [[ 6.0 8.0]
# [10.0 12.0]]
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

data x + data y = kết quả



Thư viện numpy

- Phép trừ giữa hai mảng

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Phép trừ của 2 mảng, cả 2 cách đều cho cùng một kết quả
# [[-4.0 -4.0]
# [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

Thư viện numpy

- Phép nhân giữa hai mảng

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Phép nhân
# [[ 5.0 12.0]
# [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

data x * data y = kết quả

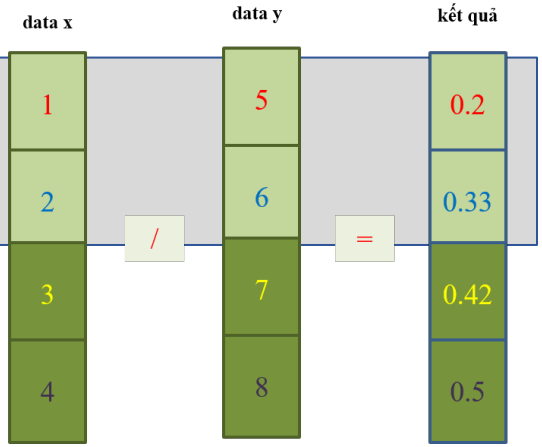
Thư viện numpy

- Phép chia giữa 2 mảng

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Phép chia
# [[ 0.2 0.33333333]
# [ 0.42857143 0.5 ]]
print(x / y)
print(np.divide(x, y))
```

```
[[0.2 0.33333333]
 [0.42857143 0.5 ]]
[[0.2 0.33333333]
 [0.42857143 0.5 ]]
```

data x / data y = kết quả



Thư viện numpy

- Tính căn bậc 2

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Phép lấy căn
# [[ 1. 1.41421356]
# [ 1.73205081 2. ]]
print(np.sqrt(x))
```

```
[[1.          1.41421356]
 [1.73205081  2. ]]
```

- Nhân giữa hai vector (inner product)

```
import numpy as np

v = np.array([9,10])
w = np.array([11, 12])

# Nhân giữa 2 vector, cả 2 đều cho kết quả 219
print(v.dot(w))
print(np.dot(v, w))
```

```
219
219
```

Thư viện numpy

- Nhân giữa vector và ma trận

```
import numpy as np

X = np.array([[1,2],[3,4]])
v = np.array([9,10])

# Nhân giữa Matrix và vector, cả 2 đều cho kết quả array [29 67]
print(X.dot(v))
print(np.dot(X, v))
```

```
[29 67]
[29 67]
```

- Nhân giữa matrix và matrix

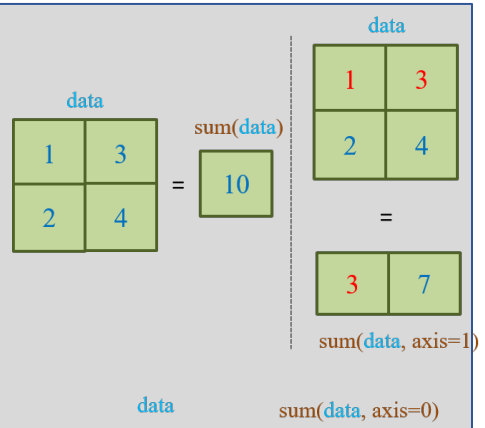
```
import numpy as np

X = np.array([[1,2],[3,4]])
Y = np.array([[5,6],[7,8]])

# Nhân giữa matrix và matrix; cả 2 cách đều cho cùng kết quả
# [[19 22]
# [43 50]]
print(X.dot(Y))
print(np.dot(X, Y))
```

```
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

Thư viện numpy

- Tính tổng cho một mảng numpy

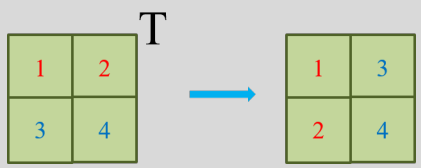
```
import numpy as np

x = np.array([[1,2],[3,4]])

# Tổng các phần tử của mảng; prints "10"
print(np.sum(x))
# Tính tổng theo từng cột
print(np.sum(x, axis=0))
# Tính tổng theo từng hàng
print(np.sum(x, axis=1))
```




```
10
[4 6]
[3 7]
```

- Chuyển vị cho một ma trận

```
import numpy as np

x = np.array([[1,2], [3,4]])
print(x) # Prints "[[1 2]
#                  [3 4]]"
print(x.T) # Prints "[[1 3]
#                    [2 4]]"
```




```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
```

Thư viện numpy

- **Broadcasting** :cho phép thực thi các phép toán trên các mảng có kích thước khác nhau.
  - Ví dụ: chúng ta muốn cộng vector cho mỗi hàng của ma trận. Chúng ta có thể làm như sau:

```
import numpy as np

X = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
v = np.array([1,0,1])
Y = np.empty_like(X)

# Cách 1: Thêm vector v vào mỗi hàng của ma trận X bằng vòng lặp
for i in range(4):
Y[i, :] = X[i, :] + v
print('Matrix X: \n', X)
print('\nVector v: \n', v)
print('\nMatrix Y: \n', Y)
```

```
Matrix X:
     [[ 1 2 3]
     [ 4 5 6]
     [ 7 8 9]
     [10 11 12]]
Vector v:
     [1 0 1]
```

```
Matrix Y:
     [[ 2  2  4]
     [  5  5  7]
     [  8  8 10]
     [ 11 11 13]]
```

Thư viện numpy

- Cách này hoạt động bình thường với ma trận X nhỏ. Khi ma trận X lớn, việc sử dụng vòng lặp này sẽ rất chậm.
- Chúng ta có thể thực hiện mục đích trên bằng cách xếp chồng nhiều bản sao của v theo chiều dọc, sau đó thực hiện phép tính tổng với X. Chúng ta có thể thực hiện phương pháp này như sau:

```
import numpy as np

X = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
v = np.array([1,0,1])

# Xếp chồng 4 bản sao của v lên nhau:
V_t = np.tile(v, (4, 1))

# Thực hiện phép cộng
Y = X + V_t

print('Matrix X: \n', X)
print('\nVector v: \n', v)
print('\nMatrix v: \n', V_t)
print('\nMatrix Y: \n', Y)
```

