

HCMUT EE MACHINE LEARNING & IOT LAB

Buổi 3

Python (tiếp theo)

Presentation By: Văn Thịnh

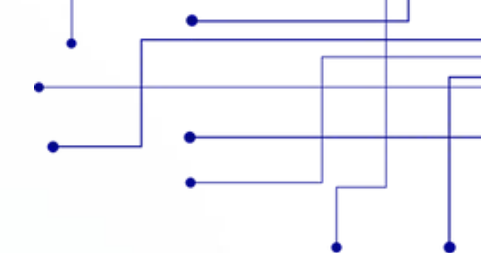
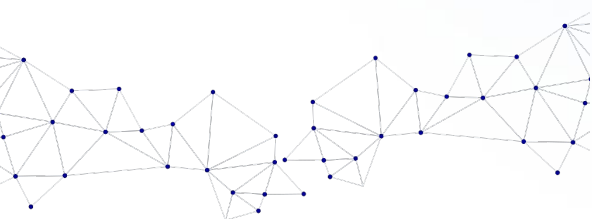


Table of Content

- I** Cấu trúc dữ liệu
- II** List Comprehension
- III** Lambda function
- IV** Lập trình hướng đối tượng
OOP
- V** Tensor



I. Cấu trúc dữ liệu

1. List

Là cấu trúc dữ liệu danh sách nằm trong ngoặc cặp vuông [], có thể thêm, xóa, sửa tùy ý (mutable)

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

```
2 fruits = ['apple', 'banana']
```

```
4 matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
6 nested_list = [[1, 2, [[[4]]], [5, 6]]]
```

2. Tuple

Cũng là dạng danh sách, nhưng không thể thay đổi sau khi khởi tạo (immutable)

```
1 a = (1, 2, 3, 4, 5)
2 colors = ('red', 'green', 'blue')
```

3. Set

Kiểu dữ liệu tập hợp, không có thứ tự (unordered), không lặp lại phần tử, mutable

```
1 a = {1, 2, 3}
2 b = {1, 2, 3, 3, 1}
3
4 print(a == b) # True
```

4. Dictionary

Kiểu dữ liệu dạng key-value có dạng {key:value, ...}, mutable

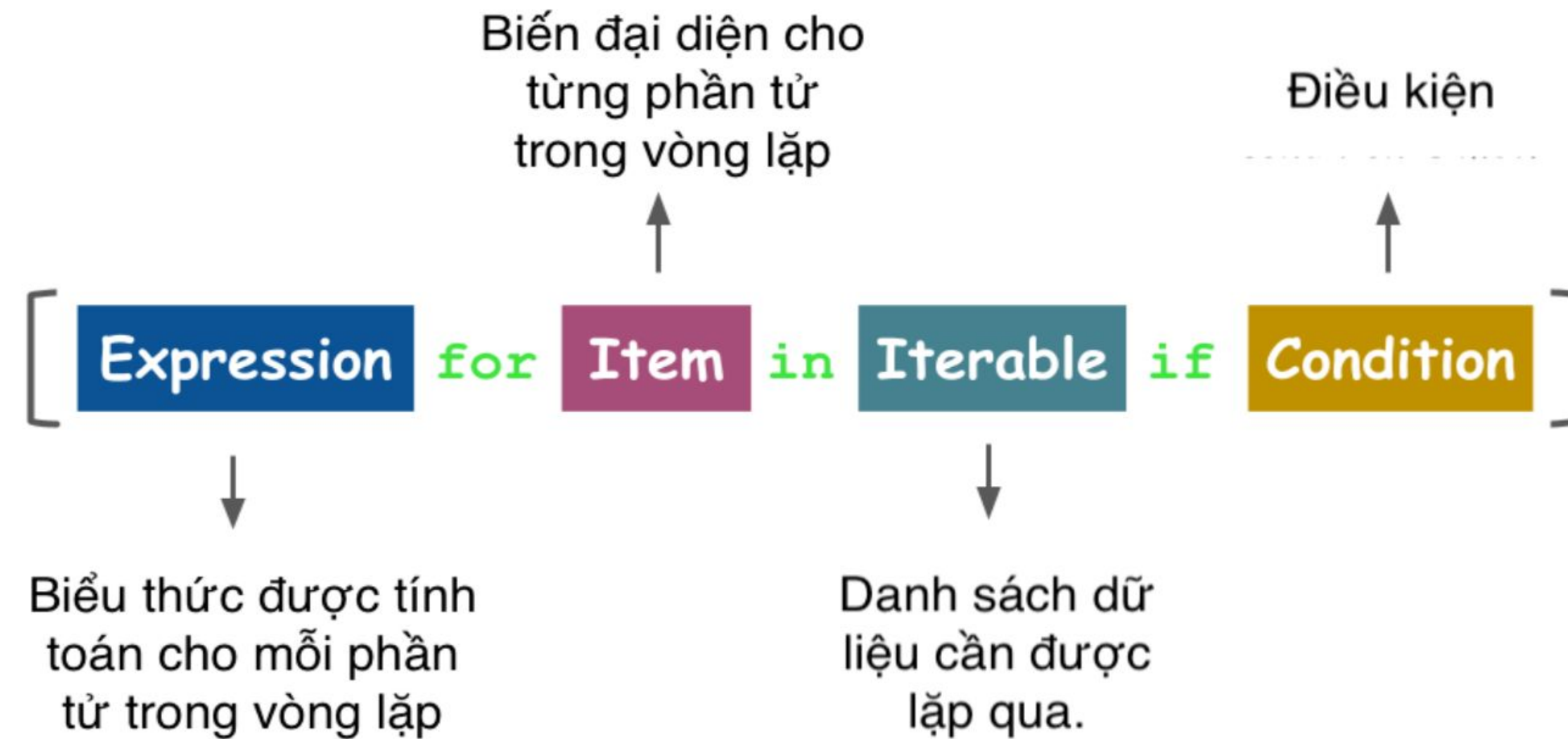
```
1 ages = {'Thinh' : 20,  
2         'Tien'  : 21,  
3         'Minh'  : 12  
4     }  
5  
6 print(ages['Minh'])
```



