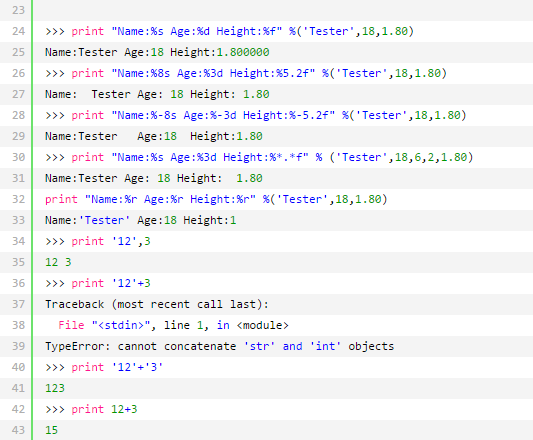
# Python特性

* 1. Texing1
  2. XX

# 输入输出

## 2.1 输出

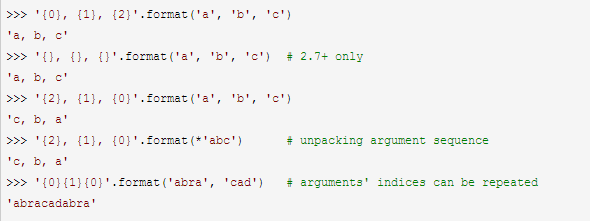




总结：  
1) print 输出字符串可以使用''或者""表示；  
2) print后可以有多个输出，以逗号分隔，最终逗号以空格输出，两边类型可以不一致；  
也可以用+号连接，但是+号要求两边类型必须一致，连接数字表示相加，连接字符串表示相连；  
3) 如果字符串本身包含""，可以使用''来表示；如果字符串本身包含''，可以使用""来表示;  
4) 如果字符串本身既包含''也包含""，可以使用\进行转义；常用转义字符：\n 换行； \t制表符； \\ 表示\本身  
5) 如果字符串本身既包含''也包含""，也可以使用'''来表示；'''一般用于输出多行文本中；  
6) 格式化输出可以指定输出字段的长度，%5.2f表示输出浮点数的小数点后有2位，总共长度为5位，默认为右对齐，不足的左侧补空格；  
%-5.2f表示左对齐，长度不足的右边补空格  
7) 对于格式未定的字段，可以通过类似%\*.\*f % (5,2,1.80)来设定表示%5.2f  
8) 对于类型不确定的字段输出可以统一用%r表示，%r的格式化指定格式后面再学习。

## 2.2 format输出

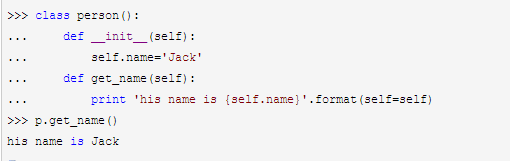
1) 接受位置参数输出



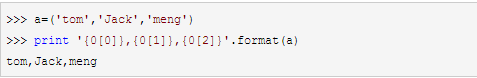
2) 接受参数名称输出



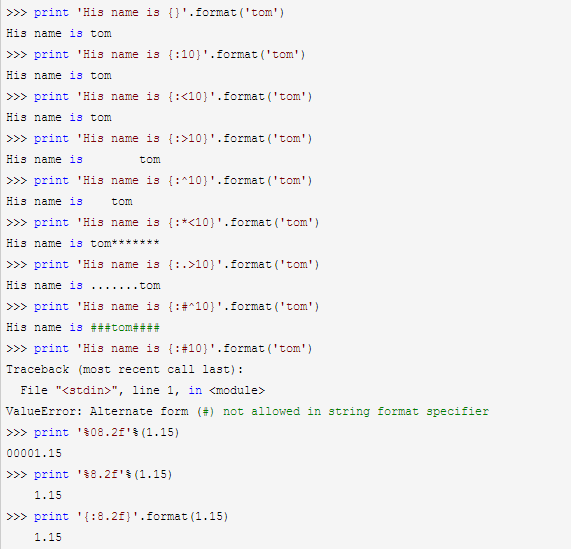
3) 接受对象属性输出



4) 接受序列的数据输出



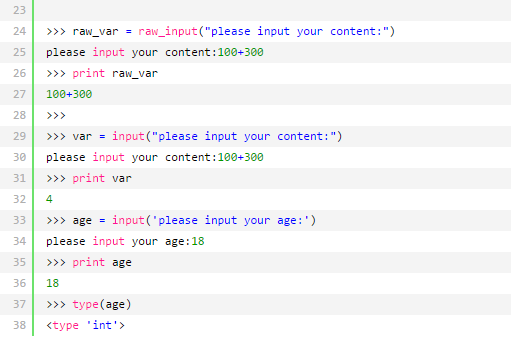
5) 格式化输出,可代替%s,%r



说明: 指定填充字符适合必须要指明对齐方向

## 2.3 input / raw\_input输入





总结：  
1) raw\_input将所有输入作为字符串看待，返回也是字符串类型  
2) input 没有将所有输入作为字符串看待，所以在输入字符串内容时候需要添加引号，否则会出错  
3) input可以接收表达式，并返回表达式结果  
4）input返回也不一定是字符串类型；如果接收的是带引号的字符串那么返回的才是字符串类型；

