NGÔN NGỮ LẬP TRÌNH KHOA HỌC

Tên học phần	Ngôn ngữ lập trình khoa học
Thời lượng:	Lý thuyết: 30 tiết; Thực hành: 30 giờ.
Số bài kiểm tra:	02 bài
Điểm chuyên cần:	Có, dự phòng
Hình thức kiểm tra:	Trên máy, 45 phút
Hình thức thi:	Trên máy, 60 phút
Điều kiện tiên quyết:	Không
Học phần tiếp theo:	**Công cụ và kỹ thuật tính toán khoa học**
Tài liệu:	[1]. Slide bài giảng: Giảng viên cung cấp. [2]. Bài thực hành (8 bài): Giảng viên cung cấp. [3]. Các tài liệu trên mạng Internet: Giảng viên cung cấp

- **MỤC TIÊU:**

- Thành thạo một ngôn ngữ lập trình dành cho các tính toán khoa học
- Sử dụng để giải quyết được một số bài toán tính toán

- **NGÔN NGỮ SỬ DỤNG TRONG KHÓA HỌC:** **PYTHON**

- **TẠI SAO LÀ PYTHON ?**

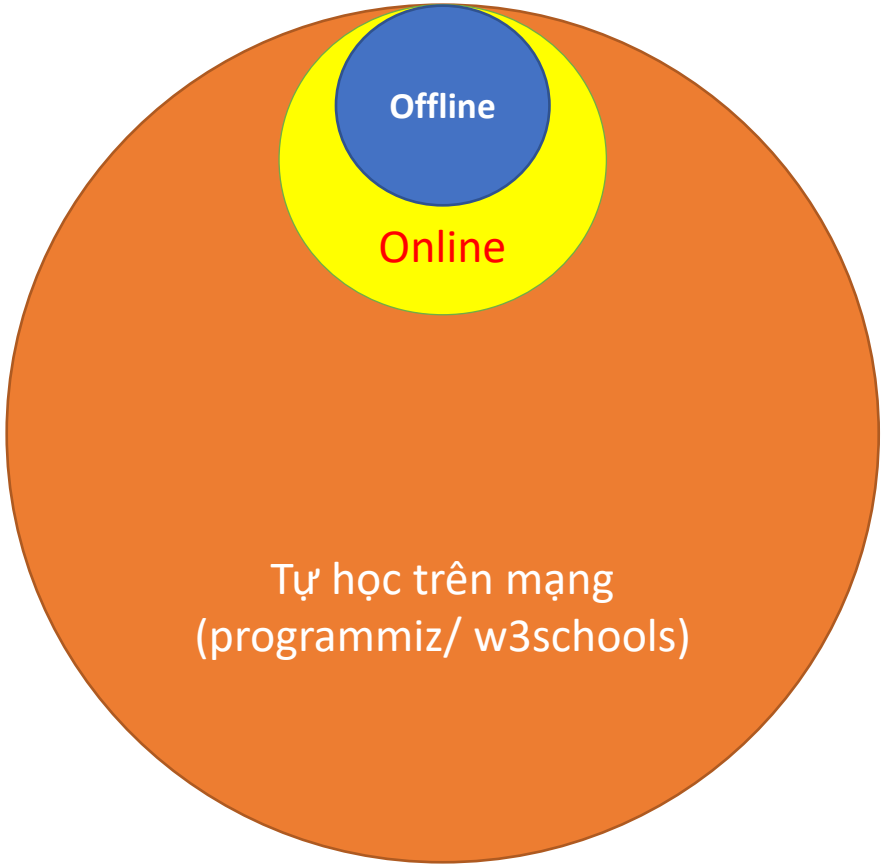
- Ngôn ngữ đơn giản, dễ tiếp cận
- Ngôn ngữ phổ biến hàng đầu
- Làm được nhiều việc: Machine Learning, Web development, Data analysis, Scientific research, Automation,...
- Sử dụng trong nhiều công ty nổi tiếng: Google, Meta, Netflix,...

TEST

Viết chương trình đơn giản sau bằng Python:

Nhập vào một chuỗi ký tự từ bàn phím. Cho biết chuỗi đó có bao nhiêu từ (từ được định nghĩa là chuỗi các ký tự liên tiếp dài nhất mà không chứa dấu cách.

TỰ HỌC



TÀI LIỆU/ CÔNG CỤ

- W3schools
- Programiz
- Pycharm
- Google Colab
- Jupyter Notebook
- …

THỰC HÀNH

PyCharm
Community version

BÀI 1

TỔNG QUAN PYTHON

BÀI 1: TỔNG QUAN PYTHON

- Python Syntax
- Python Expression
- Python In/Out
- Python Control Structures

Python Syntax – Cú pháp

- **Python Execution – Thực thi lệnh trong Python**
- **Python Indentation – Tab trong Python**
- **Python Variables – Biến trong Python**
- **Python Comments – Chú thích trong Python**

1. Python Syntax – cú pháp

- **Thực thi mã lệnh:**
  - Chế độ dòng lệnh (command line)
  - Code file: *.py

1. Python Syntax – cú pháp

- **Tab:**
  - Tab đầu dòng: Không phải chỉ để cho dễ đọc mà rất quan trọng trong Python
  - Khối lệnh: được tạo ra bằng các tab đầu dòng.

- **Biến:**
  - Không có lệnh để khai báo một biến
  - Các biến được tạo khi bạn gán giá trị cho nó
  - **Kiểu biến**
    - Biến không định kiểu
    - Các kiểu nguyên thủy
    - Ép kiểu/ lấy kiểu
  - **Quy tắc đặt tên**
    - Tên biến đặt theo quy tắc (tương tự C++)

1. Python Syntax – cú pháp

- **Chú thích:**
  - Bắt đầu dòng comment bằng: #
  - Khối comment (comment trên nhiều dòng):

```
#,
''' comment '''
""" comment """
```

HỌC NGÔN NGỮ PYTHON – PHẦN CƠ BẢN

- Biến
- Biểu thức
- Nhập/ xuất
- Các cấu trúc điều khiển

2. Python Expression – Biểu thức

- **Toán tử**
  - Arithmetic operators: +, -, *, /, %, **, //
  - Assignment operators: =, +=, -=, *=,....
  - Comparison operators: ==, !=, >, <, >=, <=
  - Logical operators: and, or, not
  - ...
- **Toán hạng**
  - Hằng, Biến, Hàm
  - Hằng số, hằng xâu, hằng ký tự, hằng date

3. Python In/Out – Nhập/xuất

- **Xuất dữ liệu ra màn hình**
  - Xuất xâu ký tự
  - Xuất dữ liệu từ biến
  - Cú pháp tổng quát lệnh print
  - Định dạng dữ liệu xuất
- **Nhập dữ liệu từ bàn phím**
  - Lệnh input + ép kiểu, split(), map()
- **Sử dụng các module**
  - Lệnh import

```
print(*objects,
            sep = ' ',
            end = '\n',
        file  =  sys.stdout,
        flush = False
           )
```

🕮 BÀI TẬP 1.1

- Nhập vào từ bàn phím hai số nguyên a, b. Tính và in ra màn hình tổng, hiệu, tích, thương của a và b:

- Mỗi kết quả in trên 1 dòng.
- Kết quả của thương: là số thực có độ chính xác 3 chữ số hàng thập phân.

- Nhập vào tọa độ của hai điểm A(x1, y1) và B(x2, y2). Tính và in ra khoảng cách Euclidean giữa A và B:

$$d(A, B) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

4. Python Control Structures – Cấu trúc điều khiển

- Python Control Structures
  - if … elif…else
  - for loops
  - while loops

🕮 BÀI TẬP 1.2

- Giải phương trình bậc 2
- Nhập vào một số nguyên n cho tới khi n ∈ [20, 30], nhập vào số thực x.

  Tính và in ra:

$$P = 2022\ x^n + \frac{1}{x} + \frac{2}{x^2} + \cdots + \frac{n}{x^n}$$

- Nhập vào một số nguyên dương n. kiểm tra xem n có phải là số nguyên tố hay không?

