项目9　python面向对象程序设计

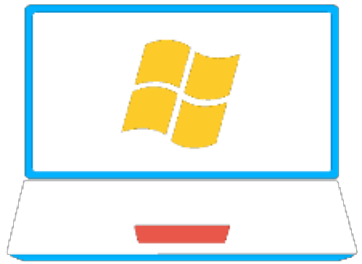
中国人民大学出版社
CHINA RENMIN UNIVERSITY PRESS

9.1 面向对象的编程思想

9.2 类与对象

9.3 私有成员、公有成员和数据成员

9.4 方法、属性、面向对象的三大特征



9.1面向对象的编程思想

1 类

2 对象

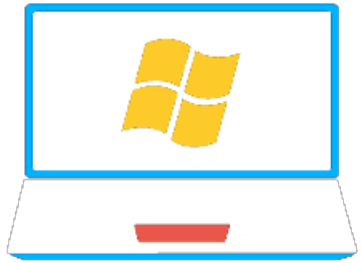
3 属性

4 方法

5 封装

6 继承

7 多态



9.1面向对象的编程思想

Python是一种面向对象的编程语言，它支持面向对象的编程思想。面向对象编程（OOP）是一种编程范式，它将数据和操作封装到对象中，通过对象之间的交互来实现程序的设计和实现。

- 类（Class）：类是对象的蓝图或模板，定义了对象的属性和方法。可以将类看作是创建对象的工厂。在Python中，使用class关键字来定义类。
- 对象（Object）：对象是类的实例，通过类的构造函数创建。每个对象都有自己的状态（属性）和行为（方法）。可以通过访问对象的属性和调用对象的方法来操作对象。
- 属性（Attribute）：属性是对象的状态，用于存储数据。可以将属性视为类的变量，每个对象都有自己的属性值。可以通过点操作符（.）来访问对象的属性。

9.1面向对象的编程思想

- 方法（Method）：方法是对象的行为，用于执行特定的操作。方法定义在类中，并且可以访问类的属性。方法可以通过点操作符调用。
- 封装（Encapsulation）：封装是将数据和操作封装到对象中的过程。通过封装，对象的内部细节对外部是隐藏的，只能通过对象的接口进行访问。这样可以提高代码的可维护性和安全性。
- 继承（Inheritance）：继承是一种机制，允许创建一个新的类（子类），从现有的类（父类）继承属性和方法。子类可以重用父类的代码，并且可以添加自己的新属性和方法。继承可以实现代码的重用和扩展。
- 多态（Polymorphism）：多态是指对象可以根据上下文表现出不同的行为。不同的对象可以对同一个方法有不同的实现。多态可以提高代码的灵活性和可扩展性。

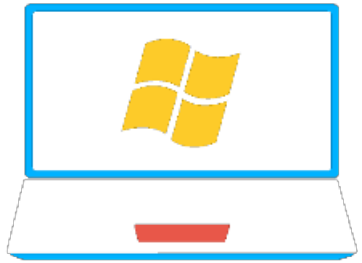
9.2类与对象

9.2.1 类

9.2.2 方法和私有化

9.2.3 类代码块

9.2.4 类的继承



9.2 类与对象

9.2.1 类

Python的类是面向对象编程的核心概念之一，提供了一种组织和管理代码的方式。要学习面向对象，首先要学会如何去创建类，Python中使用 class 关键字来定义类，通过以下例子说明类是如何创建的。

下面代码将展示如何创建一个类，并利用其创建两个对象，并调用其中的方法。具体实现代码如下：

9.2 类与对象

9.2.1 类

```
#创建一个person类
class Person:
#定义了类的构造函数 __init__
    def __init__(self, name):
#将构造函数中传入的 name 参数赋值给对象的 name 属性
        self.name = name
    #定义了一个名为 say_hello 的方法
    def say_hello(self):
#通过 self.name，我们可以访问对象的 name 属性
        print("Hello, my name is", self.name)
# 创建两个Person对象
person1 = Person("Alice")
person2 = Person("Bob")
# 调用对象的方法
person1.say_hello()
person2.say_hello()
```

9.2 类与对象

9.2.1 类

该程序的运行结果为：

```
Hello, my name is Alice
Hello, my name is Bob
```

9.2 类与对象

9.2.1 类

通过以上例子，可了解关于python类的相关知识：

- python中，类使用class来定义，类名跟在class后面。
- 类其实是代码块，后面要紧跟冒号（：）
- 类的方法本质是函数，因函数定义在类的内部，所以将类内部的函数称为方法。
- 每个方法至少制定一个self参数，且定义的方法通常使用 self 作为第一个参数，表示对当前对象的引用。在调用方法时，系统会自动将对象传入该参数。
- 使用类创建对象与函数的调用类似。
- 调用对象可直接通过对象变量调用，也可通过类调用方法。

9.2 类与对象

9.2.2 方法和私有化

一般来说，python所有方法都可被外部访问，但很多其他语言如C、C++、Java等，没有类的概念，只能通过提供特定关键字private实现方法的私有化，在Python中，可以通过在方法名称前加上两个下划线 __ 来将方法私有化。这样的方法被称为私有方法，它们只能在类的内部访问，无法从类的外部直接访问。

