



## BÀI 5

# Lập trình hướng đối tượng OOP - Python

## Tóm Tắt Nội Dung

---

Trong bài học này chúng ta sẽ đi tìm hiểu lần lượt các nội dung

- 1 Tổng quan về lập trình hướng đối tượng OOP Python
- 2 Khởi tạo Lớp, Đối Tượng
- 3 Phương thức `__init__` và `__str__`
- 4 Tính thừa kế trong lập trình hướng đối tượng Python
- 5 Ghi đè phương thức trong Class Python

## 5.1 Tổng quan về lập trình hướng đối tượng OOP



### Giới thiệu

- ◆ Lập trình hướng đối tượng là một phương pháp lập trình tập trung vào việc tổ chức code xung quanh các đối tượng (objects) và các mối quan hệ giữa chúng.
- ◆ Trong OOP, chúng ta xem mỗi đối tượng như một thực thể có trạng thái (state) và hành vi (behavior), và chúng ta tạo ra các lớp (classes) để định nghĩa các đối tượng

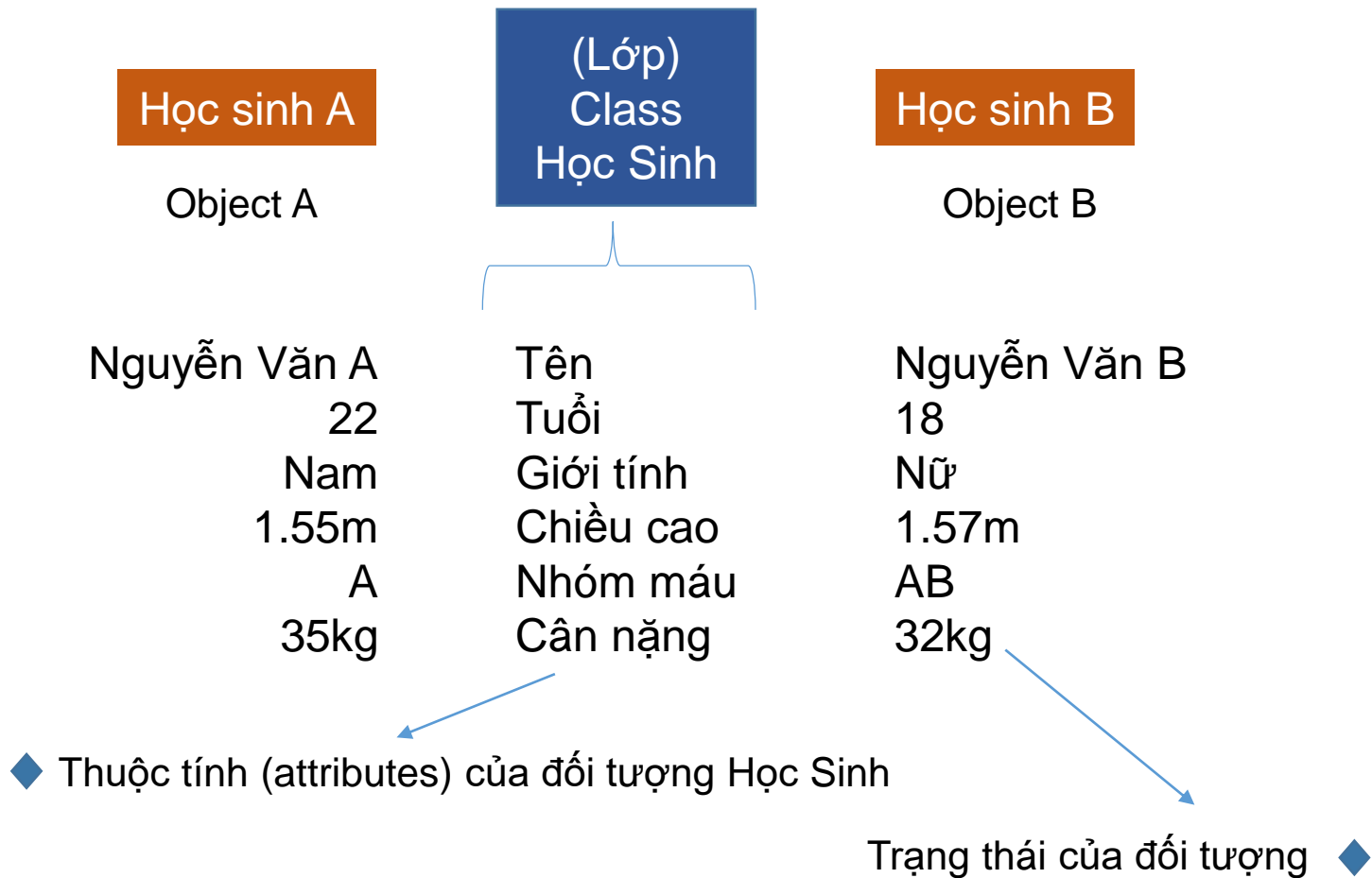
Trong thực tế các học sinh A, học sinh B được gọi là đối tượng thì ứng với trong lập trình nó là đối tượng (Object)