# 3. 数据类型和变量

## 3.1 变量

Python 变量仅是名称。变量名的创建是通过把类型的实例赋值给它们。变量本身并没有类型，而与它们绑定在一起的对象拥有类型。

名称只是个标签，同样，它也可以被一个完全不同的对象重新赋值。赋值操作使用=操作符，所以把一个值赋给一个变量就如下所示：

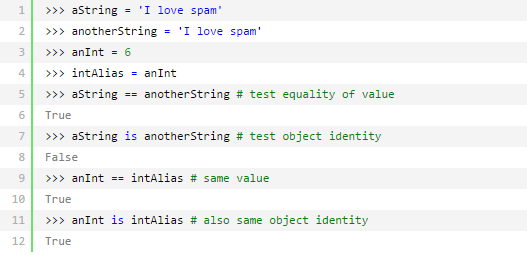
aVariable = aValue

这段代码把值aValue 绑定到变量名aVariable 上。如果此变量名不存在，解释器会把这个名称添加到合适的命名空间中。

因此，在Python 中区别变量和它指向的对象是非常重要的。

### 3.1.1 变量判断

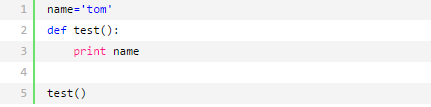
is判断两个变量是否指向同一个变量,  ==判断两个变量指向的对象是否相等



### 3.1.2 变量种类

变量根据作用域不同分为全局变量和局部变量。

例如1：



输出：  
tom  
[Finished in 0.2s]  
说明：函数中可以直接使用全局变量，打印name时候查找name变量，先查找局部name没有找到，再查找全局name，找到了然后输出。

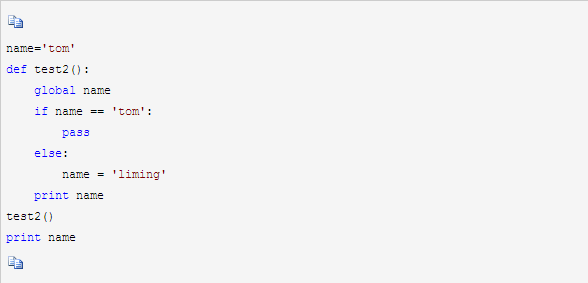
例如2：



执行报错:UnboundLocalError: local variable 'name' referenced before assignment  
说明：函数中先引用name判断是否等于'tom'，先找局部name没有找到，然后再找全局name找到了，所以使用全局name判断，然后else中又对这个全局name进行赋值，所以报错。

从上面代码中也可以看出函数中即使变量赋值的部分不会执行也会报错的。  
所以如果函数中的变量是引用的全局变量，那么函数中不能包含对该变量直接重新赋值的代码（不管是否会执行），如果必须重新赋值那么需要在函数中先通过global引用，如下

例如3：

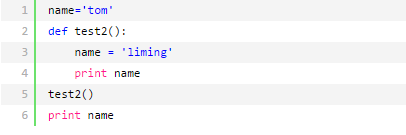


正常输出:

tom  
tom  
[Finished in 0.1s]

说明：这个函数中通过使用global引用了全局name，所以在函数中可以对该全局name重新赋值。

例如4：



正常输出：  
liming  
tom  
[Finished in 0.3s]  
说明：此处在函数中直接定义了一个局部name，注意这个是定义局部变量而不会查找全局变量。局部变量和全局变量都可以正常输出，互不影响。

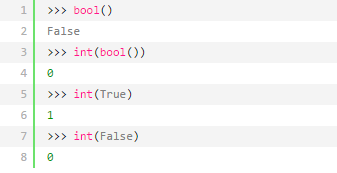
## 3.2 数据类型

### 3.2.1 数值类型:

数值类型包含整数和浮点数

### 3.2.2 布尔类型

在Python中 None,  False, 空字符串"", 0, 空列表[], 空字典{}, 空元组()都相当于False：



### 3.2.3 空值类型None

空值是Python里一个特殊的值，用None表示。所有对None对象的引用都是同一个实例



### 3.2.4 字符串

### 3.2.5 列表

#### 3.2.5.1 概念

1) list是一种有序的集合，可以随时添加和删除其中的元素; 空列表定义a=[]

2) 可以用索引访问其中某一个元素，索引从0开始；当索引超出了范围时，Python会报一个IndexError错误，所以，要确保索引不要越界，最后一个元素的索引是len(a) - 1  
    如果要取最后一个元素，除了计算索引位置外，还可以用-1做索引，直接获取最后一个元素,以此类推，可以获取倒数第2个、倒数第3个

