**北京图灵学院Python全系列教程全栈工程师（基础知识篇）**

**--------井关小熊学习笔记**

主讲人：刘大拿老师

记录汇总（版权拥有人）：井关小熊

版权所有：转发需注明来源

前面讲了一段如何安装软件的方法，此处不作为笔记内容。

2-1课 python概述

讲解了python的历史及特征，强调了学习方法。

2-2课 变量定义

变量就是代号（定义变量就是赋值）

年龄 = 18岁

（变量名） = （数值）

赋值的三种方式：

1，age = 18

2，age1 = age2 = age3 = 18

3，age1 , age2 , age3 = a1 , a2 , a3

2-3课 变量1

从python中学习菜单单词

File（文件） edit（编辑） view（视图） insert（插入）

cell（单元格） Kernel（系统） widgets（工具）

Code（代码） markdown（标记） raw NBConvert（原始NBConvert）

heading（标题）

变量类型（共6种）

-数字 Number

-字符串 Str ……………………属于不可变对象，赋值运算后地址就变了，只是同名而已（在后面的赋值运算中会有涉及）

-列表 list [ ] ………属于可变对象，赋值运算后地址不变（在后面的赋值运算中会有涉及）

-元组 tuple ( )

-字典 dict

-集合 set { }

Jupyter 笔记本的快捷方式

代码 --- Esc→B

标记（笔记） ---Esc→M

编辑模式 ----Enter

常见数字分类

-整数

·没有小数部分

·包含正数、负数、0

·二进制

--只有0和1数字

--以0b开头的01串

·八进制

---以0o开头的0-7构成的串

·十六进制

---以0x开头的0-9，a-f构成的串

-浮点数

·就是小数量

·案例：

3.1415

3.

0.4

.4

-科学计数法

·定义与数学相同

·写法就是e后面跟整数用来表示10的倍数

-复数

·与数学定义一致

·复数的虚部用j/J表示

·例如：

5+4j

4j

(4j)

2-4 课 变量2

2-5 课 变量3

-布尔值

·布尔值就是用来表示真假的值

·只有两个值： True/False

·在python中，布尔值可以当数字使用

---True=1 ,False=0

☆ 如果数字用来当布尔值使用时，0=False,其余当作True

-字符串

--表达文字信息的内容，比如“我爱王晓静”

--形式上是引号引起来的一段内容

--引号包括

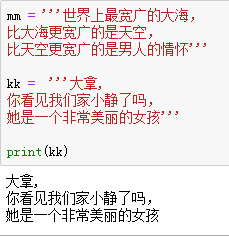
单引号

双引号

三引号

-- 单双引号含义一致

-- 三引号用来表示多行信息，如果不是打印内容的话，那就是注释的作用。



-None类型

--表示没有，通常用来占位

--比如返回，用来表示返回一个空

2-6 课 运算符-算数运算符

2-7 课 运算符-结束

-表达式

--由一个或者几个数字或者变量或者运算符 合成第一行代码

--通常返回一个结果#

-运算符

--由一个以上的值经过一系列的运算得到新值的过程就叫做运算

--用来操作运算的符号叫运算符

--运算符的分类

----算数运算符

☆实际操作注意点：

（+，-，\*，/ // %）

除法有三种，正常除法(/)， 地板除(//)， 取余除(%)

（7\*\*2 表示7的平方）

(字符串\*数字，表示重复字符串多少次的意思)

----比较或者关系运算符

对两个内容进行比较的运算符

结果一定是布尔值，即 Ture/False

☆实际操作注意点：

(等于==; 不等于!=; 大于>; 大于等于>=; 小于<; 小于等于<=)

----赋值运算符

-意思是 把一个值放到变量里面去

☆实际操作注意点：

(赋值符号=)

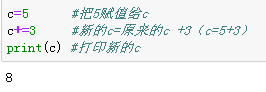
方法有三种

a=9（含义：把9赋值给a）

a=b=9（含义：把9赋值给b，b再赋值给a）

a,b=1,2（含义：把2赋值给吧，把1赋值给a）

赋值的缩写：



所有的数学运算符都可以缩写

-=; \*=; /=; //=; %=; \*\*=

----逻辑运算符

-对布尔类型变量或者值进行运算的符号

-and: 逻辑与（看做乘法）

-or: 逻辑或（看做加法）

-not: 逻辑非 （）

☆ 关键难点

数字转换为布尔值的时候：0=false,其余为ture;

布尔值转换为数字的时候：ture=1, false=0

----位运算符

------成员运算符

-用来检测一个值或者变量是否在某个集合里面

-得到的回复是 ture 或者false

-in：成员运算符

-not in：不在里面的意思

--身份运算符

-用来确定两个变量是否是同一变量

※值一样，未必是变量相同，变量是指内存地址

（-5~256之间，值相同的话代表变量也相同，以外的数值相同不代表变量相同）

-得到的回复是 ture 或者false

-is： 变量运算符

-is not： 不是

-运算符优先级的问题

--小括号具有最高优先级

\*\*指数（最高优先级）

~+- 按位运算

2-8课 分支结构

-程序结构

--程序三种结构

顺序

循环

分支

-分支结构

--分支结构的基本语法

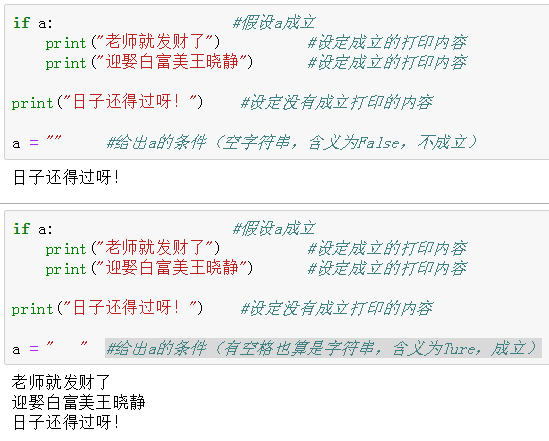
If 条件表达式：

语句1

语句2

语句3

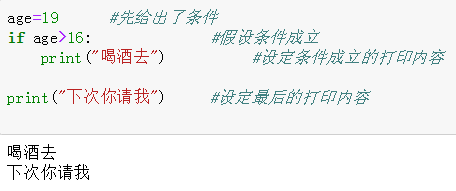
………………



a= “ ”中即使是空格也算是有字符串，代表条件成立Ture

a= “”中为空字符串时，代表条件不成立False

注意：只有空字符串为False，其余都为Ture



条件表达式就是计算结果必须为布尔值的表达式

表达式后面的冒号不能少

注意if 后面的出现的语句，如果属于if语句块，则必须同一个缩进等级

条件表达式结果为Ture,那么继续执行if后面的缩进语句块。

-双向分支

If 条件表达式：

语句1

语句2

else:

语句3

语句4

语句5



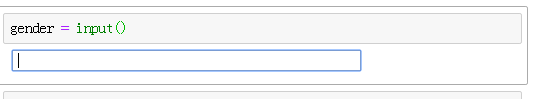
input 的作用

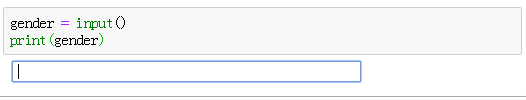
---在屏幕上输入括号内的字符串

---接受用户输入的内容并返回到程序

----input返回的内容一定是字符串类型

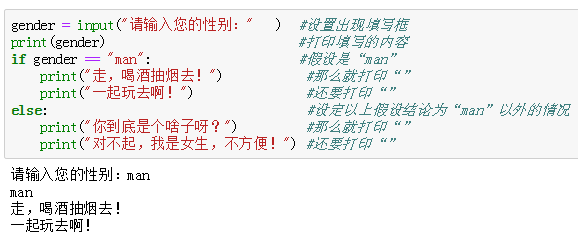
如下3种方式供参考：

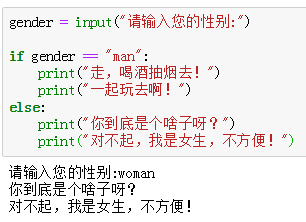






有input填写框的 if……else…… 格式

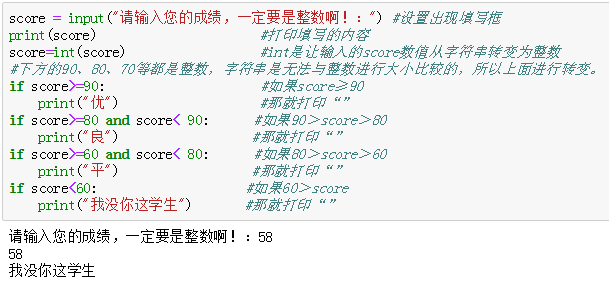


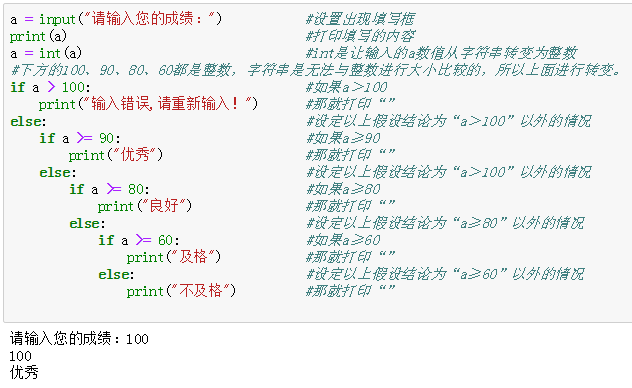


int 的作用

---是将字符串转变为整数的函数

一般可以配合填写框进行使用





2-9课 循环结构

字符串转变为整形的方式：score=int(score)中的int（）就是转变的意思。

-多路分支

if 条件表达式：

语句1

elif 条件表达式：

语句1

elif 条件表达式：

语句1

else：(注意：else后面不能有条件表达式，否则会报错)

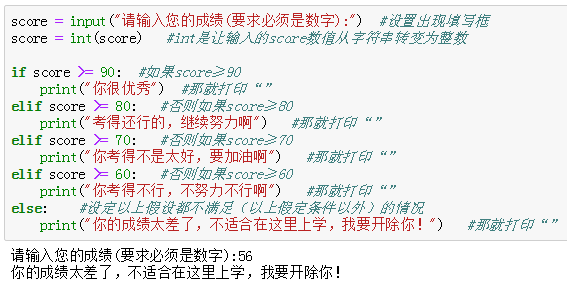
语句1

………………..

------elif 可以有好多个，根据实际情况

------else 可选

-------多路分支最多只会执行一种情况



-循环语句

分类

1. for

② while（当……时候）

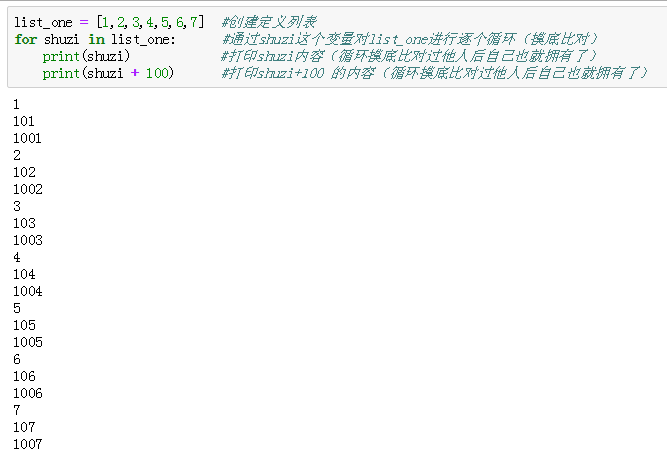
1、关于for循环

For变量in序列：

语句1

语句2

…………..



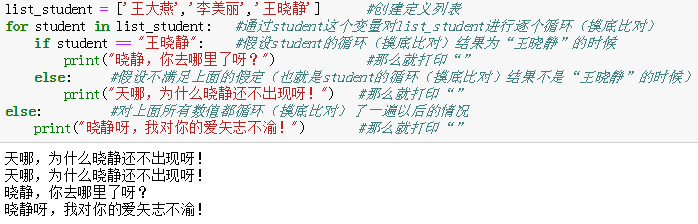
·for-else语句

-for…..in…….

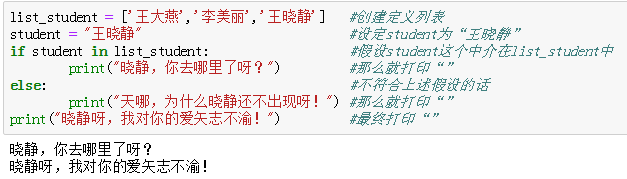
if……

else……

else……



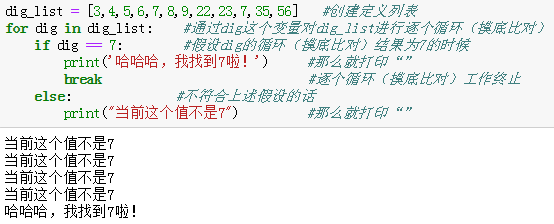
个人尝试：不希望一个一个的判断，希望是总判定，出具在还是不在的结论，但可惜的是没有用到于for……in…….循环，如下：



-break, continue, pass

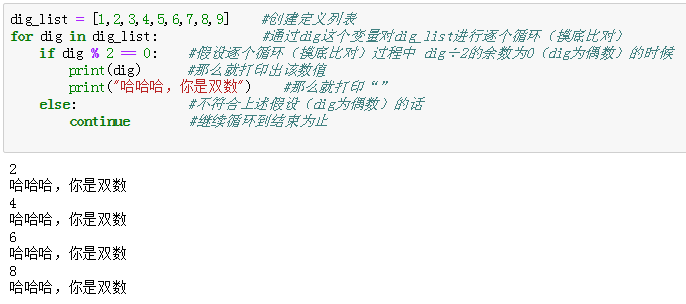
·break：表示无条件结束整个循环，简称循环猝死。

找到第一个目标后就立即终止，不再继续循环（摸底比对）。



·continue:表示继续循环到列表结束为止。

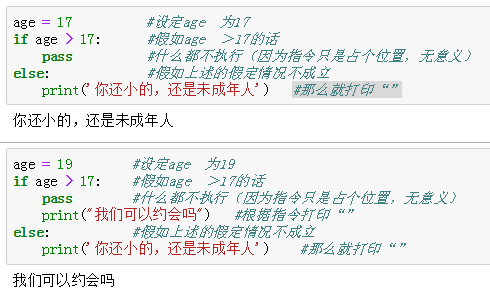
下方示例是要求在数字1~10中寻找出偶数（这种情况肯定是要循环到底为止）



也可以用奇数排除来找偶数，如下示例：



·pass:表示就是个占位符，并没有跳过的意思，就是蹲着一个废物，后面可以继续执行。

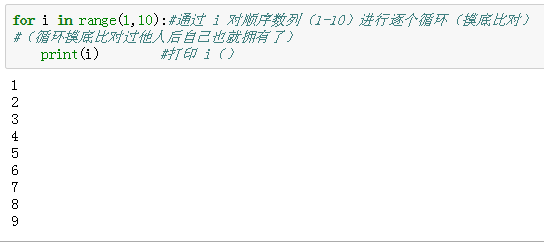


range函数

-生成有序数列

-生成数字队列可以定制

例：



2-10课 While循环

- 一个循环语句

- while后面先给出循环的条件

-当循环条件超出后，循环就会停止，适应于不知道具体循环次数。



2-11课 函数（1）

-函数是代码的一种组织形式

-函数应该能完成一项特定的工作，而且一般一个函数值完成一项工作

-有些余元，分“函数”和“过程”两个概念，通俗解释是有返回结果的叫函数，无返回结果的叫过程。

函数的使用：

·函数的使用需要先定义，def就是表示定义函数的意思；

顺序如下：

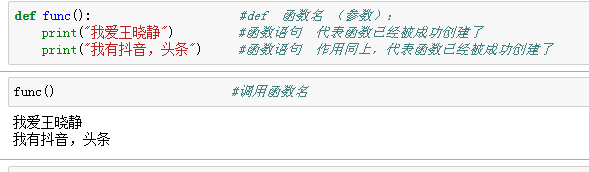
def 函数名 （参数）：

函数语句 #以上代表函数已经被成功创建了

函数名（）#调用函数名

……………………………………………………………………..

返回值



·使用函数，俗称调用。

调用函数的方法：

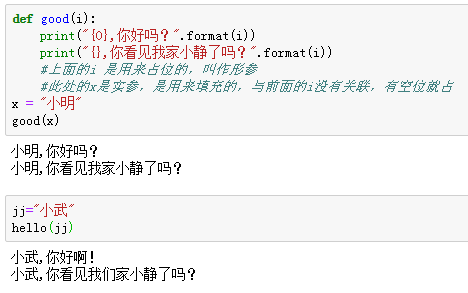
-直接写出函数名，后面的小括号不能省略，括号内的内容根据情况。

2-12课 函数（2）

参数：负责给函数传递一些必要的数据或者信息

·形参（形式参数）：在函数定义的时候用到的参数，没有具体值，只是一个占位符号；

·实参（实际参数）：在调用函数的时候输入的值。



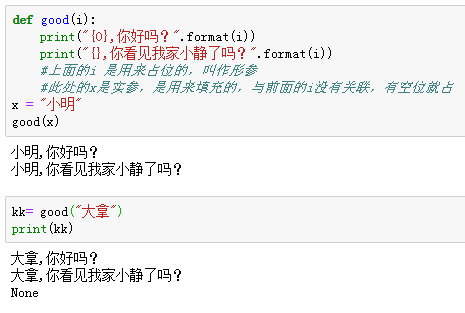
--返回值：调用函数的时候一个执行结果

使用return返回结果

如果没有值需要返回，推荐使用return none，表示函数结束

函数一旦执行return，则函数立即结束

如果函数没有return关键字，则函数默认返回None



备注：上面format 表示格式化的意思，后面2-15课 有讲解。

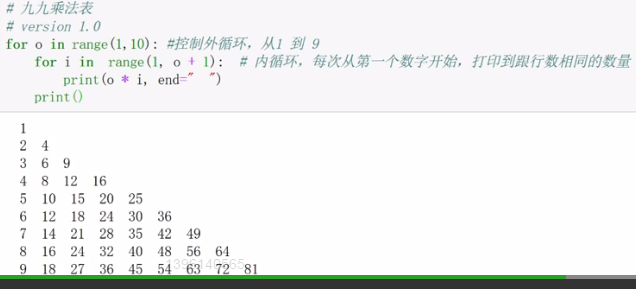
创建《九九乘法表》练习

乘法口诀是循环，想到了for…….in…… 和while 这两种循环方式，区别是有明确的循

环次数，所以用for……..in……。

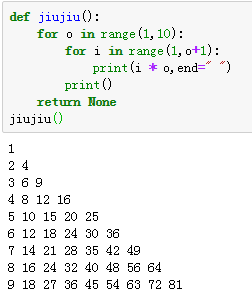
横着循环一次（1~9），还要竖着循环一次（1~9），就存在嵌套循环。

循环结束指示 是用空格来表示。



2-13课 函数3

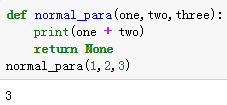
把上一课创建的《九九乘法表》改变成一个函数形式，为了今后能方便赋用（赋值调用）



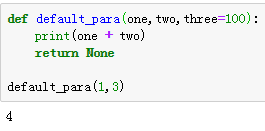
这里有一段不太懂！！！

参数详解:

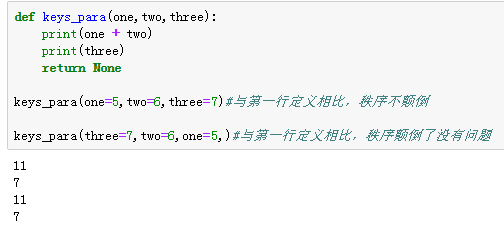
-普通参数/位置参数



-默认参数(有时候想偷懒，想少写参数，于是用默认的参数)



-关键字参数



-收集参数

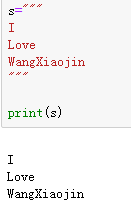
老师累了，没有讲。

2-14课 Str模块1

字符串

-表示文字信息

-用单引号，双引号，三引号括起来



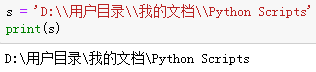
转义字符

-用一个特色的方法表示出一系列不方便写的内容，比如回车键，换行键，退格键。

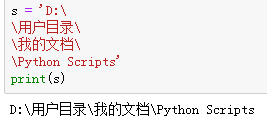
-借助反斜杠字符，一旦字符串出现反斜杠，则反斜杠后面一个或者几个字符已经不是原来的意思了，进行了转义。

-不同系统对换行操作有不同的表示。

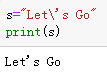




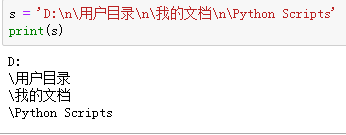
**☆，不要与行尾的反斜杠符号作用搅混了，行尾的为续行符（单个反斜杠表示此行未结束，出于美观，需要下一行继续）**，如下示例：



可以使用单双引号嵌套



另外\n在Windows下表示回车换行的指令（**☆，不要与行尾的反斜杠符号作用搅混了，行尾的为续行符**）。



格式化

-把字符串按照一定格式进行打印或者填充

-格式化的分类

-传统的格式化

-format

·传统格式化

字符串的传统格式化方法

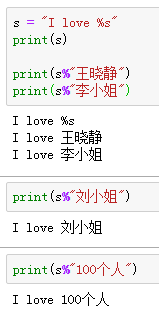
-使用%进行格式化

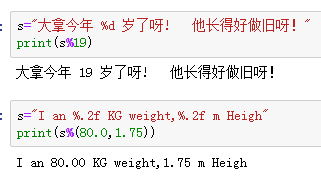
-%也叫占位符

%s（表示字符串类型占位）

%d（表示整数类型占位）

%f（表示二进制浮点占位）





2-15课 str2

·format格式化

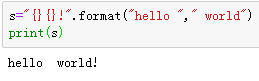
-使用函数形式进行格式化，代替以前的百分号

-不用指定位置，按顺序读取的

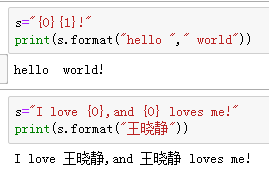
例1：



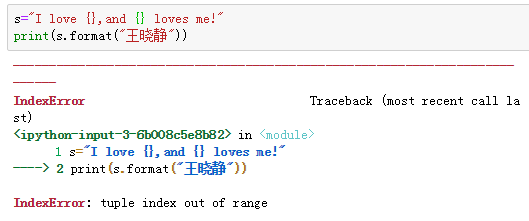
例2：



例3：设定指定位置



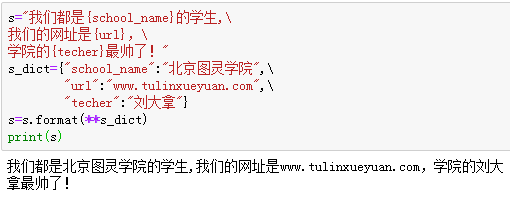
不设定位置就会有数值不够分产生报错现象，如下：



例4，使用命名参数：



例5，通过字典设置参数，进行解包：



2-16课 str内置函数和列表

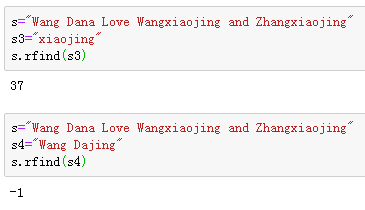
str内置函数

-字符串查找类，find，index，islower

-find:查找字符串中是否包含一个子串。

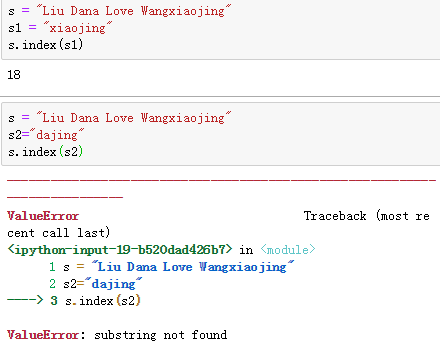
返回正数值表示找到的值所在位置，返回负值表示没有找到。

rfind：表示从右开始查找；lfind：表示从左开始查找。



-index:查找字符串中是否包含一个子串。

返回正数值表示找到的值所在位置，返回报错表示没有找到。



判断类函数

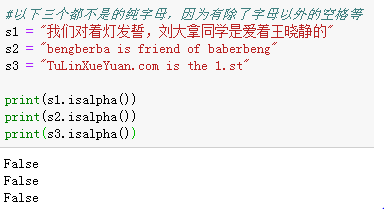
-此类函数的特点是一般都用is开头，比如islower

· isalpha:判断是否是字母，需要注意两点：

-此函数默认的前提是字符串至少包含一个字符，如果没有，同样返回False

-汉字被认为是alpha，所以此函数不能作为区分英语字母还是汉字的标识，区分中英文请使用unicode码。

-注意使用区别，防止被坑。



内容判断类

-starswith/endswith:是否以xxx开头或者结尾

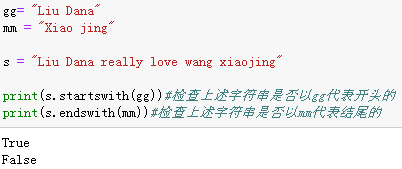
-检测某个字符串是否以某个子串开头，常用三个参数

·suffix：被检查的字符串，必须有

·start：检查范围的开始范围

·end：检查范围的结束范围

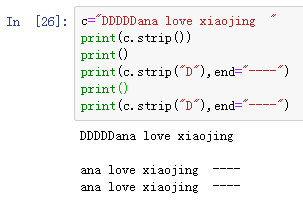
·lower\supper：判断是否为大小写（字符串中只要有一个字母不符合，都会被判定为False；中文字符不参与判断，会直接跳过。）



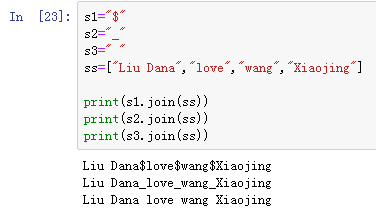
操作类函数

-format：格式化用

-strip：这个函数主要作用是删除字符串两边的空格，其实这个函数允许你去定义删除字符串两边的哪个字符，只不过如果不指定的话默认是空格。同样还有lstrip和rstrip，此处r和l分别表示左右，默认空格。需要注意的是，此处的删除不是删除一个，是指从头开始符合条件的连续字符。