II. List Comprehension

II. List Comprehension

Ta có thể tạo một list mới với cú pháp như sau:



```
1 even_nums = [i for i in range(5) if i%2 == 0]
2
3 print(even_nums) # [0, 2, 4]
```

III. Lambda Function

1. Lambda function

Giúp tạo hàm không có tên (hàm ẩn danh)

Cú pháp:

lambda <args> : f(<args>)

```
3 print((lambda x, y: x + y)(1,2)) # 3
4
5 sum = lambda x, y : x + y
6 print(sum(1, 2)) # 3
```

2. Hàm bậc cao

1. Hàm *map()*: ánh xạ các phần tử của danh sách qua một hàm lambda

```
1 a = [1, 2, 3, 4, 5]
2
3 print(list(map(lambda x : x*2, a))) # [2, 4, 6, 8, 10]
```

2. Hàm *filter()*: lọc ra các giá trị thỏa điều kiện nhất định

```
1 a = [1, 2, 3, 4, 5]
2
3 print(list(filter(lambda x : x%2==0, a))) # [2, 4]
```

3. Hàm *reduce()*: hàm tích lũy

```
1 from functools import reduce
2
3 a = [1, 2, 3, 4, 5]
4 res = reduce(lambda x, acc : x + acc, a, 0)
5
6 print(res) # 15
```

