**模块**

模块和文件

什么是模块

模块支持从逻辑上组织python代码

当代码量变得很大的时候，最好把代码分成一些由组织的代码段

代码段相互间有一定的联系，可能是一个包含数据成员和方法的类，也可以是一组相关但彼此独立的操作函数

这些代码段是共享的，所以python允许“调入”一个模块，允许使用其他模块的属性来利用之前的工作成果，实现代码重用

模块文件

说模块是按照逻辑来组织python代码的方法，文件是物理层上组织模块的方法

一个文件被看作是一个独立的模块，一个模块也可以被看作是一个文件

模块的文件名就是模块的名字加上扩展名.py

名称空间

名称空间就是一个从名称到对象的关系映射集合

给定一个模块后，只可能有一个模块被导入到python解释器中，所以在不同模块间不会出现名称交叉现象

每个模块都定义了它自己的唯一的名称空间



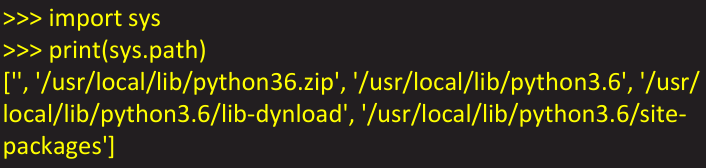
导入模块

搜索路径

模块的导入需要一个叫做“路径搜索”的过程

python在文件系统“预定义区域”中查找要调用的模块

搜索路径在sys.path中定义





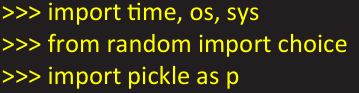
模块导入方法

使用import导入模块

可以在一行导入多个模块，但是可读性会下降

可以只导入模块的某些属性

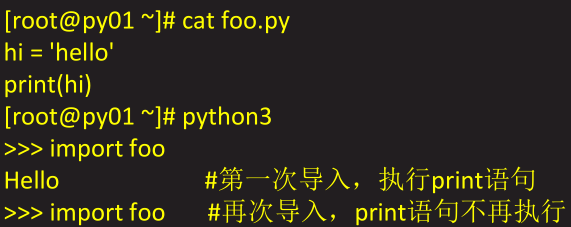
导入模块时，可以为模块取别名



导入和加载

当导入模块时，模块的顶层代码会被执行