下列演示了一个类中同时存在可在外部访问的公共方法和无法在外部访问的私有方法的情况，具体实现代码如下：

9.2 类与对象

9.2.2 方法和私有化

```
class MyClass:
    def public_method(self):
        print("This is a public method")
    def __private_method(self):
        print("This is a private method")
obj = MyClass()
# 从外部访问公共方法
obj.public_method()
# 尝试从外部访问私有方法
obj.__private_method()
```

9.2 类与对象

9.2.2 方法和私有化

在上述代码中，MyClass 类定义了一个公共方法 public_method 和一个私有方法 __private_method。因为公共方法可以从类的外部直接访问，所以可以通过对象 obj 访问公共方法 public_method()；由于私有方法不应该直接从类的外部访问，所以当尝试通过对象 obj 访问私有方法 __private_method()时，会抛出 AttributeError 异常，提示该对象没有名为 __private_method 的属性。

9.2 类与对象

9.2.3 类代码块

代码块是一组相关的代码语句，它们被包含在一对花括号（如{}）或者缩进的空白行中，用于将多个语句组织在一起，并在程序执行时作为一个整体执行，代码块可以包含其他代码块，形成嵌套的层次结构。因此，class、for、while语句都属于代码块，因此定义类就是执行代码块的过程。示例代码如下所示：

```
class MyClass:
    print("MyClass")
```

9.2 类与对象

9.2.3 类代码块

下列代码定义了一个名为 MyClass 的类，其中包含一个类变量 count 和一个实例方法 counter。通过创建 MyClass 的实例对象，我们可以调用方法来增加并修改 count 的值。此外，还展示了如何动态地向对象添加新的变量。具体实现代码如下：

```
#定义了一个名为 MyClass 的类
class MyClass:
#在类定义中添加了一个 print 语句，该语句在定义类时执行
    print("MyClass")
    count=0
    def counter(self):
        self.count += 1
my=MyClass()
my.counter()              #调用了 my 对象的 counter 方法
```

9.2 类与对象

```
print(my.count)          #打印了 my 对象的 count 值
my.counter()             #再次调用了 my 对象的 counter 方法
print(my.count)          #打印了 my 对象的 count 值
my.count="abc"           #将 my 对象的 count 属性的值修改为字符串 "abc"
print(my.count)          #打印了 my 对象的 count 值，输出结果为“abc”
my.name="Hello"          #动态地向 my 对象添加了一个名为 name 的新属性，并将其值设置为字符串 "Hello"
print(my.name)           #打印了 my 对象的 name 值，输出结果为“Hello”
```

该程序的运行结果为：

```
MyClass
1
2
abc
Hello
```

9.2 类与对象

9.2.4 类的继承

在Python中，类的继承是一种重要的特性，类的继承是面向对象编程中的一个重要概念，它允许一个类（称为子类或派生类）从另一个类（称为父类或基类）继承属性和方法。继承是建立类层次结构、实现代码重用和提高代码可维护性的关键机制。在类的继承中，有父类（基类）和子类（派生类）的概念。父类是被继承的类，子类是继承父类的类。父类可以提供通用的属性和方法，而子类可以继承这些属性和方法，并可以在子类中添加新的属性和方法。

9.2 类与对象

9.2.4 类的继承

在Python中，需要在子类定义时将父类放在类名的圆括号中以指定继承关系。示例代码如下所示：

```
# 父类
class Filter:
    def filter1(self):
        return 20
class MyFilter(Filter):
    def filter2(self):
        return 30
```

9.2 类与对象

9.2.4 类的继承

```
#父类
class FatherClass:
    def method1(self):
        print("好好学习")
#子类
class BabyClass(FatherClass):
    def method2(self):
        print("天天向上")
child=BabyClass()
child.method1()                    #调用父类的方法
child.method2()
```

该程序的运行结果为：

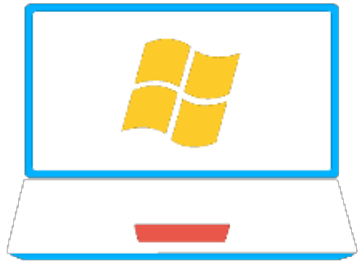
```
好好学习
天天向上
```

9.3 私有成员、公有成员和数据成员

9.3.1 私有成员

9.3.2 公有成员

9.3.3 数据成员



9.3 私有成员、公有成员和数据成员

9.3.1 私有成员

在Python中，私有成员是类定义中的一种成员，用于封装类的内部实现细节，并限制外部访问和修改。私有成员的命名约定是在成员名称前面添加两个下划线 __。以下是对私有成员的详细描述。

1.访问限制：私有成员只能在类的内部被访问和使用，对于类的外部是不可见的。当尝试通过类的实例直接访问私有成员时，会引发 AttributeError 异常。

9.3私有成员、公有成员和数据成员

9.3.1 私有成员

下列代码定义了一个类 MyClass，其中包含一个私有成员变量 __private_variable 和一个私有方法 __private_method，并尝试通过实例直接访问私有成员。具体实现代码如下：

```
class MyClass:
    def __init__(self):
        self.__private_variable = 42
    def __private_method(self):
        print("This is a private method")
my_instance = MyClass()
# 尝试直接访问私有成员变量
print(my_instance.__private_variable)
# 尝试直接调用私有方法
my_instance.__private_method()
```

该程序的运行结果为：

```
AttributeError: 'MyClass' object has no attribute '__private_variable'
```