第8章
类和对象

CS, ZJU

2018年12月

Overview

- 类和对象的概念
- Python类的定义和使用对象
- 面向对象特性：封装、继承、多态

8.1类和对象的概念

- 面向对象程序设计（Object Oriented Programming，OOP）：使用对象进行程序设计，实现代码复用和设计复用，使得软件开发更高效方便。
- 对象三个要素：（id,type,value）
- 面向过程程序设计
- 面向函数程序设计

Python是面向对象语言

- Python是面向对象的高级动态编程语言，完全支持面向对象的基本功能，如封装、继承、多态以及对基类方法的覆盖或重写。
- Python中对象的概念很广泛，Python中的一切内容都是对象，除了数值、字符串、列表、元组、字典、集合、range对象等等，函数也是对象，类也是对象。

内置类（类型）和对象

例如：字符串类和字符串对象

```
>>> s1=str(123)
>>> s2='abc'
>>> print(s1+'  '+s2)
123  abc
>>> s1.islower()
False
>>> s2.isnumeric()
False
```

已学的类（类型）

- 类（类型）：

  int、 float 、bool 、string

  list 、tuple 、dict 、set

- 对象举例：

  3，[5.7,’a’]，{1:4,2:5}

  type函数判断对象的类

类的名字空间

- 每个类有自己的名字空间，类名是空间名
- 每个对象也有自己的名字空间
- 存储在类名称空间中的名称是类的成员
- 类成员包含
- 类属性
- 类方法
- 右边以list类为例

- >>> dir(list)　#显示类成员
- ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
- 。。。。。。
- 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

- #使用list作为名字空间访问
- >>> list.sort
- <method 'sort' of 'list' objects>

类方法是类名字空间中定义的函数

- lst=[4,97,7,34,-1]
- lst.sort()
- 相当于：
- list.sort(lst)
- list是lst的类名

- 字典 d.keys()
- 相当于：
- dict.keys(d)

- instance.method(…)
- 相当于：
- class.method(instance,…)

- class是instance的类型

对象继承类的方法

- >>> dir(list)
- ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
- >>> lst=[4,97,7,34,-1]　　#lst是list的对象
- >>> dir(lst)
- ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
- 不同对象一般有相同的方法，但属性可以不同

类和对象概念小结

- 类(class)：定义属性(数据)和行为（方法）的模板 ，
- 实例(instance)：是用类产生的对象，
- 术语对象(object)和实例(instance)经常是可以互换的。
- 使用圆点运算符(.)引用方法和属性
- 类有自己名字空间
- 每个对象也有自己的名字空间
- 同义词：

对象	实例	实例对象
类	类型	类对象
属性	变量	
方法	操作	函数

8.2类和对象的创建和使用

- Python使用class关键字来定义类，class关键字之后是一个空格，然后是类的名字，再然后是一个冒号，最后换行并定义类的内部实现。
  Python定义类方法：

  class 类名(参数) :　　def　函数名(参数):

  　　initializer　　函数块

  　　methods

Student类和对象（1）

```
class Student:     #学生类：包含成员变量和成员方法
        def __init__ (self,mname,mnumber):  #构造方法
                self.name = mname       #成员变量
                self.number = mnumber
                self.Course_Grade = {}
                self.GPA = 0
        def getInfo(self):                  #成员方法
            print(self.name,self.number)
s1 = Student("wang","317000010")   #创建s1对象
s1.getInfo()
#Student.getInfo(s1）
s2=Student("zhang","317000011")         #创建s2对象
s2.getInfo()
```

Student类和s1对象的名字空间

- >>> dir(Student)
- ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'getInfo']
- >>> dir(s1)
- ['Course_Grade', 'GPA', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'getInfo', 'name', 'number']

Student类和s1对象

- 上述程序定义Student类:
  - 两个方法：__init__，getInfo。
  - 实例化了s1与s2两个对象。
  - **该类对象包括四个成员变量：**name，number，Course_Grade，GPA；

__init__方法

- __init__方法是Python类中的一种特殊方法，方法名的开始和结束都是双下划线，该方法称为构造方法，当创建类的对象时，它被自动调用。
- __init__方法中可以声明类所产生的对象属性，并可为其赋初始值。该方法有一个特点，不能有返回值，因为它是用来构造对象的，调用后实例化了一个该类型的对象。

Self参数

- 类的实例方法按惯例有一个名为self的参数，并且必须是方法的第一个形参（如果有多个形参的话），self参数代表将来要创建的对象本身。
- 在类的方法中访问实例变量（数据成员）时需要以self为前缀。
- 在外部通过对象调用对象方法时并不需要传递这个参数，如果在外部通过类调用对象方法则需要显式为self参数传值。

创建对象

- 定义了类之后，可以用来实例化对象，并通过“对象名.成员”的方式来访问其中的数据成员或方法成员。

```
s1 = Student(“ji","100001")
s1.getInfo()
```

