第8部分　面向对象入门

为什么要面向对象?

面向对象的基本概念

- 利用构建不同的“实例”来编程并完成任务。
  - 一个实例就是在现实世界中可以被区分为不同类别的主体. 例如，一个学生、一个桌子、一个圆、一个按钮或者一笔钱都可以称为一个“实例”.

- 描述一个实例通常需要两方面信息：属性和方法。属性也被称为“字段”或者“特征”，包含这个实例的基本信息，例如学生的年龄、圆的半径、一笔钱的金额等等；而方法则描述了这个实例的行为动作，比如对于一个字符串按句号隔开，用到了split方法。

- 这个实例的属性和方法由定义他的类来声明。例如描述学生需要用到年龄、
- 性别、专业等属性，和选课、退课等方法，都可以定义在Student类当中。

面向对象程序设计

- 面向对象程序设计

  ✓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。

- Python是面向对象的解释型高级动态编程语言，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

为什么要面向对象?


復旦大學
FUDAN UNIVERSITY
首页
校园资讯
一站式服务
生命周期
管理系统
快速链接
服务月历
服务导航
密码修改
窦一凡
服务搜索
重点推荐
年度考评
一卡通网银...
延聘申请
人事管理
一卡通查询
教务管理
干部年度考核
场馆预约
热门服务
教务管理
场馆预约
一卡通网银...
一卡通查询
东航折扣机票
离校服务管理
财务查询
年度考评
信息办本月重点业务支持
延聘申请
教职工出...
人事信息...
尚景园公...
eLearnin...
我的生活...
外购存货...
电子点名...
我的收藏
门户通知(1)
更多
门户通知
Portal中面向个人的服务，正在逐步迁往ehall，在ehall中注册的服务将同步在portal上注销，如需访问请前往ehall平台，http://ehall.fudan.edu.cn
(2016-10-12) 查看详情
日程安排
2017年 四月
一 二 三 四 五 六 日
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
2017年4月01日日程安排

1 类的定义与使用

- Python使用class关键字来定义类，class关键字之后是一个空格，接下来是类的名字，如果派生自其它基类的话则需要把所有基类放到一对圆括号中并使用逗号分隔，然后是一个冒号，最后换行并定义类的内部实现。

- 类名的首字母一般要大写，当然也可以按照自己的习惯定义类名，但是一般推荐参考惯例来命名，并在整个系统的设计和实现中保持风格一致，这一点对于团队合作非常重要。

```
class Car(object):              #定义一个类，派生自object类
    def infor(self):            #定义成员方法，而非普通函数（可以在类里定义函数，但函数不能被用作方法）
        print("This is a car")
```

1 类的定义与使用

■定义了类之后，就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成员方法。

```
>>> car = Car()          #实例化对象
>>> car.infor()          #调用对象的成员方法
This is a car
```

a. sorted() 和 sorted(a) 有什么差别?

对象里面带函数 vs 函数里面调用对象

```
>>> x = [1, 2, 3]
>>> x.append(4)          #在尾部追加元素
>>> x.insert(0, 0)       #在指定位置插入元素
>>> x.extend([5, 6, 7])  #在尾部追加多个元素
>>> x
[0, 1, 2, 3, 4, 5, 6, 7]
>>> x.pop()              #弹出并返回尾部元素
>>> x.clear()            #删除所有元素
```

```
fib(1000)

def fib(n):
    a, b = 1, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

1 类的定义与使用

■在Python中，可以使用内置函数isinstance()来测试一个对象是否为某个类的实例，或者使用内置函数type()查看对象类型。

```
>>> isinstance(car, Car)
True
>>> isinstance(car, str)
False
>>> type(car)
<class '__main__.Car'>
```

2 数据成员与成员方法

■创建类时用变量形式表示对象特征的成员称为数据成员（attribute），用函数形式表示对象行为的成员称为成员方法（method），数据成员和成员方法统称为类的成员。