```
Matrix X:          Vector v:       Matrix v:          Matrix Y:
     [[ 1 2 3]          [1 0 1]         [[1 0 1]           [[ 2 2 4]
     [ 4 5 6]                           [1 0 1]            [ 5 5 7]
     [ 7 8 9]                           [1 0 1]            [ 8 8 10]
     [10 11 12]]                        [1 0 1]]           [11 11 13]]
```

Thư viện numpy

numpy.title(v, (4,1))

v =

1	0	1

V_t =

	0	1	2
0	1	0	1
1	1	0	1
2	1	0	1
3	1	0	1

---

v

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

+

V_t

	0	1	2
0	1	0	1
1	1	0	1
2	1	0	1
3	1	0	1

=

Y

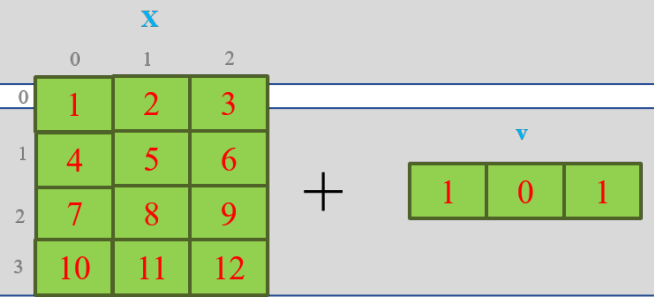
	0	1	2
0	2	2	4
1	5	5	7
2	8	8	10
3	11	11	13

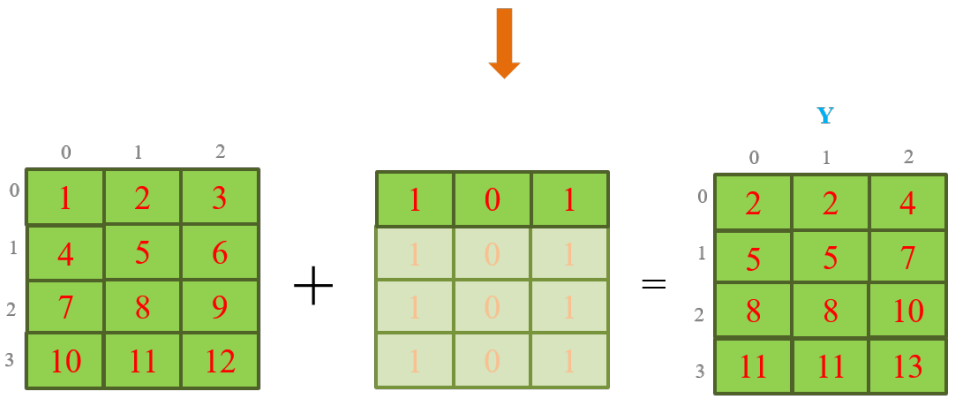
Thư viện numpy

- Numpy broadcasting cho phép chúng ta thực thi tính toán này mà không cần phải làm thêm các bước thêm nào.

```
import numpy as np

X = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
v = np.array([1,0,1])
Y = X + v
print(Y)
```

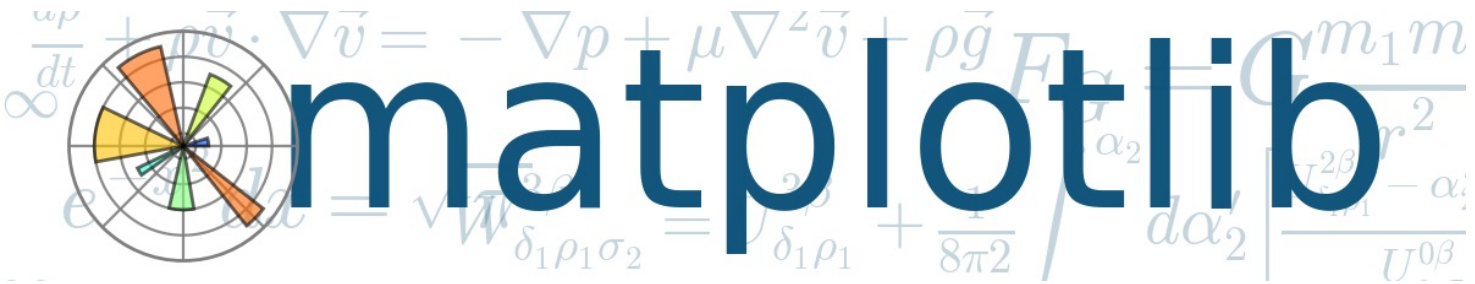
```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```





Thư viện matplotlib

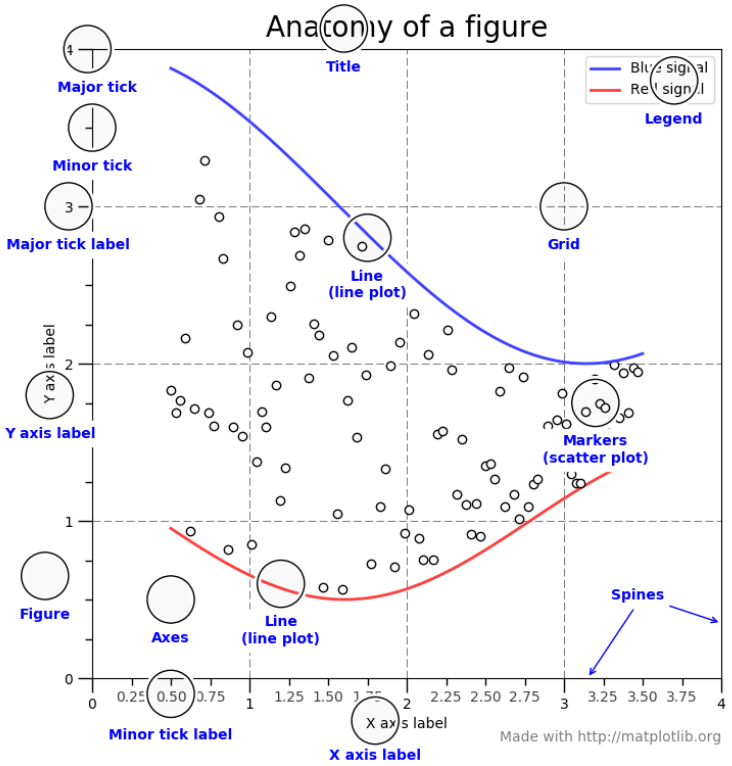
- Matploblib là một thư viện trực quan hoá dữ liệu phổ biến trong Python. Nó có thể vẽ được nhiều loại đồ thị khác nhau, và rất hữu ích khi làm việc cùng với NumPy.



- Hướng dẫn cài đặt
  - Nếu sử dụng Anacoda, cài đặt thông qua Terminal: `conda install matplotlib`
  - cài đặt Matplotlib từ Terminal bằng: `pip install matplotlib`
  - Nếu sử dụng colab không cần cài đặt

Thư viện matplotlib

- Cấu trúc của Plot :Gồm 2 phần chính là: Figure & Axes
  - Figure: là cửa sổ chứa tất cả mọi thứ được vẽ trên nó. Nó có thể chứa nhiều thành phần độc lập nhau như: các Axes khác nhau, tiêu đề, chú thích,...
  - Axes: là khu vực chứa dữ liệu được vẽ lên và các tiêu đề, chú thích... *gắn với nó.*



