Python期末复习

只有字典和集合是无序的，因为字典和集合没法索引

元组&number&string是不可变的

集合中的元素不允许有重复数据

元组可以用来做字典的key

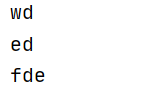
元组和list都可以 + 可以 \*

列表：

str = "lawedf"

print(str[-3:-1])

print(str[-1:-4:-1])



列表是可以有重复元素的

list.extend(seq)是在列表后追加一个列表中的元素

list.sort(key=None,reverse=False)从小到大

reverse() 函数用于反向列表中元素，没有返回值，但是会把列表反向

List.append()是用来增加元素的

字典：

dict = {'one':"this is one",2:"this is 2"}

print(dict['one'])

print(dict.keys())



dict1 = dict.fromkeys(list,10) //用列表去生成字典

print(dict1)

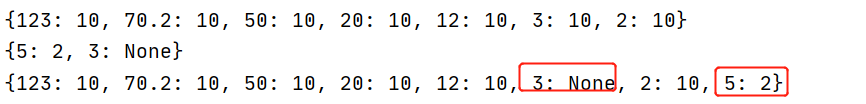
dict = {5:2}

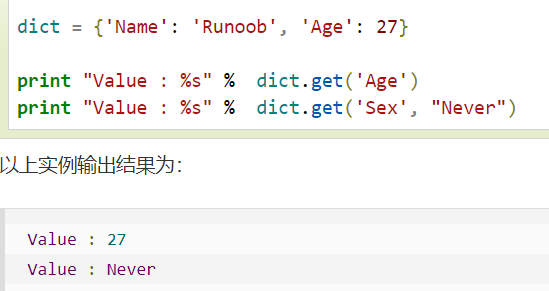
dict.setdefault(3,None)

print(dict)

dict1.update(dict) //这里是用dict去更新dict1,其实就是合并字典

print(dict1)

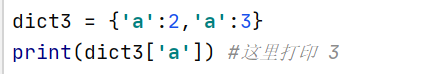


这里get若是没这个key是不会往dict里加没有的那个元素的。

Del是彻底抹掉，clear是还有字典，只不过里面空了

Dict.pop(key)是删除指定的元素，key不能不给出

Dict.popitem()才是删除字典中的最后一个，然而列表中则可以用无参的pop

字典允许重复元素，但会覆盖

列表与字典的对比：

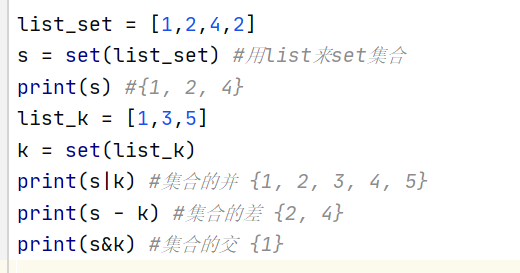
1. 字典的查找和搜索速度不会随key的增加而变化
2. 字典查找和搜索速度很快
3. 字典占用内存多

元组

1. ）这样也算元组，但是必须有逗号，元组是不可变的

集合

无序的，不可重复的



集合用add的方法

可以使用大括号 { } 或者 set()

函数创建集合，注意：创建一个空集合必须用

set() 而不是 { }，因为 { }

是用来创建一个空字典

基础语法：

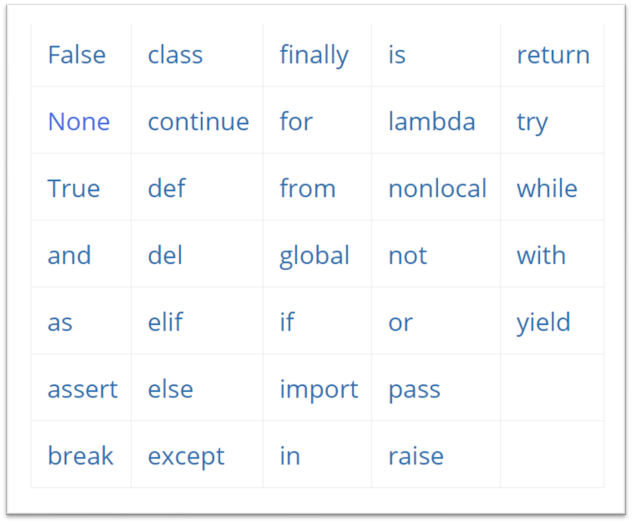
print()函数也可以接受多个字符串，用逗号“,”隔开，就可以连

成一串输出。

print()会依次打印每个字符串，遇到逗号“,”会输出一个空格



下面是关键字，现在还有Async 和await

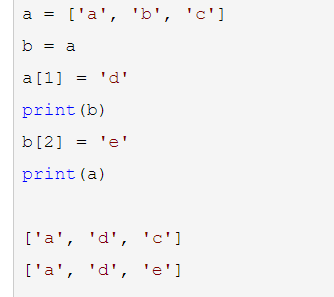
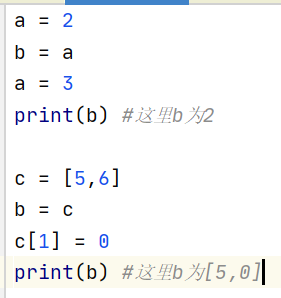


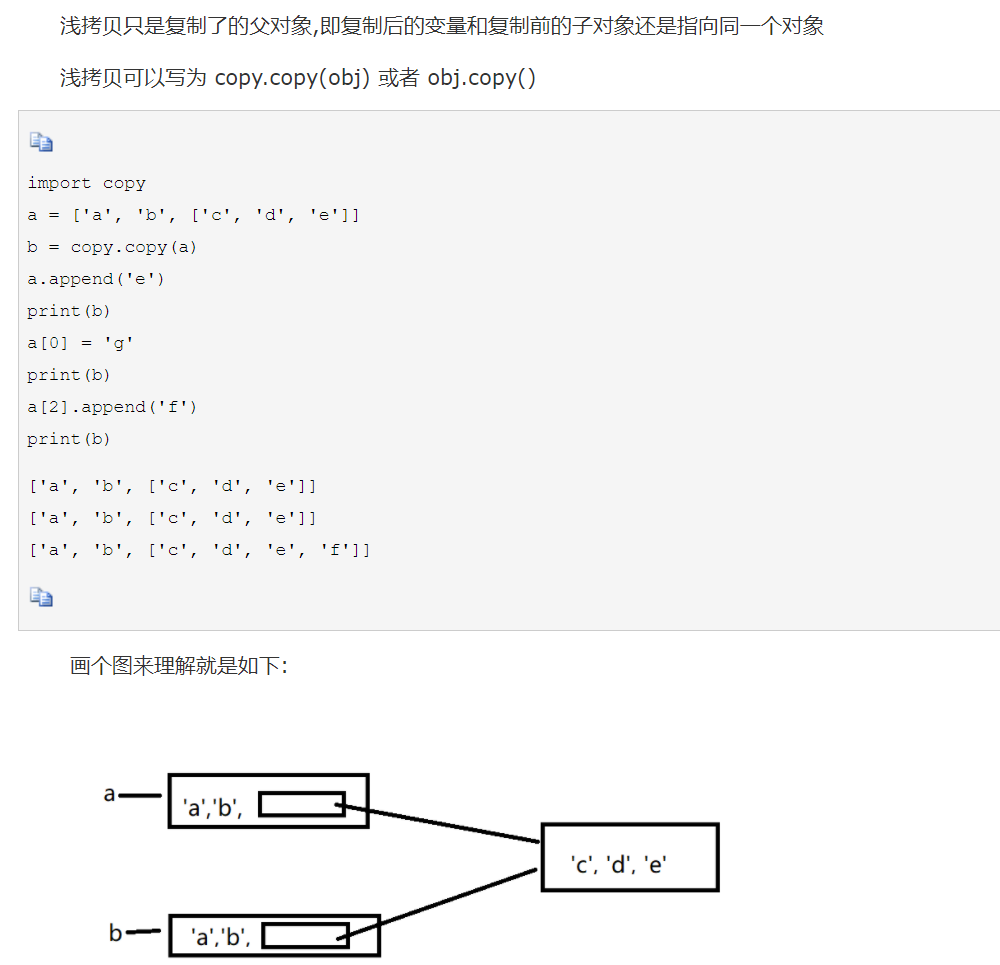
对象：python中一切皆对象，每个对象由:标识( identity )、类型(type )、 value (值组成）。

对象的本质就是:一个内存块,拥有特定的值,支持特定类型的相关操作。

引用：

要画图，用a和b去指那个内存块，列表不存具体的值在内存中，只存ID，a,b的ID一样



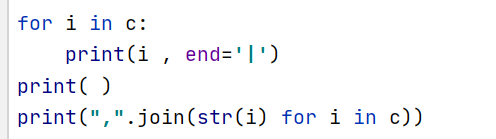


标识符：第一个字母必须是字母，下划线，其后是字母，数字，下划线

选择与循环：

If elif else

while语句后面可以加else



结果是：

5|0|

5,0

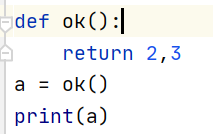
函数

内置函数：max，min，abs(绝对值)