BÀI 2

FUNCTIONS & MODULES

BÀI 2: FUNCTIONS & MODULES

- Python functions
- Arguments
- Global variables
- Python modules
- Python package

- Định nghĩa một hàm
  - **Ví dụ 2.1:** Viết hàm tính n!. Sử dụng hàm đó để tính:

$$P = (n+1)! + (2n)!$$

```
def fuctorial(n):
    g = 1
    for i in range(1, n + 1):
        g = g * i
    return g
-----------------------------------------
n = int(input("n = "))
P = fuctorial(n+1) + fuctorial(2*n)
print("Result: P= ", P)
```

• Python functions – Hàm trong Python

- Định nghĩa một hàm

```
def  ⟨tên_hàm⟩ ( [đối số] ):
        ⟨lệnh 1⟩
        ⟨lệnh 2⟩
          ...
```

  - ⟨tên_hàm⟩ : Đặt theo quy tắc
  - [đối số] : Thông qua nó ta truyền các giá trị vào hàm
  - ⟨lệnh 1⟩, ⟨lệnh 2⟩,... : Các câu lệnh trong thân hàm
  - Lệnh return có thể được sử dụng để trả về các giá trị

```
return  ⟨giá_trị_1⟩, ⟨giá_trị_2⟩, ...
```

- Định nghĩa một hàm
  - **Ví dụ 2.2:** Viết hàm tìm số lớn nhất trong ba số nguyên a, b, c. Sử dụng hàm đó để tìm số lớn nhất trong 5 số x, y, z, t, q.

```
def max(a, b, c):
        m = a
        if b > m:
                m = b
        if c > m:
                m = c
        return m
x = int(input("x = "))
y = int(input("y = "))
z = int(input("z = "))
t = int(input("t = "))
q = int(input("q = "))
F = max(x, y, z)
S = max(F, t, q)
print("Max of 5 numbers: ", S)
```

🕮 BÀI TẬP 2.1

- Viết hàm tính khoảng cách Euclidean giữa hai điểm A(x1, y1) và B(x2, y2).
- Viết hàm kiểm tra xem hai điểm A, B, điểm nào gần tâm O hơn.
- Chương trình chính: Nhập vào tọa độ của hai điểm A, B như trên. Sử dụng hàm trên để tính và in ra:
  - Khoảng cách A, B
  - Chu vi tam giác OAB
  - Cho biết A hay B gần tâm O hơn

- Truyền tham số cho hàm

```
def greet(name, msg):
    print("Hello", name + ', ' + msg)


greet("Monica", "Good morning!")
```

- Nếu hàm có đối số (argument), khi gọi hàm, ta cần truyền giá trị cho đối số (tham số, paramenter).
- Thông thường, truyền tham số cần:
  - Đủ về số lượng
  - Đúng thứ tự

- Đối số có giá trị mặc định – Default arguments

```
def greet(name, msg = “How are you ?”):
        print("Hello", name + ', ' + msg)
greet("Monica")
greet("Monica“, “My name is Ricky”)
```

- Nếu đối số có giá trị mặc định, khi gọi hàm ta có hai lựa chọn:
  - Không truyền tham số cho đối số đó
  - Truyền tham số như bình thường.
- Không được định nghĩa một đối số “không mặc định” theo sau một đối số “mặc định”.

- Truyền tham số theo từ khóa

```
def greet(name, msg):
        print("Hello", name + ', ' + msg)
greet(name = "Monica“, msg = “How are you?”)
greet(msg = “How are you?”, name = "Monica“)
greet("Monica“, msg = “How are you?”)
```

  - Sử dụng từ khóa là “tên đối” để truyền tham số:

⟨Tên_đối_số⟩ = ⟨Giá_trị⟩

  - Không được truyền tham số theo từ khóa, theo sau đó là 1 tham số không theo từ khóa.

- Đối số là một tập hợp (các bộ - tuple)

```
def  greet(*names):
        print("Hello", names)
greet("Monica", "Luke", "Steve", "John")
```

- Sử dụng đối số là một tuple:
  - *⟨tên_đối⟩
  - Số lượng tham số truyền vào là linh động

• Global Variables – Biến toàn cục

• Sử dụng biến toàn cục

- Biến được khai báo bên ngoài hàm
- Có thể được truy cập bên trong hoặc bên ngoài hàm
- Trong hàm, muốn sử dụng biến toàn cục, cần khai báo:

global ⟨tên_biến_toàn_cục⟩

```
k = 0                    # Biến k là biến toàn cục

def Func1():
    k = 3                # biến k là cục bộ
    print("K inside Func1= ", k)
def Func2():
    global k    # Chỉ định k là biến toàn cục
    k = 5
    print("K inside Func2= ", k)

Func1()
print("K outside: ", k)
Func2()
```

- Lập trình theo modules
  - Module: một tệp chứa các câu lệnh và định nghĩa Python. Mỗi file .py có thể coi là một module, tên module chính là tên file
  - Một chương trình được tạo nên từ nhiều module. Một module có thể được sử dụng trong nhiều chương trình
  - Để sử dụng một module: **import toàn module**

import ⟨tên_module⟩ [as ⟨bí_danh⟩]

✍ Trong trường hợp không tìm ra module, chương trình sẽ báo lỗi.

(Đặt đường dẫn tới thư mục chứa module: xem trực tiếp trên Pycharm.)

- import

mymodule.py

```
PI = 3.14


def add(a, b):
    return a + b
```

myprogram.py

```
import mymodule
a = 3
b = 5
print("Tong = ", mymodule.add(a, b))
print("PI   = ", mymodule.PI)
```

✍ Để xem trong module có chứa những gì, ta dùng **dir**:

```
import mymodule as m
print(dir(m))
```

myprogram.py

```
import mymodule as m
a = 3
b = 5
print("Tong = ", m.add(a, b))
print("PI   = ", m.PI)
```

- from/ import

mymodule.py

```
PI = 3.14


def add(a, b):
    return a + b
```

myprogram.py

```
from mymodule import add
from mymodule import PI as P
a = 3
b = 5
print("Tong = ", add(a, b))
print("PI   = ", P)
```

- **Chỉ import một vài thứ trong module:**

from ⟨tên_module⟩ import ⟨tên⟩ [as ⟨bí_danh⟩]

- import * from

mymodule.py

```
PI = 3.14


def add(a, b):
    return a + b
```

myprogram.py

```
from mymodule import *
a = 3
b = 5
print("Tong = ", add(a, b))
print("PI   = ", PI)
```

- **import tất cả các tên trong một module:**

from ⟨tên_module⟩ import *

🕮 BÀI TẬP 2.2

- **Tổ chức chương trình thành các module:**

Module 1:

- **Định nghĩa tỷ giá: USD = 23000, EUR = 26000, RUB = 170**

Module 2:

- **Các hàm quy đổi n USD/ EUR/ RUB ra VND (ba hàm)**
- **Định nghĩa các hàm cộng ba số thực, cộng hai số thực**

Chương trình chính:

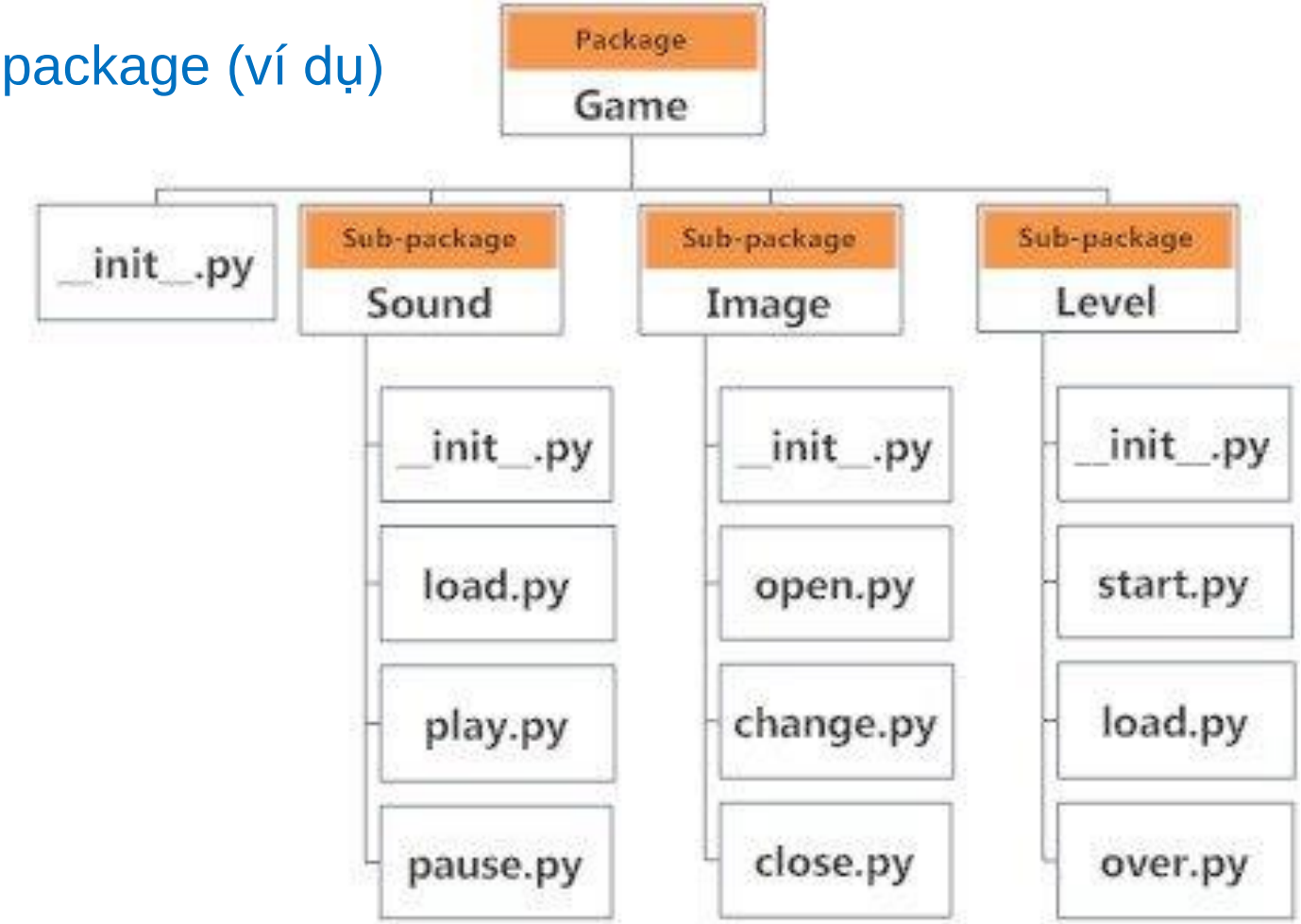
- **Sử dụng các hàm trong hai module để: Nhập vào số tiền USD, EUR, RUB hiện có; In ra tổng số tiền VND sau quy đổi.**