- 在Python中，可以使用内置方法isinstance()来测试一个对象是否为某个类的实例。

```
>>> isinstance(s1, Student)
True
>>> isinstance(s1, str)
False
```

8.3 使用对象编写程序

例8-1：计算身体质量指数BMI

BMI是根据体重和身高来衡量健康

一种方法。通过以千克为单位的体重除以以米为单位的身高的平方计算出BMI。下面是中国16岁以上人群的BMI图表：

构造函数

```
class BMI:
    def __init__(self, name, age, weight, height):
 #构造方法，创建对象时调用
        self.__name = name
        self.__age = age
        self.__weight = weight
        self.__height = height

    def getBMI(self):  #计算BMI
        bmi = self.__weight  /  (self.__height *
                 self.__height )
        return round(bmi * 100) / 100
```

getStatus(self)方法

```
def getStatus(self):
        bmi = self.getBMI()
        if bmi < 18.5:
            return "偏瘦"
        elif bmi < 24:
            return "正常"
        elif bmi < 30:
            return "偏胖"
        else:
            return "肥胖"
```

其他方法

- def getName(self):
- 	return self.__name
-
- def getAge(self):
- 	return self.__age
-
- def getWeight(self):
- 	return self.__weight
-
- def getHeight(self):
- 	return self.__height

创建对象

```
def main():
    bmi1 = BMI("赵四", 18, 70, 1.75)
    print( bmi1.getName(), "的BMI是",
        bmi1.getBMI(), bmi1.getStatus())

    bmi2 = BMI("王超", 38, 75, 1.70)
    print( bmi2.getName(), "的BMI是",
        bmi2.getBMI(), bmi2.getStatus())

main() # Call the main function
```

设计向量类

- 设计一个三维向量类，并实现向量的**加法**、**减法**以及向量与标量的**乘法**和**除法**运算

向量类方法（1）

```
class Vecter3:
  def __init__(self, x=0, y=0, z=0):#构造方法
    self.X = x
    self.Y = y
    self.Z = z

  def __add__(self, n): #加法
    r = Vecter3()
    r.X = self.X + n.X
    r.Y = self.Y + n.Y
    r.Z = self.Z + n.Z
    return r
```

向量类方法（2）

```
def __sub__(self, n):   #减法
    r = Vecter3()
    r.X = self.X - n.X
    r.Y = self.Y - n.Y
    r.Z = self.Z - n.Z
    return r

def __mul__(self, n):  #乘法
    r = Vecter3()
    r.X = self.X * n
    r.Y = self.Y * n
    r.Z = self.Z * n
    return r
```

实例

```
def __truediv__(self, n):  #除
    r = Vecter3()
    r.X = self.X / n
    r.Y = self.Y / n
    r.Z = self.Z / n
    return r

def __floordiv__(self, n):  #整除
    r = Vecter3()
    r.X = self.X // n
    r.Y = self.Y // n
    r.Z = self.Z // n
   return r

def show(self):  #打印向量值
     print((self.X,self.Y,self.Z))
```

```
def add3(self):
        self.X = self.X + 3
        self.Y = self.Y + 3
        self.Z = self.Z + 3
v1 = Vecter3(1,2,3)
v2 = Vecter3(4,5,6)
v3 = v1+v2
v3.show()
v4 = v1-v2
v4.show()
v5 = v1*3
v5.show()
v6 = v1/2
v6.show()
v1.add3()
v1.show()
```

思考add3()方法与其他区别？

8.4 封装

- **封装**：
  - 将数据和对数据的操作组合起来构成类，类是一个不可分割的独立单位
  - 类中既要提供与外部联系的接口，同时又要尽可能隐藏类的实现细节。
  - Python类中成员分为数据（变量、属性）成员和方法（函数）成员。

方法和属性类型

- Python类中成员：
  - 数据成员（变量、属性）
    - 类数据成员
    - 实例数据成员
  - 方法（函数）
    - 实例方法：公有
      私有：方法名以下划线’__’开头
    - 类方法：　@classmethod
    - 静态方法：@staticmethod

Python社团约定

- 在Python中，以下划线开头的方法名和变量名有特殊的含义，尤其是在类的定义中。
  - _xxx：当做内部名，不应该在外部使用，
  - 不能用'from module import *'导入；
  - __xxx：解释器会换名，不能使用对象直接访问到这个成员。
  - __xxx__：系统定义的特殊成员；不要创建这种标识符

改名和创建新变量

```
class A:
    def __init__(self):
        self.x = 1
        self.__y = 1

    def getY(self):
        return self.__y

a = A()
a.__y = 45
print(a.getY())
```

