[面向对象编程(May24) 1](#_Toc515348879)

[1 ject（对象） 1](#_Toc515348880)

[1.1 什么是对象 1](#_Toc515348881)

[1.2 什么是面向对象 1](#_Toc515348882)

[1.3 对象有什么特征 1](#_Toc515348883)

[2 类(class) 1](#_Toc515348884)

[2.1 什么是类 1](#_Toc515348885)

[2.2类的创建 1](#_Toc515348886)

[3 构造函数 2](#_Toc515348887)

[3.1实例方法 2](#_Toc515348888)

[3.2实例的属性 4](#_Toc515348889)

[3.3删除属性 6](#_Toc515348890)

[4 初始化方法 6](#_Toc515348891)

[5析构方法 7](#_Toc515348892)

[6 预置实例（对象）属性 8](#_Toc515348893)

[7 面向对象的综合示例 8](#_Toc515348894)

[May25 11](#_Toc515348895)

[1 类变量class variable（也叫类属性） 11](#_Toc515348896)

[1.1 类变量的应用案例 12](#_Toc515348897)

[2 类的文档字符串 12](#_Toc515348898)

[3 类的\_\_slots\_\_列表 12](#_Toc515348899)

[4 类方法 @classmethod 13](#_Toc515348900)

[5 静态方法 @staticmethod 14](#_Toc515348901)

[6 继承inheritance和派生derived 14](#_Toc515348902)

[6.1 单继承 15](#_Toc515348903)

[6.2 继承说明 15](#_Toc515348904)

[7 覆盖override 15](#_Toc515348905)

[6.2 子类对象显式调用基类方法的方式 16](#_Toc515348906)

[May28 19](#_Toc515348907)

[1 用于类的函数 19](#_Toc515348908)

[2 封装enclosure 19](#_Toc515348909)

[3 多态polymorphic 19](#_Toc515348910)

[面向对象的编程语言的特征 20](#_Toc515348911)

[4 多继承multipe inheritance 20](#_Toc515348912)

[4.1 多继承的缺陷 21](#_Toc515348913)

[5 函数重写override 23](#_Toc515348914)

[5.1 对象转字符串函数重写 23](#_Toc515348915)

[5.2 对象转字符串函数重写方法 23](#_Toc515348916)

[5.3数值转换函数重写方法 24](#_Toc515348917)

[5.3内建函数重写方法 24](#_Toc515348918)

[**5.3.1 布尔测试函数的重写** 25](#_Toc515348919)

[6 迭代器（高级） 25](#_Toc515348920)

[6.1迭代器协议 25](#_Toc515348921)

[6.2可迭代对象 26](#_Toc515348922)

面向对象编程(May24)

1 ject（对象）

1.1 什么是对象

对象是现实中的物体或实体

1.2 什么是面向对象

把一切看成对象（实例），让对象和对象之间建立关联关系

1.3 对象有什么特征

对象有很多属性（特征，名词）------------------------->变量

姓名、年龄、性别

对象有很多行为（动作，动词）

学习、吃饭、睡觉、踢球、工作-------------------------->函数（方法method）

2 类(class)

2.1 什么是类

拥有相同属性和行为的对象分为一组，即为一个类。类是用来描述对象的工具，用类可以创建同类对象。

2.2类的创建

语法：

class类名（继承列表）：

’’’类的文档字符串’’’

实例方法定义（类内的函数称为方法method）

类变量定义

类方法定义

静态方法定义

作用：

创建一个类

用于描述此类对象的行为和属性

类用于创建此类的一个或多个对象（实例）

|  |  |  |
| --- | --- | --- |
| 类 | 对象 | 实例 |
| class | object | instance |

示例：

class Dog: #定义一个类，类名为Dog

pass

dog1 = Dog( ) #创建Dog类的对象

print(id(dog1))

dog2 = Dog( ) #创建Dog类的另一个对象

print(id(dog2))

#类似于如下语法

int1 = int( )

int2 = int( )

3 构造函数

表达式：

类名（[创建传参列表]）

作用：

创建这个类的实例对象，并返回此实例对象的引用关系

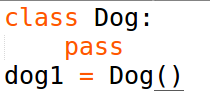
实例（对象）说明：

实例有自己的作用域和名字空间，可以为该实例添加实例变量（属性）

实例可以调用类方法和实例方法

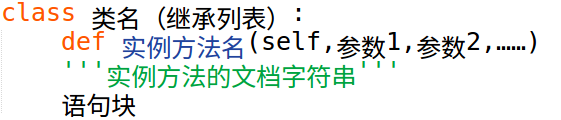
实例可以访问类变量和实例变量

示例：



3.1实例方法

语法：



作用：

用于描述一个对象的行为，让此类型的全部对象都拥有相同的行为

说明：

实例方法实质是函数，是定义在类内的函数

实例方法至少有一个形参，第一个形参代表调用这个方法的实例，一般命名为’self’

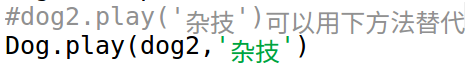
实例方法的调用语法（调用参数）

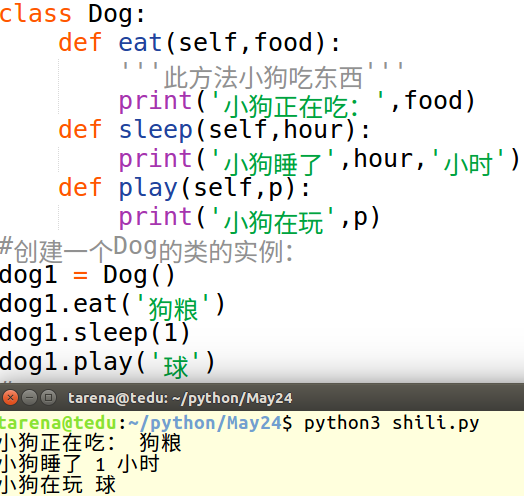
或

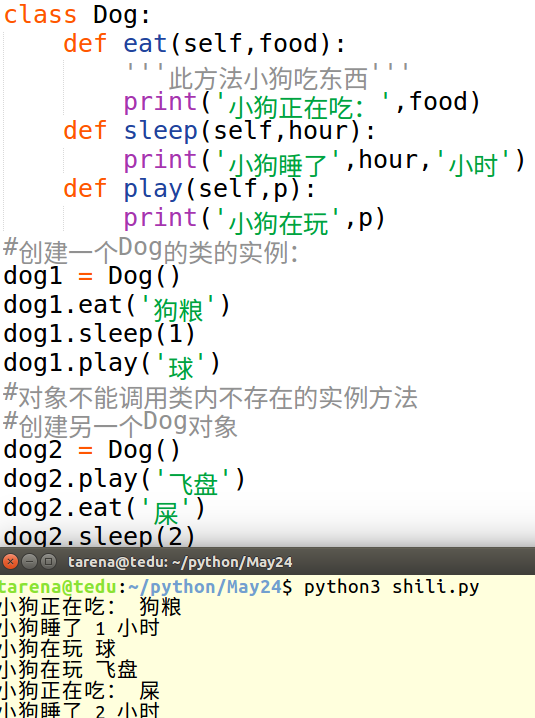
类名.实例方法名（实例，调用传参）

示例：









3.2实例的属性

属性attribute（也叫做实例的变量）

每个实例都可以有自己的变量，此变量称为实例变量（也叫属性）

属性的使用语法：

实例.属性名

赋值规则：

首次为属性赋值则创建此属性

再次为属性赋值必改变属性的绑定关系

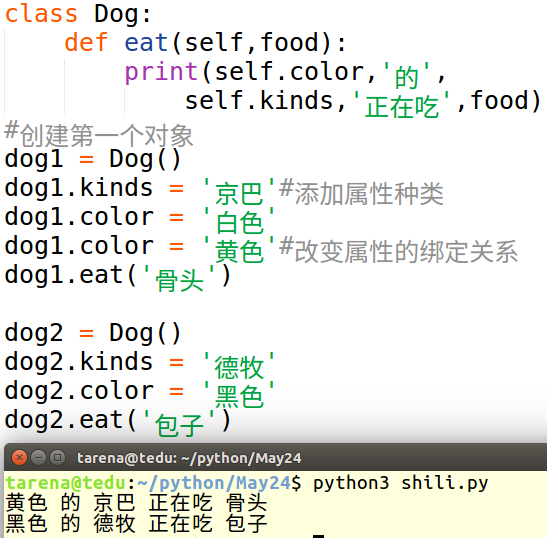
作用：

用来记录对象自身的数据

示例：



**实例方法和实例变量（属性）的结合：**



练习：

1.定义一个学生类：

class student:

def set\_info(self,name,age):

给学生对象添加’姓名’和’年龄’属性

def show\_info(self):

s1 = student()

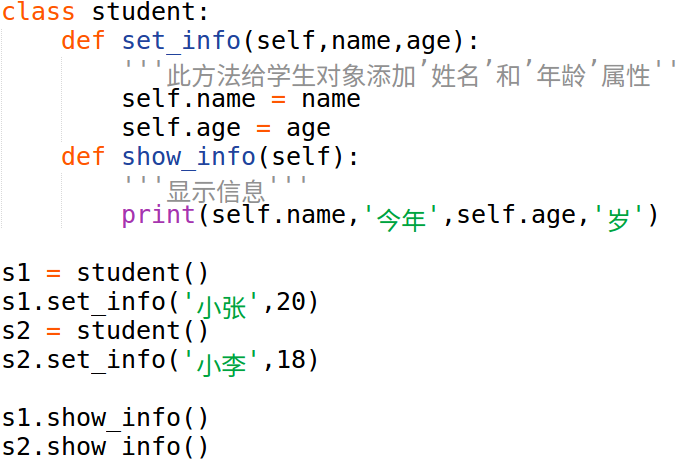
s1.set\_info(‘小张’,20)

s2 = student()

s2.set\_info(‘小李’,18)

s1.show\_info()#小张 今年 20 岁

s2.show\_info()