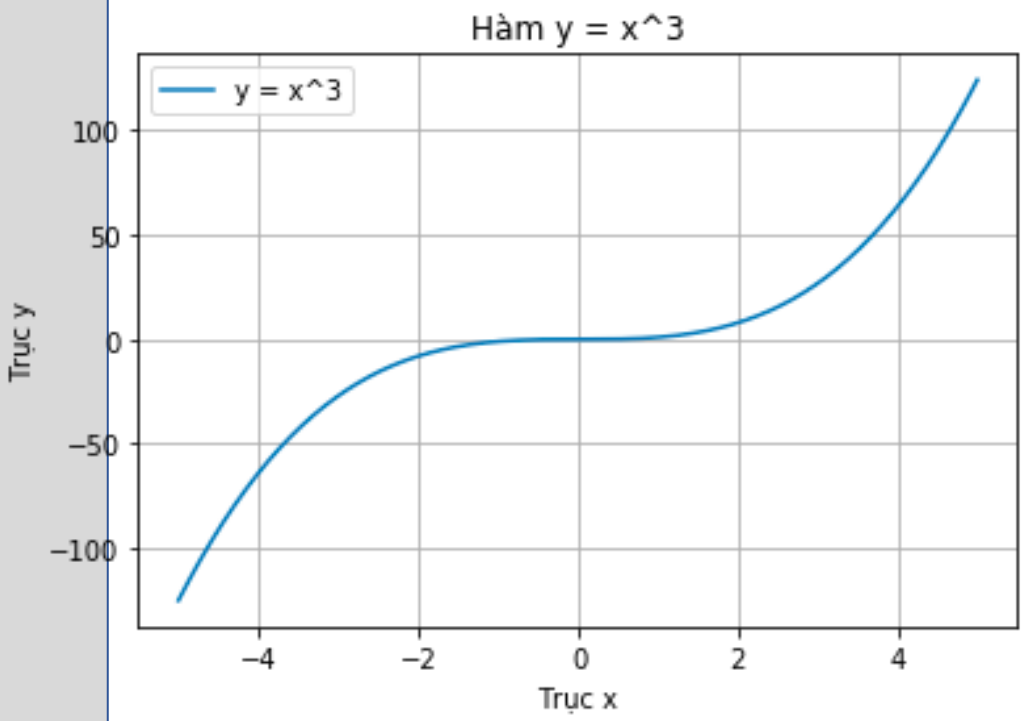
Thư viện matplotlib

- Cách 1: Sử dụng lệnh Matplotlib cơ bản
- Một vài cú pháp cơ bản:
  - `plt.figure()`: tạo một figure
  - `plt.plot()`: vẽ đồ thị với giá trị trục x so với trục y
  - `plt.xlabel()`: đặt tên cho trục x
  - `plt.ylabel()`: đặt tên cho trục y
  - `plt.title()`: đặt tên cho figure
  - `plt.grid()`: hiển thị các đường lưới
  - `plt.legend()`: hiển thị chú thích cho đồ thị
  - `plt.savefig()`: lưu hình về máy
  - `plt.show()`: hiển thị hình vẽ
  - `plt.clf()`: xoá hình hiện tại (hữu ích khi có nhiều figure trong 1 chương trình)

Thư viện matplotlib

- Ví dụ: vẽ đồ thị hàm số: y = x^3

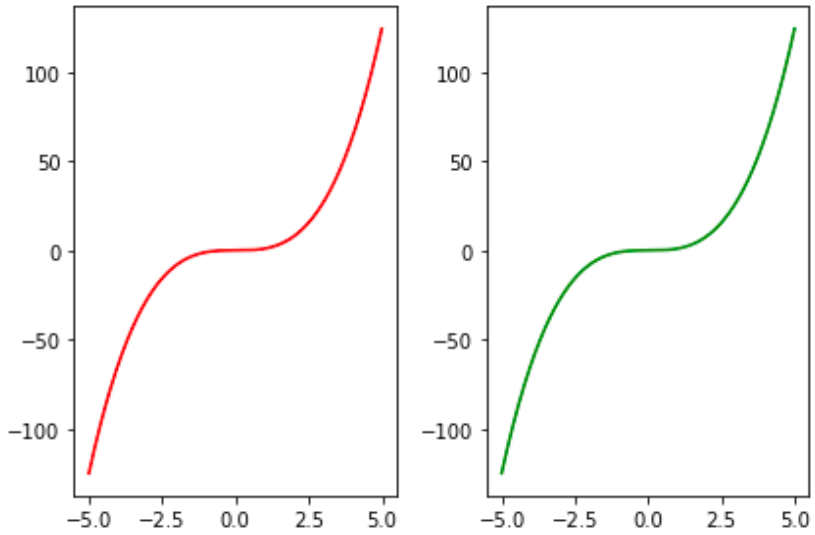
```
import numpy as np
x = np.arange(-5, 5, 0.01)
y = x ** 3

# Đặt tên
plt.title('Hàm y = x^3')
plt.xlabel('Trục x')
plt.ylabel('Trục y')

# Hiển thị lưới
plt.grid()

# Vẽ plt.plot()
plt.plot(x, y, label='y = x^3')

# HIển thị chú thích
plt.legend()

# Hiển thị hình vẽ
plt.show()
```



Thư viện matplotlib

- Matplotlib còn cho chúng ta có thể vẽ nhiều hình trên 1 Figure:
- Chúng ta có thể sử dụng phương thức `.subplot()` gồm có 3 thông số cụ thể:
  - `nrows`: số lượng hàng trên Figure
  - `ncols`: số lượng cột trên Figure
  - `plot_number`: vị trí của plot trong Figure

```
# plt.subplot(nrows, ncols, plot_number)

# nrows = 1, ncols = 2 (1 hàng, 2 cột)
plt.subplot(1, 2, 1) # vẽ ở vị trí 1
plt.plot(x, y, 'red')

plt.subplot(1, 2, 2) # vẽ ở vị trí 2
plt.plot(x, y, 'green')

# Để tránh trường hợp các giá trị trục y của
Subplot bên phải nằm sát với Subplot bên
trái,
# ta có thể sử dụng phương thức
tight_layout()
plt.tight_layout()
```



Thư viện matplotlib

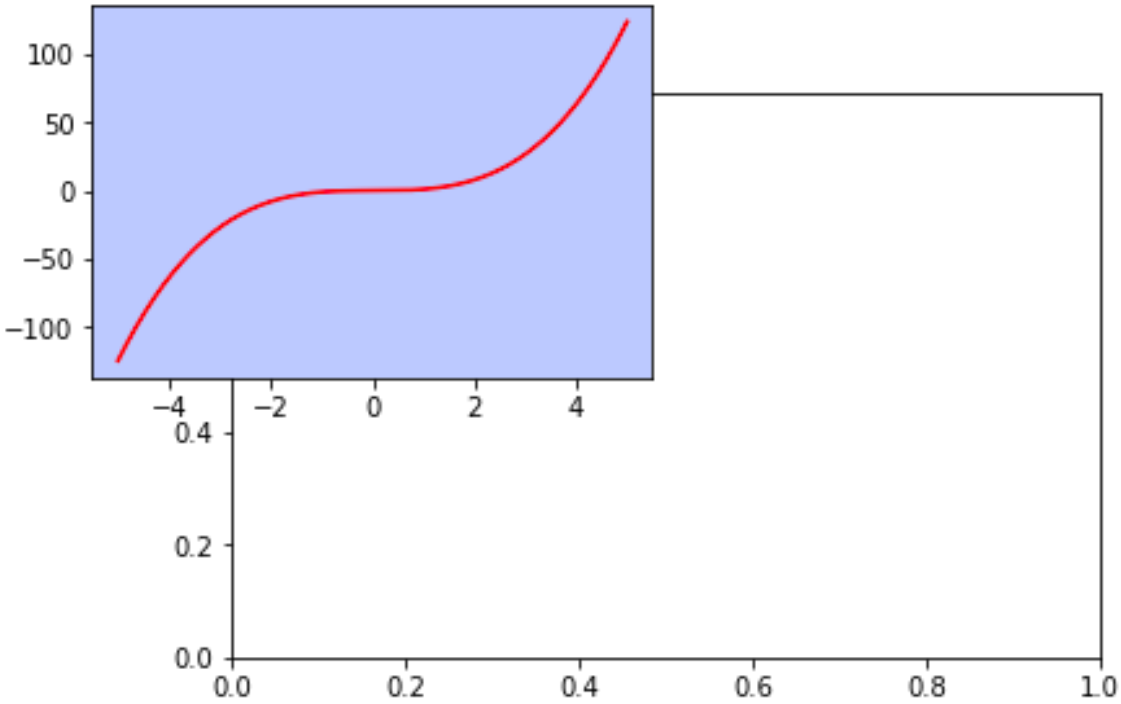
- Cách 2: Phương thức hướng đối tượng
- Figure : thành phần chính của **figure** là các **axes**. Một **figure** có thể chứa một hoặc nhiều **axes**. Hay nói cách khác, **figure** chỉ là khung chứa, chính xác **axes** mới thật sự là nơi các hình được vẽ lên.

```
# Khởi tạo figure trống
fig = plt.figure()

# Hiển thị figure hiện tại
plt.show()
```

