



Lập trình hướng đối tượngg (OOP)



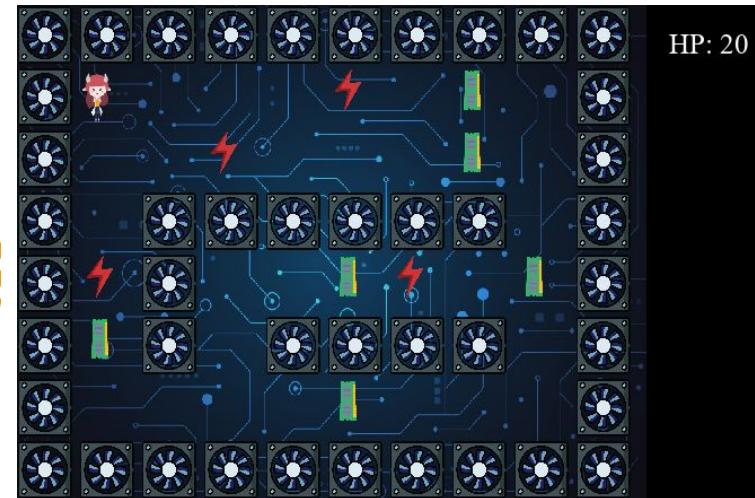
Đây là một sản phẩm trí tuệ có bản quyền thuộc về STEAM for Vietnam. Các bên chỉ được sử dụng với mục đích học tập, nghiên cứu, và không được quyền sử dụng sản phẩm này nhằm mục đích thu lợi nhuận dù trực tiếp hay gián tiếp.



Buổi học hôm nay

Trầu muốn viết một **game** để
giúp các bạn nhỏ học những khái
niệm khoa học máy tính

https://bit.ly/S4V_CS101_L6_Warm_Up





Viết chương trình

Làm thế nào để viết một game như thế nhỉ?

Hàm
Function

def

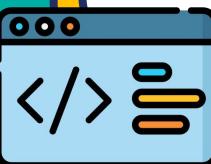
Files



Cấu trúc
dữ liệu



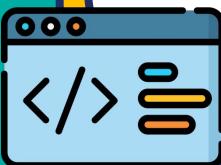
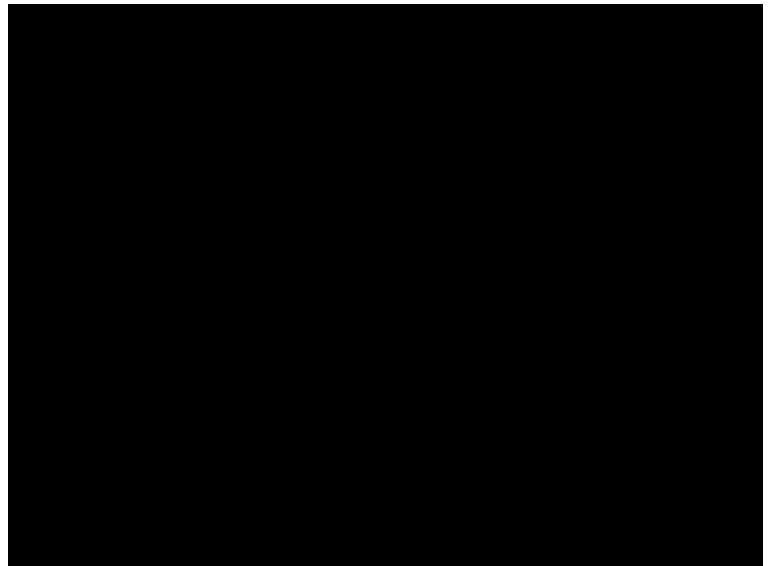
Hãy tách lớn ra nhỏ!





Phân tích chương trình

Đầu tiên, chúng ta hãy xem game của
mình hoạt động ra sao nhé!

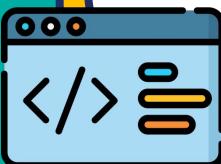
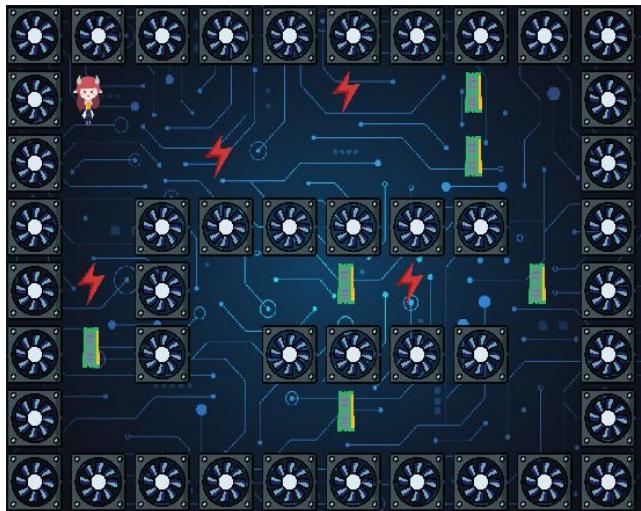




Phân tích chương trình

Game của chúng ta có gì?

- Bạn Trầu ở trong một căn phòng và có thể di chuyển trái phải lên xuống. Xung quanh đó có rất nhiều thanh RAM và tia điện.
- Khi chạm vào thanh RAM, Trầu sẽ học được một kiến thức mới và thêm 10 “máu”
- Khi chạm vào tia điện, Trầu sẽ bị giật và mất 30 “máu”

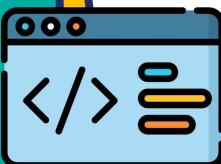
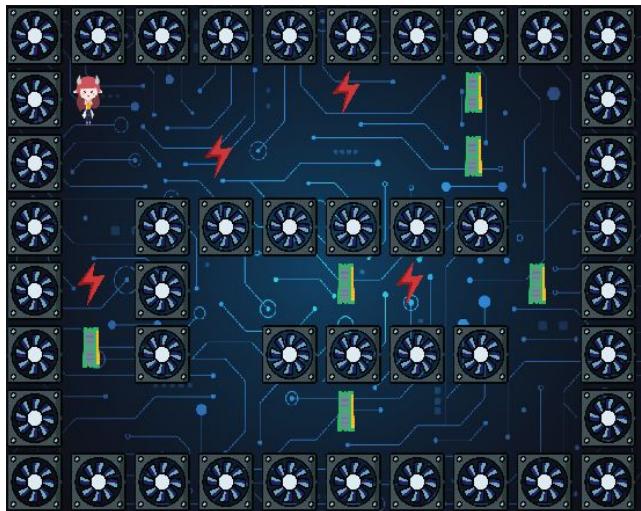




Phân tích chương trình

Chú ý những vật có trong game

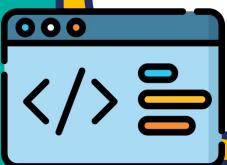
- 1 • Bạn Trầu ở trong một căn phòng và có thể di chuyển. Em phải lén xung quanh. Xung quanh đó có rất nhiều **thanh RAM** và **tia điện**.
- 2 • Khi chạm vào thanh RAM, Trầu sẽ học được một kiến thức mới và thêm 10 “máu”
- 3 • Khi chạm vào tia điện, Trầu sẽ bị giật và mất 30 “máu”





Phân tích chương trình

Chúng ta thường dùng các **nhân vật, đồ vật** cùng với **đặc điểm** và các **hành động** của chúng để mô tả chương trình





Phân tích chương trình

Trầu sẽ có đặc điểm và hành động gì trong game?

Đặc điểm

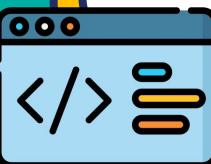
Vị trí: Ở góc trên bên trái
Máu: 20

Hành động

Di chuyển
Nhặt đồ vật

Động từ

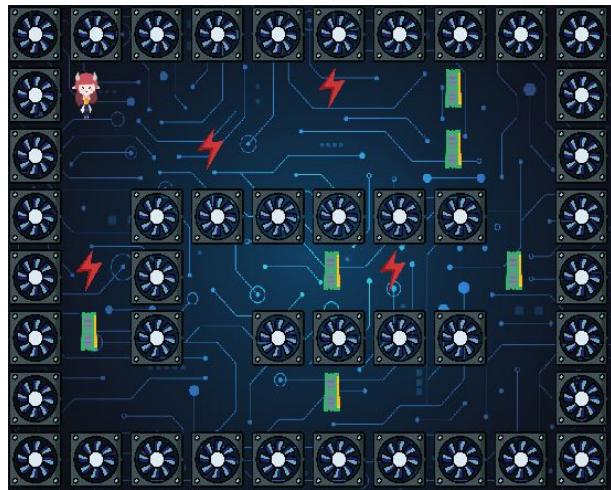
thường là



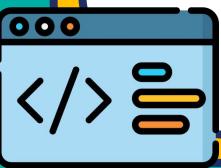


Phân tích chương trình

Từ một chương trình phức tạp, chúng ta đã chia nhỏ thành các vật thể đơn giản



Hô biến!!!





LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

(Object-Oriented Programming)

Cách chia nhỏ chương trình thành các **đối tượng** (**object**) có các **đặc điểm** và **hành động** được gọi là **lập trình hướng đối tượng** (**Object-Oriented Programming**)



Quiz #1

Trong các nhóm từ sau, nhóm nào mô tả đặc điểm và nhóm nào mô tả hành động của Doraemon?

A

- Bàn tay tròn
- Không có tai

B

- Ăn bánh rán
- Ngủ
- Sử dụng bảo bối

- a. B mô tả đặc điểm, A mô tả hành động
- b. A mô tả đặc điểm, B mô tả hành động
- c. Cả A và B mô tả đặc điểm
- d. Cả A và B mô tả hành động



Quiz #1

Trong các nhóm từ sau, nhóm nào mô tả đặc điểm và nhóm nào mô tả hành động của Doraemon?

Đặc điểm

- Bàn tay tròn
- Không có tai

Hành động

- Ăn bánh rán
- Ngủ
- Sử dụng bảo bối

- a. B mô tả đặc điểm, A mô tả hành động
- b. A mô tả đặc điểm, B mô tả hành động**
- c. Cả A và B mô tả đặc điểm
- d. Cả A và B mô tả hành động



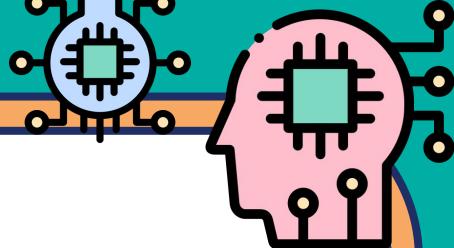
CỘT MỐC 1: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Cách chia nhỏ chương trình thành các **đối tượng (object)** có các **đặc điểm** và **hành động** được gọi là **lập trình hướng đối tượng (Object-Oriented Programming)**

01

ĐỐI TƯỢNG & LỚP (Object & Class)





ĐỐI TƯỢNG & LỚP

ĐỐI TƯỢNG (Object)

Một đối tượng bao gồm các **đặc điểm** và **hành động**

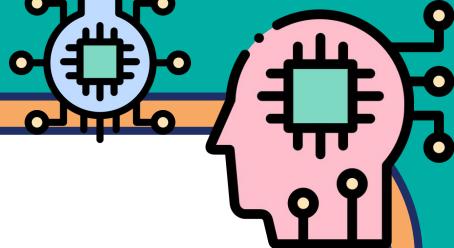


Đặc điểm

- Vị trí ở góc trái
- Máu: 20

Hành động

- Di chuyển
- Nhặt đồ vật



ĐỐI TƯỢNG & LỚP

ĐỐI TƯỢNG (Object)

Trong OOP, **đặc điểm** của đối tượng được gọi là **thuộc tính (attribute)**

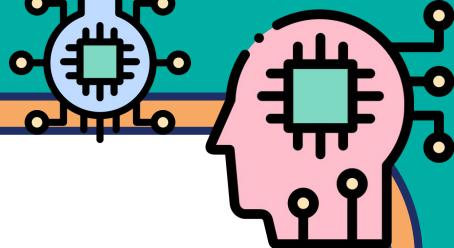


Thuộc tính

- Vị trí ở góc trái
- Máu: 20

Hành động

- Di chuyển
- Nhặt đồ vật



ĐỐI TƯỢNG & LỚP

ĐỐI TƯỢNG (Object)