类成员变量与实例成员变量

- **实例的成员变量**一般是指在构造方法__init__()中定义的，定义和使用时以self作为前缀
- **类的成员变量**是在类中所有方法之外定义的

- 特性：在主程序中（或类的外部），实例变量属于实例(对象)，只能通过对象名访问；而类变量属于类，可以通过类名或对象名可以访问。

类成员与实例成员

```
class Car:
    price = 100000              #定义类成员变量
    def __init__(self, c):
        self.color = c          #定义实例成员变量

car1 = Car("Red")               #实例化对象
car2 = Car("Blue")
print(car1.color, Car.price)    #查看实例变量和类变量的值
Car.price = 110000              #修改类变量
Car.name = 'QQ'                 #动态增加类变量
car1.color = "Yellow"           #修改实例变量
print(car2.color, Car.price, Car.name)
print(car1.color, Car.price, Car.name)
```

Car类和car1对象的名字空间

- >>> dir(Car)
- ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'price']
- >>> dir(car1)
- ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'color', 'price']

练习

```
class Car:
    price = 300000              #定义类变量

    def __init__(self, name):
        self.name=name          #定义实例变量
        self.color = ""     #存储汽车颜色

    def setColor(self,color):     #设置汽车的颜色
        self. color = color

car1 = Car("奥迪")              #创建car1对象
car2 = Car("宝马")              #创建car2对象
print(car1.name, Car.price)         #打印实例变量和类变量的值
Car.price = 310000              #修改类变量
car1.setColor ( '黑色')          #car1的汽车型号为QQ
car1.name = "新奥迪"              #修改实例变量
print(car1.name, Car.price, car1.color)
print(car2.name, Car.price, car2.color)
```

类方法（class method）

- 类方法会作用于整个类，对类作出的任何改变会对它的所有实例对象产生影响。
- #打印对象的个数
- class A():
- 　　count = 0
- 　　def __init__(self):
- 　　　　A. count += 1
- 　　def exclaim(self):
- 　　　　print("我是A!")
- 　　@classmethod
- 　　def kids(cls):　　　　#cls是class缩写
- 　　　print("A有",cls.count,"个对象")
- one = A()
- two = A()
- three= A()
- A.kids()

Pass语句作用

- Python提供了一个关键字“pass”，类似于空语句，可以用在类和函数的定义中或者选择结构中。当暂时没有确定如何实现功能，或者为以后的软件升级预留空间，或者其他类型功能时，可以使用该关键字来“占位”。

```
>>> class A:
        pass
>>> def demo():
        pass
>>> if 5>3:
        pass
```

成员的输入显示

- 在IDLE环境中，在对象或类名后面加上一个圆点“.”，稍等则会自动列出其所有公开成员，模块也具有同样的用法。
- 如果在圆点“.”后面再加一个**下划线**“_”，则会列出该对象、类或模块的所有成员，包括私有成员。

8.5类的继承和多态

- 一个类通过继承成新的类
- 扩展list类，增加随机选取功能
- import random
- class Mylist(list): #list是父类名
- 　　def choice(self):
- 　　　　return random.choice(self)
-
- lst=Mylist("good moring")
- print(lst.choice())

super函数

```
class Car():
    price = 300000              #定义类变量
    def __init__(self, name):
            self.name=name          #定义实例变量
            self.color = ""         #存储汽车颜色
    def setColor(self,color):           #设置汽车的颜色
            self. color = color
class ECar(Car):
      def __init__(self,name):
            super().__init__(name)   #初始化父类的属性
            self.battery_size = 300
      def getEcar(self):
            print("我是电动汽车"+self.name+"电瓶容量为"+
str(self.battery_size)
                  +"公里")

car=ECar("曹操专车")
car.getEcar()
```

多态

- **python的多态性是指具有不同功能的运算符（函数）可以使用相同的符号，这样就可以用一个符号调用不同的功能**

```
>>> 12+6
18
>>> '12'+'6'
'126'
>>>
```

运算符是方法，也是函数

- >>>2+3
- 5
- >>> int(3).__add__(2)
- 5
- >>> int.__add__(3,2)
- 5

运算符重载（1）

- 运算符重载是Python设计理念的核心之一

运算符	方法	数字	字符串和列表
a+b	a.__add__(b)	加法	拼接
a-b	a.__sub__(b)	减法	---------
a*b	a.__mul__(b)	乘法	自连接
a/b	a.__truediv__(b)	除法	------
a//b	a.__floordiv__(b)	整除	------
a&b	a.__mod__(b)	余数	------
a==b	a.__eq__(b)	等于	
a!=b	a.__ne__(b)	不等于	
a>b	a.__gt__(b)	大于	
a>=b	a.__ge__(b)	大于或等于	

运算符重载（2）

运算符	方法	数字	字符串
a<b	a.__lt__(b)	小于	
a<=b	a.__le__(b)	小于等于	
str(a)	a.__str__()	字符串表示形式	
len(a)	a.__len__()	-----	容器大小
type(a)	type.__init__(a)	构造函数	
[]（取值）	__getitem__()		
[]（赋值）	__setitem__()		