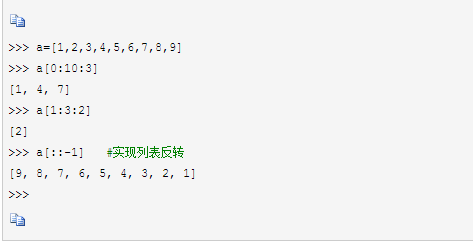
3）可以通过切片访问其中的多个元素，并且切片访问不会报索引越界，只会返回空列表[]  
4) list相关函数，可以通过help(list)查看

注意：通过索引获取单个元素返回的是对应得元素数据类型，通过切片访问返回的是list类型

#### 3.2.5.2 切片操作

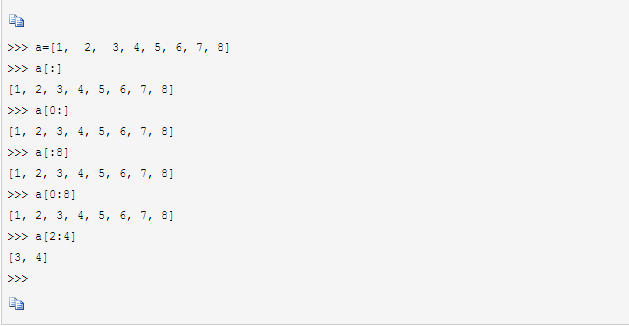
a[1:3:2]

表示获取索引从1到2的元素（不包括索引为3的元素），并且每两个元素只保留一个。

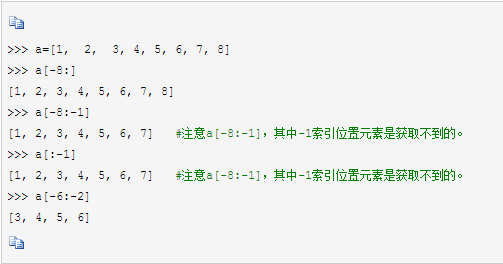


通过切片获取元素，可以先想象通过给对象中每个元素作标记，  
标记1:  0, 1, 2, 3, 4, 5, 6, 7  
      a=[1, 2, 3, 4, 5, 6, 7, 8]  
标记2: -8,-7, -6,-5, -4,-3,-2,-1

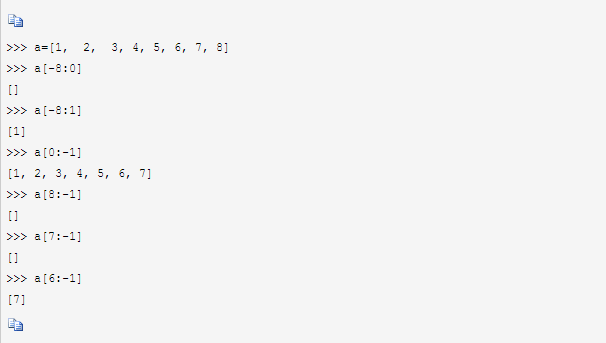
标记之后有3种获取方法，原则只要在两个标记位之间有元素存在就可以获取到，否则返回空列表。  
第一种：通过标记1顺序获取：



第二种：通过标记2顺序获取



第三种：通过混合标记1和2顺序获取：



a[b:c]获取时候，始终获取的是a[b]和a[c]之间的元素，当然不包括a[c]，并且a[b]元素在a中的位置始终是在a[c]的前面

>>> a=[1,2,3,4,5,6,7,8,9]

>>> a[-4:5]

[]

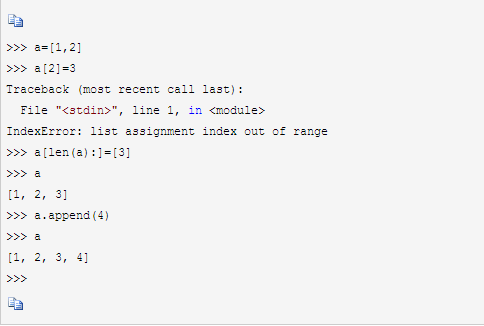
>>> a[-4:6]

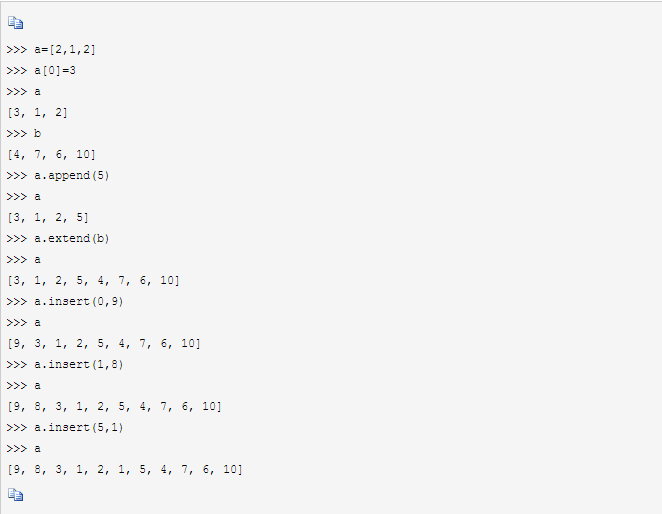
[6]