一个模块不管被导入（import）多少次，只会被加载（load）一次



从zip文件中导入

在2.3版中，python加入了从ZIP归档文件导入模块的功能

如果搜索路径中存在一个包含python模块（.py、.pyc或.pyo文件）的.zip文件，导入时会把ZIP文档当作目录处理



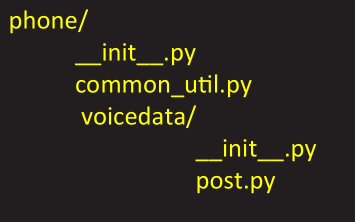
包

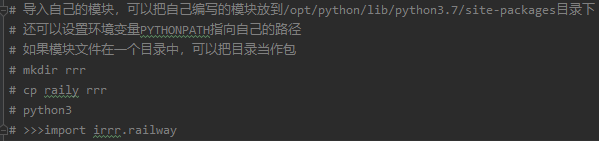
目录结构

包是一个有层次的文件目录结构，为平坦的名称空间加入有层次的组织结构

允许程序员把有联系的模块组合到一起

包目录下必须有一个\_\_init\_\_.py文件





绝对导入

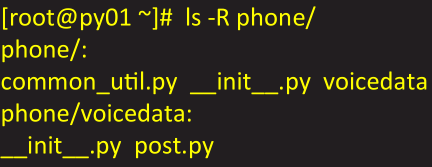
包的使用越来越广泛，很多情况下导入子包会导致和真正的标准库发生冲突

因此，所有的导入现在都被认为是绝对的，也就是说这些名字必须通过python路径（sys.path或PATHONPATH）来访问

相对导入

绝对导入特性使得程序员失去了import的自由，为此出现了相对导入

因为import语句总是绝对导入的，所以相对导入只应用于from-import语句

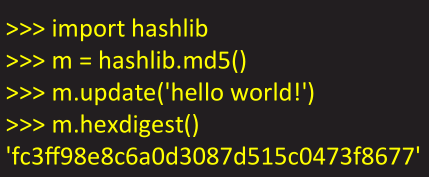


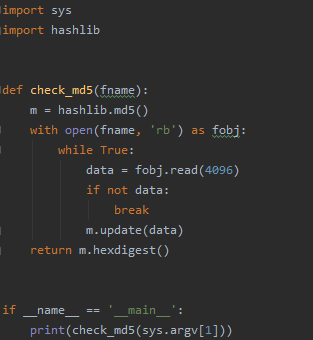
内置模块

hashlib模块

hashlib用来替换md5和sha模块，并使用他们的API一致，专门提供hash算法