3.3删除属性

用del语句可以删除一个对象的实例变量

语法：

del 对象.实例名

实例：

class Cat:

pass

c1 = Cat() #创建变量

c1.color = ‘白色’ #添加属性

print(c1.color)

del c1.color #删除属性

print(c1.color) #属性错误

问题：

class Dog:

pass

dog1 = Dog() #正确

dog2 = Dog(‘京巴’,’白色’) #错误

4 初始化方法

作用：

对新创建的对象添加实例变量（属性）或相应的资源

语法格式：

class 类名（继承列表）:

def \_\_init\_\_ (self [,形参列表]) :

语句块

说明：

1.初始化方法必须为\_\_init\_\_不可改变

2.初始化方法会在构造函数创建实例后自动调用，且将实例自身通过第一个参数self传入\_\_init\_\_方法

3.构造函数的实参将通过\_\_init\_\_方法的形参列表传入\_\_init\_\_方法中

4.初始化方法内部如果需要返回则只能返回None

示例：



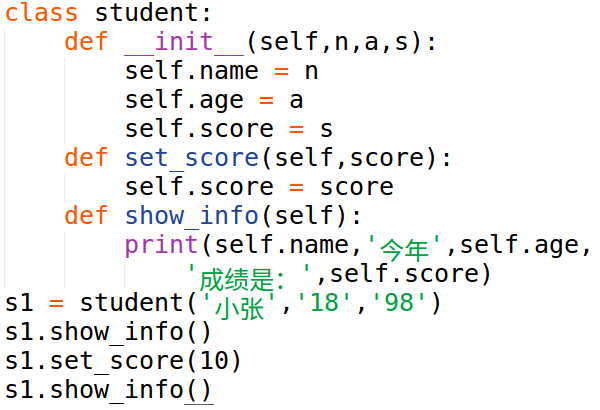
练习：

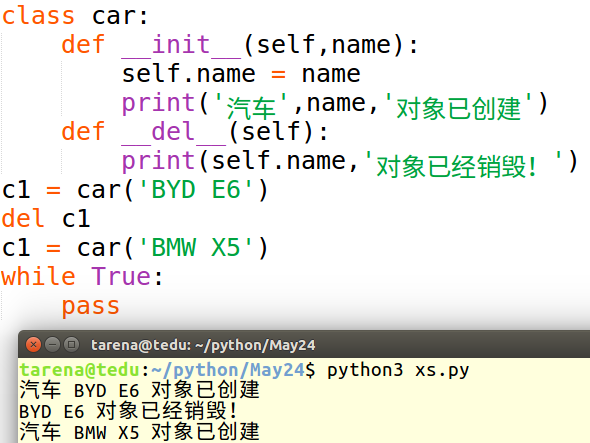
修改前的student类：

1）为该类添加初始化方法，实现在创建对象时自动设置‘姓名’，‘年龄’，‘成绩’属性

2）添加set\_score方法能为对象修改成绩信息

3）添加show\_info方法打印学生对象的信息





5析构方法

语法：

class 类名（继承列表）:

def\_\_del\_\_(self) :

语句块

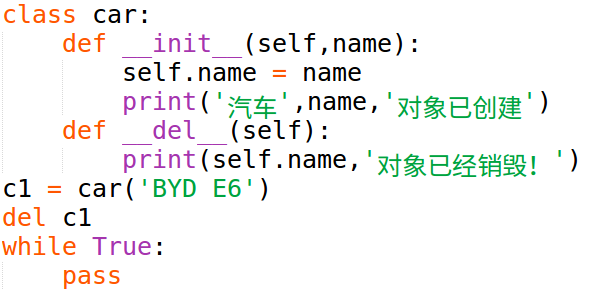
说明：

析构方法在对象销毁时被自动调用

作用：

清理此对象所占用的资源

示例：



python不建议在析构方法内做任何事，因为对象销毁时间难以确定

6 预置实例（对象）属性

\_\_dict\_\_属性

此属性绑定一个存储此实例自身实例变量的字典

示例：

class Dog:

pass

dog1 = Dog()

print(dog1,\_\_dict\_\_)

dog1.kinds = ‘京巴’

print(dog1.\_\_dict\_\_)

\_\_class\_\_属性

此属性用来绑定创建此实例的类

作用：

可以借助此属性来访问创建实例类

示例：

class Dog:

pass

dog1 = Dog()

dog2 = Dog()

dog3 = dog1.\_\_class\_\_() #创建dog1的同类对象

7 面向对象的综合示例

有两个人：

姓名：张三 姓名：李四

年龄：35 年龄：8

行为： 1）教别人学东西teach

2）赚钱

3）借钱

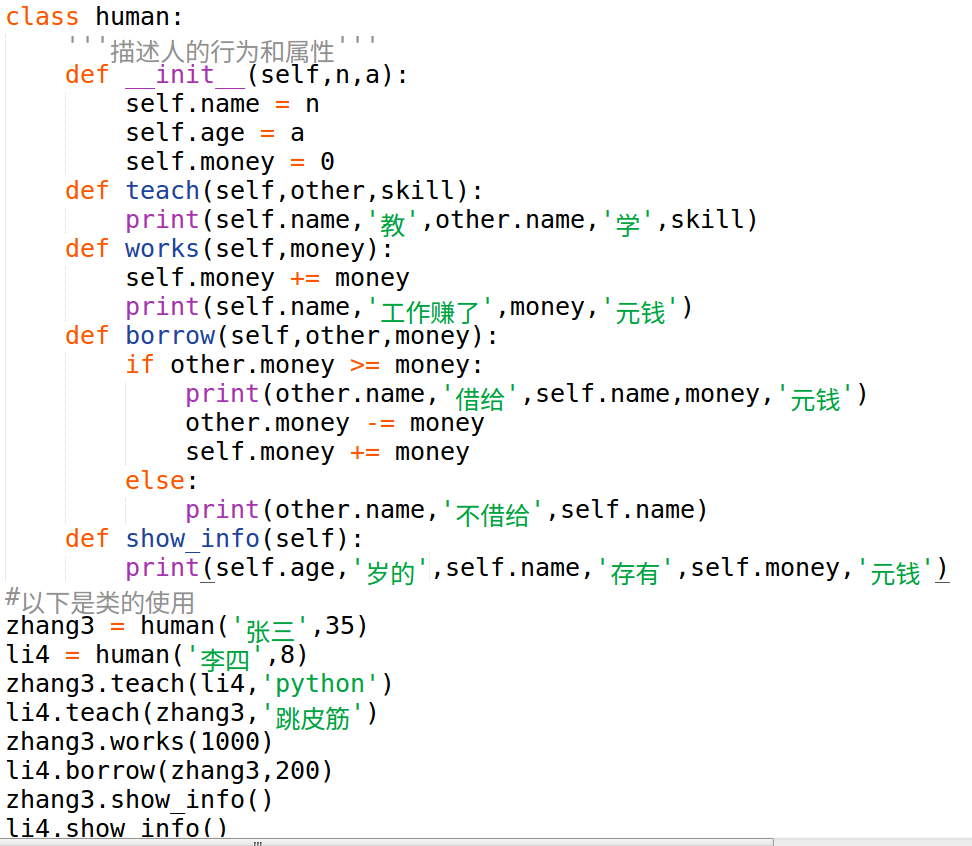
事情：

张三 教 李四 学python

李四 教 张三 学 跳皮筋

张三 上班赚了 1000元钱

李四 向张三借了 200元



用于类的函数：

isinstance(obj,class\_or\_tuple)返回这个对象obj是否某个类class或某些类的实例，如果是则返回True，否则返回False

type(obj)返回对象的类型

示例：

class Dog:

pass

class Cat:

pass

animal = Dog()

isinstance(animal,Dog)

isinstance(animal,Cat)

isinstance(animal,(Cat,int,list))

isinstance(animal,(Cat,int,Dog))