#### 3.2.5.3 列表操作

* 添加

append(元素) : 末尾添加元素，相当于 a[len(a):] = [x]  
extend(列表) : 将一个给定列表中的所有元素都添加到另一个列表中  
insert(索引位置，元素): 在指定位置插入一个元素。例如 a.insert(0, x) 会插入到整个列表之前，而 a.insert(len(a), x) 相当于 a.append(x)





注： 两个列表相并除了用extend方法，用+可以扩展另外一个列表，用\*可以扩展自身列表

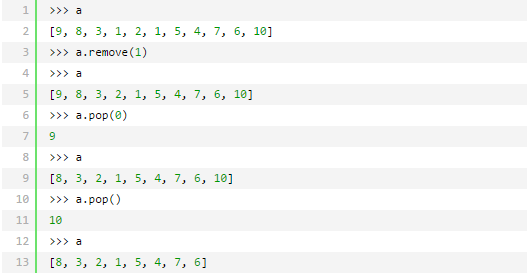


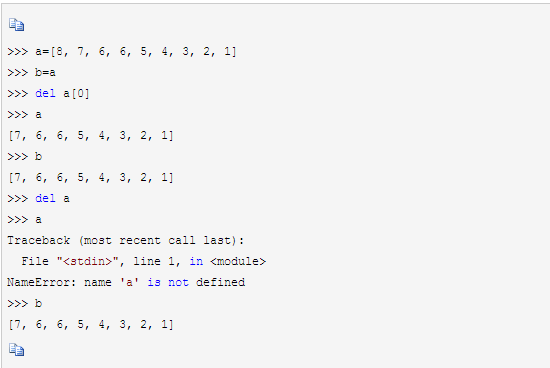
* 删除

remove(元素): 删除指定元素，如果有多个相同元素只删除第一个；  
pop(索引位置): 删除指定位置元素，如果不指定表示删除最后一个元素；

del:

注意三者之间的区别.





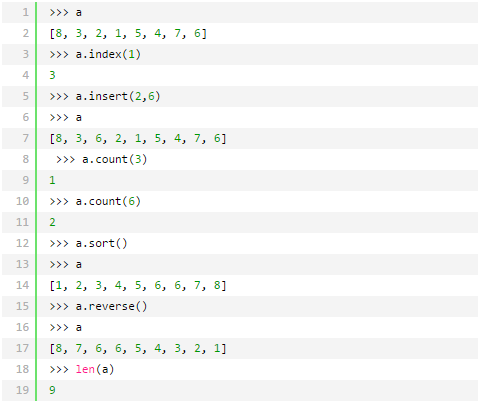
* 修改：

直接根据列表索引位置重新赋值即可，注意首先必须保证索引位置没有超出列表的索引范围；  
>>> a=[]  
>>> a[0]=3  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
IndexError: list assignment index out of range

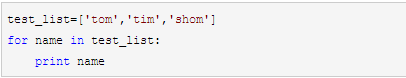
* 查询

index(元素): 根据元素内容查询其所在的索引位置;  
count(元素）: 统计某个元素在列表中有多少个；  
index和count的区别： 两者都可以用来查询列表元素，如果元素不在列表中的话index会出现异常，而count会返回0

len(列表): 统计列表的长度，也就是有多少个元素；  
sort():   
reverse():



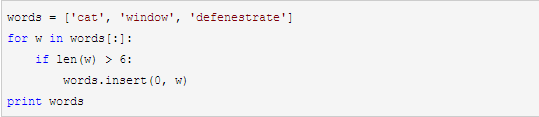
* 遍历:



输出结果：

tom  
tim  
shom  
[Finished in 0.1s]

若要在循环内部修改正在遍历的列表，可以通过对test\_list[:]遍历，test\_list[:]相当于对原始列表的拷贝，例如：



输出结果：

['defenestrate', 'cat', 'window', 'defenestrate']  
[Finished in 0.1s]

### 3.2.6 元组

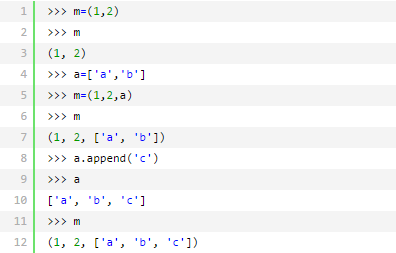
概念

1) tuple称为元组，有序但是元素不可变更，一旦初始化之后就不能再修改；

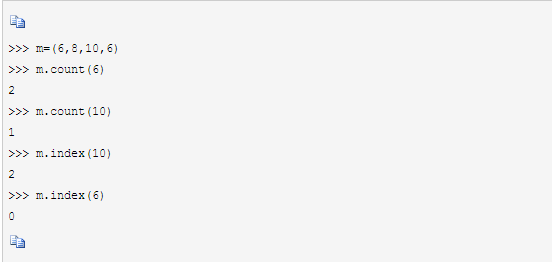
2) 空元组定义为 a=()