```
class Car(object):          #定义一个类，派生自object类
    color='red'             #定义数据成员（属性，氪金买皮肤）
    def infor(self):        #定义成员方法
        print("This is a car")
```

2.1 私有成员与公有成员

- 私有成员在类的外部不能直接访问，一般是在类的内部进行访问和操作，或者在类的外部通过调用对象的公有成员方法来访问，而公有成员是可以公开使用的，既可以在类的内部进行访问，也可以在外部程序中使用。
  - ✓ __xxx：私有成员，只有类对象自己能访问，子类对象不能直接访问到这个成员
  - ✓ _xxx：非私有、受类保护成员，可以通过object._xxx在外部访问
  - ✓ __xxx__：系统定义的特殊成员，比如__init__为当进行实例化对象时系统默认执行，相当于自动执行的初始化工作；

2.1 私有成员与公有成员

```
>>> class A:
    def __init__(self, value1=0, value2=0):
        self._value1 = value1
        self.__value2 = value2
    def setValue(self, value1, value2):
        self._value1 = value1
        self.__value2 = value2
    def show(self):
        print(self._value1)
        print(self.__value2)

>>> a = A()    #__init__在将类A付给a时就被执行了，所以才有_value1的值
>>> a._value1
0
>>> a._A__value2
0
#在外部访问对象的私有数据成员（不支持的用法）
```

2.1 私有成员与公有成员

- 在Python中，以下划线开头的变量名和方法名有特殊的含义，尤其是在类的定义中。
  - ✓ _xxx：非私有、受类保护成员（类似函数的局部变量），可以在外部直接访问；
    - ✓ 没有_的变量也可以直接在类外调用（类似全局变量，甚至不用加对象名），不建议使用
  - ✓ __xxx：私有成员，只有类对象自己能访问，子类对象不能直接访问到这个成员
    - ✓ 但在对象外部可以通过“对象名._类名__xxx”这样的特殊方式来访问（这会破坏类的封装性的初始目的，不建议这样做。
    - ✓ 也就是说：Python中不存在严格意义上的私有成员（被产品开发诟病）
  - ✓ __xxx__：系统定义的特殊成员；

2.2 数据成员

- 数据成员可以大致分为两类：属于对象的数据成员和属于类的数据成员。
  - 属于对象的数据成员在定义和在实例方法中访问数据成员时以self作为前缀，同一个类的不同对象（实例）的数据成员之间互不影响
  - 属于类的数据成员是该类所有对象共享的，不属于任何一个对象，在定义类时这类数据成员一般不在任何一个成员方法的定义中。

- 在主程序中或类的外部，对象数据成员属于实例(对象)，只能通过对象名访问；而类数据成员属于类，可以通过类名或对象名访问。

```
# 演示类成员和变量的差异
class A:
    color="red"
    def __init__(self, value1=0, value2=0):
        self._value1 = value1
        self.__value2 = value2
    def setValue(self, value1, value2):
        self._value1 = value1
        self.__value2 = value2
    def show(self):
        print(self._value1)
        print(self.__value2)

a=A()
a1=A()
print(a._value1, a1._value1)
print(a.color + " " + a1.color)
A._value1=10
print(a._value1, a1._value1)
A.color="green"
print(a.color + " " + a1.color)
```

2.2 数据成员

- 为什么要搞__init__ 这种系统默认
  - 比如：结合类数据成员的共享性，可以实时获得该类的对象数量，并且可以控制该类可以创建的对象最大数量。例如：

```
class Demo(object):
    total = 0
    def __new__(cls, *args, **kwargs):          #该方法在__init__()之前被调用
        if cls.total >= 3:                      #最多允许创建3个对象
            raise Exception('最多只能创建3个对象')
        else:
            return object.__new__(cls)
    def __init__(self):
        Demo.total = Demo.total + 1
t1 = Demo()
t2 = Demo()
t3 = Demo()
t4 = Demo()
Exception: 最多只能创建3个对象
t4
NameError: name 't4' is not defined
```

2.3 成员方法

- a. sorted() 和 sorted(a) 有什么差别?

- 方法一般指与特定实例绑定的函数，通过对象调用方法时，对象本身将被作为第一个参数自动传递过去，普通函数并不具备这个特点。