IV. Lập trình hướng đối tượng OOP

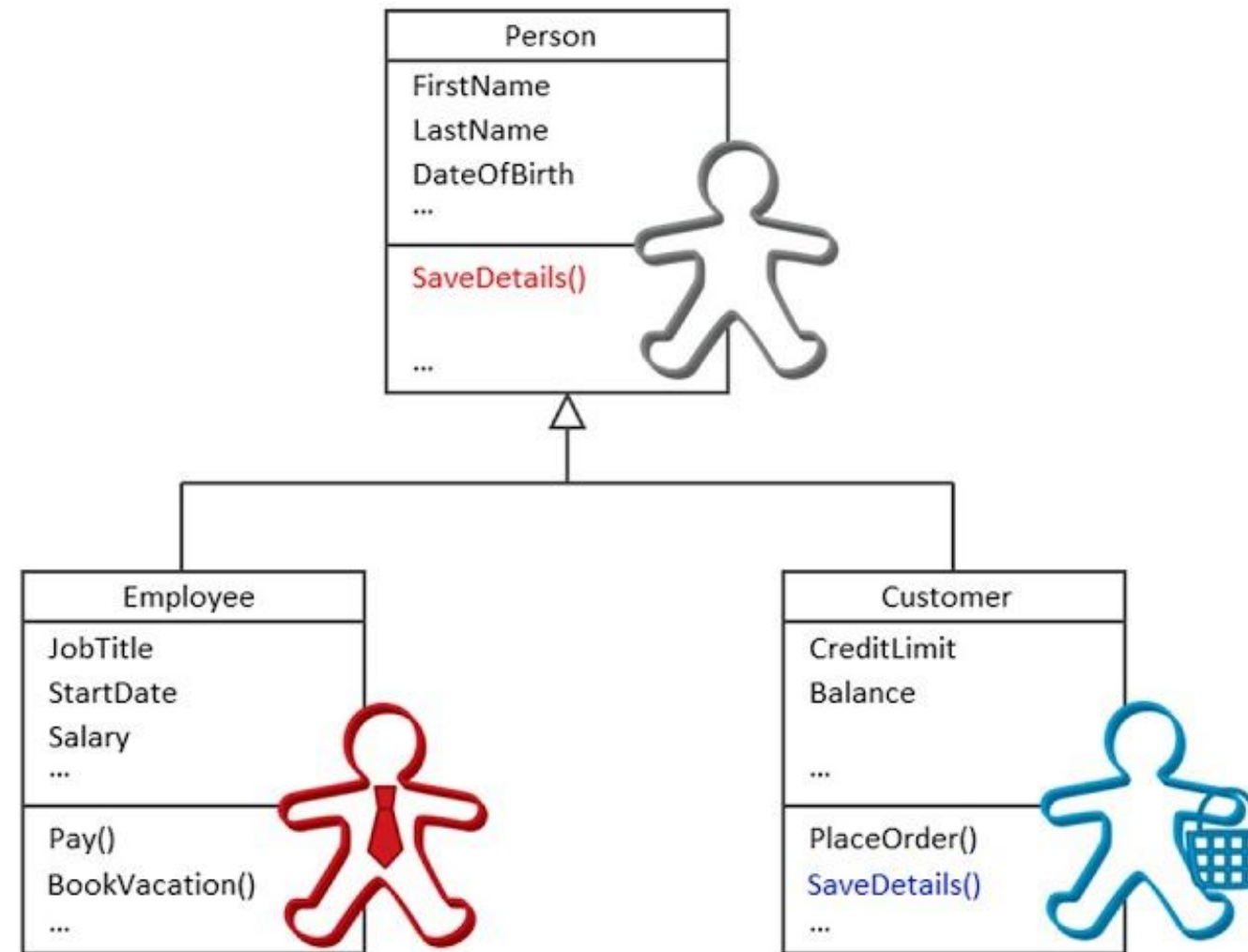
OBJECT ORIENTED PROGRAMMING

Abstraction

Encapsulation

Inheritance

Polymorphism



1. Khai báo lớp với Python

```
2 class ClassName:
3     def __init__(self, parameter1, parameter2):
4         self.attribute1 = parameter1
5         self.attribute2 = parameter2
6
7     def method_name(self):
8         # Behavior of the class
9         print("Executing method!")
```

2. Tính trừu tượng - Abstraction

Là khả năng ẩn chi tiết hiện thực và chỉ hiển thị những gì cần thiết cho người dùng.

Mục đích:

- Giảm sự phức tạp.
- Tập trung vào “cái gì” thay vì “như thế nào”.

3. Tính kế thừa - Inheritance

Cho phép một lớp con kế thừa thuộc tính và phương thức từ một lớp cha.

Mục đích:

- Tái sử dụng code
- Mở rộng hoặc tùy biến hành vi của lớp cha

```
1 class Animal:
2     def eat(self):
3         print("Eating...")
4
5 class Dog(Animal): # Dog kế thừa từ Animal
6     def bark(self):
7         print("Barking...")
8
9 d = Dog()
10 d.eat() # Kế thừa từ Animal
11 d.bark() # Của riêng lớp Dog
```

4. Tính đa hình - Polymorphism

Một hàm hoặc phương thức có thể có nhiều cách hành xử khác nhau tùy thuộc vào đối tượng thực thi.

Bao gồm:

- Đa hình lúc biên dịch (compile-time): nạp chồng hàm (function overloading).
- Đa hình lúc chạy (run-time): ghi đè hàm (function overriding).

```
1 class Animal:
2     def speak(self):
3         print("Animal sound")
4
5 class Cat(Animal):
6     def speak(self):
7         print("Meow")
8
9 class Dog(Animal):
10    def speak(self):
11        print("Woof")
12
13 animals = [Cat(), Dog(), Animal()]
14 for a in animals:
15     a.speak() # Gọi đúng theo từng loại đối tượng
```