练习：

1.修改之前学生管理系统：

原学生数据使用字典存储，现改为用对象进行存储

要求自定义student类来封装学生的信息和行为

类 file : student.py

class student:

def\_\_init\_\_(self,n,a,s)

self.name = n

……

May25

1 类变量class variable（也叫类属性）

类变量是类的属性，此属性属于类

作用：

用来记录类相关的数据

说明：

类变量可以通过类直接访问

类变量可以通过类的实例间接访问

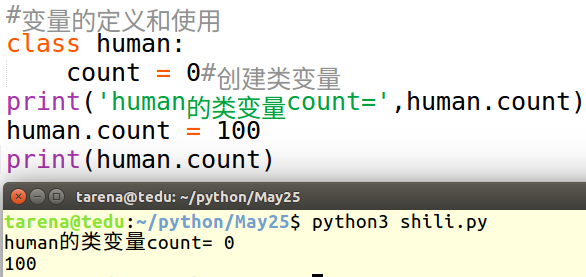
类变量可以通过此类的实例的\_\_class\_\_属性间接访问

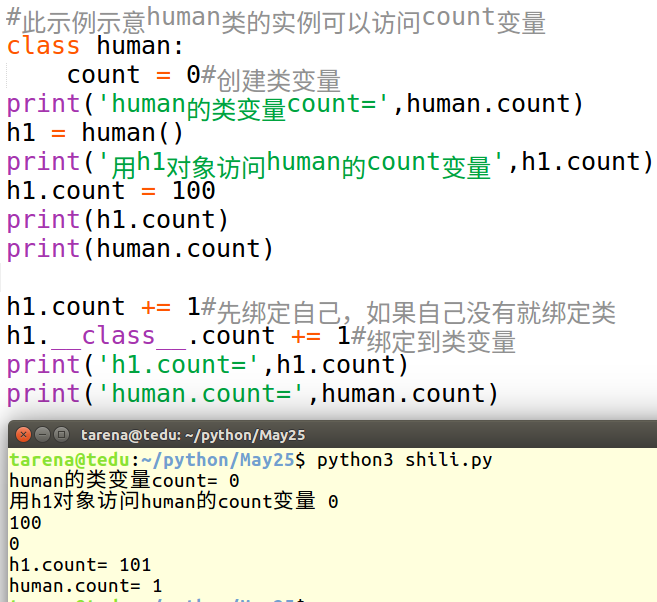
示例：

class human:

count = 0#创建类变量

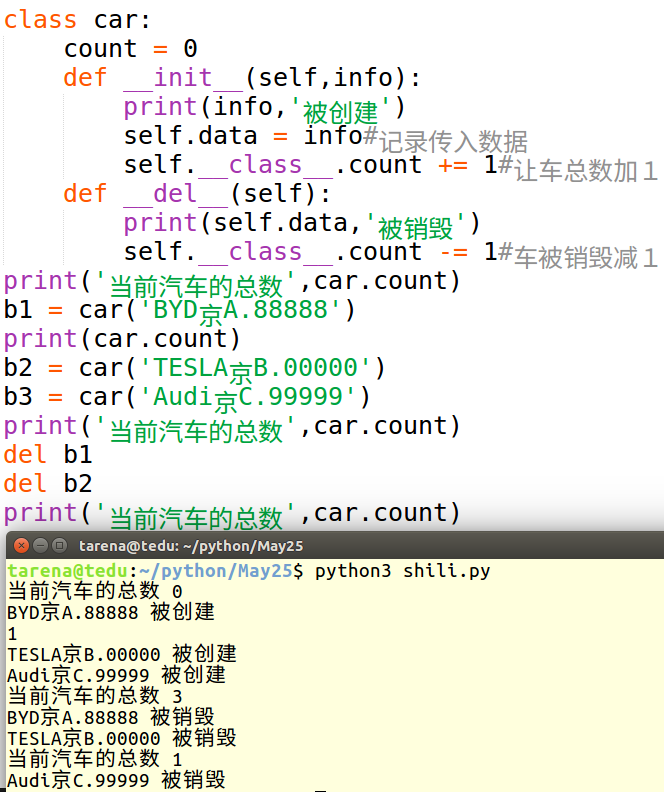
详见：





1.1 类变量的应用案例

用类变量记录对象的个数



2 类的文档字符串

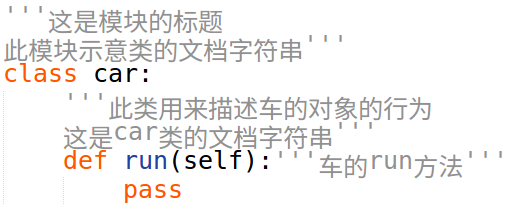
类内第一个没有复制给任何变量的字符串是类的文档字符串

说明：

类的文档字符串用类\_\_doc\_\_属性可以访问

类的文档字符串可以用help()函数查看

示例：

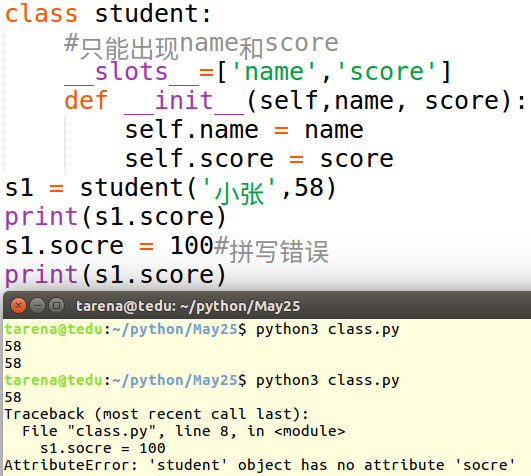


3 类的\_\_slots\_\_列表

作用：

限定一个类的实例只能有固定的属性（实例列表），通常为防止错写属性名而发生的运行错误

示例：



说明：

含有\_\_slots\_\_列表的类创建的实例对象没有\_\_dict\_\_属性，即此实例不用字典来保存对象的属性（实例变量）

1.对象内可以有：实例变量，实例方法

2.类内可以有：类变量，类方法

4 类方法 @classmethod

类方法是描述类的行为方法，类方法属于类

说明：

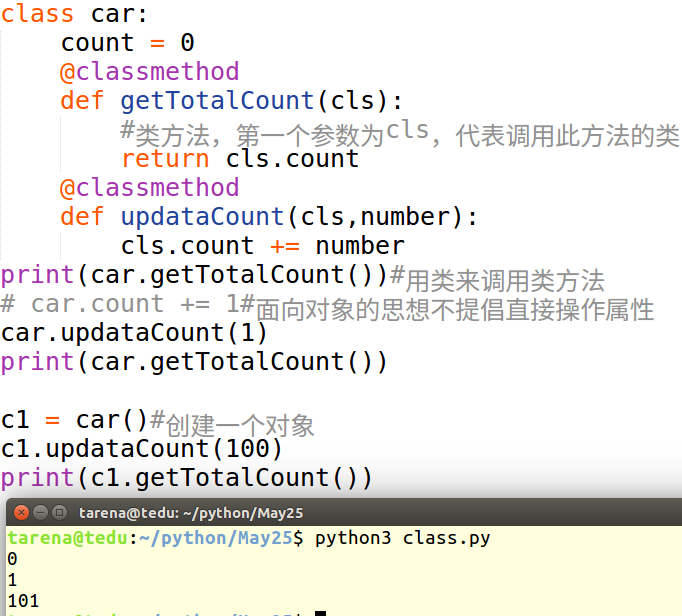
1.类方法需要用@classmethod装饰器定义

2.类方法至少有一个形参，第一个形参用于绑定类，约定写为’cls’

3.类和该类的实例都可以调用类方法

4.类方法不能访问此类创建的实例的属性（只能访问类变量）

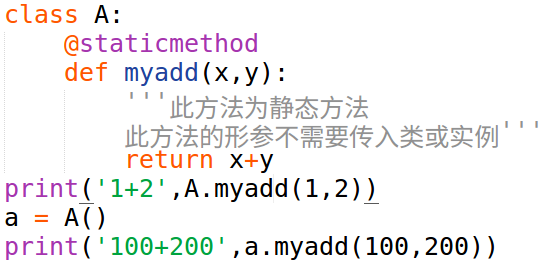
示例：



5 静态方法 @staticmethod

静态方法不属于类，也不属于类的实例，它相当于定义在类内普通函数，只是它的作用域属于类

示例：



练习：

1.用类来描述一个学生的信息（可以修改之前写的student类）

class student:

