学习目标

1. 了解面向对象三大特点的概念;
2. 掌握继承语法;
3. 掌握方法重写;
4. 掌握类属性和类方法的使用;

目录

第 1 章 面向对象进阶------三大特点

一、面向对象程序设计三大特性：

1. 封装------根据职责将属性和方法封装到一个抽象的类中 ；

2. 继承------实现代码的重用，相同的代码不需要重复的编写 ；

3. 多态------不同的对象调用相同的方法，产生不同的执行结果，增加代码的灵活度 。

第 2 章 面向对象进阶------封装

一、类的私有属性和私有方法

- 私有属性就是对象不希望公开的属性 ；
- 私有方法就是对象不希望公开的方法 。

1. 定义方式

在定义属性或方法时，在属性名或者方法名前 增加两个下划线，定义的就是私有属性或方法。

对于私有属性和私有方法，只能在类的内部访问，类的外部无法访问。

```
class woman:
    def __init__(self, weight = 0, age = 0, name = ""):
        # __weight 为私有属性
```

```
        self.__weight = weight
        # __age 为私有属性
        self.__age = age
        self.name = name

    # __secret 为私有方法
    def __secret(self):
        print("我的体重是%d，年龄是%d" % (self.__weight,
self.__age))

    def show_secret(self):
        self.__secret()
```

2. 课堂练习---

设计一个类 user，属性和方法如下：

- 属性：

  name：姓名

- 方法：

  show_name(self)

- 私有属性：

  __passwd：密码

- 私有方法：

  def __show_passwd(self)

第 3 章 面向对象进阶------继承

一、继承的概念、语法和特点

继承的概念：子类拥有父类的所有方法和属性。

二、继承的语法

class 类名(父类名):

pass

子类继承自父类，可以直接享受父类中已经封装好的方法，不需要再次开发

子类中应该根据职责，封装子类特有的属性和方法。

```
class animal:
    def sleep(self):
        print("睡")

    def eat(self):
        print("吃")

class dog(animal):
    def run(self):
        print("跑")
```

三、专业术语

dog 类是 animal 类的**子类**，animal 类是 dog 类的**父类**，dog 类从 animal 类**继承**

dog 类是 animal 类的**派生类**， animal 类是 dog 类的**基类**， dog 类从 animal 类**派生**

```
class animal:
```

```
    def sleep(self):
        print("睡")

    def eat(self):
        print("吃")

class dog(animal):
    def run(self):
        print("跑")

class fish(animal):
    def swimming(self):
        print("游水")

class bird(animal):
    def fly(self):
        print("飞")
```

四、继承的传递性

- C 类从 B 类继承，B 类又从 A 类继承；
- 那么 C 类就具有 B 类和 A 类的所有属性和方法；
- 子类拥有父类以及父类的父类 中封装的所有属性和方法 。

```
class animal:
    def sleep(self):
        print("睡")

    def eat(self):
        print("吃")

class dog(animal):
```

```
    def run(self):
        print("跑")

class erha(dog):
    def kawayi(self):
        print("萌萌嘀")
```

五、方法的重写

当父类的方法实现不能满足子类需求时，可以对方法进行 重写(override)。

重写父类方法有两种情况：

1. 覆盖父类的方法 ；

2. 对父类方法进行扩展 。

1. 覆盖父类的方法

如果在开发中，父类的方法实现和子类的方法实现，完全不同 ，就可以使用覆盖的方式，在子类中重新编写父类的方法实现 。

具体的实现方式，就相当于在子类中定义了一个 和父类同名的方法并且实现 。

重写之后，在运行时，只会调用子类中重写的方法，而不再会调用父类封装的方法。

```
class animal:
    def sleep(self):
        print("睡")

    def eat(self):
        print("吃")

class dog(animal):
    def run(self):
        print("跑")
    # 覆盖了父类的同名方法
    def eat(self):
        print("吃肉")
```

2. 对父类方法进行扩展

如果在开发中，既要使用父类的方法，又想增加功能， 就可以使用扩展的方式 。

1. 在子类中重写父类的方法 ；

2. 在需要的位置使用 super().父类方法来调用父类方法的执行 ；

3. 代码其他的位置针对子类的需求，编写子类特有的代码实现 。

```
class animal:
    def sleep(self):
        print("睡")

    def eat(self):
        print("吃")
```

```
class dog(animal):
    def run(self):
        print("跑")
    def eat(self):
        print("吃肉")
    # 对父类的 sleep 方法进行了扩展
    def sleep(self):
        super().sleep()
        print("睡的更多")
```