Trong OOP, **hành động** của đối tượng được gọi là **phương thức (method)**

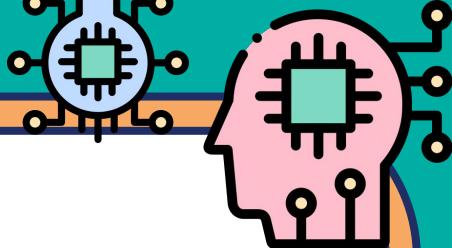


Thuộc tính

- Vị trí ở góc trái
- Máu: 20

Phương thức

- Di chuyển
- Nhặt đồ vật



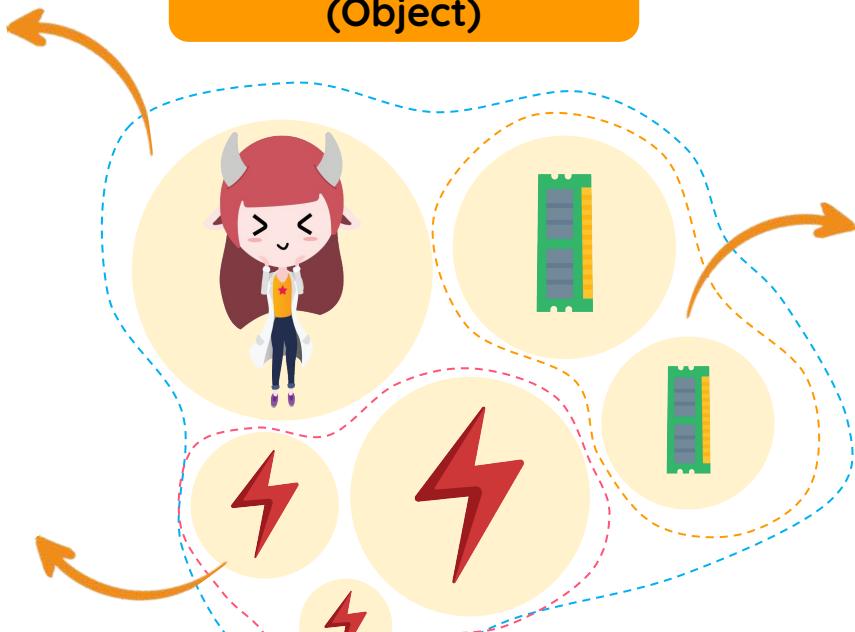
ĐỐI TƯỢNG & LỚP

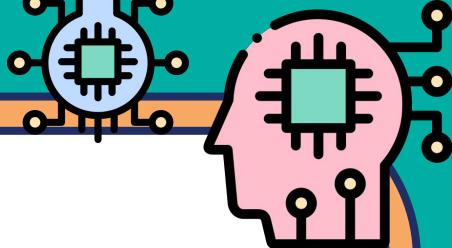
ĐỐI TƯỢNG (Object)

Đây là 6 đối tượng
(object)

Có 2 đối tượng
thanh RAM

Có 3 đối tượng
tia điện

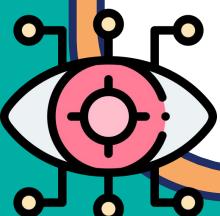
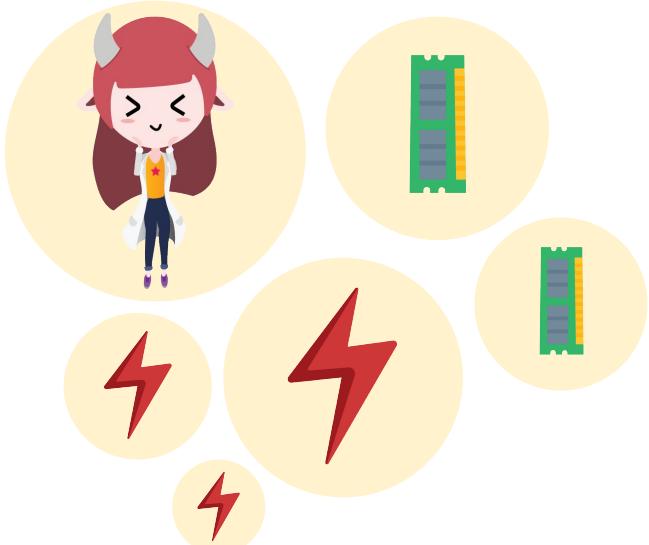


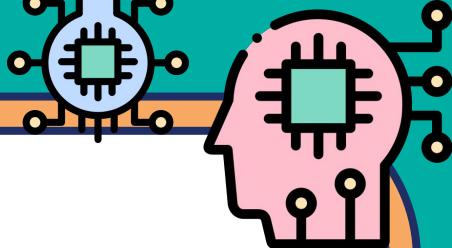


ĐỐI TƯỢNG & LỚP

ĐỐI TƯỢNG (Object)

Vậy làm sao để tạo được các đối tượng này?



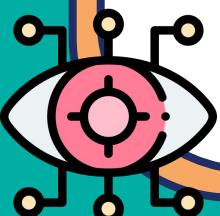
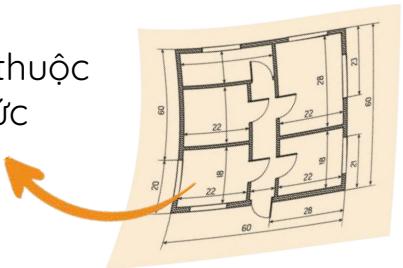


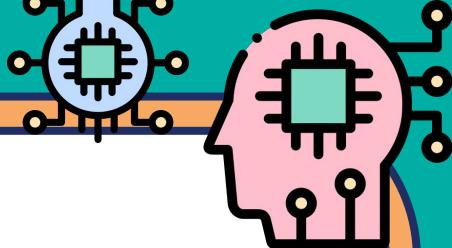
ĐỐI TƯỢNG & LỚP

LỚP
(Class)

Lớp (class) là một **bản thiết kế** để tạo ra
các **đối tượng (object)**

Lớp sẽ **có sẵn** những thuộc
tính và phương thức

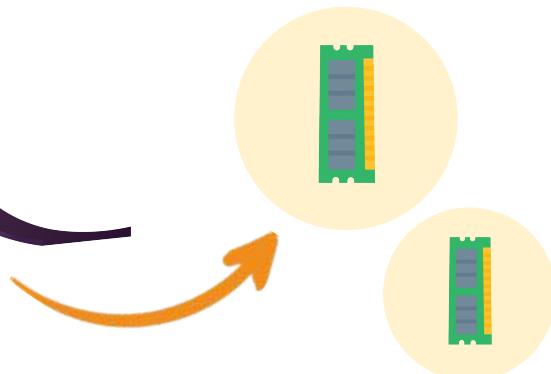
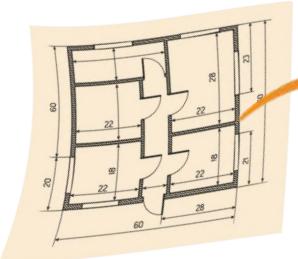


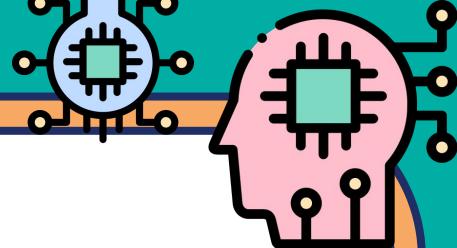


ĐỐI TƯỢNG & LỚP

LỚP (Class)

Dựa vào lớp (class), Python sẽ tạo ra nhiều đối tượng thuộc lớp đó.

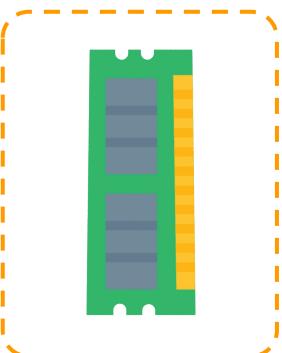




ĐỐI TƯỢNG & LỚP

LỚP (Class)

Lớp RAM có thể được mô tả như sau



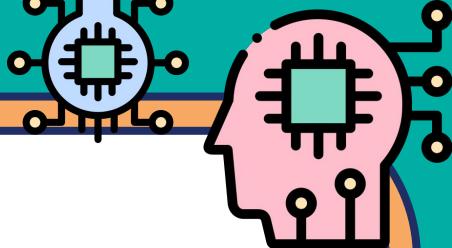
Thuộc tính

- Hình ảnh
- Tọa độ X, Y
- Từ vựng
- Đơn vị máu tăng

Đơn vị máu người chơi nhận được khi nhặt lên

Phương thức

- Xuất hiện trên bảng thông báo

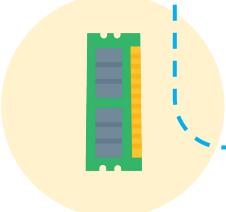


ĐỐI TƯỢNG & LỚP

Các đối tượng được tạo ra từ một lớp là các **vật riêng biệt**.
Thuộc tính của chúng có thể có các **giá trị khác nhau**.

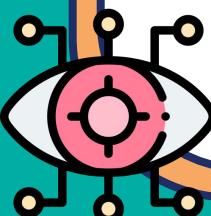
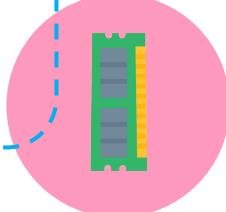
Thuộc tính

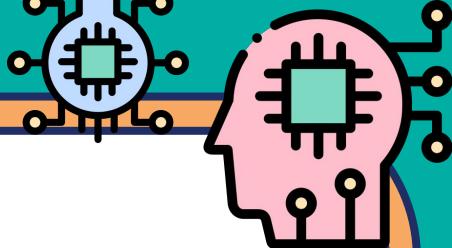
- Hình ảnh như bên
- Tọa độ X: **100**
- Tọa độ Y: **250**
- Từ vựng: “**stack**”
- Đơn vị máu tăng: **10**



Thuộc tính

- Hình ảnh như bên
- Tọa độ X: **200**
- Tọa độ Y: **150**
- Từ mới: “**queue**”
- Đơn vị máu tăng: **10**





ĐỐI TƯỢNG & LỚP

Các đối tượng được tạo ra từ một lớp có các phương thức của lớp đó

Thuộc tính

- Hình ảnh như bên
- Tọa độ X: **100**
- Tọa độ Y: **250**
- Từ vựng: “**stack**”
- Đơn vị máu tăng: **10**



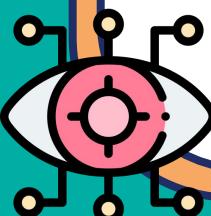
Thuộc tính

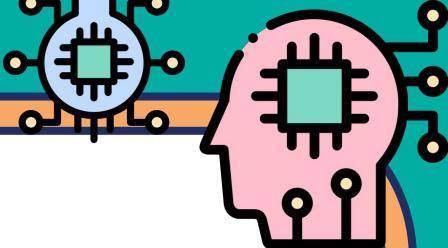
- Hình ảnh như bên
- Tọa độ X: **200**
- Tọa độ Y: **150**
- Từ mới: “**queue**”
- Đơn vị máu tăng: **10**



Phương thức

- Xuất hiện trên bảng thông báo





ĐỐI TƯỢNG & LỚP

Giả sử mình có một lớp **Student** như sau

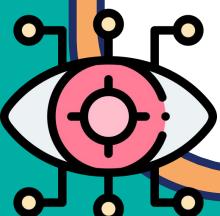
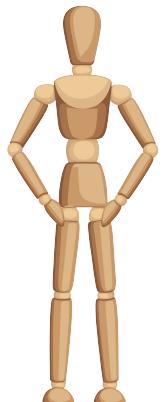
Thuộc tính

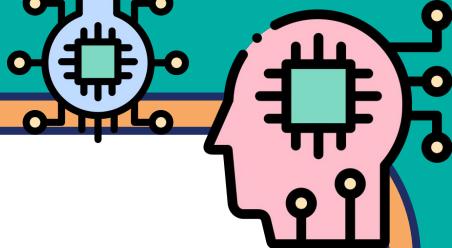
- Tên
- Tuổi
- Trường học
- Địa chỉ

Phương thức

- Giới thiệu bản thân
- Học bài
- Lập trình

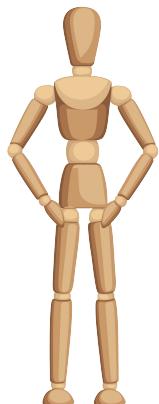
Đây là 1 lớp (class)





ĐỐI TƯỢNG & LỚP

Lớp và đối tượng khác gì nhau nhỉ?



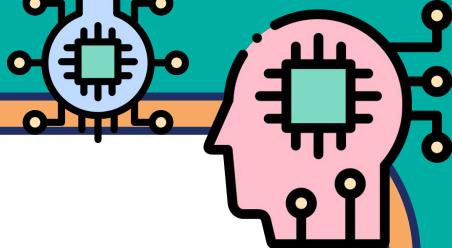
Thuộc tính

- Tên
- Tuổi
- Trường học
- Địa chỉ

Thuộc tính

- Tên: **Trẩu**
- Tuổi: **12**
- Trường học: **ABC**
- Địa chỉ: **Việt Nam**

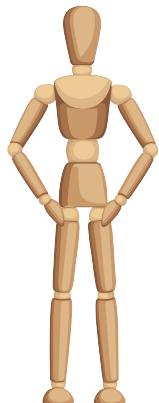




ĐỐI TƯỢNG & LỚP

Lớp và đối tượng khác gì nhau nhỉ?