#靠自己

要求该类对象用于存储学生的信息：姓名，年龄，成绩。将这些对象存于列表中，可以任意添加和删除学生信息

1）打印出学生的个数

2）打印出所有学生的平均成绩

（建议用类变量存储学生的个数，也可以把列表当做类变量）

6 继承inheritance和派生derived

什么是继承/派生

1.继承是指从已有的类中衍生出新类，新类具有原类的行为，并能扩展新的行为。

2.派生就是从一个已有类中衍生（创建）新类，在新类上可以添加新的属性和行为。

继承和派生的目的：

继承是延续旧类的功能

派生是为了在旧类的基础上添加新的功能

作用：

1.在继承派生机制，可以将一些共有功能加载基类中，实现代码的共享

2.在不改变基类的基础上改变原有功能。

继承/派生的名词

基类（base class），超类（super class），父类（father class）

派生类（derived class）/子类（child class）

6.1 单继承

语法：

class类名（基类名）：

语句块

说明：

单继承是指派生类由一个基类衍生出来的类

示例：



6.2 继承说明

任何类都是直接或间接的继承自object类

object类是一切类的超类（祖类）

类的\_\_base\_\_属性

\_\_base\_\_属性用来记录此类的基类

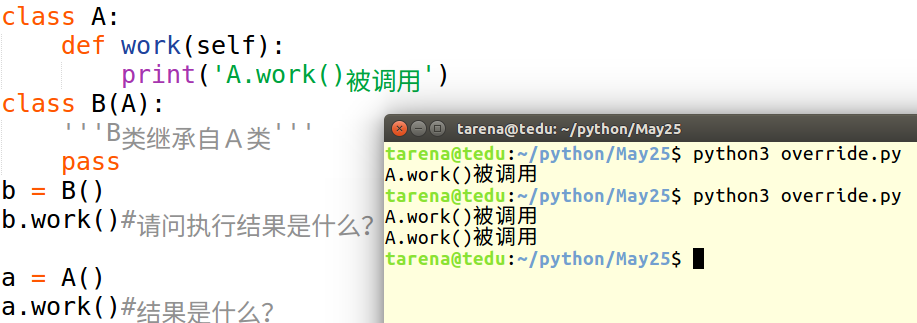
（寻找父类）

7 覆盖override

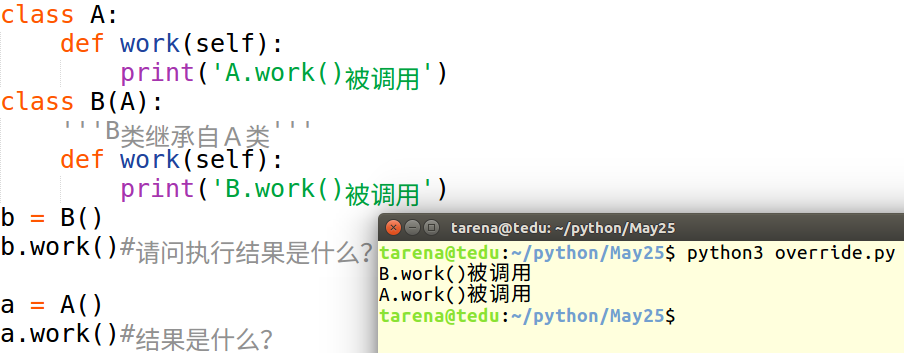
什么是覆盖？

覆盖是指在有继承关系的类中，子类中实现了与基类同名的方法，在子类实例调用该方法时，实例调用的是子类中的覆盖版本的方法，这种现象叫做覆盖。

示例：

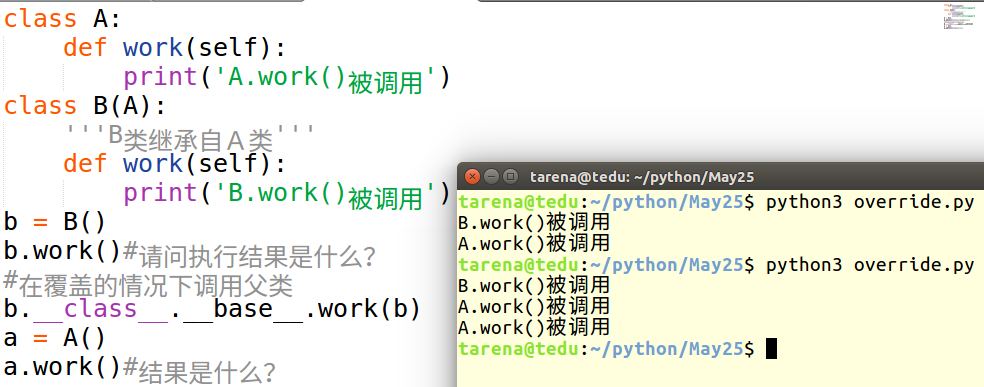


----------------------------------覆盖的例子----------------------------------



---------------------------------问题--------------------------------------------

当覆盖发生时，子类对象能否访问父类中的方法？



6.2 子类对象显式调用基类方法的方式

基类名.方法名（实例，实际调用传参）

super函数

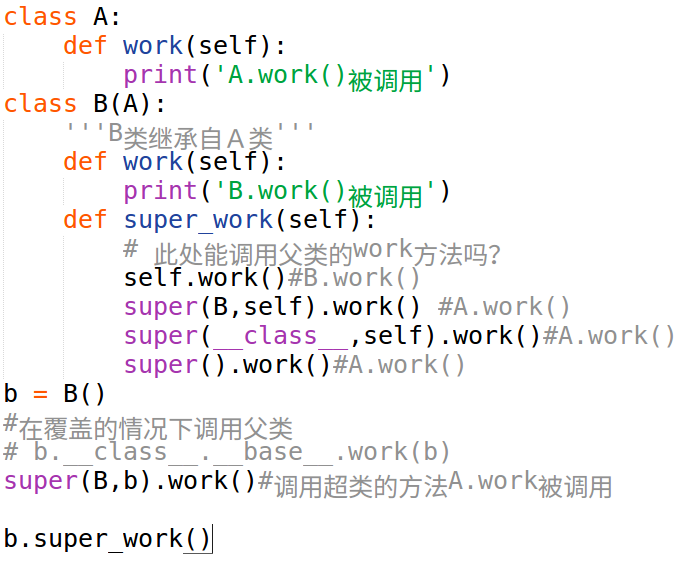
super(type,obj)返回绑定超类的实例

super( ) 返回绑定超类的实例，等同于：

super(\_\_class\_\_,实例方法的第一个参数)

（必须在方法内调用）

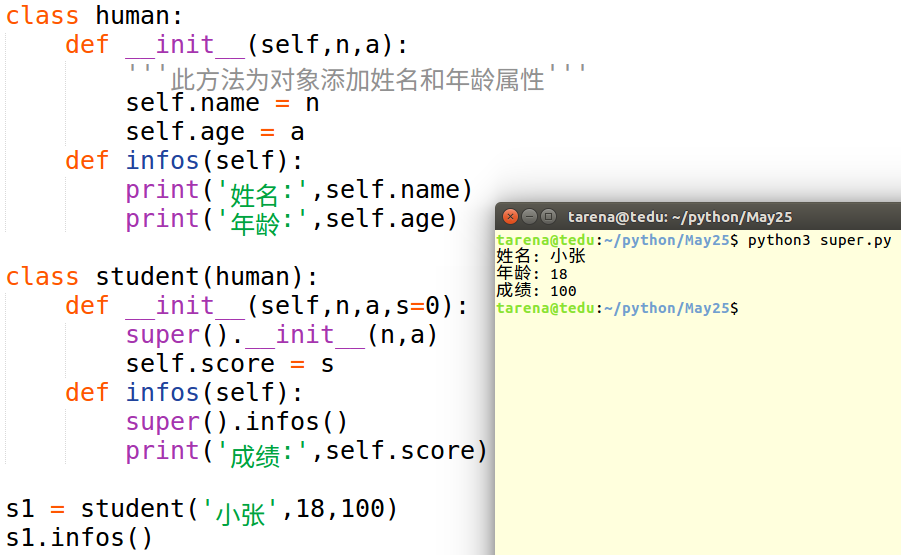
示例：



显式调用基类的初始化方法：

当子类中实现了\_\_init\_\_方法时，基类（父类）的\_\_init\_\_方法并不会被自动调用，此时需要显式调用

示例：



练习：

写一个类Bicycle类，有run方法，调用时，显示骑行里程km

class Bicycle:

def run(self,km):

print(‘自行车骑了’,km,’公里’)

再写一个类 EBicycle类，在Bicycle类的基础上添加电池电量volume属性，

有两个方法：

1.file\_charge(self,vol)用来充电,vol为电量

2.run(km)方法每骑行10km消耗电量1度，同时显式当前电量，当电量耗尽时，则调用Bicycle的run方法骑行

class EBicycle(bicycle):

……

b = Bicycle()

b.run(10)#自行车骑行了10公里

e = EBicycle(5)

e.run(10) #电动自行车骑行了10公里

e.run(100) #电动自行车骑行了40公里，自行车骑行了60公里

b.fill\_charge(10)

b.run(100)

May28

1 用于类的函数

issubclass(cls,class\_or\_tuple)

判断一个类是否继承自其它的类，如果此类cls是class或tuple中的一个派生子类则返回True，否则返回False

直间的关系为 或的关系

示例：

class A:

pass

class B(A):

pass

class C(B):

pass

issubclass(C,(A,B)) #True

issubclass(C,(int,str)) #False

2 封装enclosure

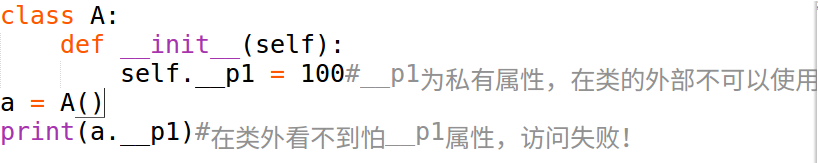
封装是指隐藏类的实现细节，让使用者不用关系这些细节

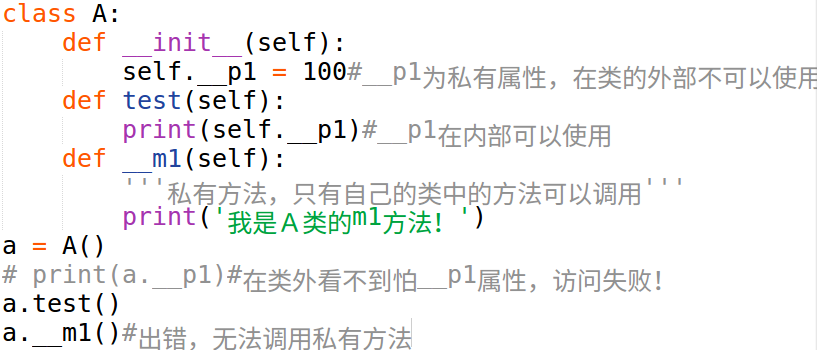
封装的目的是让使用者尽可能少的用实例变量（属性）进行操作

私有属性：

python类中，以双下划线“\_\_”开头，不以双下划线结尾的标识符为私有成员，在类的外部无法直接访问

示例：





3 多态polymorphic

字面意思“多种状态”

