LẬP TRÌNH PYTHON CƠ BẢN

BÀI 5

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG PYTHON

NỘI DUNG

1. Giới thiệu
2. Cài đặt lớp
3. Quan hệ kế thừa
4. Quan hệ hợp thành
5. Lớp cơ sở trừu tượng
6. Tính đa hình

3. KẾ THỪA

3. KẾ THỪA

Kế thừa trong Python

- Là một ngôn ngữ hướng đối tượng, Python hỗ trợ kế thừa lớp. Nó cho phép chúng ta tạo một lớp mới từ một lớp hiện có.
- Lớp mới được tạo ra được gọi là ***subclass*** (lớp con hoặc lớp dẫn xuất)
- Lớp hiện tại được lớp con kế thừa được gọi là ***superclass*** (lớp cha hoặc lớp cơ sở).
- Lớp con kế thừa tất cả thành phần của lớp cha, có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới.
- Kế thừa là một mối quan hệ **is-a**.
  - VD: Car **is a** Vehicle.

3. KẾ THỪA

Kế thừa trong Python

- Cú pháp:

```
# Định nghĩa lớp cha
class super_class:
    # Định nghĩa các thuộc tính và phương thức

# Định nghĩa lớp con
class sub_class(super_class):
    # Thuộc tính và phương thức của lớp cha
    # Thuộc tính và phương thức của lớp con
```

- Lớp cha sẽ không bị ảnh hưởng khi định nghĩa lớp con
- Lớp cha phải được định nghĩa trước khi thực hiện định nghĩa lớp con, nếu không chương trình sẽ không hiểu nó kế thừa từ lớp nào.
- Khi định nghĩa lớp con chúng ta cần thêm dấu ngoặc tròn và chỉ rõ tham số phía trong là lớp con đó kế thừa từ lớp cha nào.

3. KẾ THỪA

Kế thừa trong Python

- Ví dụ:
  - Định nghĩa lớp cha Person
  - Lớp Student kế thừa lớp Person mà không thêm bất cứ thuộc tính và phương thức nào

  => Lớp Student có cùng thuộc tính và phương thức giống như lớp Person

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

    def info(self):
        print(self.name + ",", self.age, "years old.")

class Student(Person):
    pass

# Tạo một đối tượng thuộc lớp Student
kane = Student("Kane", 29)
kane.info()
```

```
Kane, 29 years old.
```

Kế thừa trong Python

Định nghĩa hàm __init__ trong lớp con

- Chúng ta có thể định nghĩa lại hàm khởi tạo __init__() trong lớp con.
- Lúc này, hàm __init__() trong lớp con sẽ ghi đè (overriding) lên hàm __init__() kế thừa được từ lớp cha.

```
# Student inherits from Person
class Student(Person):
    def __init__(self, name, age):
        Person.__init__(name, age)
```

```
# Student inherits from Person
class Student(Person):
    def __init__(self, name, age):
        super().__init__(name, age)
```

- Hàm ***super()*** giúp lớp con kế thừa các thuộc tính và phương thức từ lớp cha.

Ghi đè phương thức

- Lớp con có thể định nghĩa lại các phương thức được kế thừa từ lớp cha (method overriding).

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

    def info(self):
        print(self.name + ",", self.age, "years old.")
```

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

    def info(self):
        print(self.name + ",", self.age, "years old, id:", self.id)