函数执行完毕也没有return语句时，自动return None

函数可以同时返回多个值，但其实就是一个tuple。

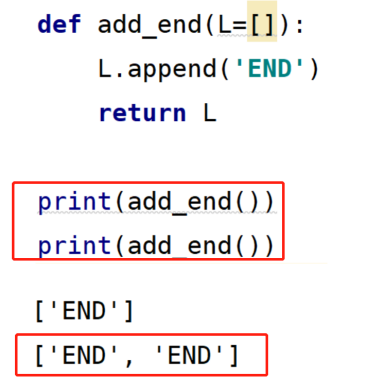
结论：函数有且只有一个返回值

结果是

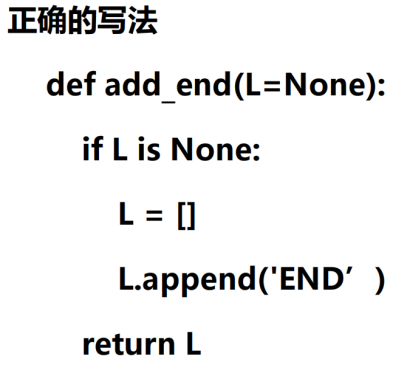
下面是默认参数的函数的错误用法：

因为L是可变参数，不是像number和string一样的，L在中间被改变了，则下次调用时，默认参数的内容就变了，不再是函数定义时的[]了。

所以，定义默认参数要牢记一点：默认参数必须指向不变对象！



下面是正确用法：因此可以用不变的None来代替[]



函数定义时：

1、\*args表示可接受任意个（包含0个）位置参数，当函数调用时，所有未使用（未匹配）的位置参数会在函数内自动组装进一个tuple对象中，此tuple对象会赋值给变量名args

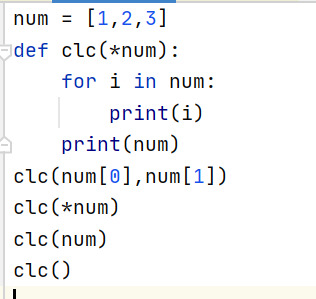
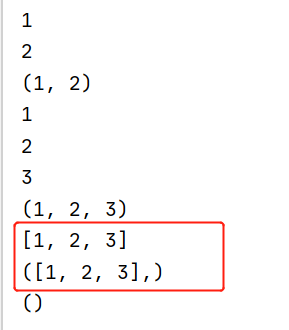
2、\*\*kwargs表示可接受任意个（包含0个）关键字参数，当函数调用时，所有未使用（未匹配）的关键字参数会在函数内组装进一个dict对象中，此dict对象会赋值给变量名kwargs

注意：函数定义时，二者同时存在，一定需要将\*args放在\*\*kwargs之前

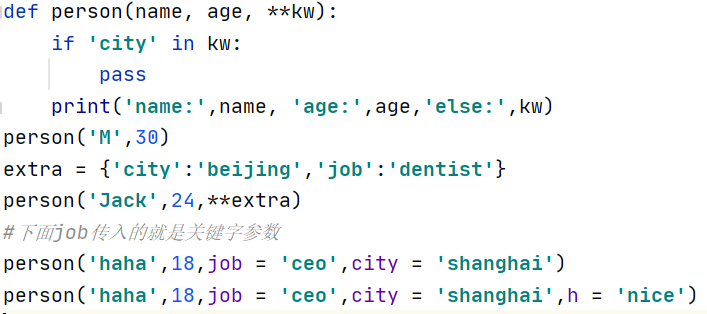
函数调用时：

1、\*args表示解包元组对象中的每个元素作为函数调用时传入的位置参数

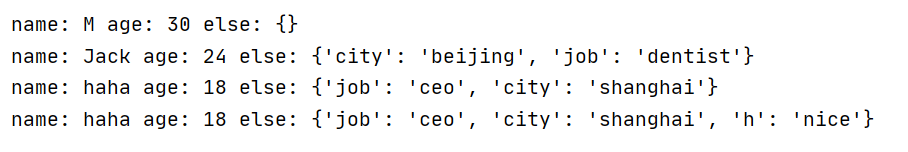
2、\*\*kwargs表示解包字典对象中的每个元素作为函数调用时传入的关键字参数

结果：

下面是\*\*的：

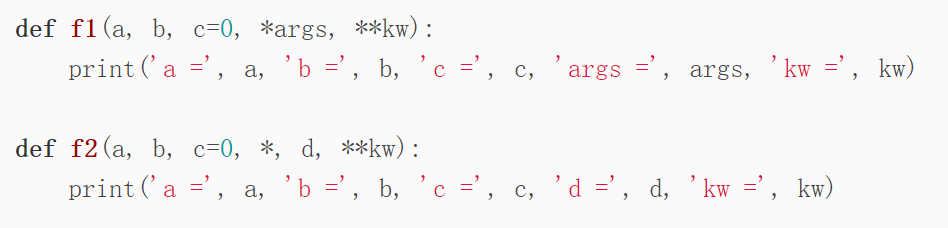


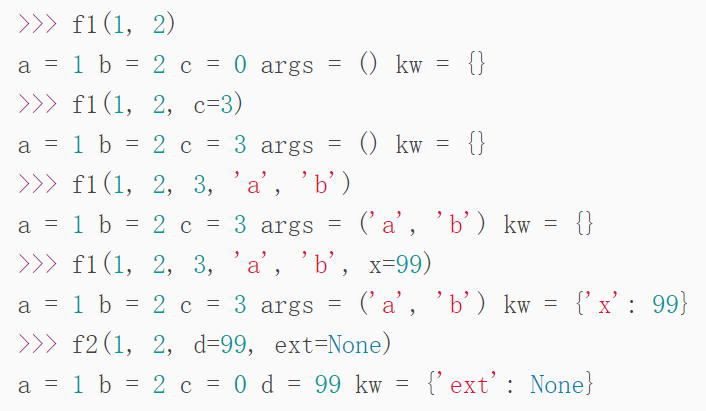
结果：

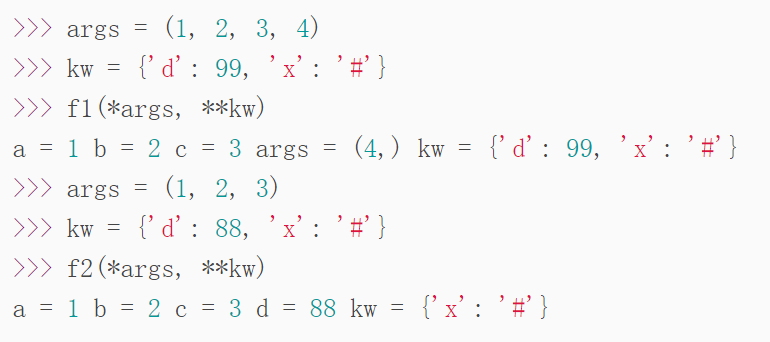


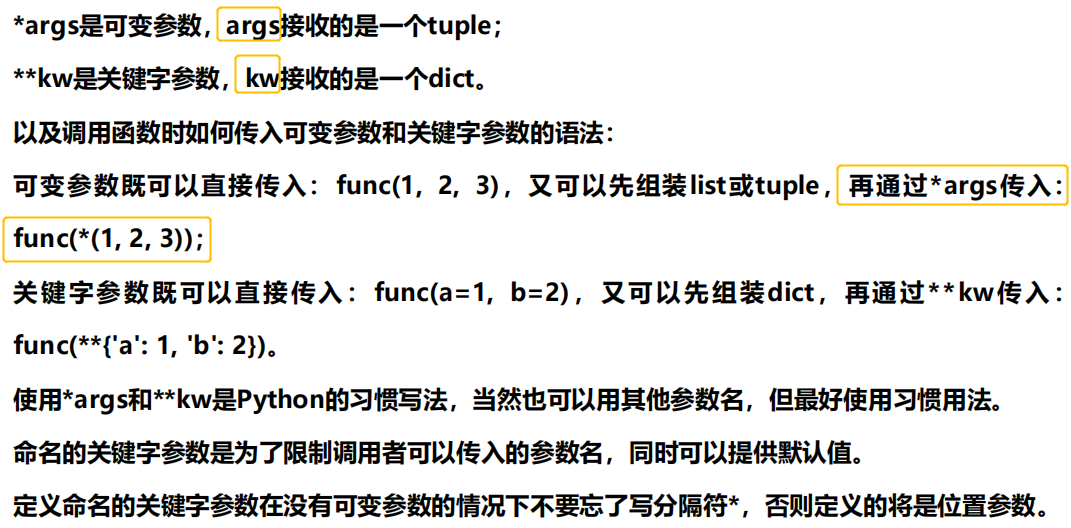
命名关键字：







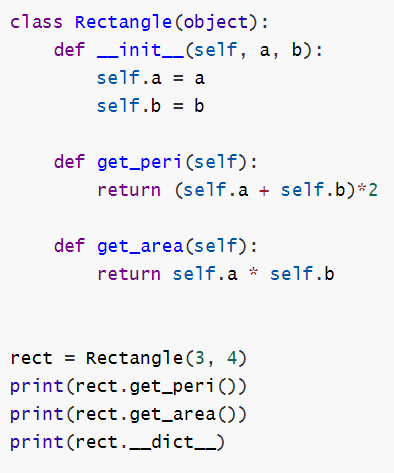
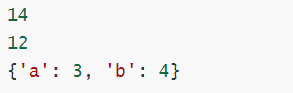