Thuộc tính của lớp
không có giá trị



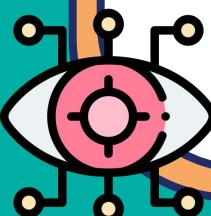
Thuộc tính

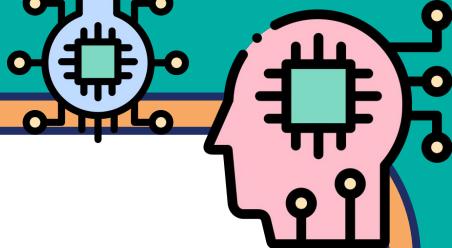
- Tên
- Tuổi
- Trường học
- Địa chỉ

Thuộc tính của đối tượng
có giá trị

Thuộc tính

- Tên: **Trẩu**
- Tuổi: **12**
- Trường học: **ABC**
- Địa chỉ: **Việt Nam**





ĐỐI TƯỢNG & LỚP

Các đối tượng được tạo ra từ cùng 1 lớp sẽ có các
phương thức giống nhau

Thuộc tính

- Tên: **Trầu**
- Tuổi: **12**
- Trường học: **ABC**
- Địa chỉ: **Việt Nam**



Phương thức

- Giới thiệu bản thân
- Học bài
- Lập trình

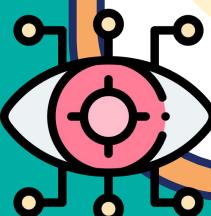
Thuộc tính

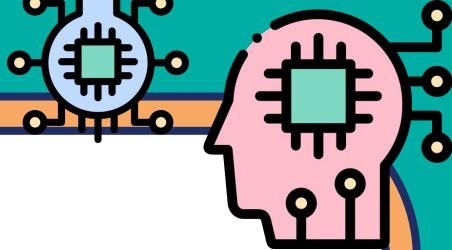
- Tên: **William**
- Tuổi: **14**
- Trường học: **XYZ**
- Địa chỉ: **Việt Nam**



Phương thức

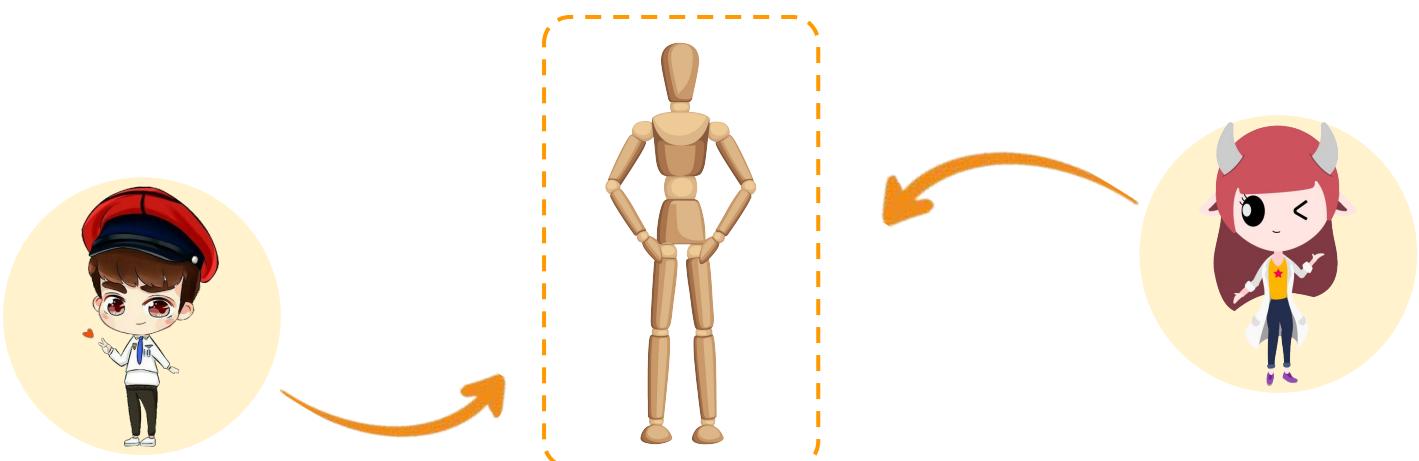
- Giới thiệu bản thân
- Học bài
- Lập trình

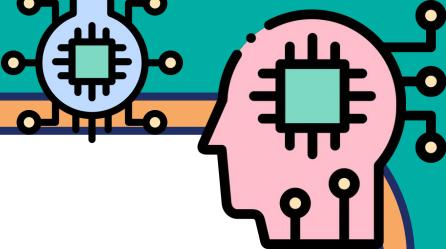




ĐỐI TƯỢNG & LỚP

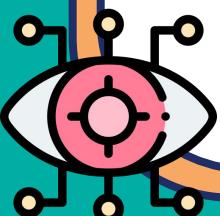
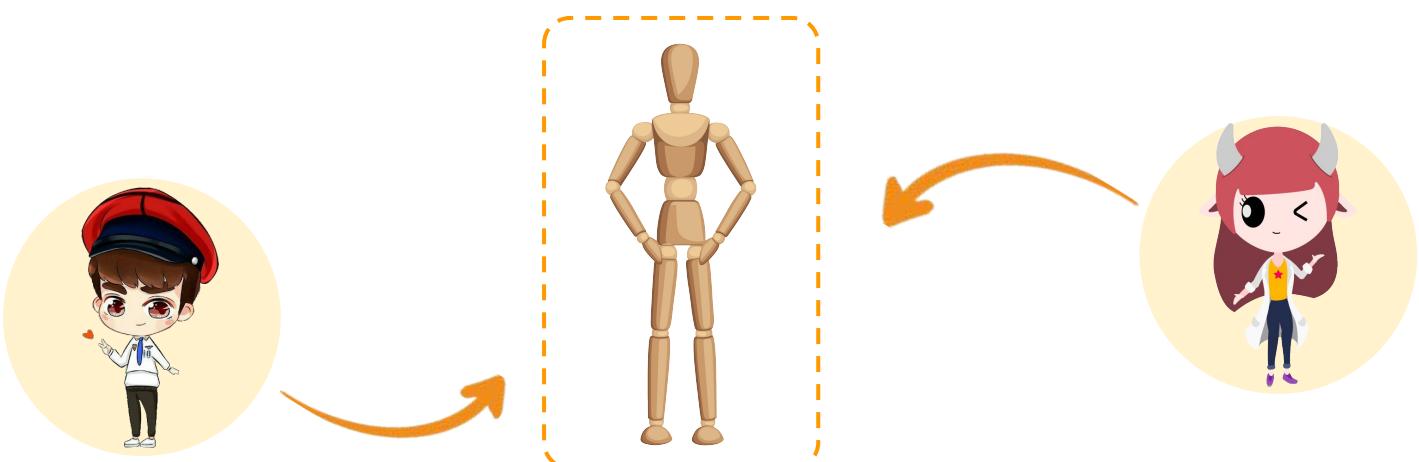
Vậy là Trầu và William là hai **đối tượng (object)** được tạo ra từ một **lớp (class)**





ĐỐI TƯỢNG & LỚP

Chúng ta còn nói Trầu và William là hai **đối tượng** thuộc **lớp** học sinh



Quiz #2

Câu nào sau đây chính xác?

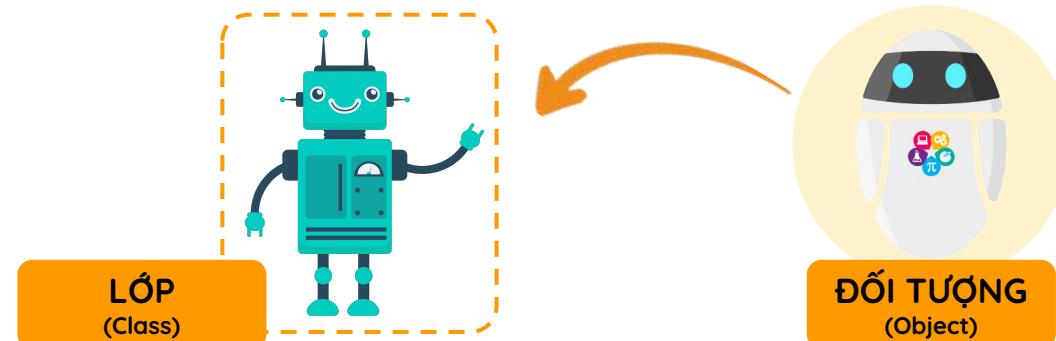
- a. Bạn TRE là một **lớp** và Robot là một **đối tượng** của lớp đó
- b. Robot là một **lớp** và bạn TRE là một **đối tượng** của lớp đó
- c. Trau và William là **lớp** của **đối tượng** Student
- d. Student là một **đối tượng** thuộc **lớp** Trầu



Quiz #2

Câu nào sau đây chính xác?

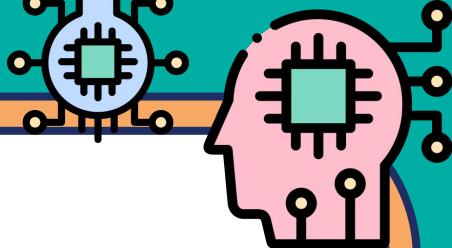
- a. Bạn TRE là một **lớp** và Robot là một **đối tượng** của lớp đó
- b. Robot là một lớp và bạn TRE là một đối tượng của lớp đó**
- c. Trau và William là **lớp** của **đối tượng** Student
- d. Student là một **đối tượng** thuộc **lớp** Trầu



02

OOP CƠ BẢN TRONG PYTHON



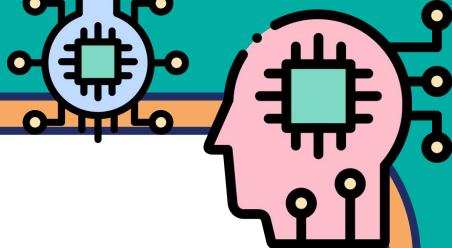


Tạo một lớp

Tên của lớp

hàm `__init__` là hàm
được chạy khi một
đối tượng được tạo ra

```
class Student:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```



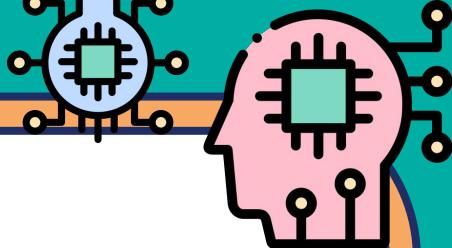
Tạo một lớp

```
class Student:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

Đây là các **thuộc tính**

self chính là đối tượng mà mình đang tạo ra

Ngay sau khi tạo đối tượng, chúng ta có thể gán **giá trị** vào các thuộc tính



Khởi tạo đối tượng

Vậy là chúng ta đã viết xong một lớp.
Hãy tạo một đối tượng từ lớp này nhé!

Biến sẽ chứa
đối tượng

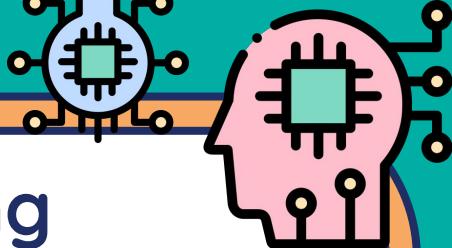


trau = Student("Trau", 12)

Tên lớp

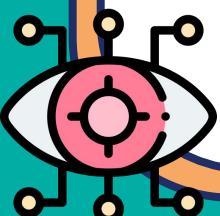
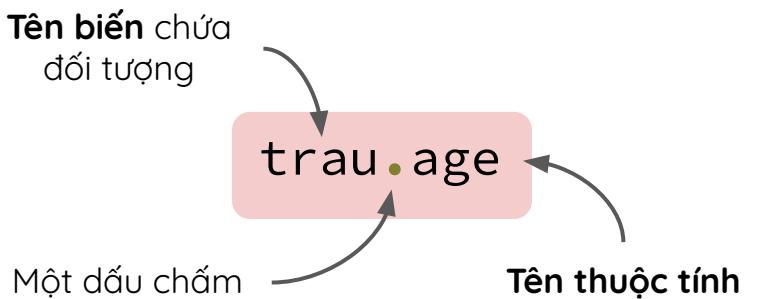
Các giá trị sẽ gán vào
thuộc tính của đối tượng

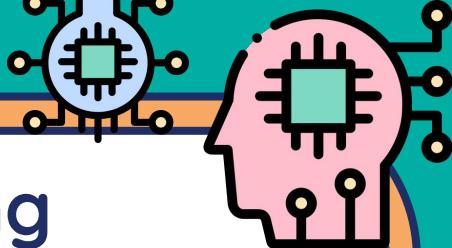
```
def __init__(self, name, age):  
    self.name = name  
    self.age = age
```



Dùng thuộc tính của đối tượng

Thuộc tính của một đối tượng trông giống như **biến**.





Dùng thuộc tính của đối tượng

Thuộc tính của một đối tượng trông giống như biến.

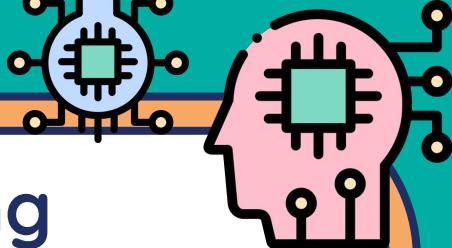
Gán giá trị cho
thuộc tính

```
trau = Student("Trau", 12)
```

```
trau.age = trau.age + 1
```

```
print(trau.age)
```

Lấy giá trị của
thuộc tính



Dùng thuộc tính của đối tượng

Giá trị thuộc tính của các đối tượng độc lập với nhau!

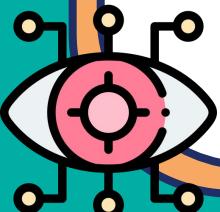


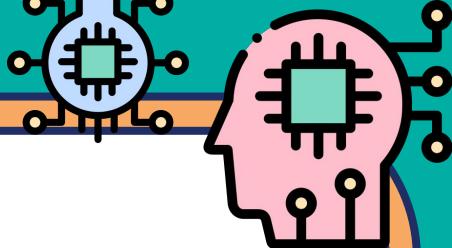
Thay đổi **age** của
trau sẽ không ảnh
hưởng đến **william**

```
trau = Student("Trau", 12)  
william = Student("William", 14)
```

```
trau.age = trau.age + 1
```

```
print(trau.age)  
print(william.age)
```



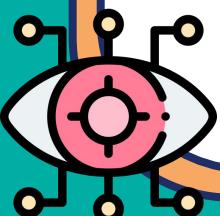


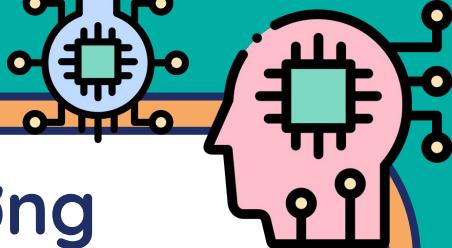
Tạo một phương thức

Bây giờ, mình sẽ viết một **phương thức** cho lớp **Student** nhé!