包括md5、sha1、sha224、sha256、sha384、sha512，使用简单方便

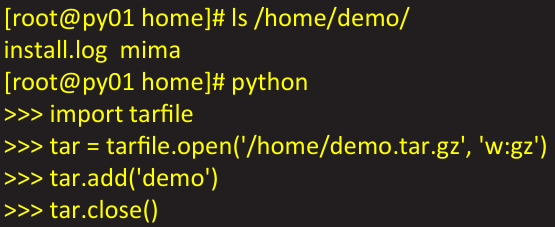




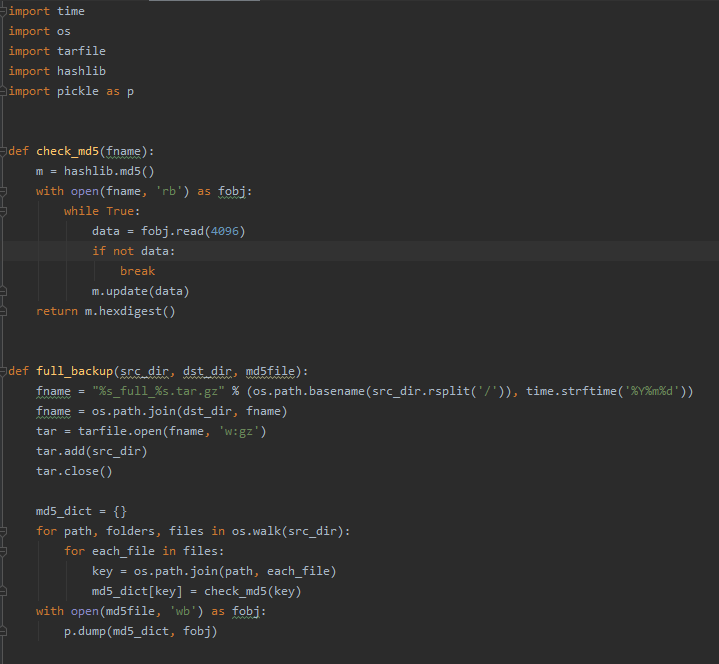
tarfile模块

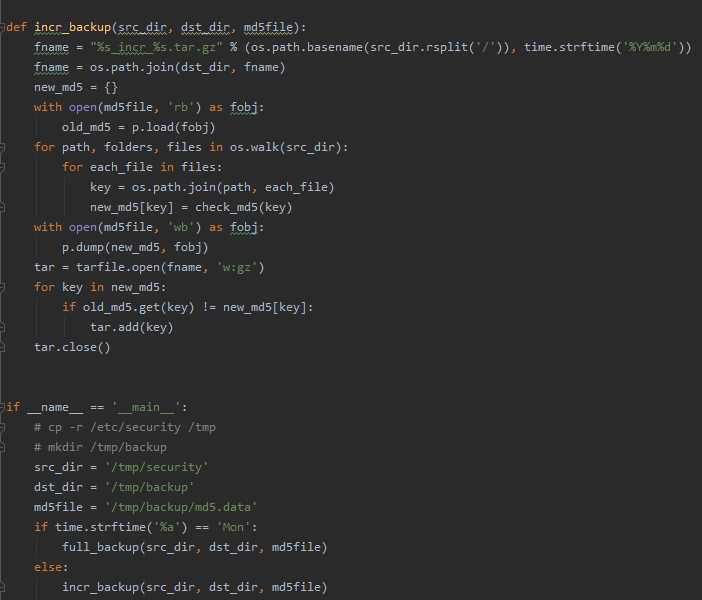
tarfile模块允许创建、访问tar文件

同时支持gzip、bzip2格式









**OOP基础**

OOP简介

基本概念

类（Class）：用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例

实例化：创建一个类的实例，类的具体对象

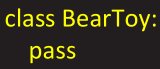
方法：类中定义的函数

对象：通过类定义的数据结构实例。对象包含两个数据成员（类变量和实例变量）和方法

创建类

使用class语句来创建一个新类，class之后为类的名称并以冒号结尾

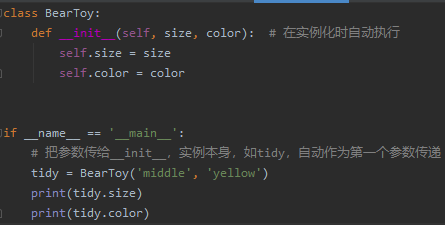
类名建议使用驼峰形式



创建实例

类是蓝图，实例是根据蓝图创建出来的具体对象



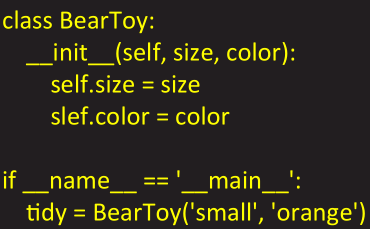


绑定方法

构造器方法

当实例化类的对象是，构造器方法默认自动调用

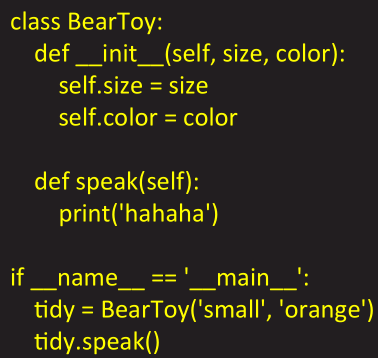
实例本身作为第一个参数，传递给self