5. Tính đóng gói - Encapsulation

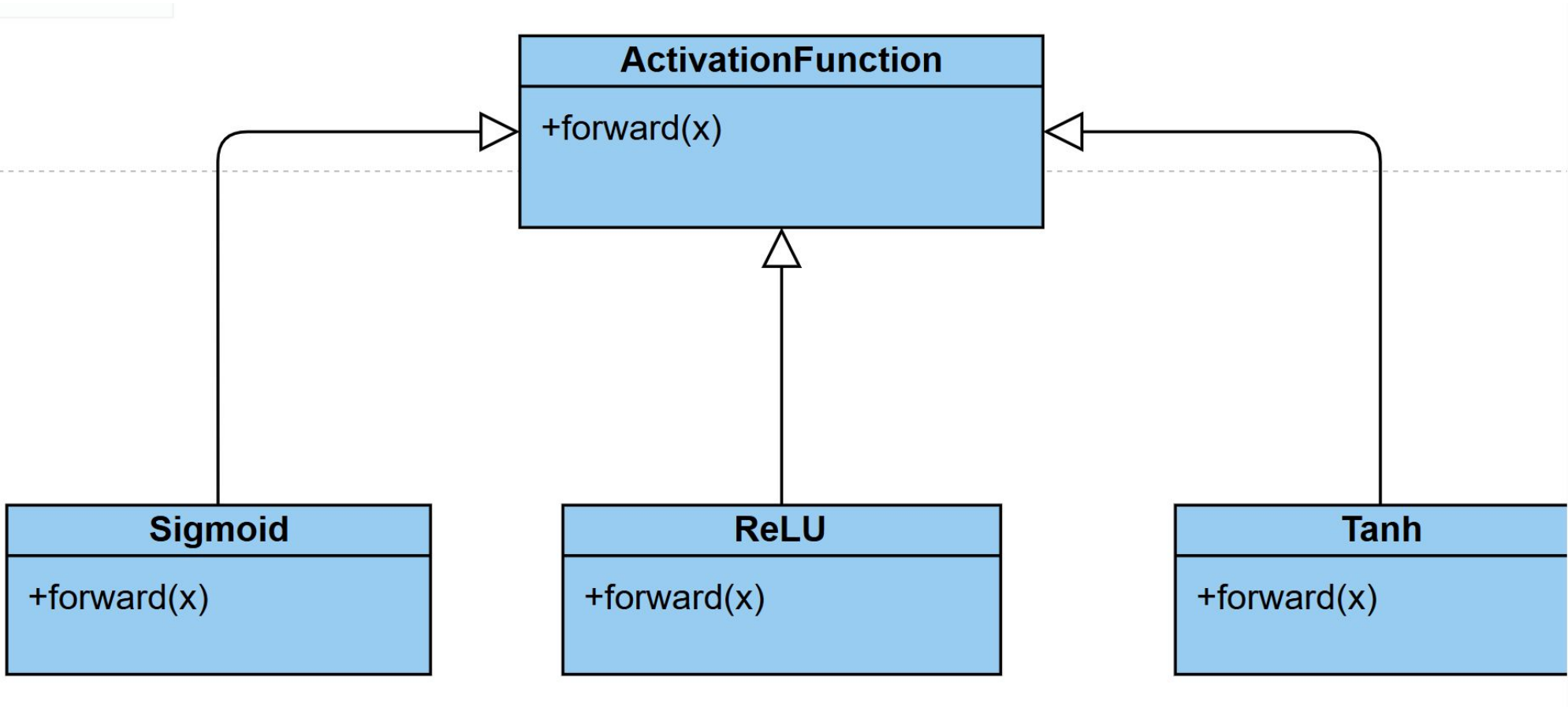
Là cơ chế che giấu thông tin bên trong đối tượng, chỉ cho phép truy cập thông qua các phương thức được định nghĩa.

