Python-面向对象

**推荐使用驼峰命名法**：

类名：没单词的首字母大写

上车：ShangChe

开车：KaiChe

函数名/变量名：除了第一个单词之位的首字母大写

上车：shangChe

开车：kaiChe 或者 kai\_che(这不是驼峰命名法)

文件名：如果一个文件是一个单独的类，那么直接使用类名来当做文件名

**类：**

格式：class 类名 :

组成：类中只有两种内容：成员属性和成员方法

成员属性：

作用：用于描述类的特征的变量

调用：类名.成员属性名

成员方法：

作用：用于描述类的功能的函数

调用：类名.成员方法名()

注意：

1. 所有对象在调用方法的时候会默认往方法里传一个参数（对象自己），所以如果对象想调用类中的方法，在创建方法的时候加一个参数即可，推荐self (自己、自我 的意思) 例如：def 方法名(self):



**对象和类的成员检测：**

类的成员

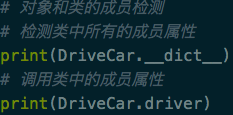
格式：类名.\_\_dict\_\_

对象的成员：

格式：对象名.\_\_dict\_\_

注意：

1. 会发现如果从来没对对象里面的成员做过修改，那么打印出来的是个空集合
2. 如果同一个类同时实例化了多个对象，只要不对类进行修改，那么所有对单个对象做的修改都不会影响到其他对象



**类和对象的操作：**

*类成员操作：*

访问类中成员：

类名.成员属性名

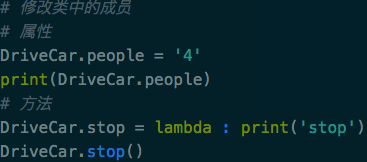
类名.成员方法名() (没有参数要求，所有的方法都可以通过类访问)



修改类中的成员:

类名.成员属性名 = 值

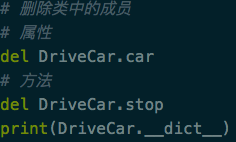
类名.成员方法名 = 新值(如果需要函数只能用lambda)



删除类中的成员:

del.类名.成员属性名

del.类名.成员方法名



添加类中的成员:

类名.新成员属性名 = 值

类名.新成员方法名 = 值(如果需要函数只能用lambda)



*对象成员操作:*

实例化:

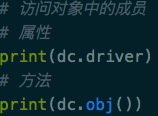
对象名 = 类名()



访问对象中的成员

对象名.成员属性名

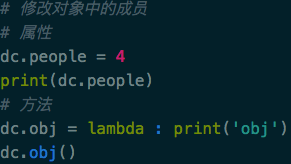
对象名.成员方法名() (对象只能调用带有对象形参的成员方法)



修改对象中的成员

对象名.成员属性名 = 值

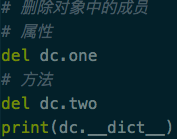
对象名.成员方法名 = 值(如果需要函数只能用lambda)



删除对象中的成员

del 对象名.成员属性名

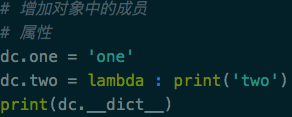
del 对象名.成员方法名



增加对象中的成员

对象名.新成员属性名 = 值

对象名.新成员方法吗 = 值(如果需要函数只能用lambda)



**面向对象的三大特征**：(封装丶继承丶多态)

***封装特性:***

封装就是对(类和对象)的成员访问进行限制,设定可以访问的方式和不可以访问的方式

封装的三个级别:

私有化封装 -> private

格式:

成员属性: \_\_属性名 = 值

成员方法: def \_\_方法名():

类调用私有化成员:

成员属性:类名.\_\_属性名

成员方法:类名.\_\_方法名()

对象调用私有化成员:

成员属性:对象名.\_类名\_\_属性名

成员方法:对象名.\_类名\_\_方法名()

类中/对象中 可以访问

类外/对象外 不可以访问

子类/子类对象 不可以访问

注意:

1. \_\_是俩个下划线,调用的时候也需要写上\_\_
2. 对象调用成员方法依然需要将对象本身传入到方法当中
3. Python3中的私有化并不是通过权限来限制的,而是通过改名来实现的
   1. 成员属性 = \_类名\_\_属性名
   2. 成员方法 = \_类名\_\_方法名
   3. 类名/方法名.\_类名\_\_成员名 不推荐/禁用这种调用方式

受保护的封装 -> protected

一定级别的封装,封装之后,只有类和子类能访问,类外不能访问

格式:

成员属性:\_属性名

成员方法:\_方法名()

类调用受保护的成员:

成员属性:类名.\_属性名

成员方法:类名.\_方法名()

对象调用受保护的成员:

成员属性:对象名.\_属性名

成员方法:对象名.\_方法名()

类中/对象中 可以访问

类外/对象外 可以访问(原则上不行,但是没实现)

子类/子类对象 可以访问

注意:

1. 依旧是通过改名来实现的
   1. 成员属性 = \_类名\_属性名
   2. 成员方法 = \_类名\_方法名
   3. 类名/方法名.\_类名\_成员名 不推荐/禁用这种调用方式

公共的封装 -> public

左右成员默认都是公共的封装级别,可以在类中,类外,以及子类中正常访问

***继承特性:***

继承就是可以获取另外一个类中的成员属性和成员方法(私有成员除外)

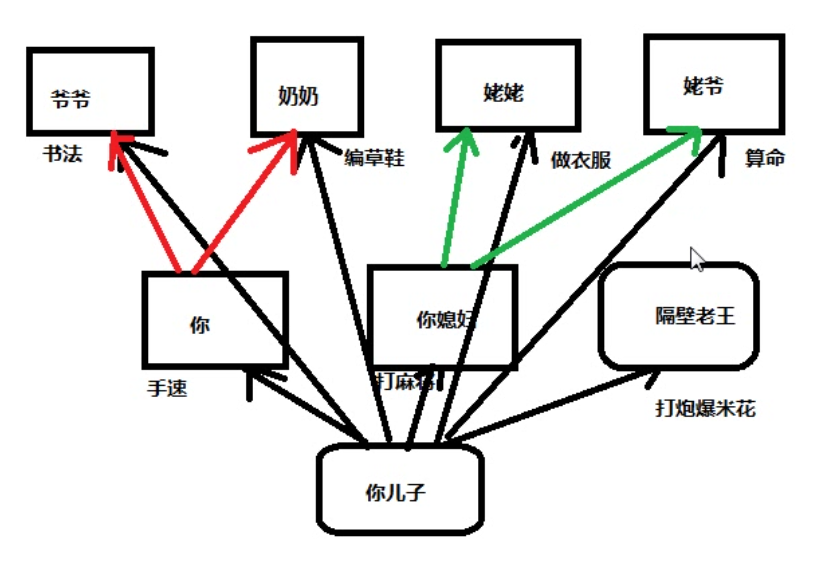
所有类都继承自object类

单继承格式: class 类名(继承的类名):

多继承格式: class 类名(继承的类名1,继承的类名2....)

多个父类中拥有同名成员的时候,会优先调用先继承的父类成员

一般不用,如果父类和子类同时继承了一个类,也就是同时继承了两遍,调用的时候会乱套



知识点:

在子类中super()是代表父类意思,可以通过super().属性/方法()调用父类成员. 注意使用super()必须实例化对象,通过对象来调用

还有菱形继承和钻石继承,和多继承一样,容易混乱,所以不常用

MRO列表:

Method Realtion Order 为了制作一个继承关系的列表

Python3中使用C3算法来计算了MRO列表

MRO列表的制作原则:

1. 子类永远在父类的前面
2. 如果继承了多个父类,那么按照()中的顺序在列表中摆放
3. 如果多个类同时继承了一个父类,孙子类中只会选取第一个父类中的父类的该方法

查看MRO列表: 类名.mro()

***多态特性:***

描述：多种状态，就是一个类，根据不同的情况，相同的方法产生不同的结果

*mixin设计模式:*

其实就是定义类的时候更加细分,方便子类继承.

该设计模式的只要作用是采用多继承方式,进行类的扩展.