-join：把设定的内容插入到另外设定内容的分隔符中



List列表

-表示一组有序数据做成的序列

-数据具有先后顺序

-数据可以不是一类数据

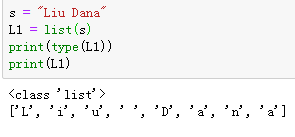
-list的创建

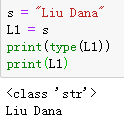
-直接创建，用中括号创建，内容直接用英文逗号隔开

-使用list创建

※列表中包含单个字符串的时候，是个特例。

List创建的特例演示





内置函数的概念

Help：帮助函数

Type：显示变量的类型

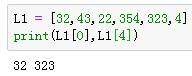
Id：显示变量的id

列表的常见操作

-访问

-使用下标操作，也叫索引。

-列表的元素索引是从0开始的

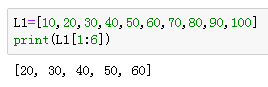


-切片操作

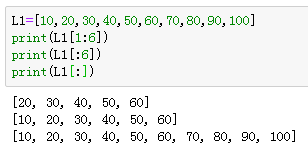
-对列表进行任意一段的截取

-截取后创建一个新的列表

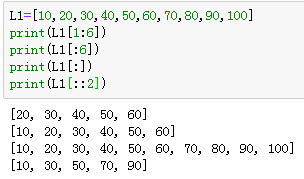
-切片操作需要注意取值范围，左包括，右不包括。



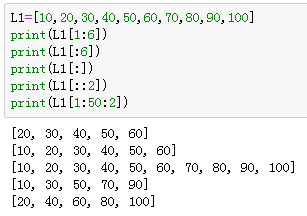
-切片下标可以为空，表示从第一个开始，以及 到最后一个结束



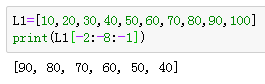
分片可以控制增长幅度，默认增长幅度为1



-下标可以超出范围，超出后不再考虑多余下标内容



-下标数值为负数时，表示从右往左，计数方式从-1开始，必须标注增长幅度



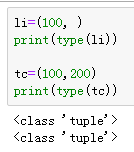
2-17课 list\_set\_tuple\_dict

Tuple（元组）

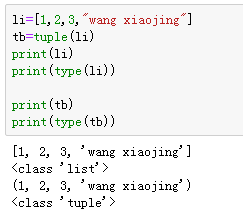
-可以理解成一个不允许更改的列表

创建元组的方式：

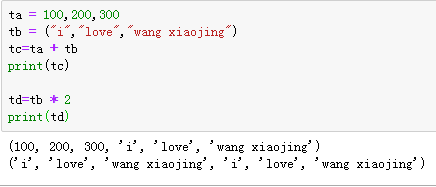
1. 用小括号内的单个元素后面加逗号，多个元素不用任何操作即可。



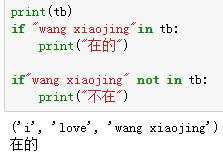
1. 用标识法创建



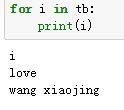
-tuple可以用加法，也可以用乘法



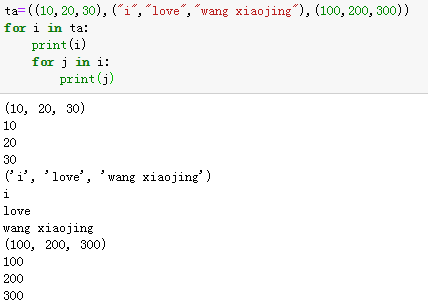
Tuple 的成员检测



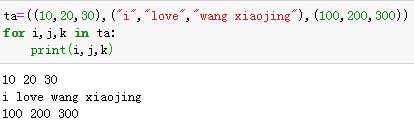
嵌套Tuple 的遍历



嵌套Tuple 的遍历双层循环

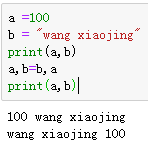


嵌套Tuple 的遍历单层循环



Tuple的特殊用法

Python特有的用法（互换）



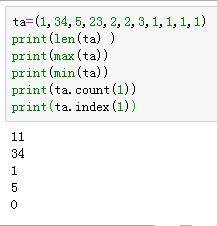
常用元组函数

Len：长度

Max/min：最大值/最小值

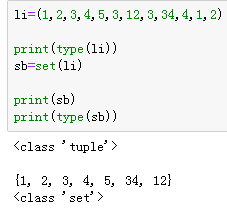
Count：计数

Index：某一元素所在的第一个位置，找到第一个元素就停止遍历。

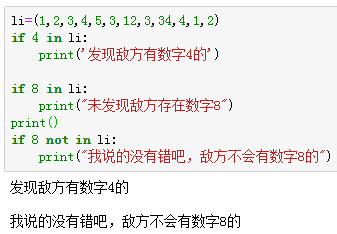


集合

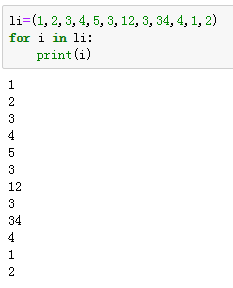
-特点：内容无序，内容不重复,使用大括号。



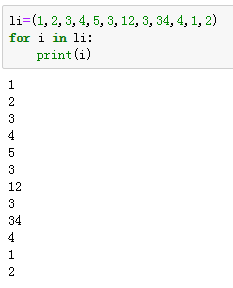
--- in 操作



-for循环遍历的使用

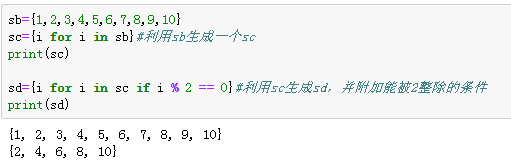


-集合的另一种遍历



--集合的生成式

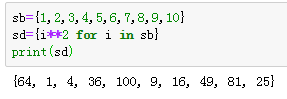
利用一个集合生成另外一个集合，并且可以附加条件。



双重for循环

把sb中的每个元素的平方生成一个新的集合，有两个方法：

方法1：



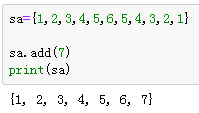
方法2：



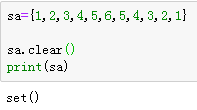
向集合中添加某个元素及删除集合内的所有元素

使用add及clear

----添加一个



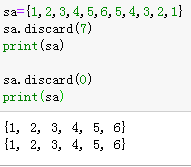
----删除集合内的所有元素



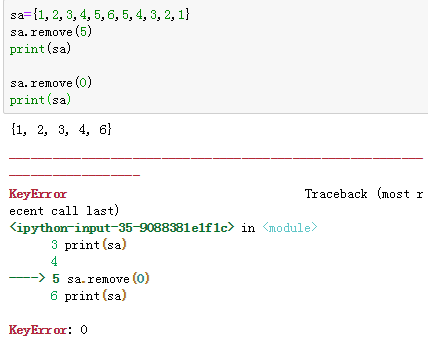
向集合内删除一个元素

Discard与remove的区别

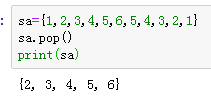
Discard：当操作要删除的元素不在集合中时不会报错



Remove：当操作要删除的元素不在集合中时，操作会被报错

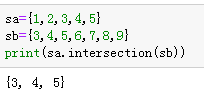


Pop表示随机删除集合中一个元素

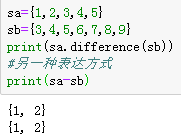


集合的数学操作

Intersection：表示交集



Difference：表示差集

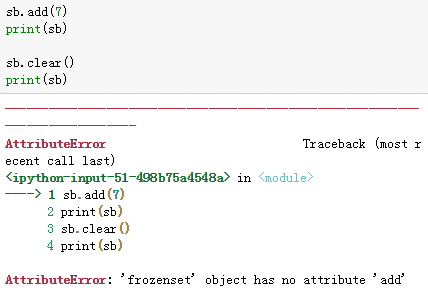


Union：表示并集

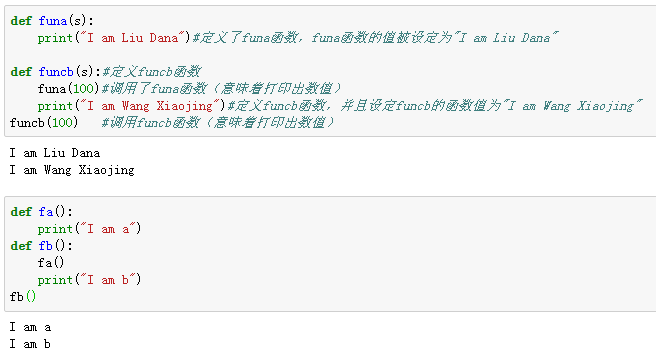


Frozenset 冰冻集合

---不允许修改的集合



2-18课 递归函数



·递归：函数间接或直接调用自己

·递归分两个过程：

----往下调用，分解的过程；

-----往上回溯，综合的过程。

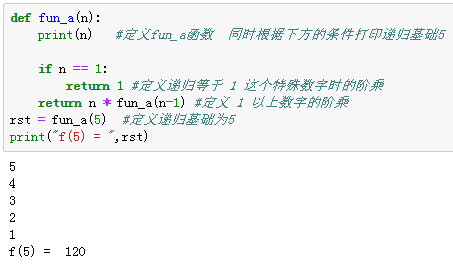
☆递归注意点：

-----一定要有结束条件

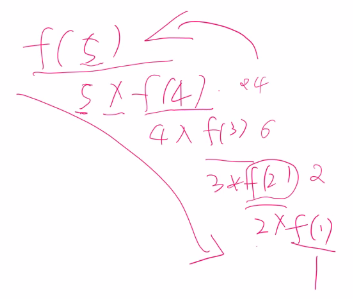
----是以资源换取编写速度，需要调用很大的资源空间

注释：func\_a 表示阶乘

阶乘就是fn=n×f（n-1）



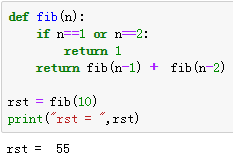
下方为f（5） = 120来源的解释（当递归向下阶乘到2时，反向往上阶乘，直至到5，出现计算数值为120）

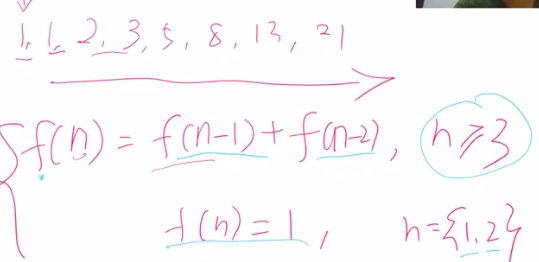


斐波那契数列

（1），1，2，3，5，8，13，21……（前面两数字相加等于后面的数）

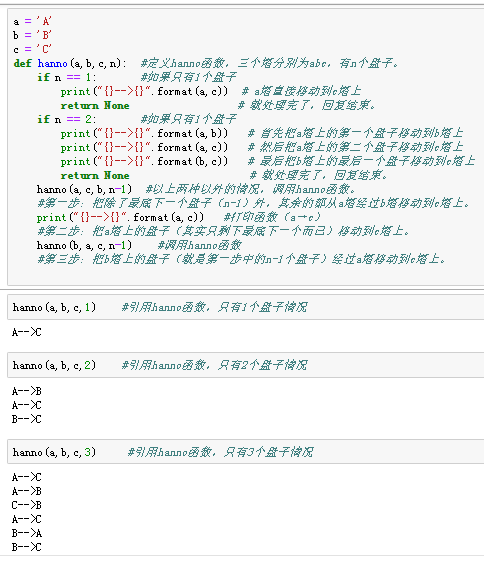
使用公式为：fib n = fib（n-1）+fib（n-2）





汉诺塔

1. 目的：A塔上的盘子→C塔上的盘子
2. 规则：每次只能动一个，ABC塔上的盘子永远只能大盘子在下面，大盘子在上面。



2-19 OOP1

-思想

----以模块化思想解决工程问题

----面向过程 vs 面向对象

-----由面向过程转向面向对象

-----面向对象的例子，我要开一个学校，叫轻羽学院，主要讲少儿编程，所需条件如下：

-讲师

-学生

-班主任

-教室

-学校

-常用名词

-OO：面向对象

-ooa：分析

-ood：设计

-oop：编程

-ooi：实现

ooa→ood→ooi

-类 vs 对象

----类：抽象，描述的是一个集合，侧重于共性（模板的概念）。

-----对象：具体，描述的是个体（具体对象的概念）

--类的内容：

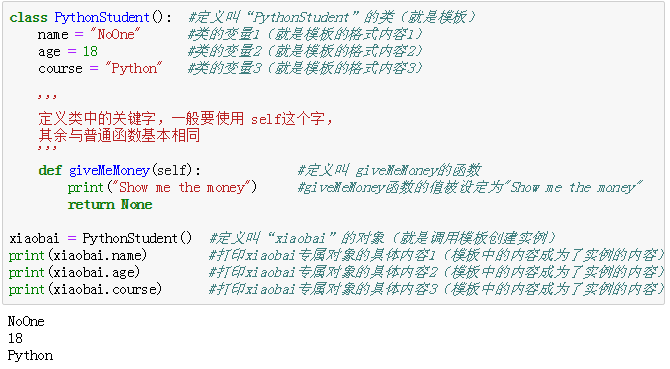
----动作，函数

----属性，变量

* 定义类：使用关键字class
* 类命名：
* --驼峰式命名法（如果类名中包括两个单词，第一个和第二个单词的首字母都大写）

☆类包括两大块内容，一个是描述静态属性（比如身高，体重，性别等），一个是描述动态的动作（比如吃饭，睡觉，打游戏等）。

那么静态的属性可以直接表述，动态的动作就需要用函数来描述。



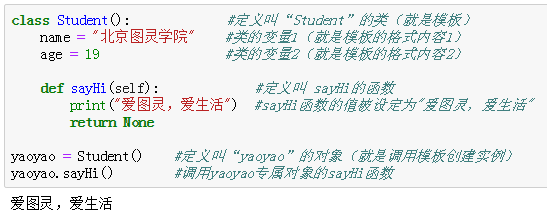
2-20课 面向对象

---解释上方 类里面函数中使用的self

--self这个词不是必须的，只是约定俗成的，也可用别的名称代替

--self本身不是关键字

--作用是指代本身

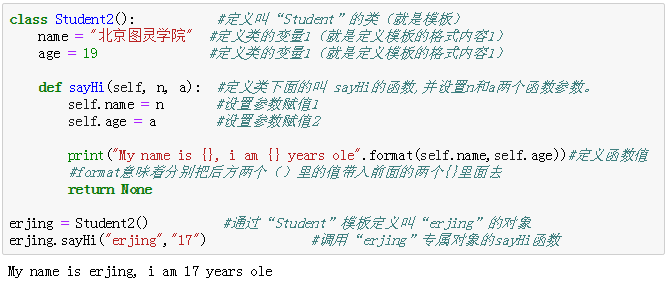


--类变量：属于类自己的变量

--实例变量：属于实例的变量

**1※ 访问实例的属性**

--访问实例的属性，如果实例没有属性，则会自动访问类的属性，如果类也没有定义，则报错。



**2※ 访问类的属性**

---在类里面如果强制访问类的属性，则需要使用\_\_class\_\_ （注意前后各有两个下划线）。

---类方法：

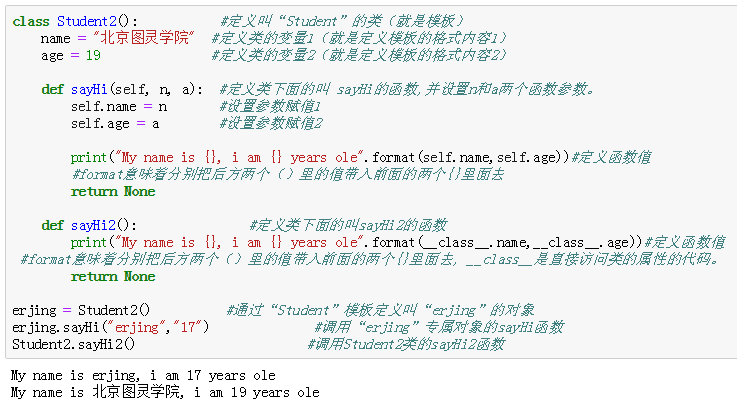
·定义类的方法的时候，是不需要self的（今后会讲需要cls，暂时可不用理会）

·类的方法中只允许使用类的内容

·使用类的方法有两种

1. Class Name（就是直接使用类的名字，“称呼模板名称”，叫谁就访问谁。）
2. \_\_class\_\_ （访问类的一个直通车代码，只要使用就代表直接访问类。）

个人理解：如果是访问类，那么只能用类的名称来访问，或者用专用关键字\_\_class\_\_，总之不能用对象的名称（访问谁，不能用其他对象的名字）。





构造函数1（后面还有2，往后找，在第21课）

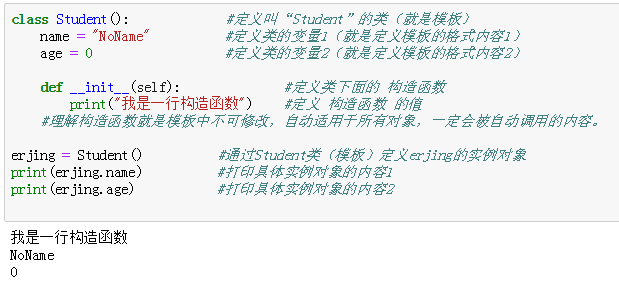
----类在实例化的时候，执行一些基础型的初始化的工作

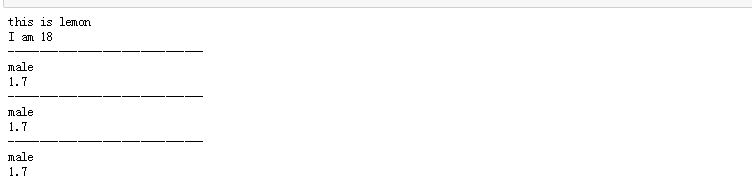
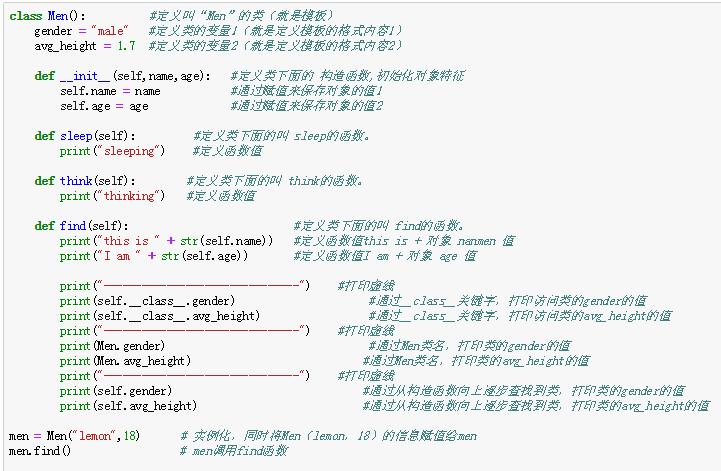
------使用特殊的名称和写法 “\_\_init\_\_”

------在实例化的时候自动执行

------是在实例化的时候第一个被执行的函数

☆个人理解：构造函数是将实例对象进行初始化的，为什么要初始化呢，作用是分配内存空间（类是一种抽象结构，而不是一种实体，并不占用存储空间。而类的对象不同，需要把类的数据进行资源分配，创造对象产品，容纳数据，占用空间，对象的数据加载进内存存储。初始化通俗点来说是“从头开始建立”）





2-21课 面向对象的三大特征

-继承

-封装

-多态

继承

-子类可以使用父类定义的内容或者行为等

-继承的实现

-----父类，基类，超类：被继承的类，Base Class，Super Class

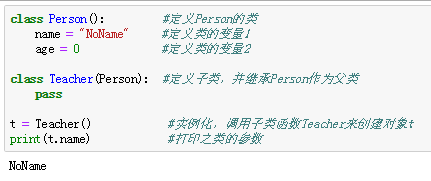
-----子类：有继承行为的类

------所有类都必须至少有一个父类

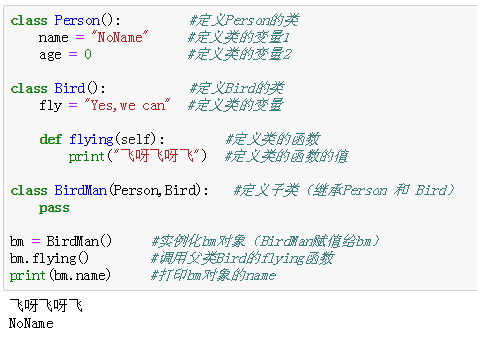
·如果没有，则默认为是object的子类



一个父类的示例：



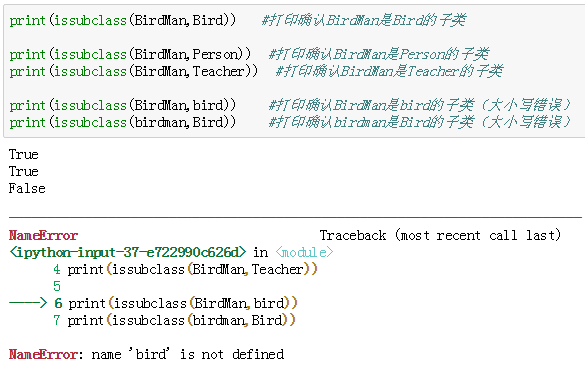
两个父类（多继承）的示例：



Issubclass检测是否是子类

·可以用来检测两个类的父子关系

利用上面的继承公式进行示例，结论如下：



构造函数2

·在函数实例化的时候调用的一个函数

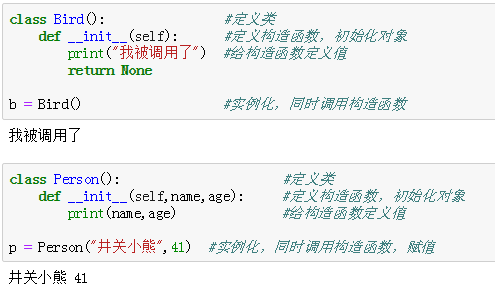
·自动调用

·要求，第一个参数必须有，一般推荐self

·构造函数的调用时间：一般认为在实例化的时候第一个被调用

·一般不用手动调用，实例化的时候自动调用，参数需要写入类名称后面的括号里。

下方演示构造函数使用的两大形式

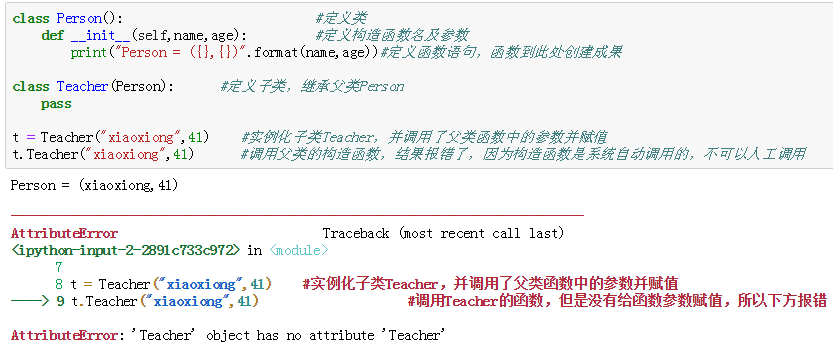


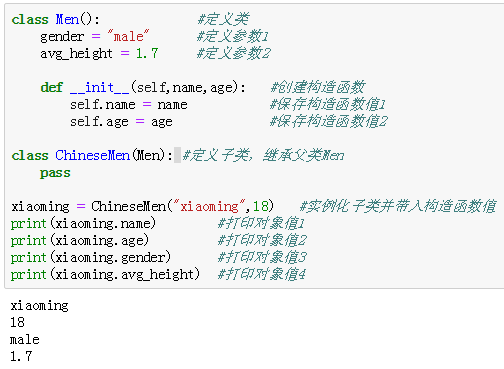
构造函数的继承

·构造函数默认继承，子类如果没有构造函数，则自动调用父类的构造函数。

·一旦子类定义了构造函数，则不再自动调用父类构造函数

☆·上面所说的调用构造函数是指系统自动调用构造函数，不是只人为调用构造函数进行打印。人为调用普通函数是可以的，人为调用构造函数打印是会报错的，见下方的示例。





3-2课 变量\_操作符

字符串

---转义字符

---格式化

---内建函数（后延）

·转义字符

---用一个特殊的方法表示出一系列不方便写出的内容，比如回车键，换行键，退格键等。

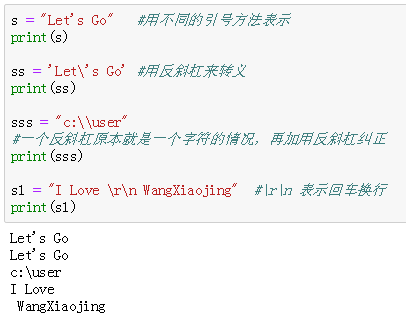
---借助反斜杠字符，一点字符串中出现了反斜杠，则反斜杠后面一个或几个字符进行了转义，已经不是原来的意思了。