六、父类的私有属性和私有方法

1. 子类对象不能在自己的方法内部，直接访问父类的私有属性或私有方法；

2. 子类对象 可以通过父类的公有方法间接访问到私有属性或私有方法；

私有属性、方法是对象的隐私，不对外公开，外界以及子类都不能直接访问

私有属性、方法通常用于做一些内部的事情 。

1. 课堂练习---

- 实现父亲类 father

属性	说明
__name	姓名：私有属性，不能继承
house	房产：可以继承的属性
方法	**说明**
eat	吃：可以继承的方法
sleep	睡：可以继承的方法
__edu_back	学历：不可以继承的方法

- 实现儿子类 son，继承自 father 类

方法	说明
show_eat	调用 father 类的 eat 方法
show_sleep	调用 father 类的 sleep 方法
show_house	显示 father 类的 house 属性

七、object 类

在 Python 3 中定义类时，如果没有指定父类，会默认使用 object 作为该类的基类 —— Python 3 中定义的类都是新式类 。

在 Python 2 中定义类时，如果没有指定父类，则不会以 object 作为基类 。

新式类和经典类在多继承时 —— 会影响到方法的搜索顺序 。

为了保证编写的代码能够同时在 Python 2 和 Python 3 运行！ 今后在定义类时，如果没有父类,建议统一继承自 object。

class 类名(object):

pass

```
class animal(object):
    def sleep(self):
        print("睡")

    def eat(self):
        print("吃")

class dog(animal):
    def run(self):
```

```
        print("跑")
```

第 4 章 面向对象进阶------多态

- 不同的子类对象调用相同的父类方法，产生不同的执行结果

```
class animal(object):
    def food(self):
        pass

    def eat(self):
        self.food()

class dog(animal):
    def food(self):

        print("肉")


class cattle(animal):
    def food(self):

        print("草")


d = dog()
# 调用父类的 eat 方法
d.eat()

c = cattle()
# 调用父类的 eat 方法
c.eat()
```

第 5 章 面向对象进阶------类属性和类方法

不需要创建类的对象，通过 类名. 的方式就可以访问类的属性或者调用类的

方法 。

一、类属性

```
class A(object):
    # name 为类属性，通过 A.name 访问
    name = "tom"
    def __init__(self):
        # 属性 age 通过对象访问
        self.age = 20

print(A.name)
a = A()
print(a.age)
```

二、类方法

- 用@classmethod 修饰的方法为类方法;
- 类方法的参数为 cls，在类方法内部通过 **cls.类属性** 或者 **cls.类方法** 来访问同一个类中的其他类属性和类方法;
- 类方法不需要实例化就可以调用，类方法只能访问同一个类中的类属性和类方法。

```
class A(object):
    # name 为类属性
    name = "tom"
    def __init__(self):
        # 属性 age 只能通过对象访问
        self.age = 20

    # show_name 为类方法
    @classmethod
    def show_name(cls):
        print(cls.name)

A.show_name()
```

三、普通方法访问类属性或者类方法

- 在普通方法中通过 **类名.类属性** 或者 **类名.类方法** 来访问类属性和类方法。

```
class A(object):
    # name 为类属性
    name = "tom"
    # show_name 为类方法
    @classmethod
    def show_name(cls):
        print(cls.name)

    # set_name 为普通方法
    def set_name(self, name):
        A.name = name


A.show_name()
a = A()
a.set_name("mary")
A.show_name()
```

1. 课堂练习---

定义一个类 my_class，实现一个类方法 count(cls),调用 count 方法可以显示自己被实例化了几次，如：

a = my_class()
b = my_class()

my_class.count() # 显示 2

c = my_class()

my_class.count() # 显示 3

```
class my_class:
    index = 0
    def __init__(self):
```

```
        my_class.index += 1

    @classmethod
    def count(cls):
        print(cls.index)

a = my_class()
b = my_class()
c = my_class()
my_class.count()
```

第 6 章 面向对象进阶-----静态方法

如果需要在类中封装一个方法，这个方法既不需要访问实例属性 或者调用实例方法也不需要访问类属性或者调用类方法，这个时候，可以把这个方法封装成一个静态方法。

- 用@staticmethod 修饰的方法为静态方法；
- 静态方法是独立存在的，不能访问类或者实例的任何属性和方法；
- 通过 **类名.静态方法** 调用静态方法 。

语法如下

@staticmethod

def 静态方法名():

pass

```
class A(object):
    # show_help 为类的静态方法
    @staticmethod
    def show_help():

        print("静态方法")
```

```
A.show_help()
```