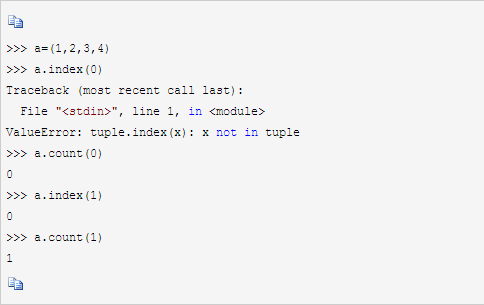
3) 只有一个元素的元组，定义为a=(1,)，不能定义为a=(1);

  
4) tuple也可以通过索引访问某一个元素，索引从0开始  
5) 元组的不可变更是指元组的元素指向不可更新，如果元素指向某个列表的时候，列表的内容更新不受限制；



6) tuple相关函数：  
index（元素): 和列表相同也是根据元祖内容查询它的索引位置；  
count(元素): 和列表一样也是统计某个元素出现的次数；  
index和count的区别： 两者都可以用来查询tuple元素，如果元素不在tuple中的话index会出现异常，而count会返回0





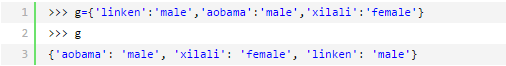
### 3.2.7 字典

#### 3.2.7.1 概念:

1) dict全称dictionary，使用键-值（key-value）存储; 空字典 a={}; 支持增删改查操作；  
2) 把数据放入dict的方法，除了初始化时指定外，还可以通过key放入;  
3) 由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值冲掉;  
4) 要避免key不存在的错误，有三种办法，一是通过in判断key是否存在：二是通过dict提供的get方法，如果key不存在，可以返回None，或者自己指定的value；还可以通过函数has\_key进行判断(这种一般不用了)；  
5) 要删除一个key，用pop(key)方法，对应的value也会从dict中删除：  
del a[key] 也可以删除某个元素  
del a 删除整个字典  
6) 请务必注意，dict内部存放的顺序和key放入的顺序是没有关系的。  
　　和list比较，dict有以下几个特点：  
　　　　　查找和插入的速度极快，不会随着key的增加而增加；  
　　　　　需要占用大量的内存，内存浪费多。  
　　而list相反：  
　　　　　查找和插入的时间随着元素的增加而增加；  
　　　　　占用空间小，浪费内存很少。  
　　所以，dict是用空间来换取时间的一种方法。  
7) dict的key必须是不可变对象,这是因为dict根据key来计算value的存储位置，如果每次计算相同的key得出的结果不同，那dict内部就完全混乱了。这个通过key计算位置的算法称为哈希算法（Hash）。要保证hash的正确性，作为key的对象就不能变。在Python中，字符串、整数等都是不可变的，因此，可以放心地作为key。而list是可变的，就不能作为key.  
8) dict的常用函数,help(dict)可以查看  
clear()：删除字典内所有元素  
get(key, default=None)：返回指定键的值，如果值不在字典中返回default值  
has\_key(key)：如果键在字典dict里返回true，否则返回false  
items()：以列表返回可遍历的(键, 值) 元组数组  
keys()：以列表返回一个字典所有的键  
values()：以列表返回字典中的所有值  
update(dict2)：把字典dict2的键/值对更新到dict里  
setdefault(key, default=None)：  
和get()类似, 如果键已经存在于字典中则返回字典中对应的value；但如果键不存在于字典中，将会添加键并将值设为default值;  
len(dict)：计算字典元素个数，即键的总数。  
fromkeys(seq,val)：创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值