- Bây giờ cần thêm các **axes** vào **figure**, bằng phương thức: **.add_axes()** gồm 4 đối số: `**bên trái, phía dưới, chiều rộng, chiều cao.**`

```
fig = plt.figure()

# Có thể gọi tắt subplot(1,1,1) thành subplot(111)
fig.add_subplot(111)

# facecolor là màu nền
ax = fig.add_axes([0, 0.5, 0.5, 0.5], facecolor='#BECAFB')
ax.plot(x, y, 'red')

plt.show()
```

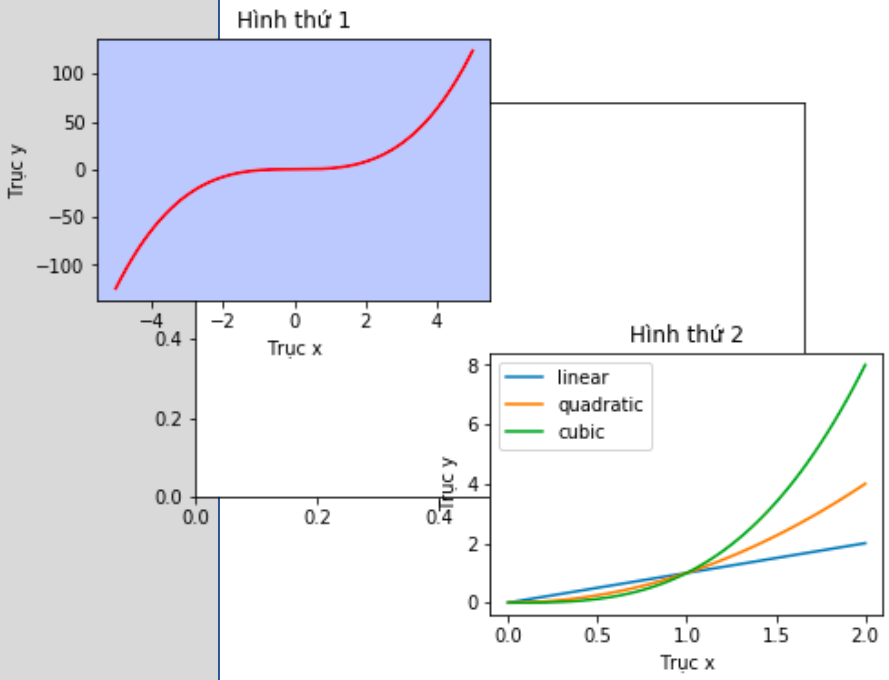
Thư viện matplotlib



Thư viện matplotlib

- Thêm tiêu đề cho các axes bằng cách:
  - `.set_xlabel()`: tiêu đề trục x
  - `.set_ylabel()`: tiêu đề trục y
  - `.set_title()`: tiêu đề axes

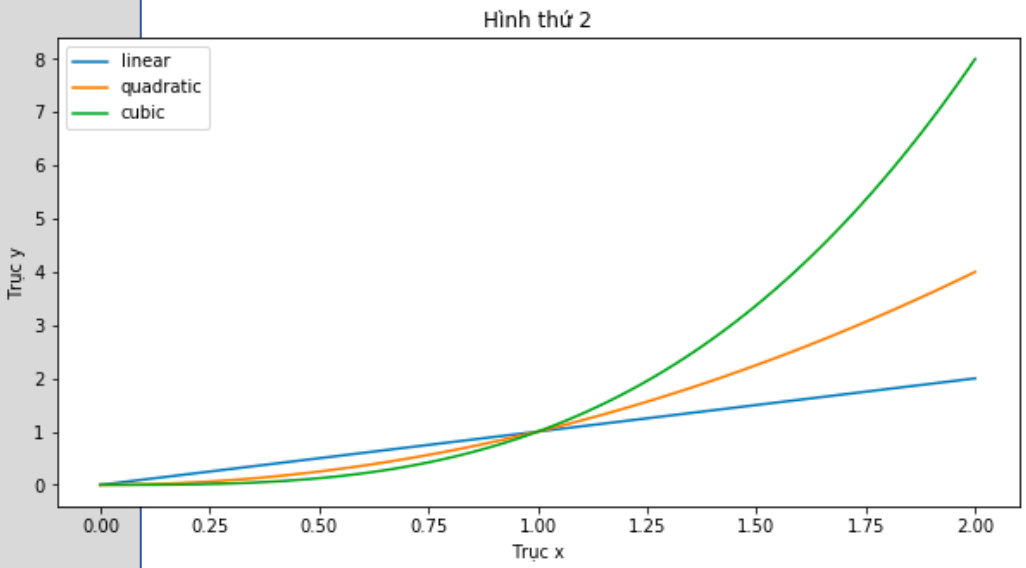
```
fig = plt.figure()
# Có thể gọi tắt subplot(1,1,1) thành subplot(111)
fig.add_subplot(111)

ax = fig.add_axes([0, 0.5, 0.5, 0.5],
facecolor='#BECAFB')
ax.plot(x, y, 'red')

ax.set_xlabel('Trục x')
ax.set_ylabel('Trục y')
ax.set_title('Hình thứ 1')

# Tạo thêm 1 axes trong cùng 1 figure
ax2 = fig.add_axes([0.5, -0.1, 0.5, 0.5])
x = np.linspace(0, 2, 100)
ax2.plot(x, x, label='linear')
ax2.plot(x, x**2, label='quadratic')
ax2.plot(x, x**3, label='cubic')

ax2.set_xlabel('Trục x')
ax2.set_ylabel('Trục y')
ax2.set_title('Hình thứ 2')
ax2.legend()
plt.show()
```



Thư viện matplotlib

- Một số thuộc tính khác của Figure
  - Chỉnh kích thước của Figure bằng phương thức `figsize` (đơn vị inch).
  - `dpi` độ phân giải của hình, giá trị mặc định là 100. `dpi` càng cao thì chất lượng ảnh càng cao.
  - Lưu Figure bằng `fig.savefig()`

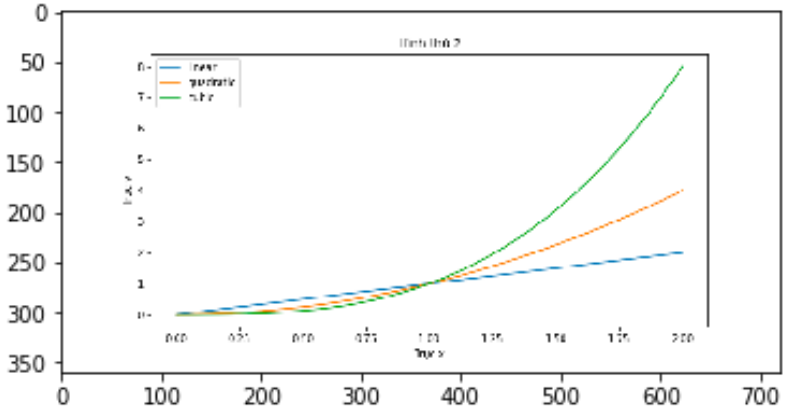
```
# Khởi tạo nhanh Figure và Axes ->
plt.subplots()
fig, ax = plt.subplots(figsize=(10,5))

x = np.linspace(0, 2, 100)
ax.plot(x, x, label='linear')
ax.plot(x, x**2, label='quadratic')
ax.plot(x, x**3, label='cubic')

ax.set_xlabel('Trục x')
ax.set_ylabel('Trục y')
ax.set_title('Hình thứ 2')
ax.legend()

plt.show()

# Lưu figure cùng thư mục với code
fig.savefig('my_figure.png')
```