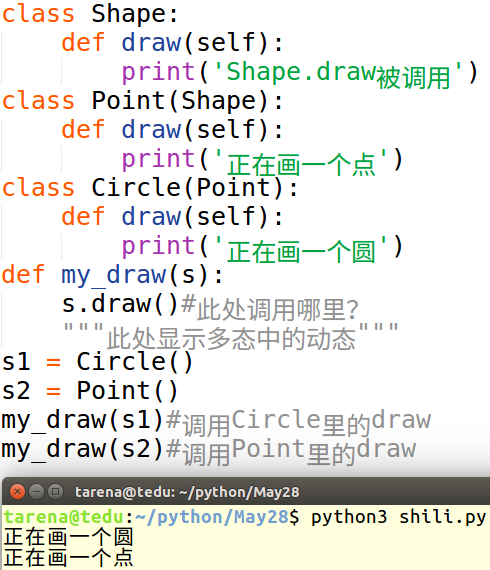
多态是指在继承/派生关系中，调用基类对象的方法，实际能调用子类的覆盖版本方法的现象叫做多态。

说明：

多态调用的方法与对象相关，不与类相关

python的全部对象都只有‘运行时状态（动态）’，没有‘C++/JAVA’里的“编译时状态（静态）”

示例：



面向对象的编程语言的特征

继承 封装 多态

如：

C++ / Java / Pyyhon / Swift / C#

4 多继承multipe inheritance

多继承是指一个子类继承自两个或两个以上的基类

语法：

class 类名(基类名1，基类名2，……):

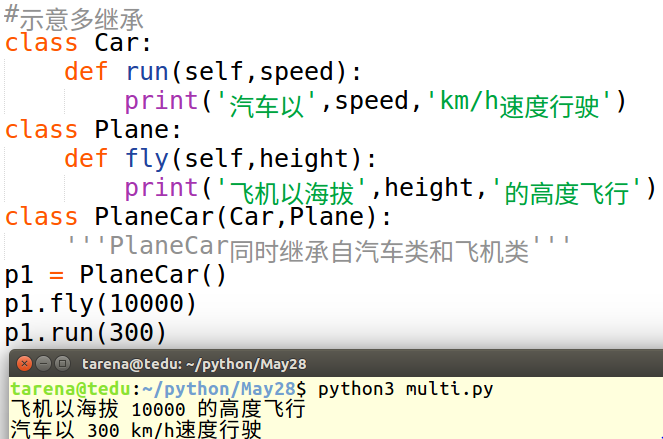
语句块

说明：

1.一个子类同时继承多个父类，父类中的方法可以同时被继承下来

2.如果两个父类中有同名的方法，而在子类中又没有覆盖此方法时，调用结果难以确定

示例：

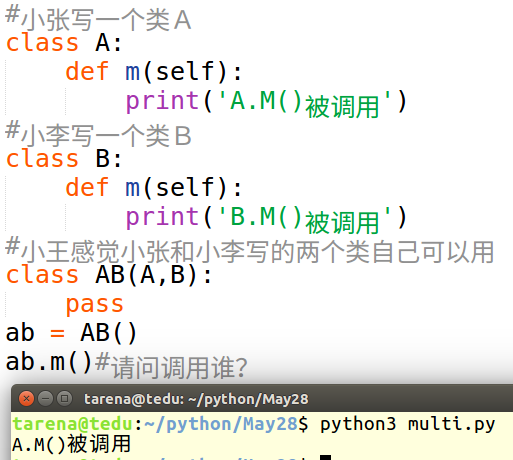


4.1 多继承的缺陷

标识符（名字空间冲突问题）

要谨慎使用多继承

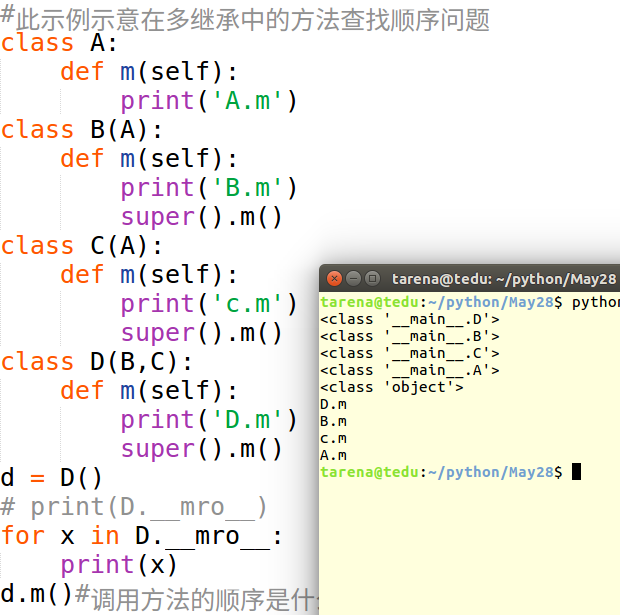
示例：



继承的MRO（Method Resolution Order）问题

类内的\_\_mro\_\_属性用来记录继承方法的查找顺序----------------------------------

示例：



查找顺序是根据\_\_mro\_\_来查找的

练习：

已知list列表类中没有insert\_head方法，写一个自定义的类Mylist，继承自list类，在Mylist类内添加

class MyList(list):

def insert\_head(self,value):

’’’以下功能自己实现,将Value插入到列表开始处’’’

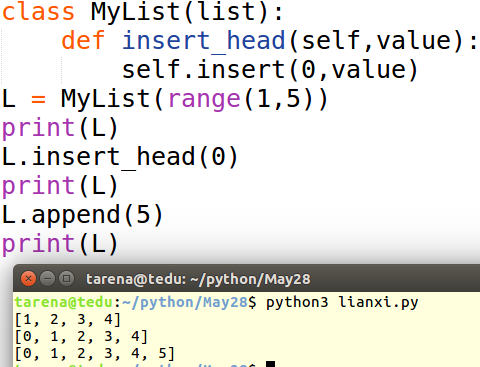
如：

L = MyList(range(1,5))

print(L)#[1,2,3,4,5]

L.insert\_head(0)

print(L)#[0,1,2,3,4]



5 函数重写override

重写是在自定义的类内添加相应的方法，让自定义的类生存对象（实例）像内建对象一样进行内建的函数操作

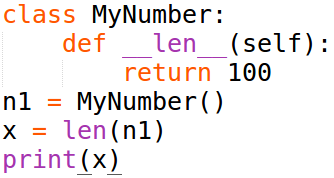
5.1 对象转字符串函数重写

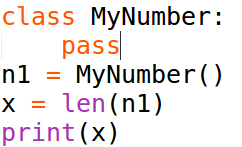
repr(obj) (给机器看的)返回一个能代表此对象的表达式字符串，通常：

eval(repr(obj)) == obj

str(obj) 通过给定的对象返回一个字符串（这个字符串通常是给人看的）

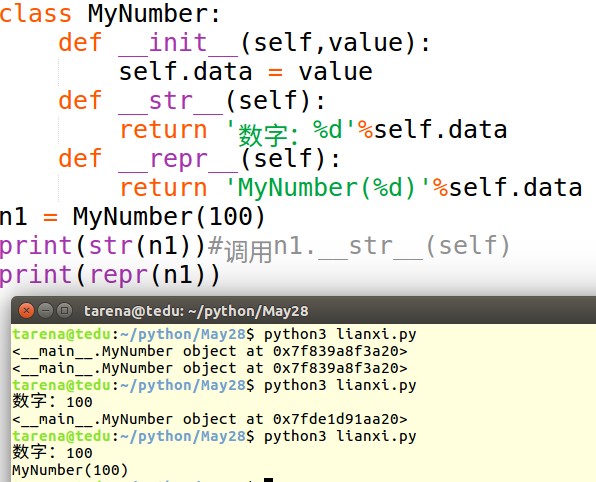
问题：



错误！

需要定义一个方法！

示例：



5.2 对象转字符串函数重写方法

repr( )函数重写的方法：

def \_\_repr\_\_(self):

return能够表达self内容的字符串

str( )函数重写的方法：

def \_\_str\_\_(self):

return人能看懂的字符串

说明：

1.str(obj)函数优先调用obj.\_\_str\_\_( )方法返回字符串

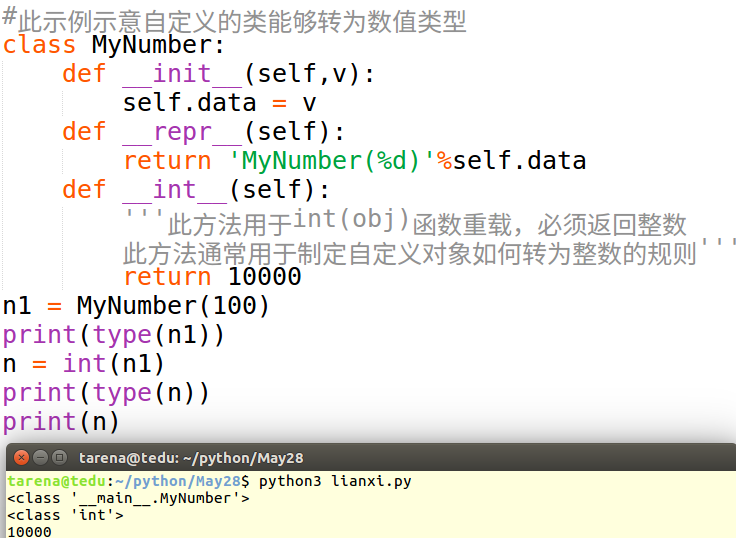
2.如果obj没有\_\_str\_\_( )方法，则调用obj.\_\_repr\_\_( )方法返回字符串