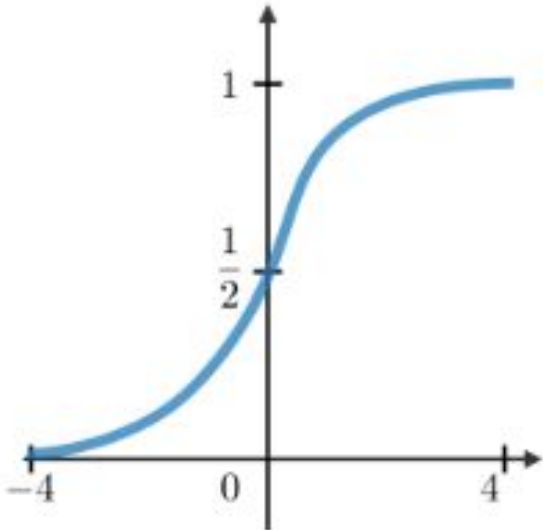
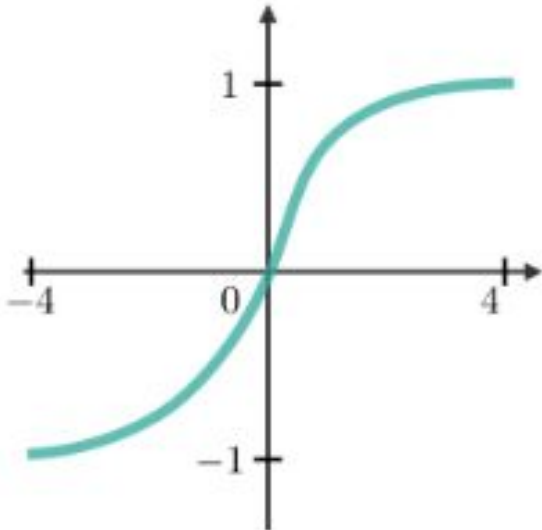
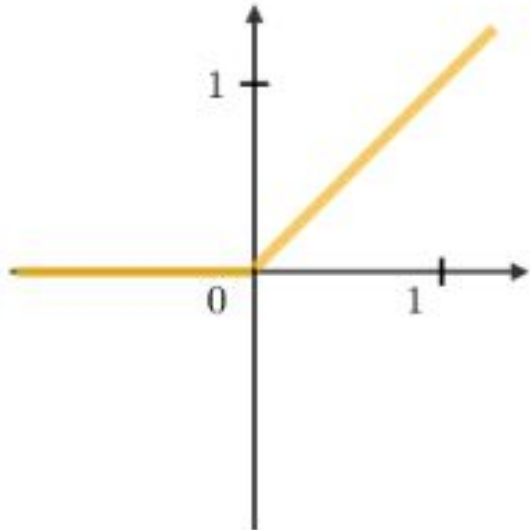
Mục đích:

- Bảo vệ dữ liệu khỏi bị truy cập hoặc sửa đổi trực tiếp từ bên ngoài.
- Giúp quản lý mã dễ dàng, tăng tính bảo trì.

```
1 class Student:
2     def __init__(self, name, age):
3         self.name = name          # public attribute
4         self.__age = age          # private attribute (đóng gói)
5
6     def set_age(self, age):
7         if age > 0:
8             self.__age = age
9
10    def get_age(self):
11        return self.__age
12
13 s = Student("Alice", 20)
14 print(s.name)                    # Truy cập được
15 # print(s.__age)                 # Lỗi: không truy cập được
16 print(s.get_age())               # Truy cập thông qua method
```

Hoàn thành hiện thực cho các hàm kích hoạt Sigmoid, Tanh và ReLU



Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

V. Tensor

1. Tensor

Tensor là một cấu trúc dữ liệu dạng ma trận đa chiều, được sử dụng phổ biến trong các mô hình AI/ML/DL

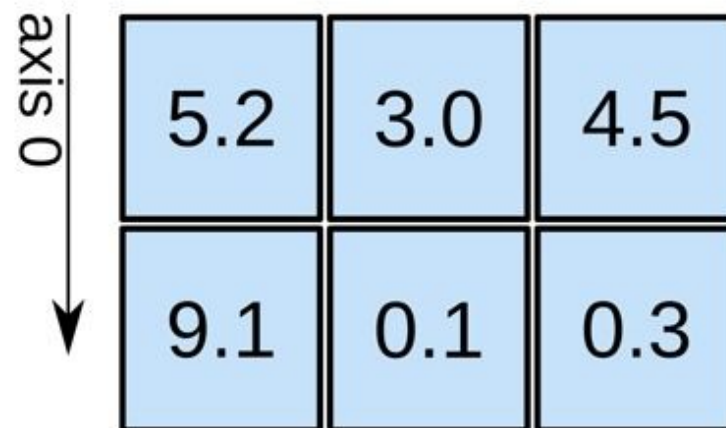
1D array



axis 0 →

shape: (4,)