Đây là phương thức
say_hello()

```
class Student:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def say_hello(self):  
        print("Hello, I'm " + self.name + "!")
```





Dùng phương thức của đối tượng

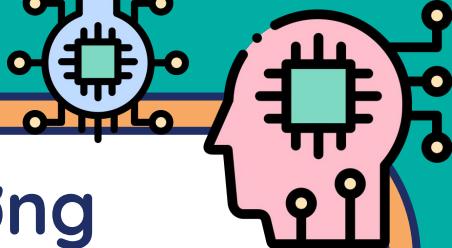
Phương thức trông giống **hàm**.

Tên biến chưa
đối tượng

trau.say_hello()

Một dấu chấm

Tên phương thức



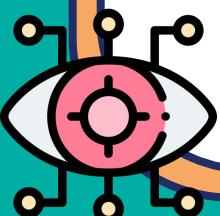
Dùng phương thức của đối tượng

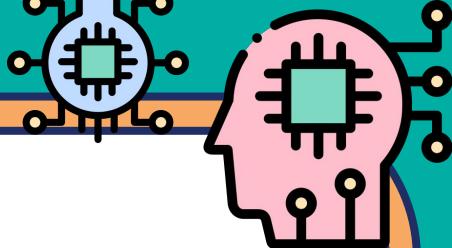
Chạy thử đoạn code này nào!

```
trau = Student("Trau", 12)  
  
trau.say_hello()
```

Đây là
kết quả nè!

Hello, I'm Trau!





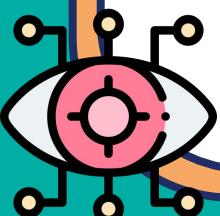
Tham số self

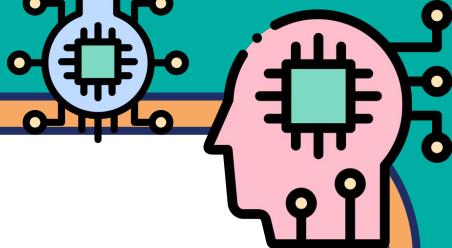
Điều đặc biệt là **tham số đầu tiên** của các phương thức luôn là **self**.

```
class Student:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def say_hello(self):  
        print("Hello, I'm " + self.name + "!")
```

Hàm `say_hello()` không nhận vào bất cứ input nào nhưng vẫn phải có `self`

Tại sao vậy nhỉ?





Tham số self

Vì **self** sẽ cho biết đối tượng nào đang gọi hàm đó.

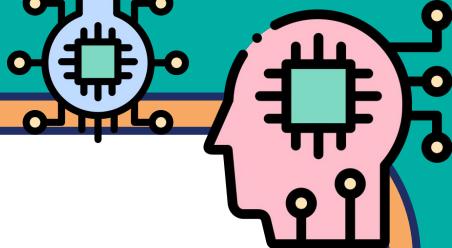


`trau.say_hello()`

Đối tượng sẽ được đưa vào **self**

```
class Student:  
    def say_hello(self):  
        print("Hello, I'm " + self.name + "!")
```

Do đó, đây sẽ là giá trị thuộc tính **name** của **trau**



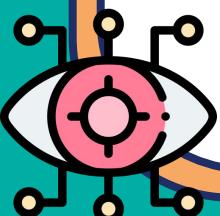
Tham số self

Nếu chúng ta thiếu **self** sẽ bị báo lỗi

```
class Student:  
    def say_hello():  
        print("Hello, I'm " + self.name + "!")
```



TypeError: say_hello() takes 0 positional arguments but 1 was given



Quiz #3

Cách nào để tạo đối tượng cho lớp Student sau đây

```
class Student:  
    def __init__(self, name):  
        self.name = name
```

- a. student1 = Student("Trau")
- b. student1 = Student.__init__("Trau")
- c. student1 = Student.create("Trau")
- d. student1 = Student()



student1

Quiz #3

Cách nào để tạo đối tượng cho lớp Student sau đây

```
class Student:  
    def __init__(self, name):  
        self.name = name
```

- a. student1 = Student("Trau")
- b. student1 = Student.__init__("Trau")
- c. student1 = Student.create("Trau")
- d. student1 = Student()



CỘT MỐC 2: OOP CƠ BẢN VỚI PYTHON

```
class Student:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def say_hello(self):  
        print("Hello, I'm " + self.name + "!")  
  
trau = Student("Trau", 12)  
trau.say_hello()  
  
william = Student("William", 14)  
william.say_hello()
```



LẬP TRÌNH GAME

Trong lúc giải lao, chúng ta hãy nhanh tay download các files để chuẩn bị viết game nhé!

PHIÊN BẢN
CƠ BẢN



bit.ly/S4V_CS101_Lesson6_Game

PHIÊN BẢN
NÂNG CAO



NGHỈ GIẢI LAO

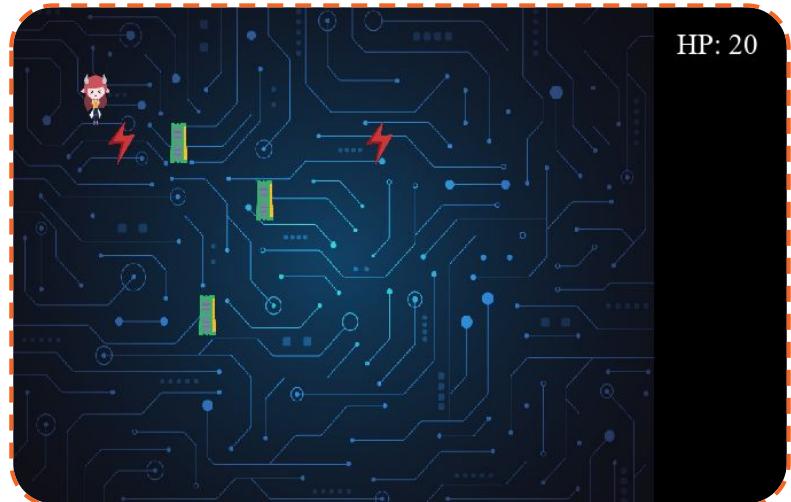


Học sinh quay lại sau 10 phút nghỉ giải lao

PHIÊN BẢN
CƠ BẢN

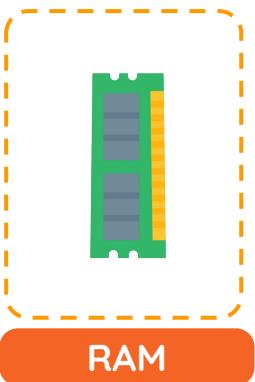
LẬP TRÌNH GAME

Đầu tiên, chúng ta làm phiên bản
đơn giản của game này nhé!



LẬP TRÌNH GAME

Trong game, chúng ta sẽ tập trung vào **3 lớp**





LẬP TRÌNH GAME

Đối với lớp **Player**, chúng ta sẽ có những thuộc tính và phương thức nào



Thuộc tính

- Hình ảnh
- Tọa độ X, Y
- Máu
- Tốc độ di chuyển

Phương thức

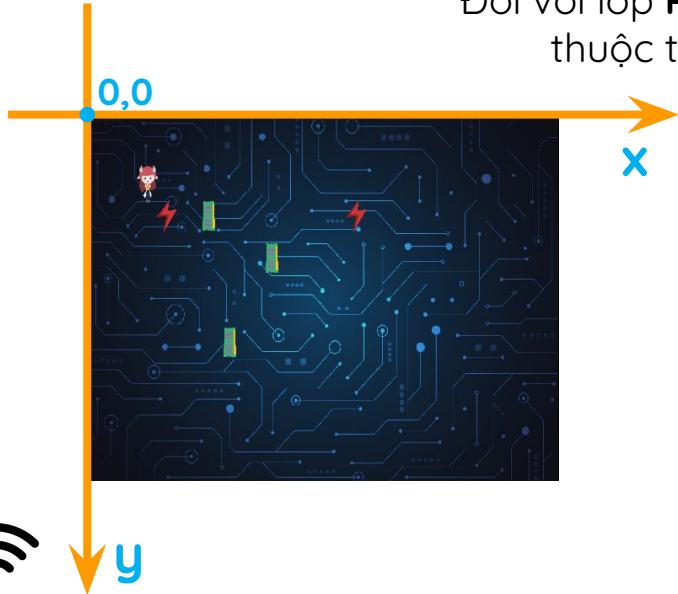
- Đi lên
- Đi xuống
- Đi sang phải
- Đi sang trái





LẬP TRÌNH GAME

Đối với lớp **Player**, chúng ta sẽ có những thuộc tính và phương thức nào



Thuộc tính

- Hình ảnh
- Tọa độ X, Y
- Máu
- Tốc độ di chuyển

X, Y là tọa độ trong màn hình game

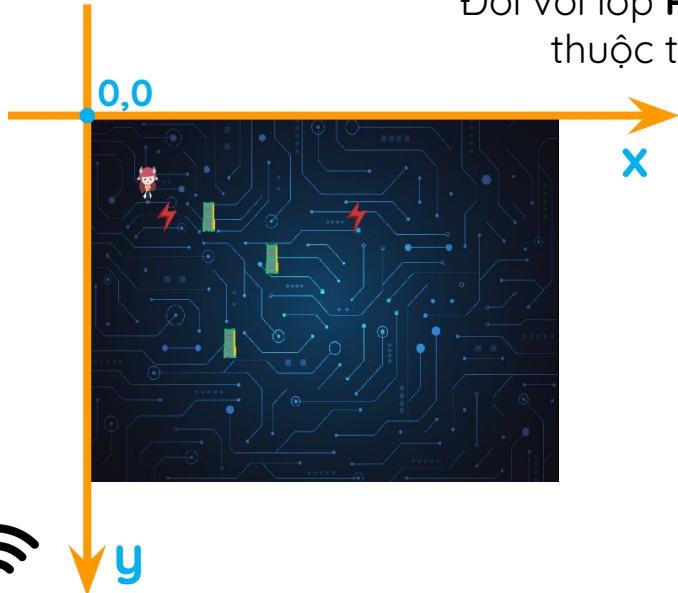
Phương thức

- Đi lên
- Đi xuống
- Đi sang phải
- Đi sang trái



LẬP TRÌNH GAME

Đối với lớp **Player**, chúng ta sẽ có những thuộc tính và phương thức nào



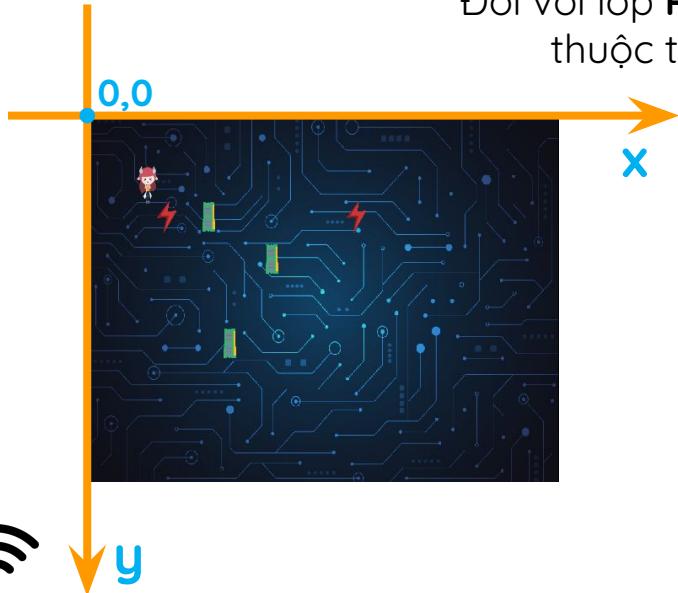
Ở đây là
thuộc tính

```
class Player:  
    def __init__(self):  
        self.sprite_path = "sprites/trau.png"  
        self.x = 60  
        self.y = 60  
        self.speed = 5  
        self.hp = 20
```



LẬP TRÌNH GAME

Đối với lớp **Player**, chúng ta sẽ có những thuộc tính và phương thức nào



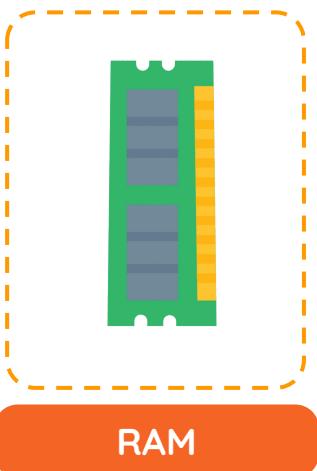
Đây là các phương thức

```
class Player:  
    def __init__(self):  
        self.sprite_path = "sprites/trau.png"  
        self.x = 60  
        self.y = 60  
        self.speed = 5  
        self.hp = 20  
  
    def moveRight(self):  
        self.x = self.x + self.speed  
    def moveLeft(self):  
        self.x = self.x - self.speed  
    def moveUp(self):  
        self.y = self.y - self.speed  
    def moveDown(self):  
        self.y = self.y + self.speed
```



