# 四 面向对象

## 1 基本简介

|  |
| --- |
| *#定义类时有2种：下面是经典类，如果是Person(object)为新式类* **class** Person:  **def** run(self):  print(**'人可以跑'**)  *#创建对象* p = Person() p.name = **'zs'** p.run() |

## 2 \_\_init\_\_方法

**class** Car:  
 **def** \_\_init\_\_(self):  
 self.name = **'benchi'** self.color = **'blue'  
  
 def** move(self):  
 print(**'run...'**)  
  
BenChi = Car();  
print(**'车的颜色为:%s'**%BenChi.color)

\_\_init\_\_()方法，在创建一个对象时默认被调用，不需要手动调用；

\_\_init\_\_(self)中，默认有1个参数名字为self，如果在创建对象时传递了2个实参，那么\_\_init\_\_(self)中出了self作为第一个形参外还需要2个形参，例如\_\_init\_\_(self,x,y)。

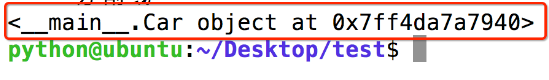
\_\_init\_\_(self)中的self参数，不需要开发者传递，python解释器会自动把当前的对象引用传递进去。

给init传递参数：

**class** Car:  
  
 **def** \_\_init\_\_(self, newColor):  
 self.color = newColor  
  
 **def** move(self):  
 print(**'车在跑，目标:夏威夷'**)  
  
BMW = Car(4, **'green'**)  
  
print(**'车的颜色为:%s'**%BMW.color)

## 3 \_\_str\_\_方法

使用print方法直接打印对象，得到的是对象的地址值。



使用\_\_str\_\_改造：

**class** Car:  
  
 **def** \_\_init\_\_(self, newWheelNum, newColor):  
 self.wheelNum = newWheelNum  
 self.color = newColor  
  
 **def** \_\_str\_\_(self):  
 msg = **"嘿。。。我的颜色是"** + self.color + **"我有"** + int(self.wheelNum) + **"个轮胎..."  
 return** msg  
  
 **def** move(self):  
 print(**'车在跑，目标:夏威夷'**)  
  
  
BMW = Car(4, **"白色"**)  
print(BMW)



在python中方法名如果是\_\_xxxx\_\_()的，那么就有特殊的功能，因此叫做“魔法”方法

当使用print输出对象的时候，只要自己定义了\_\_str\_\_(self)方法，那么就会打印从在这个方法中return的数据

## 4 私有属性

如果有一个对象，当需要对其进行修改属性时，有2种方法

对象名.属性名 = 数据 ---->直接修改

对象名.方法名() ---->间接修改

为了更好的保存属性安全，即不能随意修改，一般的处理方式为

将属性定义为私有属性

添加一个可以调用的方法，供调用

**class** People(object):  
  
 **def** \_\_init\_\_(self, name):  
 self.\_\_name = name  
  
 **def** getName(self):  
 **return** self.\_\_name  
  
 **def** setName(self, newName):  
 **if** len(newName) >= 5:  
 self.\_\_name = newName  
 **else**:  
 print(**"error:名字长度需要大于或者等于5"**)  
  
xiaoming = People(**"dongGe"**)  
print(xiaoming.\_\_name)

Python中没有像C++中public和private这些关键字来区别公有属性和私有属性

它是以属性命名方式来区分，如果在属性名前面加了2个下划线'\_\_'，则表明该属性是私有属性，否则为公有属性（方法也是一样，方法名前面加了2个下划线的话表示该方法是私有的，否则为公有的）。

## 5 \_\_del\_\_方法

创建对象后，python解释器默认调用\_\_init\_\_()方法；

当删除一个对象时，python解释器也会默认调用一个方法，这个方法为\_\_del\_\_()方法。

**import** time  
**class** Animal(object):  
  
 *# 初始化方法  
 # 创建完对象后会自动被调用* **def** \_\_init\_\_(self, name):  
 print(**'\_\_init\_\_方法被调用'**)  
 self.\_\_name = name  
  
  
 *# 析构方法  
 # 当对象被删除时，会自动被调用* **def** \_\_del\_\_(self):  
 print(**"\_\_del\_\_方法被调用"**)  
 print(**"%s对象马上被干掉了..."**%self.\_\_name)  
  