Học sinh A, học sinh B được gọi chung là Nhóm Học Sinh thì tương ứng trong lập trình nó là Lớp (Class)

## 5.1 Tổng quan về lập trình hướng đối tượng OOP



### Mô hình hóa khái niệm



## 5.1 Tổng quan về lập trình hướng đối tượng OOP



### Các khái niệm cơ bản trong OOP

- ➊ **Lớp (Class):** Định nghĩa một mô hình hoặc bản thiết kế cho các đối tượng
- ➋ **Đối tượng (Object):** Một thể hiện cụ thể của một lớp, có trạng thái và hành vi riêng
- ➌ **Trạng thái (State):** Các thuộc tính (attributes) của một đối tượng, biểu diễn trạng thái hiện tại của nó
- ➍ **Hành vi (Behavior):** Các phương thức (methods) của một đối tượng, biểu diễn các hành động mà đối tượng có thể thực hiện
- ➎ **Phương thức (Method):** Định nghĩa các hành vi của một lớp, làm việc với dữ liệu và thực hiện các tác vụ cụ thể
- ➏ **Thuộc tính (Attribute):** Các biến được liên kết với một đối tượng hoặc một lớp, lưu trữ thông tin về đối tượng hoặc lớp đó

## 5.2 Khởi tạo Lớp, đối tượng



Lệnh tạo Lớp (Class) trong Python

**class** ten\_lop:  
**Trạng thái của Class**

```
class HocSinh:  
    name = 'Nguyễn Văn A'  
    age = 22  
    gender = 'male'  
    weight = 35
```

Qua ví dụ trên thì chúng ta đã khai báo được một **lớp** HocSinh, có 4 thuộc tính name, age, gender, weight

## 5.2 Khởi tạo Lớp, đối tượng



### Khởi tạo đối tượng

Sau khi bạn định nghĩa một lớp (class), chúng ta có khởi tạo một đối tượng từ lớp đó như sau:

```
hs1 = HocSinh()  
print(hs1.name)
```

- ◆ Khởi tạo đối tượng **hs1** từ **Class HocSinh**
- ◆ Chúng ta gọi những đối tượng như **hs1** là một thể hiện riêng (**Instance**) của lớp đã cho
- ◆ Để truy cập đến thuộc tính cụ thể của đối tượng chúng ta có cách viết

**<đối tượng>.<tên thuộc tính>**

Ví dụ: `hs1.name`

## 5.2 Khởi tạo Lớp, đối tượng



### Khởi tạo đối tượng

Chúng ta có thể khởi tạo nhiều đối tượng hơn từ lớp HocSinh

```
hs2 = HocSinh()
```

- ◆ Tuy nhiên hs2 đang có trạng thái thông tin mặc định của Class
- ◆ Chúng ta có thể thay đổi giá trị của thuộc tính cho đối tượng hs2 bằng cách:

**<đối tượng>.<tên thuộc tính = <giá trị mới>**

```
hs2.name = 'nguyễn thị B'  
hs2.gender = 'Female'
```



## 5.2 Khởi tạo Lớp, đối tượng



### Phương thức (method) trong Lớp (Class)

Khai báo thêm các phương thức cho lớp HocSinh

```
class HocSinh:
    name = 'Nguyễn Văn A'
    age = 22
    gender = 'male'
    weight = 35
    def show(seft):
        print(seft.name, seft.gender)
    def update(seft, name, gender):
        seft.name = name
        seft.gender = gender
```

```
hs1 = HocSinh()
hs1.show() #Không cần truyền seft
```

- ◆ **seft** là từ dùng để chỉ chính bản thân đối tượng, có thể đặt tên khác nhưng Python khuyên dùng tên seft.
- ◆ Trong một số ngôn ngữ khác seft là this
- ◆ Khi gọi phương thức, chúng ta không cần truyền đối số seft vào

## 5.2 Khởi tạo Lớp, đối tượng

Chúng ta có thể sử dụng phương thức update để thay đổi giá trị của thuộc tính

```
hs1 = HocSinh()  
hs1.update('Nam', 'Male')  
  
print(hs1.name) #output Name
```