LẬP TRÌNH GAME

Đối với lớp **RAM**, chúng ta sẽ có những thuộc tính và phương thức nào?



```
class RAM:  
    def __init__(self, x, y, word):  
        self.sprite_path = "sprites/ram.png"  
        self.x = x  
        self.y = y  
        self.hp_change = 10  
        self.word = word  
  
    def get_info(self):  
        return "New word: " + self.word
```

Ở đây là
thuộc tính

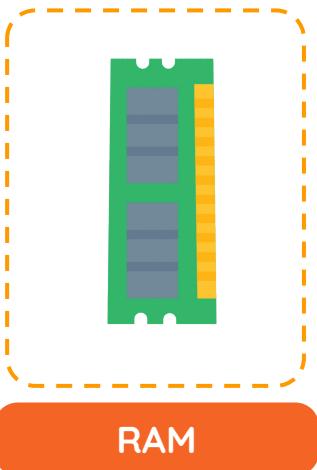
Và đây là
phương thức





LẬP TRÌNH GAME

Đối với lớp **RAM**, chúng ta sẽ có những thuộc tính và phương thức nào



Thuộc tính

- Hình ảnh
- Tọa độ X, Y
- Từ vựng
- Đơn vị máu thay đổi

Đơn vị máu người chơi **THAY ĐỔI** khi nhặt lên



Phương thức

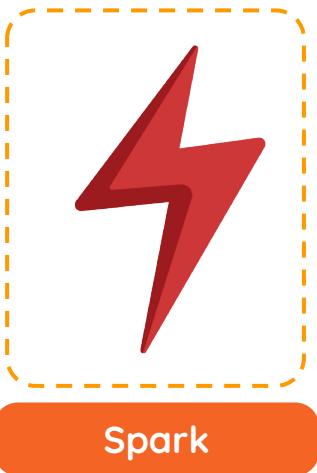
- Xuất hiện trên bảng thông báo





LẬP TRÌNH GAME

Đối với lớp **Spark**, chúng ta sẽ có những thuộc tính và phương thức nào?



```
class Spark:  
    def __init__(self, x, y):  
        self.sprite_path = "sprites/spark.png"  
        self.x = x  
        self.y = y  
        self.hp_change = -30
```

Ở đây là
thuộc tính

```
def get_info(self):  
    return "Spark!"
```

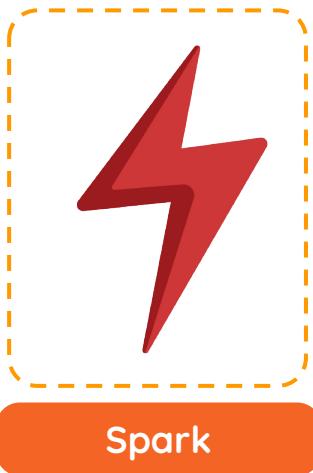
Và đây là
phương thức





LẬP TRÌNH GAME

Đối với lớp **Spark**, chúng ta sẽ có những thuộc tính và phương thức nào



Thuộc tính

- Hình ảnh
- Tọa độ X, Y
- Đơn vị máu thay đổi

Đơn vị máu người chơi **THAY ĐỔI** đi khi nhặt lên



Phương thức

- Xuất hiện trên bảng thông báo

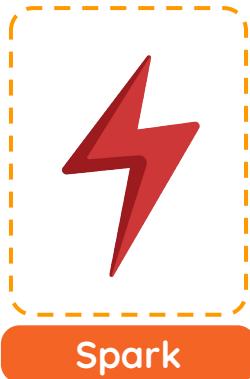


CỘT MỐC 3: ĐỐI TƯỢNG TRONG GAME

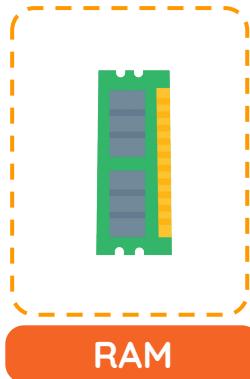
Hiểu được 3 **lớp** sau trong game



Player



Spark



RAM

03

TÍNH LIÊN KẾT (Association)





TÍNH LIÊN KẾT

Chúng ta có **3** đối tượng thuộc lớp Student



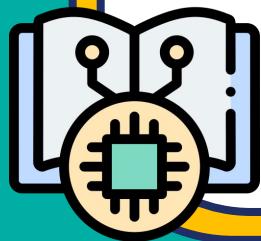
name: Ngạn



name: Hà Lan



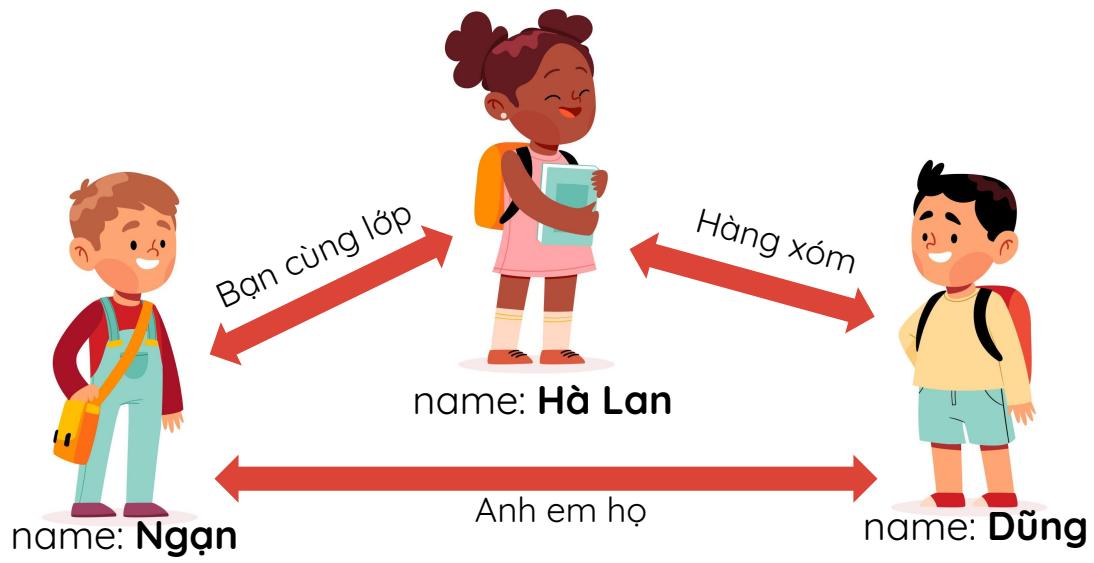
name: Dũng





TÍNH LIÊN KẾT

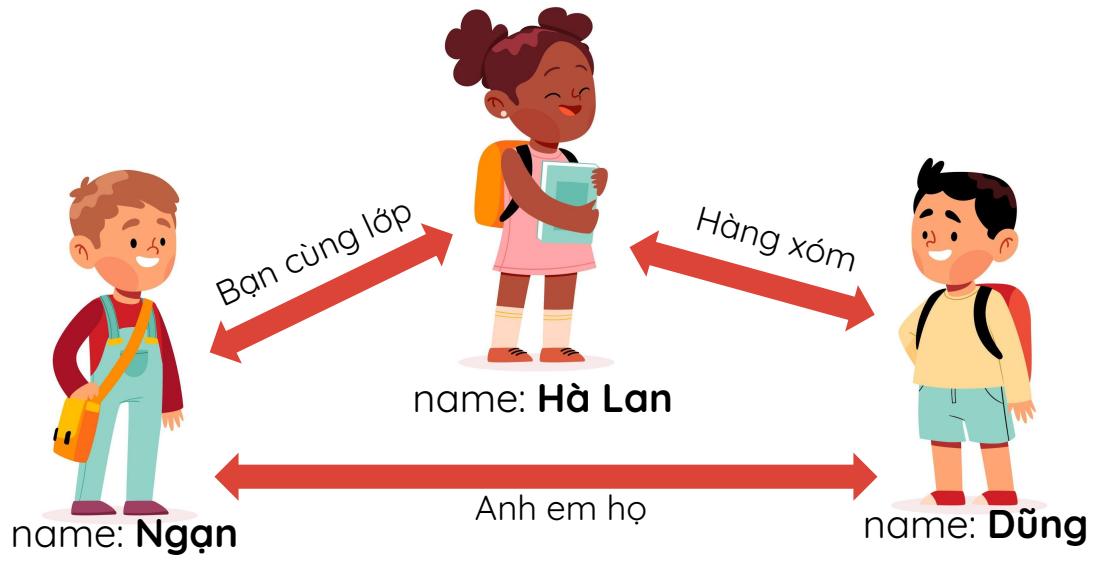
3 đối tượng này đều có mối **liên kết** với nhau





TÍNH LIÊN KẾT

Vậy là một đối tượng có thể **liên kết** với các đối tượng khác





TÍNH LIÊN KẾT

Vậy là một đối tượng có thể **liên kết** với các đối tượng khác



```
class Student:  
    def __init__(self, name):  
        self.name = name  
        self.friends = []
```

Một mảng để lưu danh sách các đối tượng khác



TÍNH LIÊN KẾT

Vậy là một đối tượng có thể **liên kết** với các đối tượng khác



```
class Student:  
    def __init__(self, name):  
        self.name = name  
        self.friends = []  
  
    def add_friend(self, friend):  
        self.friends.append(friend)
```

Một phương thức để
thêm đối tượng khác vào
danh sách



TÍNH LIÊN KẾT

Vậy là một đối tượng có thể **liên kết** với các đối tượng khác



```
class Student:  
    def __init__(self, name):  
        self.name = name  
        self.friends = []  
  
    def add_friend(self, friend):  
        self.friends.append(friend)  
  
    def list_friends(self):  
        if len(self.friends) == 0:  
            print("I have no friends yet :(")  
        else:  
            print("I have "+str(len(self.friends))+" friends")  
            for friend in self.friends:  
                print(friend.name)
```

Phương thức để in ra những người bạn trong danh sách



TÍNH LIÊN KẾT

Vậy là một đối tượng có thể **liên kết** với các đối tượng khác



```
class Student:  
    def __init__(self, name):  
        self.name = name  
        self.friends = []  
  
    def add_friend(self, friend):  
        self.friends.append(friend)  
  
    def list_friends(self):  
        if len(self.friends) == 0:  
            print("I have no friends yet :(")  
        else:  
            print("I have "+str(len(self.friends))+" friends")  
            for friend in self.friends:  
                print(friend.name)
```

Dùng thuộc tính
của đối tượng khác



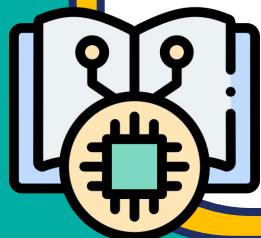
TÍNH LIÊN KẾT

Vậy là một đối tượng có thể **liên kết** với các đối tượng khác



```
lan = Student("Ha Lan")
ngan = Student("Ngan")
dung = Student("Dung")
```

```
lan.add_friend(ngan)
lan.add_friend(dung)
lan.list_friends()
```



Quiz #4

Bạn William có bao nhiêu người bạn?

```
minh = Student("Minh")
william = Student("William")
```

```
trau = Student("Trau")
trau.add_friend(minh)
trau.add_friend(william)
```

```
print(trau.friends)
print(william.friends)
```

- a. 2
- b. 1
- c. 0
- d. 3



Quiz #4

Bạn William có bao nhiêu người bạn?

```
minh = Student("Minh")
william = Student("William")
```

```
trau = Student("Trau")
trau.add_friend(minh)
trau.add_friend(william)
```

```
print(trau.friends)
print(william.friends)
```

- a. 2
- b. 1
- c. 0
- d. 3





LẬP TRÌNH GAME

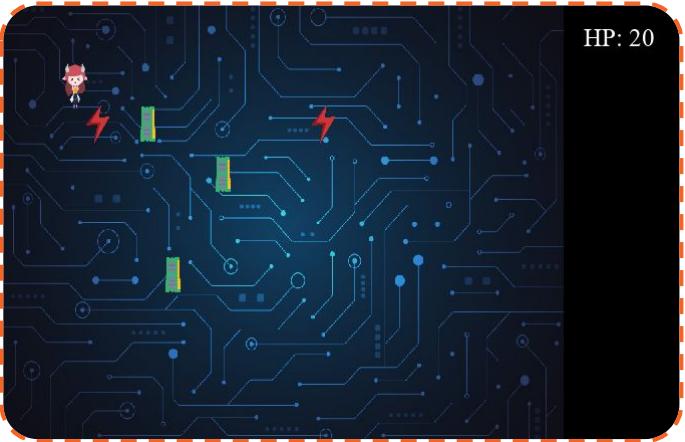
Làm thế nào để **đối tượng Player** lưu các
đối tượng đã nhặt được?