#### 3.2.7.2 示例

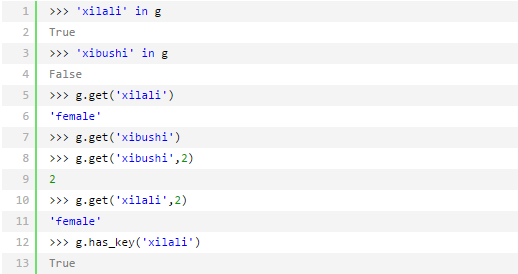
* 创建字典



* 访问字典元素



* 字典元素判断

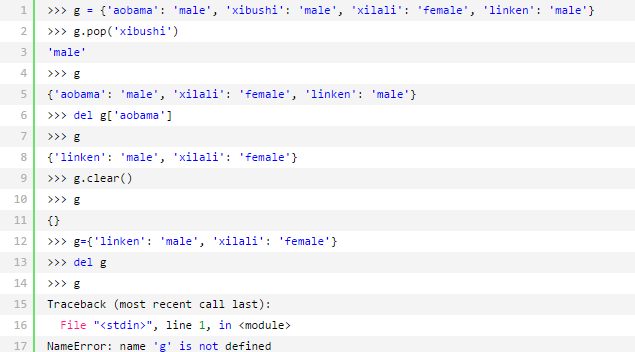


* 修改字典

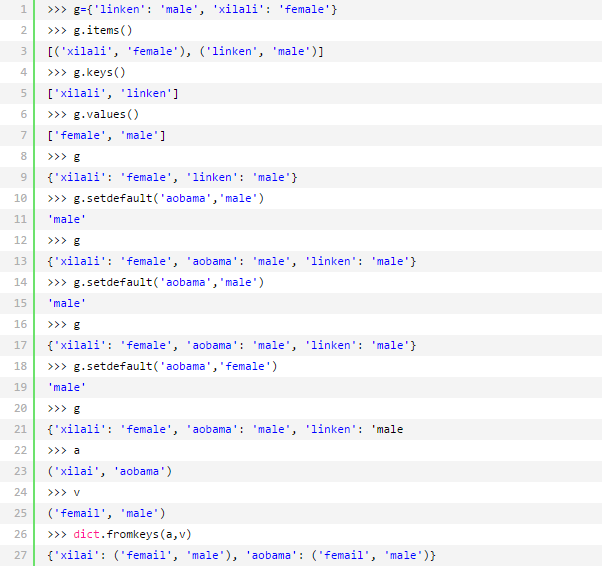


update方法更新字典，如果两个字典元素中key值有重复则取用新的value

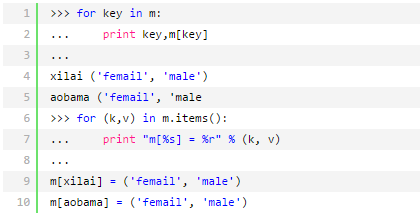
* 删除字典



* 字典函数



* 字典遍历



### 集合

## 3.3 可变类型

python中，万物皆对象。python中不存在所谓的传值调用，一切传递的都是对象的引用，也可以认为是传址。

python中，对象分为可变(mutable)和不可变(immutable)两种类型。

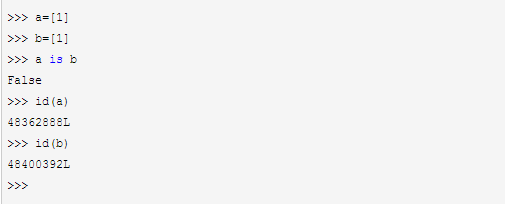
元组（tuple)、数值型（number)、字符串(string)均为不可变对象，

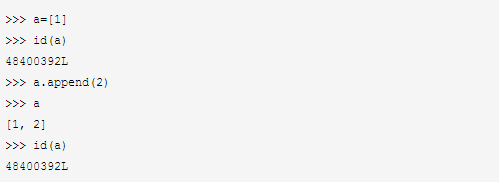
而字典型(dictionary)和列表型(list)的对象是可变对象。

通过上面知道可以通过is判断是否指向同一个对象。还可通过id查看内存地址来区别是否指向同一个对象。

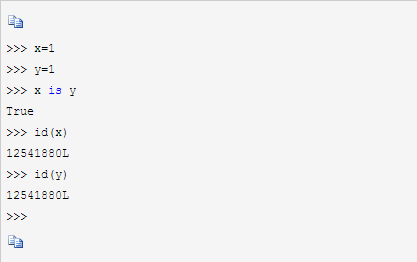
另外是否可变是指两方面，一个是内存地址中的内容是否可以变更，二是同一内容是否会存储在不同内存。