Trong ví dụ trên hàm thuộc tính **name** đã được thay đổi thành giá trị mới “Nam”

## 5.3 Phương thức `__init__` và `__str__`



### Phương thức `__init__`

- ◆ Ở ví dụ trước chúng ta đã biết cách khởi tạo nhiều đối tượng `hs2`, `hs3` từ Class `HocSinh`, sau đó đi thay đổi lại giá trị thuộc tính cho các đối tượng đó. Thì ngoài cách đó ra chúng ta còn có thể sử dụng phương thức `__init__`
- ◆ Để sử dụng `__init__` thì Class `HocSinh` được đổi thành như sau

```
class HocSinh:  
    def __init__(self, name, age, gender, weight)  
        self.name = name  
        self.age = age  
        self.gender = gender  
        self.weight = weight  
    ...
```

## 5.3 Phương thức `__init__` và `__str__`



### Phương thức `__init__`

Thì khi đó `__init__` sẽ được thực thi ngay sau khi đối tượng được khởi tạo

```
hs1 = HocSinh('Nguyễn Văn A', 18, 'Male', 30)  
hs2 = HocSinh('Nguyễn Thị B', 20, 'Female', 18)
```

Bạn khai báo giá trị cho thuộc tính ngay khi khởi tạo đối tượng, thay vì như cách ở ví dụ trước là khai báo trước trong thân của Class

```
print(hs1.show())  
print(hs2.show())
```

Chúng ta cũng được kết quả tương tự như ví dụ trước

## 5.3 Phương thức `__init__` và `__str__`



### Phương thức `__str__`

là một phương thức đặc biệt trong Python, được sử dụng để định nghĩa một chuỗi biểu diễn của đối tượng khi chúng ta muốn đối tượng đó thành (trả về) một chuỗi.

```
class HocSinh:
    ...
    def __str__(self):
        return f"HocSinh(name='{self.name}', age={self.age})"

# Tạo một đối tượng HocSinh
hs1 = HocSinh('Nguyễn Văn A', 18, 'Male', 30)
# Có thể dùng lệnh print để in hs1
print(hs1)
```

Có thể hiểu nó như function có return trả về và dữ liệu trả về là kiểu chuỗi

## 5.2 Khởi tạo Lớp, đối tượng



### Các đặc tính của lập trình hướng đối tượng

- 1 **Tính giao tiếp (Interface):** là khả năng của một đối tượng để tương tác với các đối tượng khác thông qua việc gửi và nhận thông điệp
- 2 **Tính chất trừu tượng (Abstraction):** Tạo ra các lớp và đối tượng trừu tượng để tập trung vào cốt lõi của vấn đề, ẩn đi các chi tiết triển khai.
- 3 **Tính kế thừa (Inheritance):** Cho phép xây dựng các lớp dẫn xuất (derived classes) dựa trên các lớp gốc (base classes), giúp mở rộng và mở rộng chức năng của mã hiện có.
- 4 **Tính đóng gói (Encapsulation):** Tách biệt dữ liệu và hành vi của đối tượng, che dấu thông tin nội bộ và chi tiết, giúp bảo mật và quản lý dễ dàng
- 5 **Tính đa hình (Polymorphism):** Cho phép sử dụng các đối tượng của các lớp khác nhau trong cùng một giao diện chung, giúp tăng tính linh hoạt và tái sử dụng

## 5.4 Tính thừa kế trong Class

- ◆ **Tính kế thừa** là khả năng của một lớp (lớp con) để kế thừa các thuộc tính và phương thức từ một lớp khác (lớp cha)
- ◆ Kế thừa cho phép tái sử dụng mã nguồn đã được viết, mở rộng chức năng của lớp cha và tạo mối quan hệ phân cấp giữa các lớp



### Cú pháp để tạo kế thừa giữa các lớp trong Python

Để tạo một lớp con, khai báo lớp con bằng cách sử dụng từ khóa "class" và sau đó chỉ định lớp cha trong dấu ngoặc đơn "(" sau tên lớp con, ví dụ:

**class ChildClass(ParentClass):**

ChildClass là Class CON (Class đi kế thừa)

ParentClass là Class CHA (Class được kế thừa)

→ Được hiểu là ChildClass kế thừa ParentClass

## 5.4 Tính thừa kế trong Class



### Ví dụ về tính kế thừa

```
class Vehicle: #Phương tiện giao thông
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color
    def drive(self):
        print(f"Chiếc xe màu {self.color} hãng {self.brand} đang chạy.")

class Car(Vehicle): #Xe ô tô
    def honk(self):
        print("Kêu Beep beep!")

class Bike(Vehicle): #Xe đạp
    def ring_bell(self):
        print("Kêu Ring ring!")
```