Thư viện matplotlib

- Kiểm tra Figure đã lưu hay chưa?

```
import matplotlib.image as mpimg

# Hiển thị hình ảnh lưu trong máy
plt.imshow(mpimg.imread('my_figure.png'))
```

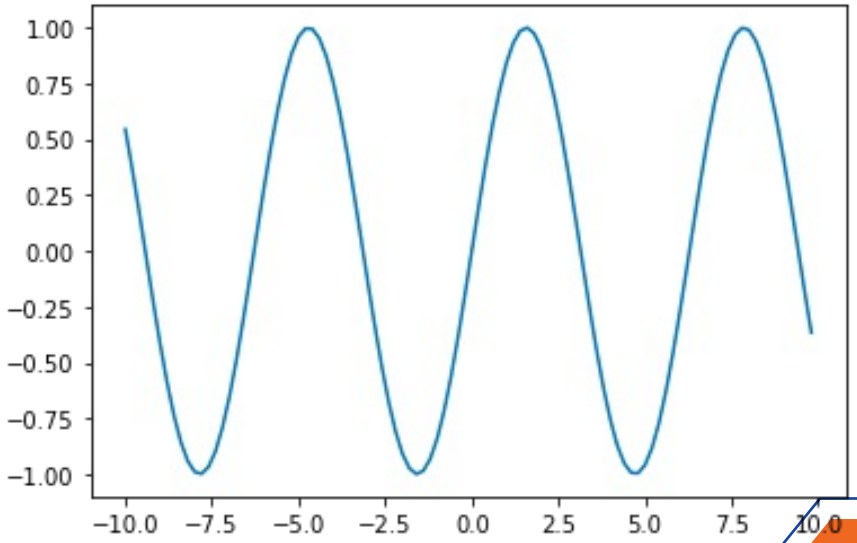


- Cách xoá, đóng 1 figure:
- Xoá nội dung của fig:
  - `fig.clf()`
  - `plt.clf()`
- Đóng fig:
  - `plt.close(<Tên fig>)`

Thư viện matplotlib

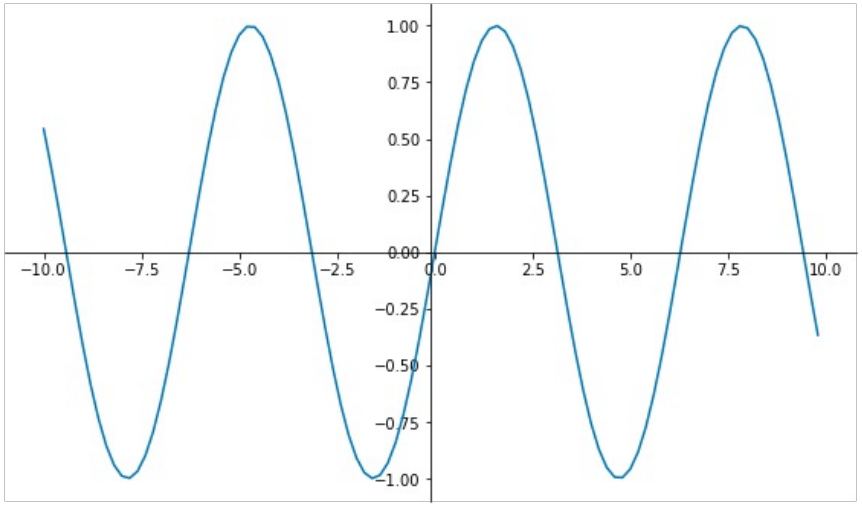
- Chỉnh trục toạ độ trong Figure (Spines và Ticks của đồ thị)
  - Axis spine - đường ghi nhận ranh giới khu vực dữ liệu. Spine là đường kết nối các dấu ticks trên trục và ghi nhận ranh giới của khu vực dữ liệu. Chúng ta có thể cài đặt tuỳ ý.
  - Ví dụ: khảo sát và vẽ đồ thị hàm số y = sin(x)

```
# Trong trường hợp chưa chỉnh các trục toạ độ
x = np.arange(-10., 10., 0.2)
y = np.sin(x)

plt.plot(x, y)
plt.show()
```



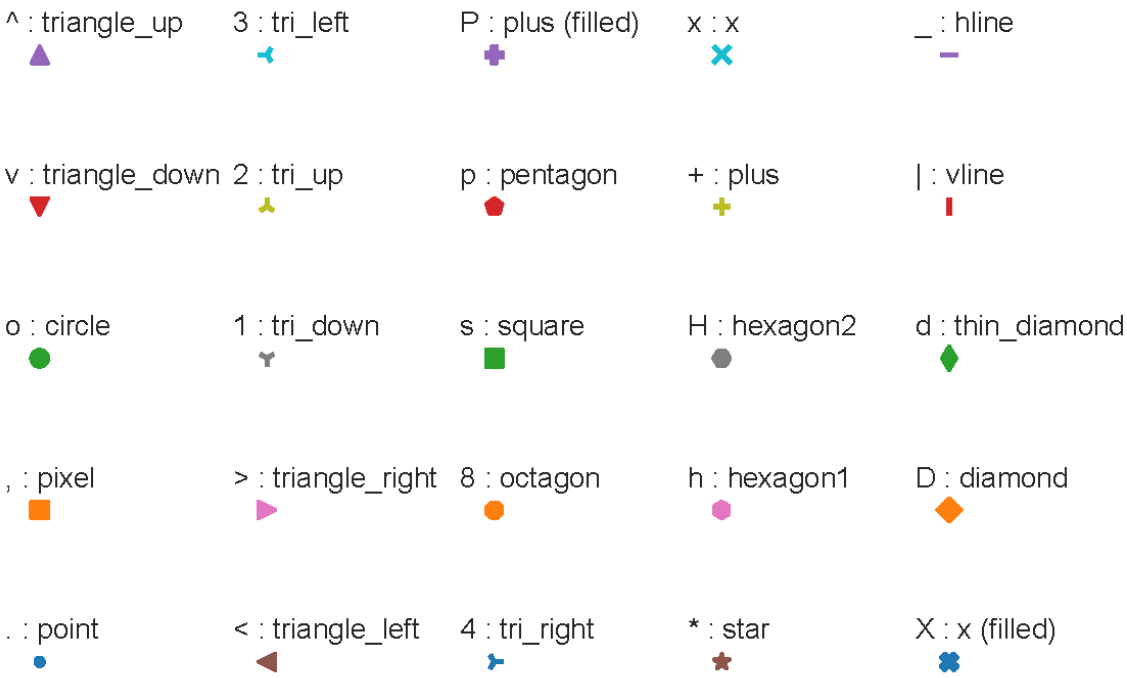
Thư viện matplotlib

```
# Chỉnh lại trục toạ độ

fig, ax = plt.subplots(figsize=(10,6))
x = np.arange(-10., 10., 0.2)
y = np.sin(x)

# Biến đường biên bên trên và bên phải -> vô
hình
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')

# Di chuyển đường bên dưới vào giữa & ở vị
trí y = 0
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))

# Di chuyển đường bên trái vào giữa & ở vị
trí x = 0
ax.yaxis.set_ticks_position('left')
# ax.spines['left'].set_position(('data',
0))

# Hoặc thay ('data', 0) thành 'center'
ax.spines['left'].set_position('center')

ax.plot(x, y)
plt.show()
```