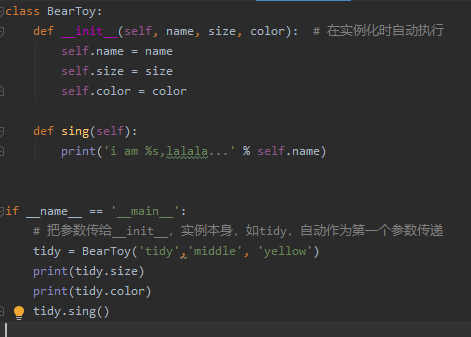
其他绑定方法

类中定义的方法需要绑定在具体的实例，由实例调用

实例方法需要明确调用







**OOP进阶**

组合和派生

什么是组合

类被定义后，目标就是要把它当成一个模块来使用，并把这些对象嵌入到代码中去

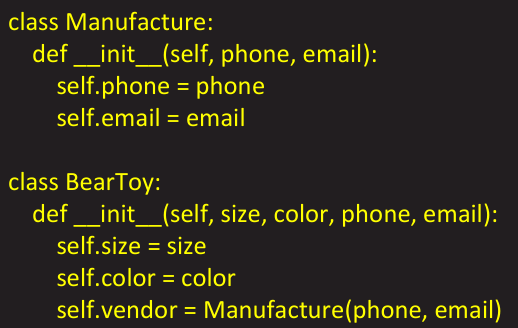
组合就是让不同的类混合并加入到其他类中来增加功能和代码重用性

可以在一个大点的类中创建其他类的实例，实现一些其他属性和方法来增强对原来的类对象

组合应用

两个类明显不同

一个类是另一个类的组件





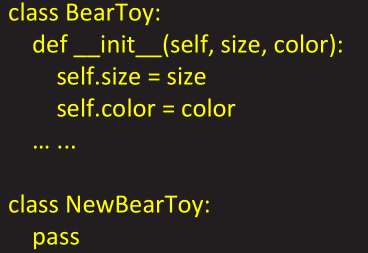
创建子类

当类之间有显著的不同，并且较小的类是较大的类所需要的组件时组合表现的比较好；但当设计“相同的类但有一些不同的功能”时，派生就是一个更加合理的选择了

OOP的更强大方面之一是能够使用一个已经定义好的类，扩展它或者对其进行修改，而不会影响系统中使用现存类的代码片段

OOD（面向对象设计）允许类特征在子孙类或子类中进行继承

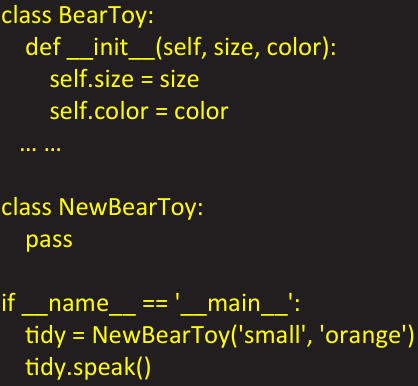
创建子类只需要在圆括号中写明从哪个父类继承即可

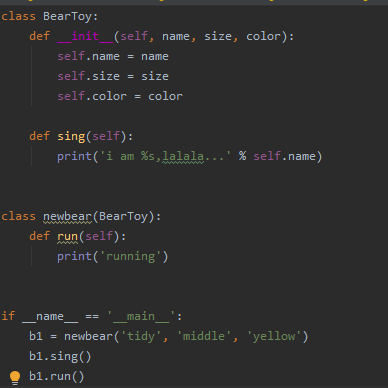


继承

继承描述了积累的属性如何“遗传”给派生类

子类可以继承它的基类的任何属性，不管是数据属性还是方法

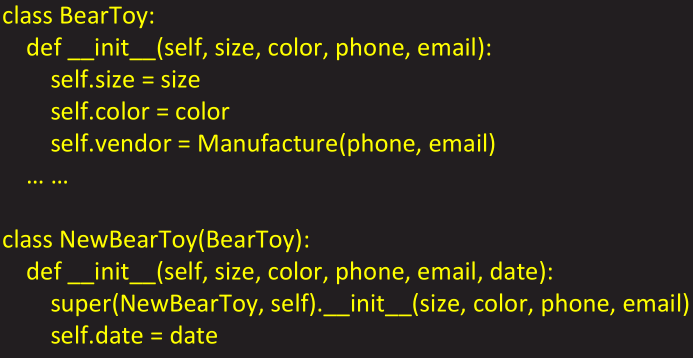