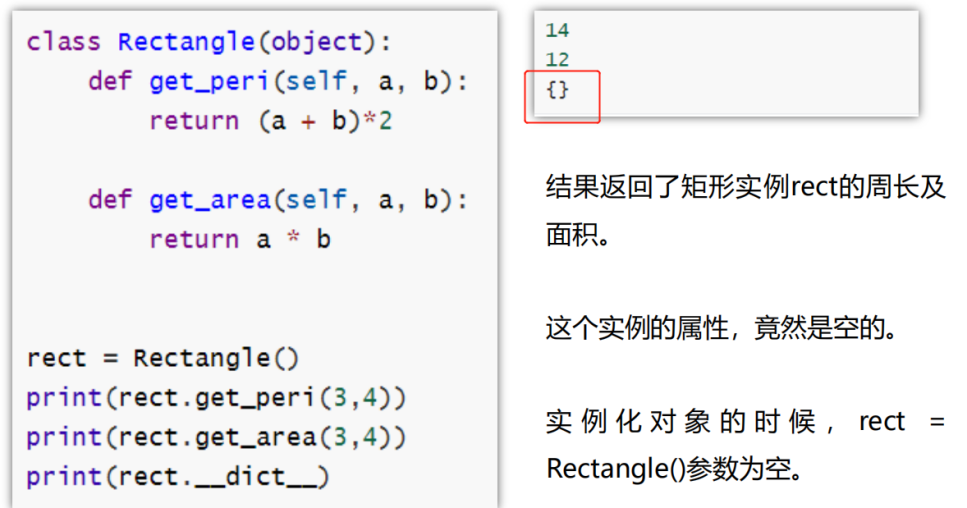


字典里还可以调用函数：



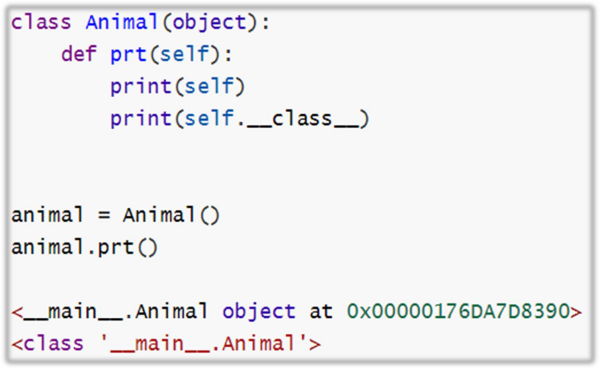
面向对象

最后一行的\_\_dict\_\_结果是

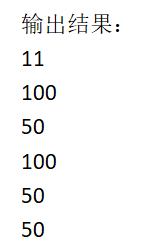
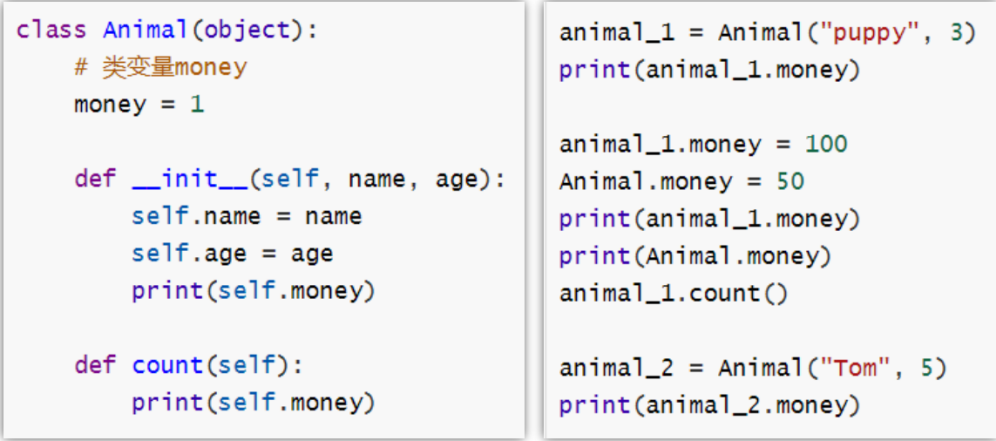


实例调用\_\_class\_\_属性时会指向该实例对应的类，然后可以再去调用其它类属性，毕竟类属性还是由类调用会比较好。

example：self.\_\_classs\_\_.\_\_name\_\_ ，首先用\_\_class\_\_将self实例变量指向类，然后再去调用\_\_name\_\_类属性，通常情况\_\_name\_\_的值为‘\_\_main\_\_’。



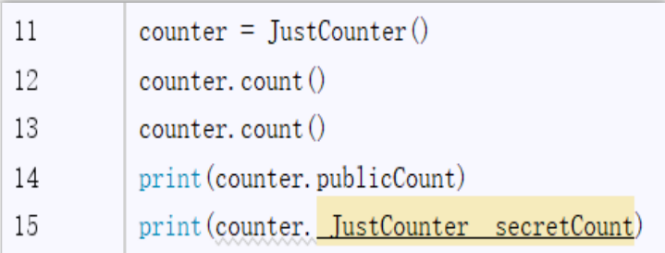
创建实例的时候，如果实例中没有初始化实例的变量，用类变量的值赋值给实例变量，从此实例变量与类变量无瓜葛。



面向对象三大特征：封装，继承，多态

类的私有属性：\_\_private\_attrs：两个下划线开头

假定没有恶意行为，python留出了类外访问私有属性和方法的方式：即\_类名\_\_属性名 或者 \_类名\_\_函数名 来直接访问类中的私有属性和私有方法



类方法：

第一个形参是类对象的方法  
需要用装饰器@classmethod来标识其为类方法，对于类方法，第一个参数必须是类对象，一般以cls作为第一个参数。

当方法中 需要使用类对象 (如访问私有类属性等)时，定义类方法  
类方法一般和类属性配合使用

静态方法：

需要通过装饰器@staticmethod来进行修饰，静态方法既不需要传递类对象也不需要传递实例对象。没有self参数

静态方法 也能够通过 实例对象 和 类对象 去访问。

Python允许多继承

实现多继承跟实现单继承基本一致，在括号中依次写上继承

父类名称即可

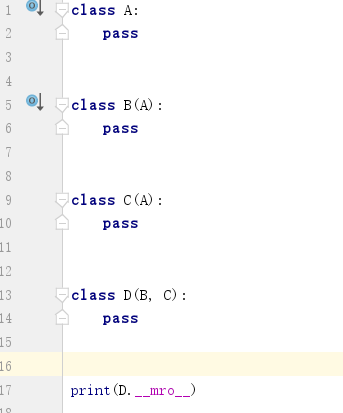
注意括号中父类顺序，当子类未指定方法名时，python从

左向右查找父类中是否含有指定方法

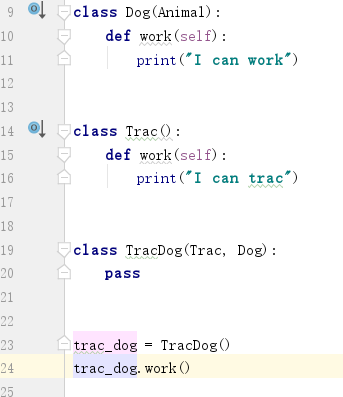
python多重继承的MRO算法选择： 经典方式、Python2.2

新式算法、Python2.3 新式算法(C3)。Python 3中只保留

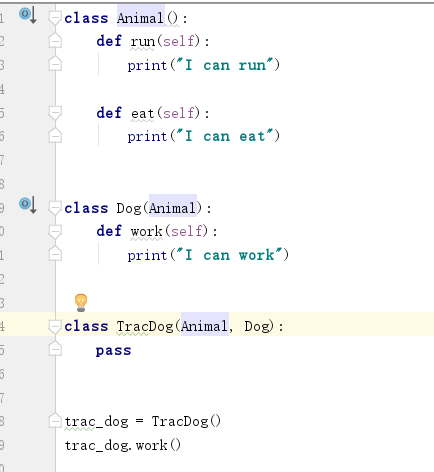
了最后一种，即C3算法

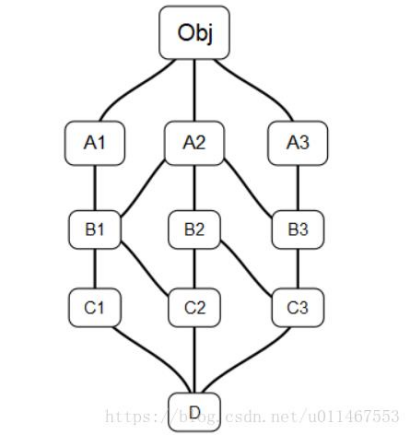
从子到父，从左到右，不要忘了object



输出i can trac，继承顺序很重要

下面这样会出错：Animal更高级应该是TracDoog（Dog,Animal）





上面这个图应该是从左到右看，一旦左边有没有子节点的就继续剪掉这个节点。Obj节点永远是最后一个被遍历的：D，C1，C2，B1，A1，C3，B2，B3，A2，A3

注意：如果不是D的父节点的则应该自动把这个节点等删掉

组合就是没有继承但是用了人家的实例的：



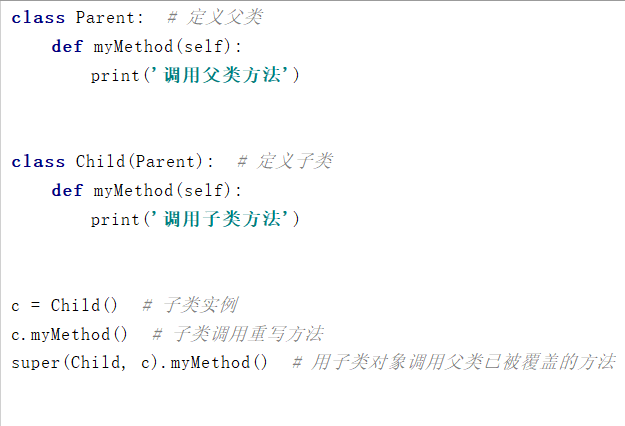
多态：



由 于 Python 是 动 态 语 言 ， 所 以 ， 传 递 给 函 数work\_with\_dog的参数dog不一定是Dog或Dog的子类型。任何数据类型的实例都可以，只要它有一个work()

super() 函数是用于调用父类(超类)的一个方法

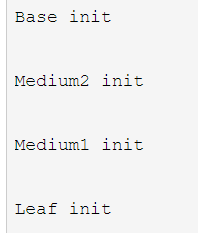
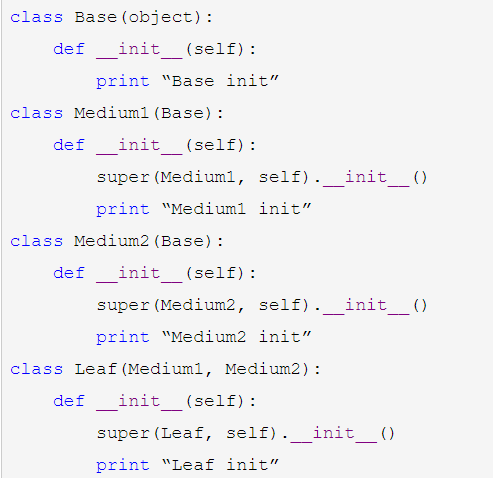
Python3.x 和 Python2.x 的一个区别是: Python 3 可以使用直接使用 super().xxx 代替 super(Class, self).xxx