3.如果obj没有\_\_repr\_\_( )方法，则调用object类的\_\_repr\_\_实例方法显示<xxx>格式的

5.3数值转换函数重写方法

|  |  |
| --- | --- |
| def \_\_complex\_\_(self) | complex(obj)函数调用 |
| def \_\_int\_\_(self) | int(obj)函数调用 |
| def \_\_float\_\_(self) | float(obj)函数调用 |
| def \_\_bool\_\_(self) | bool(obj)函数调用 |

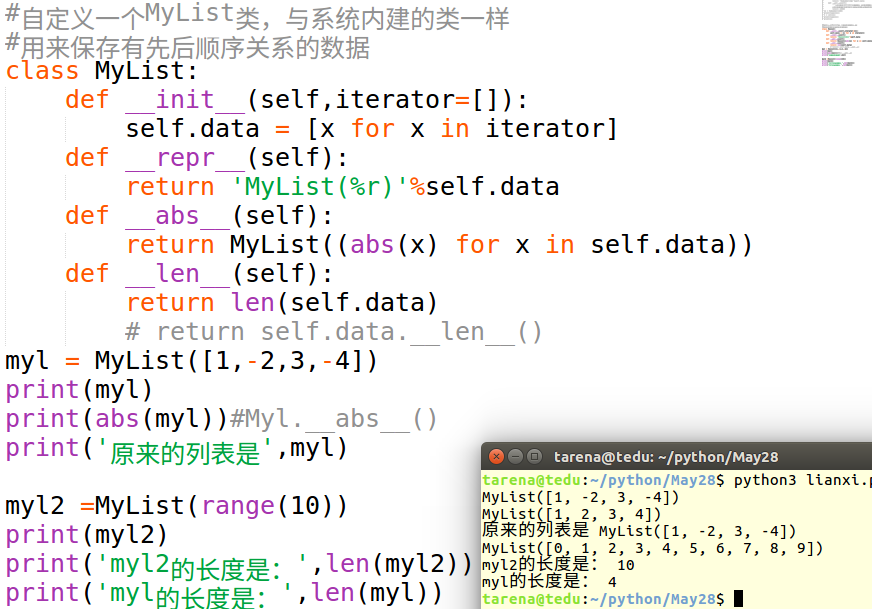
示例



5.3内建函数重写方法

|  |  |
| --- | --- |
| \_\_abs\_\_ | abs(obj) |
| \_\_len\_\_ | len(obj) |
| \_\_reversed\_\_ | reversed(obj) |
| \_\_round\_\_ | round(obj) |

示例：



**5.3.1 布尔测试函数的重写**

格式:

def \_\_bool\_\_(self):

……

作用：

用于bool（obj）函数取值

用于if语句真值表达式中

用于while语句真值表达式中

说明：

1.优先调用\_\_bool\_\_方法取值

2.如果不存在\_\_bool\_\_方法，则用\_\_len\_\_( )方法取值后判断值是否为零，如果不为零则返回True，否则返回False

3.如果再没有\_\_len\_\_方法，则直接返回True

6 迭代器（高级）

什么是迭代器

可以通过next(it)函数取值的对象就是迭代器

6.1迭代器协议

迭代器协议是指对象能够使用next函数获取下一项数据，在没有下一项数据时触发一个StopIterator来终止迭代的约定

实现方法：

类内需要有\_\_next\_\_(self)方法来实现迭代器的协议

语法形式：

class MyIterator

def \_\_next\_\_self:

迭代器协议的实现

return数据

6.2可迭代对象

是指能用iter(obj)函数返回迭代器的对象（实例）

可迭代对象内部一定要定义\_\_iter\_\_(self)方法来返回迭代器

可迭代对象的语法形式：

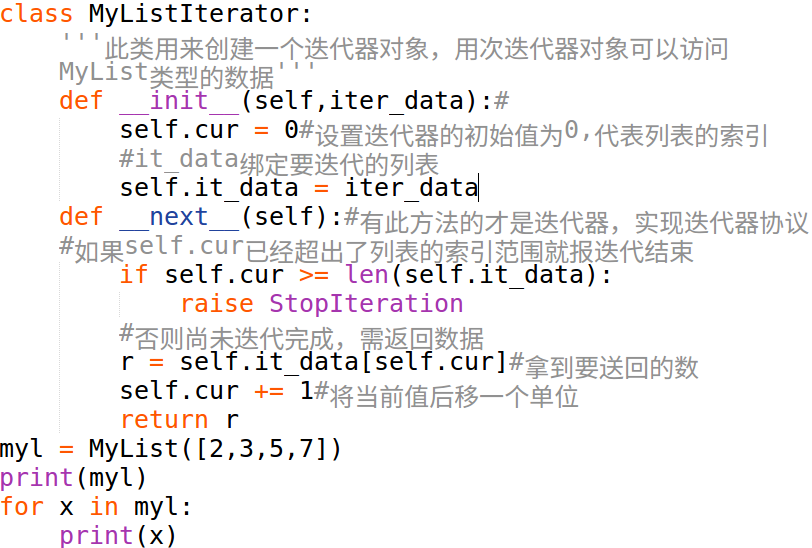
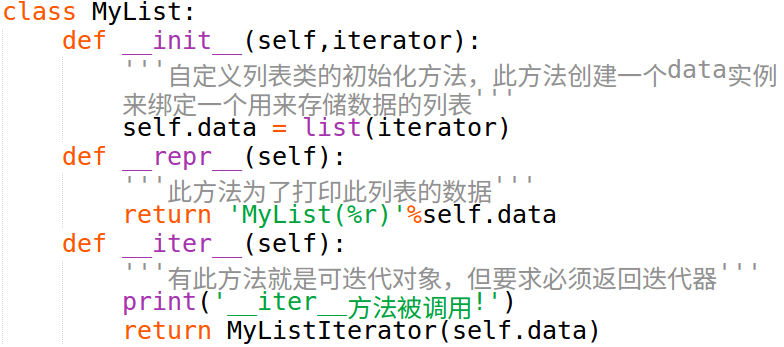
class MyIterable:

def \_\_iter\_\_(self):

语句块

return 迭代器

示例：



练习：

1.把学生信息管理系统中的信息：姓名，年龄，成绩。改为私有属性，不让此类以外的函数访问，实现封装。（需要修改Student类）

2.定一个类，Fibonacci实现迭代器协议，此类的对象可以作为可迭代对象生成相应的斐波那契数

1 1 2 3 5 8 ……

class Fibonacci:

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

……

def \_\_iter\_\_(self):

def \_\_next\_\_(self):

实现如下操作：

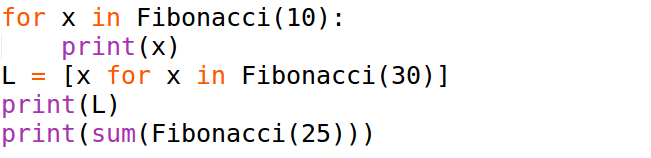
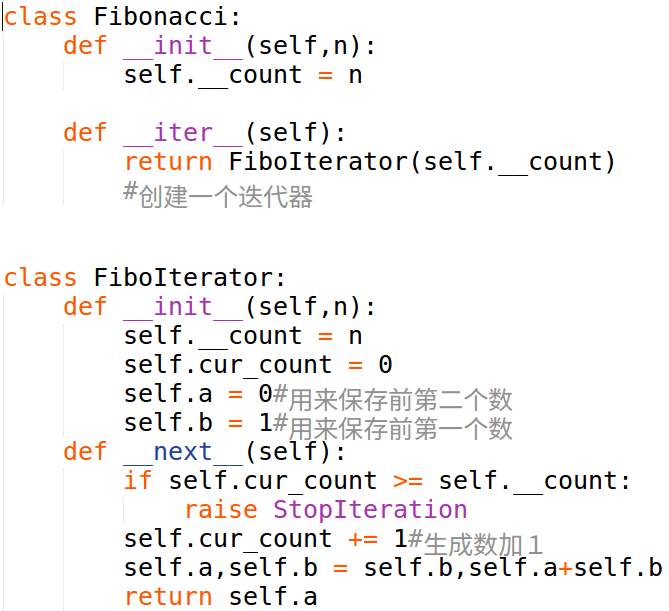
for x in Fibonacci(10):

print(x)

L = [x for x in fibonacci(30)]

print(sum(Fibonacci(25)))

(需要实现迭代器协议)



May29

1 异常（高级）

with语句

语法：

with 表达式 [as 变量1]，表达式2 [as 变量2]：

语句块

作用：

使用于对资源进行访问的场合，确保使用过程中不管是否发生异常，都会执行必须的‘清理’操作，并释放资源

如:

文件使用后自动关闭，线程中锁的自动获取和释放等

示例:



说明：

能够用于with语句进行管理的对象必须是环境管理器

2 环境管理器

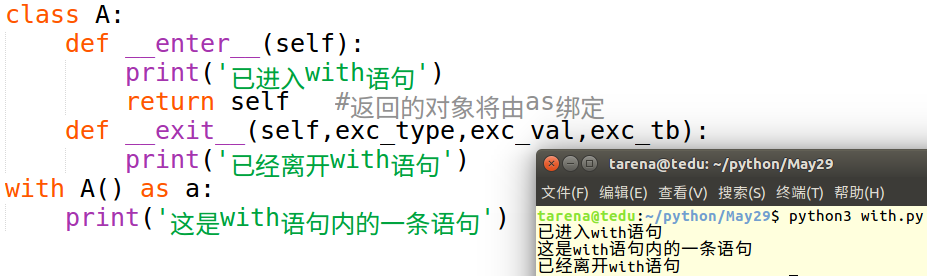
1.类内有\_\_enter\_\_和\_\_exit\_\_实例方法的类被称为环境管理器

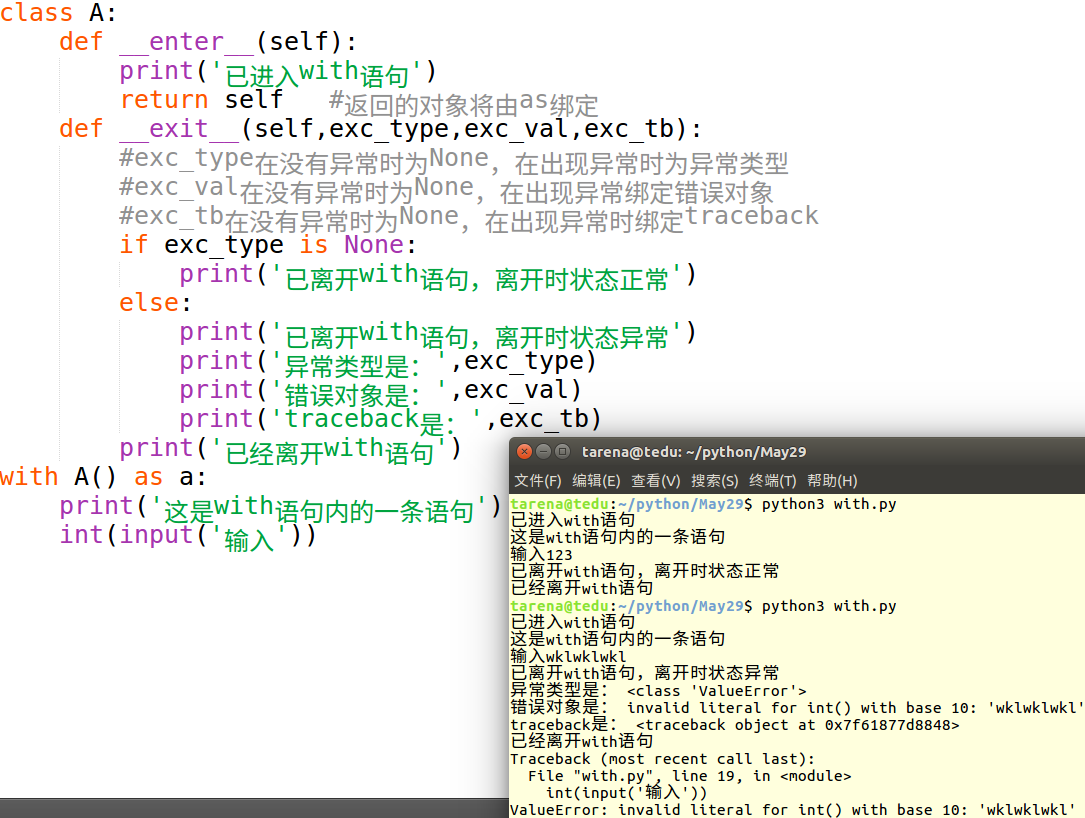
2.能够用with语句管理的对象必须是环境管理器

3.\_\_enter\_\_方法将在进入with语句时被调用，并返回由as变量管理的对象

4.\_\_exit\_\_将在离开with语句时被调用，且可以用参数来判断在离开with语句时是否有异常发生并作出相应的处理

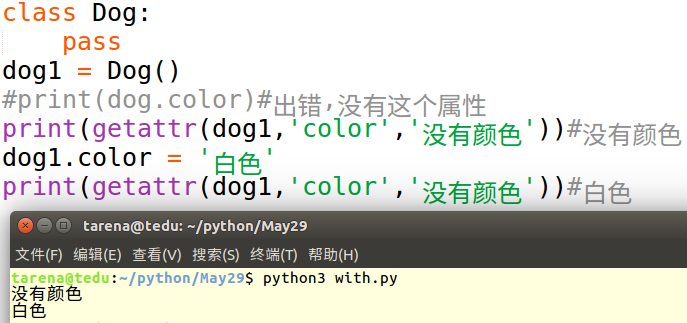
示例：





3 对象的属性管理

| **函数** | **说明** |
| --- | --- |
| getattr(obj, name[, default]) | 从一个对象得到对象的属性；getattr(x, 'y') 等同于x.y; 当属性不存在时,如果给出default参数,则返回default,如果没有给出default 则产生一个AttributeError错误 |
| hasattr(obj, name) | 用给定的name返回对象obj是否有此属性,此种做法可以避免在getattr(obj, name)时引发错误 |
| setattr(obj, name, value) | 给对象obj的名为name的属性设置相应的值value, set(x, 'y', v) 等同于 x.y = v |
| delattr(obj, name) | 删除对象obj中的name属性, delattr(x, 'y') 等同于 del x.y |



练习：

写一个Car类，属性有：

颜色color

品牌brand

class Car:

def \_\_init\_\_(self,c,b):

……

添加一个方法

def get\_car\_attr(self,attr\_name):

此方法用于获取对象的属性，如果属性名attr\_name在此对象内不存在则返回None

示例：

c1 = Car(‘黑色’,’Benz’)

v = c1.get\_car\_attr(‘color’)

if v is None:

print(‘没有颜色属性’)

else:

print(‘颜色是：’,v)



4 运算符重载

让自定义的类生成的对象（实例）能够使用运算符进行操作

作用：

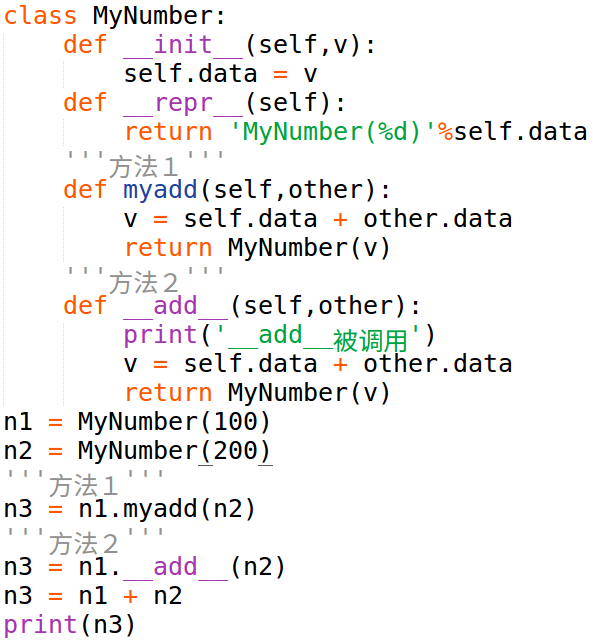
让自定义的类的实例像内建对象一样运行运算符操作

让程序简洁易读

让自定义的对象，将运算符赋予新的运算规则

算数运算符的重载：

|  |  |  |
| --- | --- | --- |
| \_\_add\_\_(self,rhs) | self + rhs | 加法 |
| \_\_sub\_\_(self,rhs) | self - rhs | 减法 |
| \_\_mul\_\_(self,rhs) | self \* rhs | 乘法 |
| \_\_truediv\_\_(self,rhs) | self / rhs | 除法 |
| \_\_floordiv\_\_(self,rhs) | self // rhs | 地板除 |
| \_\_mod\_\_(self,rhs) | self % rhs | 求余 |
| \_\_pow\_\_(self,rhs) | self \*\* rhs | 幂 |
| rhs(right hans side)右手边 | | |



练习：

实现两个自定义的列表相加：

class MyList:

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

self.data = list(iterable)

L1 = MyList([1,2,3])

L2 = MyList([4,5,6])

L3 = L1 +L2

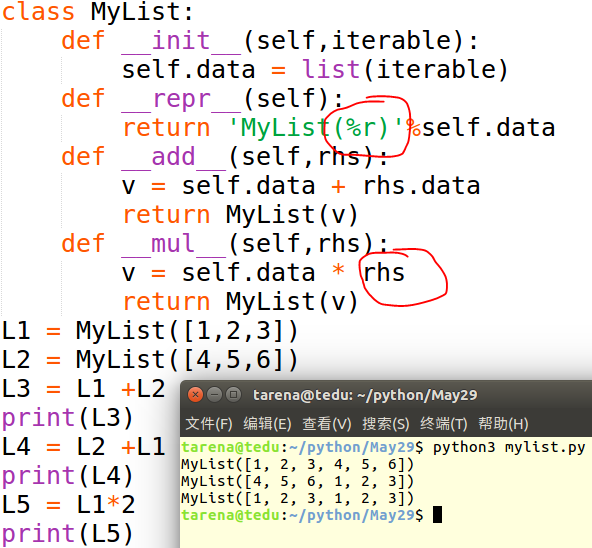
print(L3)

L4 = L2 +L1

print(L4)