- Package

  - Package: là gói chứa các gói con (sub-package) hoặc các module
  - Trong một package/ sub-package có chứa:

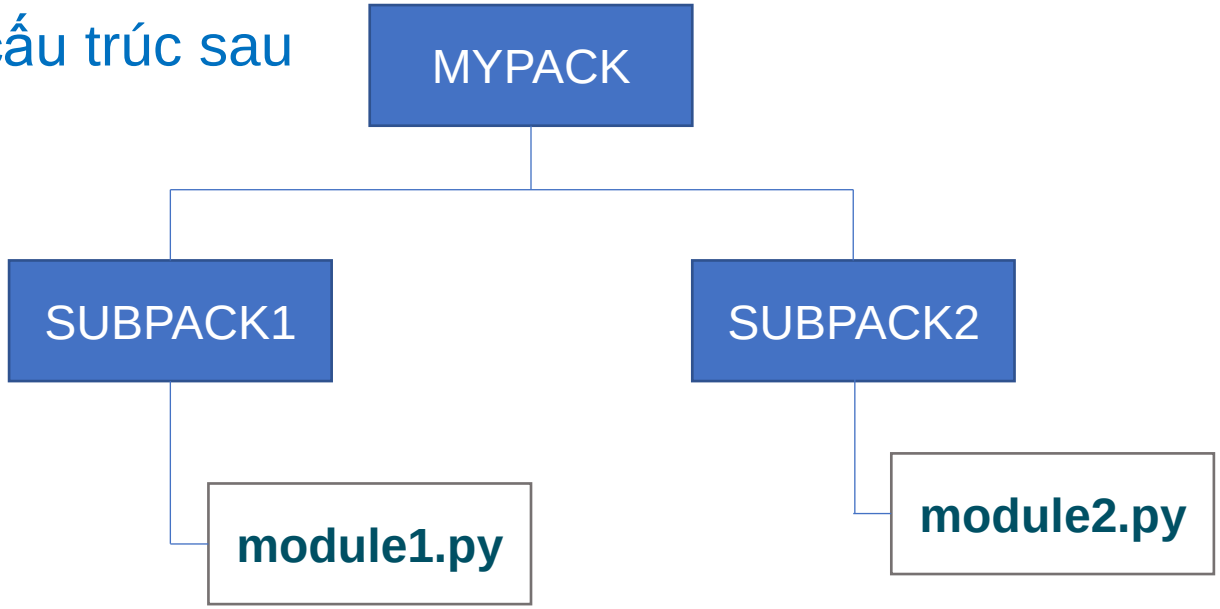
    - File chỉ dấu: __init__.py (bắt buộc)
    - Các file module.
    - Subpackages

- Package
  - Cấu trúc của một package (ví dụ)



🕮 BÀI TẬP 2.3

- Tạo một package theo cấu trúc sau



Với module1.py và module2.py là hai module đã tạo ra ở BÀI TẬP 2.2

- Viết chương trình như trong BÀI TẬP 2.2 để sử dụng package

BÀI 3

DATA STRUCTURES

BÀI 3: DATA STRUCTURES

- Python list
- Python tuple
- Python set
- Python dictionary
- Python string

Python list

- **List (danh sách)**
  - Cấu trúc để lưu trữ một dãy các phần tử
  - Khởi tạo một list: **a = [1, 3, 2, 4, 5]**
    b = [“Hoa”, “Hải”, “Hồng”, “Hoàng”]

k = [2*i **for i in range (1, 10)**] → [2, 4, 6, 8, 10, 12, 14, 16, 18]

  - Các phần tử trong một list không nhất thiết phải cùng kiểu:

c = [1, 2, “Hoa”, 3.1, “Hồng”, 5]
d = [ ]

  - Các phần tử lại có thể là một list (**nested list**):

e = [1, 2, [4, 5, 6], 9, 8]

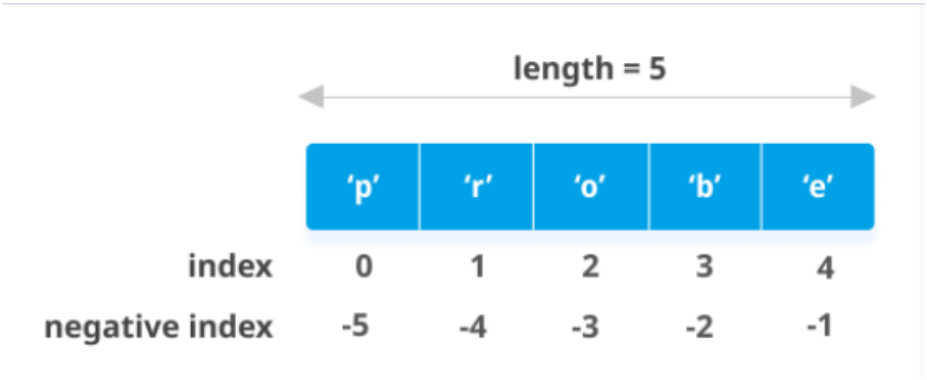
- **List index (chỉ số)**
  - Chỉ số được tính từ 0
  - Chỉ số của nested list:

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

c[1] = 2, c[2] = [3, 4, 5], c[2][1] = 4

  - Chỉ số âm: **c = [1, 2, [3, 4, 5], 6]**

c[-1] = 6, c[-2] = [3, 4, 5], c[-2][2] = c[-2][-1] = 5



🕮 BÀI TẬP 3.1

- Khởi tạo một list a với bộ 5 giá trị bất kỳ, in ra màn hình
- Khởi tạo và in một list b với 100 số nguyên chẵn đầu tiên [2, 4, 6, ...]
- Sử dụng list để nhập vào từ bàn phím một mảng c gồm n phần tử nguyên. Tính tổng các phần tử trong mảng.
- Sử dụng list để nhập vào từ bàn phím một mảng hai chiều d(n x m) các số thực. Hiển thị mảng d lên màn hình.

- **List slicing (chia cắt list)**
  - List slicing: Truy cập vào các phần tử liên tiếp trong list

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

c[1 : 3] = c[1], c[2]

c[1 :] = c[1], c[2], c[3]

c[:] = c

✍ List slicing: không tính phần tử ở đầu mút bên phải

- **Thao tác cơ bản trên list**

BỔ SUNG

- Thêm 1 phần tử vào cuối: append()
- Chèn 1 phần tử : insert()
- Thêm nhiều phần tử: extend()

XÓA

- Xóa phần tử: del
- Xóa phần tử theo giá trị: remove()
- Xóa phần tử theo chỉ số: pop ()

- **Thao tác cơ bản trên list**

TÌM KIẾM

- Lấy chỉ số đầu tiên theo giá trị: index()
- Đếm số lần xuất hiện của 1 giá trị: count()

SẮP XẾP

- Sắp xếp list: sort()
- Đảo ngược list: reverse()
- Sao chép một list: copy()
- Ghép hai lits: +
- Nhân bản list: *

🕮 BÀI TẬP 3.2

- Nhập vào từ bàn phím một mảng a gồm n số nguyên. Sắp a tăng dần và in ra màn hình
- Tìm max, tìm min, tìm vị trí của phần tử chẵn đầu tiên trong a, cho biết trong mảng có chứa số 3 hay không.
- Chèn một phần tử vào vị trí k; xóa toàn bộ các phần tử chẵn trong mảng.
- Nhập thêm mảng b gồm m phần tử nguyên từ bàn phím. Nhân bản b lên gấp đôi, đảo ngược b sau đó ghép a với b để thu được mảng c.

Python tuple

- **Python tuple**
  - Gần tương tự như list
  - Không giống như list, bộ giá trị của tuple là bất biến.
  - Tuy nhiên, nếu bản thân phần tử là một kiểu dữ liệu có thể thay đổi (vd list), thì các giá trị bên trong phần tử đó có thể được thay đổi.
  - Khởi tạo tuple:

```
a = (1, 3, 2, 4, 5)
b = (“Hoa”, [1, 2, 3], “Hồng”, (2, 2))
```

- **Python list vs. tuple**
  - tuple thường dùng để lưu tập giá trị mà các phần tử khác kiểu; list thường dùng để lưu tập giá trị cùng kiểu.
  - Các phần tử của tuple là bất biến; các phần tử của list là có thể thay đổi.
  - Duyệt trên tuple nhanh hơn một chút so với duyệt list.
  - Nếu dữ liệu là một tập các phần tử không thay đổi, sử dụng tuple để lưu trữ.

🕮 BÀI TẬP 3.3

- Khởi tạo một tuple c gồm 10 số nguyên bất kỳ. In c ra màn hình và cho biết số phần tử của c.
- Cho biết c có bao nhiêu phần tử chẵn; Nhập vào một giá trị x từ bàn phím và cho biết c có chứa x hay không.
- Khởi tạo một tuple chứa các số thực là tập các trọng lượng của các bình gas. Cho biết trọng lượng lớn nhất, nhỏ nhất, số bình gas có trọng lượng 12 (kg) có trong tuple.