**描述符:**

功能：主要用于描述对象中的属性，主要作用是对属性操作进行限制、验证、管理等相关的权限操作

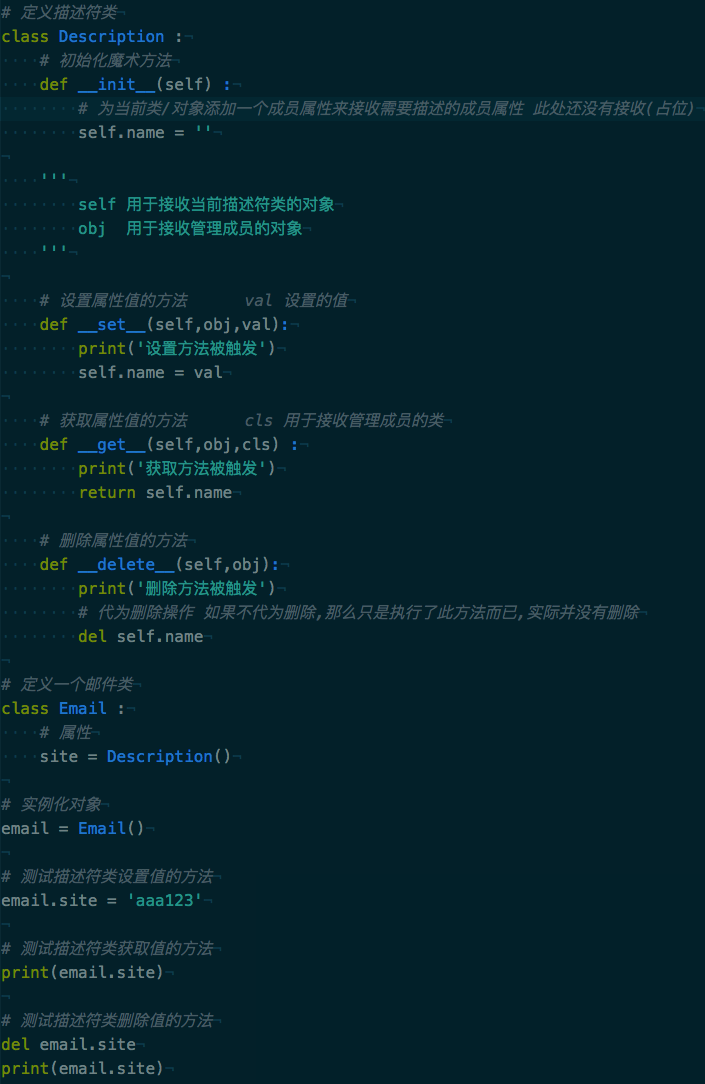
描述符主要有三种需要设置：

get 获取属性的操作

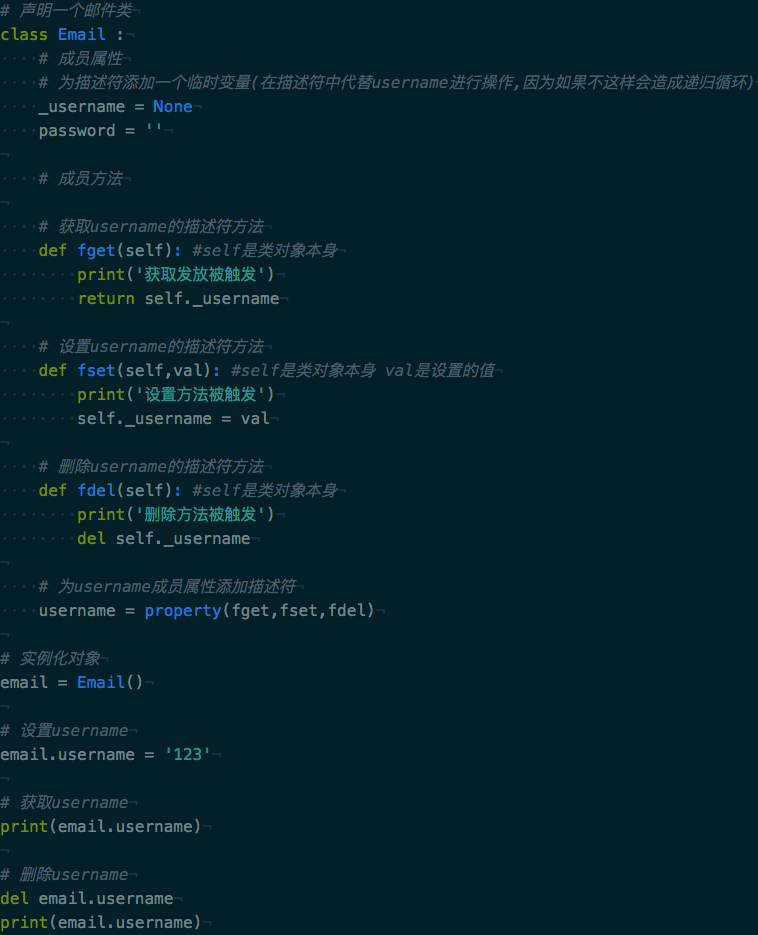
set 设置属性的操作

delete 删除属性的操作

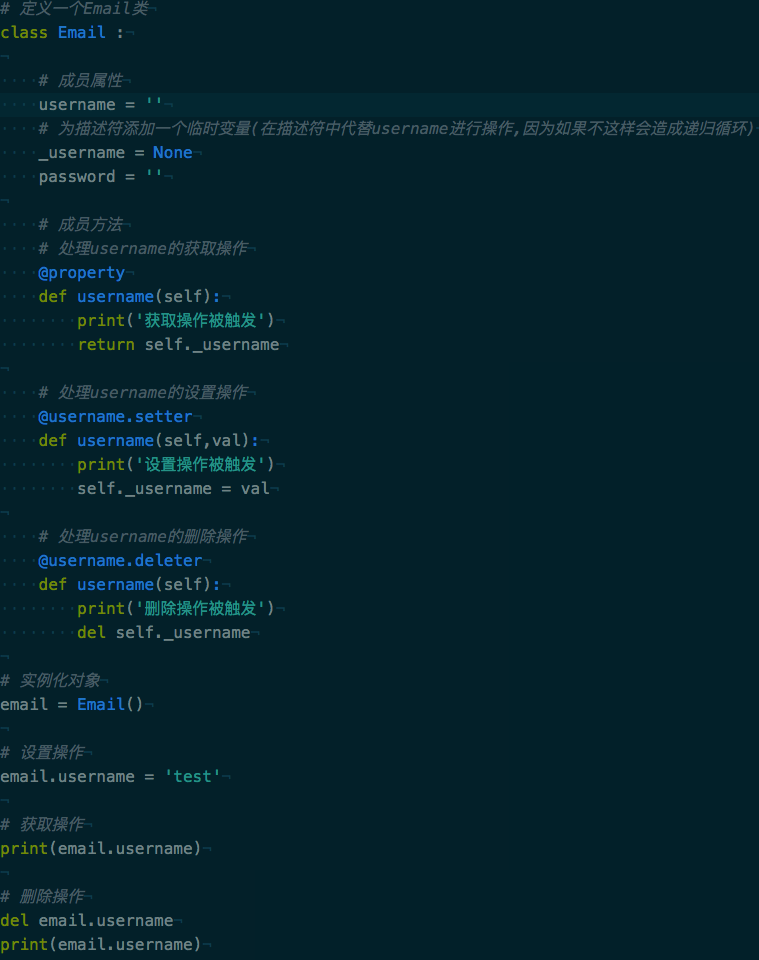
***方法1（描述符）:***



***方法2（描述符）:***



***方法3（修饰符）:***

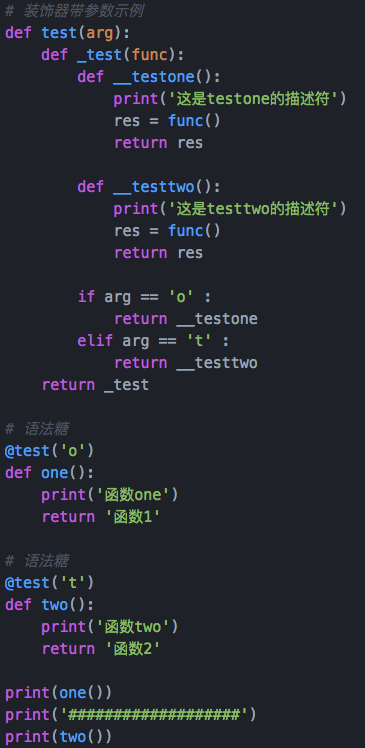


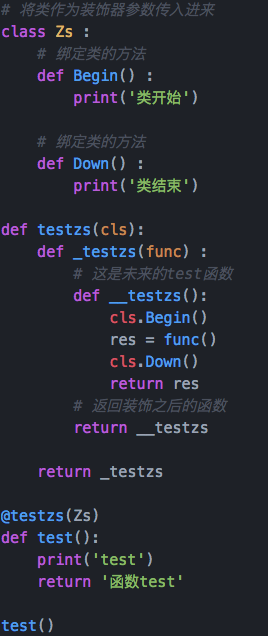
**装饰器：**

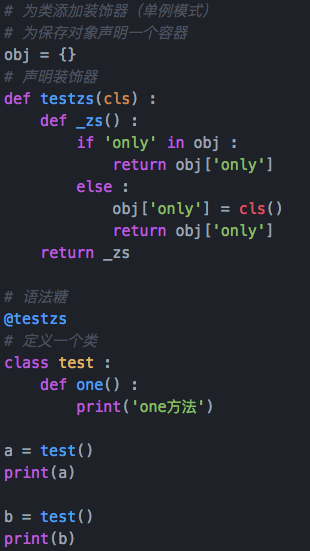
功能：装饰器本质上是一个Python函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外功能，装饰器的返回值也是一个函数对象。它经常用于有切面需求的场景，比如：插入日志、性能测试、事务处理、缓存、权限校验等场景。装饰器是解决这类问题的绝佳设计，有了装饰器，我们就可以抽离出大量与函数功能本身无关的雷同代码并继续重用。概括的讲，装饰器的作用就是为已经存在的对象添加额外的功能。