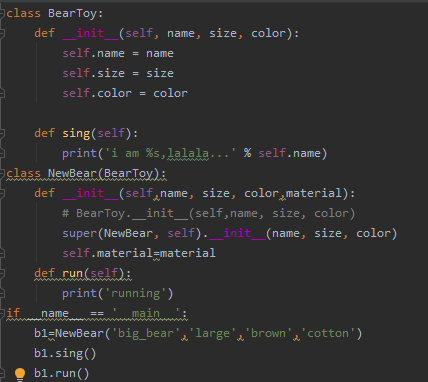


通过继承覆盖方法

如果子类中有和父类同名的方法，父类方法将被覆盖

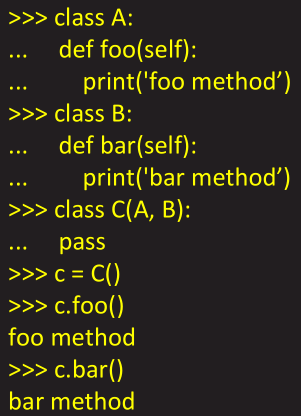
如果需要访问父类的访问，则要调用一个未绑定的父类方法，明确给出子类的实例

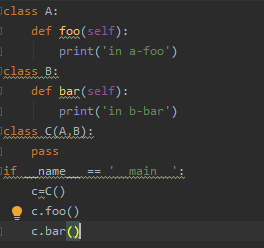




多重继承

python允许多重继承，即一个类可以是多个父类的子类，子类可以拥有所有父类的属性



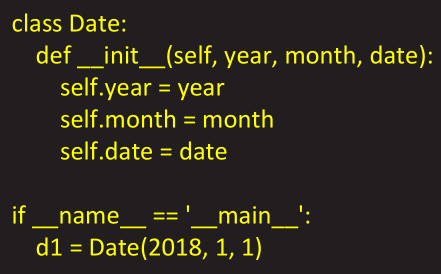


特殊方法

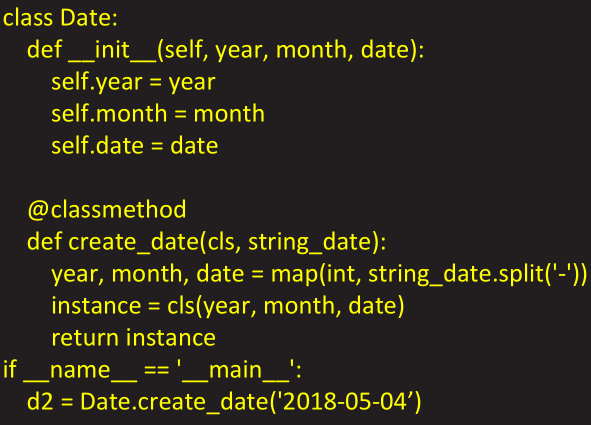
类方法

使用classmethod装饰器定义

第一个参数cls表示类本身



类方法实例



静态方法

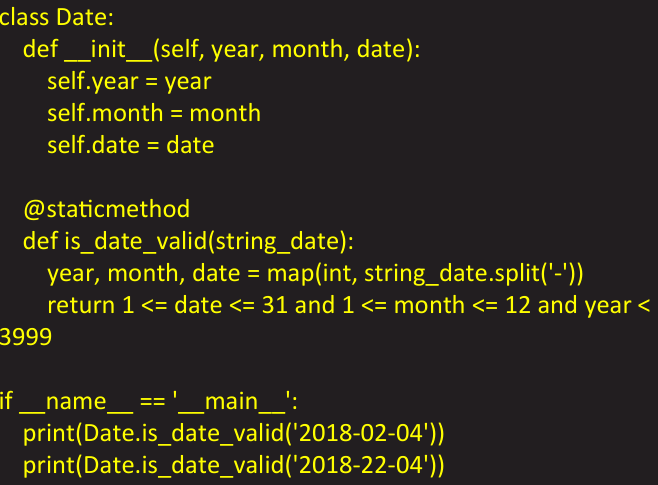
基本上就是一个函数

在语法上就像一个方法

没有访问对象和它的字段和方法

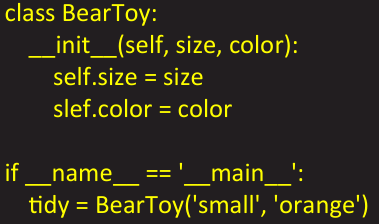
使用staticmethod装饰器定义

静态方法实例



\_\_init\_\_方法

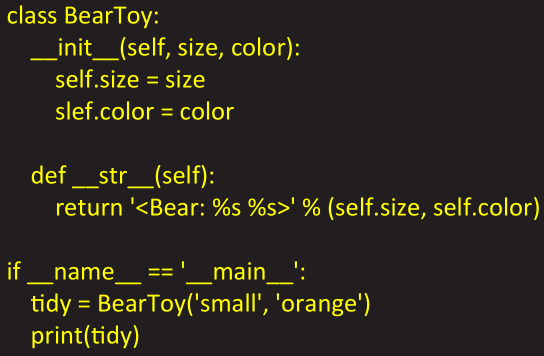
实例化类实例默认会调用的方法



\_\_str\_\_方法

打印/显示实例时调用的方法

返回字符串



\_\_call\_\_方法

用于创建可调用的实例