*# 创建对象*dog = Animal(**"哈皮狗"**)  
  
*# 删除对象***del** dog  
  
  
cat = Animal(**"波斯猫"**)  
cat2 = cat  
cat3 = cat  
  
print(**"---马上 删除cat对象"**)  
**del** cat  
print(**"---马上 删除cat2对象"**)  
**del** cat2  
print(**"---马上 删除cat3对象"**)  
**del** cat3  
  
print(**"程序2秒钟后结束"**)  
time.sleep(2)

当有1个变量保存了对象的引用时，此对象的引用计数就会加1

当使用del删除变量指向的对象时，如果对象的引用计数不会1，比如3，那么此时只会让这个引用计数减1，即变为2，当再次调用del时，变为1，如果再调用1次del，此时会真的把对象进行删除。

## 6 继承

*# 定义一个父类，如下:***class** Cat(object):  
  
 **def** \_\_init\_\_(self, name, color=**"白色"**):  
 self.name = name  
 self.color = color  
  
 **def** run(self):  
 print(**"%s--在跑"**%self.name)  
  
  
*# 定义一个子类，继承Cat类如下:***class** Bosi(Cat):  
  
 **def** setNewName(self, newName):  
 self.name = newName  
  
 **def** eat(self):  
 print(**"%s--在吃"**%self.name)  
  
  
bs = Bosi(**"印度猫"**)  
print(**'bs的名字为:%s'**%bs.name)  
print(**'bs的颜色为:%s'**%bs.color)  
bs.eat()  
bs.setNewName(**'波斯'**)  
bs.run()

虽然子类没有定义\_\_init\_\_方法，但是父类有，所以在子类继承父类的时候这个方法就被继承了，所以只要创建Bosi的对象，就默认执行了那个继承过来的\_\_init\_\_方法

子类在继承的时候，在定义类时，小括号()中为父类的名字；

父类的属性、方法，会被继承给子类。

注意：

私有的属性，不能通过对象直接访问，但是可以通过方法访问

私有的方法，不能通过对象直接访问

私有的属性、方法，不会被子类继承，也不能被访问

一般情况下，私有的属性、方法都是不对外公布的，往往用来做内部的事情，起到安全的作用。

## 7 多继承

*# 定义一个父类***class** A:  
 **def** printA(self):  
 print(**'----A----'**)  
  
*# 定义一个父类***class** B:  
 **def** printB(self):  
 print(**'----B----'**)  
  
*# 定义一个子类，继承自A、B***class** C(A,B):  
 **def** printC(self):  
 print(**'----C----'**)  
  
obj\_C = C()  
obj\_C.printA()  
obj\_C.printB()

## 8 重写父类方法

所谓重写，就是子类中，有一个和父类相同名字的方法，在子类中的方法会覆盖掉父类中同名的方法。

*#coding=utf-8***class** Cat(object):  
 **def** sayHello(self):  
 print(**"halou-----1"**)  
**class** Bosi(Cat):  
  
 **def** sayHello(self):  
 print(**"halou-----2"**)  
bosi = Bosi()  
bosi.sayHello()

调用父类方法：

*#coding=utf-8***class** Cat(object):  
 **def** \_\_init\_\_(self,name):  
 self.name = name  
 self.color = **'yellow'  
  
class** Bosi(Cat):  
  
 **def** \_\_init\_\_(self,name):  
 *# 调用父类的\_\_init\_\_方法1(python2)  
 #Cat.\_\_init\_\_(self,name)  
 # 调用父类的\_\_init\_\_方法2  
 #super(Bosi,self).\_\_init\_\_(name)  
 # 调用父类的\_\_init\_\_方法3* super().\_\_init\_\_(name)  
  
 **def** getName(self):  
 **return** self.name  
  
bosi = Bosi(**'xiaohua'**)  
  
print(bosi.name)  
print(bosi.color)