son = Student("Son", 30, "0469191517")
son.info()
```

```
Son, 30 years old, id: 0469191517
```

BÀI TẬP 5.2

Viết chương trình Python thực hiện các yêu cầu sau:

- Tạo một lớp cơ bản **Shape** với thuộc tính **color** (màu sắc) và phương thức **draw** (vẽ).
- Lớp **Rectangle** kế thừa lớp **Shape**, có thêm các thuộc tính **width** (chiều rộng) và **height** (chiều cao) và phương thức **draw**.
- Hãy tạo một hình chữ nhật và thực hiện hành vi vẽ hình chữ nhật đó.

BÀI TẬP 5.2

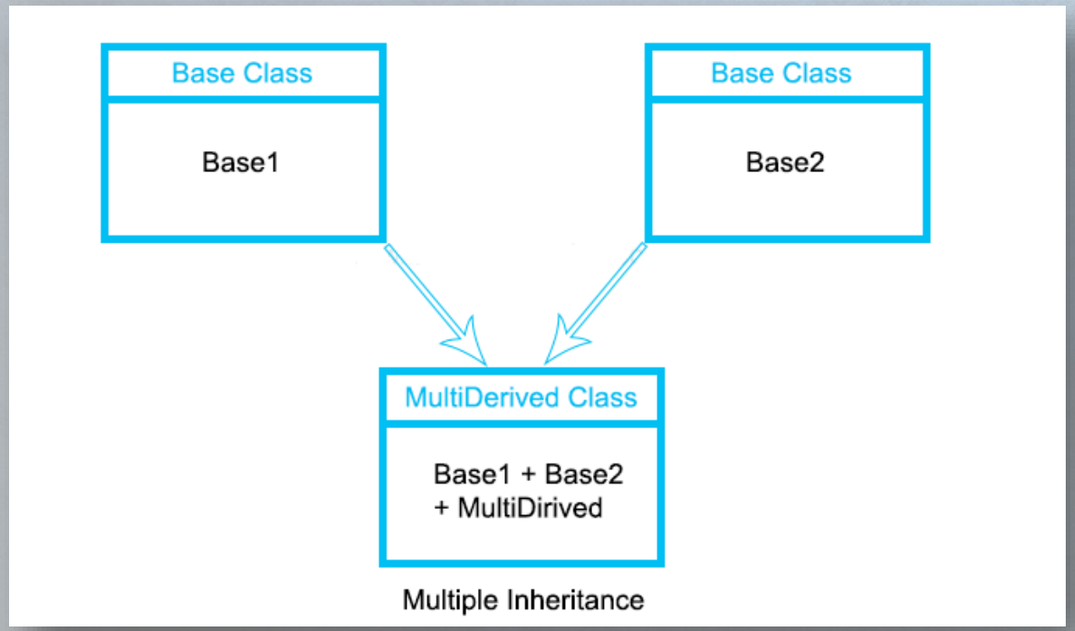
```
class Shape:
    def __init__(self, color):
        self.color = color

    def draw(self):
        pass  # Phương thức này sẽ được ghi đè bởi lớp con
```

```
class Rectangle(Shape):
    def __init__(self, color, width, height):
        super().__init__(color)
        self.width = width
        self.height = height

    def draw(self):
        print(f"Vẽ HCN có màu {self.color}, rộng {self.width} và cao {self.height}")
```

```
rectangle = Rectangle("xanh dương", 5, 10)
rectangle.draw()
```

Đa kế thừa (multiple inheritance)

- Một lớp con có thể kế thừa từ nhiều hơn một lớp cha. Đó gọi là đa kế thừa.
- Tất cả các thuộc tính và phương thức của tất cả lớp cho được kế thừa bởi lớp con.




