# python复习5 面向对象1

1. 面向对象三大特性

封装：封装是指将数据和操作数据的方法（函数）打包在一起，形成一个类（class）。

```python
class Car:
    def __init__(self, brand, model):
        self.brand = brand  # 公有属性
        self.__model = model  # 私有属性

    def get_model(self):  # 公有方法
        return self.__model

    def set_model(self, new_model):  # 公有方法
        self.__model = new_model

my_car = Car("Toyota", "Corolla")
print(my_car.brand)  # 可以直接访问公有属性
# print(my_car.__model)  # 无法直接访问私有属性，会报错
print(my_car.get_model())  # 通过公有方法访问私有属性
my_car.set_model("Camry")  # 通过公有方法修改私有属性
```

继承：继承允许一个类（子类）继承另一个类（父类）的属性和方法，以便重用代码并扩展现有功能。

```python
class Animal:
    def __init__(self, species):
        self.species = species

    def make_sound(self):
        pass  # 抽象方法

class Dog(Animal):  # Dog类继承自Animal类
    def __init__(self, breed):
        super().__init__("Dog")
        self.breed = breed

    def make_sound(self):
        return "Woof!"

```

```
my_dog = Dog("Labrador")
print(my_dog.species)  # 继承了父类的属性
print(my_dog.breed)  # 子类独有的属性
print(my_dog.make_sound())  # 子类重写了父类的方法
```

多态：多态允许不同类的对象对同一方法做出响应，实现了不同类对象对同一消息做出不同响应的能力。

```
class Cat:
    def make_sound(self):
        return "Meow!"

class Duck:
    def make_sound(self):
        return "Quack!"

def animal_sound(animal):
    return animal.make_sound()

cat = Cat()
duck = Duck()

print(animal_sound(cat))  # Cat对象调用自己的make_sound方法
print(animal_sound(duck))  # Duck对象调用自己的make_sound方法
```

2. 定义类的三种方法

- 实例方法

没有任何装饰器，有默认self的普通函数

```
class MyClass:
    def instance_method(self, arg1, arg2):
        # 使用self来访问实例属性
        self.arg1 = arg1
        self.arg2 = arg2
        return self.arg1 + self.arg2

obj = MyClass()
result = obj.instance_method(3, 4)
```

- 类方法

有classmethod装饰的函数。既可以被类调用，也可以被实例调用

```
class MyClass:
    class_variable = 10

    @classmethod
    def class_method(cls, arg):
        cls.class_variable += arg
        return cls.class_variable

result = MyClass.class_method(5)
```

- 静态方法

有staticmethod装饰的函数。静态方法无法通过self或cls访问到实例或类中的属性。静态方法可以被实例或类调用

```
class MyClass:
    @staticmethod
    def static_method(arg1, arg2):
        return arg1 + arg2

result = MyClass.static_method(3, 4)
```

区别：

- 实例方法需要一个实例，而且可以访问实例属性。
- 类方法需要类作为第一个参数，可以访问和修改类属性，但不能直接访问实例属性。
- 静态方法不需要实例或类作为参数，不能访问实例或类属性。

3. 绑定方法

**实例方法和类方法都是绑定方法。**

类中的方法或函数，默认都是绑定给实例使用的；绑定方法有自动传值的功能，传递的值就是对象本身（self）。

当类调用实例方法，实例方法仅被视为函数，无自动传值的功能。

通过classmethod装饰器，将绑定给实例的方法，绑定到了类。

**静态方法是类的非绑定方法。**

通过staticmethod装饰器，可以解除绑定关系，将一个类中的方法，变为一个普通函数。

静态方法中参数传递跟普通函数相同，无需考虑自动传参等问题。

4. 方法与函数

- 方法（method）

实例方法和类方法都是方法。

**方法是一种和对象（实例、类）绑定的特殊函数。**

- 函数（function）

普通函数（未定义在类里）是函数。

静态方法是函数。

5. 私有变量与私有方法

- 类的私有属性

使用两个下划线开头声明该属性为私有属性。

私有属性不能在类的外部被直接访问。

```
class MyClass:
    def __init__(self):
        self.public_attribute = "I am public"
        self.__private_attribute = "I am private" # 私有属性

obj = MyClass()
print(obj.public_attribute)  # 可以直接访问公有属性
# print(obj.__private_attribute)  # 不能直接访问私有属性，会报错
```