Python set

- **Python set: tập hợp không có chỉ số**
  - set là một tập các mục không có chỉ số, mọi phần tử tập hợp là duy nhất (đơn trị) và bất biến.
  - Các phần tử trong một set có thể khác kiểu.
  - Ta có thể thêm phần tử vào hoặc xóa phần tử từ một set.
  - Thường được sử dụng để thực hiện các phép toán tập hợp như: Hợp, Giao, Trừ, ...

- **Các thao tác cơ bản trên set**
  - Khởi tạo set
  - Thêm một phần tử: add()
  - Thêm nhiều phần tử: update()
  - Xóa một phần tử đang tồn tại: remove()
  - Xóa một phần tử nếu tồn tại: discard()
  - Xóa tất cả: clear()
  - Lấy ra một phần tử ngẫu nhiên: pop()

- **Các thao tác không thể thực hiện trên set**

  - Khởi tạo một set rỗng: ví dụ a = { }
  - Khởi tạo phần tử trùng nhau: ví dụ a = {1, 2, 1, 3, 1}
  - Khởi tạo 1 phần tử của set là một list: ví dụ a = {1, [2, 3], 4}
  - Truy cập vào phần tử thông qua chỉ số: ví dụ a[1], a[2],...

- **Các phép toán tập hợp trên set**
  - Phép hợp: | hoặc union() vd: a = b | c, a = b.union(c)
  - Phép giao: & hoặc intersection() vd: a = b & c, a = b.intersection(c)
  - Phép trừ: - hoặc difference() vd: a = b - c, a = b.difference(c)
- **Các phép kiểm tra trên set**
  - Kiểm tra xem hai tập hợp có rời nhau: isdisjoint()
  - Kiểm tra xem một tập có là tập con của tập khác: issubset()
  - Kiểm tra xem một tập có là tập mẹ của tập khác: issuperset()

- **Frozenset: Một set bị đóng băng**

Bài đọc thêm

- Cách tạo frozenset
- Tính chất của frozenset
- Các thao tác cơ bản trên frozenset

🕮 BÀI TẬP 3.4

- Khởi tạo một set với các giá trị hỗn hợp kiểu số và xâu ký tự. Thực hiện các thao tác sau:
  - Bổ sung một phần tử vào set
  - Bổ sung hai phần tử vào set
  - Xóa một phần tử trong set
  - Lấy ra một giá trị trong set
  - Xóa toàn bộ set

Ghi chú: Mỗi thao tác đều in kết quả ra màn hình.

🕮 BÀI TẬP 3.5

- Khởi tạo hai set a và b với các giá trị bất kỳ. Thực hiện các thao tác sau:
  - In ra hợp, giao, hiệu của hai tập hợp
  - Kiểm tra xem tập a có là tập con của b không
  - Kiểm tra xem tập a có chứa tập b không
  - Kiểm tra xem một giá trị nhập từ bàn phím có nằm trong set a hay set b không.

Python dictionary

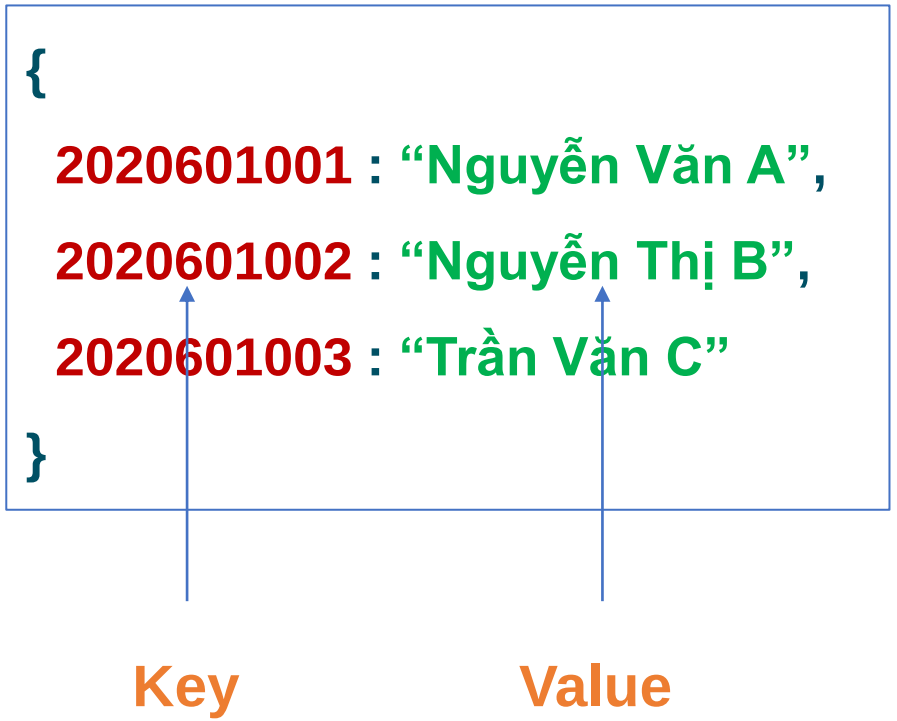
- **Python dictionary**

  - Dictionary: một cấu trúc dữ liệu mà mỗi mục (item) là một cặp:

    Key : Value

  - Học gì về dictionary ?

    Created: Tạo
    Accessing: Truy cập
    Adding: Thêm
    Removing: Xóa
    built-in functions: DS hàm



- **Tạo dictionary**

a = { item1, item2, item3,...., itemn}

item có dạng: ⟨Key⟩ : ⟨Value⟩

⟨Value⟩: có kiểu bất kỳ

⟨Key⟩: có kiểu số, xâu ký tự hoặc tuple; phải đơn trị. Các Key trong một dictionary không nhất thiết phải cùng kiểu.

Khởi tạo một dictionary rỗng:

```
a = {} hoặc a = dict() hoặc a = dict({})
```

- **Truy xuất các mục**
  - dictionary không được truy xuất qua chỉ số (index) mà thông qua khóa (Key).
  - Truy xuất bằng [Key]: a[2], a[“name”], ...
  - Truy xuất bằng get(Key): a.get(2), a.get(“name”)
- **Thêm/Sửa mục**

```
⟨tên_dict⟩ [⟨Key⟩] = ⟨Value⟩
```

- **Xóa mục**
  - Xóa mục theo Key: ⟨tên_dict⟩.pop(⟨Key⟩), del ⟨tên_dict⟩[⟨Key⟩]
  - Xóa mục ngẫu nhiên: ⟨tên_dict⟩.popitem()
  - Xóa toàn bộ các mục: ⟨tên_dict⟩.clear()
  - Xóa toàn bộ dictionary: del ⟨tên_dict⟩
- **Các phương thức khác**

`fromkeys(seq[, v])` `values()` `keys()` `items()`

- **Built-in functions**

  - Kiểm tra xem mọi key đều True: all()
  - Kiểm tra xem có ít nhất 1 key True: any()
  - Lấy số phần tử trong dict: len()
  - Sắp xếp các key trong dict (key phải cùng kiểu): sorted()

🕮 BÀI TẬP 3.6

- Tạo một từ điển chứa tập các mục về ID, Họ và tên của các sinh viên trong một lớp học. In từ điển ra màn hình.
- Cho biết sinh viên có mã “2020601001” có trong từ điển hay không? Nếu có, hãy cho biết Họ và tên của sinh viên đó.
- Hãy sao chép các sinh viên có mã là số chẵn sang một từ điển mới. In kết quả ra màn hình.

🕮 BÀI TẬP 3.7

- Tạo một từ điển chứa nội dung của một file config như sau:
- Bổ sung tham số: Status: Active
- Xóa thông tin về thời gian (Time)
- Sửa lại thông tin về Name = “Sa”
- Kiểm tra xem trong từ điển có chứa thông tin về “Server” hay chưa.

Date	15:03:22
Time	09:24:05
Server	HaUIJungle
Name	Root
Pass	****

Python string

- **Hằng xâu**
  - Các ký tự đặt giữa: ‘ ‘, “ ”, ”” ””, ”””” ””””
- **Truy xuất các ký tự trong xâu**
  - Sử dụng chỉ số (index): Chỉ số bắt đầu từ 0
  - **Không thể thay đổi các ký tự trong xâu một khi nó được gán**
- **Duyệt xâu:**

```
for letter in ⟨Xâu⟩:
        # truy xuất letter
```

```
for i in range(len(a)):
        # truy xuất a[i]
```

- **Ghép/nhân bản xâu**
  - Ghép hai xâu: + a = “Hello. ” + “How are you”
  - Nhân bản xâu: * a = a * 3
- **Định dạng xâu:** Dùng phương thức format
  - Phần định dạng chứa dấu ngoặc nhọn {} dưới dạng giữ chỗ

⟨Xâu⟩.format(⟨gt1⟩, ⟨gt2⟩, ...)