```
class Base1:
  pass

class Base2:
  pass

class MultiDerived(Base1, Base2):
  pass
```

Đa kế thừa (multiple inheritance)

- Ví dụ:
  - Lớp ***Teacher*** kế thừa từ lớp ***Person*** và ***Employee***.
  - Lớp ***Teacher*** sẽ kế thừa và sử dụng các phương thức ***info()*** và ***showSalary()*** từ 2 lớp trên.

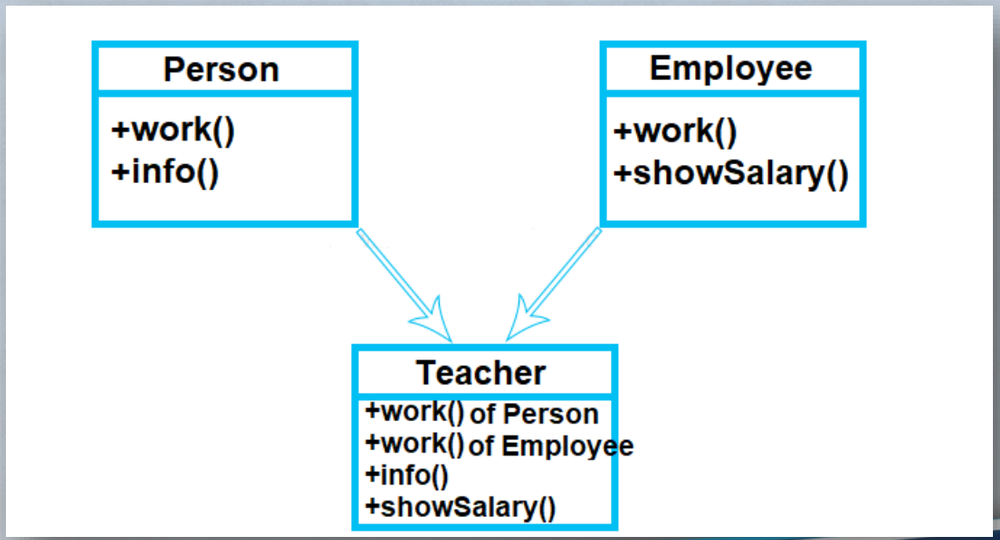
```
Info of a person.
Salary of an employee.
```

```
class Person:
    def info(self):
        print("Info of a person.")

class Employee:
    def showSalary(self):
        print("Salary of an employee.")

class Teacher(Person, Employee):
    pass

john = Teacher()
john.info()
john.showSalary()
```

Đa kế thừa (multiple inheritance)

Method Resolution Order (MRO)

- Có trường hợp các lớp cha có những phương thức cùng tên với nhau và lớp con gọi những phương thức này. Vậy thì phương thức của lớp cha nào sẽ được thực thi khi lớp con gọi đến?



3. KẾ THỪA

Đa kế thừa (multiple inheritance)

Method Resolution Order (MRO)

- Python sử dụng Method Resolution Order (MRO) để quy định phương thức được gọi trong trường hợp này
- MRO chỉ ra một thứ tự lớp nào được ưu tiên tìm và gọi phương thức trước
- Có thể sử dụng thuộc tính **__mro__** hoặc hàm **mro()** để xem thứ tự này.

```
class Person:
    def info(self):
        print("Info of a person.")
    def work(self):
        print("A person works.")

class Employee:
    def showSalary(self):
        print("Salary of an employee.")
    def work(self):
        print("An employee works.")

class Teacher(Person, Employee):
    pass

print(Teacher.__mro__)
john = Teacher()
john.work()
```

```
(<class '__main__.Teacher'>, <class '__main__.Person'>, <class '__main__.Employee'>, <class 'object'>)
A person works.
```

Đa kế thừa (multiple inheritance)

Method Resolution Order (MRO)

- Có trường hợp các lớp cha có những phương thức cùng tên với nhau và lớp con gọi những phương thức này. Vậy thì phương thức của lớp cha nào sẽ được thực thi khi lớp con gọi đến?

```
class Teacher(Employee, Person):
  pass

print(Teacher.mro())
john = Teacher()
john.work()
```

```
(<class '__main__.Teacher'>, <class '__main__.Employee'>, <class '__main__.Person'>, <class 'object'>)
An employee works.
```

Đa kế thừa (multiple inheritance)

Hàm __init__() trong đa kế thừa

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

    def info(self):
        print(self.name + ",", self.age, "years old.")

class Employee:
    def __init__(self, salary):
        self.salary = salary

    def showSalary(self):
        print("salary:", self.salary)
```

```
print(Teacher.mro())
john = Teacher("John", 35)
# call method inherits from Person
john.info()
# AttributeError: 'Teacher' object has no attribute 'salary'
john.showSalary()
```

```
[<class '__main__.Teacher'>, <class '__main__.Person'>,
<class '__main__.Employee'>, <class 'object'>]
John, 35 years old.
Traceback (most recent call last):
  File "c:\python-examples\main.py", line 20, in <module>
    john.showSalary()
  File "c:\python-examples\main.py", line 13, in showSalary
    print("salary:", self.salary)
AttributeError: 'Teacher' object has no attribute 'salary'
```

