第7讲 类和对象

王肖燕

内容回忆

一．函数定义和调用

- def, 圆括号, return

二．参数传递

- 形参
  - ✓ 是否有默认值（默认参数），个数是否确定（可变长度参数）
- 实参
  - ✓ 关键字形式（关键字参数），组合数据类型作实参（序列解包）

三．变量作用域

- 局部变量、全局变量

四．lambda函数

- 无名小函数

内容介绍

一．封装

二．什么是对象，对象=属性+方法

三．类→对象：实例化

四．进一步剖析Python的类和对象

五．继承和多态

六．举例

封装

- 数据层面的封装
  - 把乱七八糟的数据扔进列表里
- 语句层面的封装
  - 把常用的代码段打包成一个函数
- 数据和代码的封装
  - 对象

什么是对象

- **乌龟**是真实世界里的一个对象，如何描述这个对象
  - 静态特征：绿色的、有四条腿、10kg重、有外壳、还有个大嘴巴……
  - 动态行为：它会爬、如果追它就会跑、你把它逼急了它会咬人、被它咬了打雷之后才会松开嘴巴、会睡觉……
  - 人作为对象，怎么描述？

对象=属性+方法

- Python中的对象也是如此
  - 一个对象的特征称为属性
  - 一个对象的行为称为方法
- 把乌龟写成代码，如下所示：

对象=属性+方法

- 把乌龟写成代码，如下所示：

定义：

```
class Turtle:
    color='green'
    weight=10
    legs=4
    shell=True
    mouth='大嘴巴'
    def climb(self):
        print("我一直在努力地向前爬...")
    def run(self):
        print("我正在飞快地向前跑...")
    def bite(self):
        print("我咬死你，咬死你...")
    def eat(self):
        print("有得吃，真满足...")
    def sleep(self):
        print("困了，睡了，晚安...")
```

Python中**类名**约定以**大写字母开头**

特征的描述称为**属性**，在代码层面其实就是**变量**

方法实际就是**函数**，通过调用这些函数来完成某些工作

类→对象

- 以上代码定义了对象的特征（属性）和行为（方法），但还不是一个完整的对象，将定义的这些称为类（class）
- 需要使用类创建一个真正的对象，这个对象就叫作这个类的一个实例（Instance），也叫实例对象（Instance Objects）

对象

- 创建一个对象，也叫类的实例化
- >>> tt=Turtle()
- 类名后面跟着小括号，和调用函数一样
- Python中约定类名用大写字母开头，函数用小写字母开头
- 调用对象里的方法，使用点操作符（.）
  - >>> tt.climb()
  - >>> tt.bite()
  - >>> tt.sleep()

Self参数

① 如果把类比作是图纸，那么由类实例化后的对象才是真正可以住的房子

② 根据一张图纸可以设计出成千上万的房子，它们长得都差不多，但它们都有不同的主人

③ Self相当于每个房子的门牌号，通过Self可以轻松找到自己的房子

④ Python的Self参数就是同一个道理，由同一个类可以生成无数对象，当一个对象的方法被调用的时候，对象会将自身的引用作为第一个参数传给该方法，那么Python就知道需要操作此类的哪个对象的方法了

举例1

```
class Ball:
    def setName(self,name):
        self.name=name
    def kick(self):
        print("我叫"+self.name+", 谁踢我？")
```

```
>>> a = Ball()
>>> a.setName("王小二")
>>> b = Ball()
>>> b.setName("张小五")
>>> a.kick()
我叫王小二，谁踢我？
>>> b.kick()
我叫张小五，谁踢我？
```

当一个对象的方法被调用时，对象会将自身的引用作为第一个参数传递给该方法，那么Python就知道要操作哪个对象的方法了

魔法方法

- Python对象天生拥有一些神奇的方法,可以给类增加魔力的特殊方法
- 如果写的对象实现了这些方法中的某一个，那么这个方法就会在特殊的情况下被python所调用
- 这一切都是自动发生的
- 魔法方法特点
  - 双下划线包围

魔法方法__init__()

- __init__()构造方法
  - 只要实例化一个对象，这个方法就会在对象被创建时自动调用
  - 实例化对象时可以传入参数，这些参数会自动传入__init__()方法中
  - 可以重写这个方法来自定义对象的初始化操作

魔法方法

```
class Turtle:
    color='green'
    weight=10
    legs=4
    shell=True
    mouth='大嘴巴'
    def __init__(self,legs):
        self.legs = legs
    def climb(self):
        print("我一直在努力地向前爬...")
    def run(self):
        print("我正在飞快地向前跑...")
    def bite(self):
        print("我咬死你，咬死你...")
    def eat(self):
        print("有得吃，真满足...")
    def sleep(self):
        print("困了，睡了，晚安...")
```

```
>>> tt=Turtle()
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    tt=Turtle()
TypeError: __init__() missing 1 required positional
>>> tt=Turtle(5)
>>> tt.legs
```

私有成员和公有成员

- 形式上
  - 成员名以两个下划线（__）开头则表示是私有成员，其在对象的外部不能直接访问，一般是在对象的内部进行访问和操作

```
class A:
    value1=2
    _value2=3
    __value3=4
    def show(self):
        print(self.value1)
        print(self._value2)
        print(self.__value3)
```

```
>>> a=A()
>>> a.value1
2
>>> a._value2
3
>>> a.__value3
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    a.__value3
AttributeError: 'A' object has no attribute
>>> a.show()
2
3
4
```