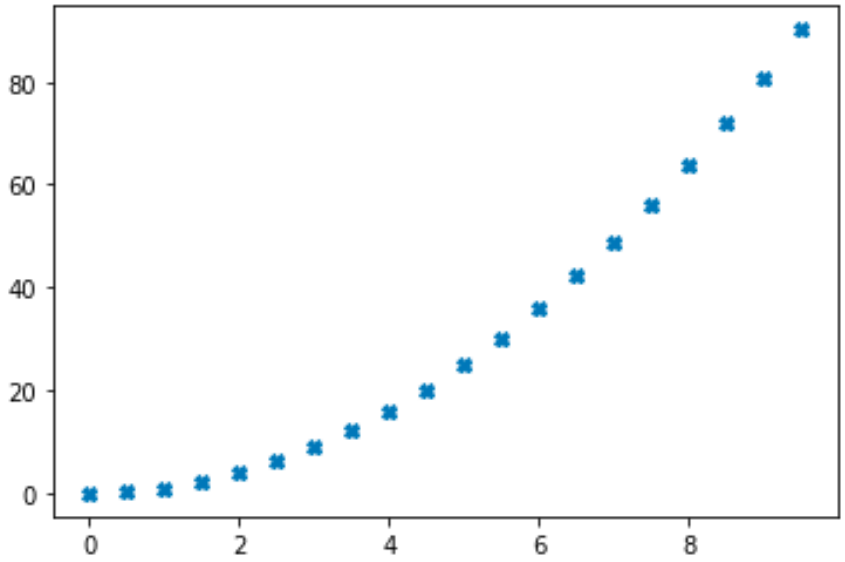
Thư viện matplotlib

- Vẽ đồ thị kiểu điểm
  - Trong nhiều trường hợp, ta muốn hiển thị từng điểm thay vì đường thẳng, việc chuyển đường thẳng sang điểm chúng ta sử dụng thêm các `marker` trong `.plot()`.
  - Các `marker` thông dụng:



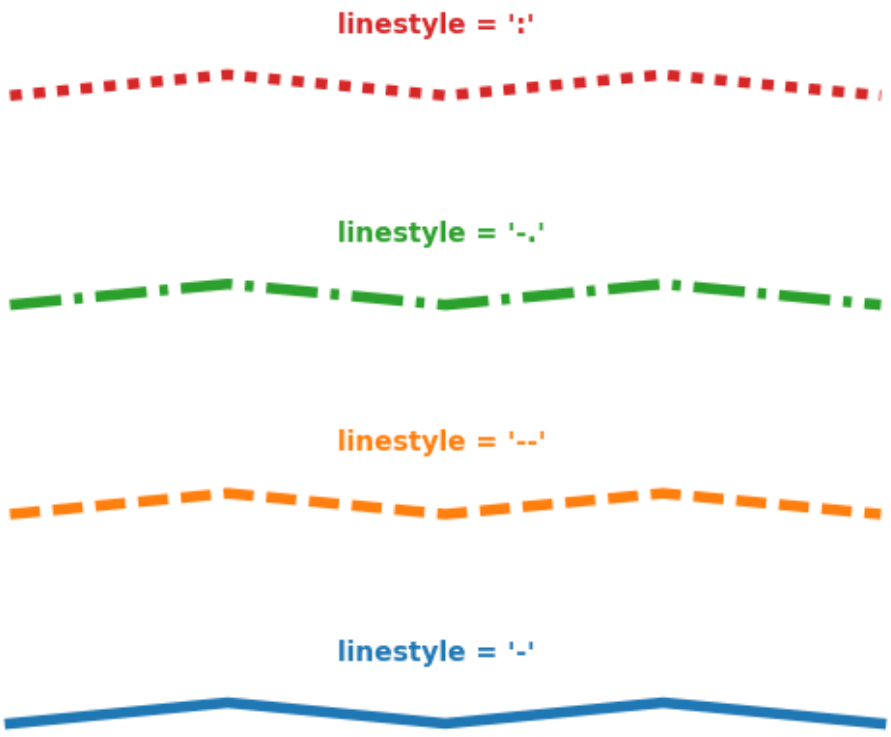
Thư viện matplotlib

```
t = np.arange(0., 10., 0.5)

plt.plot(t, t**2, 'X')
plt.show()
```



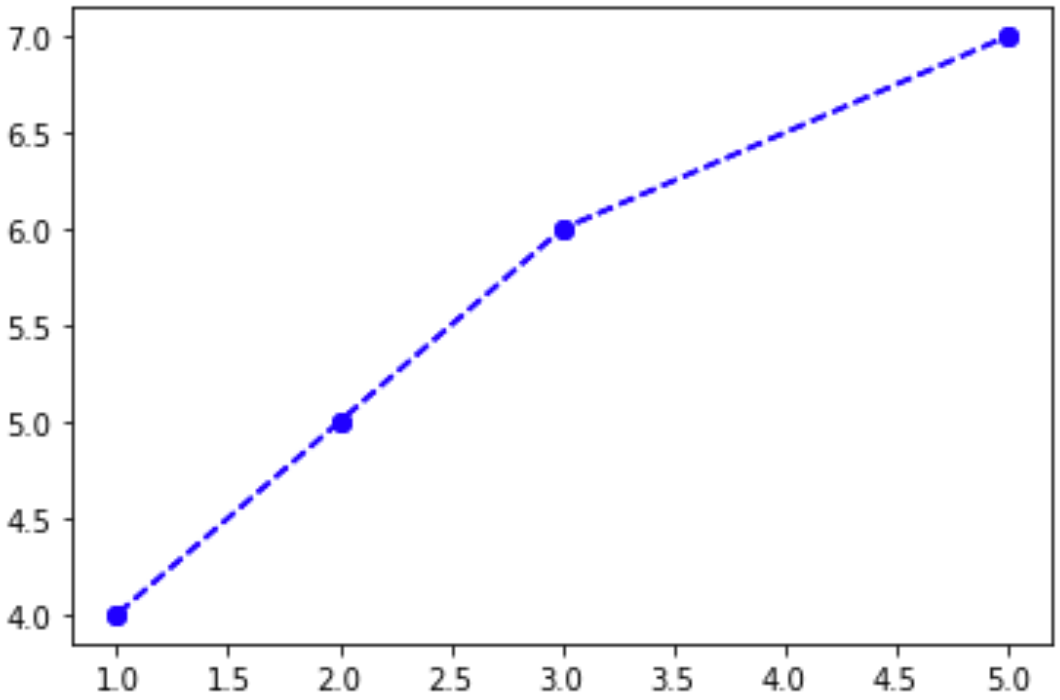
Thư viện matplotlib

- Cách vẽ vừa điểm, vừa đường:
- Cách 1: nối các điểm bằng đường
  - Một số kiểu `linestyle`:



Thư viện matplotlib

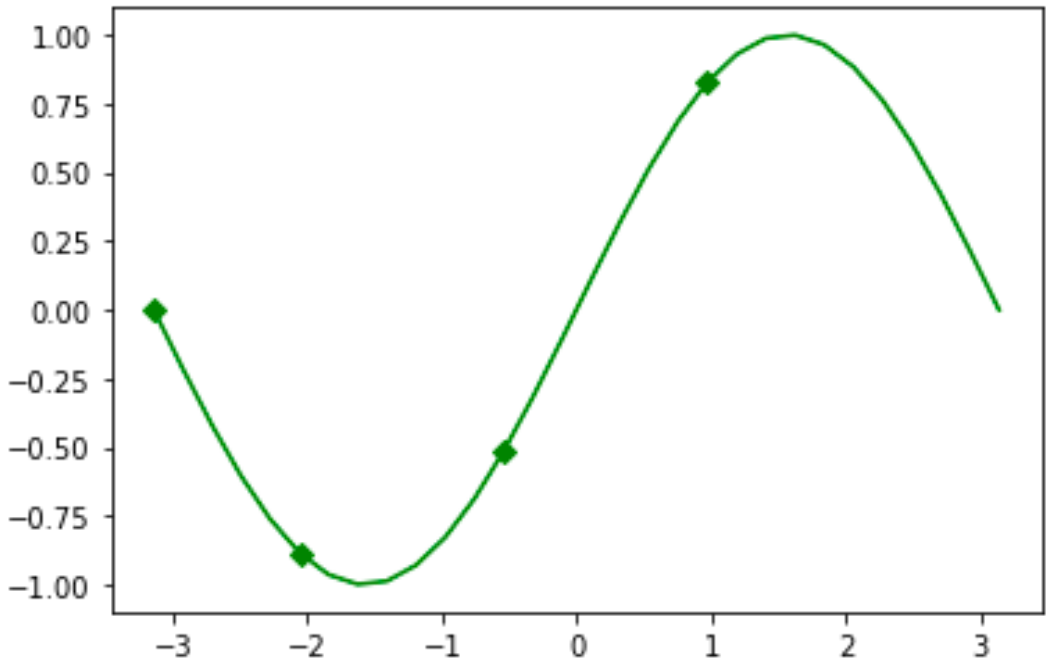
```
x = [1, 2, 3, 5]
y = [4, 5, 6, 7]

plt.plot(x, y, linestyle='--', marker='o', color='b')

# Một cách ngắn hơn
plt.plot(x, y, '--bo')
plt.show()
```



Thư viện matplotlib

- Cách 2: Vẽ một vài điểm trên đường

```
x = np.linspace(-np.pi, np.pi, 30)
y = np.sin(x)
markers_on = [0, 19, 12, 5]
plt.plot(x, y, '-gD', markevery=markers_on)
plt.show()
```



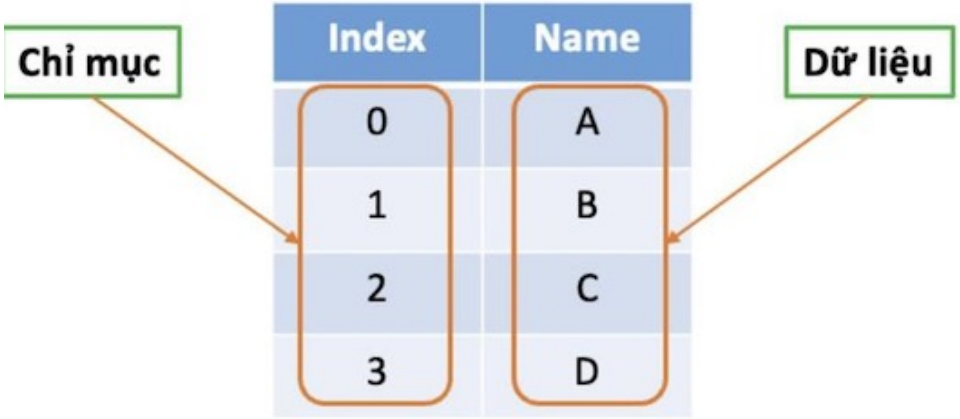
Thư viện Pandas

- Thư viện trong Python, ưu điểm nhanh, mạnh, linh động dễ sử dụng, mã nguồn mở , công cụ dùng dể phân tích và thao tác dữ liệu (cleaning, analyzing và manipulating)
- Được xây dựng dựa trên Numpy
- Giúp extract valuable insights của các tập dữ liệu
- Hiệu quả khi làm việc với dữ liệu bảng (SQL, Excel)

Xử lý dữ liệu với Pandas

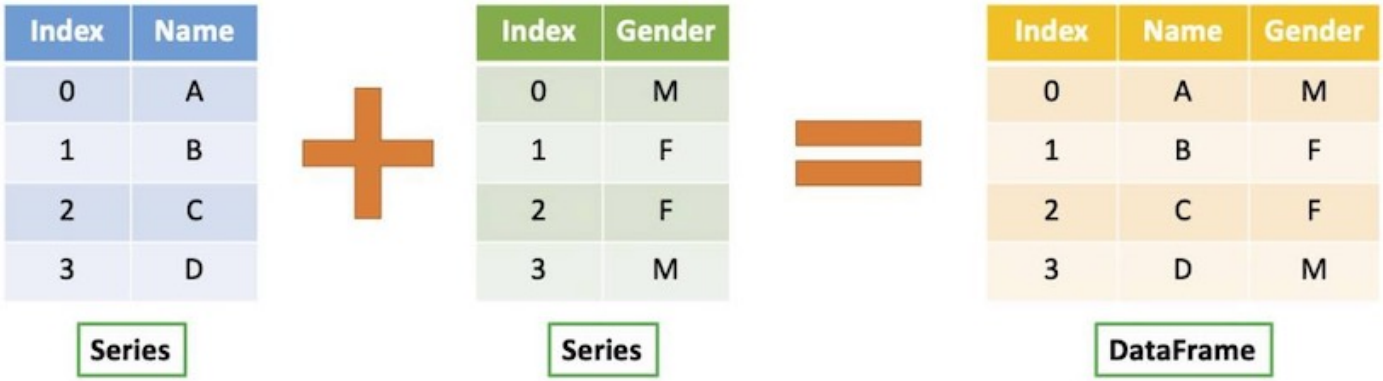
- Một số đặc điểm của Pandas:
  - Thao tác với các nguồn dữ liệu từ file csv,excel, SQL, JSON
  - Cung cấp các loại cấu trúc khác nhau như Series, DataFrame, Panel
  - Có thể đáp ứng nhiều dạng dataset khác nhau như time series, heterogenous data, tabular, matrix
  - Có thể làm việc với missing data, parsing, conversion
  - Cung cấp các kỹ thuật lọc dữ liêuh
  - Cung cấp time series functionality – date range generation, frequency conversion...
  - Tích hợp tốt với Scikit-learn, statmodels, Scipy
  - Cấu trúc dữ liệu: Series, DataFrame, Panel.