钻石继承：如果只用父类的名字调用\_\_init\_\_方法，则会出现多次初始化

因此，用super按照mro顺序查找，一旦找不到就往后找，super().\_\_init\_\_() 等价于super(Medium1, self).\_\_init。https://blog.csdn.net/yangjiajia123456/article/details/80382583

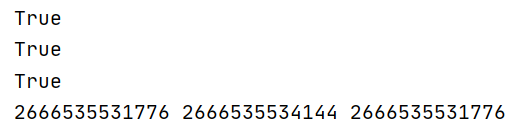
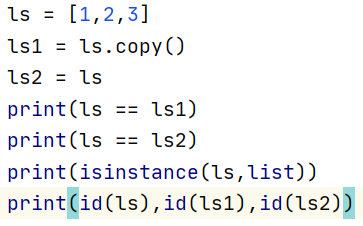




https://www.cnblogs.com/czaiz/p/7772194.html

对象三要素：value（==），type（type&isinstance(x,class)），id（is）

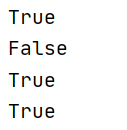
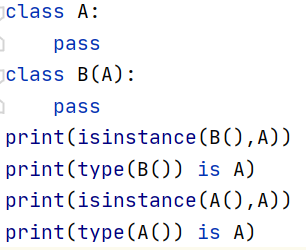
无论是copy还是=得到的value都一样，但是copy后的id会变



type() 和 isinstance() 函数的区别:

isinstance(object, classinfo)：判断参数 object 是否是 classinfo 类或者其子类的实例

type()：只能判断是否是该类本身的实例

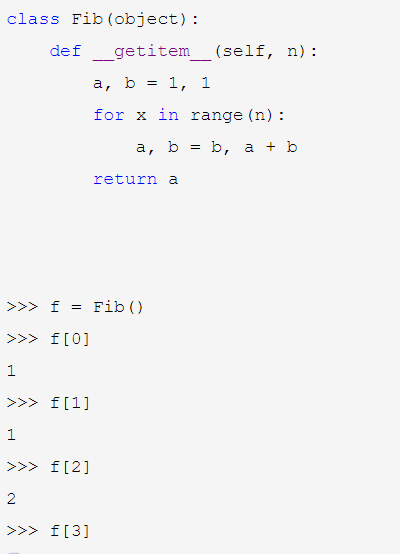


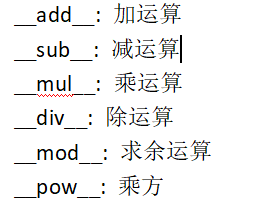
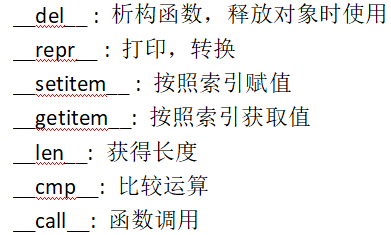
魔法函数：

实现 def \_\_iter\_\_(self):方法才有迭代器

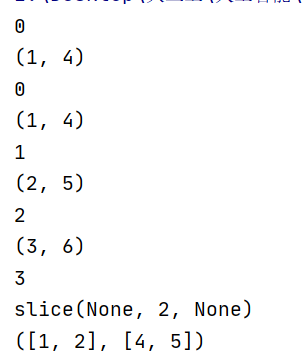
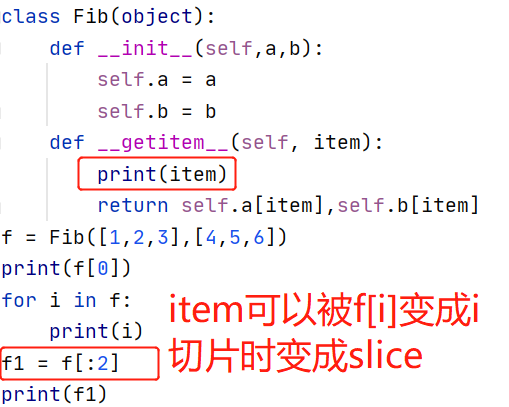
python语法会做一些优化，如果拿不到迭代器（即没有实现 \_\_iter\_\_ 方 法 ） ， python 解 释 器 会 去 寻 找\_\_getitem\_\_()方法，直到整个对象取完。

如果一个类想被用于for ... in循环，类似list或tuple那样，就必须实现一个\_\_iter\_\_()方法，该方法返回一个迭代对象（self），然后，Python的for循环就会不断调用该迭代对象的next()方法拿到循环的下一个值，直到遇到StopIteration错误时退出循环。





因为print函数会首先调用str()函数将参数转化为str类型，而str()函数会调用这个参数所属类中的\_\_str\_\_方法(如果类中有此方法的话)，我们可以重写这个\_\_str\_\_方法。

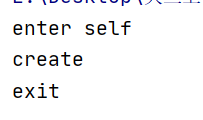
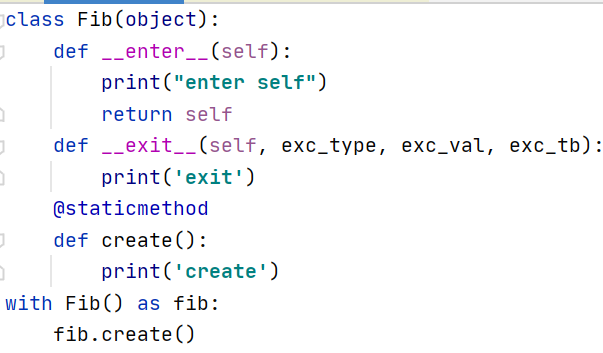


没有\_\_str\_\_方法被实现时，系统会找到\_\_repr\_\_()方法并且用这个序列来完成str方法的模拟

with如何工作？（文件处理的时候，上下文管理，不用手动关闭文件）

这看起来充满魔法，但不仅仅是魔法，Python对with的处理还很聪明。基本思想是with所求值的对象必须有一个\_\_enter\_\_()方法，一个\_\_exit\_\_()方法。

紧跟with后面的语句被求值后，返回对象的\_\_enter\_\_()方法被调用，这个方法的返回值将被赋值给as后面的变量。当with后面的代码块全部被执行完之后，将调用前面返回对象的\_\_exit\_\_()方法。