Đa kế thừa (multiple inheritance)

Hàm __init__() trong đa kế thừa

- Ta sử dụng câu lệnh

  super().__init__(name, age)

gọi hàm khởi tạo của lớp cha **Person**.

- Sau đó, thêm thuộc tính **salary** cho các **object** của **Teacher**.

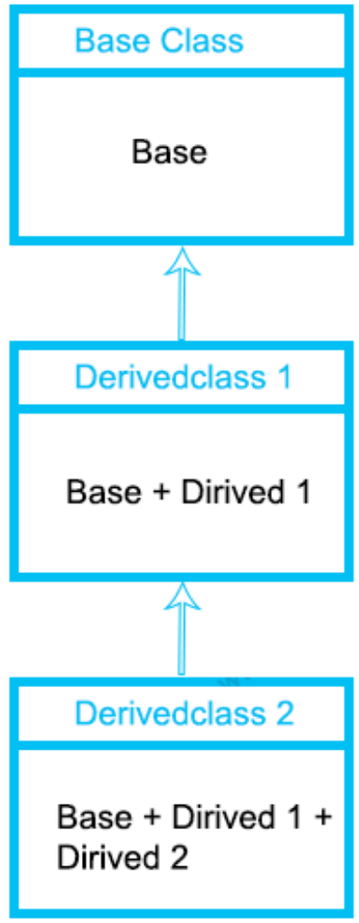
```
class Teacher(Person, Employee):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary

# create an object of Teacher with it's __init__() function
john = Teacher("John", 35, "5000000")
# call method inherits from Person
john.info()
# call method inherits from Employee
john.showSalary()
```

```
John, 35 years old.
salary: 5000000
```

3. KẾ THỪA

Kế thừa đa cấp (multilevel inheritance)

- Ta cũng có thể tạo ra một lớp mới bằng cách kế thừa từ một lớp con. Đó gọi là kế thừa nhiều cấp (multilevel inheritance).
- Python không giới hạn số cấp kế thừa.
- Trong multilevel inheritance, tất cả thuộc tính và phương thức của lớp cha và lớp con sẽ được kế thừa bởi lớp con mới.



Multilevel Inheritance

Kế thừa đa cấp (multilevel inheritance)

- Ví dụ:

```
class Base:
    def methodBase(self):
        print("This is a method of Base.")


class Derived1(Base):
    def methodDerived1(self):
        print("This is a method of Derived1.")


class Derived2(Derived1):
    def methodDerived2(self):
        print("This is a method of Derived2.")
```

```
# create an object of Derived2
obj = Derived2()
# call method inherits from Base
obj.methodBase()
# call method inherits from Derived1
obj.methodDerived1()
# call method of Derived2
obj.methodDerived2()
```

```
This is a method of Base.
This is a method of Derived1.
This is a method of Derived2.
```

BÀI TẬP 5.3

Viết chương trình Python thực hiện các yêu cầu sau:

- Tạo một lớp cơ bản NhanVien với các thuộc tính chung:
  - ma_so: Mã số của nhân viên.
  - ho_ten: Họ và tên của nhân viên.
  - luong_co_ban: Lương cơ bản.
- Tạo hai lớp con: NhanVienBanHang và NhanVienQuanLy.
- Lớp NhanVienBanHang kế thừa từ lớp NhanVien. Nó có một thuộc tính bổ sung:
  - doanh_so: Doanh số bán hàng của nhân viên.
- Lớp NhanVienQuanLy cũng kế thừa từ lớp NhanVien. Nó có một thuộc tính bổ sung:
  - so_ngay_cong: Số ngày làm việc trong tháng của nhân viên.
- Cả hai lớp con đều có phương thức tinh_luong() để tính lương của nhân viên theo công thức sau:
  - Đối với NhanVienBanHang: Lương = Lương cơ bản + 0.05 * Doanh số.
  - Đối với NhanVienQuanLy: Lương = Lương cơ bản + 50 * Số ngày làm việc.
- Tạo và hiển thị thông tin của một số nhân viên bán hàng và nhân viên quản lý, bao gồm mã số, họ tên và lương của họ.

BÀI TẬP 5.3

```
class NhanVien:
    def __init__(self, ma_so, ho_ten, luong_co_ban):
        self.ma_so = ma_so
        self.ho_ten = ho_ten
        self.luong_co_ban = luong_co_ban

    def tinh_luong(self):
        pass  # Được cài đặt trong các lớp con
```

```
class NhanVienBanHang(NhanVien):
    def __init__(self, ma_so, ho_ten, luong_co_ban, doanh_so):
        super().__init__(ma_so, ho_ten, luong_co_ban)
        self.doanh_so = doanh_so

    def tinh_luong(self):
        return self.luong_co_ban + 0.05 * self.doanh_so
```

```
class NhanVienQuanLy(NhanVien):
    def __init__(self, ma_so, ho_ten, luong_co_ban, so_ngay_cong):
        super().__init__(ma_so, ho_ten, luong_co_ban)
        self.so_ngay_cong = so_ngay_cong

    def tinh_luong(self):
        return self.luong_co_ban + 50 * self.so_ngay_cong
```

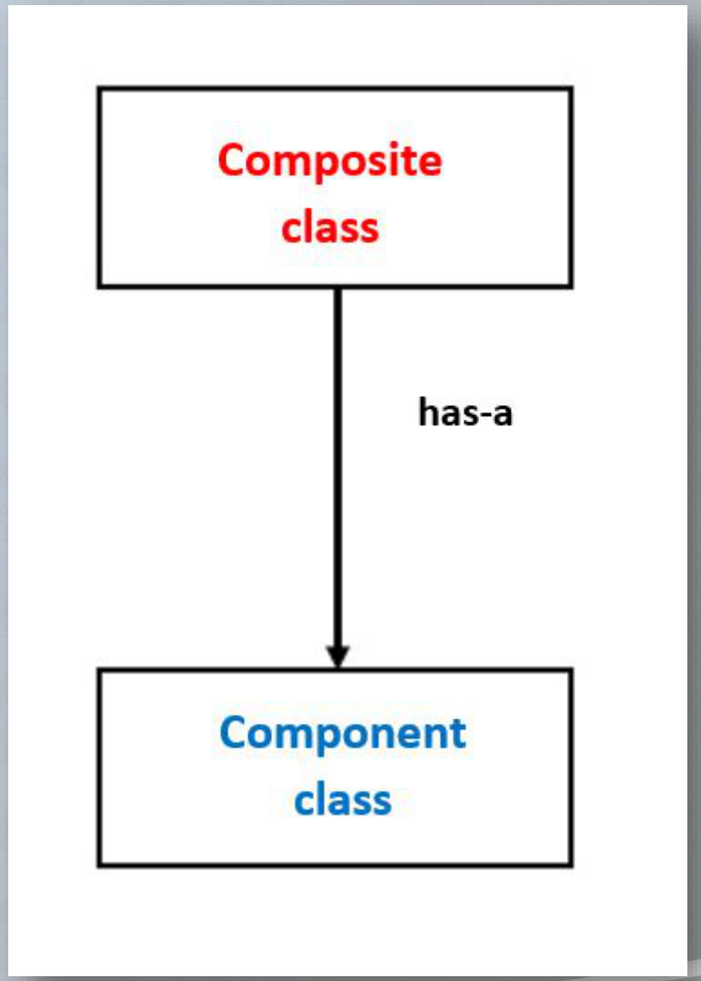
```
nhan_vien_ban_hang = NhanVienBanHang("NV001", "Nguyen Van A", 1000000, 5000000)
nhan_vien_quan_ly = NhanVienQuanLy("NV002", "Tran Thi B", 2000000, 25)

print("Thông tin nhân viên bán hàng:")
print("Mã số:", nhan_vien_ban_hang.ma_so)
print("Họ tên:", nhan_vien_ban_hang.ho_ten)
print("Lương:", nhan_vien_ban_hang.tinh_luong())

print("\nThông tin nhân viên quản lý:")
print("Mã số:", nhan_vien_quan_ly.ma_so)
print("Họ tên:", nhan_vien_quan_ly.ho_ten)
print("Lương:", nhan_vien_quan_ly.tinh_luong())
```