L5 = L1\*2

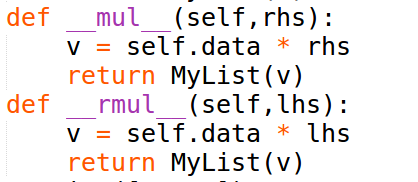
print(L5)



4.1 反向运算符的重载

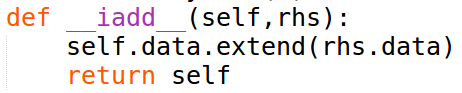
当左手边的类型为内建类型，右手边为自定义类型时，要实现运算必须用以下方法重载

|  |  |  |
| --- | --- | --- |
| \_\_radd\_\_(self,rhs) | Lhs +self | 加法 |
| \_\_rsub\_\_(self,rhs) | Lhs - self | 减法 |
| \_\_rmul\_\_(self,rhs) | Lhs \* self | 乘法 |
| \_\_rtruediv\_\_(self,rhs) | Lhs / self | 除法 |
| \_\_rfloordiv\_\_(self,rhs) | Lhs // self | 地板除 |
| \_\_rmod\_\_(self,rhs) | Lhs % self | 求余 |
| \_\_rpow\_\_(self,rhs) | Lhs \*\* self | 幂 |



4.2 复合赋值运算符的重载

|  |  |
| --- | --- |
| \_\_iadd\_\_(self,rhs) | rhs +=self 加法 |
| \_\_isub\_\_(self,rhs) | rhs -= self 减法 |
| \_\_imul\_\_(self,rhs) | rhs \*= self 乘法 |
| \_\_itruediv\_\_(self,rhs) | rhs /= self 除法 |
| \_\_ifloordiv\_\_(self,rhs) | rhs //= self 地板除 |
| \_\_imod\_\_(self,rhs) | rhs %= self 求余 |
| \_\_ipow\_\_(self,rhs) | rhs \*\*= self 幂 |



4.3 比较运算符的重载

|  |  |  |
| --- | --- | --- |
| \_\_lt\_\_(self,rhs) | self<rhs | 小于 |
| \_\_le\_\_(self,rhs) | self<=rhs | 小于等于 |
| \_\_gt\_\_(self,rhs) | self>rhs | 大于 |
| \_\_ge\_\_(self,rhs) | self>=rhs | 大于等于 |
| \_\_eq\_\_(self,rhs) | self==rhs | 等于 |
| \_\_ne\_\_(self,rhs) | self!=rhs | 大等于 |
| 注：比较运算符通常返回True或False | | |

4.4 位运算符

|  |  |  |
| --- | --- | --- |
| \_\_invert\_\_(self) | self | 取反（一元运算符） |
| \_\_and\_\_(self,rhs) | self&rhs | 位与 |
| \_\_or\_\_(self,rhs) | Self | rhs | 位与 |
| \_\_xor\_\_(self,rhs) | self^rhs | 位与 |
| \_\_lshift\_\_(self,rhs) | self<<rhs | 左移 |
| \_\_rshift\_\_(self,rhs) | self>>rhs | 右移 |
| 反向位运算符 \_\_rand\_\_(self,lhs) lhs & self | | |
| 复合赋值位运算符重载 \_\_iand\_\_(self,rhs) self =& rhs | | |

4.5 一元运算符的重载

|  |  |  |
| --- | --- | --- |
| \_\_neg\_\_(self) | -self | 负号 |
| \_\_pos\_\_(self) | +self | 正号 |
| \_\_invert\_\_(self) | ~self | 取反 |

重载方法:

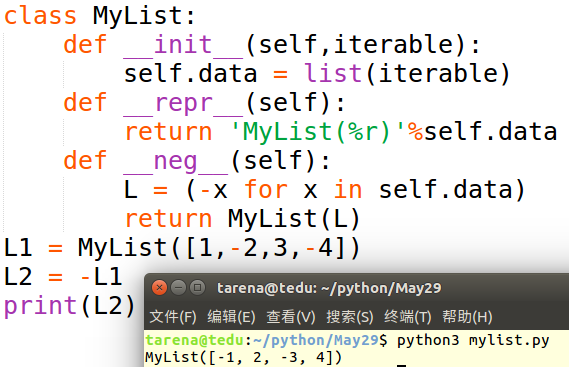
class 类名:

def \_\_xxx\_\_(self):

……

提示：运算符重载不能改变运算符的优先级

示例：



python类名最好用驼峰命名法：

MyList MyRange 大驼峰（所有单词首字母大写，其余小写）

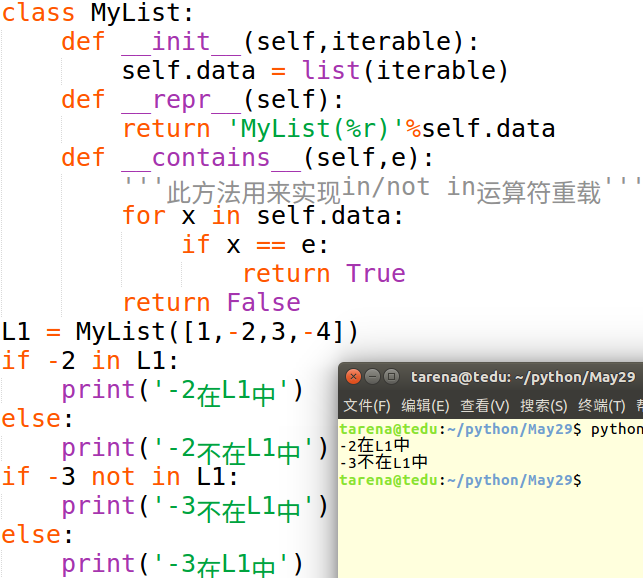
getStudentAge 小驼峰（第一个单词首字母小写，其他首字母大写）

4.6 in/not in4.5运算符的重载

重载方法：

def \_\_contains\_\_(self,e): #e in self成员运算

……



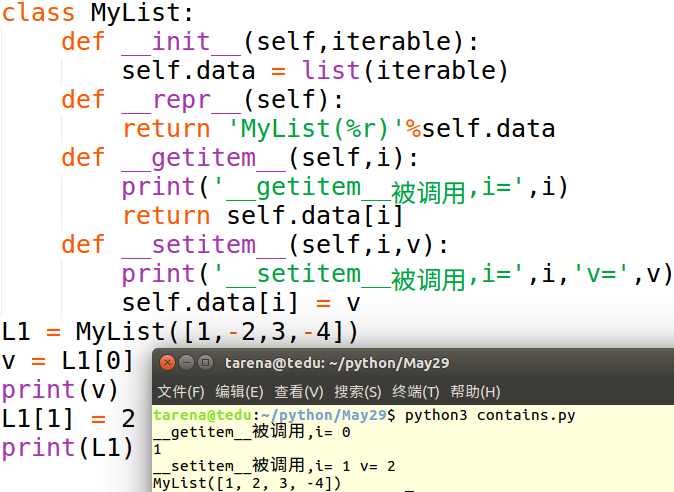
4.7 索引和切片运算符的重载

|  |  |  |
| --- | --- | --- |
| \_\_getitem\_\_(self,i) | x=self[i] | 索引/切片取值 |
| \_\_setitem\_\_(self,i) | self[i]=v | 索引/切片赋值 |
| \_\_delitem\_\_(self,i) | del | del语句删除索引等 |

作用：

让自定义的类型的对象能够支持索引和切片操作

示例：



4.8 slice函数

作用：

用于创建一个slice切片对象，此对象存储一个切片的起始值，终止值和步长

slice(start,stop = None,step = None)创建一个切片对象

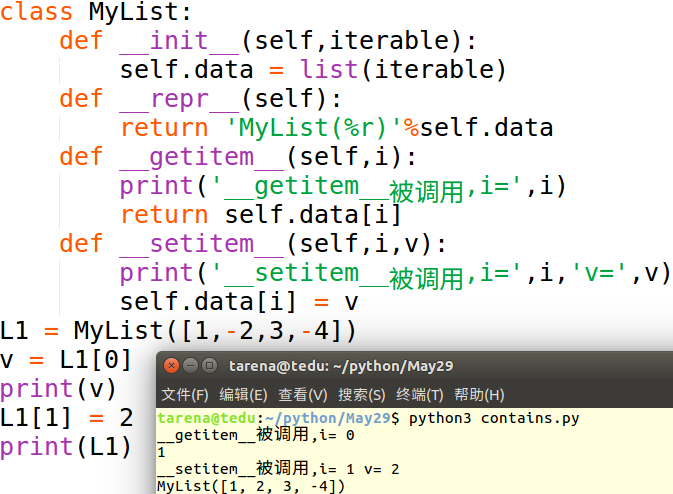
slice的对象的属性：

s.start 切片起始值,默认为None

s.stop 切片终止值,默认为None

s.step 切片步长,默认为None

示例：



练习：

实现有序集合OrderSet类，能实现两个集合的交集&，并集 | ，补集 -，

对称补集^,==,!=,in/not in 集合操作（要求集合内用list存储数据）

s1 = OrderSet([1,2,3,4])

s2 = OrderSet([3,4,5])

print(s1&s2)

print(s1| s2)

print(s1^s2)

if OrderSet([1,2,3]) !=OrderSet([1,2,3,4]):

print(‘不相等’)

if s2 == OrderSet(3,4,5):

print(‘s2和OrderSet(3,4,5)相等’)

if 2 in s1:

print(‘2 in s1’)