---在字符串中，一旦出现反斜杠就要加倍小心，可能由转义字符出现

---不同系统对换行操作有不同的表示

-windows：\n

-linux: \r\n



·字符串的格式化

---把字符串按照一定格式打印或者填充

---格式化有两种方法

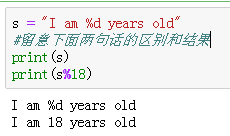
--利用百分号%格式化

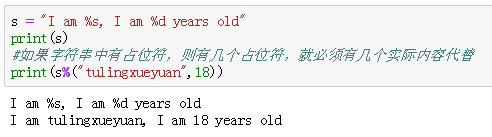
·在字符串中，利用%表示一个特殊的含义，表示对字符进行格式化

·%d：此处应该放入一个整数

·%s：表示此处应该放入一个字符串



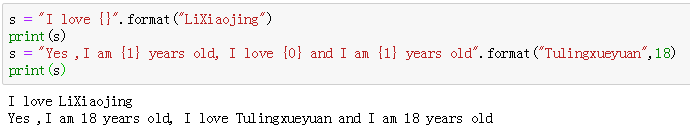




--利用format函数

·推荐使用format函数进行格式化

·在使用上，以{ } 和： 代替%，后面使用format带参数完成



--None

·表示什么都没有

·如果函数没有返回值，可以返回None

·用来占位置

·用来接触变量绑定

--表达式

·由一个或者几个数字或者变量和运算符组合成的一行代码

·通常会返回一个结果  
 --运算符

·由一个以上的值经过变化得到新值的过程就叫运算

·用于运算的符号叫做运算符

·运算符分类

---算数运算符

----比较或者关系运算符

----赋值运算符

----逻辑运算符

----位运算

----成员运算符

----身份运算符

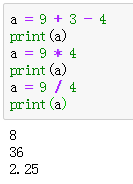
--算数运算符

·进行算数运算的符号

+ ，- 跟正常算术运算符一模一样

乘号用星号（\*）代替

除号用斜杠（/）代替



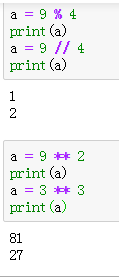
% 表示取余运算

两个数字相除应该有商数和余数

% 只会得到余数

// 表示取商运算

\*\* 表示幂运算（次方）



--比较运算符

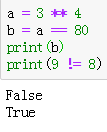
·对两个变量或者值进行比较的运算符

·比较的结果是布尔值，即True/False

== 代表等于号

!= 代表不等于号

>, <, >=, <=

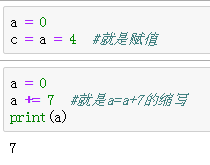


--赋值运算符

= 表示赋值

+= 表示缩写

-=， ×=， /=, //=, %=, \*\*= 都是缩写形式



--逻辑运算符

·对布尔值进行计算的符号

·and逻辑与

·or 逻辑或

·not 逻辑非

·python中逻辑运算没有异或运算

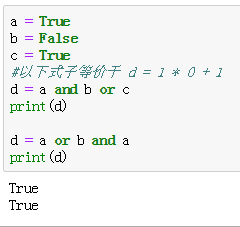
---运算规则：

and 看作乘法； or看作加法

True 看作1 ； False看作0

则逻辑运算就能转换成整数数学运算

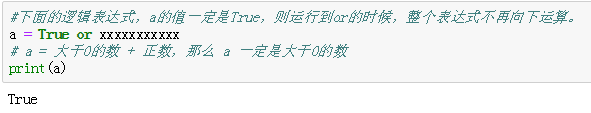
☆ 如果数字用来当布尔值使用时，0=False, 其余当作True



逻辑运算的短路问题

--逻辑运算式，按照运算顺序计算，一旦能够确定整个式子未来的值，则不再进行计算，直接返回。

逻辑运算的短路案例

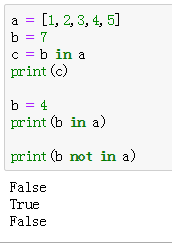


--成员运算符

·用来监测某一个变量是不是另外一个变量的成员

-- in

-- not in

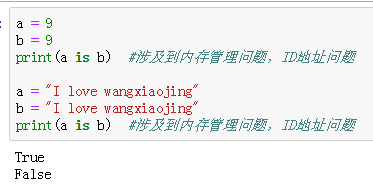


--身份运算

--- is ：用来检测两个变量是否是同一个变量

·语法就是var1 is var2

--- is not ：两个变量不是同一个变量



---运算符的优先级问题

--永远记住，括号具有最高优先级

程序结构

--顺序

--分支

--循环

3-3课 分支\_for循环

-- 分支

分支的基本语法

If 条件表达式

语句1

语句2

语句3

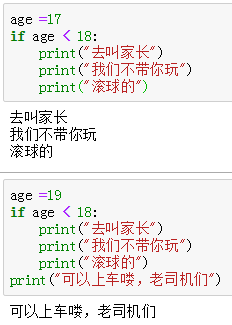
……

-条件表达式就是计算结果必须是布尔值的表达式

-表达式后面的冒号不能少

-注意if后面出现的语句，如果属于if语句块，则必须同一个缩进等级。

-条件表达式结果为True执行if后面的缩进的语句块



--双向分支

- if……else…… 语句

If 条件表达式

语句1

语句2

……

else:

语句1

语句2

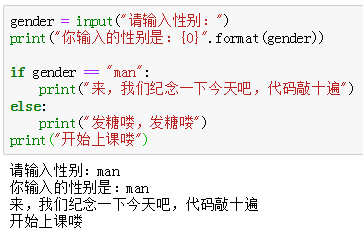
……

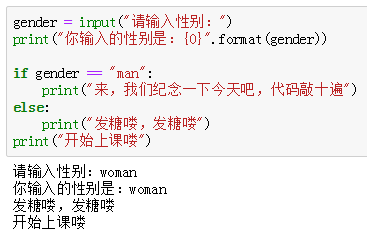
·双向分支有两个分支，当程序执行到if…else…语句时，一定会执行if或者else中的一个，也仅执行一个

·缩进问题，if和else一个层次，其余语句一个层级

* input的作用：

1. 在屏幕上输出括号内的字符串；
2. 接受用户输入的内容并返回到程序；
3. input返回的内容一定是字符串类型。





--多路分支

----很多分支的情况，简称多路分支

if 条件表达式：

语句1

……

elif 条件表达式：

语句1

……

elif 条件表达式：

语句1

……

……

else:

语句1

……

· elif可以有很多个

· elif是可选的



--if语句其他

--if语句可以嵌套使用，但不推荐

--Python没有switch-case语句

--循环语句

--重复执行某些固定动作或者处理基本固定的事物

--分类

-for循环

-while循环

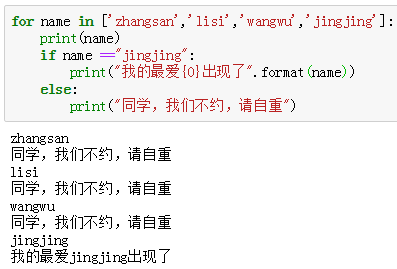
---for循环

For变量 in 序列：

语句1

语句2

……



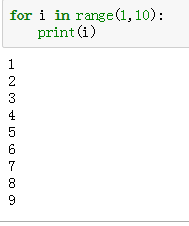
--range介绍

--生成一个数字序列

--具体范围可以设定

☆注意：如果由表示数字范围的两个数，一般是包含左边数字不包含右边数字。

但是，randint是特例，他左右都包含。



* for-else语句

---当for循环结束的时候，会执行else语句

--else语句是可选语句

3-4课 while\_函数

--while循环

----一个循环语句

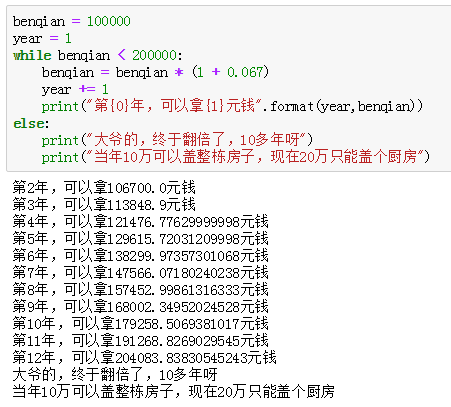
----表示当某条件成立的时候，就循环

----不知道具体循环次数，但能确定循环的成立条件的时候用while循环

----while语法：

while 条件表达式：

语句块



---函数

- 代码的一种组织形式

- 一个函数一般完成一项特定的功能

- 函数的使用

·函数需要先定义

·使用函数，俗称调用

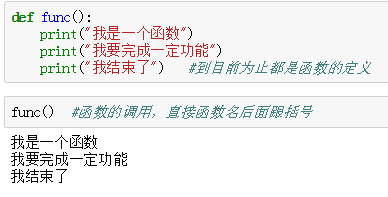
定义一个参数，但只是定义的话是不会执行的。

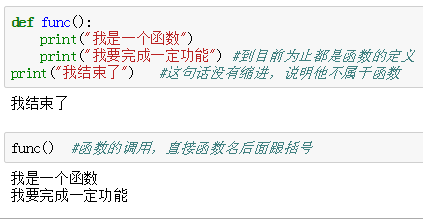
1、Def是关键字，后面要跟一个空格；

2、函数名 自己定义，起名要遵循命名规则（不能用数字开头），约定俗成，不要使用大驼峰，大驼峰命名只给类名用。

3、后面括号和冒号不能省，括号内可以有参数

4,、函数内所有代码缩进





---函数的参数和返回值

--参数：负责给函数传递一些必要的数据或者信息：

形参（形式参数）：在函数定义的时候用刀的参数，没有具体值，只是一个占位符号。

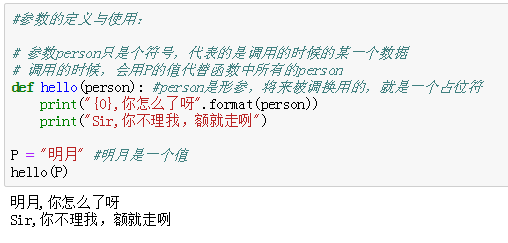
实参（实际参数）：在调用函数的时候输入的值

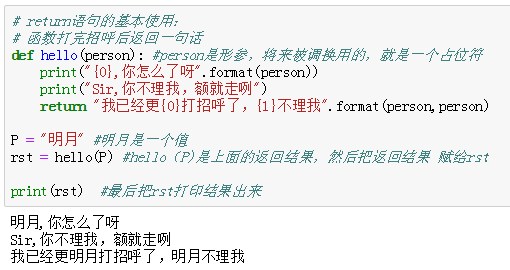
--返回值：函数的执行结果

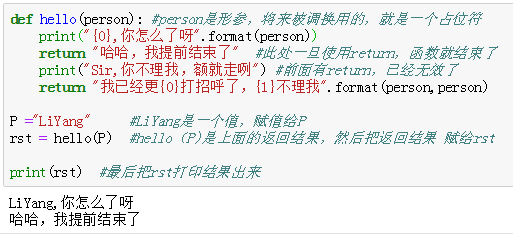
使用return关键字

如果没有return，默认返回一个None

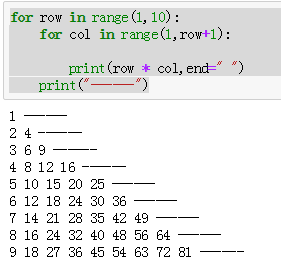
函数一旦执行return语句，则无条件返回，即结束函数的执行

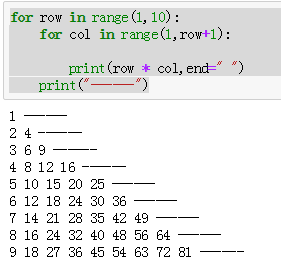


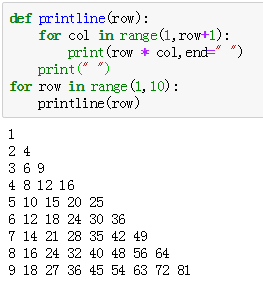




九九乘法表演示







--参数详解

【参考资料】（<https://www.cnblogs.com/bingabcd/p/6671368.html>）

-python参考资料：headfirst python →零基础入门学习python（小甲鱼），本讲义参考的是《流畅的python》→习题

--参数分类

--普通参数

--默认参数

--关键字参数

--收集参数

--普通参数

-C参见上例

-定义的时候直接定义变量名

-调用的时候直接把变量或者值放入指定位置

def 函数名（参数1，参数2，……）

函数体

#调用

函数名（value1，value2，……）

#调用的时候，具体值参考的是位置，按位置赋值

--默认参数

--形参带有默认值

调用的时候，如果没有对相应形参赋值，则使用默认值。

def func\_name(p1=v1,p2=v2,……)：

func\_block

#调用1

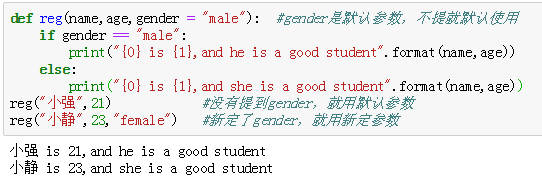
func\_name( )

#调用2

value1 = 100

value2 = 200

func\_name(value1 , value2)



3-5课 函数完结

--关键字参数

语法：

def func(p1 = v1 , p2 = v2,……):

func\_body

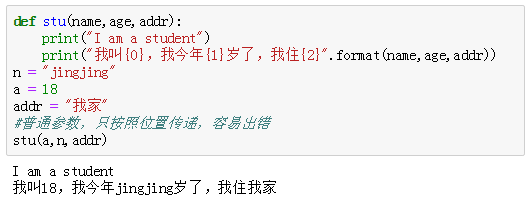
#调用参数

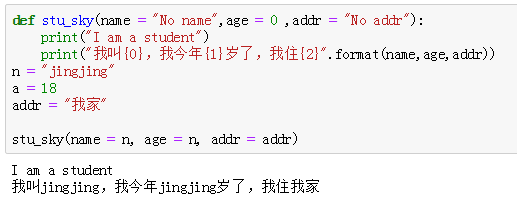
Func(p1 = value1 , p2 = value2,……)

-比较麻烦，但是也有好处：

--不容易混淆，一般实参和形参只是按照位置一一对应即可，容易出错。

--使用关键字参数，可以不考虑参数位置





--收集参数

--把没有位置，不能和定义时的参数位置相对应的参数，放入一个特定的数据结构中

--语法：

def func(\*args):

Func\_body

#调用函数

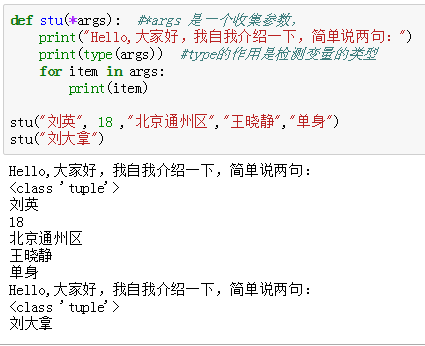
Func(p1 , p2 , p3 ,……)

--参数名args不是必须这么写，但是，我们推荐直接用args，约定俗成。

--参数名args前还需要一个小星星\*

--收集参数可以和其他参数共存

-- \*args接收的参数是元组形式



--收集参数之关键字收集参数

---把关键字参数按字典格式存入收集参数

语法：

def func(\*\*kwargs):

func\_body

#调用：

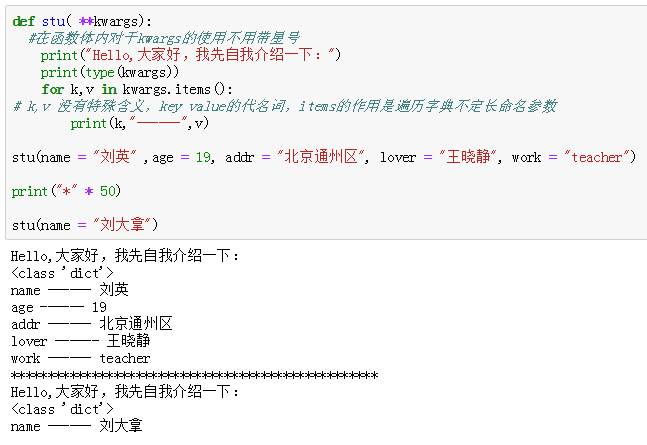
func(p1 = v1, p2 = v2, p3 = v3,……)

--kwargs一般约定俗成

--调用的时候，把多余的关键字参数放入kwargs

--访问kwargs需要按字典格式访问

-- \*\*kwargs接收是字典形式



--收集参数混合调用的顺序问题

--收集参数，关键字参数，普通参数可以混合使用

--使用规则就是，普通参数和关键字参数优先

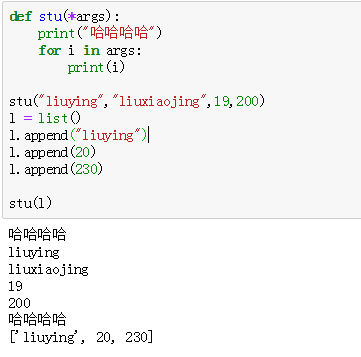
☆ 定义的时候顺序必须是： 普通参数（必须参数），关键字参数（可变参数），收集参数（默认值参数），收集参数（可变关键字参数，属于dict）



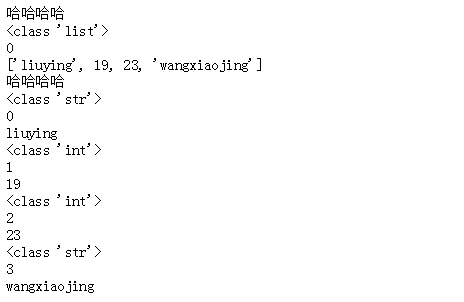
-收集参数的解包问题

--把参数放入list或者字典中，直接把list/dict中的值放入收集参数中

--语法：参看案例







--返回值

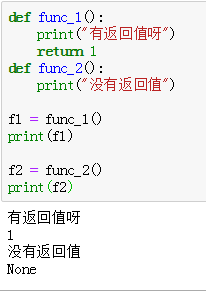
--函数和过程的区别

--有无返回值

--需要用return显示返回内容

--如果没有返回，则默认返回None

--推荐写法，无论有无返回值，最后都要以return结束



--函数文档

--函数的文档的作用是对当前函数提供使用相关的参考信息

--文档的写法：

--在函数内部开始的第一行使用三引号（’’’ ‘’’）字符串定义符

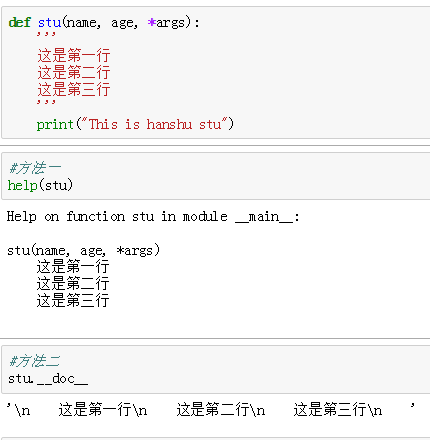
--一般具有特定格式

--参看案例

--文档查看

--使用help函数，形如help（func）

--使用\_\_doc\_\_，参看案例



3-6课 变量作用域和列表

--变量作用域

--变量由作用范围限制

--分类：按照作用域分类

-全局（global）：在函数外部定义

-局部 （local）：在函数内部定义

--变量的作用范围：

-全局变量：在整个全局范围都有效

-全局变量在局部可以使用（即函数内部可以访问函数外部定义的变量）

-局部变量在局部范围可以使用

-局部变量在全局范围无法使用

☆个人理解：从内部访问外面是可以的，从外面不可以访问内部。

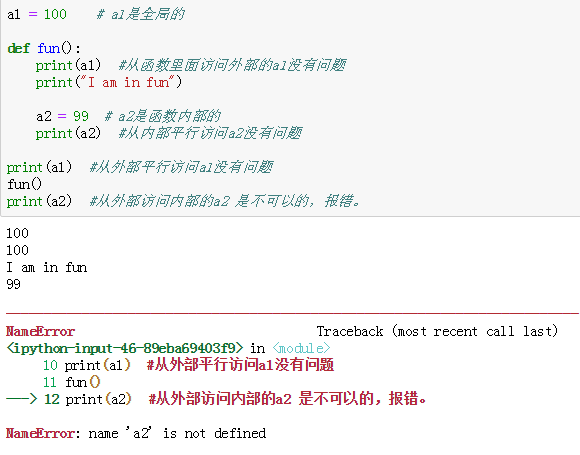
--LEGB原则

- L(local)局部作用域

- E(enclosing)外部嵌套函数作用域

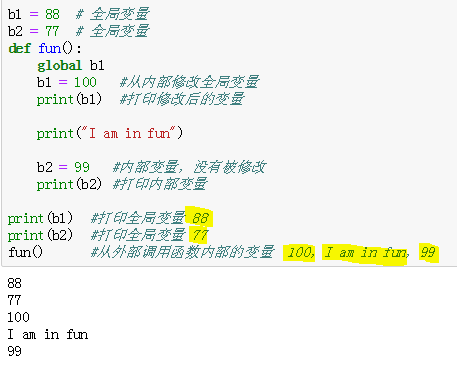
- G(global)函数定义所在模块作用域

- B(built)python内置模块的作用域



--提升局部变量为全局变量

--使用global



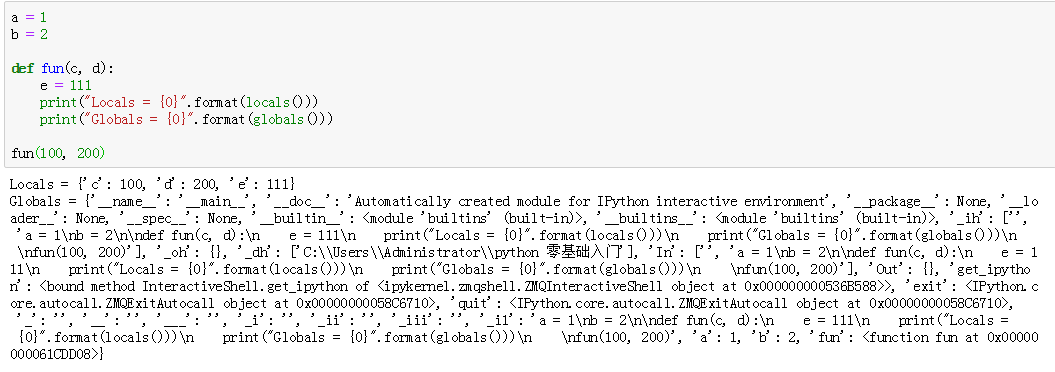
* globals，locals函数

-可以通过globals和locals显示出全局变量和局部变量

-globals和locals叫作内建函数

globals 是全局变量

locals 是局部变量

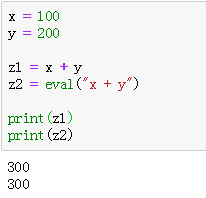


--eval（）函数

---把一个字符串当成一个表达式来执行，返回表达式执行后的结果

---语法：

eval(string\_code, globals = None, locals = None)

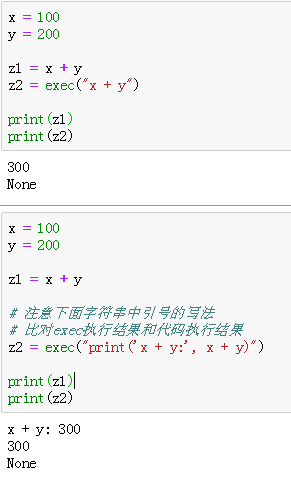


--exec（）函数

---跟eval功能类似，但是，不返回结果

---语法：

exec(string\_code, globals = None, locals = None)



---递归函数

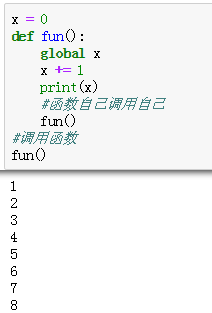
--函数直接或者间接调用自身

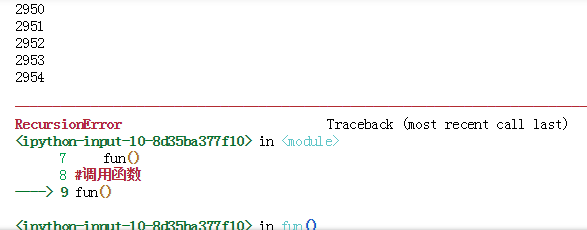
--优点：简洁，理解容易

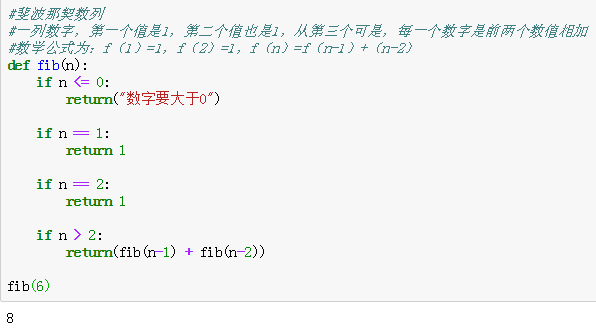
--缺点：对递归深度有限制，消耗资源大

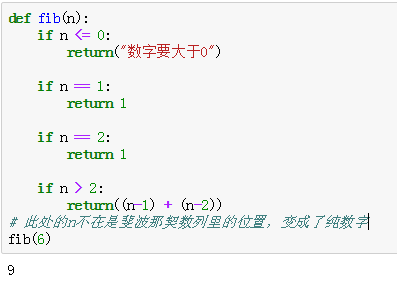
--python里面对递归深度有限制，超过限制报错

--在写递归程序的时候，一定注意结束条件









--内置数据结构（变量类型）

--list

--set

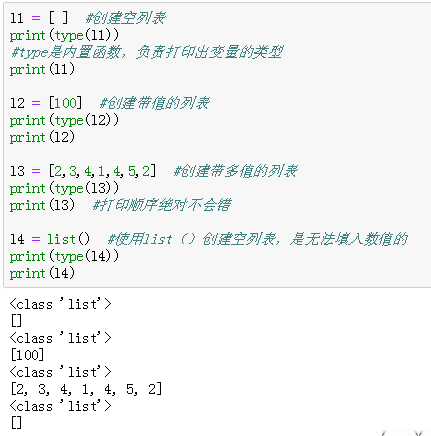
--dict

--tuple

--list（列表）

- 一组由顺序的数据的组合

-创建列表的4种方式如下：

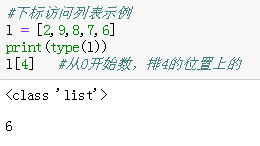


--列表常用操作

·访问

-使用下标操作（索引）

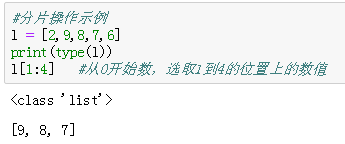
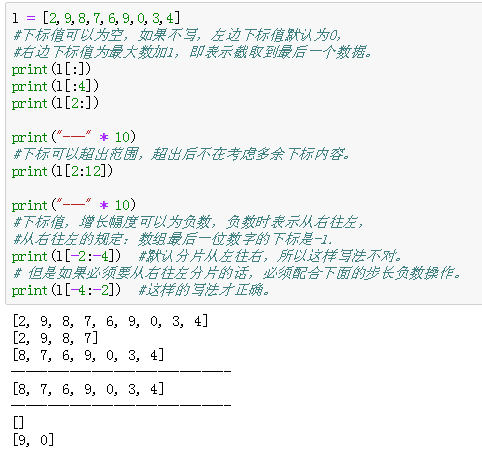
-列表的位置是从0开始