2D array

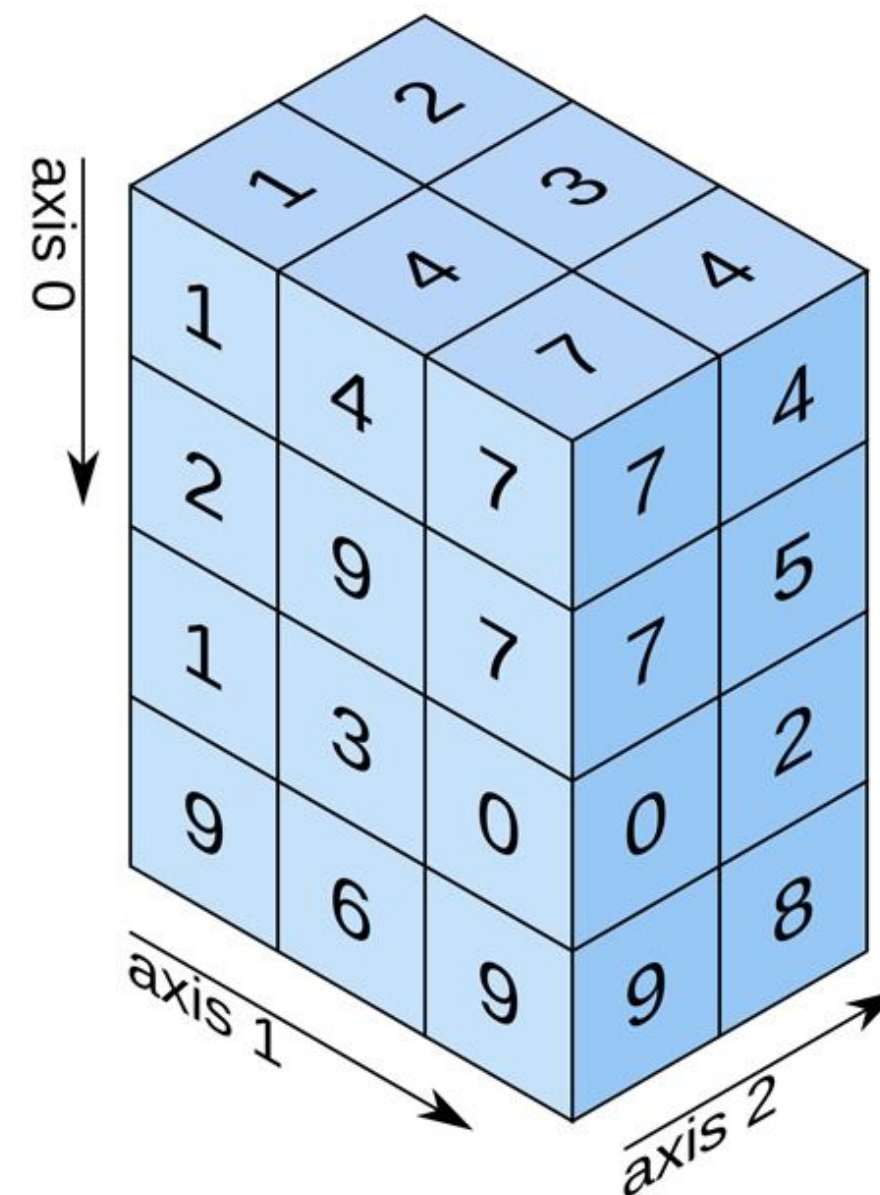


axis 0 ↓

axis 1 →

shape: (2, 3)

3D array



shape: (4, 3, 2)