Giữ chỗ: `{}, {0}, {1}, ...`

Căn lề: `{:<10}, {:^10}, {:>10}`

Định dạng số: `{0:b}, {0:e}, {0:o} {0:.3f}`

- **String methods**

  - Chuyển xâu về ký tự thường: `lower()`
  - Chuyển xâu về ký tự hoa: `upper()`
  - Chia tách một xâu thành các từ: `split()`
  - Tìm kiếm xâu con: `find()`
  - Thay thế cụm ký tự: `replace()`

🕮 BÀI TẬP 3.8

- Nhập một xâu ký tự bất kỳ từ bàn phím. Cho biết xâu vừa nhập có bao nhiêu từ (giả sử từ là các cụm ký tự ngăn cách nhau bởi dấu cách).
- Hãy băm một xâu để thu được một từ điển các từ của xâu
- Nhập vào một biểu thức toán học dưới dạng một xâu ký tự. Ví dụ:

((a + b) * (c – d)) / (a + b)

Biểu thức được gọi là hợp lệ nếu các dấu mở/ đóng ngoặc được đặt phù hợp. Hãy cho biết biểu thức vừa nhập có hợp lệ không.

BÀI 4

FILE INPUT/ OUTPUT

BÀI 4. FILE INPUT/ OUTPUT

- File Open/ Close
- Write data to file
- Read data from file
- Directory

Mở một file

```
f = open(⟨filename⟩, [mode], [encoding])
```

⟨filename⟩	:	Có thể chứa đường dẫn
[mode]	:	r, w, a, mặc định là r
[encoding]	:	'utf-8'

```
f = open("test.txt", mode='r', encoding='utf-8')
```

Đóng file

```
f .close()
```

- **Ghi dữ liệu vào file**
  - **Mở file để ghi: [mode]=** w hoặc a
  - **Ghi file:**

```
f.write(⟨Dữ_Liệu⟩)
```

⟨Dữ_Liệu⟩: Cần có kiểu văn bản

VD: f.write("my first file\n")

🕮 BÀI TẬP 4.1

- Nhập một mảng a gồm n phần tử nguyên từ bàn phím. Ghi dữ liệu của a vào tệp.
- Nhập một ma trận b(nxm) gồm các phần tử thực từ bàn phím. Ghi dữ liệu của b vào tệp.

- **Đọc dữ liệu từ file**
  - **`Mở file để đọc: [mode]=`** r hoặc mặc định
  - **`Con trỏ tệp: trỏ tới vị trí đang đọc`**

Kiểm tra vị trí con trỏ tệp: f.tell()

Di chuyển con trỏ tệp: f.seek(⟨vị_trí⟩)

  - **`Phương thức read()`**

Đọc n ký tự: f.read(n)

Đọc toàn bộ phần còn lại của tệp: f.read()

- **Đọc dữ liệu từ file**
  - **`Phương thức readline()`**

Đọc 1 dòng (cả dấu xuống dòng): f.readline()

  - **`Phương thức readlines()`**

Đọc toàn bộ các dòng (từ vị trí con trỏ): f.readlines()

Dữ liệu trả về một list, mỗi phần tử là 1 dòng

🕮 BÀI TẬP 4.2

- Tạo một ma trận a(nxm) số thực và lưu trữ trong một tệp như dưới đây.

- Đọc dữ liệu từ tệp lên các biến n, m , b(nxm)

```
3 5
1 3 2 5 4
3 2 5 3 6
2 3 5 4 7
```

🕮 BÀI TẬP 4.3

- Cho bộ dữ liệu theo đường link sau (sinh viên tải bộ dữ liệu về máy tính).
- Đọc dữ liệu từ bộ dữ liệu lưu vào các biến tương ứng

Directory - Thao tác trên thư mục

- **Các thao tác với thư mục**
  - `import os`
  - Lấy thư mục hiện tại: os.getcwd()
  - Chuyển tới một thư mục: os.chdir("Tên_Mới")
  - Hiển thị nội dung thư mục: os.listdir(["Tên"])
  - Tạo thư mục: os.mkdir(["Tên"])
  - Đổi tên thư mục/file: os.rename("Tên", "Tên_mới")
  - Xóa thư mục rỗng/file: os.rmdir()/ os.remove("Tên")

Directory - Thao tác trên thư mục

- **Các thao tác với thư mục**
  - `import shutil`
  - Xóa thư mục không rỗng: shutil. rmtree(“Tên”)
  - Copy một file: shutil. copy(src, des)
  - Copy một file: shutil. copyfile(src, des)

🕮 BÀI TẬP 4.4

- Tạo một thư mục trong ổ đĩa với tên bất kỳ
- Di chuyển file dữ liệu vào thư mục vừa tạo
- Hiển thị nội dung thư mục
- Đổi tên thư mục
- Xóa file
- Xóa thư mục

BÀI 5

MATRIX & VECTOR

BÀI 5

Creating

Indexing

Operation

BÀI 5. MATRIX & VECTOR

- Numpy
  - Array creating
  - Array indexing
  - Array operation
- Scipy: high-level scientific computing

(Đọc thêm)

- **Tạo mảng với Numpy**
  - `import numpy as np`
  - Khởi tạo mảng một chiều (vector):

```
A = np.array([1, 2, 3, 4, 5])
```

  - Khởi tạo mảng hai chiều (matrix):

```
B = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 8, 7, 6]])
```

  - Các thông số:

```
A.ndim, A.shape, len(A), A.itemsize, A.dtype, A.size
```

- **Tạo mảng với Numpy**
  - Hàm khởi tạo mảng
    - np.arange()

```
np.arange(n),
np.arange(start, end, step),  end exclusive
```

    - np.linspace()

```
np.linspace(start, end, numpoints)
np.linspace(start, end, numpoints, endpoint=False)
```

- **Tạo mảng với Numpy**
  - Hàm khởi tạo mảng
    - np.ones(), np.zeros(), np.eye(), np.diag()

```
np.ones(n),    np.ones((3,3))
np.zeros(n),   np.zeros((3,3))
np.eye(n)                                  ma trận đơn vị size n
np.diag([1, 2, 3, 4])                      ma trận đường chéo
```

    - np.random Khởi tạo ngẫu nhiên

```
np.random.rand(n)          Ngẫu nhiên n phần tử trong [0, 1]
np.random.randn(n)         Ngẫu nhiên n phần tử theo phân bố chuẩn
np.random.seed(n)          Bật chế đố seed cho các hàm random
```

🕮 BÀI TẬP 5.1

- Tạo một mảng numpy a(5) với các phần tử nguyên. In mảng ra màn hình; Tạo một mảng b với n phần tử, các phần tử nằm trong đoạn [1, 10]. In mảng ra màn hình.
- Tạo một ma trận c(3 x 5) các số nguyên bất kỳ. Cho biết: số chiều, kích thước mỗi chiều, kiểu của các phần tử trong mảng.
- Tạo một ma trận đơn vị d(nxn). In ma trận ra màn hình; Tạo một ma trận đường chéo e(nxn) với các giá trị trên đường chéo lấy từ một mảng một chiều. In mảng ra màn hình.
- Tạo một mảng một chiều f(n) với các giá trị tuân theo phân bố chuẩn (Gauss). In mảng ra màn hình.

Array index– Chỉ số của mảng

- **Truy cập mảng theo chỉ số**
  - Mảng 1-D

```
a = np.array([1, 2, 3, 4, 5])  Index:  i = 0, n-1
```

  - Mảng 2-D

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

```
b[0, 0] = 1,   b[0, 1] = 2,    b[1, 2] = 7, ....
```

- *Mảng 0-D hay Scalar: là các phần tử trong một mảng*

```
a = 42
```

Array index– Chỉ số của mảng

- **Truy cập mảng theo chỉ số**
  - Slicing: sử dụng dấu ':' như list

```
a = [1 2 3 4 5]
```

- `a[1:3]` → `[2 3],`
- `a[:]` → `[1 2 3 4 5]`
- `a[2:]` → `[3 4 5]`
- `a[:2]` → `[1 2]`

- **`a[start : end]`**
- **`a[start : end : step]`**

Array index– Chỉ số của mảng

- **Truy cập mảng theo chỉ số**
  - Slicing trên mảng hai chiều:

```
a = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(a[1, 1:4])          → [7, 8, 9]
```

```
a = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(a[0:2, 2])          → [3, 8]
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])      → [[2 3 4] [7 8 9]]
```

🕮 BÀI TẬP 5.2

- Nhậpmột ma trận a(nxm) phần tử nguyên từ bản phím. In mảng vừa nhập ra màn hình.
- Tách cột thứ k của ma trận ra một mảng riêng, in cột tách được ra màn hình.
- Lọc ra các phần tử có chỉ số lẻ của mảng (mảng tách được), in các phần tử lọc được ra màn hình.