Python的类属性不是在\_\_init\_\_里的：

@property使用属性函数的最简单的方法之一，是将它作为一个方法的装饰器来使用。这可以让你将一个类方法转变成一个类属性。

Property的setter额getter方法：

下面的两个方法有一点区别：



结果都是：

Song

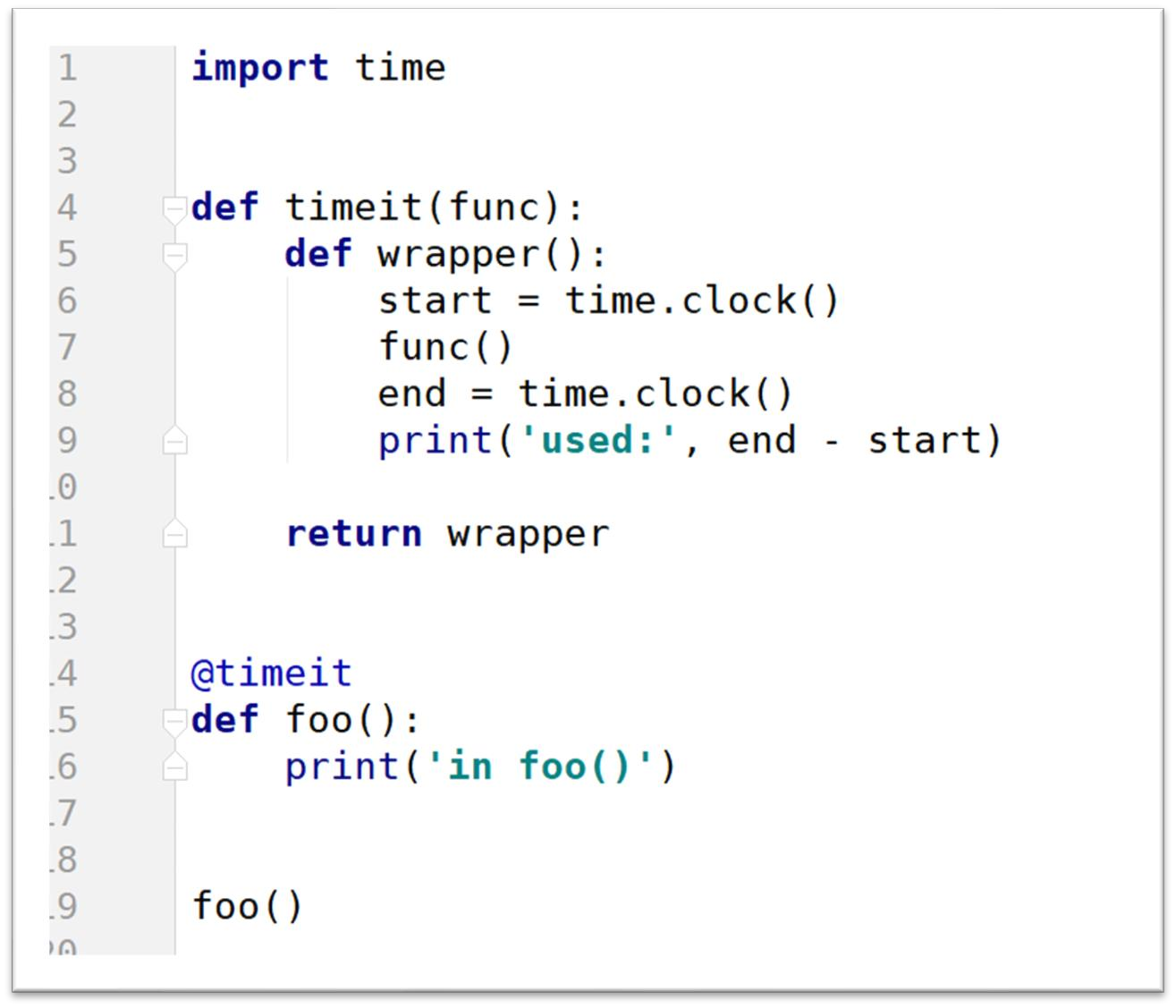
100

**类的方法**分为实例（对象）方法，静态方法，类方法三大类

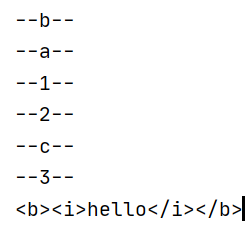
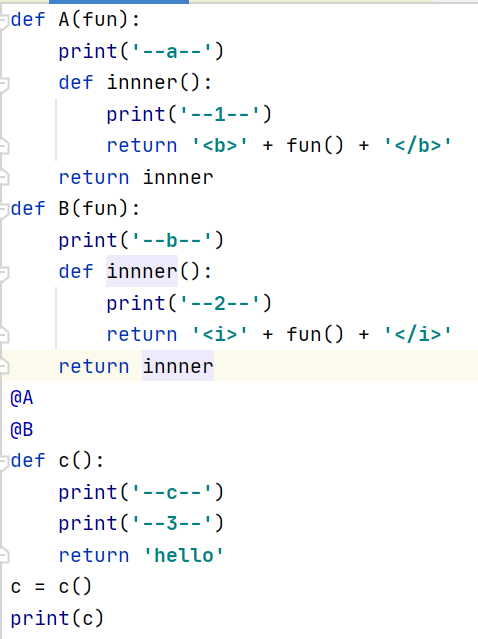
Global就是设置全局变量，值会一直被改变，并被记住

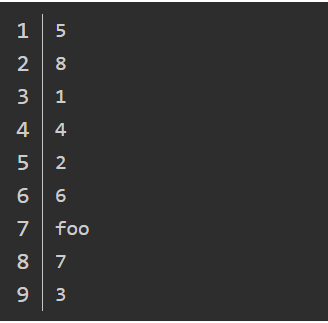
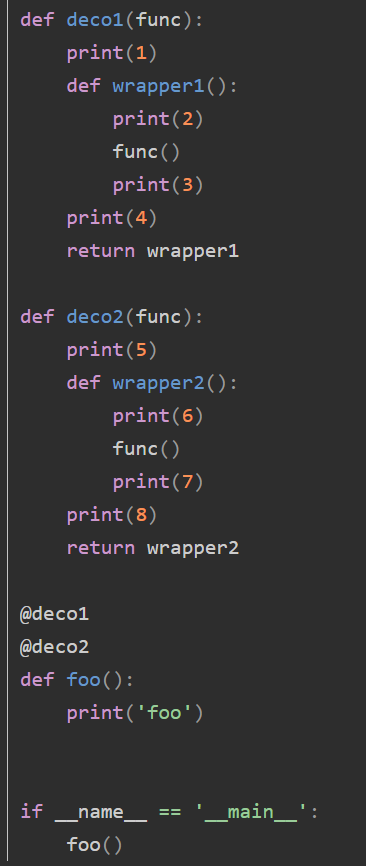
装饰器

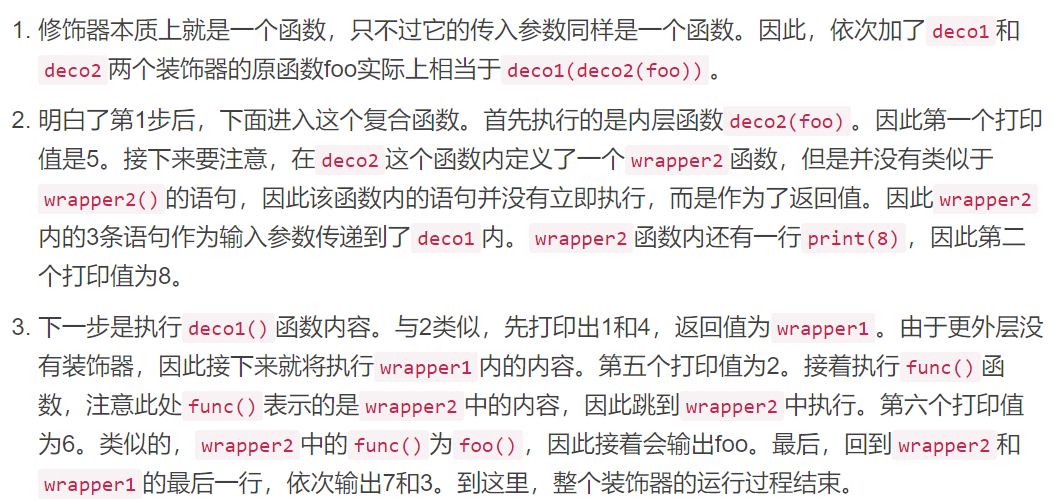
就是给函数return一个封装好了的功能，函数接受返回值



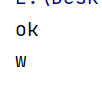
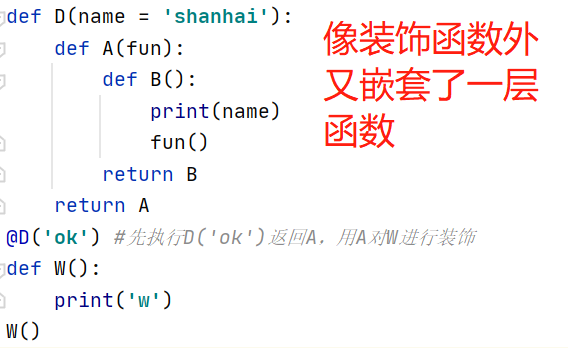
这样，一个简易的**计时器**就做好了！我们只需要在定义foo以后调用foo之前，加上foo = timeit(foo)，就可以达到计时的目的，这也就是装饰器的概念，看起来像是foo被timeit装饰了。在这个例子中，函数进入和退出时需要计时，这被称为一个横切面(Aspect)。



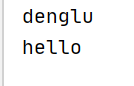
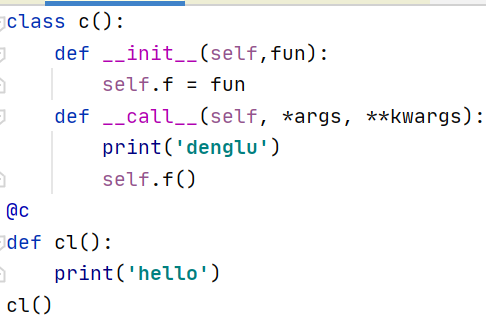




装饰器可以实现多种功能：

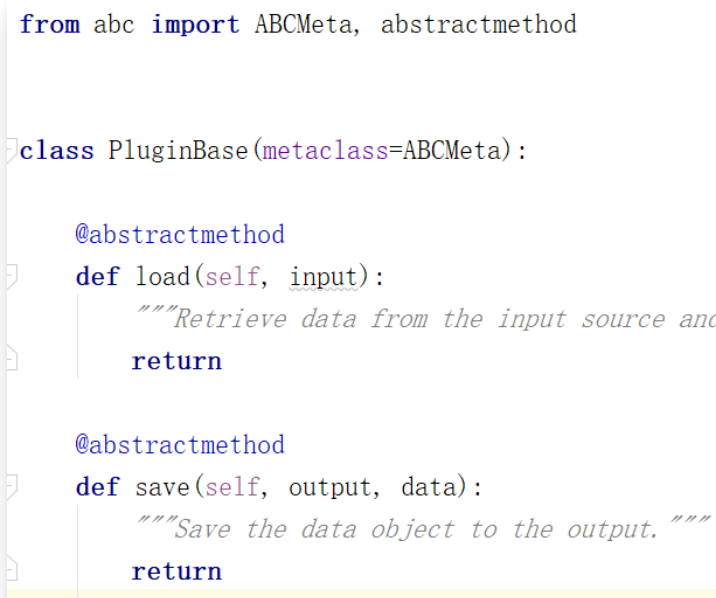


类装饰器的作用：



抽象类：可以用abs变成抽象方法。不能被实例化的类

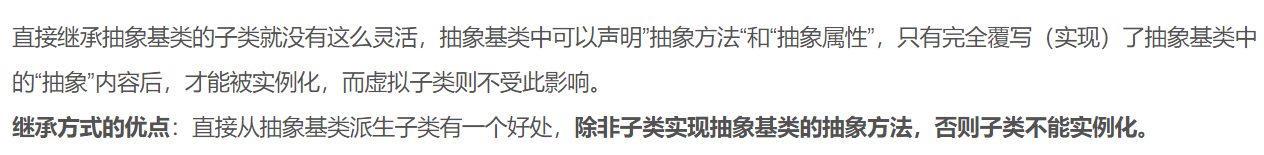
定义抽象类：



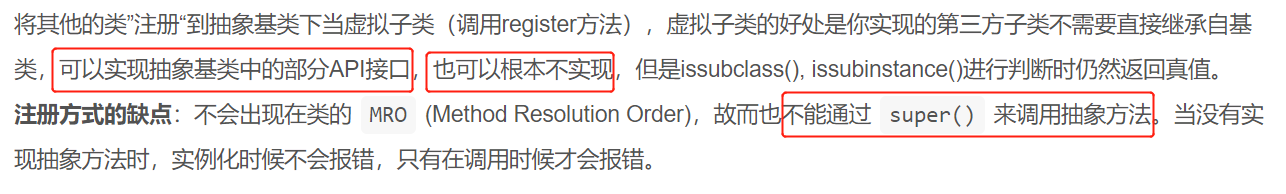
isinstance(obj,cls)检查是否obj是否是类 cls 的对象

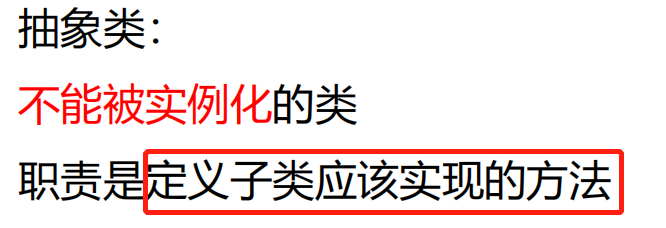
issubclass(sub, super)检查sub类是否是 super 类的派生类