示例：







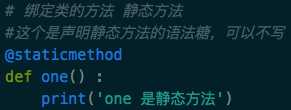


**方法的分类：**

绑定类的方法 静态方法

参数：不需要参数

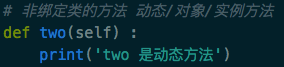
实例：



非绑定类的方法 动态方法

参数：必须把对象传进去(对象调用这个方法的时候会自动传入)

实例：



类方法

参数：必须把类传进去(类调用这个方法的时候回自动传入)

实例：



**抽象类：**

具有抽象方法的类就是抽象类

抽象方法就是没完成的方法，只有方法名称和参数，没有方法内容的方法。

作用：适用于领导指定开发规范及多人协作完成类。

注意：使用抽象类必须使用abc模块

示例：



**模块：**

描述：一个包含Python代码的文件就是一个模块。他可以提高维护和管理的便捷性、提高代码的利用率、可以当做命名空间。

自定义模块存放位置：

安装目录/lib/

尽量和Python自带的模块放到一起，方便后期引用和维护。

导入模块：

import 模块名 （导入模块）

使用：

模块名.变量名

模块名.函数名

模块名.类名.方法名()

模块名.类名.属性名

import 模块名 as 别名 （导入模块，并为模块起一个别名）

使用：

别名.变量名

别名.函数名

别名.类名.方法名()

别名.类名.属性名

from 模块名 import 成员属性名/类名 （从指定的模块中导入指定的内容）

from 模块名 import 类,类,函数,函数 （从指定的模块中导入多个指定的内容）

from 模块名 import \* （从指定的模块中导入所有内容）

使用：

变量名

函数名

类名.方法名()

类名.属性名

**包：**

描述：包就是文件夹，它用于存放文件也就是模块。包中也可以存放包。

存放位置：

安装目录/lib/

包的结构：

1. \_\_init\_\_.py 包的标志文件(可为空)

2. 模块

3. 子包 (也需要\_\_init\_\_.py)

4. 子模块

调用：

引入：

import 包.模块

import 包.模块 as 别名

from 包.模块 import 函数1,方法2,类

使用：

包.模块.变量名

包.模块.函数名()

包.模块.类名.属性名

包.模块.类名.方法名()