Array operation – Thao tác trên mảng

- **Các thao tác cơ bản trên mảng**
  - Numerical operations
  - Reshape
  - Iterating Arrays
  - Join & Split
  - Searching
  - Sorting

- **Các thao tác cơ bản trên mảng**
  - Numerical operations: Các thao tác số học
    - Thao tác trên tất cả các phần tử (Scalar)

```
a = a + 3    a = a-3    a = a*3    a = a/3    a = a**3
a = a + b    a = a-b    a = a*b    a = a/b
a > b        a == b
```

    - Nhân hai ma trận

```
c = a.dot(b)
```

Số cột của a phải bằng số dòng của b

- **Các thao tác cơ bản trên mảng**
  - Reshape: Định hình lại mảng
    - Shape: số phần tử của mỗi chiều
    - Reshape: From 1-D to 2-D

```
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
b = a.reshape((4, 3))  → b(4×3)
```

Cần đảm bảo số phần tử của mảng a và b bằng nhau

    - flattening: from 2-D to 1-D

```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.reshape(a, -1) hoặc b = a.flatten()  → b = [1 2 3 4 5 6]
```

🕮 BÀI TẬP 5.3

- Nhập vào từ bàn phím hai mảng số nguyên a(n) và b(n) cùng kích thước. In hai mảng ra màn hình.
- Cho biết những chỉ số nào mà hai phần tử tương ứng của hai mảng bằng nhau.
- Tăng các phần tử của b lên 1 đơn vị, in mảng kết quả ra màn hình.
- Chuyển mảng b thành một mảng hai chiều, cho biết ndim và shape của mảng kết quả.
- Chuyển mảng hai chiều về mảng một chiều, in kết quả ra màn hình.

- **Các thao tác cơ bản trên mảng**
  - Iterating Arrays: Duyệt mảng
    - Duyệt mảng 1 chiều

```
a = np.array([1, 2, 3])
for x in a:  print(x)                    #duyệt theo giá trị
for i in range(a.size):  print(a[i])     #duyệt theo chỉ số
```

    - Duyệt mảng 2 chiều

```
a = np.array([[1, 2, 3], [4, 5, 6]])
```

```
for x in a:
  for y in x:
      print(y)
```

```
for i in range(len(a)):
  for j in range(a.itemsize):
      print(a[i, j])
```

- **Các thao tác cơ bản trên mảng**
  - Iterating Arrays: Duyệt mảng
    - Duyệt mảng bằng **nditer()**

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for x in np.nditer(a):
        print(x)
```

    - Duyệt mảng với chỉ số bằng **ndenumerate()**

```
a = np.array([1, 2, 3])
for idx, x in np.ndenumerate(a):
        print(idx, x)
```

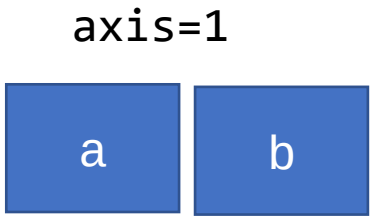
```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for idx, x in np.ndenumerate(a):
  print(idx, x)
```

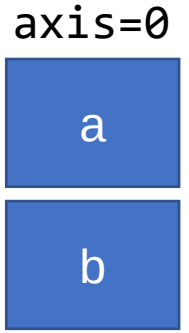
- **Các thao tác cơ bản trên mảng**
  - Join & Split: Nối, tách mảng
    - Nối mảng với concatenate()

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.concatenate((a, b))
print(c)  → c = [1 2 3 4 5 6]
```

    - Nối mảng hai chiều:

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





- **Các thao tác cơ bản trên mảng**
  - **Join & Split: Nối, tách mảng**
    - Tách 1 mảng thành nhiều mảng với array_split()

```
a = np.array([1, 2, 3, 4, 5, 6])
b = np.array_split(a, 3)
→ b chứa 3 mảng b[0] = [1 2], b[1] = [3 4], b[2] = [5 6]
```

    - Tách mảng hai chiều: Tách theo chiều ngang

```
a = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
b = np.array_split(a, 3)
→ b[0] = [[1 2], [3 4]]        b[1] = [[5 6], [7 8]]     ...
```

(thêm axis=1 sẽ tách theo chiều dọc)

🕮 BÀI TẬP 5.4

- Nhập vào một mảng a gồm n phần tử nguyên từ bản phím. Mảng a được gọi là sắp tăng nếu a[i] <= a[i+1] Sắp giảm nếu a[i] >= a[i+1], sắp tăng ngặt nếu a[i] < a[i+1], sắp giảm ngặt nếu a[i] > a[i+1] ∀ i = 0..n-1; còn lại là chưa được sắp. Hãy kiểm tra xem mảng a đã được sắp chưa? Sắp theo kiểu gì?
- Tách mảng a thành 3 mảng b, c, d. In ba mảng kết quả ra màn hình.
- Nối ba mảng lại theo thứ tự d, c, b thu được mảng e và in kết quả ra màn hình.
- Chuyển mảng e thành mảng hai chiều, sau đó cắt ra hai dòng đầu tiên. In kết quả.

- **Các thao tác cơ bản trên mảng**
  - Searching: Tìm kiếm
    - Tìm chỉ số theo điều kiện: where()

```
a = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(a == 4)
→ x[0] chứa các chỉ số mà tại đó mang giá trị 4


y = np.where(a % 2 == 0)
print(x)
→ y[0] chứa các chỉ số của các số chẵn trong a
```

- **Các thao tác cơ bản trên mảng**
  - Searching: Tìm kiếm
    - Tìm kiếm nhị phân trên mảng được sắp: searchsorted()

```
a = np.array([6, 7, 8, 9])
x = np.searchsorted(a, 7)
→ x: Vị trí có thể chèn số 7 vào để không phá vỡ tính được
     sắp của mảng.

a = np.array([1, 3, 5, 7])
x = np.searchsorted(a, [2, 4, 6])
→ x = [1, 2, 3] (các vị trí chèn 2, 4, 6 vào mảng)
```

- **Các thao tác cơ bản trên mảng**
  - Sorting: Sắp xếp
    - Sắp tăng dần

```
a = np.array([3, 2, 0, 1])
b = np.sort(a)        →      b = [0 1 2 3], a không đổi
```

    - Lựa chọn thuật toán sắp:

```
b = np.sort(a, kind='heapsort')
kind: quicksort (mặc định), heapsort, mergesort, stable
```

    - Sắp giảm dần

```
b = np.sort(a)[::-1]
```

🕮 BÀI TẬP 5.5

- Nhập vào một mảng a gồm n phần tử nguyên từ bàn phím.
- Cho biết vị trí của các phần tử chẵn trong mảng, tính tổng các phần tử chẵn.
- Sắp a giảm dần vào in kết quả ra màn hình.
- Nhập 1 phần tử nguyên k từ bàn phím, cho biết vị trí trong mảng a để chèn k vào a mà không phá vỡ tính được sắp của mảng.