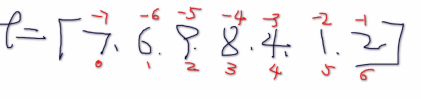


·分片操作

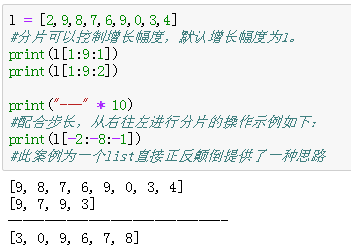
-对列表进行任意一段的截取

- l[ : ] 从0开始，不包含位尾数值，注意

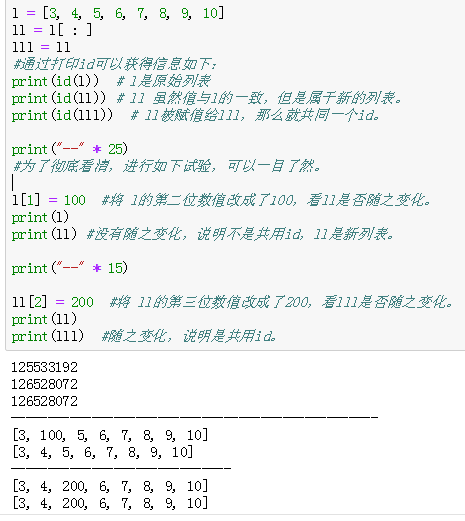


--分片同时可以控制增长幅度（步长），一般默认步长为1，也可单独设置。



--分片操作是生成一个新的列表

　·内置函数id，负责显示一个变量或者数据的唯一确定编号。



3-7 內建结构listdictset

汉诺塔问题

--规则：

1.每次移动一个盘子

2.任何时候大盘子在下面，小盘子在上面

--方法：

1. n = 1：直接把A上的一个盘子直接移动到C上，A→C

2. n = 2：

①把小盘子从A放到B上，A→B

②把大盘子从A放到C上，A→C

③把小盘子从B放到C上，B→C

3. n = 3：

①把A上的盘子通过C放到B上，调动递归

②把A上剩下的一个最大盘子移动到C上，A→C

③把B上两个盘子通过A放到C上，调用递归

4. n = n：

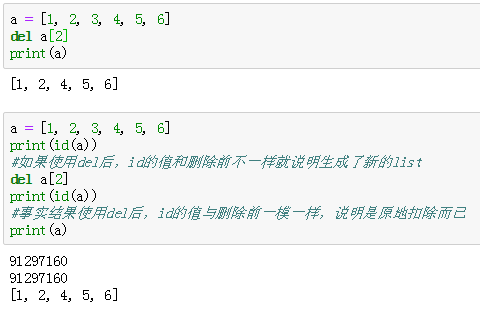
①把A上的n-1个盘子通过C放到B上，调动递归实现

②把A上最大盘子，也是唯一的一个，移动到C上，A→C

③把B上n-1个盘子通过A放到C上，调用递归

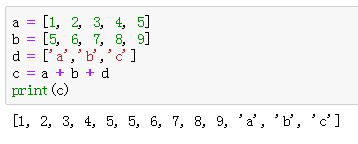
List （列表）

* · del 删除



列表相加

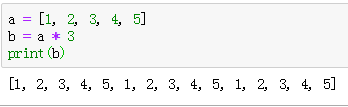
·使用加号链接两个列表



使用乘号操作列表

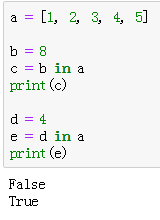
列表直接与一个整数相乘

相当于把n个列表链接在一起

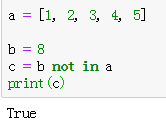


成员资格运算

就是判断一个元素是否在a的list里面

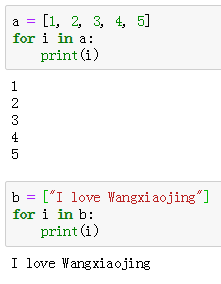


或者用 not in 的方式

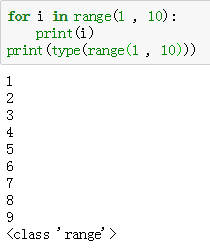


列表的遍历（for range while ）

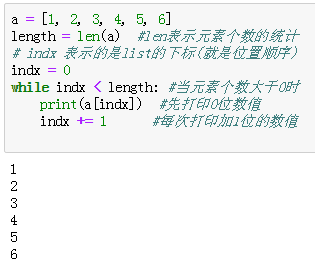
* for 循环



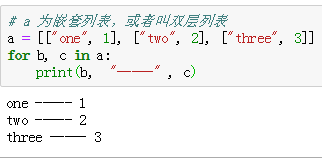
·range 循环



·while 循环



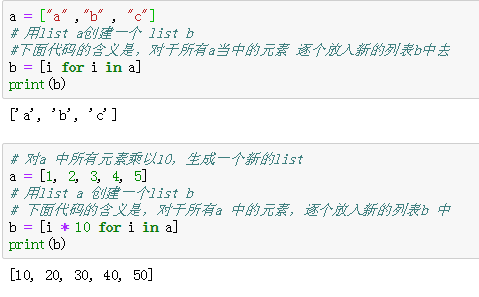
双层列表循环



列表内涵

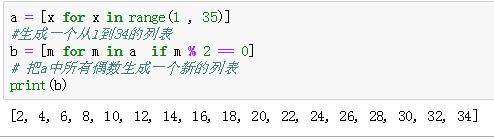
--通过简单方法创作列表

用for 创建

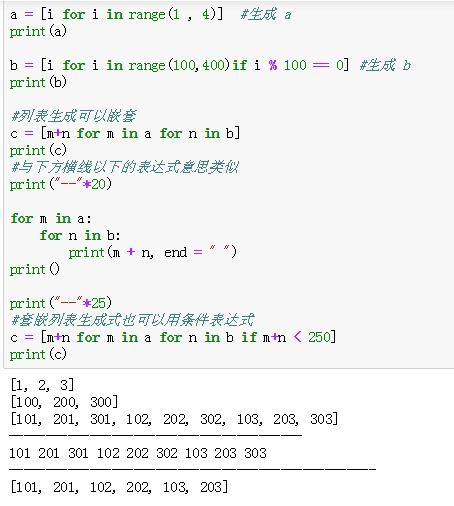


还可以过滤原来list当中的内容放入新列表

比如原有列表a，需要把所有a 当中的偶数生成新的列表 b

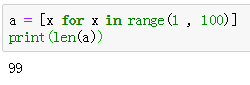


列表生成式可以嵌套

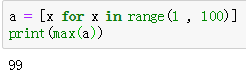


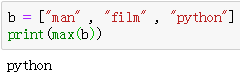
关于列表的常用函数

Len：求列表的长度

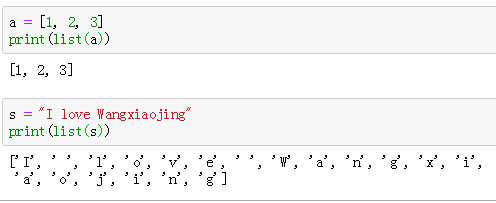


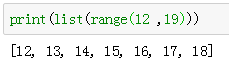
Max：求列表中的最大值





List：将其他格式的数据转换成list

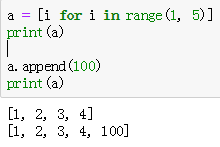




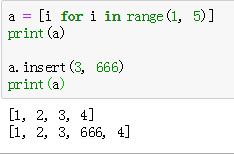
3-8课 內建结构listdictset

关于列表的函数

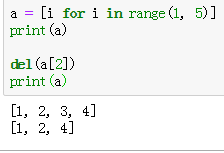
Append：在末尾追加一个内容



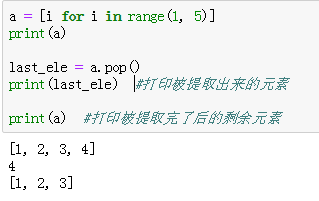
Insert：指定位置插入



Del： 删除



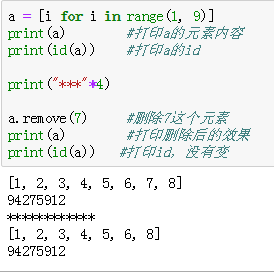
Pop：把最后一个元素提取掉



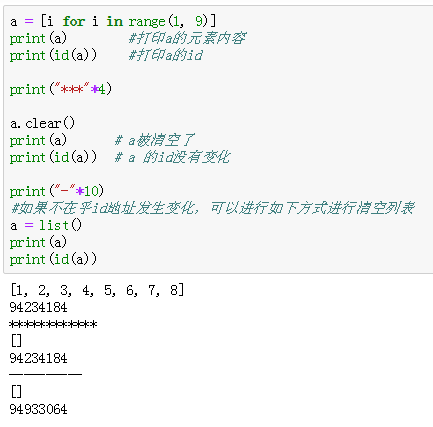
Remove：在列表中删除指定的值的元素

☆如果指定的值不存在，则会报错。

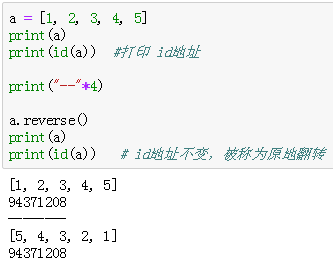
所以一般会事先使用if x in list，然后再操作list.remove(x)



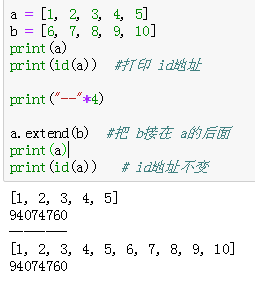
Clear：清空



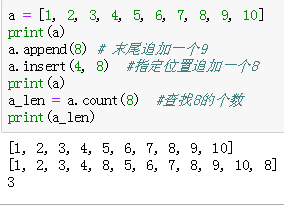
Reverse:翻转（id地址不会变，被称为原地翻转）



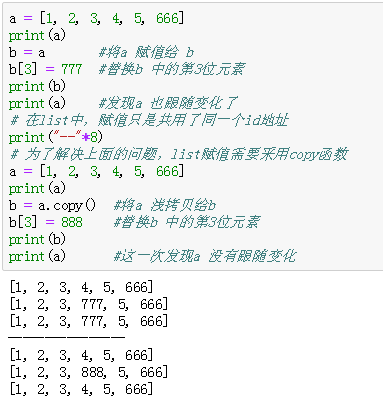
Extend：扩展列表，两个列表，把一个直接拼接到后一个上面



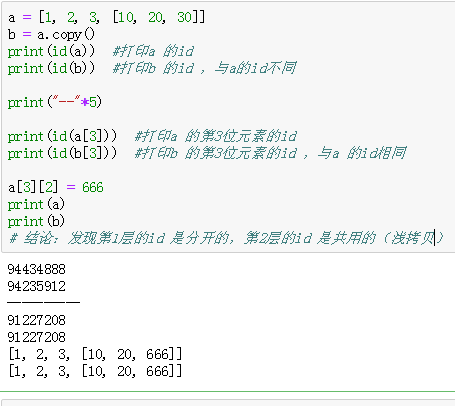
Count：查找列表中指定值或元素的个数



Copy：拷贝，浅拷贝

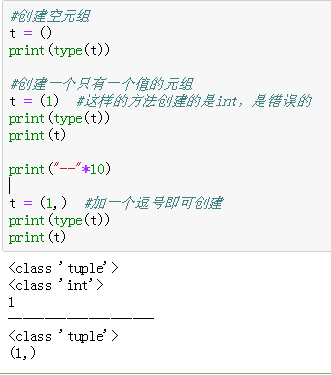


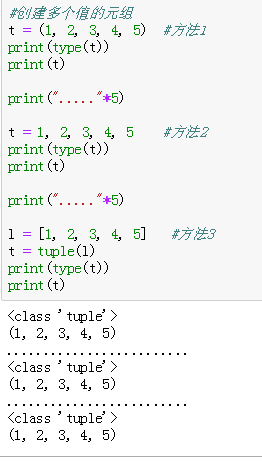
浅拷贝与深拷贝的区别



元组

---元组可以看成是一个不可更改的list





元组的特性

--是序列表，有序

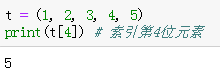
--元组数据值可以访问，不能修改，不能修改，不能修改

--元组数据可以是任意类型

--总之，list所有特性，除了可修改外，元组都具有

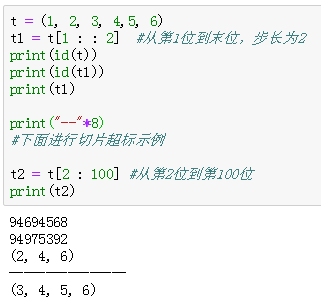
--也就意味着，list具有的一些操作，比如索引，分片，序列相加，相乘，成员资格操作等，一模一样。

索引操作示例：



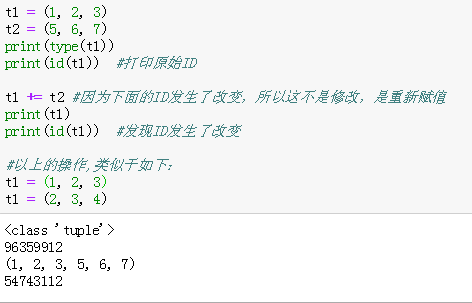
但是，索引如果超标（超出位数）就会报错

☆ 切片是可以超标的

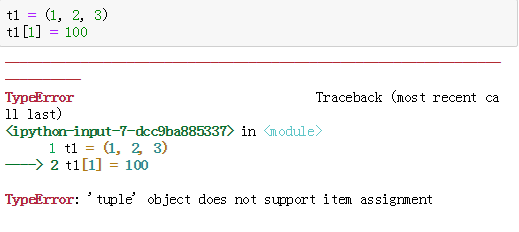


序列相加示例

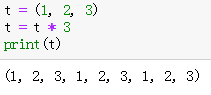
* 元组是不允许修改的，这里指的是内容不可以修改，但是允许重新赋值，见下面的示例。



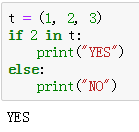
元组不允许修改的示例如下，修改就报错：



元组相乘示例

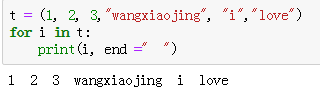


成员检测

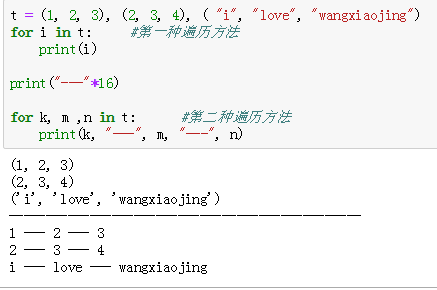


元组遍历，一般采用for

1、单层元组遍历



2、双层元组的遍历

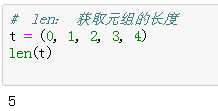


3-9 內建结构listdictset

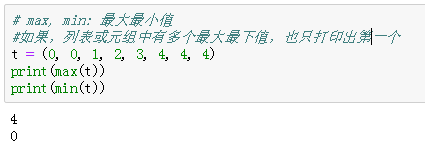
以下函数方式，对list也基本适用。

关于元组的函数

---Len：获取元组的长度



--max，min：最大最小值



--tuple：转化或创建元组

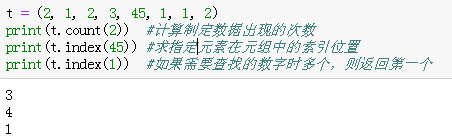


元组的函数

基本跟list通用

Count： 计算制定数据出现的次数

Index： 求指定元素在元组中的索引位置



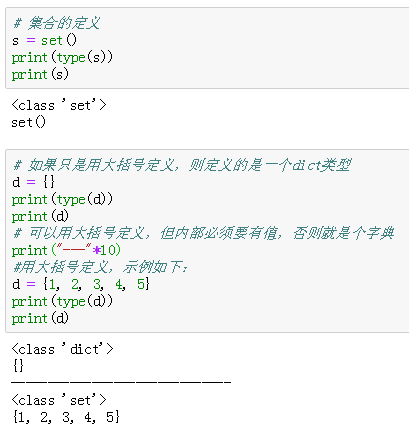
元组变量交换法



集合-set

集合是高中数学中的一个概念

一堆确定的、无序的、唯一的数据，集合中每一个数据成为一个元素



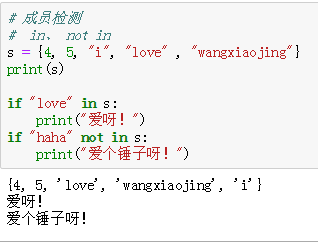
集合的特征

--集合内部数据无序，即无法使用索引和分片

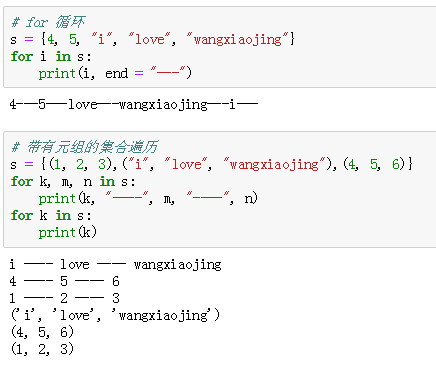
--集合内部数据元素具有唯一性，可以用来排除重复数据

--集合内的数据， str、int、float、tuple、冰冻集合等，即内部只能放可哈希的数据。

集合的序列操作



集合的遍历操作



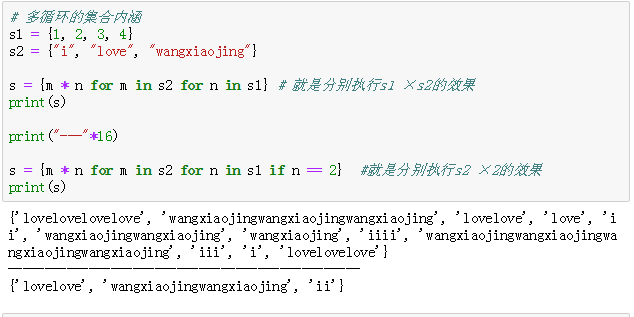
集合的内涵

--普通集合内涵

--带条件的集合内涵

--多循环的集合内涵



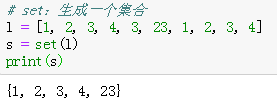


集合函数/关于集合的函数

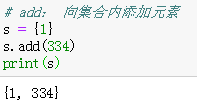
Len、max、min： 与其他基本函数一致



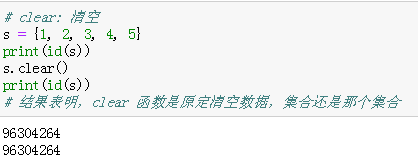
Set：生成一个集合



Add： 向集合内添加元素



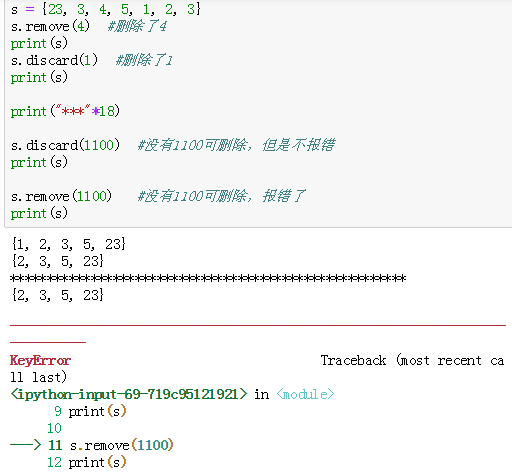
Clear：清空



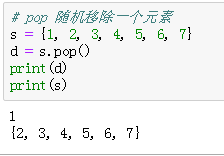
Copy：拷贝

Remove： 移除指定的值，直接改变原有值，如果要删除的值不存在，报错。

Discard： 移除集合中指定的值，与remove一样，但如果要删除的话，不报错。



Pop： 随机移除一个元素



集合函数

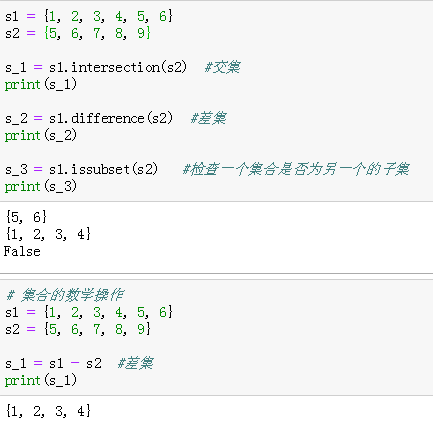
Intersection：交集

Differnce：差集

Union：并集

Issubset：检查一个集合是否为另一个子集

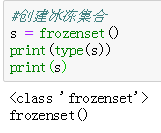
Issuperset：检查一个集合是否为另一个超集



Frozen set：冰冻集合

--冰冻集合就是不可进行任何修改的集合

--就是一种特殊集合，除了不能修改的特性之外，没有其他不同。



Dict 字典

--字典是一种组合数据，没有顺序的组合数据，数据以键值对形式出现



字典的特征

--字典是序列类型，但是是无序序列，所以没有分片和索引

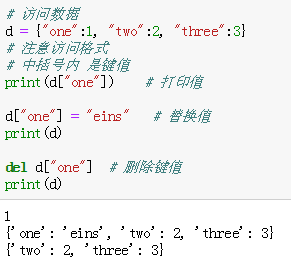
--字典中的数据每个都由键值对组成，即K V对

--key：必须是可哈希的值，比如int，string，float，tuple，但是list，set，dict不行。

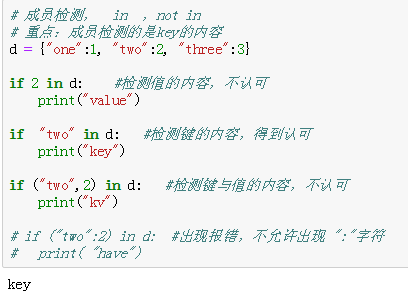
--value：任何值

字典常见操作

--访问数据

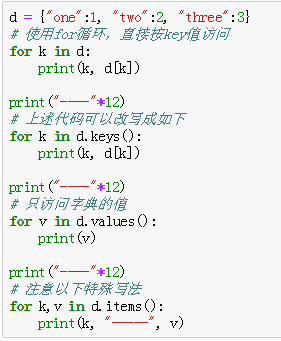


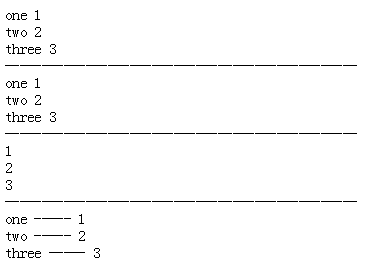
--成员检测：in ，not in



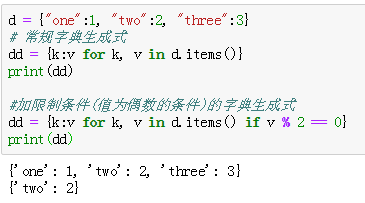
--遍历

用key及values来使用for循环





字典生成式

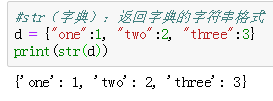


字典的相关函数

通用函数：len，max，min，dict

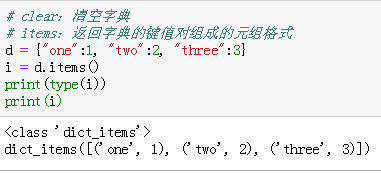
特殊函数如下：

# str（字典）：返回字典的字符串格式



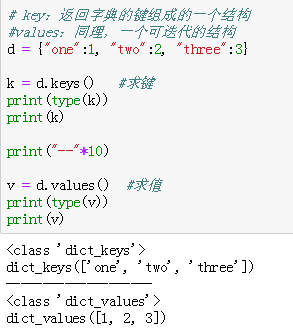
# clear：清空字典

# items：返回字典的键值对组成的元组格式



# key：返回字典的键组成的一个结构

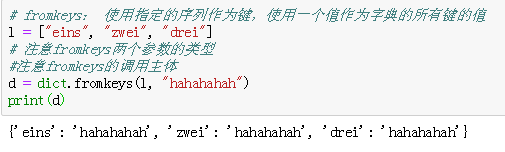
#values：同理，一个可迭代的结构



# get：根据指定键返回相应的值，好处是，可以设置默认值



# fromkeys： 使用指定的序列作为键，使用一个值作为字典的所有键的值



3-10 课 OOP-python面向对象

--python的面向对象

·面向对象编程

-基础

-公有私有

-继承

-组合，minxi

·魔法函数

-魔法函数概述

-构造类魔法函数

-运算类魔法函数

1、 面向对象概述（object oriented，00）

-OOP思想

-接触到任意一个任务，首先想到的是任务这个世界的构成，是有模型构成的

-几个名词

--OO:面向对象

--OOA:面向对象的分析

--OOD:面向对象的设计

--OOI:XXX的实现

--OOP:XXX的编程

--OOA→OOD→OOI:

-类和对象的概念

--类：抽象名词，代表一个集合，共性的事物

--对象：具象的事物，单个个体

--类与对象的关系

·一个具象，代表一类事物的某一个个体

·一个是抽象，代表的是一大类事物

-类中的内容，应该具有两个内容

--表明事物的特征，叫做属性（变量）

--表明事物功能或动作，称为成员方法（函数）

2、类的基本实现

-类的命名

--遵守变量命名的规范

--大驼峰（由一个或者多个单词构成，每个单词首字母大写，单词跟单词直接相连） --尽量避开跟系统命名相似的命名

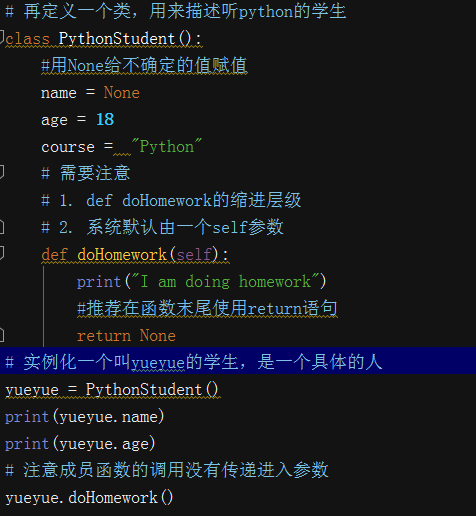
-如何声明一个类

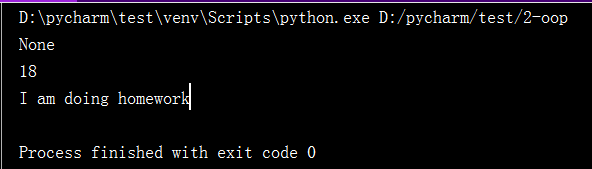
--必须用class

--类由属性和方法构成，其他不允许出现

--成员属性定义可以直接使用变量赋值，如果没有，只允许使用None

--案例 01.py





-实例化类

变量 = 类名（） #最简单的一个实例化

-访问对象成员

--使用点操作符

obj.成员属性名称

obj.成员方法

-可以通过默认内置变量检查类和对象的所有成员

--对象所有成员检查

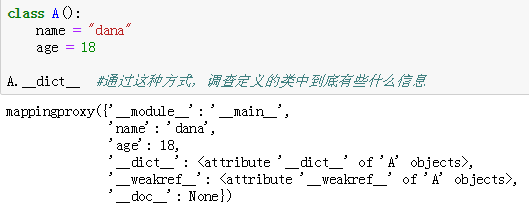
# dict 前后各有两个下划线

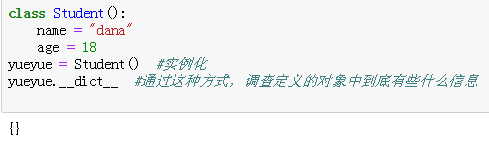
obj.\_\_dict\_\_

-类所有的成员

# dict前后各有两个下划线

class\_name.\_\_dict\_\_





3、anaconda基本使用

-anaconda 主要是一个虚拟环境管理器

-还是一个安装包管理器

-conda list：显示anaconda安装的包

-conda env list：显示anaconda的虚拟环境列表

-conda create –n xxx python=3.6：创建python版本为3.6的虚拟环境，名称为xxx

3-11 OOP

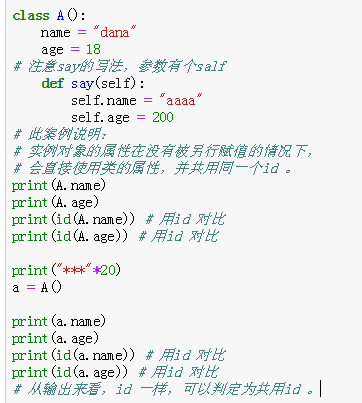
4、类和对象的成员分析

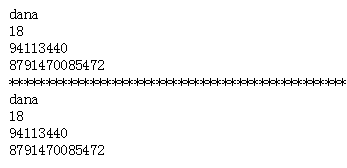
-类和对象都可以存储成员，成员可以归类所有，也可以归对象所有

-类存储成员时使用的是与类关联的一个对象

-对象存储成员时存储在当前的对象中

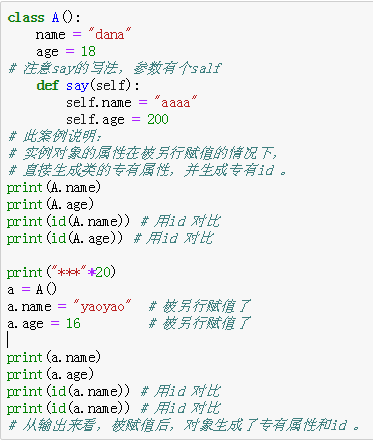
-对象访问一个成员时，如果对象在没有该成员，尝试访问类中的同名成员，如果对象中有此成员，一定使用对象中的成员。

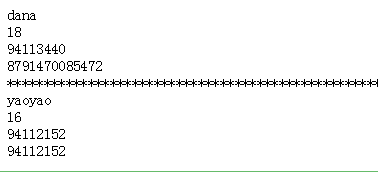




-创建对象的时候，类中的成员不会放入对象当中，而是得到一个空对象，没有成员。

-通过对象对类中成员重新赋值或者通过对象添加成员时，对应成员会保存在对象中，而不会修改类成员。



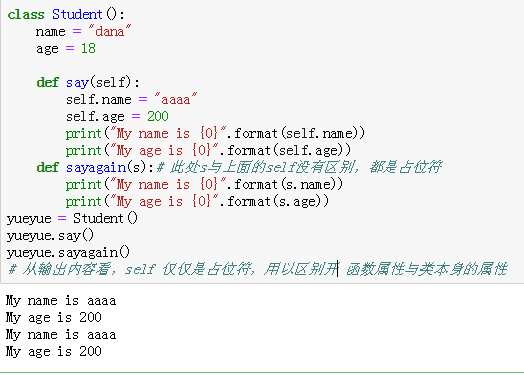


5、关于self

-self在对象的方法中标示当前对象本身，如果通过对象调用一个方法，那么该对象会自动传入到当前方法的第一个参数里面。

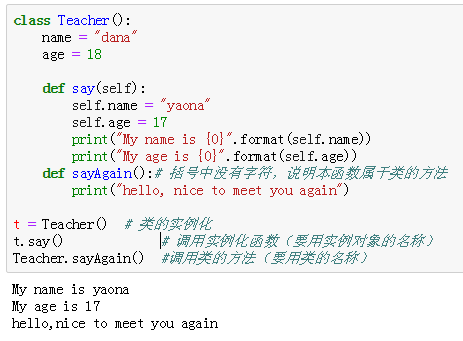
-self并不是关键字，只是一个用于接受对象的普通参数，理论上可以用任何一个普通变量名代替。

☆ 个人理解，为什么类的函数的括号中一定要使用self或等同于self的占位符呢，为什么不能空着呢。我认为加self或等同于self的占位符是少不了的，①这是为了区分函数类型的，一般函数一定要有类似self的占位符，后面会提到类的方法的函数在括号中加cls，静态方法的函数的括号中是空着的；②这是为了区分 类的属性与类的函数的属性。类的属性直接表达name age，类的函数的属性表达方式是self.name self.age ，不区分开会混淆；③所有带有self的函数或数值都是面对实例对象的，通俗来说就是专门给实例对象使用的。



- 方法中有self形参的方法成为非绑定类的方法，可以通过对象访问，没有self的是绑定类的方法，只能通过类访问。

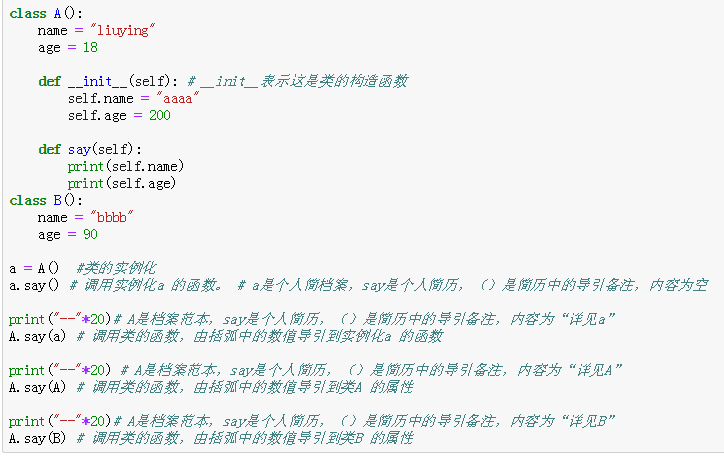
☆ 通俗说：类下面的函数如果没有self，就是类的方法，类的方法属于类，不属于对象，对象不能调用类的方法，类才可以调用类的方法。类下面的函数有self时，就是类的面向对象的函数，是可以被对象调用的。

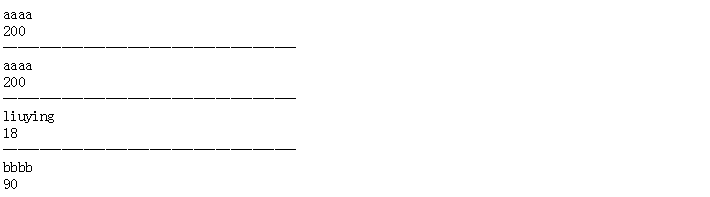


-使用类访问绑定类的方法是，如果类方法中需要访问当前类的成员，可以通过\_\_class\_\_成员名来访问。



# 继续关于self的案例





6、面向对象的三大特性

-封装

-继承

-多态

6.1 封装

- 封装就是对对象的成员进行访问限制

- 封装的三个级别

·公开 public

·受保护的 protected

·私有的 private

·public，private，protected不是关键字

-判别对象的位置

-对象内部

-对象外部

-子类中

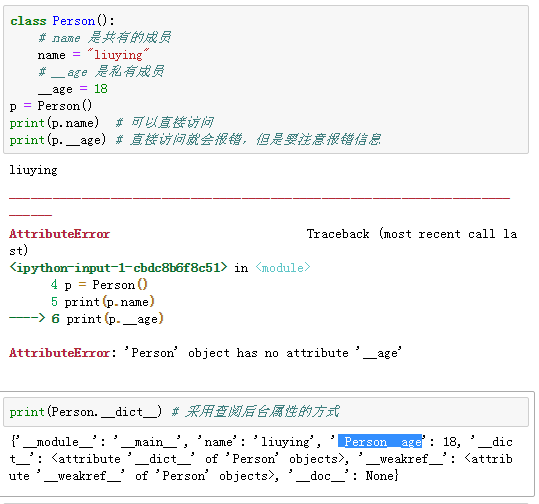
-私有

-私有成员是最高级别的封装，只能在当前类或对象中访问

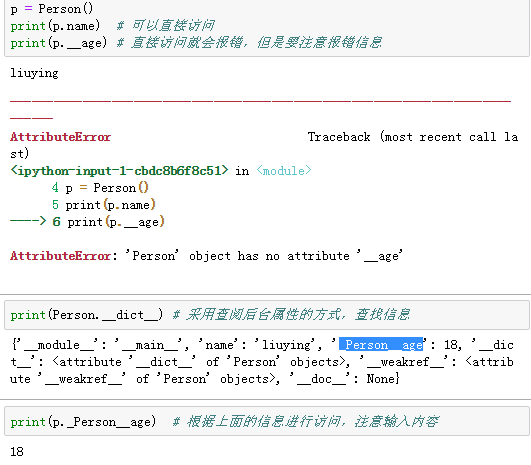
-在成员前面添加两个 两个下划线即可



-Python的私有不是真私有，是一种称为name mangling的改名策略，可以使用对象.\_class name\_attributename来访问，以上禁止访问的解决方法见下例



找到后台的信息后，通过信息口令进行访问，成功访问情况如下：



-受保护的封装 protected

-受保护的封装是将对象成员进行一定级别的封装，然后在类中或者子类中都可以进行访问，但是在外部不可以。

- 封装方法：在成员名称前添加一个下划线即可。

3-12课 OOP

-公开的，公共的 public

- 公共的封装对实际成员没有任何操作，任何地方都可以访问。

## 3.2 继承

-继承就是一个类可以获得另外一个类中的成员属性和成员方法。

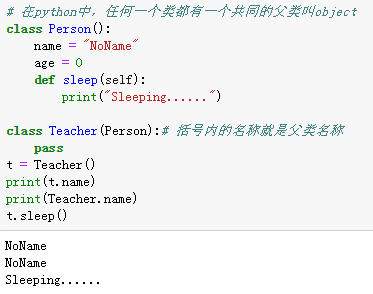
-作用：减少代码，增加代码的复用功能，同时可以设置类与类直接的关系。

-继承与被继承的概念：

-被继承的类叫父类，也叫基类，也叫超类。

-用于继承的类，叫子类，也叫派生类。

-继承与被继承一定存在一个 is—a—的关系。



-继承的语法，参加oop-2.jpynb

-继承的特征

--所有的类都继承自object类，即所有的类都是object类的子类。

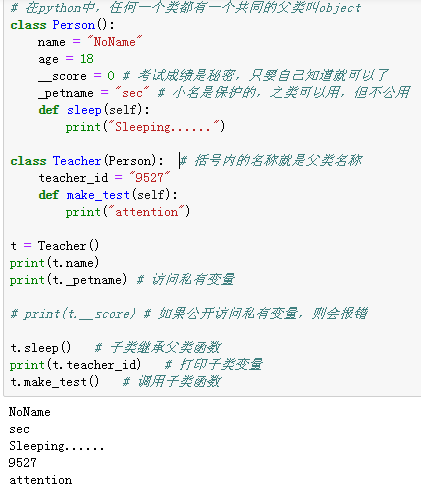
--子类一旦继承父类，则可以使用父类中除私有成员外的所有内容。

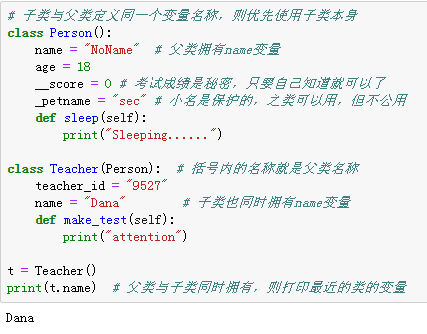
--子类继承父类后并没有将父类成员完全赋值到子类中，而是通过引用关系访问调用。

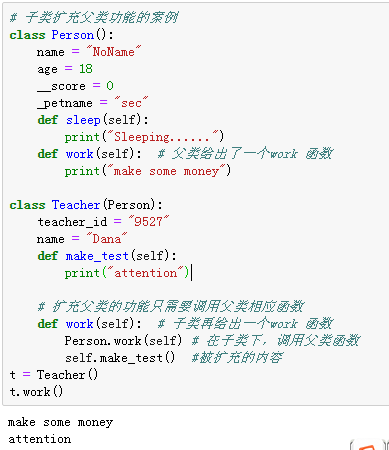
--子类中可以定义独有的成员属性和方法。

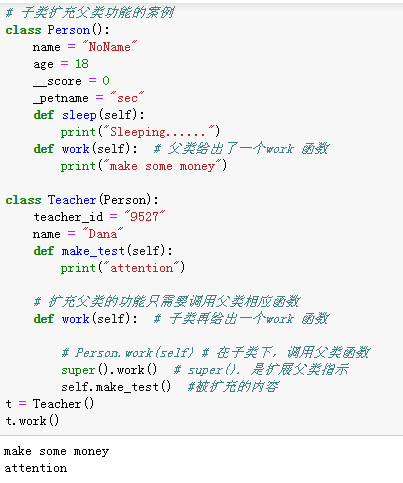
--子类中定义的成员与父类成员如果相同，则优先使用子类成员。

--子类如果想扩充父类的方法，可以在定义新方法的同时访问父类成员来进行代码重用，可以使用：【父类名.父类成员】 的格式来调用父类成员，也可以使用：【super().父类成员】 的格式来调用。









-继承变量函数的查找顺序问题

- 优先查找自己的变量

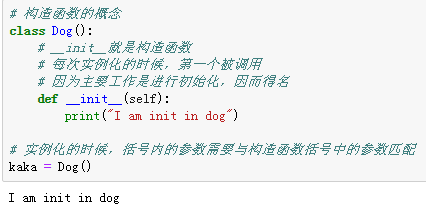
- 没有再查找父类

- 构造函数如果本类中没有定义，则自动查找调用父类的构造函数

- 如果本类有定义，那就不再继续往上查找了

-构造函数

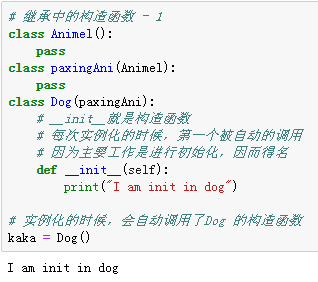
-构造函数是一类特殊的函数，在类进行实例化之前进行调用



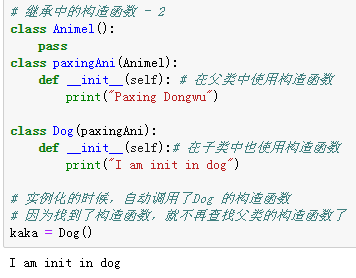
-构造函数的自动调用

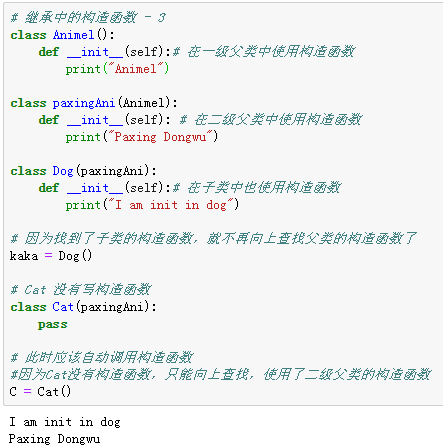
-如果定义了构造函数，则实例化时自动使用自己的构造函数，不查找父类的构造函数。

-如果没有定义，则自动查找父类的构造函数。

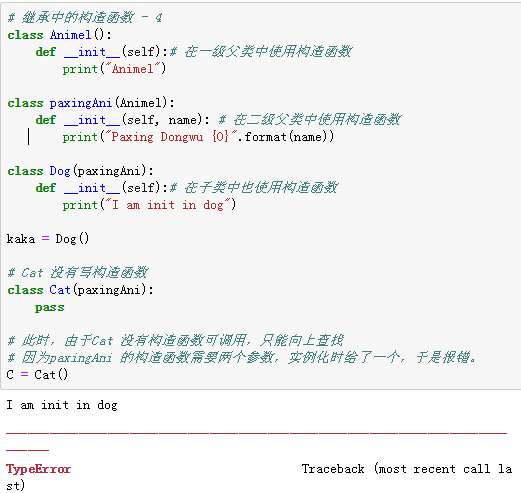


构造函数在子类中存在时，优先使用子类的构造函数，如果子类没有构造函数，则按层级向父类逐层逐级向上查找使用。





-如果子类没有定义，父类的构造函数带参数，则构造对象时的参数应该按父类参数构造。





-super

- super 是对父类的扩展指示

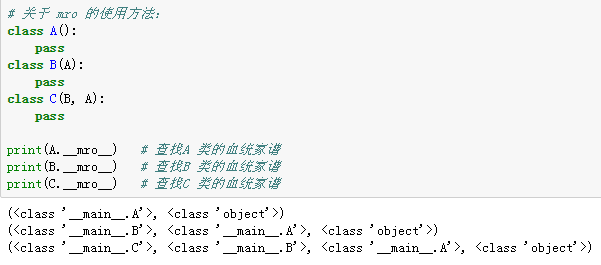
（子类如果想扩充父类的方法，可以在定义新方法的同时访问父类成员来进行代码重用，可以使用：【父类名.父类成员】 的格式来调用父类成员，也可以使用：【super().父类成员】 的格式来调用。）

- super 不是关键字，而是一个类

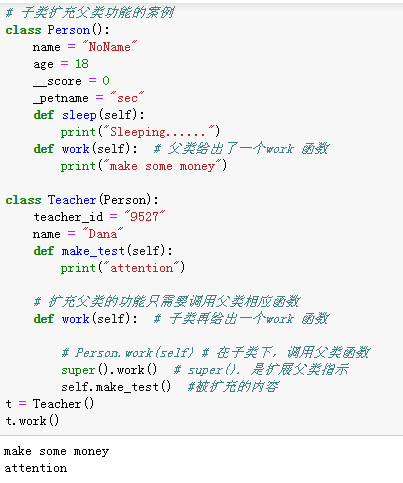
- super 的作用是获取MRO（MethodResolustionOrder）列表中的第一个类