LẬP TRÌNH GAME

Chúng ta viết thêm cho lớp Player một **thuộc tính** để lưu **danh sách các đồ vật** mà người chơi đã nhặt



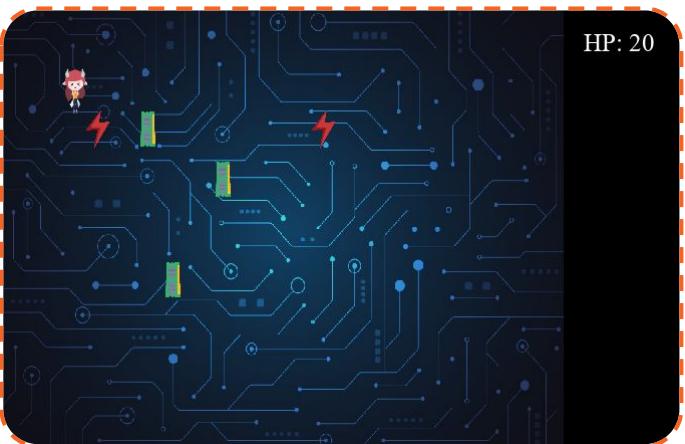
```
class Player:  
    def __init__(self):  
        self.x = 60  
        self.y = 60  
        self.speed = 5  
        self.hp = 20  
        self.items = []
```





LẬP TRÌNH GAME

Sau đó, viết thêm một phương thức để thêm đồ vật vào thuộc tính **items**



```
class Player:  
    def __init__(self):  
        self.x = 60  
        self.y = 60  
        self.speed = 5  
        self.hp = 20  
        self.items = []
```

```
def pick_up_item(self, item):  
    self.items.append(item)
```

Cần một đồ vật được đưa vào phương thức

CỘT MỐC 4: LIÊN KẾT CÁC ĐỐI TƯỢNG

```
class Player:  
    def __init__(self):  
        self.sprite_path = "sprites/Trau_shy.png"  
        self.x = 60  
        self.y = 60  
        self.speed = 5  
        self.hp = 20  
        self.items = []  
  
    def pick_up_item(self, item):  
        self.items.append(item)  
        self.hp = self.hp + item.hp_change
```

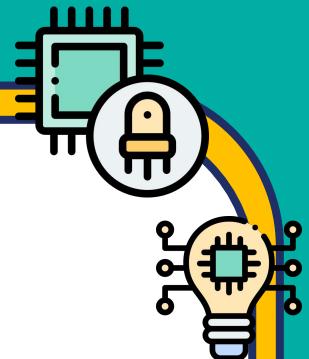
bit.ly/S4V_Lesson6_Basic

04(Nâng cao)

TÍNH KẾ THỪA

(Inheritance)





Các lớp có thể có điểm chung

Các lớp sau có thuộc tính và phương thức gần như giống nhau



Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Leo cây



Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Bay

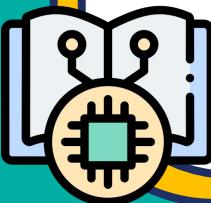


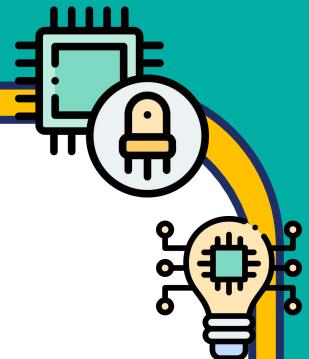
Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Bơi





Các lớp có thể có điểm chung

Mỗi lớp chỉ khác ở một phương thức



Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Leo cây



Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Bay

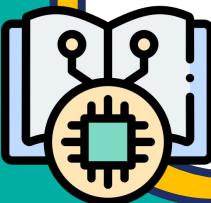


Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Bơi





Các lớp có thể có điểm chung

Chúng ta có thể viết một lớp chứa các **thuộc tính và phương thức chung** đó

Động
vật

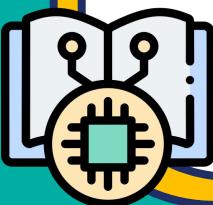
Thuộc tính

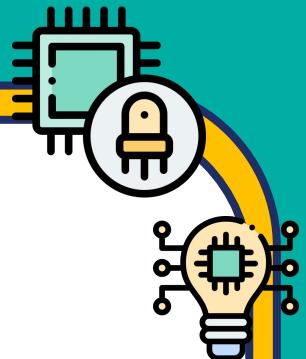
- Tên
- Thức ăn ưa thích

Phương thức

- Ăn

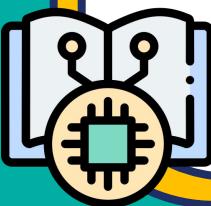
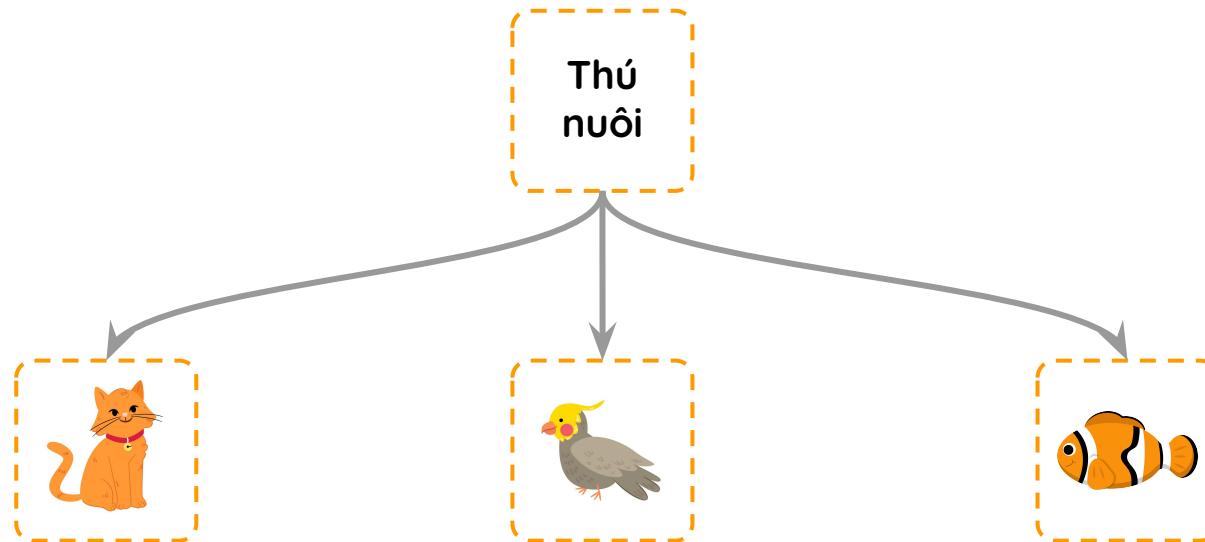
Không có phương thức
khác nhau

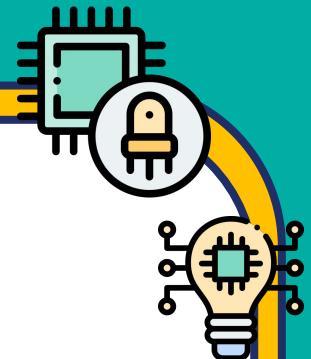




Các lớp có thể có điểm chung

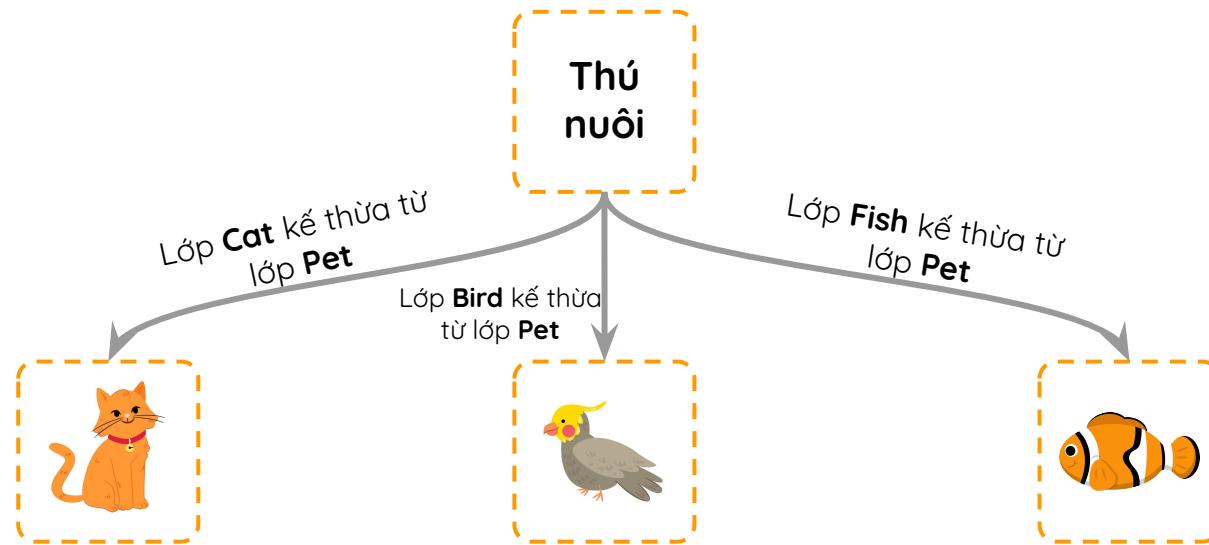
Sau đó, các lớp **Cat**, **Bird**, **Fish** sẽ **thừa hưởng**, dùng lại các thuộc tính và phương thức của lớp **Pet**

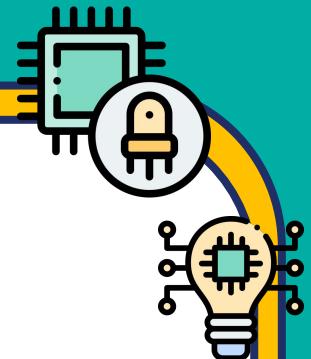




Các lớp có thể có điểm chung

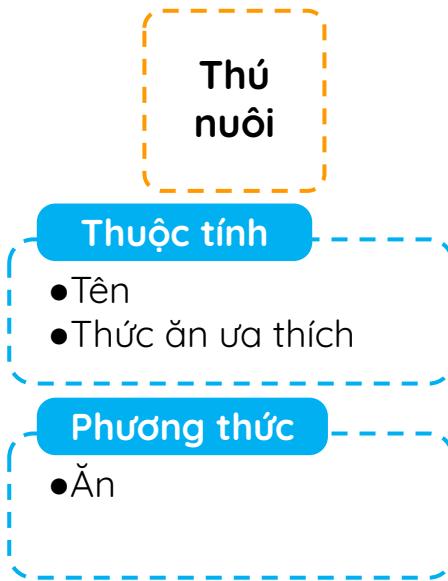
Trong OOP, chúng ta gọi đây là
tính kế thừa (inheritance)





Các lớp có thể có điểm chung

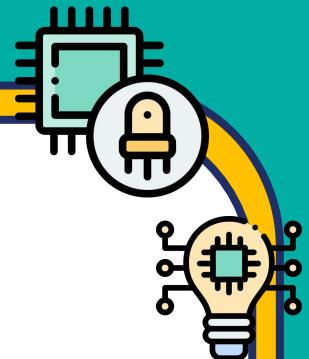
Viết một lớp **Pet** gồm
2 thuộc tính và 1 phương thức



```
class Pet:  
    def __init__(self, name, favorite_food):  
        self.name = name  
        self.favorite_food = favorite_food  
  
    def eat(self):  
        print("I'm eating" + self.favorite_food)
```

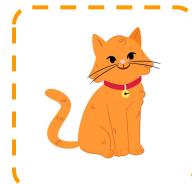
Đây là
phương thức

Hai thuộc tính nè!



Các lớp có thể có điểm chung

Lớp **Cat** chỉ cần kế thừa lớp **Pet**
và thêm một phương thức



Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Leo cây

Thêm tên lớp kế thừa
vào trong ngoặc

```
class Cat(Pet):
```

```
    def climb_tree(self):  
        print(self.name, " can climb tree")
```

Chỉ cần thêm một phương thức
climb_tree



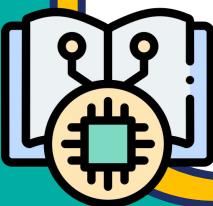
Các lớp có thể có điểm chung

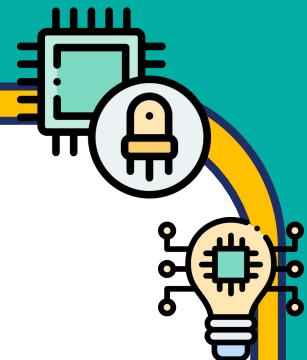
Chạy thử đoạn code dưới đây nào

```
mao_mao = Cat("Mao Mao", "fish")  
  
mao_mao.eat()  
mao_mao.climb_tree()
```

Đây là
kết quả nè!