BÀI 6

DATA VISUALIZATION

BÀI 6

Plotting

Decor

Subplot

Scatter

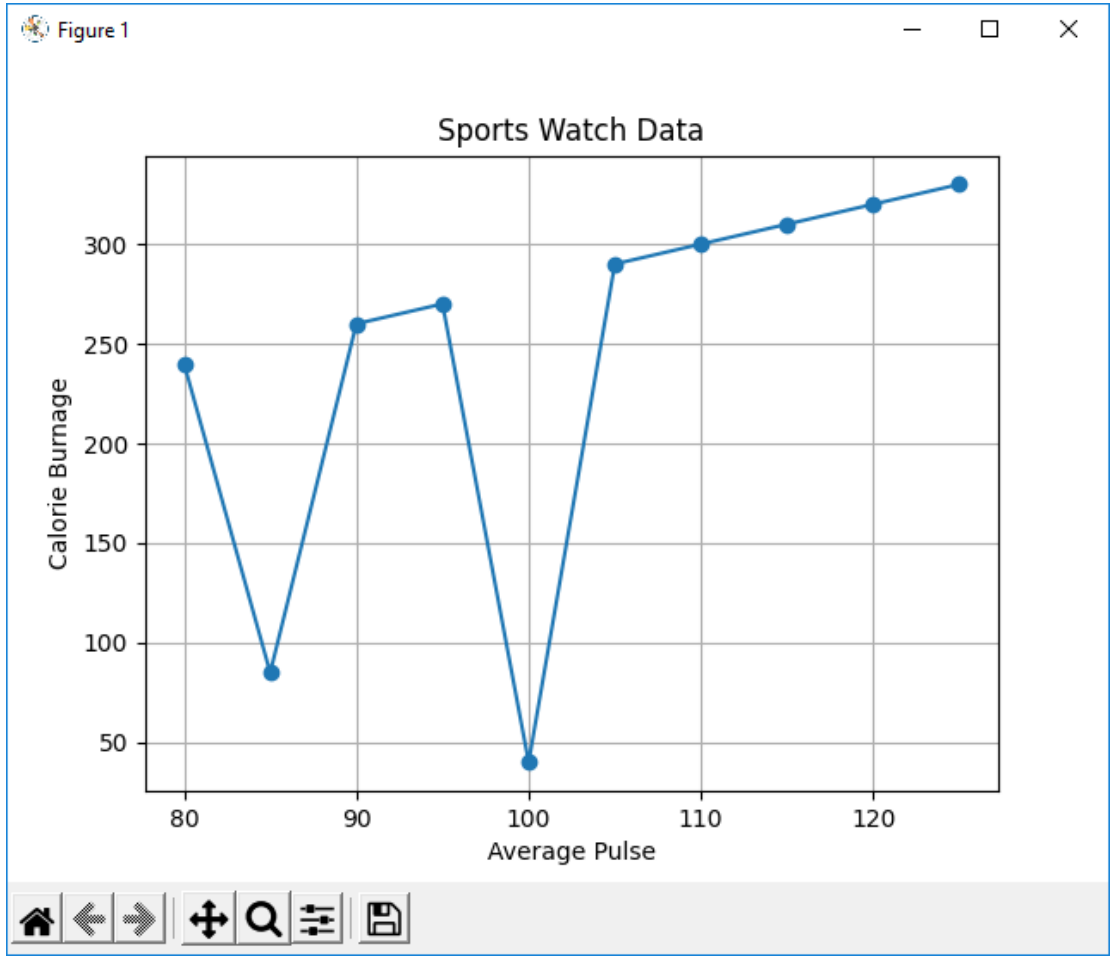
Bar

Histogram

Pie

🕮 BÀI 6. DATA VISUALIZATION

- Plotting
- Plot decor
- Subplot
- Scatter
- Bar
- Histogram
- Pie



- **Plotting**

```
import matplotlib.pyplot as plt
import numpy as np
```

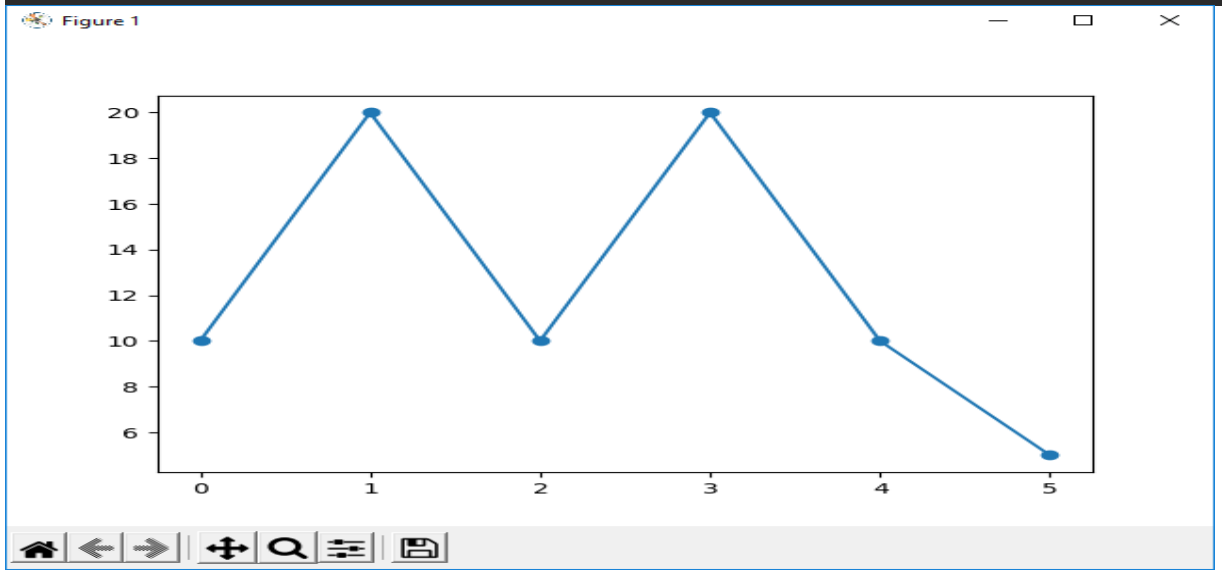
  - Đồ thị được tạo nên bằng cách nối các điểm: A(x, y)
  - x = mảng chứa n tọa độ trên trục x
  - y = Mảng chứa n tọa độ trên trục y tương ứng với x

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

✍ **x cần được sắp tăng, x và y phải có cùng kích thước**

- **Plotting**
  - Nếu thiếu mảng x, tọa độ x sẽ là các số nguyên liên tiếp từ 0

```
y = np.array([10, 20, 10, 20, 10, 5])
plt.plot(y)
plt.show()
```



- **Plotting**
  - Vẽ đồ thị hàm số

```
x = np.arange(100)
y = 2*x*x - 3*x + 5
plt.plot(x, y)
plt.show()
```

  - Hàm số tự định nghĩa

```
def f(x):
    return x*x + 2
x = np.array([1,  1.5,  2,  2.5,  3])
y = f(x)
plt.plot(x,y)
plt.show()
```

🕮 BÀI TẬP 6.1

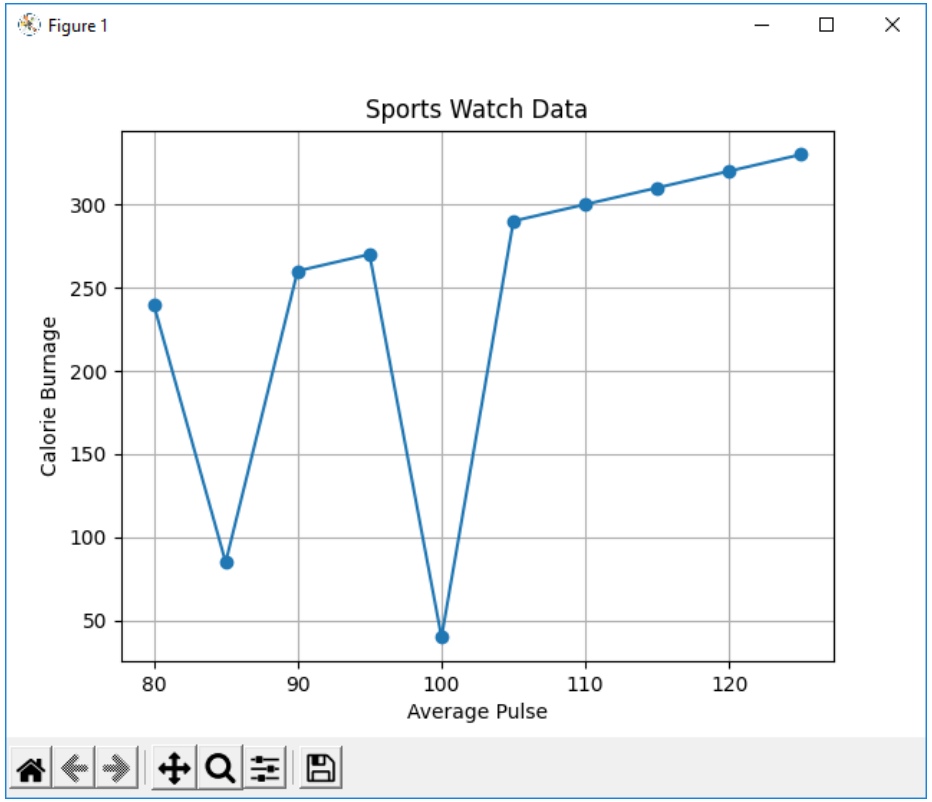
- Tạo 10 điểm có tọa độ bất kỳ, vẽ đồ thị bằng các điểm vừa tạo.
- Vẽ đồ thị hàm số $y = 2x^3 – 3x^2 + x + 1$ với 100 điểm thuộc đoạn [-10, 10]

- **Plotting decor**
  - Title

```
plt.title("TIÊU ĐỀ")
```

  - Label

```
plt.xlabel("Nhãn trên trục x")
plt.ylabel("Nhãn trên trục y")
```



- **Plotting decor**
  - Format font: font, font color, font size

```
font = {'family':'serif','color':'blue','size':20}
plt.title("Tiêu đề", fontdict=font)
plt.xlabel("Nhãn trên trục x", fontdict=font)
plt.ylabel("Nhãn trên trục y", fontdict=font)
```

  - Alignment: Ox: left, right, center; Oy: top, center, bottom

```
plt.xlabel("Nhãn trên trục x", loc='left')
plt.ylabel("Nhãn trên trục y", loc='top')
```

- **Plotting decor**
  - Markers

```
plt.plot(x, y, marker='o')
plt.plot(x, y, marker='o', ms=20)
plt.plot(x, y, marker='o', ms=20, mec = 'r', mfc='b')
```

  - Line

```
plt.plot(x, y, linestyle='dotted')
plt.plot(x, y, ls=':')
plt.plot(x, y, ls=':', color='r')
```

  - Marker | Line | Color

```
plt.plot(x, y, 'o:r')
```

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

- **Plotting decor**
  - Grid

```
plt.grid()

plt.grid(axis = 'x')

plt.grid(axis = ‘y')

plt.grid(color = 'g', linestyle = '--', linewidth = 0.5)
```

- **📖 BÀI TẬP 6.2**
  - Vẽ đồ thị hàm sin trên đoạn -10, 10
  - Trang trí đồ thị:
    - Đặt Tile, xLabel, yLabel tùy ý
    - Đặt font chữ, cỡ chữ, màu chữ, căn lề cho Title và các label
    - Đặt Marker hình *, đặt kích thước, màu viền, màu nền cho marker.
    - Đặt kiểu đường, màu đường, độ rộng đường
    - Đặt Grid kiểu dash, màu sắc và kích thước grid