Xử lý dữ liệu với Pandas



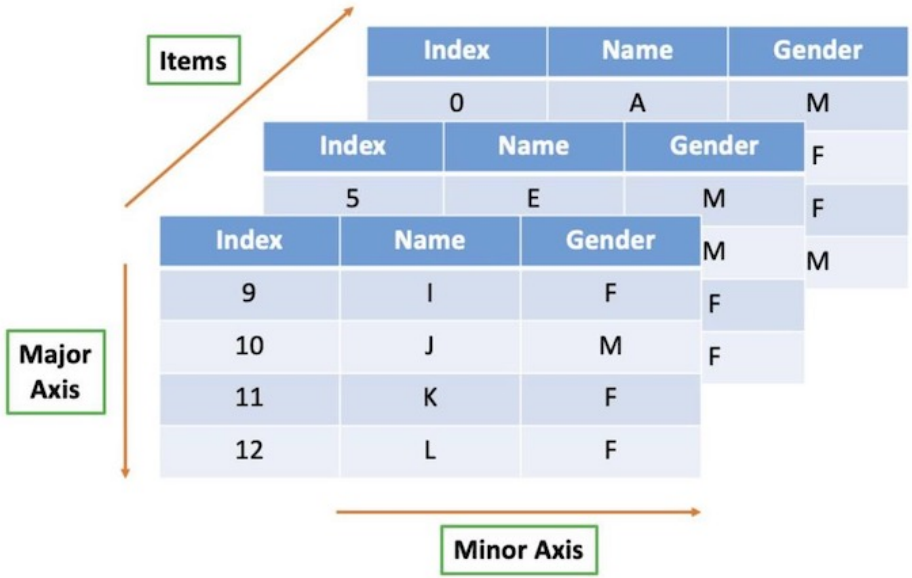
- **Series:** có cấu trúc là mảng 1D với dữ liệu đồng nhất, loại dữ liệu có thể là integer, string, float,.. trục đánh nhãn được gọi là chỉ mục (index). Kích thước của series là không thể thay đổi (immutable) và giá trị dữ liệu có thể thay đổi (mutable). Để khởi tạo Series có thể dùng pandas.Series(data, index, dtype, copy), trong đó:
  - **data:** nhận các giá trị có dạng ndarray, list, dictionary, constant,...
  - **index:** giá trị index phải là duy nhất (unique), có thể hash và có kích thước bằng **data**, mặc định **index** có giá trị 0, 1, 2.
  - **dtype:** loại dữ liệu của giá trị bên trong Series.
  - **copy:** copy input **data**, mặc định có giá trị False.

Xử lý dữ liệu với Pandas



- **DataFrame:** là cấu trúc dữ liệu 2D, có dạng bảng bao gồm các cột và hàng, các cột có thể định nghĩa loại dữ liệu khác nhau. Các cột có các kiểu dữ liệu khác nhau như float64, int, bool,.. **Một cột của DataFrame là một cấu trúc Series**. Các chiều DataFrame được đánh nhãn theo các hàng và cột. Từ đó, ta có thể thao tác trên cả hàng và cột. Để khởi tạo DataFrame, có thể thực hiện bởi pandas.DataFrame(data, index, columns, dtype, copy).
  - **data:** nhận các giá trị như ndarray, series, map, lists, dict, constants và DataFrame khác.
  - Các tham số khác tương tự như Series, pandas DataFrame có thể được tạo dùng các input như Lists, Dict, Series, Numpy ndarrays, DataFrame khác.

Xử lý dữ liệu với Pandas



- **Panel:** là một 3D container, trong đó:
  - items: axis 0, mỗi item tương ứng DataFrame chứa bên trong.
  - major_axis: axis 1, nó là các hàng (rows) của mỗi DataFrame.
  - minor_axis: axis 2, nó là các cột (columns) của mỗi DataFrame

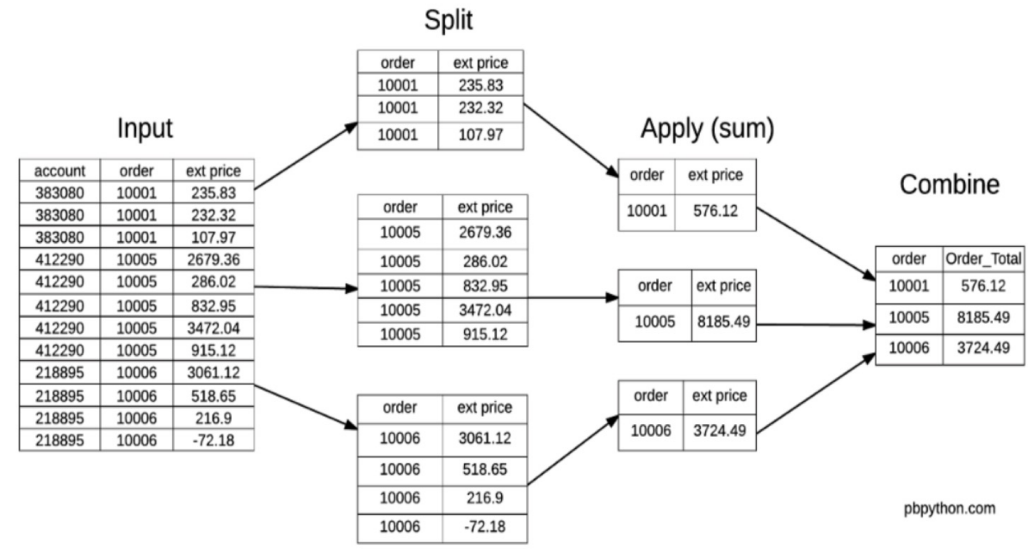
Xử lý dữ liệu với Pandas

Một số function trên Pandas thường dùng để xử lý dữ liệu:

- **Handle missing values**: isna(), notna() – tìm kiếm các giá trị NA, isnull().
- **Indexing and slicing in Pandas**: .loc (label based), .iloc (integer based), .ix (label and integer based).
- **Các query như trong excel hay SQL**: where(), query().
- **Sort**: sort_index(), sort_values().
- **Series basic functionality**: axes, dtype, empty, ndim, size, values, head(), tail().
- **Dataframe basic functionality**: T, axes, dtypes, empty, ndim, shape, size, values, head(), tail().
- **Các function liên quan thống kê**: count(), sum(), mean(), median(), model(), std(), min(), max(), abs(), prod(), cumsum(), cumprod(), describe(), ptc_change(), cov(), corr(), rank(), var(), skew(), apply().
- **Các function filter data**: groupby(), get_group(), merge(), concat(), append(), melt(), pivot(), pivot_table().
- **Một số function khác**: get_option(), set_option(), reset_option(), describe_option(), option_context().

Xử lý dữ liệu với Pandas - GroupBy

- Pandas GroupBy là một hàm mạnh mẽ và linh hoạt trong Python.
  - Chia dữ liệu thành các nhóm dựa trên một số tiêu chí
  - Áp dụng một số tính toán cho từng nhóm một cách độc lập
  - Kết hợp các kết quả vào một cấu trúc dữ liệu



Xử lý dữ liệu với Pandas - GroupBy

DataFrame.groupby(by=None, level=None, as_index=True, sort=True, dropna=True)

- **by** : chỉ định tiêu chí, đối tượng cần nhóm, như cột 'order' trong ví dụ trên.
- **level** : kiểu số nguyên hoặc chuỗi: Nếu trục là MultiIndex (phân cấp), thì nhóm theo các cấp cụ thể.
- **as_index** : kiểu bool, default=True: Biến cột tiêu chí thành Index (chỉ số), as_index = False thì giữ nguyên index cũ.
- **sort** : kiểu bool, default=True: Sắp xếp cột tiêu chí theo thứ tự.
- **dropna** : kiểu bool, default=True: Loại bỏ những hàng và cột có giá trị Nan

Q & A

- https://madewithml.com/courses/foundations/numpy/
- https://madewithml.com/courses/foundations/pandas/
- https://madewithml.com/courses/foundations/python/