```
t = Demo()
def test(self, v):
    self.value = v
t.test1 = test                    #动态增加普通函数
t.test(t, 3)                      #需要为self传递参数
print(t.value)
3
```

2.3 成员方法：公有方法、私有方法

- 类似数据成员的分类，Python类的成员方法大致可以分为公有方法、私有方法、和抽象方法(暂不要求)这几种类型。

- 私有方法的名字以两个__开始，共有方法无__

- 公有方法通过对象名直接调用，私有方法不能通过对象名直接调用，只能在其他实例方法中通过前缀self进行调用或在外部通过特殊的形式来调用。

- 所有实例方法（包括公有方法、私有方法）都必须至少有一个名为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参数代表当前对象。

- 在实例方法中访问实例成员时需要以self为前缀，但在外部通过对象名调用对象方法时并不需要传递这个参数。

- 如果在外部通过类名调用属于对象的公有方法，需要显式为该方法的self参数传递一个对象名，用来明确指定访问哪个对象的成员。

```
class Root:
    __total = 0
    __value=1

    def show(self):                          #普通实例方法
            print('self.__value:', self.__value)
        print('Root.__total:', Root.__total)
        self.__test(2)
        print(self.__total)                   # 对象的私有成员被私有方法改变了
            print(Root.__total)              # 类的私有成员没有被改变, 除非直接针对类

       def __test(self, v):    #构造方法
            self.__total = v
        #Root.__total = v+1

r=Root()
r.show()
#r.__test(2)
```

3 继承、多态

- 封装、继承、多态是面向对象编程的三大要素
  - 封装：之前讲的类 vs 对象，成员（数据和方法，公有 vs 私有）
  - 继承：遗传（儿女具有双亲相同的基因，表现为相同的性状），同一名字的成员（主要是方法）相同功能
  - 多态：变异（儿女发育出不同的基因，表现为不同的性状），同一名字的成员（主要是方法）不同功能

- 继承：
  - 继承是用来实现代码复用和设计复用的机制，是面向对象程序设计的重要特性之一。设计一个新类时，如果可以继承一个已有的设计良好的类然后进行二次开发，无疑会大幅度减少开发工作量。
  - 在继承关系中，已有的、设计好的类称为父类或基类，新设计的类称为子类或派生类。派生类可以继承父类的公有成员，但是不能继承其私有成员。如果需要在派生类中调用基类的方法，可以使用内置函数super()或者通过“基类名.方法名()”的方式来实现这一目的。
  - Python支持多继承，如果父类中有相同的方法名，而在子类中使用时没有指定父类名，则Python解释器将从左向右按顺序进行搜索。

3.2 多态

■所谓多态（polymorphism），是指基类的同一个方法在不同派生类对象中具有不同的表现和行为。派生类继承了基类行为和属性之后，还会增加某些特定的行为和属性，同时还可能会对继承来的某些行为进行一定的改变，这都是多态的表现形式。

■Python大多数运算符可以作用于多种不同类型的操作数，并且对于不同类型的操作数往往有不同的表现，这本身就是多态，是通过特殊方法与运算符重载实现的。

3.2 多态

```
class Animal(object):      #定义基类
    def show(self):
        print('I am an animal.')

class Test(Animal):        #继承，没有覆盖基类的show()方法
    pass


class Cat(Animal):         #派生类，覆盖了基类的show()方法
    def show(self):
        print('I am a cat.')

class Dog(Animal):         #派生类
    def show(self):
        print('I am a dog.')

class Tiger(Animal):       #派生类
    def show(self):
        print('I am a tiger.')

class TigerRepeat(Animal):         #继承，增加新方法
    def shout(self,v):
        for i in range(v):
            print('I am not a tiger.')
```

```
x = [item() for item in (Animal, Test, Cat, Dog, Tiger, TigerRepeat)]
for item in x:        #遍历基类和派生类对象并调用show()方法
    item.show()

print("Tiger Shout")
a=TigerRepeat()
a.shout(3)
```

为什么要面向对象?

游戏世界的设计：从一花一草到一人一畜

软件系统的设计：从菜单按钮到功能模块