I'm eating fish
Mao Mao can climb tree





Các lớp có thể có điểm chung

Lớp **Bird** chỉ cần kế thừa lớp **Pet**
và thêm một phương thức



Thuộc tính

- Tên
- Thức ăn ưa thích

Phương thức

- Ăn
- Bay

Thêm tên lớp kế thừa
vào trong ngoặc

```
class Bird(Pet):
```

```
    def fly(self):
```

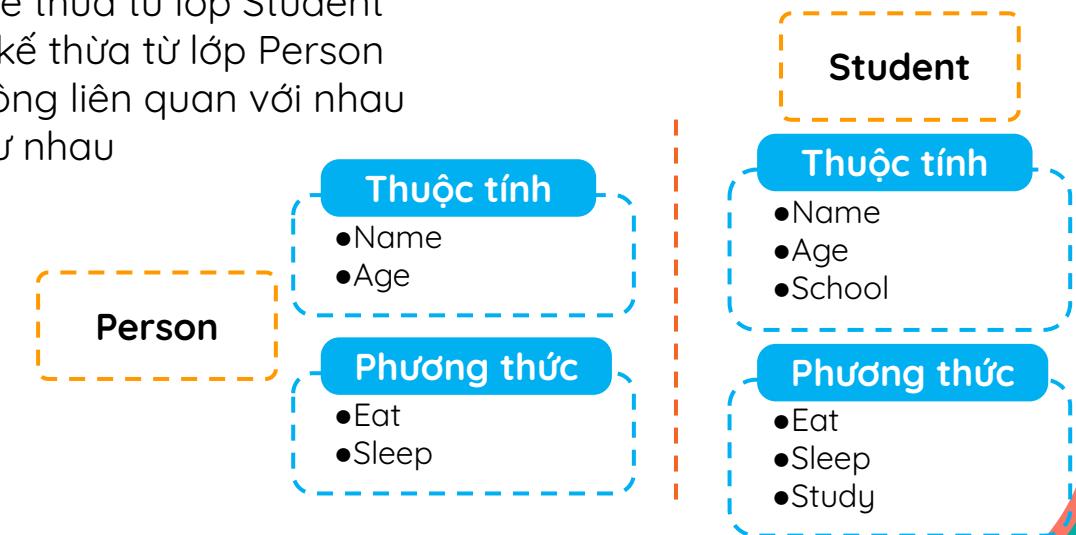
```
        print(self.name, " can fly")
```

Chỉ cần thêm một
phương thức **fly**

Quiz #5

Trong chương trình của mình, ta có lớp **Student** với các thuộc tính **name, age, school** và phương thức **eat, sleep, study**. Ta cũng có lớp **Person** với các thuộc tính **name, age**, và phương thức **eat, sleep**. Hai lớp trên có thể kế thừa nhau không?

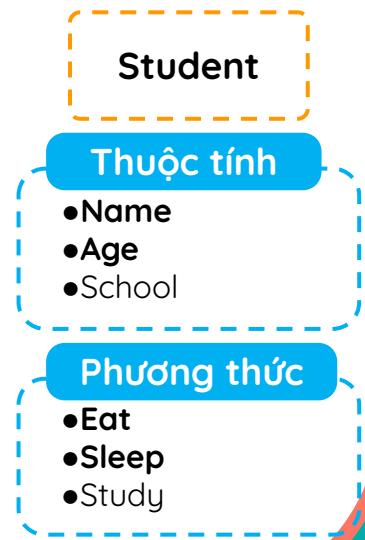
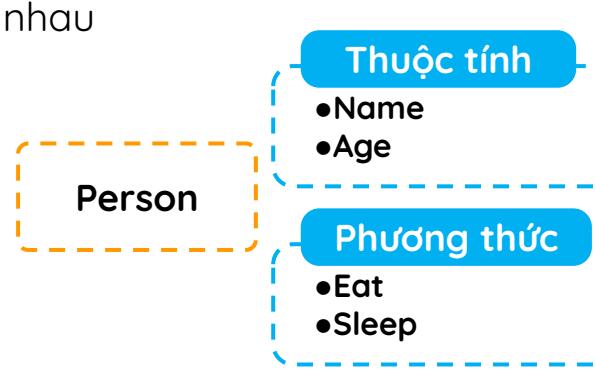
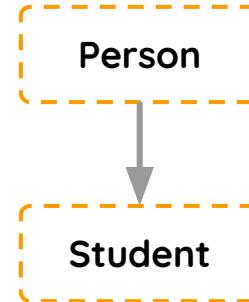
- a. Có, lớp Person có thể kế thừa từ lớp Student
- b. Có, lớp Student có thể kế thừa từ lớp Person
- c. Không, vì 2 lớp này không liên quan với nhau
- d. Không, vì 2 lớp này như nhau



Quiz #5

Trong chương trình của mình, ta có lớp **Student** với các thuộc tính **name, age, school** và phương thức **eat, sleep, study**. Ta cũng có lớp **Person** với các thuộc tính **name, age**, và phương thức **eat, sleep**. Hai lớp trên có thể kế thừa nhau không?

- a. Có, lớp Person có thể kế thừa từ lớp Student
- b. Có, lớp Student có thể kế thừa từ lớp Person**
- c. Không, vì 2 lớp này không liên quan với nhau
- d. Không, vì 2 lớp này như nhau





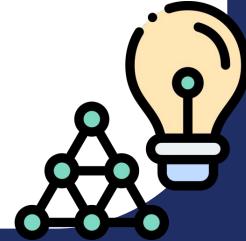
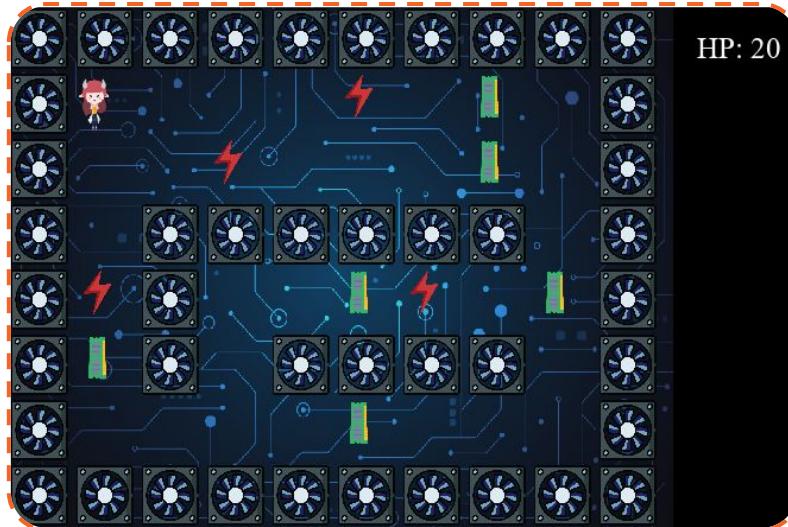
STEAM
FOR VIETNAM

PHIÊN BẢN
NÂNG CAO



LẬP TRÌNH GAME

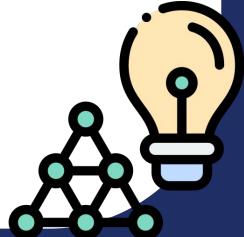
Bây giờ, chúng ta sẽ viết phần nâng cao
cho game này nhé!





LẬP TRÌNH GAME

OOP có một lợi ích là chúng ta **không cần** phải **code tất cả** mà có thể **dùng các lớp** đã được viết sẵn để **tạo đối tượng** theo ý thích của mình!





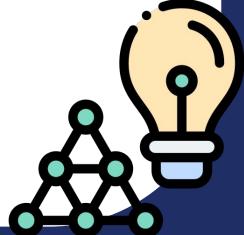
LẬP TRÌNH GAME

Thầy cô STEAM đã viết cho chúng ta
một lớp **AdvancedGame**

```
maze_game = AdvancedGame(maze_map, items_dict)  
maze_game.run()
```

Phương thức là
run

Thuộc tính là
một **bản đồ** và
các **đồ vật**

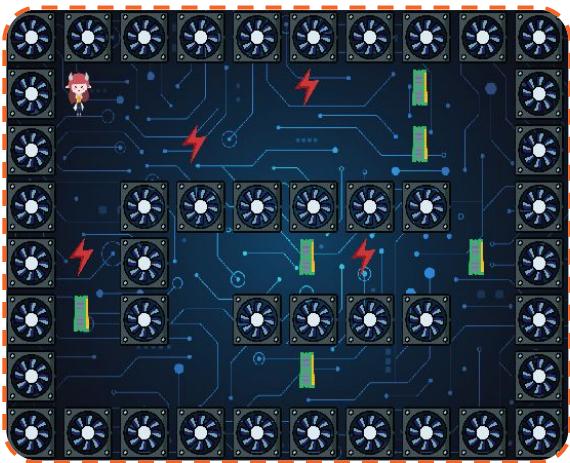




LẬP TRÌNH GAME

Tương tự như bản đồ mê cung của thây Ken,
chúng ta cũng có một bản đồ là **các ký tự**

```
maze_map = [  
    "WWWWWWWWWWWW",  
    "W * $ W",  
    "W * $ W",  
    "W WWWW W",  
    "W*W $* $W",  
    "W$W WWW W",  
    "W $ W",  
    "WWWWWWWWWW",  
]
```

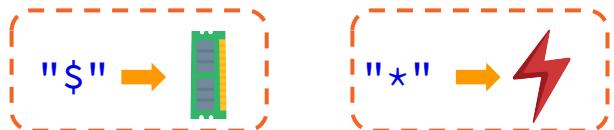
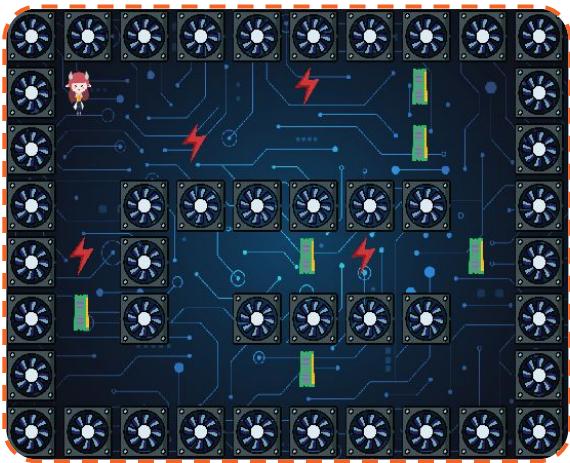




LẬP TRÌNH GAME

Tương tự như bản đồ mê cung của thây Ken,
chúng ta cũng có một bản đồ là **các ký tự**

```
maze_map = [  
    "WWWWWWWWWWWW",  
    "W * $ W",  
    "W * $ W",  
    "W WWWW W W",  
    "W*W $* $W",  
    "W$W WWW W W",  
    "W $ W",  
    "WWWWWWWWWWWW",  
]
```





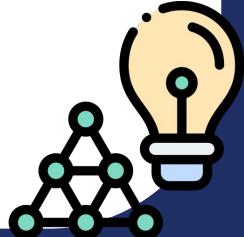
LẬP TRÌNH GAME

Chúng ta cũng cần **một từ điển** các đối tượng
để đặt vào trong bản đồ.

```
maze_map = [  
    "WWWWWWWWWWWW",  
    "W * $ W",  
    "W * $ W",  
    "W WWWW W W",  
    "W*W $* $W",  
    "W$W WWW W W",  
    "W $ W W",  
    "WWWWWWWWWWWW",  
]
```

Có bao nhiêu ký tự
phải có bấy nhiêu đồ
vật trong danh sách

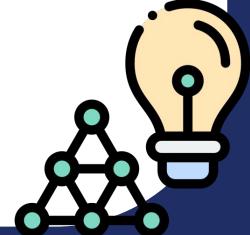
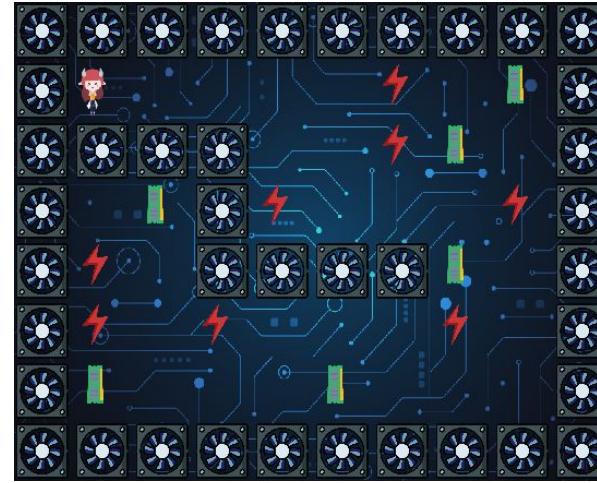
```
items_dict = {  
    "$": [RAM("inheritance"),  
          RAM("method"),  
          RAM("object"),  
          RAM("class"),  
          RAM("attribute"),  
          RAM("self")],  
    "*": [Spark(), Spark(),  
          Spark(), Spark()]  
}
```





LẬP TRÌNH GAME

Ahh! Vậy là chúng ta có thể tự tạo
những bản đồ cho riêng mình!