类的私有属性可以使用类名调用

```
class JustCounter:
    __secret_count = 0
    public_count = 0

    def count(self):
        self.__secret_count += 1
        self.public_count += 1
        print(self.__secret_count)


counter = JustCounter()
counter.count()
counter.count()
print(counter.public_count)
print(counter._JustCounter__secret_count)
```

- 私有方法

使用两个下划线开头，如__method()，来声明该方法为私有方法。

私有方法不能在类的外部被直接调用。

```
class MyClass:
    def __init__(self):
        self.public_method()

    def public_method(self):
        print("I am a public method")
        self.__private_method()

    def __private_method(self):
        print("I am a private method")

obj = MyClass()
# obj.__private_method()  # 不能直接调用私有方法，会报错
```

- 类的有限封装

Python的私有属性和方法的外部访问

类的外部外部可以通过：**_类名__属性名** 或者 **_类名__方法名** 来直接访问类中的私有属性和私有方法。

6. 类的继承

使用继承时要遵循"是一个"（is-a）关系，即子类应该是父类的一种特殊情况。

- 基类（父类）：被继承的类
- 派生类（子类）：继承的类
- 方法的重写

```
class ParentClass:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def display(self):
        print(f"x: {self.x}, y: {self.y}")

class ChildClass(ParentClass):
    def __init__(self, x, y, z):
        # 调用父类的构造方法
        super().__init__(x, y)
        self.z = z
```

```
14
15      # 重写父类的display方法
16      def display(self):
17          print(f"x: {self.x}, y: {self.y}, z: {self.z}")
18
19  # 创建子类的实例
20  child_obj = ChildClass(1, 2, 3)
21
22  # 调用子类的display方法，会先调用子类的方法，然后调用父类的方法
23  child_obj.display()
```