- super 与父类之间没有任何实质性关系，但通过super 可以调用到父类

3-13课 OOP



- super 使用两个方法，参见在构造函数中调用父类的构造函数



-单继承和多继承

-单继承：每个类只能继承一个类

-多继承：每个类允许继承多个类

-单继承与多继承的优缺点

-单继承：

-优点：传承有序、逻辑清晰、语法简单、隐患少

-缺点：功能不能无限扩展，只能在当前唯一的继承链中扩展

-多继承：

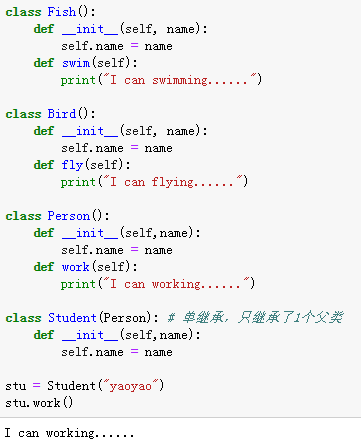
-优点：类的功能扩展方便

-缺点：继承关系混乱

多继承示例如下：

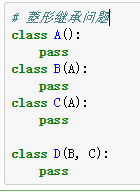


单继承示例如下：



-菱形继承问题 / 钻石继承问题

-多个子类继承之同一个父类，这些子类又被同一个类继承，于是继承关系图 形成一个菱形图谱。



- 关于多继承的MRO

- MRO就是多继承中，用于保存继承顺序的一个列表。

- python 本身采用C3算法来读多继承的菱形继承进行计算的结果

- MRO列表的计算原则：

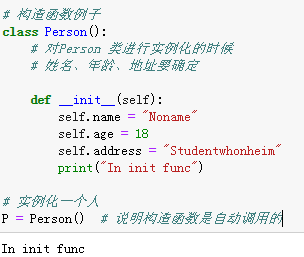
-之类永远在父类的前面

-如果多个父类，则根据继承语法中括号内类的书写顺序存放。

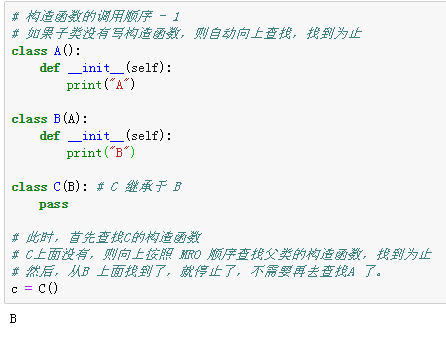
-如果多个类继承了同一个父类，孙子类中只会选取继承语法括号中第一个父类的父类。

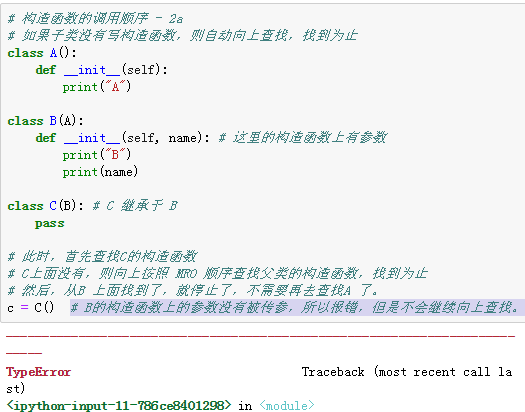
-构造函数

-在对象进行实例化的时候，系统自动调用的一个函数叫构造函数，通常吃函数用来对实例化对象进行



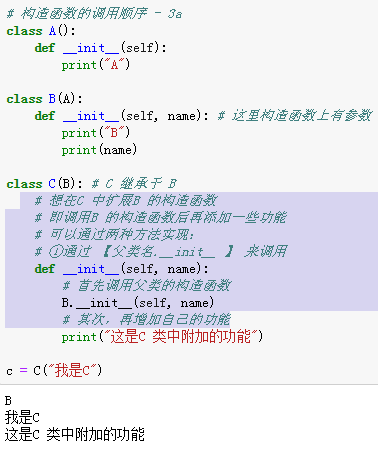
-构造函数一定要有，如果没有，就会自动向上查找，按照MRO顺序，直到找到为止。

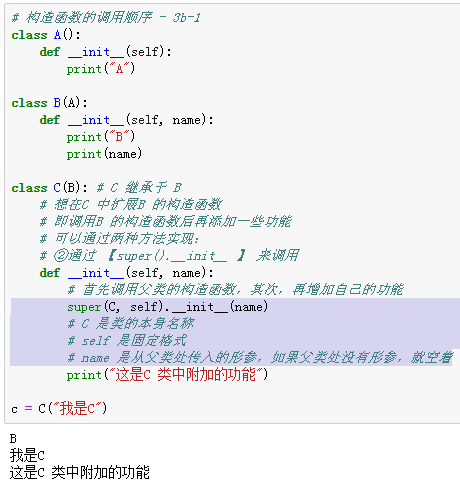


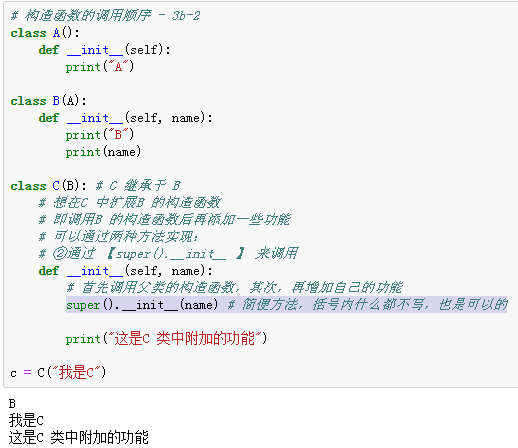




构造函数的扩展（类继承过程中，构造函数的继承扩展）







3.3 多态

-多态就是同一个对象在不同情况下以不同的状态出现

-多态不是语法，是一种设计思想

-多态性：一种调用方式，不同的执行效果

-多态：同一事物的多种形态，动物分成人类，狗类，猪类

-Mixin设计模式

- 主要采用多继承方式对类的功能进行扩展

-我们使用多继承语法来实现Mixin

-使用Mixin实现多继承的时候要非常小心

-首先他必须表示某一单一功能，而不是某个物品。

-职责必须单一，如果由多个功能，则写多个Mixin

-Mixin不能依赖于子类的实现

-子类即使没有继承这个Mixin类，也能照常工作，只是缺少了某个功能。

-优点

-使用Mixin可以在不对类进行任何修改的情况下，扩充功能

-可以方便的组织和维护不同功能组件的划分

-可以根据需要任意调整功能类的组合

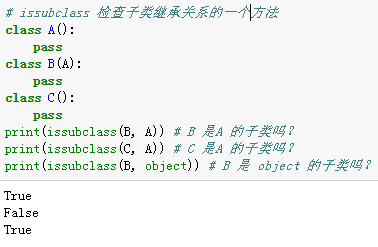
-可以避免创建很多新的类，导致类的继承混乱



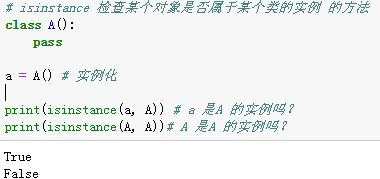


4 类相关函数

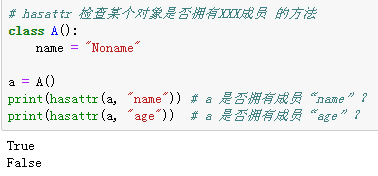
---- issubclass： 检测一个类是否为另一个类的子类



---- isinstance： 检测一个对象是否是一个类的实例



---- hasattr： 检测一个对象是否有成员XXX

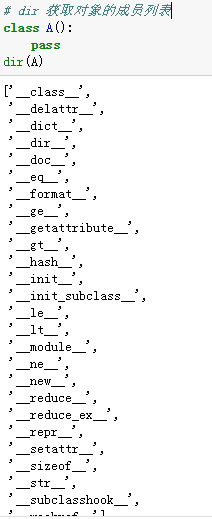


---- getattr： get attribute

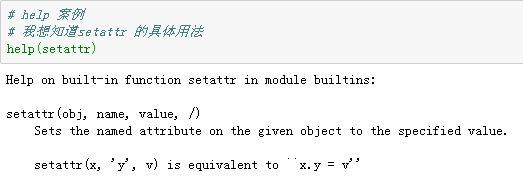
---- setattr： set attribute

---- delattr： delete attribute

---- dir: 获取对象的成员列表



不懂的可以使用help 进行查询，实例如下：



3-14课 OOP

5 类的成员描述符（属性）（建议去网上搜，有很多很多）

-类的成员描述符是为了在类中对类的成员属性进行相关操作而创建的一种方式。

- get：获取属性的操作

-set： 修改或者添加属性的操作

-delete： 删除属性的操作

-如果想使用类的成员描述符，大概有三种方法

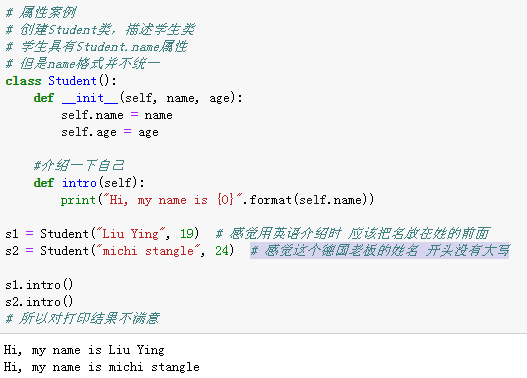
-使用类实现描述器

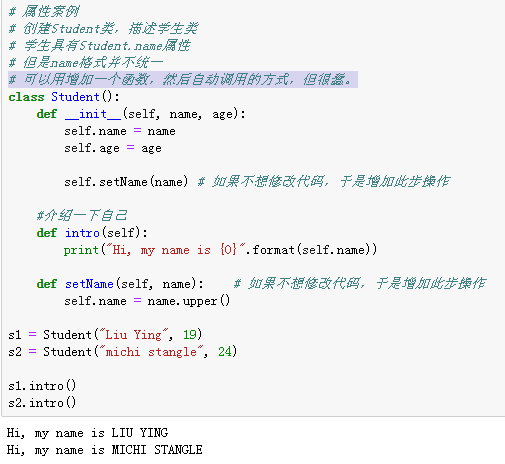
-使用属性修饰符

-使用property函数（注：3-15课 里还有提高课程的内容）

-property函数很简单

-property（fget， fset， fdel， doc）





Property的使用案例：



-无论哪种修饰符都是为了对成员属性进行相应的控制

- 类的方式： 适合多个类中的多个属性共用一个描述符。

- property： 使用当前类中使用，可以控制一个类中多个属性

-属性修饰符：使用于当前类中使用，控制一个类中的一个属性。

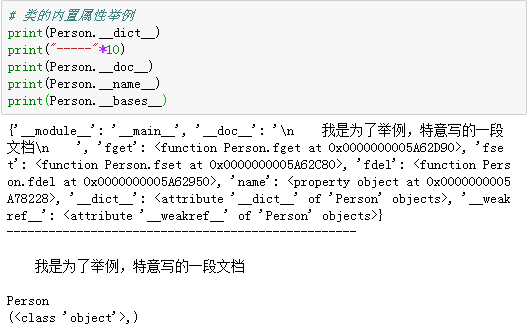
6 类的内置属性

\_\_dict\_\_: 以字典的方式显示类的成员组成。

\_\_doc\_\_: 获取类的文档信息

\_\_name\_\_: 获取类的名称，如果在模块中使用，获取模块的名称。

\_\_bases\_\_: 获取某个类的所有父类，以元组的方式显示

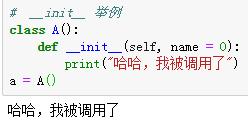


7 类的常用魔术方法

-魔术方法就是不需要人为调用的方法，基本是在特定的时刻自动触发。

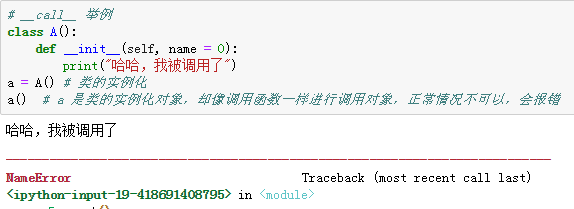
-魔术方法的统一特征，方法名被前后各两个下划线包裹

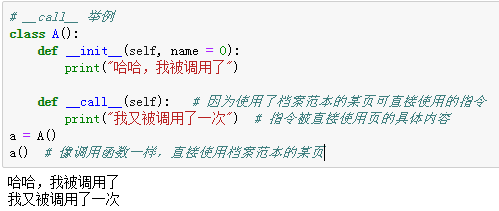
-“\_\_init\_\_”:构造函数（最常用）



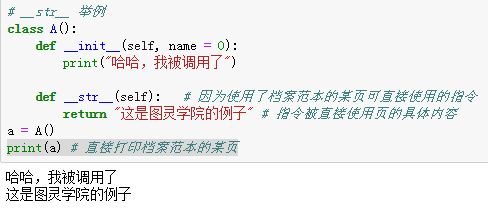
- “\_\_new\_\_”:对象实例化方法（最特殊），此函数特殊，一般不要使用。

- “\_\_call\_\_”: 像调用函数一样 来调用对象（常用）（个人理解：相当于 个人档案中直接取自范本中的某页，不做任何修改，直接拿来使用）





-“\_\_str\_\_”: 把对象当作字符串使用并调用（个人理解：与前面的\_\_call\_\_相似，相当于 个人档案中直接取自范本中的某页，不做任何修改，直接拿来使用，内容是以字符串的形式被返回打印出来）



-“\_\_repr\_\_”: 返回字符串，与前面的“\_\_str\_\_”具体区别请百度

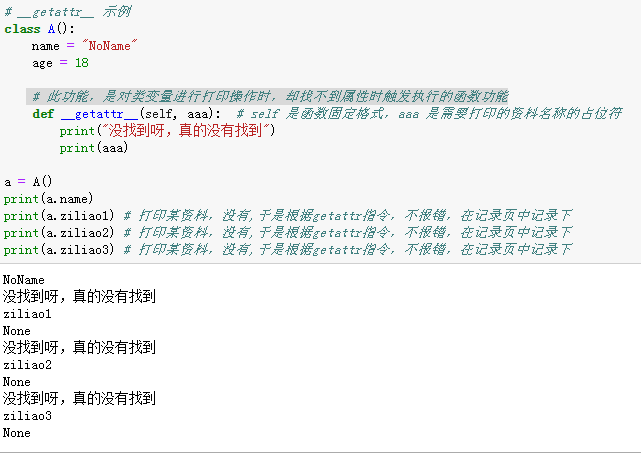
-描述符相关

-“\_\_set\_\_”  
-“\_\_set\_\_”

-“\_\_set\_\_”

-属性操作相关

-“\_\_getattr\_\_”：访问一个不存在的属性时触发（个人理解：相当于设置一份资料不全的记录页，当需要打印某一份资料时，发现该资料没有，于是不采用报错方式，而是在资料不全记录页中记录下，发生了资料不全的话，以及不全资料的名称）



-“\_\_setattr\_\_”：对成员属性进行设置的时候触发

-参数：

-self 用来获取当前对象

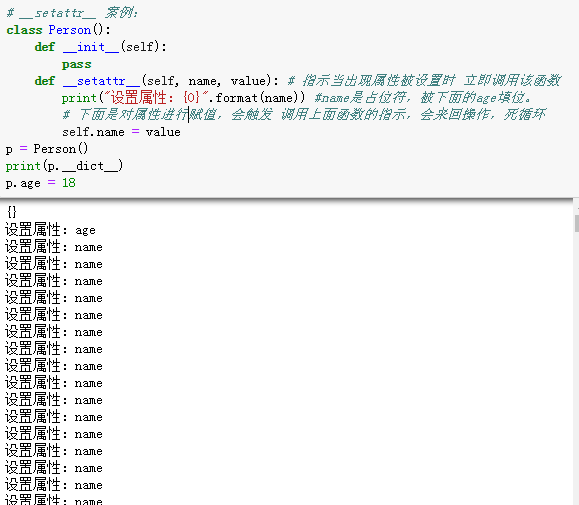
-被设置的属性名称，会以字符串的形式出现

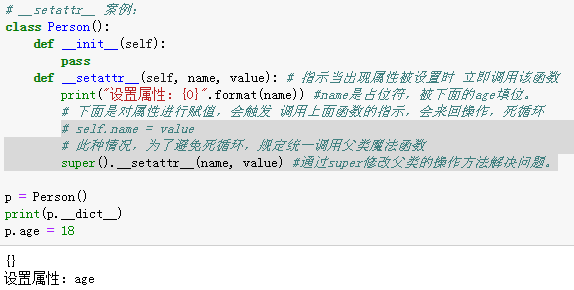
-需要对属性名称设置的值

-作用：进行属性设置的时候进行验证或者修改

-注意：在该方法中不能对属性直接进行赋值操作，否则会导致死循环。

-参看案例：





-运算分类相关魔术方法

☆：这一段没有太明白，没有学通透。

- \_\_gt\_\_ : 进行大于判断的时候触发的函数

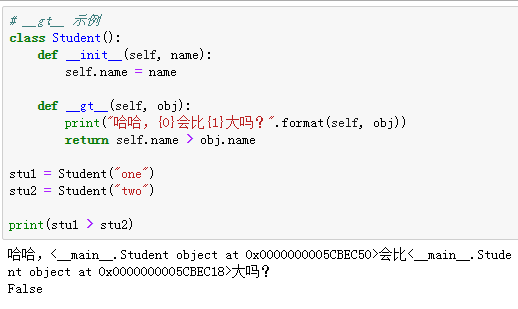
- 参数：

- self

-第二个参数是第二个对象

-返回值可以是任意值，推荐返回布尔值

-示例



8 类和对象的三种方法

①实例方法

-需要实例化对象才能使用的方法，使用过程中可能需要截止对象的其他对象的方法完成。

②类的方法

-不需要实例化

☆ 从今天开始，类的方法的函数前面必须要有 @classmethod这句话，而且必须在函数的括号中写入cls 。两者必须同时配合使用，如果只使用其中一个，会报错。

③静态方法

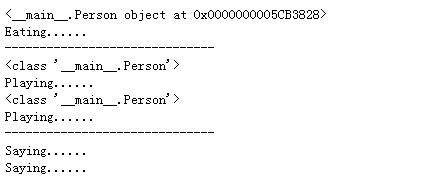
-不需要实例化，通过类直接访问

☆ 从今天开始，函数的括号中不写参数的函数就属于静态方法，静态方法的函数前面必须要有@staticmethod 这句话，否则不能被实例对象调用。

-参看案例

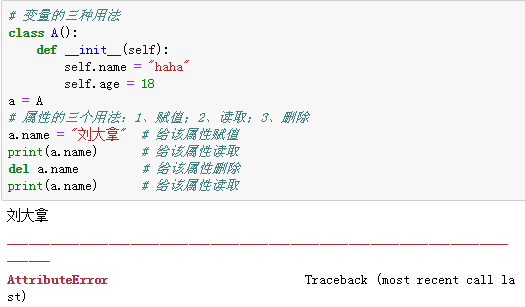
-三个方法的具体区别请自行百度





3-15课 python基础完结

变量的三种用法



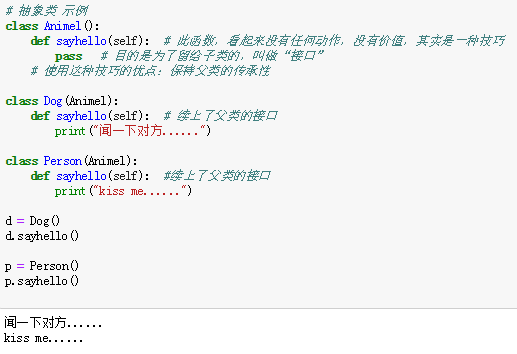
-- Property的应用提升讲解



9 穿插 画图软件的广告时间

10 抽象类

-抽象方法：没有具体实现内容的方法成为抽象方法



-抽象方法的主要意义是规范了子类的行为和接口

-抽象类的使用需要借助 a b c 模块。

import abc

-抽象类：包括抽象方法的类叫做抽象类，通常成为ABC类

-抽象类的使用

-抽象类可以包含抽象方法，也可以包含具体方法

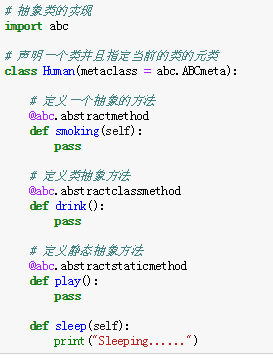
-抽象类中可以有方法，也可以有属性

-抽象类不允许直接实例化

-必须继承才可以使用，且继承的子类必须实现所有继承来的抽象方法。

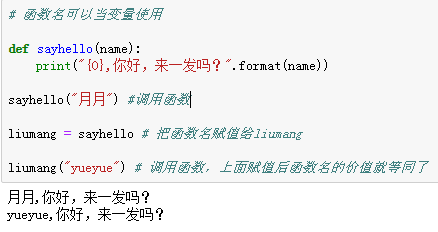
-假定子类没有实现所有继承的抽象方法，则子类也不能实例化。

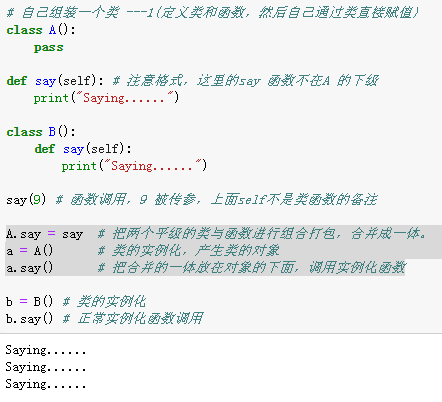
-抽象类的主要作用是设定类的标准，以便于开发的时候具有统一的规范。

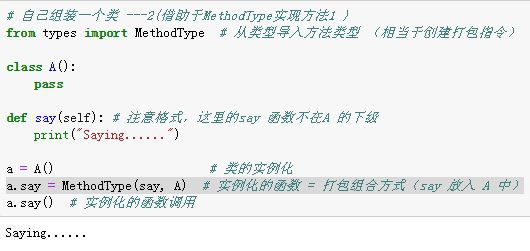


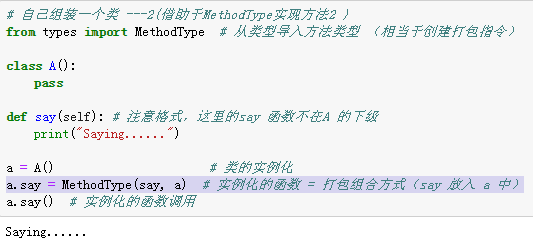
11 自定义类

-类其实是一个类定义和各种方法的自由组合，创建类的方法有以下方式：











①可以定义类和函数，然后自己通过类直接赋值

②可以借助于MethodType实现

③可以借助与type实现

④可以利用元类实现- MetaClass

-元类是类

-被用来创造孕育其他类（相当于母类）

4 -16课 包

1. 模块

-一个模块就是一个包含python代码的文件：后缀名是.py就可以，模块就是个python文件。

-为什么我们用模块

-程序太大，编写维护非常不方便，需要拆分

-模块可以增加代码重复利用的方式

-当作命名空间使用，避免命名冲突

-如何定义模块

-模块就是一个普通文件，所以任何代码可以直接书写。

-不过根据模块的规范，最好在模块中缩写以下内容。

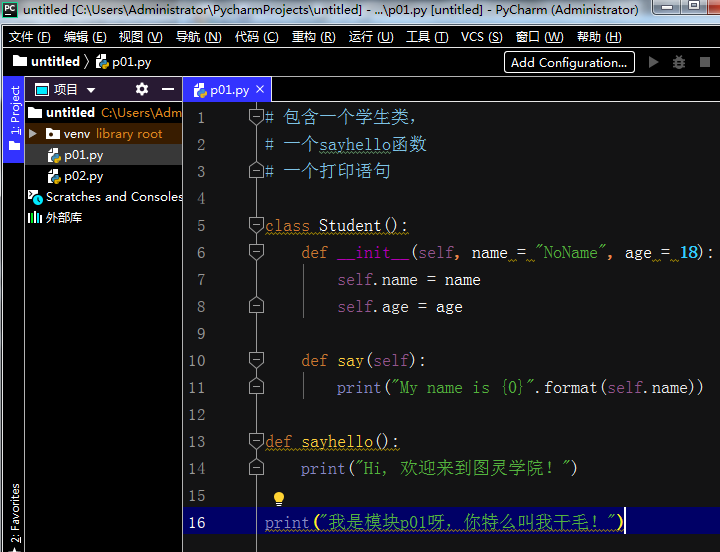
-函数（单一功能）

-类（相似功能的组合，或者类似业务模块）

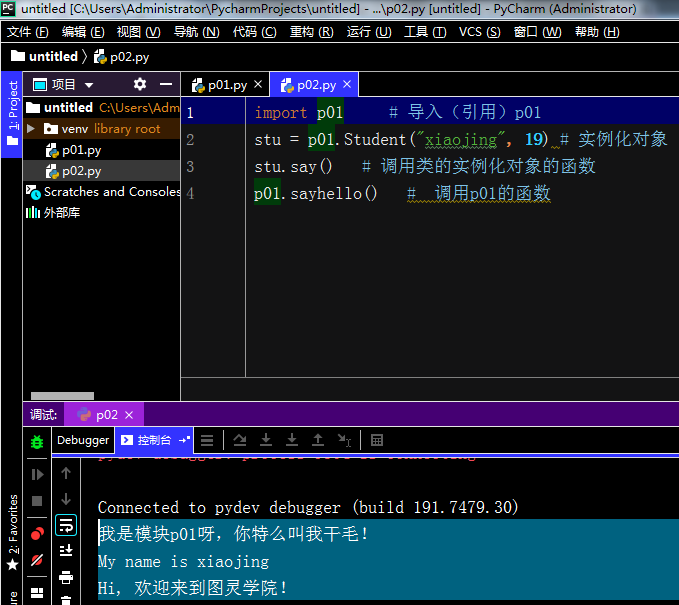
-测试代码

-如何使用模块

首先，要有一个可供被导入的模块，为了举例，创建python的p01文件及内部内容，如下：



然后，要有一个可以导入模块的文件，为了举例，创建python的p02文件及内部内容，如下：



-语法

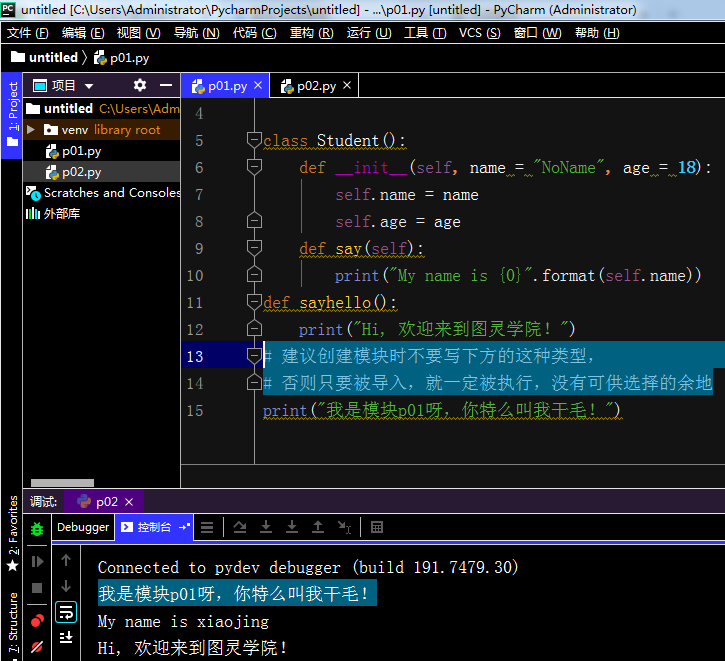
impor module\_name

module\_name.function\_name

module\_name.class\_name

注意下方的问题：

建议创建模板时不要使用调用模板就一定会被执行的代码，否则在导入模板时就已订购会被执行，没有可供选择的余地。

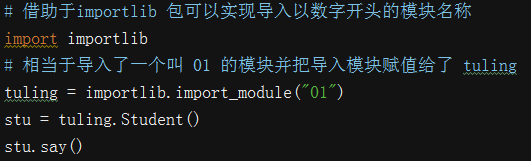


-模块直接导入

-通常模块名称是以数字开头的情况，是无法正常导入的。

-假如模块名称直接以数字开头，可以有两种方法进行导入：

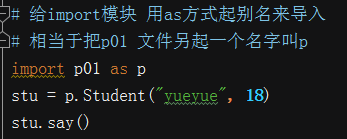
①借助importlib帮助



②给inport 模块用 as 方式起别名来导入

-导入的同时给模块起一个别名

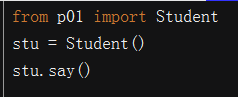
-其余用法跟第①种相同



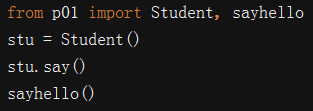
-间接的有选择性的导入

---- from module\_name import func\_name,class\_name

a、仅仅调入类的示例：



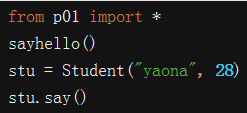
b、不仅调入类，同时调入与类同等级的函数的示例：



---- from module\_name import \*

上面的\* 表示所有内容，意味着指令导入模块的所有内容

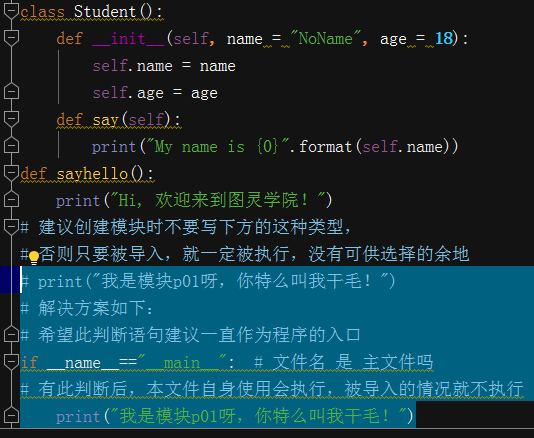
（使用这个格式省事，可以省略前缀，但是容易出现命名污染。）



-采用 if \_\_name\_\_ == “\_\_main\_\_” 的使用

-可以有效避免模块代码被导入的时候被动执行的问题

-建议所有的程序入口都以此代码为入口。



2 模块的搜索路径和存储

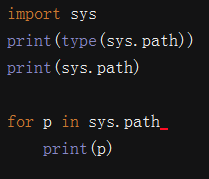
-什么是模块的搜索路径：

-加载模块的时候，系统会在哪些地方寻找此模块

-系统默认的模块搜索路径

import sys

sys.path 属性可以获取路径列表



-添加搜索路径

sys.path.append(dir)

-模块的加载顺序

1.搜索内存中已经加载好的模块

2.搜索python的内置模块

3.搜索sys.path路径

-包

-包是一种组织管理代码的方式，包里面存放的是模块

-用于将模块包含在一起的文件夹就是包

-自定义包的结构

/--- 包

/---/--- \_\_init\_\_.py 包的标志文件

/---/--- 模块1

/---/--- 模块2

/---/--- 子包（子文件夹）

/---/---/--- \_\_init\_\_.py 包的标志文件

/---/---/--- 子包模块1

/---/---/--- 子包模块2

-包的导入操作

-import package\_name

-直接导入一个包，可以使用\_\_init\_\_.py 中的内容

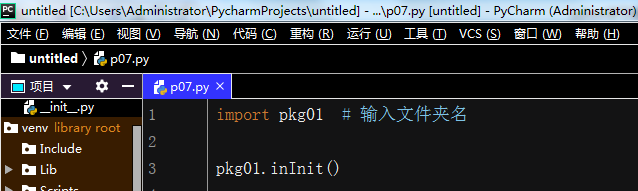
-使用方式是：

package\_name.func\_name

package\_name.class\_name

-此种方式的访问内容 示例如下：

首先新建一个包，包的名称为pkg01 ，然后在另一个文件中导入，见下方示例



-import package\_name as p

-就是采用起别名的方法，具体用法与作用方式，同上述简单导入一致

-注意的是此种方法是默认对\_\_init\_\_.py内容的导入

-import package.module

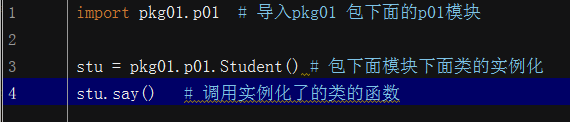
-导入包中某一个具体的模块

-使用方法

Package.module.func\_name

Package.module.class.func()