List可变：



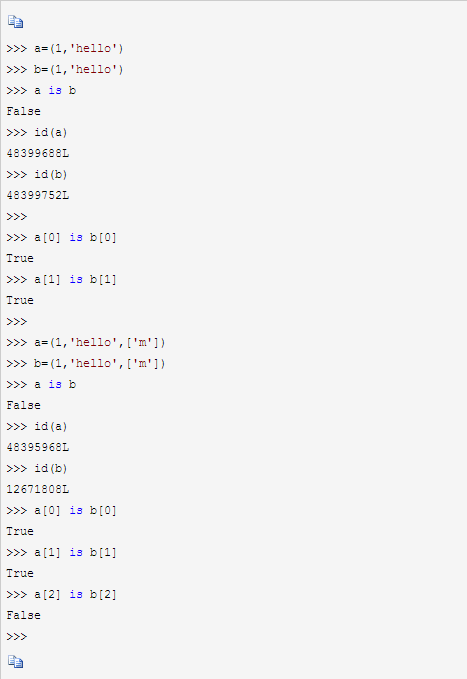


数值型不可变：



字符串不可变：

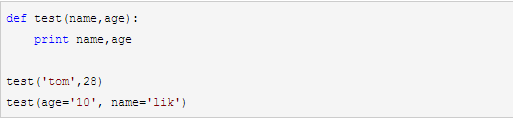




# 函数

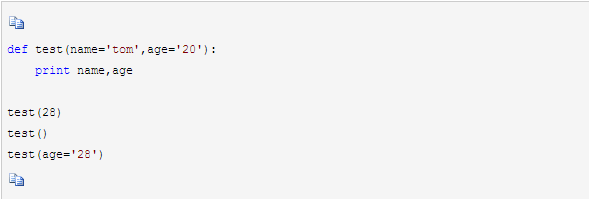
## 4.1 函数参数

### 4.1.1 必选参数(非关键字参数形式)

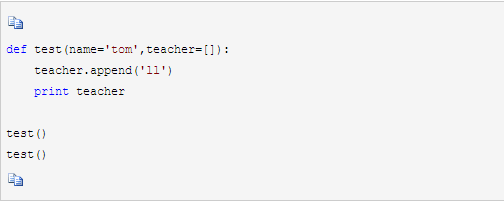


输出结果：  
tom 28  
lik 10  
[Finished in 0.2s]  
说明：函数定义的必选参数，调用函数时候参数的个数以及位置要分别对应；如果在调用时候想不按照位置顺序传递，那么必须指定参数名，这个叫指定关键字参数

### 4.1.2 默认参数(关键字参数形式)



输出结果：  
28 20  
tom 20  
tom 28  
[Finished in 0.2s]  
说明：函数定义的参数带有默认值，这个叫默认参数。针对默认参数，函数调用时候可以不指定参数值，也可以指定参数名重新赋值，也可以不指定参数名此时按参数顺序传递；  
另外默认参数要定义为不可变对象，不要定义为可变对象如list，否则会发生逻辑问题,如下每次调用test时候teacher默认参数的值没有被清空。



### 4.1.3 非关键字可变参数

针对必选参数样式的参数，但是又不是必选参数，我们可以定义为非关键字可变参数\*args.

args是一个list或者tuple,通过在前面加一个\*就表示list或者tuple中的单个元素，从而表示多个必选样式的可选参数值



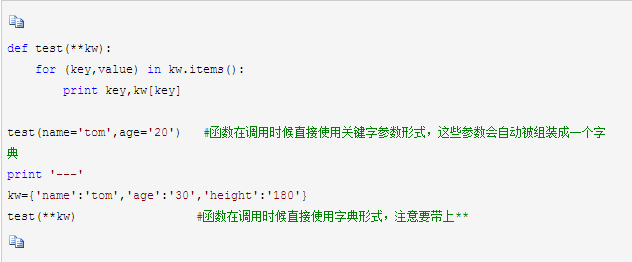
输出结果：

('tom', 18)  
tom  
18  
('tom', 20, 'address')  
tom  
20  
address  
('tim', 20, 'address', 'work')  
tim  
20  
address  
work  
[Finished in 0.1s]

### 4.1.4 关键字可变参数

针对默认参数样式的参数，但是又不是必选参数，我们可以定义为关键字可变参数\*\*kw

kw是一个字典,通过在前面加一个\*\*就表示字典中的元素，从而表示多个默认参数样式的可选参数值



输出结果：

age 20  
name tom  
---  
age 30  
name tom  
height 180  
[Finished in 0.1s]

### 4.1.5 混合参数

混合参数定义，必须遵守参数定义顺序：必选参数->默认参数->非关键字可变参数->关键字可变参数。另外默认参数最好定义为不可变对象，不要定义为list。

混合参数调用，必须遵守参数设置顺序：关键字参数之后不能再出现非关键字参数。通俗的说就是有指定参数名的参数之后，不能再出现没有指定参数名的参数。



输出结果：

tom 18  
()  
{}  
----  
tom 18  
('grade', 'address')  
{}  
----  
tom 18  
()  
{'grade': '3', 'address': 'suzhou'}  
----  
tom 18  
('3',)  
{'address': 'suzhou'}  
[Finished in 0.1s]

下面是一些错误的调用方法, 违背了规则：关键字参数之后不能再出现非关键字参数

test(name='tom', 18)   
test('tom', 18, grade='3','address')

## 4.2 函数执行

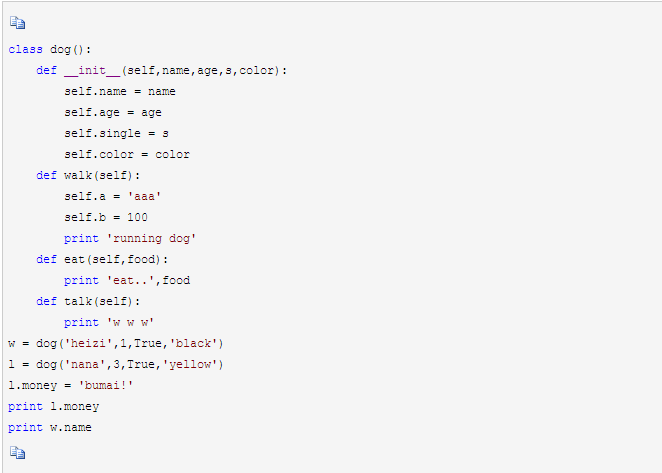
1）函数体内部的语句在执行时，一旦执行到return时，函数就执行完毕并将结果返回。哪怕是在循环中循环还未结束也会结束函数执行。  
如果没有return语句，函数执行完毕后也会返回结果，只是结果为None。return None可以简写为return  
2) 空函数,如果想定义一个什么事也不做的空函数，可以用pass语句：