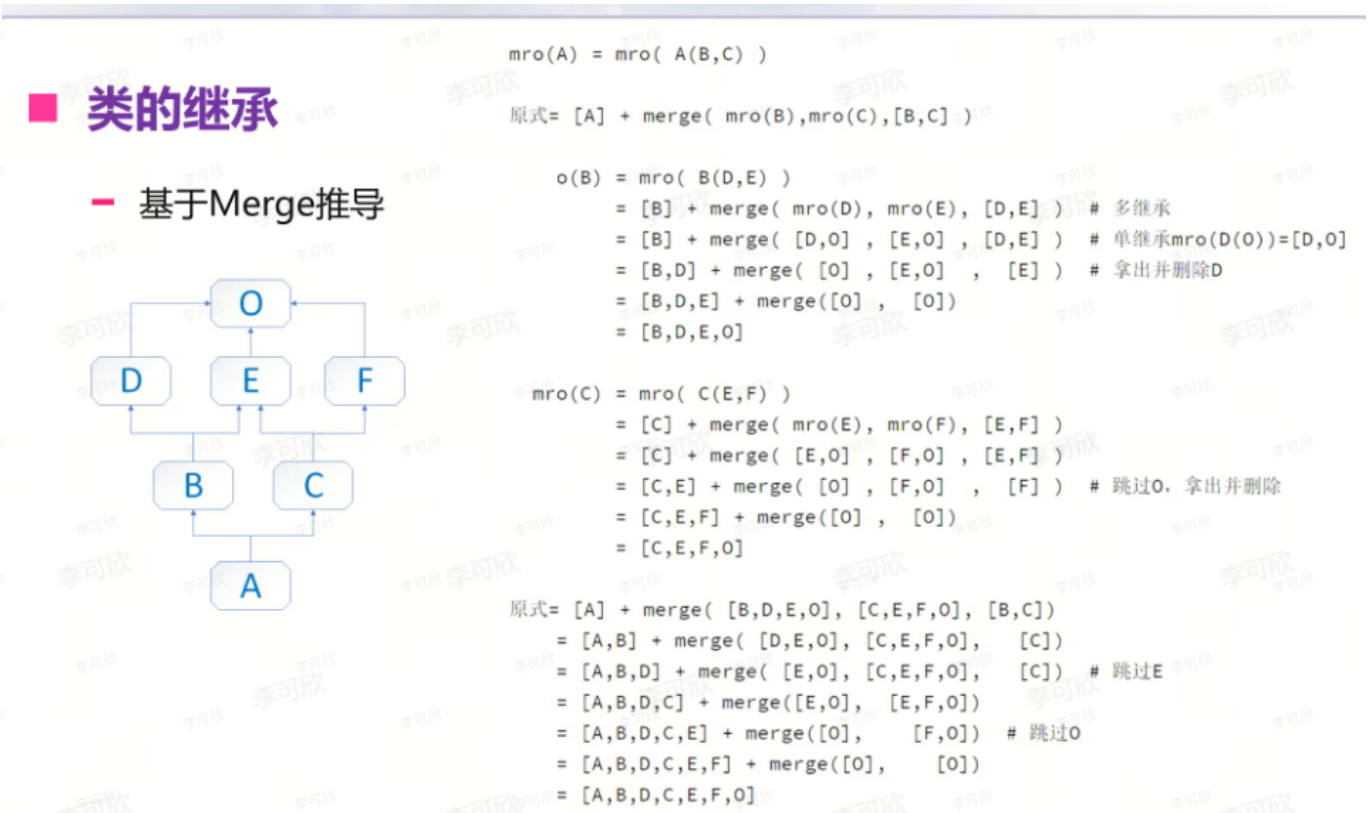
- 多继承

Python支持多重继承，即一个类可以继承多个父类，但要谨慎使用以避免复杂性。

MRO

MRO（Method Resolution Order，方法解析顺序）是指在多重继承中确定方法调用顺序的规则。在Python中，每个类都有一个特定的MRO，它影响了方法查找的顺序。

C3算法：基于Merge推导




- super()

子类中定义了与父类同名的方法，则子类的方法会覆盖父类方法。

super()实现了对父类方法的重写，即父类方法原有的功能保持不变，通过在子类中定义与父类同名的方法，增加新的或者修改父类原有的功能。

super()和类均可调用父类实例方法，区别在于super()后跟的方法不需要传self参数，父类调用实例方法，第一个参数需要传self(方法的绑定)。

```python
class Animal:
    def __init__(self, food):
        self.food = food

    def eat(self):
        print(f"They love {self.food}.")


class Bird(Animal):
    def __init__(self, name):
        self.name = name

    def swim(self):
        print(f"{self.name} swims fast.")
```

```python
class Penguin(Bird):
    def __init__(self, name, food):
        super().__init__(name)
        Animal.__init__(self, food)


peggy = Penguin("XB", "fish")
peggy.swim()
peggy.eat()
```

```
XB swims fast.
They love fish.
```

- Mixin类（接口）

Mixin是一种设计模式。

Python语法上无接口一说。

## Mixin类实现多继承需遵循的原则

- 责任明确：必须表示某一种功能，而不是某个物品(Python 对于Mixin类的命名方式一般以 Mixin, able, ible 为后缀)；
- 功能单一：若有多个功能，则添加多个Mixin类；
- 绝对独立：不能依赖于子类的实现，子类即便没有继承这个Mixin类，也照样可以工作，只是缺少了某个功能。

```
class Vehicle:  # 交通工具
    pass


class FlyableMixin:
    def fly(self):
        # 飞行功能相应的代码
        print("I am flying", self)


class CivilAircraft(FlyableMixin, Vehicle):  # 民航飞机
    pass


class Helicopter(FlyableMixin, Vehicle):  # 直升飞机
    pass


class Car(Vehicle):  # 汽车
    pass
```

7. 类的多态

Python中，不同的对象调用同一个接口(同名的方法)，从而表现出不同状态。

多态发生的条件：

继承：发生在父类和子类间

重写：子类重写父类方法

```
class Animal:
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

# 函数接受Animal类型的参数，但可以传入不同子类的对象
def make_sound(animal):
    return animal.sound()
```

```
16
17 # 使用多态性调用不同子类的方法
18 dog = Dog()
19 cat = Cat()
20
21 print(make_sound(dog))  # 输出: Woof!
22 print(make_sound(cat))  # 输出: Meow!
23
```

8. 抽象基类ABC

**包含抽象方法的类，是特殊类，只能被继承，不能被实例化，且子类必须实现抽象方法**

9. 判断类实例的函数

type()和isinstance()函数

- isinstance(obj,cls)：判断obj是否是cls类或者其子类的实例(考虑继承关系)。
- type(obj)：获取实例obj的类型，不考虑继承关系。

10. 元类metaclass

元类是创建类的类。

默认的元类：type

- 要创建一个class对象，type()函数依次传入3个参数：
  - class的名称
  - class继承的父类元组，如只有一个父类，注意tuple单元素写法
  - class属性、方法构成的字典

```
def func(self): pass

Foo = type("Foo", (object,), {"count": 0, "func": func})
```

可省略object类
不能省略空元组