Package.module.class.var



-import package.Module as pm

方法与上面近似，就不再举例了

-from……..import 导入

-from package import module

-此种导入方法不执行\_\_init\_\_的内容

from pkg01 import p01

p01.sayHello（）

-from package import \*

-表示导入当前包\_\_init\_\_.py 文件中所有的函数和类

-使用方法

Func\_name()

Class\_name.func\_name()

Class\_name.var

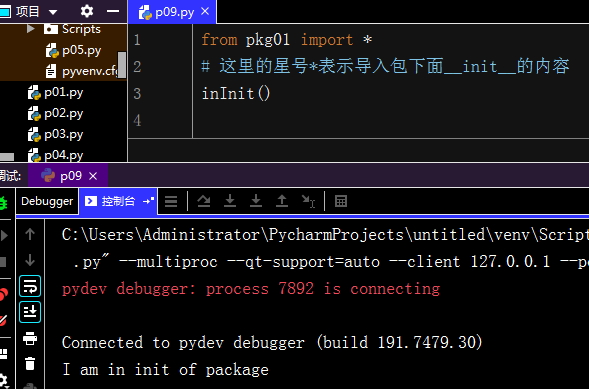
-from package.module import \*

-导入包中指定的模块的所有内容

-使用方法

func\_name()

class\_name.func\_name()



-在开发环境中经常会，所以用其他模块，可以在当前包中直接导入其他模块中的内容

-import 完整的包或者模块的路径

-“\_\_all\_\_”的用法

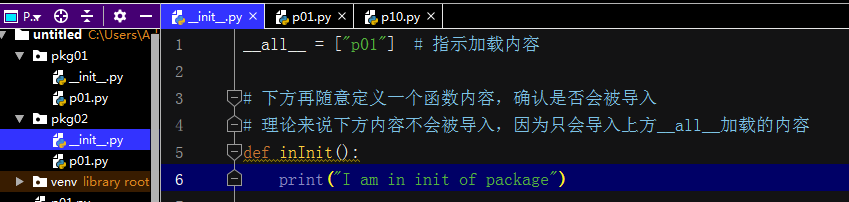
-在使用from package import \* 的时候， \* 可以导入的内容

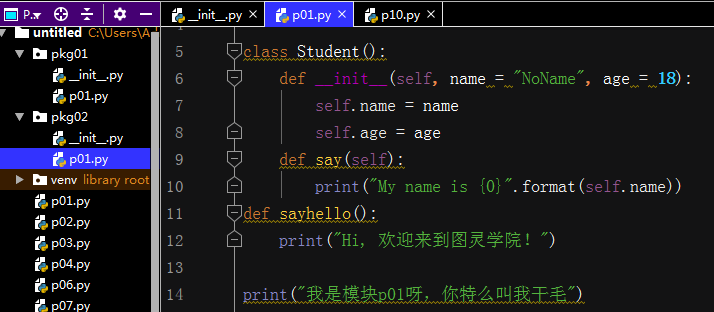
-“\_\_init\_\_.py” 中如果文件为空，或者没有“\_\_all\_\_”,那么只可以把“\_\_init\_\_”中的内容导入。

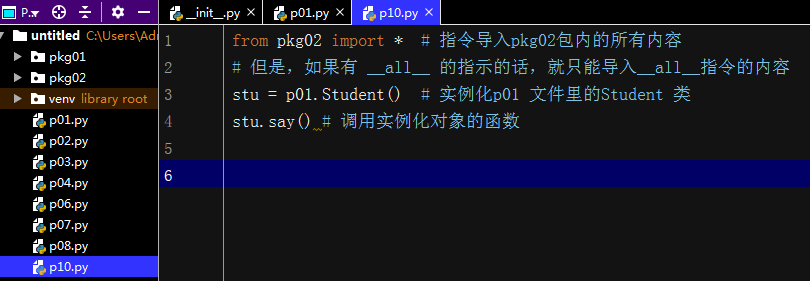
-“\_\_init\_\_”如果设置了“\_\_all\_\_”的值，那么则按照“\_\_all\_\_”指定的子包或者模块进行，如此则不会再载入 “\_\_init\_\_”中的内容

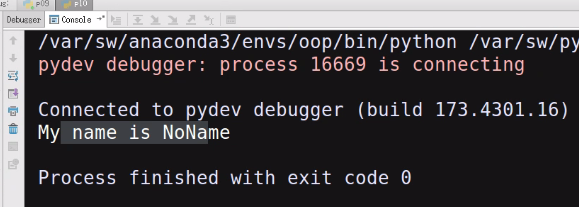
☆ 注意： 一旦在包下面的\_\_init\_\_文件中写入了“ \_\_all\_\_=[“module1”, “module2”, “package1”......] ”的内容，就是说明只载入[ ]内的这些内容，其余内容一概无关了。

示例如下： pkg02包内建有\_\_init\_\_ 文件和 p01 文件， 在其他文件包内调用p01内容。









-命名空间

-用于区分不同位置不同功能但相同名称的函数或者变量的一个特定前缀

-作用是防止命名冲突

setName()

Student.setName()

Dog.setName()

4-16课 异常使用

-异常

-光义上的错误分为错误和异常

-错误指的是可以人为避免

-异常是指在语法逻辑正确的前提下，出现的问题

-在python中，异常是一个类，可以处理和使用

-异常处理

-不能保证程序永远正确运行

-但是，必须保证程序在最坏的情况下得到的问题被妥善处理

-python的异常处理模块全部语法为：

Try:

尝试实现某个操作，

如果没有出现异常，任务就就可以完成；

如果出现异常，将异常从当前代码块扔出去尝试解决异常。

Except 异常类型1：

解决方案1：用于尝试在此处处理异常解决问题。

Except 异常类型2：

解决方案2：用于尝试在此处处理异常解决问题。

Except （异常类型1，异常类型2 ……）

解决方案：针对多个异常使用相同的处理方式。

Except：

解决方案：所有异常的解决方案。

Else：

如果没有出现任何异常，将会执行此处代码

Finally：

管你有没有异常都要执行的代码

-流程

1、执行try下面的语句

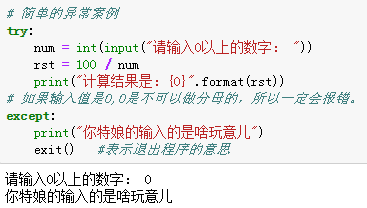
2、如果出现异常，则在except语句里查找对应异常并进行处理

3、如果没有出现异常，则执行else语句内容

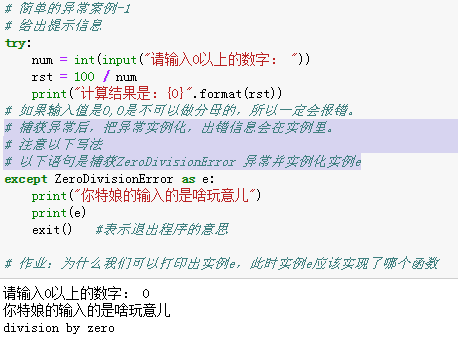
4、最后，不管是否出现异常，都要执行finally语句

-除except（最少一个）以外，else 和finally 可选

简单异常案例如下：



打印出异常的提示信息的案例：



程序公示错误提示的案例：



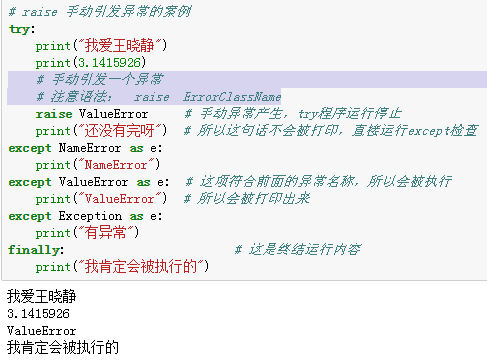
异常的最终拦截 示例



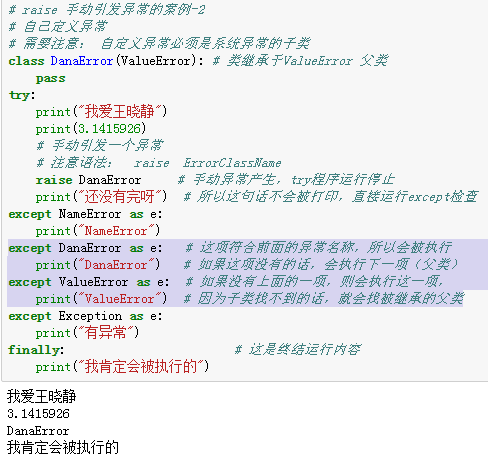
-用户手动引发异常

-当某些情况，用户希望自己引发一个异常的时候，可以使用

-raise 关键字来引发异常



涉及继承类的异常处理



-else 语句案例



-关于自定义异常

-只要是raise异常，则推荐自定义异常

-在自定义异常的时候，一般包含以下内容：

-自定义发生异常的异常代码

-自定义发生异常后的问题提示

-自定义发生异常的行数

-最终的目的是，一旦发生异常，方便程序员快速定位错误现场

3-17课 异常使用

-常用模块

-calendar

-time

-datetime

-timeit

-os

-shutil

-zip

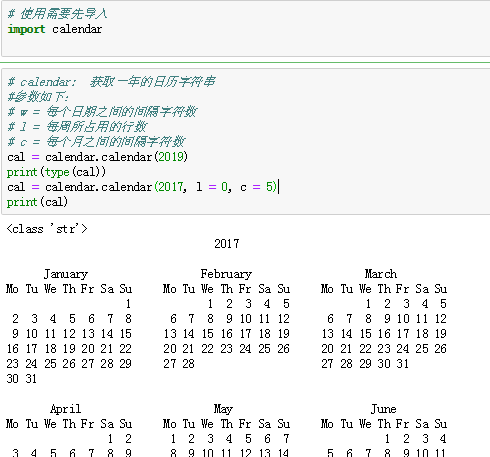
-math

-string

-上述所有模块使用理论上都应该先导入，string是特例

-calendar，time，datetime的区别参考中文意思

-calendar：与日历相关的模块

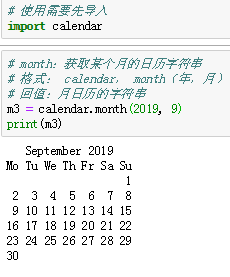


-isleap：判断某一年是否闰年

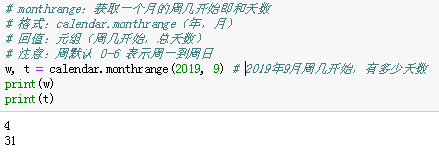
-leapdays：获取指定年份之间的闰年的个数



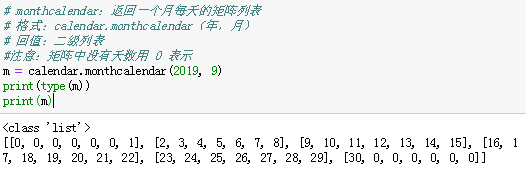
-month：获取某个月的日历字符串



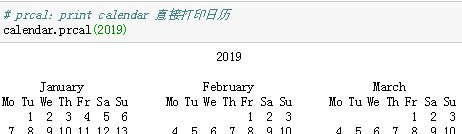
-monthrange：获取一个月的周几开始即和天数



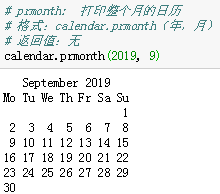
-monthcalendar：返回一个月每天的矩阵列表



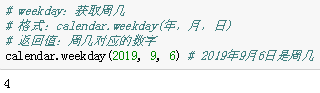
-prcal：print calendar 直接打印日历



-prmonth: 打印整个月的日历



-weekday：获取周几



-time模块

-时间戳

-一个时间表示，根据不同语言，可以是整数或者浮点数

-是从1970年1月1日0时0分0秒到现在经历的秒数

-如果表示的时间是1970年以前或者太遥远的未来，可能出现异常

-32位操作系统能够支持到2038年

-UTC时间

-UTC又称为世界协调时间，以英国的格林尼治天文台所在地区时间为参考时间，也叫世界标准时间。

-中国的时间是 UTC+8 东八区

-夏令时

-夏令时就是在夏天的时候将时间调快一个小时，本意是督促大家早睡早起省蜡烛，本质没变，还是24小时。

-时间元组

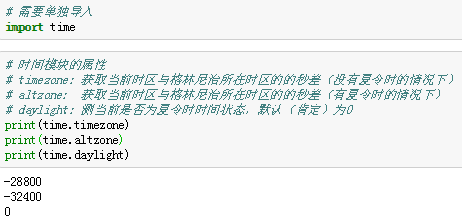
-一个包含时间内容的普通元组



-timezone: 获取当前时区与格林尼治所在时区的的秒差（没有夏令时的情况下）

-altzone: 获取当前时区与格林尼治所在时区的的秒差（有夏令时的情况下）

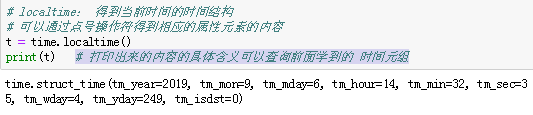
- daylight: 测当前是否为夏令时时间状态，默认（肯定）为0



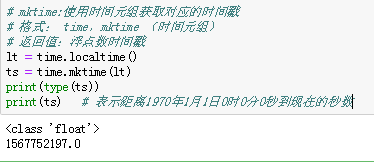
-时间戳



-localtime： 得到当前时间的时间结构

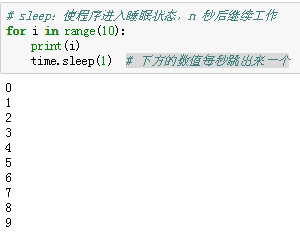


-mktime:使用时间元组获取对应的时间戳

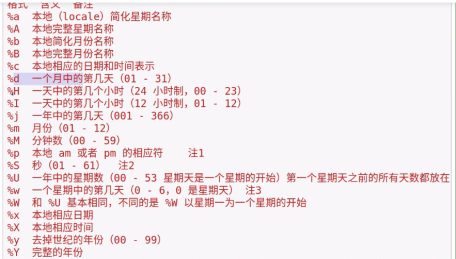


-clock: 获取cpu时间，3.0-3.3版本直接使用，3.6调用有问题

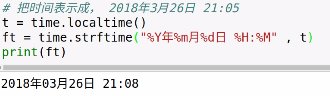
-sleep：使程序进入睡眠状态，n 秒后继续工作



-strftime：将时间元组转化为自定义的字符串格式



使用示例如下：



-datetime模块

-datetime 提供日期和时间的运算和表示

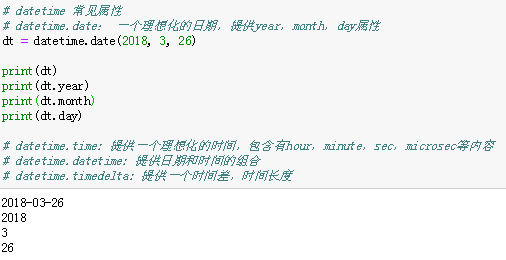
- datetime的常见属性如下：

- datetime.date： 一个理想化的日期，提供year，month，day属性

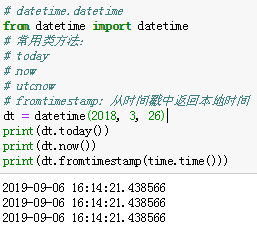
- datetime.time: 提供一个理想化的时间，包含有hour，minute，sec，microsec等内容

- datetime.datetime: 提供日期和时间的组合

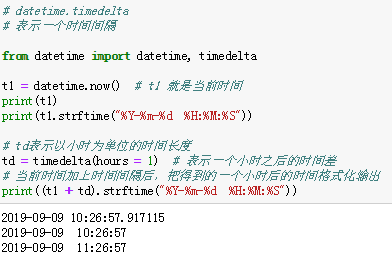
- datetime.timedelta: 提供一个时间差（时间间隔），时间长度



- datetime.datetime：提供日期和时间的组合



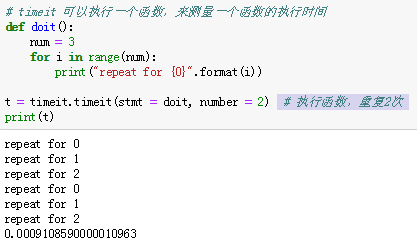
- datetime.timedelta的示例：提供一个时间差（时间间隔），时间长度

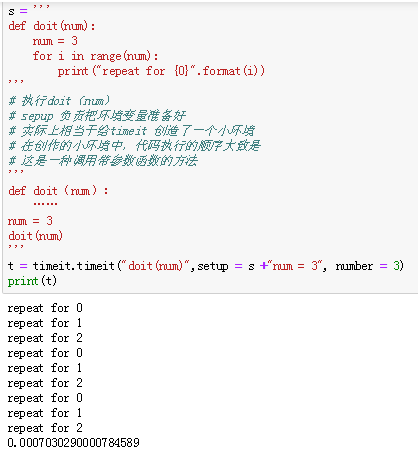


-timeit：时间测量工具

-可以测量程序运行时间间隔



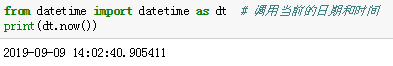




4-18课 常用包介绍

-datetime.datetime 模块

-提供比较好用的时间而已



-os:操作系统相关

-跟操作系统相关，主要是文件操作

-与系统相关的操作，主要包含在三个模块里

-os，操作系统目录相关

-os.path，系统路径相关操作

-shutil， 高级文件操作，目录树的操作，文件赋值，移动

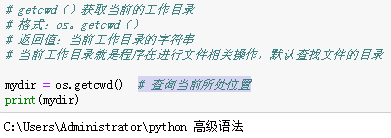
-路径：

-绝对路径：总是从根目录上开始

-相对路径：基本以当前环境为开始的一个相对的地方

os模块

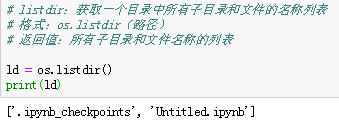
- getcwd：获取当前的工作目录



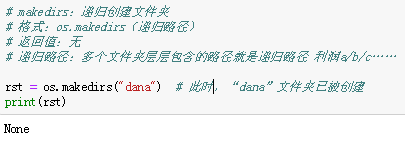
- chair： 改变当前的工作目录

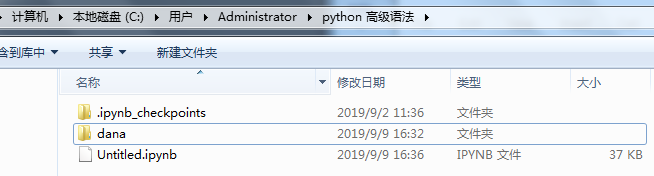


-listdir：获取一个目录中所有子目录和文件的名称列表

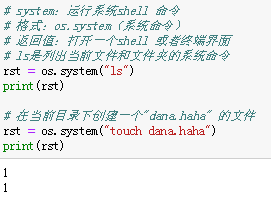


-makedirs：递归创建文件夹





-system：运行系统shell 命令



-getenv：获取指定的系统环境变量值



-exit：退出当前程序



-值部分

-os.curdir: current dir,当前目录

-os.pardir: parent dir,父亲目录

-os.sep: 当前系统的路径分隔符（不要手动写，系统不同写法不一样就不能用，将来要学习使用os.path模块，是一种可以替换的方式）

- windows的分隔符是“\”

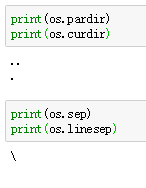
-unix,linux,macos的分隔符是“/”

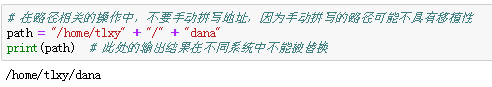
-os.linesep: 当前系统的换行符号（不要手动写，系统不同写法不一样就不能用，将来要学习使用os.path模块，是一种可以替换的方式）

- windows的换行符是“\r \n”

-unix,linux,macos的换行符是“\n”

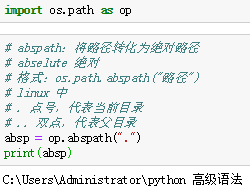
-os.name: 当前系统名称



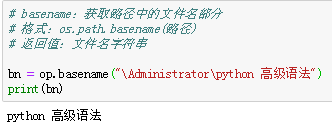


-os.path 模块，跟路径相关的模块

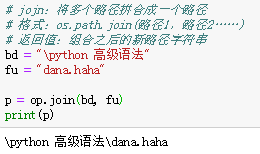
-abspath：将路径转化为绝对路径



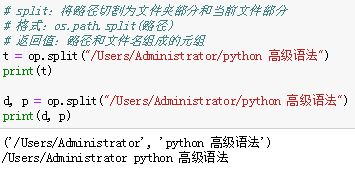
-basename：获取路径中的文件名部分



-jojn：将多个路径拼合成一个路径



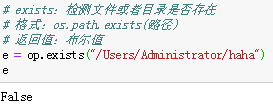
-split：将路径切割为文件夹部分和当前文件部分



-isdir：检测是否是目录



-exists：检测文件或者目录是否存在



-shutil 模块

-copy：复制文件

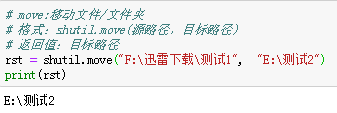


-copy2：复制文件，保留原数据（文件信息）

-copyfile：将一个文件中的内容复制到另一个文件当中去



-move：移动文件/文件夹

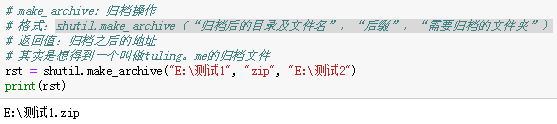


-归档和压缩

-归档：把多个文件或者文件夹合并到一个文件当中

-压缩：用算法把多个文件或者文件夹无损或者有损合并到一个文件当中

-make\_archive:归档操作



-unpack\_archive:解包操作

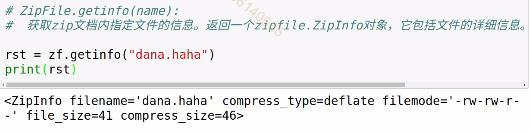


-ZIP-压缩包

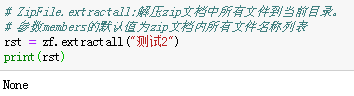
-包的名称叫zipfile



-ZipFile.getinfo(name): 获取zip文档内指定文件的信息。返回一个zipfile.ZipInfo对象，包括文件的详细信息



-ZipFile.extractall:解压zip文档中所有文件到当前目录。参数members的默认值为zip文档内所有文件名称列表



-random

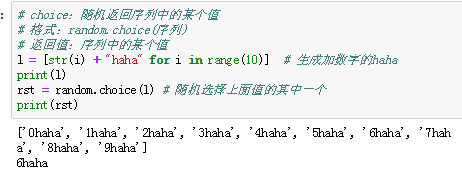
-随机数

-所有的随机模块都是伪随机

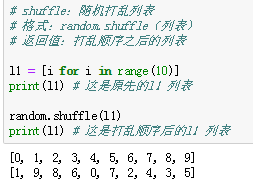
-random：获取0-1之间的随机小数



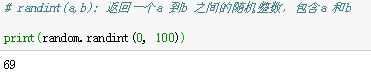
-choice：随机返回序列中的某个值



-shuffle：随机打乱列表



-randint(a,b)：返回一个a到b之间的随机整数，包含a和b



4-19课 函数式编程

-python语言的高级特性

-函数式编程（functional programming）

-基于lambda演算的一种编程方式

-程序中只有函数

-函数可以作为参数，同样可以作为返回值

-纯函数式编程语言：LISP , Haskell

-Python函数式编程只是借鉴函数式编程的一些特点，可以理解成一半函数式，一半是python

-本阶段需要讲述的内容如下：

-高阶函数

-返回函数

-匿名函数

-装饰器

-偏函数

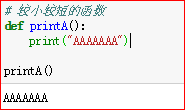
-lambda表达式

-函数：最大程度复用代码

-存在问题：如果函数很小，很短，则会造成啰嗦，请看下面的例子

-如果函数被调用次数少，则会造成浪费

-对于阅读者来说，造成阅读流程的被迫中断

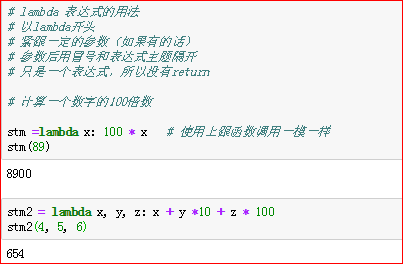


-lambda表达式（匿名函数）：

-一个表达式，函数体相对简单

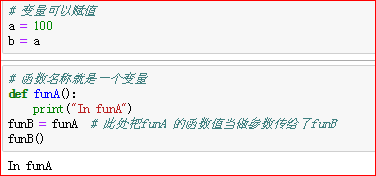
-不是一个代码块，仅仅是一个表达式

-可以有参数，有多个参数也可以，用逗号隔开，请看下面的例子



-高阶函数

-把函数作为参数使用的函数，叫高阶函数

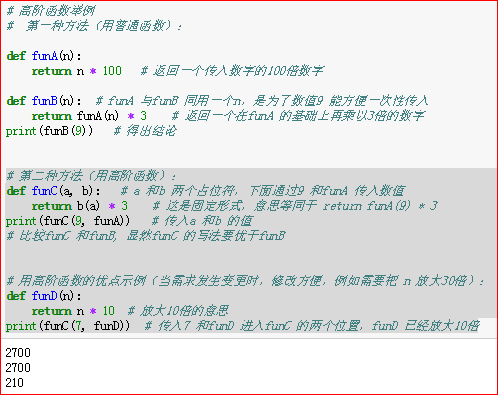


-以上代码得出的结论：

-函数名称是变量

-funB 和funA 只是名称不一样而已

-既然函数名称是变量，则应该可以被当做参数传入另一个函数



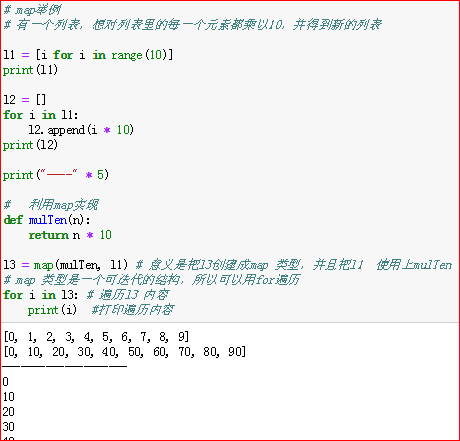
-系统高阶函数-map

-原意就是映射，即把集合或者列表的元素，每一个元素都按照一定规则进行操作，生成一个新的列表或者集合

- map函数就是系统提供的具有映射功能的函数，返回值是一个迭代对象。

-map举例

-有一个列表，想对列表里的每一个元素都乘以10，并得到新的列表



-reduce

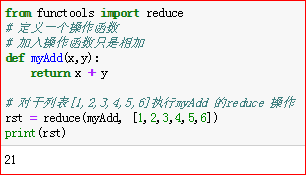
-原意是归并，缩减

-把一个可迭代的对象最后归并成一个结果

-对于作为参数的函数要求：必须有两个参数，必须有返回结果

-reduce（[1,2,3,4,5]）==f(f(f(f(1,2,),3),4),5)