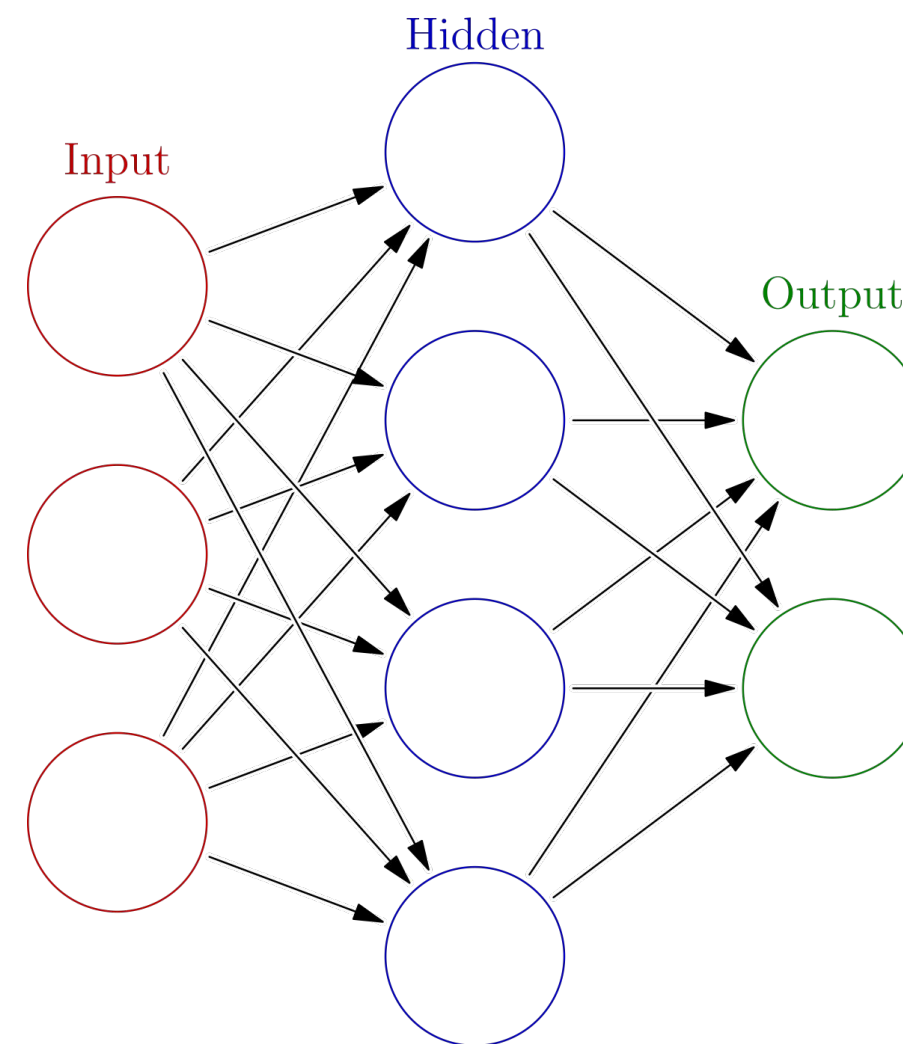
1. Tensor

Tensor là một cấu trúc dữ liệu dạng ma trận đa chiều, được sử dụng phổ biến trong cái mô hình AI/ML/DL

- Ảnh, video
- Chữ, câu, đoạn văn
- Âm thanh
- Bảng, chuỗi...



$\begin{bmatrix} 1, & 2, & 3 \\ 4, & 5, & 6 \\ 7, & 8, & 9 \end{bmatrix}$

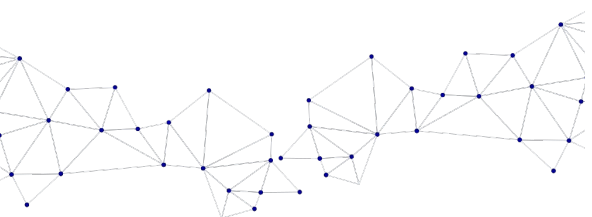
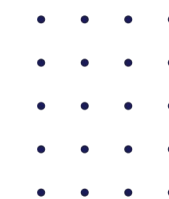