直接继承：

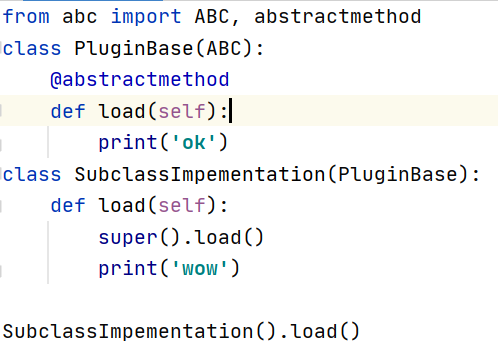


虚拟子类：





但是抽象类中可以包含具体的方法，在子类中可以用super来调用。

输出：ok 和wow



异常

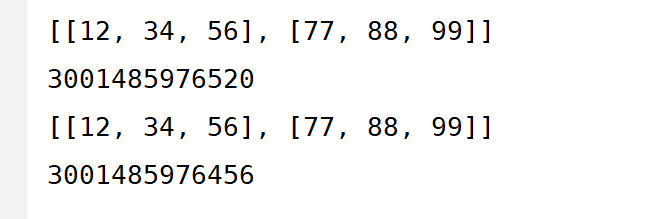
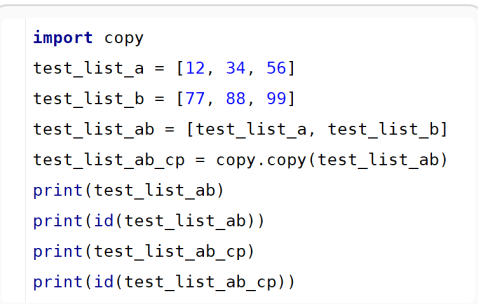
Try excpt else

final是无论异常与否都要执行

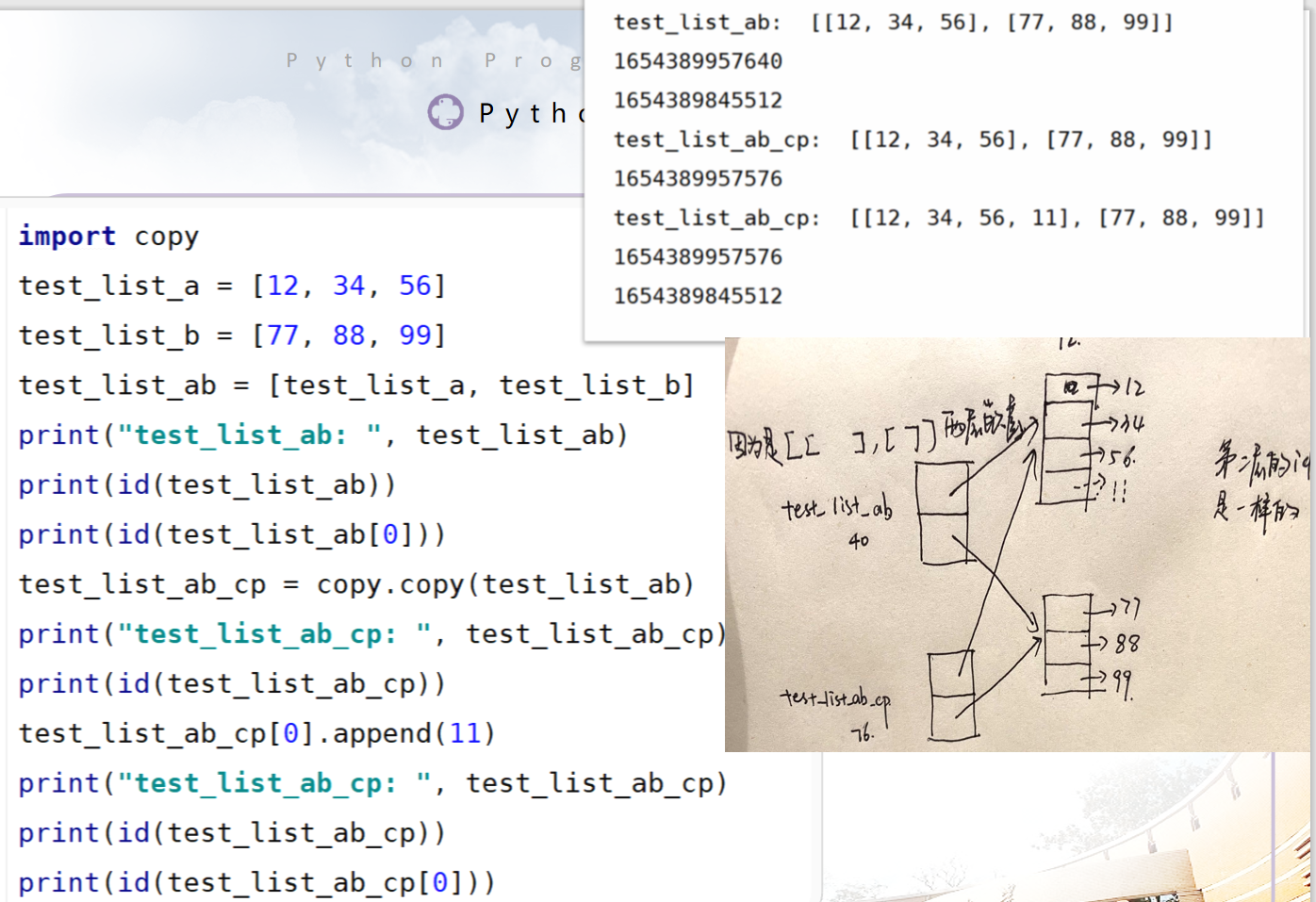
深拷贝:copy.deepcopy()

浅拷贝：copy() 不仅与 [:] 还与copy.copy()功能相同

直接等于号的话，id是一样的

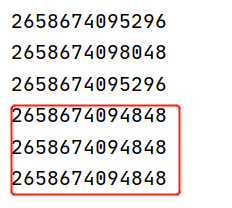
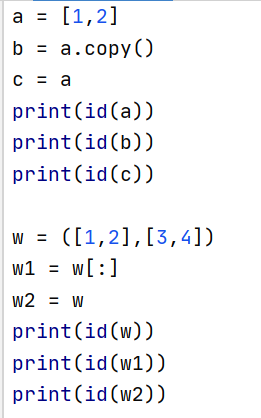


下面是浅拷贝，第二层嵌套的id还是一样的，但是深拷贝，第二层id都不一样

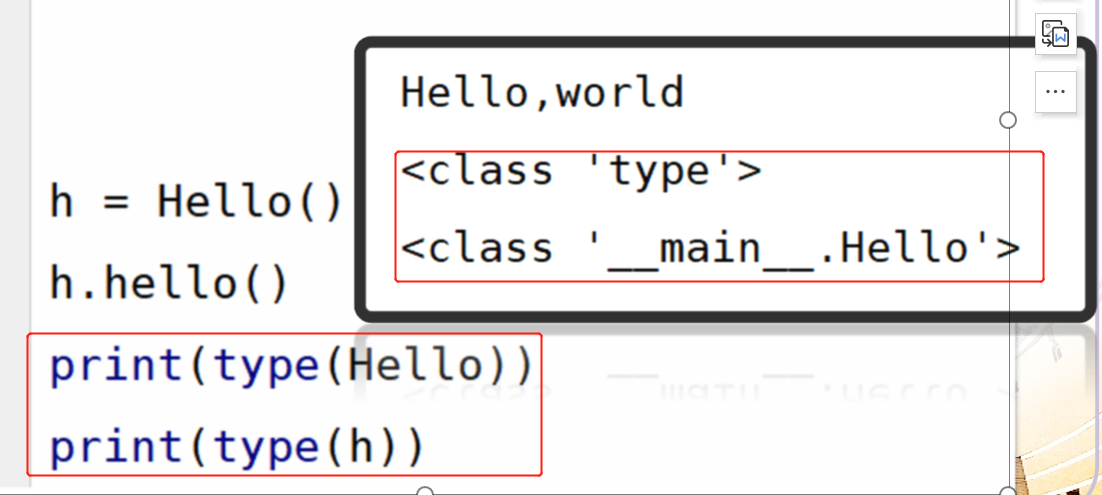


总结：当最外层对象为可变类型时，copy后得到的对象指向新的内存空间，当最外层的对象为不可变类型时，copy后得到的对象指向原对象的内存空间（注意：浅拷贝的对象的最外层是否是可变类型），但是不是copy是直接等于的话，那么都是一样的id