## 5.4 Tính thừa kế trong Class



### Ví dụ về tính kế thừa

```
# Tạo các đối tượng từ các lớp con
car = Car("Toyota", "red")
bike = Bike("Giant", "blue")

# Gọi phương thức từ lớp cha
car.drive() # Output: Chiếc xe màu red hãng Toyota đang chạy.
bike.drive() # Output: Chiếc xe màu blue hãng Giant đang chạy.

# Gọi phương thức từ các lớp
con car.honk() # Output: Kêu Beep beep!
bike.ring_bell() # Output: Kêu Ring ring!
```

## 5.4 Tính thừa kế trong Class



### Ví dụ về tính kế thừa

- Trong ví dụ này, chúng ta có lớp cha **Vehicle** với phương thức **drive()**, mô phỏng hành động của việc lái xe. Các lớp con **Car** và **Bike** kế thừa phương thức **drive()** từ lớp cha **Vehicle**

```
# Gọi phương thức từ lớp cha  
car.drive() # Output: Chiếc xe màu red hãng Toyota đang chạy.  
bike.drive() # Output: Chiếc xe màu blue hãng Giant đang chạy.
```

- Lớp **Car** có một phương thức riêng là **honk()** để còi xe,  
Lớp **Bike** có phương thức riêng là **ring\_bell()** để chuông xe đạp

```
# Gọi phương thức từ các lớp  
con car.honk() # Output: Kêu Beep beep!  
bike.ring_bell() # Output: Kêu Ring ring!
```

## 5.4 Tính thừa kế trong Class



### Truy cập đến thuộc tính, phương thức của Class Cha

Chúng ta hoàn toàn có thể truy cập đến thuộc tính của class Cha từ Class Con khi đang ở trong phần thân của Class Con như sau

```
class WheelChair(Vehicle): #Xe lăn
    def __init__(self, brand, color, object): #khởi tạo Class wheelchair
        super().__init__(brand, color) #khởi tạo Class CHA Vehicle
        self.object = object #Thuộc tính riêng của Wheelchair

    def move(self): #Phương thức riêng của Wheelchair
        super().drive() #Truy cập đến phương thức drive của CHA
        print(f"Xe lăn màu {super().color} hãng {super().brand} điều khiển  
bằng tay bởi {self.object}.")

xelan = WheelChair('Global', 'Black', 'Jane')
xelan.move() #Output: Chiếc xe màu Black hãng Global đang chạy
Xe lăn màu Black hãng Global điều khiển bởi Jane
```

## 5.4 Tính thừa kế trong Class



### Truy cập đến thuộc tính, phương thức của Class Cha

Qua ví dụ trên, chúng ta phải khởi tạo phương thức `__init__` của Class CHA từ class CON, và sau đó có thể truy cập đến các phương thức của class CHA bằng phương thức `supper()` với cú pháp:

◆ `supper().tên_thuộc_tính`

```
print(f"Xe lăn màu {supper().color} hãng {supper().brand} điều khiển bằng  
tay bởi {self.object}.")
```

◆ `supper().tên_phương_thức()`

```
super().drive() #Truy cập đến phương thức drive của CHA
```

## 5.5 Ghi đè phương thức trong Class

- ◆ Ví dụ cho một class CHA như sau:

```
class Animal(): #Động vật
    def speak(self):
        print("Động vật kêu")
```

- ◆ Và bạn muốn lớp CON ví dụ Dog kế thừa từ Animal nhưng Dog bạn lại muốn nó kêu kiểu khác thì bạn có thể **ghi đè** lại phương thức **speak()** của lớp CHA như sau

```
class Dog(Animal): #Lớp Con Chó
    def speak(self):
        print("Chó kêu gâu gâu")
```



Đặt **trùng tên** với phương thức CHA và **code khác** đi thì có nghĩa bạn đang **ghi đè** phương thức đó từ Class CHA

# Tổng kết lại bài 5

---

1. Nắm bắt được tổng quan về lập trình hướng đối tượng OOP Python
2. Nắm được khái niệm cơ bản trong lập trình hướng đối tượng, Biết cách Khởi tạo Lớp, Đối Tượng
3. Biết cách sử dụng phương thức `__init__` và `__str__`
4. Hiểu như thế nào là tính thừa kế trong lập trình hướng đối tượng Python, và cách để kế thừa từ một lớp CHA
5. Hiểu thế nào là ghi đè phương thức trong Class Python