Dữ liệu thô

Tensor

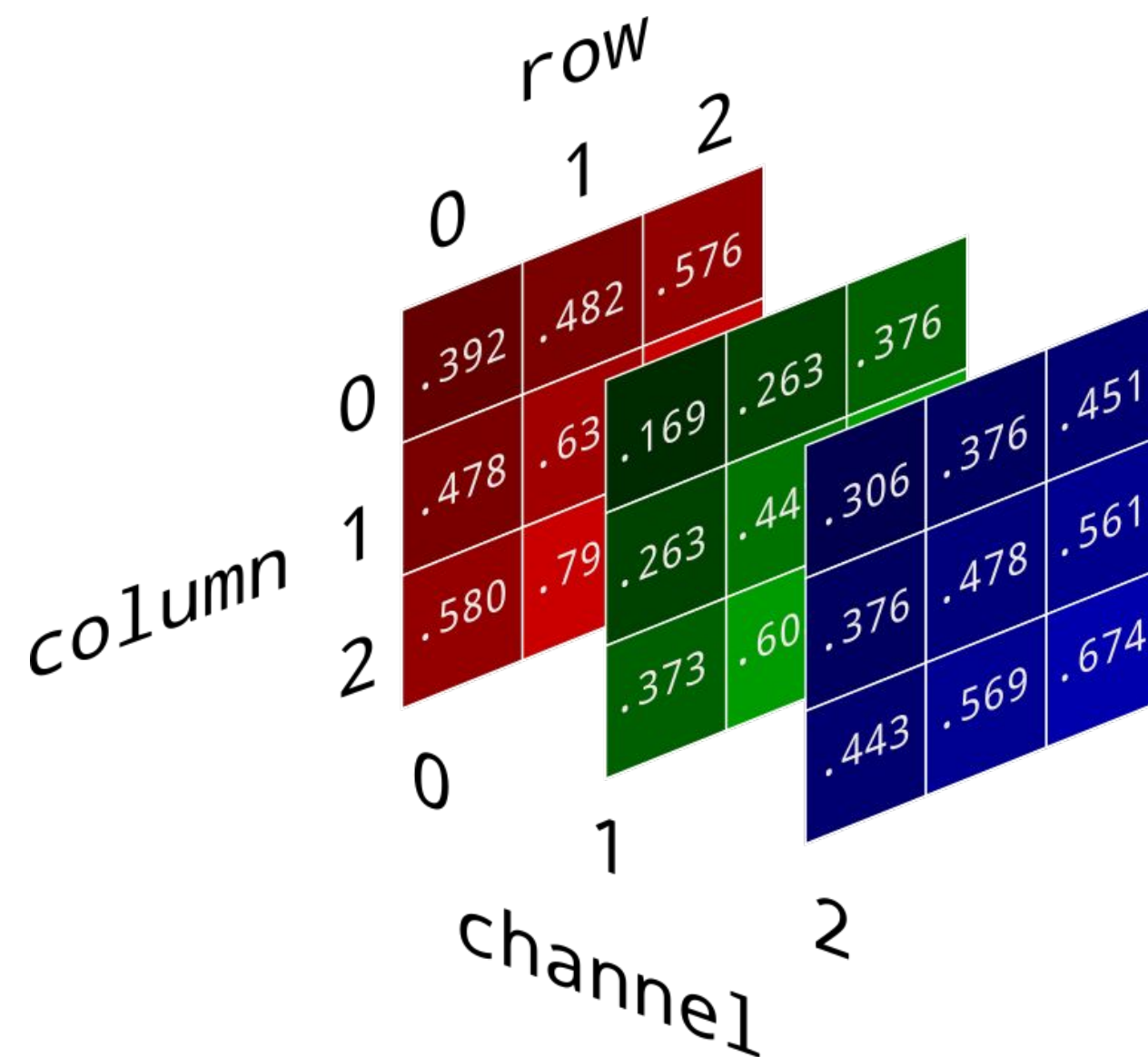
Mô hình

Kết quả



1. Tensor

Tensor là một cấu trúc dữ liệu dạng ma trận đa chiều, được sử dụng phổ biến trong cái mô hình AI/ML/DL



1. Tensor

Tensor là một cấu trúc dữ liệu dạng ma trận đa chiều, được sử dụng phổ biến trong các mô hình AI/ML/DL



2. Tensor Operators

$$\begin{bmatrix} 4 & 8 \\ 3 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 4+1 & 8+0 \\ 3+5 & 7+2 \end{bmatrix}$$

Phép cộng

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix} \times \begin{bmatrix} G \\ H \end{bmatrix} = \begin{bmatrix} A \times G + B \times H \\ C \times G + D \times H \\ E \times G + F \times H \end{bmatrix}$$

Dot Product

$$a \circ b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}_{(n \times 1)} \circ \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}_{(n \times 1)} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ a_3 b_3 \\ a_4 b_4 \\ a_5 b_5 \end{bmatrix}_{(n \times 1)}$$

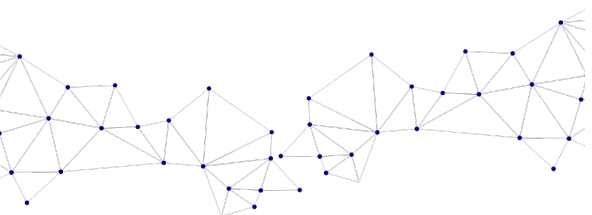
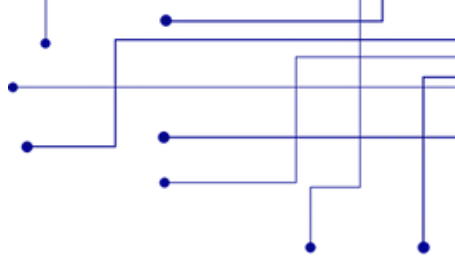
Element wise Product

$$a \otimes b = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_0 b_0 & a_0 b_1 & a_0 b_2 & a_0 b_3 \\ a_1 b_0 & a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_0 & a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_0 & a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix} \dots$$

Outer Product

2. Tensor Operators

- Tensor slicing
- Transpose
- Invert
- ...



THANK YOU

CONTACT US

-  403.1 H6, BKHCM Campus 2
-  mliandiotlab@gmail.com
-  ml-iotlab.com
-  facebook.com/hcmut.ml.iot.lab
-  youtube.com/@mliotlab