4. HỢP THÀNH

4. HỢP THÀNH

Hợp thành trong Python

- Hợp thành (Composition) là một khái niệm thiết kế hướng đối tượng theo mô hình quan hệ **has-a**.
- Một lớp được gọi là tổng thể chứa một đối tượng hoặc thành phần của một lớp khác. Nói cách khác, một **lớp toàn thể** có chứa đối tượng của một **lớp thành phần**.
- Bằng cách sử dụng tên lớp hoặc bằng cách tạo một đối tượng, chúng ta có thể truy cập thành viên của một lớp bên trong một lớp khác.



Hợp thành trong Python

- Ví dụ:

```
obj2 = Composite()

# calling m2() method of composite class
obj2.m2()
```

```
Component class object created...
Composite class object also created...
Composite class m1() method executed...
Component class m2() method executed...
```

```
class Component:
    def __init__(self):
        print('Component class object created...')

    def m1(self):
        print('Component class m1() method executed...')
```

```
class Composite:
    def __init__(self):
        self.obj1 = Component()
        print('Composite class object also created...')

    def m2(self):
        print('Composite class m2() method executed...')
        self.obj1.m1()
```

BÀI TẬP 5.5

Viết chương trình Python thực hiện các yêu cầu sau:

- Cài đặt lớp **Diem** (điểm) gồm các thuộc tính là hoành độ và tung độ trên mặt phẳng và các phương thức cần thiết.
- Cài đặt lớp **DoanThang** (đoạn thẳng) gồm các thuộc tính là hai điểm mút và các phương thức cần thiết.
- Cài đặt lớp **TamGiac** (tam giác) kế thừa lớp đoạn thẳng và có thêm một điểm và các phương thức cần thiết.
- Thực hiện nhập tọa độ 3 đỉnh của một tam giác, tính và in ra màn hình diện tích tam giác đó.

BÀI TẬP 5.5

```
class Point:
    def __init__(self, name):
        print(f'Nhập tọa độ điểm {name}:')
        self.x = float(input('x = '))
        self.y = float(input('y = '))
```

```
class Line:
    def __init__(self, name1, name2):
        self.p1 = Point(name1)
        self.p2 = Point(name2)

    @staticmethod
    def length(A, B):
        return math.sqrt((B.x - A.x) ** 2 + (B.y - A.y) ** 2)
```

```
class Triangle(Line):
    def __init__(self, name1, name2, name3):
        super().__init__(name1, name2)
        self.p3 = Point(name3)

    def area(self):
        a = self.length(self.p1, self.p2)
        b = super().length(self.p2, self.p3)
        c = Line.length(self.p3, self.p1)
        p = (a + b + c) / 2
        return math.sqrt(p * (p - a) * (p - b) * (p - c))
```

```
triangle = Triangle('A', 'B', 'C')
print(f"Diện tích tam giác là: {triangle.area():0.2f}")
```

KẾT THÚC