LẬP TRÌNH GAME

Thầy cô STEAM đã viết cho chúng ta
một lớp **Item**

Item

Lớp này sẽ giúp xác định
tọa độ của một đồ vật

```
class Item:  
    def set_position(self, x, y):  
        self.x = x  
        self.y = y
```

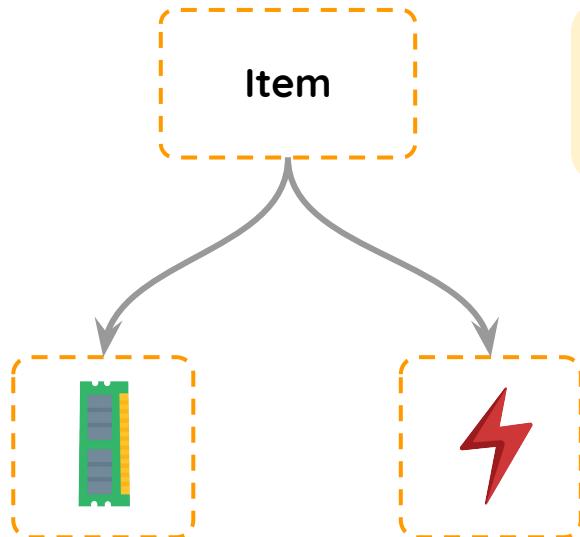
Phương thức gán tọa độ
sẽ **dựa vào bản đồ** để
đặt các đồ vật





LẬP TRÌNH GAME

Chúng ta hãy cho lớp **RAM** và **Spark**
kế thừa từ lớp **Item** này

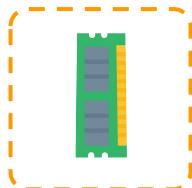


```
class Item:  
    def set_position(self, x, y):  
        self.x = x  
        self.y = y
```





LẬP TRÌNH GAME



Lớp **RAM** có thể kế thừa từ lớp **Item**

Thuộc tính

- Đường dẫn tới file hình
- Từ vựng
- Đơn vị máu tăng

Phương thức

- Thông tin khi nhặt được

Đầu tiên là kế thừa
từ lớp **Item**

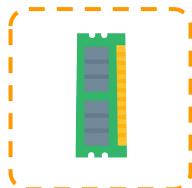
class RAM(Item):





LẬP TRÌNH GAME

Không cần tọa
độ X,Y nữa vì đã
có bản đồ



Lớp **RAM** có thể kế thừa từ lớp **Item**

Thuộc tính

- Đường dẫn tới file hình
- Từ vựng
- Đơn vị máu tăng



Phương thức

- Thông tin khi nhặt được



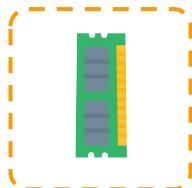
Cần thêm **3 thuộc tính**

```
class RAM(Item):  
    def __init__(self, word):  
        self.sprite_path = "sprites/ram.png"  
        self.hp_change = 10  
        self.word = word
```



LẬP TRÌNH GAME

Không cần tọa
độ X,Y nữa vì đã
có bản đồ



Lớp **RAM** có thể kế thừa từ lớp **Item**

Thuộc tính

- Đường dẫn tới file hình
- Từ vựng
- Đơn vị máu tăng

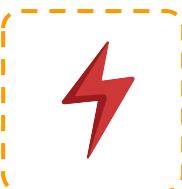
Phương thức

- Thông tin khi nhặt được

```
class RAM(Item):  
    def __init__(self, word):  
        self.sprite_path = "sprites/ram.png"  
        self.hp_change = 10  
        self.word = word  
  
    def get_info(self):  
        return "New word: " + self.word
```

Và một phương thức

LẬP TRÌNH GAME



Lớp **Spark** có thể kế thừa từ lớp **Item**

Thuộc tính

- Đường dẫn tới file hình
- Đơn vị máu tăng

Phương thức

- Thông tin khi nhặt được

Đầu tiên là kế thừa
từ lớp **Item**

```
class Spark(Item):
```





LẬP TRÌNH GAME

Không cần tọa
độ X,Y nữa vì đã
có bản đồ



Lớp **Spark** có thể kế thừa từ lớp **Item**

Thuộc tính

- Đường dẫn tới file hình
- Đơn vị máu tăng



Phương thức

- Thông tin khi nhặt được



Cần thêm **2 thuộc tính**

```
class Spark(Item):
    def __init__(self):
        self.sprite_path = "sprites/spark.png"
        self.hp_change = -30
```



LẬP TRÌNH GAME

Không cần tọa
độ X,Y nữa vì đã
có bản đồ



Thuộc tính

- Đường dẫn tới file hình
- Đơn vị máu tăng

Phương thức

- Thông tin khi nhặt được

Lớp **Spark** có thể kế thừa từ lớp **Item**

```
class Spark(Item):  
    def __init__(self):  
        self.sprite_path = "sprites/spark.png"  
        self.hp_change = -30  
  
    def get_info(self):  
        return "Spark!"
```

Và một phương thức



THÊM NHÂN VẬT MỚI

Chúng ta cũng có thể tự tạo một lớp đồ vật khác cho riêng mình



Thuộc tính

- Đường dẫn tới file hình
- Đơn vị máu tăng
- Tên

Phương thức

- Thông tin khi nhặt được

Cần xuất ra
“Robot” + tên





THÊM NHÂN VẬT MỚI

Chúng ta cũng có thể tự tạo một lớp đồ vật khác cho riêng mình



Thuộc tính

- Đường dẫn tới file hình
- Đơn vị máu tăng
- Tên

Phương thức

- Thông tin khi nhặt được

Đầu tiên là kế thừa
từ lớp **Item**

```
class Robot(Item):
```





THÊM NHÂN VẬT MỚI

Chúng ta cũng có thể tự tạo một lớp đồ vật khác cho riêng mình



Thuộc tính

- Đường dẫn tới file hình
- Đơn vị máu tăng
- Tên

Phương thức

- Thông tin khi nhặt được

```
class Robot(Item):
    def __init__(self, name):
        self.sprite_path = "sprites/tre.png"
        self.hp_change = 40
        self.name = name
```

Sau đó thêm **3 thuộc tính**



THÊM NHÂN VẬT MỚI

Chúng ta cũng có thể tự tạo một lớp đồ vật khác cho riêng mình



Thuộc tính

- Đường dẫn tới file hình
- Đơn vị máu tăng
- Tên

Phương thức

- Thông tin khi nhặt được

```
class Robot(Item):
    def __init__(self, name):
        self.sprite_path = "sprites/tre.png"
        self.hp_change = 40
        self.name = name

    def get_info(self):
        return "Robot: " + self.name
```



Và thêm một phương thức

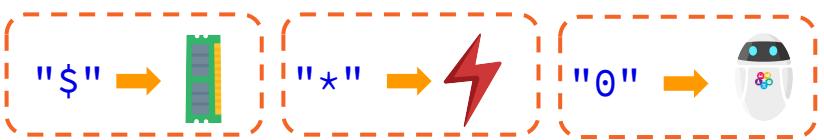
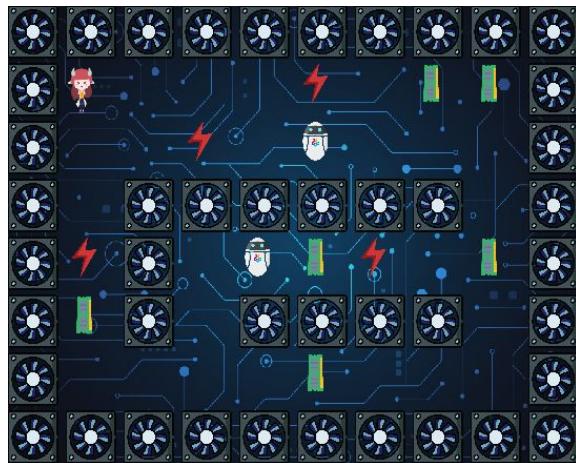




CHỈNH BẢN ĐỒ

Để thêm Robot vào bản đồ chúng ta
cần chỉnh lại bản đồ

```
maze_map = [  
    "WWWWWWWWWWWW",  
    "W * $ $ W",  
    "W * Ø W",  
    "W WWWW W W",  
    "W*W Ø$* $W",  
    "W$W WWW W W",  
    "W $ W",  
    "WWWWWWWWWWWW",  
]
```



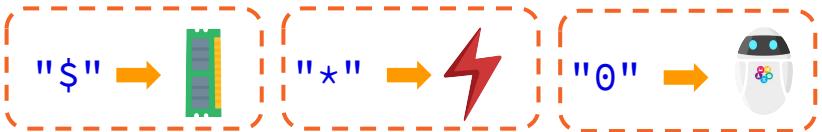


CHỈNH BẢN ĐỒ

Sau đó, mình sẽ sửa lại danh sách các đồ vật

```
maze_map = [  
    "WWWWWWWWWWWW",  
    "W * $ $ W",  
    "W * Ø W",  
    "W WWWWW W",  
    "W*W Ø$* $W",  
    "W$W WWWW W",  
    "W $ W",  
    "WWWWWWWWWW",  
]
```

```
items_dict = {  
    ...,  
    "Ø": [Robot("TRE002"),  
          Robot("TRE001")]  
}
```



CỘT MỐC 5: TÍNH KẾ THỪA

```
class Robot(Item):
    def __init__(self, name):
        self.sprite_path = "sprites/tre.png"
        self.hp_change = 20
        self.name = name

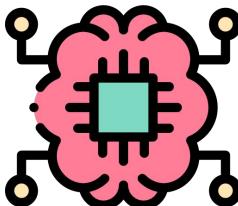
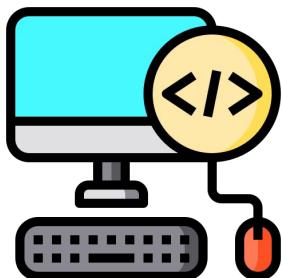
    def get_info(self):
        return "Robot: " + self.name
```

[bit.ly/S4V Lesson6 Advanced](http://bit.ly/S4V_Lesson6_Advanced)



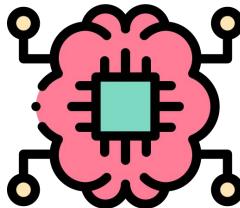
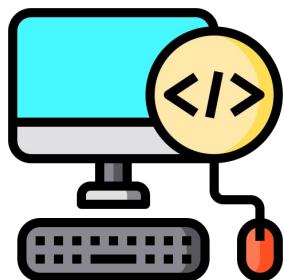
LỢI ÍCH CỦA OOP

- **Dễ hiểu** vì nó gần gũi với cách con người quan sát thế giới
- Giúp lập trình viên **chia nhỏ chương trình** để thực hiện
- **Giảm** sự **lặp lại** code





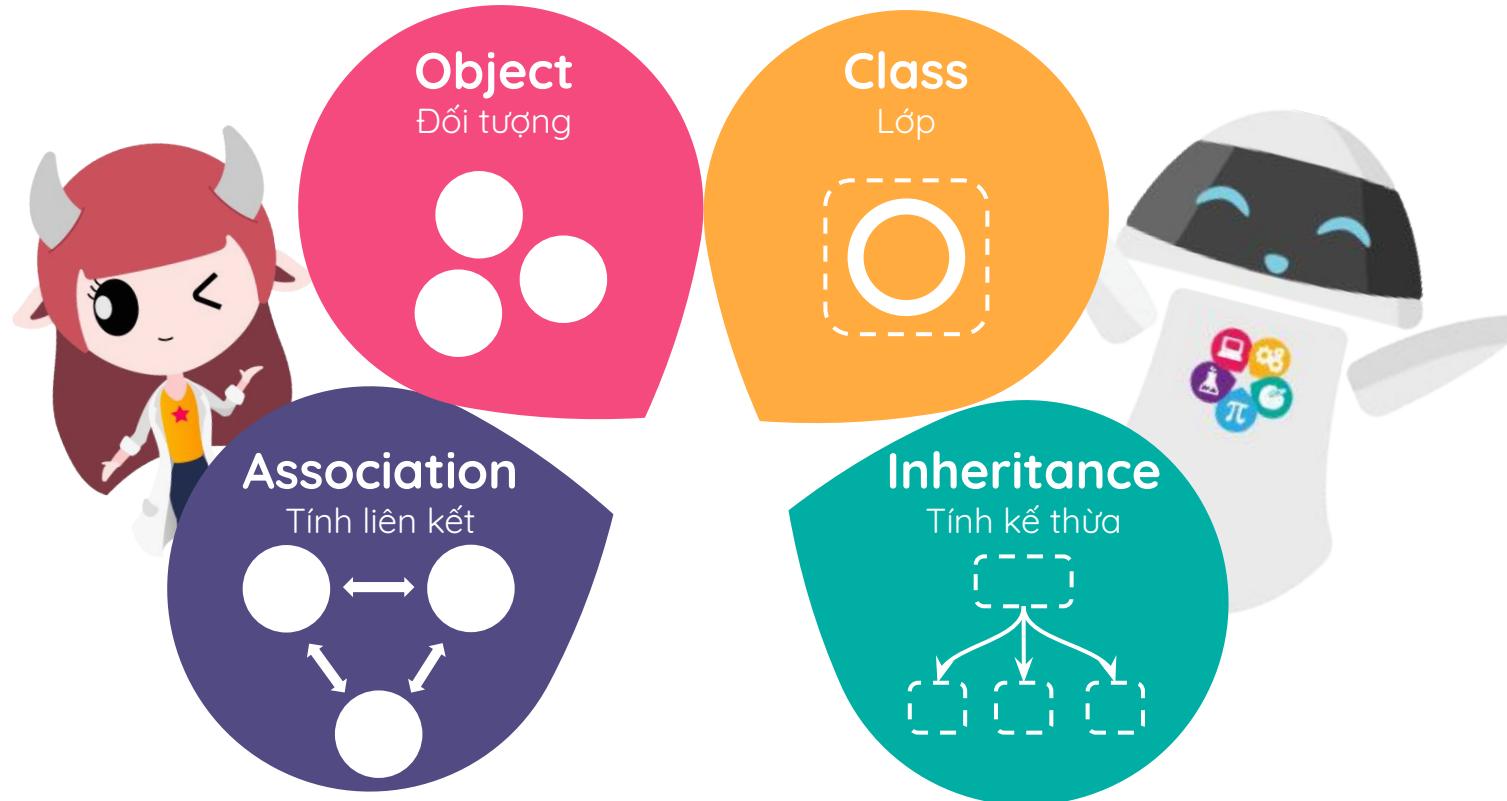
PHƯƠNG PHÁP HỌC LẬP TRÌNH HIỆU QUẢ



TỔNG KẾT



HÔM NAY CHÚNG TA ĐÃ HỌC GÌ?





THE END