拷贝的内容中只要有一个是个可变类型，那么deepcopy一定是深拷贝



动态创建类：

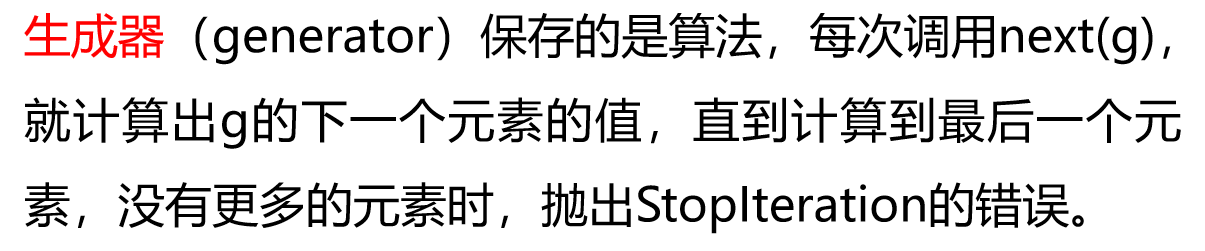


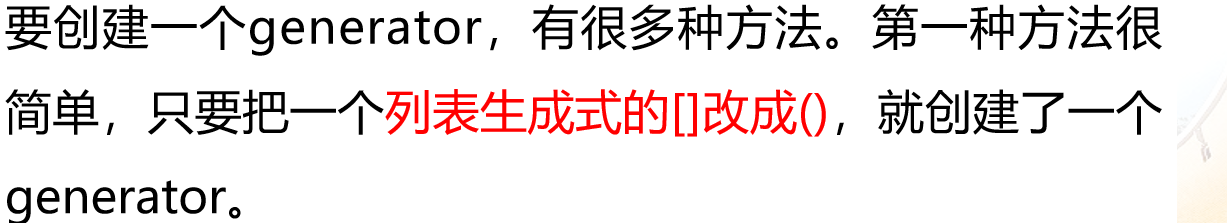


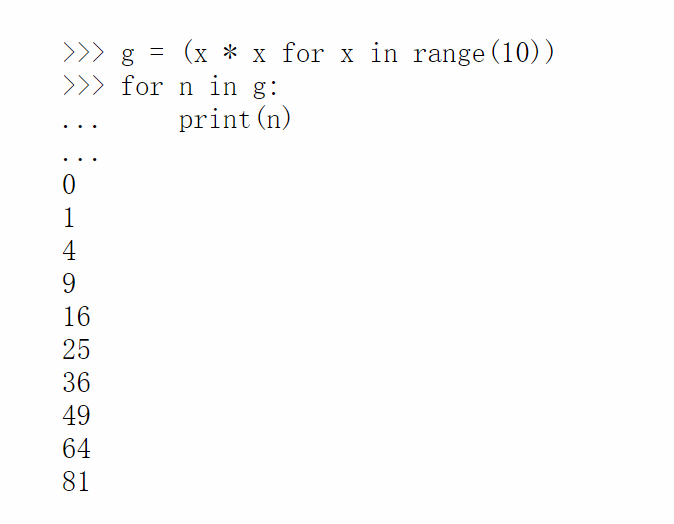
\_\_bases\_\_是查看直接父类，不是爷爷啥的

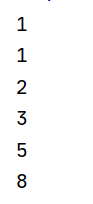
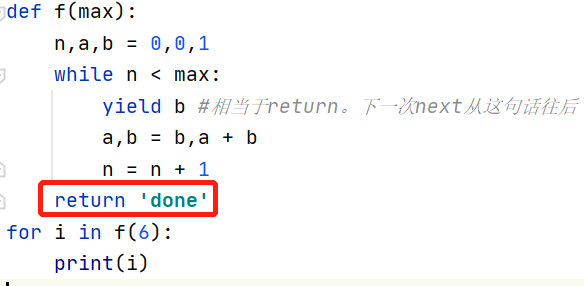


生成器:









上面的图片得不到‘done’是因为，应该捕捉错误，返回值包含在StopIteration的value中。

简要理解：yield就是 return 返回一个值，并且记住这个返回的位置，下次迭代就从这个位置后(下一行)开始。

带有yield的函数不仅仅只用于for循环中，而且可用于某个函数的参数，只要这个函数的参数允许迭代参数。比如array.extend函数，它的原型是array.extend(iterable)。

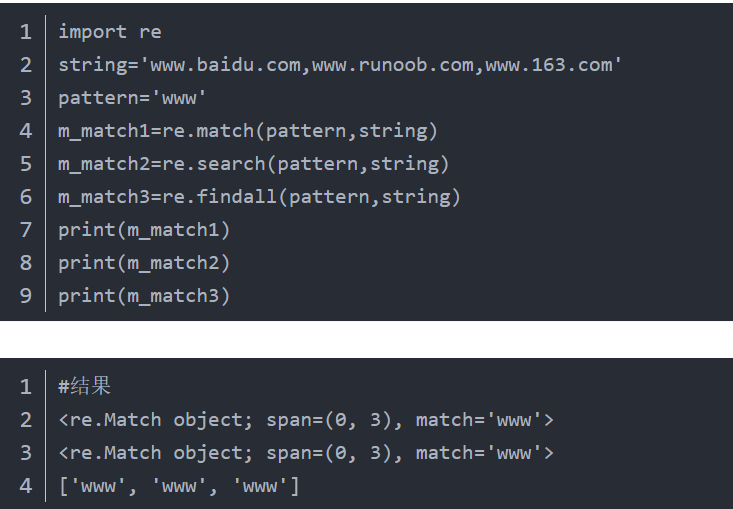
正则表达式：

Match和search和findall

\*\*re.match匹配方式：\*\*从字符起始位置匹配，若起始位置匹配不成功返回none。即只匹配起始位置.match &.serch

re.match 只能在起始位置匹配，而re.search可以扫描整个字符串并返回第一个成功的匹配

上述两个都只能找到一个匹配结果，而findall可以找到所有满足匹配条件的结果，并以列表的形式返回



compile()

可以返回一个对象，然后用exec来调用这个字节的对象

re.compile(pattern,flags=0)

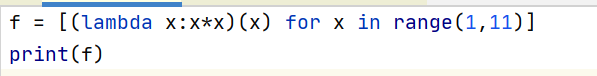


在Python中，通常有这三种方式来表示时间：时间戳、元组(struct\_time)、格式化的时间字符串

三器一魔：

lambda表达式返回一个函数对象

例子：func = lambda x,y:x+y



[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

老师的知识点的补充:

荷兰人多·范罗苏姆(Guido van Rossum)

强类型

swapcase() 将字符串中大写转换为小写，小写转换为大写

rstrip() 删除字符串字符串末尾的空格

list.extend(seq)在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）

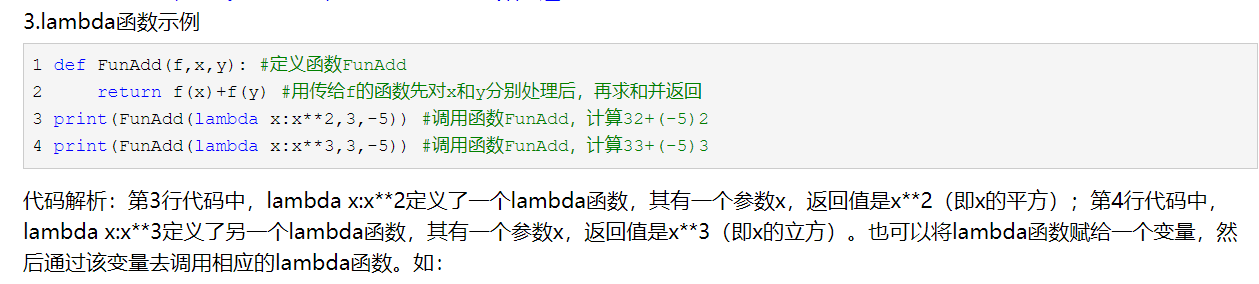
但是，如果我们想要限制class的属性怎么办？比如，只允许对Student实例添加name和age属性。

为了达到限制的目的，Python允许在定义class的时候，定义一个特殊的\_\_slots\_\_变量，来限制该class能添加的属性

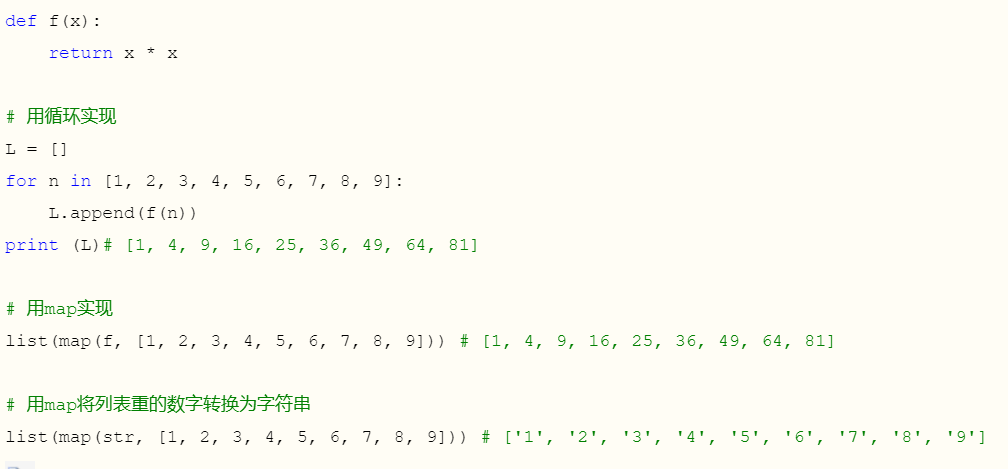
使用\_\_slots\_\_的类的实例不在使用字典来存储数据。相反，会使用基于数组的更加紧凑的数据结构。