- 封装离不开私有化，将某些属性限制在某个区域内

私有成员的操作

```
class Test:
    def __init__(self,v1,v2):
        self.__value1 = v1
        self.__value2 = v2
    def __getV1(self):
        return self.__value1
    def __setV1(self,v):
        self.__value1 = v
    def __getV2(self):
        return self.__value2
    def __setV2(self,v):
        self.__value2 = v
    def show(self):
        print(self.__getV1()+self.__getV2())
```

```
>>> t=Test(3,4)
>>> t.show()
7
```

属性私有化

- 定义公有set和get方法

```
 8 class Person:
 9     __slots__=['__name','__flag']
10     def __init__(self, name):
11         self.__name=name
12         self.__flag=False
13     def get_name(self):
14         if self.__flag:
15             return self.__name
16         else:
17             print("没有权限查看用户名")
18     def set_name(self,name):
19         if len(name)>=6:
20             self.__name=name
21         else:
22             print("名字必须要大于等于6位")
```

```
p1=Person("Jessica")
p1.name="jerry"
```

方法装饰器

```
27 #装饰器@property,@name.setter
28 class Person:
29     __slots__=['__name','__flag']
30     def __init__(self, name):
31         self.__name=name
32         self.__flag=True
33 
34     @property
35     def name(self):
36         if self.__flag:
37             return self.__name
38         else:
39             print("没有权限查看用户名")
40     @name.setter
41     def name(self,name):
42         if len(name)>=6:
43             self.__name=name
44         else:
45             print("名字必须要大于等于6位")
```

```
p1=Person("Jessica")
print(p1.name)
p1.name="jerryZhao"
print(p1.name)
```

继承

① 对鱼类细分：金鱼、鲤鱼、三文鱼、鲨鱼

② 能不能不要每次都从头到尾重新定义一个新的鱼类，因为大部分鱼的属性和方法是相似的

③ 如果有一种机制可以让这些相似的东西得以自动传递，会方便快捷很多

④ 继承机制

继承

- **语法**
  - Class 类名(被继承的类):

    ……
- 被继承的类称为基类、父类或超类
- 继承者称为子类
- 一个子类可以继承它的父类的任何属性和方法

举例2

```
class Parent:
    def hello(self):
        print("正在调用父类的方法...")

class Child(Parent):
    pass
```

```
>>> p=Parent()
>>> p.hello()
正在调用父类的方法...
>>> c=Child()
>>> c.hello()
正在调用父类的方法...
```

如果子类中定义与父类同名的方法或属性，则会自动覆盖父类对应的方法或属性。

```
class Parent:
    def hello(self):
        print("正在调用父类的方法...")

class Child(Parent):
    def hello(self):
        print("正在调用子类的方法...")
```

```
>>> p=Parent()
>>> p.hello()
正在调用父类的方法...
>>> c=Child()
>>> c.hello()
正在调用子类的方法...
```

多态

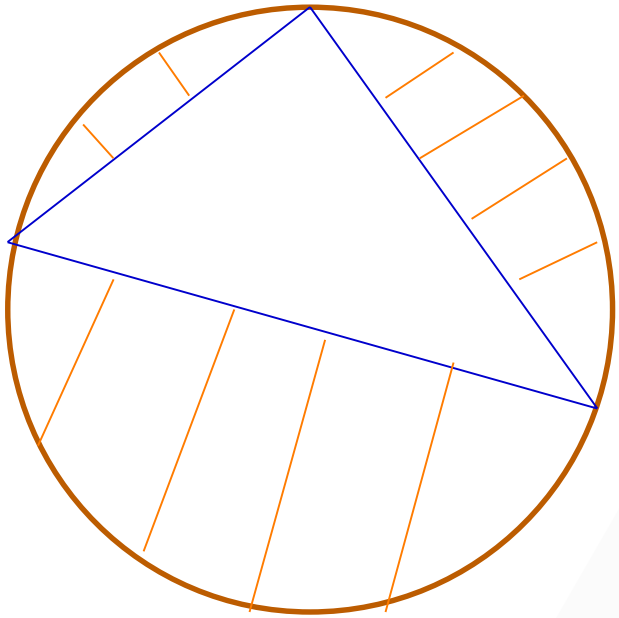
- 同一种行为具有不同表现形式和形态的能力，即是对象多种表现形式的体现
  - Count()的作用是数一数某个元素在对象中出现的次数，但并没有限定其参数类型：
    - >>> “This is a book”.count(‘s’)
    - >>> [1,2,3,4,5,3].count(3)
  - f=lambda x,y:x+y
    - >>> f(2,3)
    - >>> f(“qiw”,”sir”)
    - >>> f([“abc”,”de”],[“fgh”,”ijk”])
  - 前提是可以进行上述操作的数据类型被传入

举例

```
 8 class Animal:
 9     def talk(self):
10         pass
11
12 class Cat(Animal):
13     def talk(self):
14         print("say miaomiao")
15
16 class Dog(Animal):
17     def talk(self):
18         print("say wangwang")
19
20 class Pig(Animal):
21     def talk(self):
22         print("say aoao")
```

练习1

- 编写圆类和三角形类，通过圆对象和三角形对象中的计算面积的方法来计算半径为12cm的圆中三角形之外的圆的面积，其中的内接三角形的三边为8，24，17.



练习2

- 定义Person类，有属性姓名、年龄、性别和身高，通过构造方法进行初始化。方法有跑、唱和学习。
- 定义Student类，继承Person类。重写学习方法。

练习3

- 编写Person类，私有化它的属性name, age, password, phone, 使用公有化方法读取这些属性。限制条件：
  - 登录成功才能读取和修改属性
  - name必须是数字开头，字母结尾
  - age必做是合理的岁数
  - phone电话号码为11位手机号
  - 除上述属性外不能为对象定义其它公有属性

Thank You!