-reduce需要导入functools包



4-20课 函数式编程

-filter函数

-过滤函数：对一组数据进行过滤，符合条件的数据会生成一个新的列表并返回

-与map 相比较：

-相同：都对列表的每一个元素逐一进行操作

-不同：

-map 会生成一个跟原来数据相对应的新队列

-filter 不一定，只要符合条件的才会进入新的数据集合

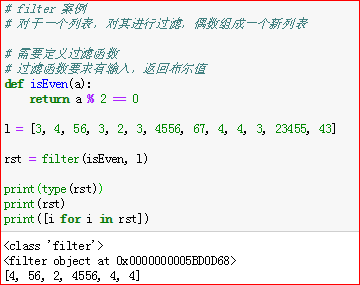
-filter函数怎么写：

-利用给定函数进行判断

-返回值一定是个布尔值

-调用格式：filter(f, data)，f是过滤函数，data是数据

- filter 示例

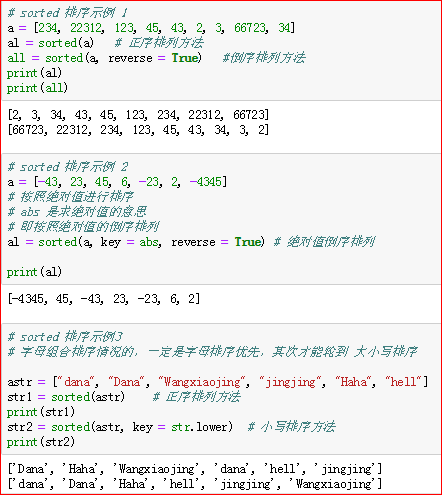


-高阶函数---排序

-把一个序列按照给定算法进行排序

-key：在排序前对每一个元素进行key函数运算，可以理解成按照key函数定义的逻辑进行排序

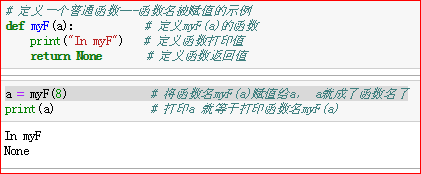
-sorted 示例



-返回函数

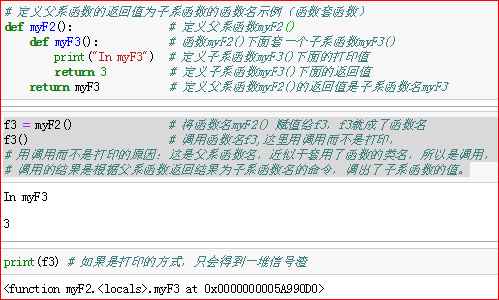
-函数可以返回具体的值

-也可以返回一个函数作为结果



-函数作为返回值返回，被返回的函数在函数体内定义

意思就是：定义父系函数的返回值为子系函数的函数名（函数套函数）



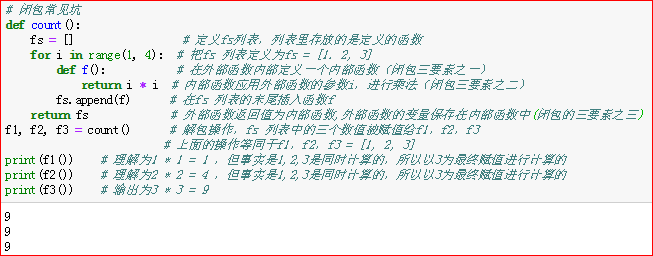
返回函数内部进行运算的例子（套嵌函数内部进行运算）



-闭包（closeure）

-一个函数的内部再套用一个函数，并且内部的函数应用外部函数的参数或者局部变量，当内部函数被当作外部函数的返回值的时候，相关参数和变量保存在返回的内部函数中，这种结果就叫做闭包。

-上面定义的myF4是一个标准的闭包结构

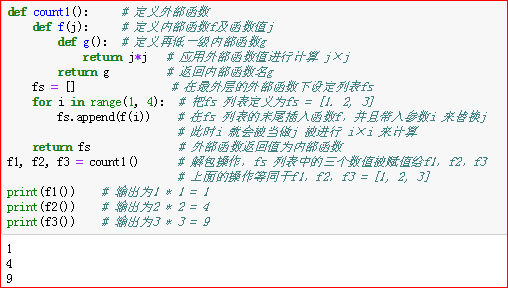


-对以上出现的问题进行分析：

-造成上述状况的原因是，返回函数应用了变量i，i并非立即执行，而是等到三个函数都返回的时候才统一使用，此时i已经变成了3，最终调用的时候，都返回的是3×3

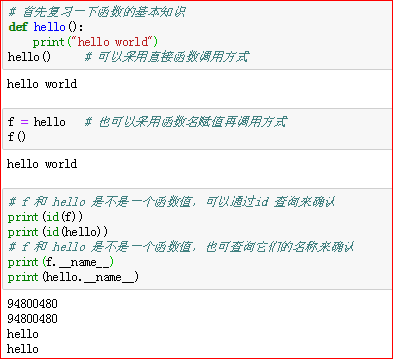
-此问题被描述成：返回闭包时，返回函数不能引用任何循环变量

-解决方案：再创建一个函数，用该函数的参数绑定循环变量的当前值，无论该循环变量以后如何改变，已经绑定了的函数参数值不再改变。



-装饰器

-讲解装饰器之前先复习一下函数的基本知识，如下



-现在对上面的示例加入新的需求：

-对hello 功能进行扩展，每次打印hello 之前都要打印一下当前系统时间。

-并且不能为了实现上述功能，而改动现有代码

-那么，就需要使用装饰器

-装饰器（decrator）

-在不改动函数代码的基础上无限制扩展函数功能的一种机制，从本质上来讲，装饰器就是一个返回函数的高阶函数。

-装饰器的要求：要以函数作为参数，并且要返回一个函数

-装饰器的使用： 使用@语法 即在每次要扩展的函数定义前使用“@+函数名”

☆ 装饰器是闭包的一种，但不是所有闭包就是装饰器，装饰器一定要注意以下三点：

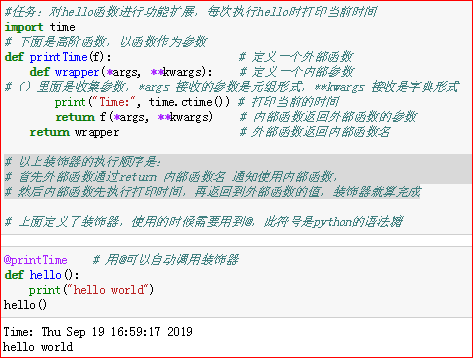
1、与闭包一样，会存在内部函数与外部函数

2、内部函数返回值是外部函数的参数（括弧内的内容）；

3、外部函数返回值是内部函数的函数名；

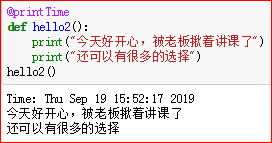
任务：

对hello函数进行功能扩展，每次执行hello时打印当前时间，示例如下：



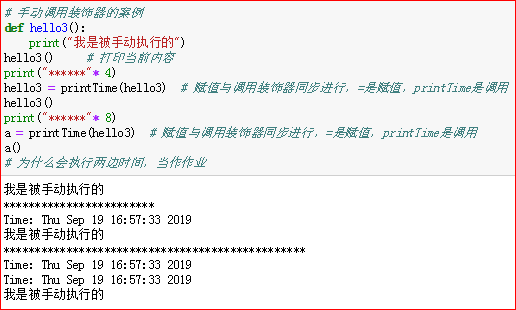
-装饰器的好处：一点定义，则可以装饰任意函数

-一旦被其装饰，则可把装饰器的功能直接添加到定义函数的功能上去。



-上面是对函数的装饰使用了系统定义的语法糖。

-下面开始手动执行下装饰器

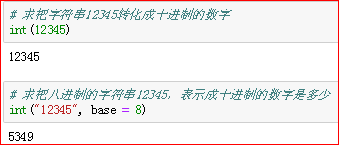


-偏函数

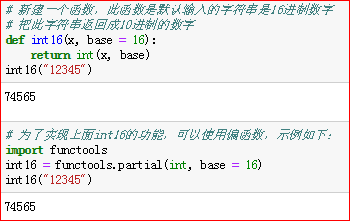
-参数固定的函数，相当于一个有特定参数的函数体

-functools.partial 的作用是，把一个函数的某些参数固定下来，返回一个新函数。

-在正式介绍偏函数之前，先进行int 的使用介绍



-下面进行偏函数与普通函数使用方法的比较操作：

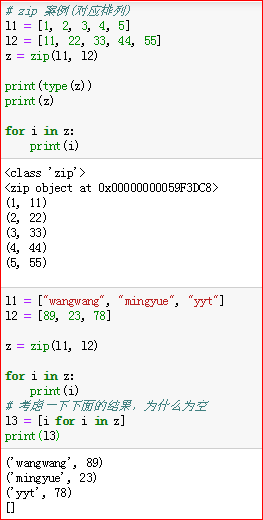


4-21课 高级语法和调试

-补充几个高级函数

-zip

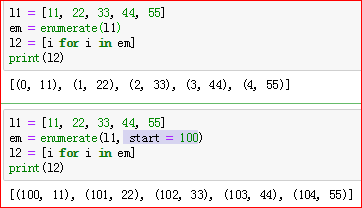
-把两个可迭代内容生成一个可迭代的tuple元素类型组成的内容



-enumerate

-跟zip功能比较像

-对可迭代对象里的每一个元素，配上一个索引（序号），然后索引和内容构成tuple类型



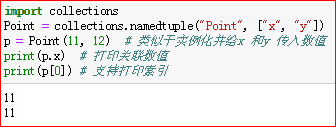
-collections模块

-namedtuple

-deque

-首先介绍namedtuple，他是一个tuple类型

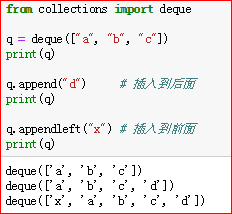
-而且是一个可命名的tuple





- deque

-比较方便的解决了频繁删除插入带来的效率问题



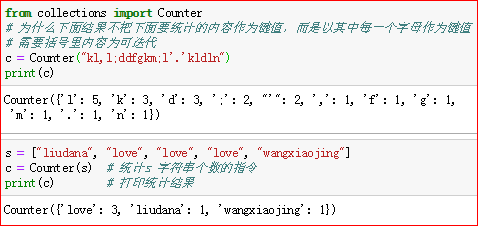
-defaultdict

-当直接读取dict不存在的属性时，直接返回默认值。



-Counter

-统计字符串个数



-调试技术

-调试流程：单元测试→集成测试→交给测试部

-分类：

-静态调试

-动态调试

-pdb调试：python 调试库

-pycharm调试

-run/debug模式

-断点：程序的某一行在debug模式下，遇到断点就会暂停。

-单元测试

4-22课 file

-文件

-长久保存信息的一种数据信息集合

-常用操作

-打开关闭（文件一旦打开，需要关闭操作）

-读写内容

-查找

-open函数

-open函数负责打开文件，带有很多参数

-第一个参数：必须有，文件的路径和名称

-mode：表明文件用什么方式打开

-r：以只读方式打开

-w：写方式打开，会覆盖以前的内容

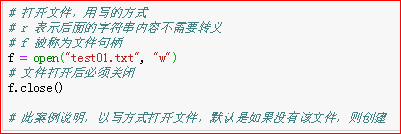
-x：创建方式打开，如文件已经存在，报错

-a：append方式，以追加的方式对文件内容进行写入

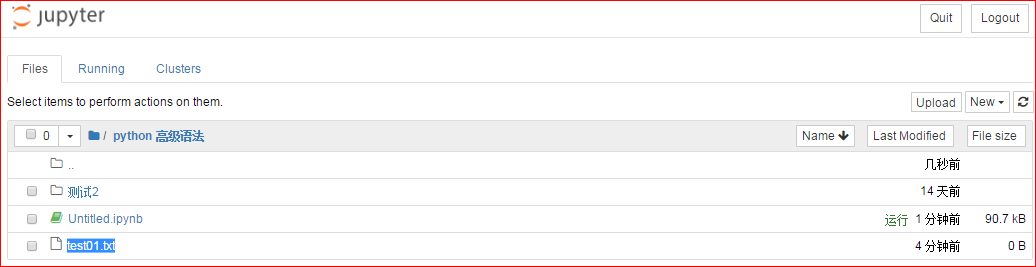
-b：binary方式，二进制方式写入

-t：文本方式打开

-+：可读写



在上级文档原本没有该名称的文件，但是因为以上操作，被创建了一个该文件



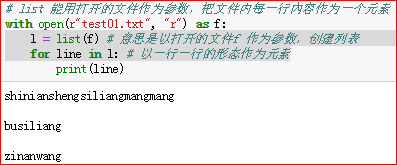
-with语句

-with语句使用的技术是一种成为上下文管理协议的技术（ContextManagementProtocal）

-自动判断文件的作用域，自动关闭不再使用的、已打开的文件句柄



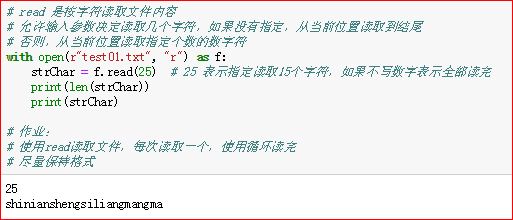
-list 能用打开的文件作为参数，把文件内每一行內容作为一个元素



-read 是按字符读取文件内容

-允许输入参数决定读取几个字符，如果没有制定，从当前位置读取到结尾

-否则，从当前位置读取制定个数的数字符



-seek（offset，from）

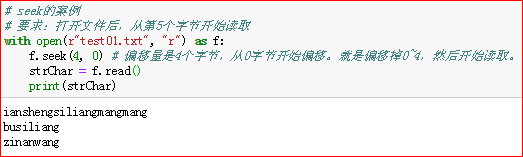
-移动文件的读取位置，也叫读取指针

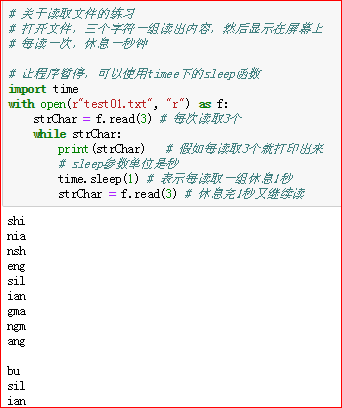
-from的取值范围：

-0：从文件头开始偏移

-1：从文件当前位置开始偏移

-2：从文件末尾开始偏移

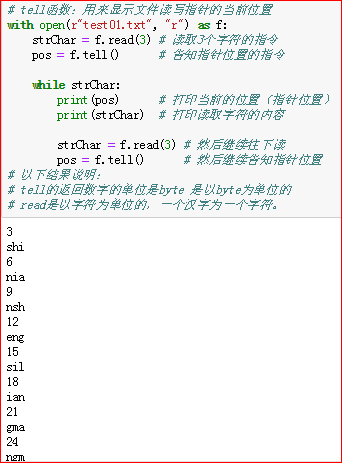




-tell函数：用来显示文件读写指针的当前位置

-tell的返回数字的单位是byte 是以byte为单位的

-read是以字符为单位的，一个汉字为一个字符。



-文件的写操作-write

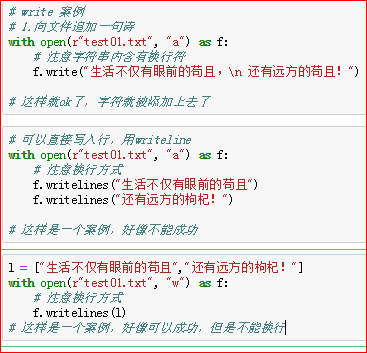
-write（str）：把字符串写入文件

-writeline（str）：把字符串按行写入文件

-区别：

-write函数参数只能是字符串

-writeline参数可以是字符串，也可以是字符序列



-持久化 –pickle

-序列化（持久化，落地）：把程序运行中的信息保存在磁盘上。

-反序列化：序列号的逆过程

-pickle：python提供的序列化模块

-pickle.dump：序列化

-pickle.load：反序列化

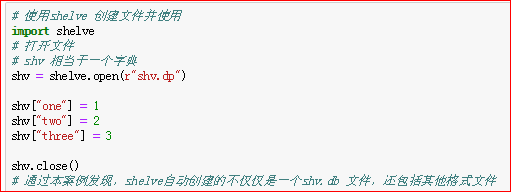


-持久化—shelve

-持久化工具

-类似字典，用kv对保存数据，存取方式跟字典也类似

-open，close





-shelve特性

-不支持多个应用并行写入

-为了解决这个问题，open的时候可以使用flag=r

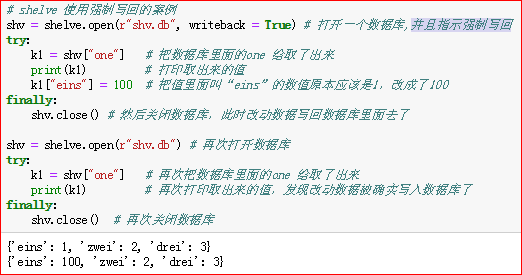
-写回问题

-shelve恶魔人情况下不会等待持久化对象进行任何修改

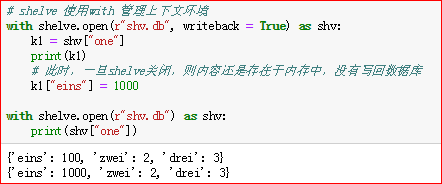
-解决办法：强制写回：writeback=True



上面的例子中的没有将改动数据写入数据库，这是一个常见的错误，该怎么办呢，应该使用“强制写回”的方式，见下方的例子：



shelve 使用with 管理上下文环境



4-46课 Log

-LOG

-https：//www.cnblogs.com/yyds/p/6901864.html

-logging

-logging 模块提供模块级别的函数记录日志

-包括四大组件

-1.日志相关概念

-日志

-日志的级别（level）

-不同的用户关注不同的程序信息

级别从低到高的排序

-DEBUG （调试）

-INFO （信息）

-NOTICE （通知）

-WARNING （警告）

-ERROR （错误）

-CRITICAL （紧急）

-ALERT （警示）

-EMERGENCY（紧急事件）

-IO操作 → 不要操作太频繁

-LOG的作用

-调试

-了解软件的运行情况

-分析定位问题

-日志信息

-时间

-地点

-level（级别）

-内容

-成熟的第三方日志

-log4j

-log4php

-logging

-2.logging模块

-日志级别

-级别可自定义

-初始化/写日志实例需要指定级别，只有当级别等于或高于指定某个级别的时候才被记录下来。

-使用方式

-直接使用logging（封装了其他组件）

-logging四大组件直接定制

2.1 logging模块级别的日志

-使用以下几个函数

logging .debug() 创建一条严重级别为DEBUG的日志记录

loggin.info() 创建一条严重级别为INFO的日志记录

logging.warning() 创建一条严重级别为WARNING的日志记录

logging.error() 创建一条严重级别为ERROR的日志记录

logging.critical() 创建一条严重级别为CRITICAL的日志记录

logging.log() 创建一条严重级别为LEVEL的日志记录

logging.basicConfig() 对root logger进行一次性配置

- logging.basicConfig() 对root logger进行一次性配置

-只在第一次调压的时候起作用

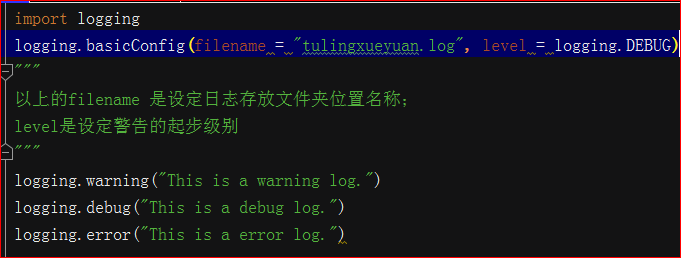
-不配置logger则使用默认值

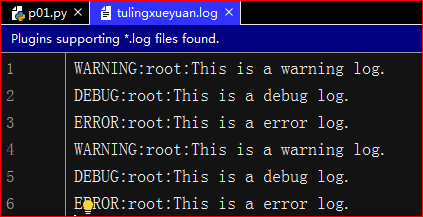
-输出：sys.stderr

-级别：WARNING（如果不配置级别，则最低显示warning级别）

-格式：level：log\_name:content

-案例01





另外一种输入format方式进行设定，如下输入：

LOG\_FORMAT = "%(asctime)s=========%(levelname)s+++++++%(message)s"

以上的asctime 表示时间；levelname表示警告级别；message表示警告内容。

-案例01

-format参数

有以下内容： asctime;created;relativeCreated;msecs;levelname;levelno;name;message;pathname;filename;module;lineno;funcName;process;processName;thread;threadName

-2.1 logging模块的处理流程

-四大组件

日志器(Logger): 产生日志的一个接口

处理器（Handler）：把产生的日志发送到相应的目的地

过滤器（Filter）：更精细的控制那些日志输出

格式器（Formatter）：对输出信息进行格式化

-logger

-产生一个日志

-操作

Logger.setlevel()

Logger.addHandler()和Logger.removeHandler()

Logger.addFilter()和Logger.removeFilter()

Logger.debug:

Logger.exception()

Logger.log()

-如何得到一个logger对象

-实例化

-logging.getLogger()

-Handler

-把log发送到指定位置

-方法

setlevel

setFormat

addFilter ; removeFilter

-不需要直接使用，Handler是基类

logging.StreamHandler

logging.FileHandler

logging.handlers.RotatingFileHandler

logging.handlers.TimeRotatingFileHandler

logging.handlers.HTTPHandler

logging.handlers.SMTPHandler

logging.NullHandler

-Format类

-直接实例化

-可以继承Format添加特殊内容

-三个参数

-fmt：指定消息格式化字符串，如果不指定该参数则默认使用message原始值

-datefmt：指定日期格式字符串，如果不指定该参数则默认使用“%Y-%m-%d %H:%M:%S”

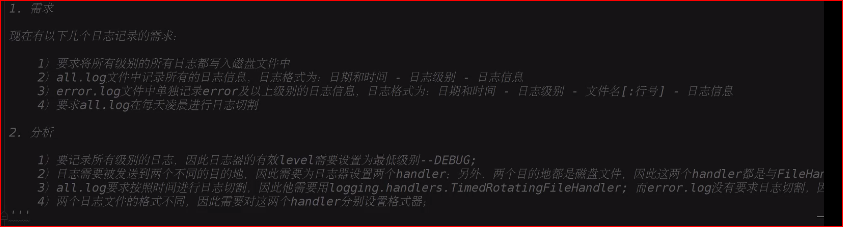
-style:Python 3.2新增参数，可取值为“%”，“{”和“%”，如果不指定该参数则默认使用“%”

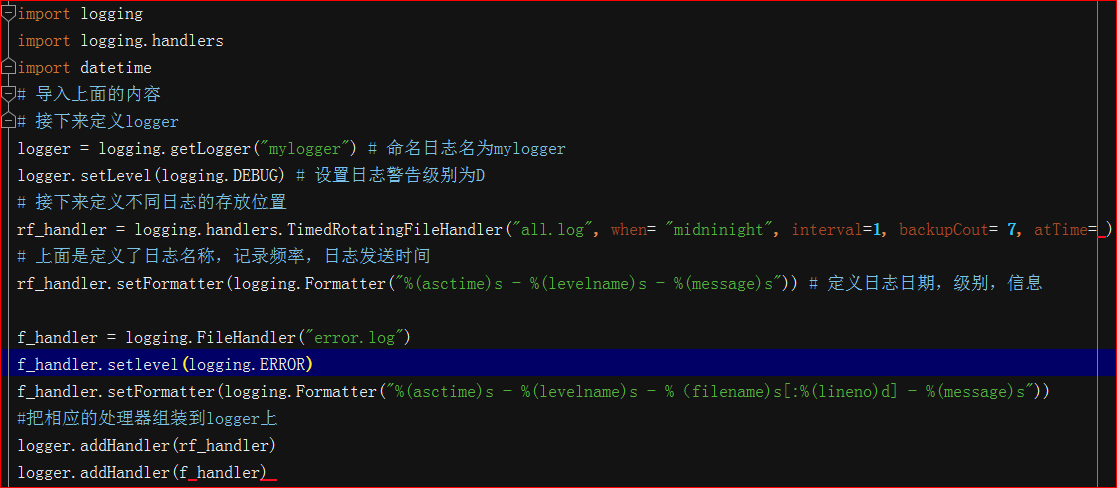
-filter类

-可以被Handler和Logger使用

-控制传递过来的信息的具体内容

-案例





接下来会省略 多线程，协程，格式化文件，正则和XPath,FTP,mail编程，http协议内容，以及GUI-Tkinter教程，

后面直接进入python爬虫内容。