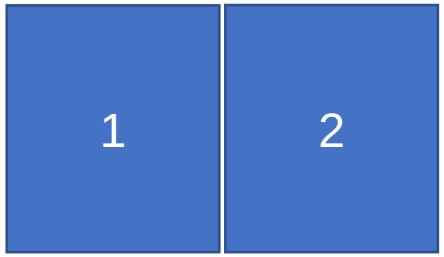
Subplot – Vẽ nhiều biểu đồ

- **Subplot**

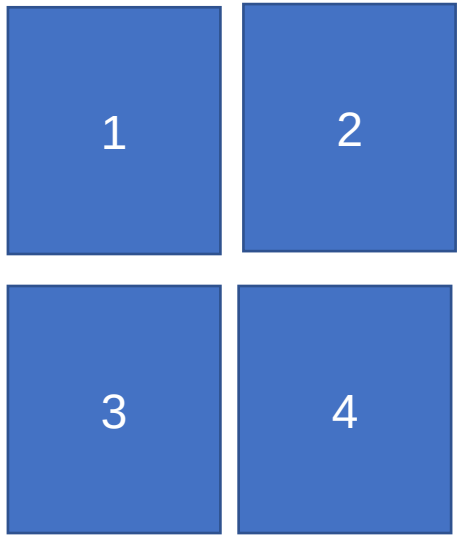
```
plt.subplot(1, 2, 1)
plt.plot(x,y)
```

```
plt.subplot(1, 2, 2)
plt.plot(x,y)
```

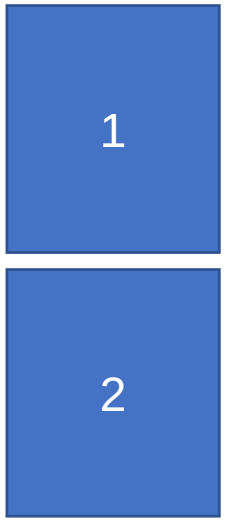
subplot(1, 2, ?)



subplot(2, 2, ?)



subplot(2, 1, ?)



Scatter – Biểu đồ dạng đám mây

- **Scatter charts**
  - Create a scatter chart

```
plt.scatter(x, y)
```

  - Color

```
plt.scatter(x, y, color = 'r')
```

  - Color bar

```
plt.colorbar()
```

  - Size of dots

```
plt.scatter(x, y, s=20)

sizes = np.array([20,50,100,200])  #sizes và x phải cùng kích thước
plt.scatter(x, y, s=sizes)
```

- **Scatter charts**
  - Dot transparency (độ đục alpha)

```
plt.scatter(x, y, alpha=0.5)
```

  - Color Size and Alpha

```
plt.scatter(x, y, c='b', s=20, alpha=0.5)
```

- 🕮 **BÀI TẬP 6.3**
  - Tạo mảng y chứa tổng sản lượng nông sản của từng tháng, trong 5 tháng, của 3 hợp tác xã khác nhau
  - Vẽ đồ thị dạng scatter với kích thước của mỗi điểm tương ứng với sản lượng nông sản của tháng tương ứng; đặt màu sắc cho đồ thị

- **Bar charts**
  - Create a bar chart

```
plt.bar(x, y)
```

Chú ý: x có thể là mảng xâu

  - Horizontal bars

```
plt.barh(x, y)
```

  - Bar width/ height

```
plt.bar(x, y, width = 0.1)
plt.barh(x, y, height = 0.1)
```

- **Histogram charts**
  - Histogram

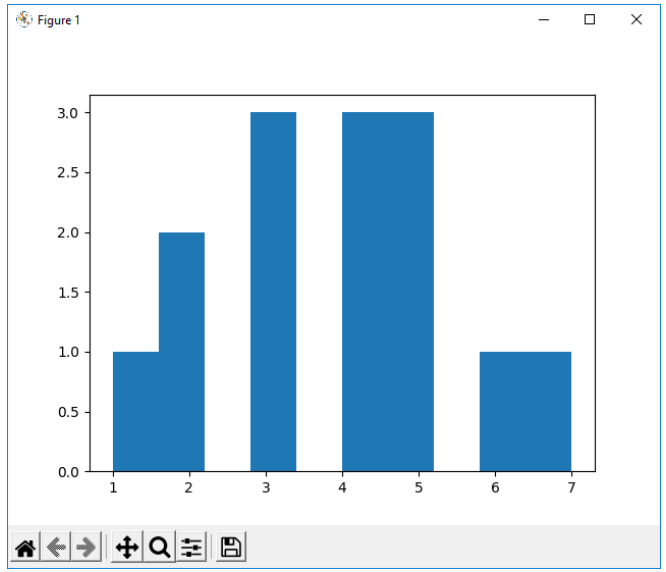
Biểu đồ tần suất xuất hiện của các giá trị

Dữ liệu chứa trong một mảng x:

```
x = np.array([1, 3, 2, 5, 4, 2, 5, 3, 6, 4, 7, 5, 4, 3])
```

Bảng tần suất thu được:

1	1
2	2
3	3
4	3
5	3
6	1
7	1



Histogram – Biểu đồ tần suất

- **Histogram charts**
  - Create a histogram

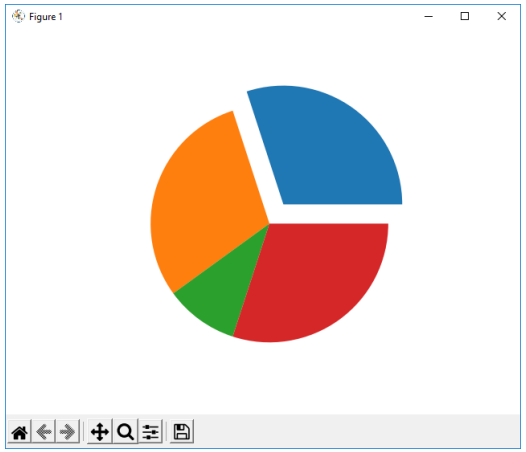
```
plt.hist(x)
```

  - Examples

```
Normal distribution (mean, sdv, numpoints)
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()

Poisson distribution (lamda, k- numpoints)
y = np.random.poisson(5, 250)
plt.hist(y)
plt.show()
```

Pie – Biểu đồ miếng



- **Pie charts**
  - Create a pie chart

```
plt.pie(x)
```

  - Labels

```
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
```

  - Start angle

```
plt.pie(y, labels = mylabels, startangle = 90)
```

  - Explode

```
myexplode = [0.2, 0, 0, 0]
plt.pie(y, labels = mylabels, explode = myexplode)
```

🕮 BÀI THU HOẠCH – ASSIGNMENT 2

Cho hai bộ dữ liệu trích 5% từ KDDCUP99 gồm 2 file: kddcup.data và kddcup.test. Viết:

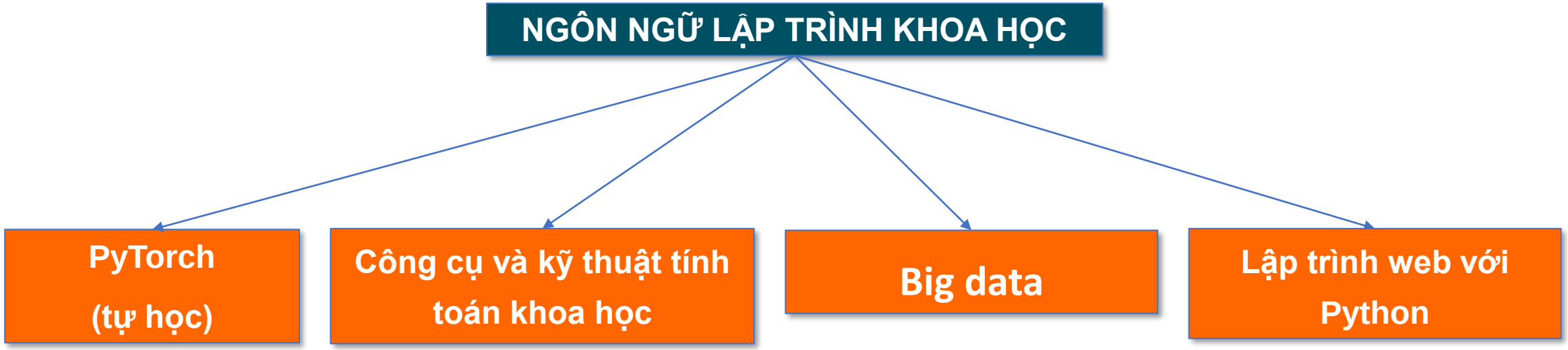
- Hàm read: đọc toàn bộ dữ liệu của 1 tệp lên mảng, trả về mảng đọc được.
- Hàm distance: tính khoảng cách Euclidean giữa hai dòng bất kỳ của một mảng x.
- Hàm distancetoall: tính khoảng cách Euclidean từ 1 dòng test tới tất cả các dòng của x.

[1]. Đọc toàn bộ tệp kddcup.data lên mảng x_train; cắt bỏ 4 cột đầu tiên và cắt cột cuối cùng của x_train ra một mảng label; chuyển x_train về kiểu float.

[2]. Đọc toàn bộ file kddcup.test lên mảng x_test; cắt bỏ 4 cột đầu và cột cuối cùng của x_test rồi chuyển mảng về kiểu thực.

[3]. Lấy 1 dòng bất kỳ từ mảng x_test hãy dự đoán label của nó bằng cách: tính khoảng cách từ nó tới tất cả các dòng của x_train. Label của nó là label của dòng gần với nó nhất.

CÁC HỌC PHẦN TIẾP THEO



AI framework của FaceBook

Pandas, Tensorflow/ Keras gói công cụ dành cho ứng dụng AI/ Data Analysis

Các open source framework cho phép xử lý big data như: APACHE Mapreduce/ Spark

Các framework phát triển ứng dụng web: Django/ Flask

KẾT THÚC