在会创建大量对象的程序中，使用\_\_slots\_\_可以显著减少内存占用和使用时间

\_\_new\_\_() 是一种负责创建类实例的静态方法，它无需使用 staticmethod 装饰器修饰，且该方法会优先 \_\_init\_\_() 初始化方法被调用。



map()函数接收两个参数，一个是函数，一个是序列，map将传入的函数依次作用到序列的每个元素，并把结果作为新的对象返回，返回值是一个可迭代对象，可以用list()方法将其转为一个列表。



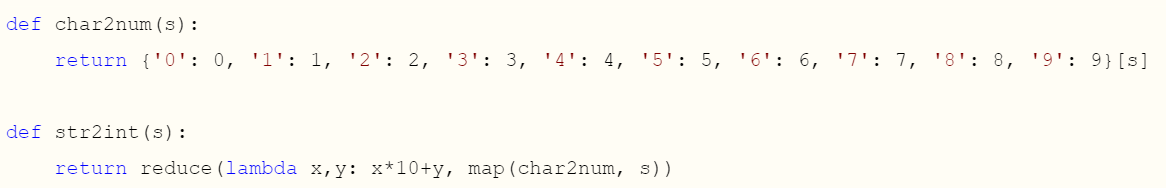


Reduce也可能有第三个参数;

然后介绍了第三个参数，有两个作用：  
一是如果初始值存在，那它应该放置在序列的前面，然后参与运算；也就是说第三个参数将做为运算时的第一个参数传入进行计算；二是当结果为空时作为默认值。

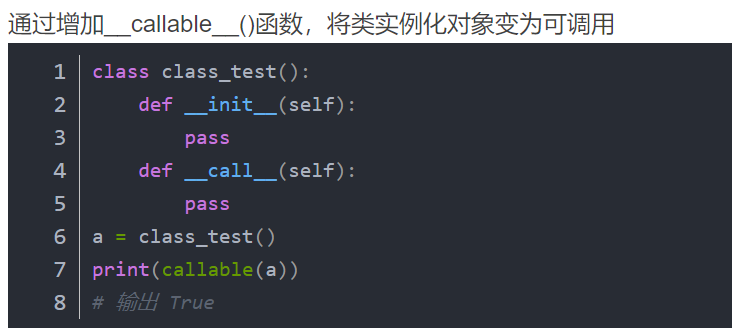
结果是13579

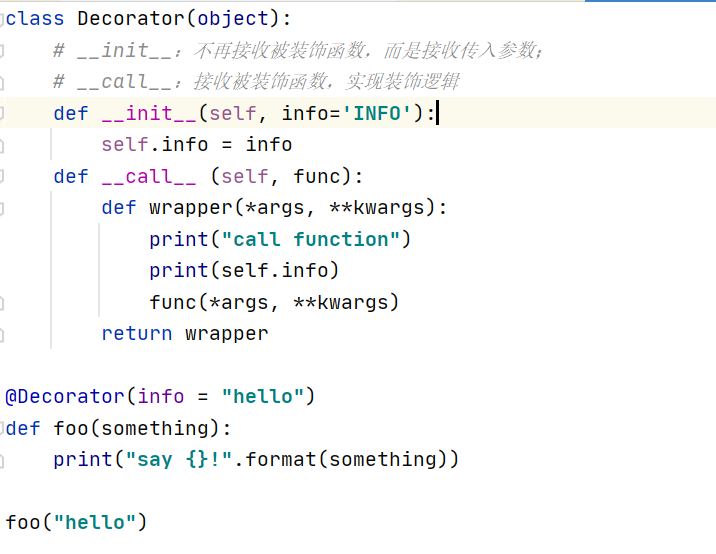
下面用lambda来简化计算：

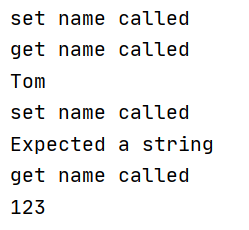
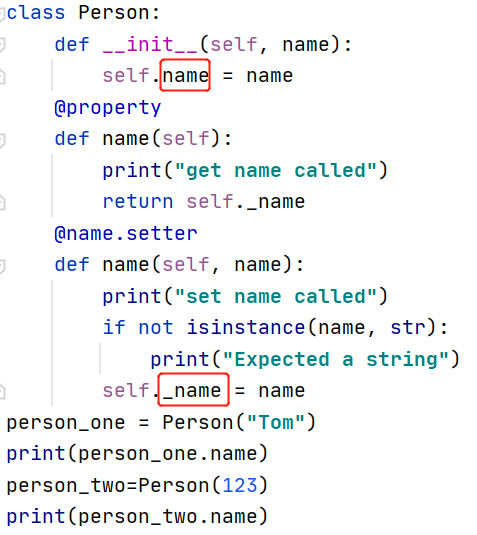


Sorted是对于数字来说是从小到大，字母来说从大到小：

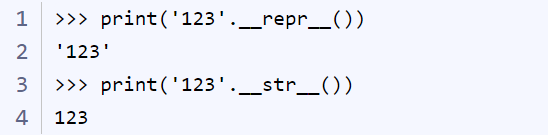








第一个name是函数名



字面意思，Iterable表示可迭代，Iterator表示迭代器

可迭代就是可以使用for循环，而且显然，迭代器肯定是可迭代的，

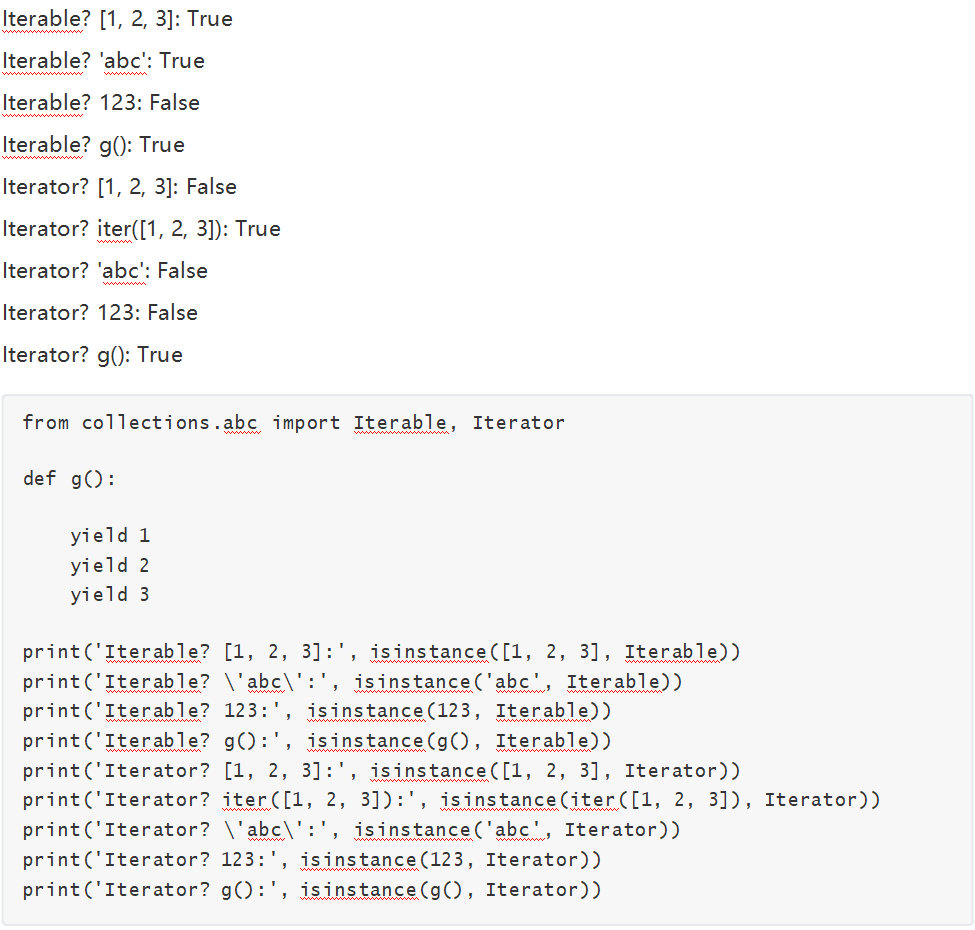
但可迭代的就不一定是迭代器

这里的Iterator（迭代器）其实就是指 generator（生成器）

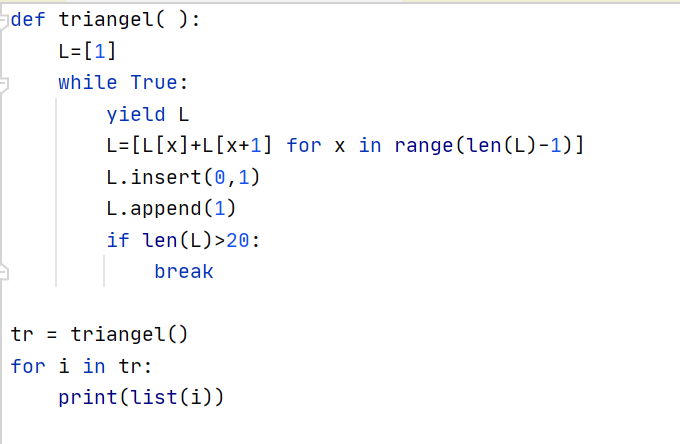
如list，tuple，dict，set都是可迭代对象，字符串也是可迭代对象，但是整数就不是可迭代对象

生成器都是Iterator对象，但list、dict、str虽然是Iterable，却不是Iterator。

把list、dict、str等Iterable变成Iterator可以使用iter()函数







bool(**"c"**)==True