## 9 多态

多态的概念是应用于Java和C#这一类强类型语言中，而Python崇尚“鸭子类型”。

所谓多态：定义时的类型和运行时的类型不一样，此时就成为多态

Python伪代码实现Java或C#的多态：

**class** F1(object):  
 **def** show(self):  
 print **'F1.show'  
  
class** S1(F1):  
 **def** show(self):  
 print **'S1.show'  
  
class** S2(F1):  
 **def** show(self):  
 print **'S2.show'***# 由于在Java或C#中定义函数参数时，必须指定参数的类型  
# 为了让Func函数既可以执行S1对象的show方法，又可以执行S2对象的show方法，所以，定义了一个S1和S2类的父类  
# 而实际传入的参数是：S1对象和S2对象***def** Func(F1 obj):  
 *"""Func函数需要接收一个F1类型或者F1子类的类型"""* print obj.show()  
  
s1\_obj = S1()  
Func(s1\_obj) *# 在Func函数中传入S1类的对象 s1\_obj，执行 S1 的show方法，结果：S1.show*s2\_obj = S2()  
Func(s2\_obj) *# 在Func函数中传入Ss类的对象 ss\_obj，执行 Ss 的show方法，结果：S2.show*

Python “鸭子类型”：

class F1(object):

def show(self):

print 'F1.show'

class S1(F1):

def show(self):

print 'S1.show'

class S2(F1):

def show(self):

print 'S2.show'

def Func(obj):

print obj.show()

s1\_obj = S1()

Func(s1\_obj)

s2\_obj = S2()

Func(s2\_obj)

## 10 类与实例

### 10.1 类属性

class People(object):

name = 'Tom' #公有的类属性

\_\_age = 12 #私有的类属性

p = People()

print(p.name) #正确

print(People.name) #正确

print(p.\_\_age) #错误，不能在类外通过实例对象访问私有的类属性

print(People.\_\_age) #错误，不能在类外通过类对象访问私有的类属性

### 10.2 实例属性

class People(object):

address = '山东' #类属性

def \_\_init\_\_(self):

self.name = 'xiaowang' #实例属性

self.age = 20 #实例属性

p = People()

p.age =12 #实例属性

print(p.address) #正确

print(p.name) #正确

print(p.age) #正确

print(People.address) #正确

print(People.name) #错误

print(People.age) #错误

### 10.3 使用实例修改类属性

class People(object):

country = 'china' #类属性

print(People.country)

p = People()

print(p.country)

p.country = 'japan'

print(p.country) #实例属性会屏蔽掉同名的类属性

print(People.country)del p.country #删除实例属性

print(p.country)

如果需要在类外修改类属性，必须通过类对象去引用然后进行修改。如果通过实例对象去引用，会产生一个同名的实例属性，这种方式修改的是实例属性，不会影响到类属性，并且之后如果通过实例对象去引用该名称的属性，实例属性会强制屏蔽掉类属性，即引用的是实例属性，除非删除了该实例属性。

### 10.4 类方法

类方法是类对象所拥有的方法，需要用修饰器@classmethod来标识其为类方法，对于类方法，第一个参数必须是类对象，一般以cls作为第一个参数（当然可以用其他名称的变量作为其第一个参数，但是大部分人都习惯以'cls'作为第一个参数的名字，就最好用'cls'了），能够通过实例对象和类对象去访问。

class People(object):

country = 'china'

#类方法，用classmethod来进行修饰 @classmethod

def getCountry(cls):

return cls.country

p = People()print p.getCountry() #可以用过实例对象引用print People.getCountry() #可以通过类对象引用

类方法还有一个用途就是可以对类属性进行修改：

class People(object):

country = 'china'

#类方法，用classmethod来进行修饰 @classmethod

def getCountry(cls):

return cls.country

@classmethod

def setCountry(cls,country):

cls.country = country

p = People()print p.getCountry() #可以用过实例对象引用print People.getCountry() #可以通过类对象引用

p.setCountry('japan')

print p.getCountry() print People.getCountry()

### 10.5 静态方法

需要通过修饰器@staticmethod来进行修饰，静态方法不需要多定义参数。

class People(object):

country = 'china'

@staticmethod

#静态方法

def getCountry():

return People.country

print People.getCountry()

从类方法和实例方法以及静态方法的定义形式就可以看出来，类方法的第一个参数是类对象cls，那么通过cls引用的必定是类对象的属性和方法；而实例方法的第一个参数是实例对象self，那么通过self引用的可能是类属性、也有可能是实例属性（这个需要具体分析），不过在存在相同名称的类属性和实例属性的情况下，实例属性优先级更高。静态方法中不需要额外定义参数，因此在静态方法中引用类属性的话，必须通过类对象来引用。