3) pass语句什么都不做，那有什么用？实际上pass可以用来作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个pass，让代码能运行起来。  
4) 调用函数时，如果参数个数不对，Python解释器会自动检查出来，并抛出TypeError: my\_abs() takes exactly 1 argument (2 given)  
但是如果参数类型不对，Python解释器就无法帮我们检查.所以需要时候可以先对参数类型做检查。  
5) 返回多个值，比如return x,y，实际返回的是一个元祖(x,y)

# 面向对象

## 类的定义和实例化



总结：  
1) class后面紧接着是类名，类的类型分为经典类和新式类。经典类后面括号不继承任何东西。新式类括号里面面一般最好添加(object),表示该类是从哪个类继承下来的。目前如果没有合适的继承类，就使用object类，这是所有类最终都会继承的类。  
2) 由于类可以起到模板的作用，因此，可以在创建实例的时候，把一些我们认为必须绑定的属性强制填进去。通过定义一个特殊的\_\_init\_\_方法，在创建实例的时候，就把name，age等属性绑上去  
3) 注意到\_\_init\_\_为初始化方法，第一个参数永远是self，表示创建的实例本身  
4) 变量w就是指向dog的类的实例变量，而dog本身就是一个类。（）里面的都是给他捆绑的属性  
5) 有了\_\_init\_\_方法，在创建实例的时候，就不能传入空的参数了，必须传入与\_\_init\_\_方法匹配的参数，但self不需要传，Python解释器自己会把实例变量传进去。

另外和普通的函数相比，在类中定义的函数只有一点不同，就是第一个参数永远是实例变量self，并且，调用时，不用传递该参数。除此之外，类的方法和普通函数没有什么区别，所以，你仍然可以用默认参数、可变参数和关键字参数。  
6) 类的方法：在上面的dog类中，每个实例就拥有各自的name和age这些数据，因为dog的实例本身就有这些数据，要访问这些数据，就没有必要从外面的函数去访问，可以直接在dog类的内部定义访问数据的函数，这样，就把“数据”给封装起来了。这些封装数据的函数是和dog类本身是关联起来的，我们称之为类的方法。要定义一个方法，除了第一个参数是self外，其他和普通函数一样。要调用一个方法，只需要在实例变量上直接调用，除了self不用传递，其他参数正常传入。

## 属性



总结：  
属性分为类的属性和实例属性。定义在类中方法之外的属性，称为类的属性，又分为类的公有属性和私有属性；  
1) 类的公有属性，可以被类和实例访问；  
2) 类的私有属性(以\_\_为前缀)，不可以被类和实例访问，只可用于类的方法中通过self.\_\_private\_attrs调用  
3) 实例的公有属性，可以被实例访问  
4) 实例的私有属性(以\_\_为前缀)，不可以被实例访问，只可用于类的方法中通过self.\_\_private\_attrs调用  
5) 如果类的公有属性和实例公有属性同名，则两者互相不影响；

（类和实例的私有属性都是只能被类中定义的方法使用)

## 方法

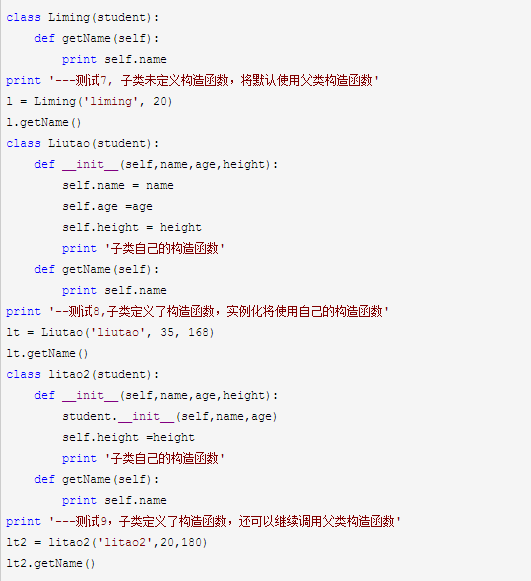




总结：  
方法包含实例方法，静态方法，类方法，其中实例方法又分为私有方法和公有方法。  
1) 私有方法以两个下划线开头，不可以被实例访问，只能在类的方法中通过self.\_\_private\_methods调用  
2) 公有方法，可以被实例访问  
3) 以上说的方法实际都是实例方法，其它还有静态方法，类方法。  
4) 静态方法使用装饰器@staticmethod定义，并且方法参数中不能有self. 类对象和实例都可以调用静态方法,但是无法访问类属性、实例属性，相当于一个相对独立的方法，跟类其实没什么关系;  
5) 类方法使用@classmethod装饰器定义，其第一个参数必须为cls，且不需要self参数。类对象和实例都可以调用类方法,可以访问类属性包含公有属性和私有属性，但是无法访问实例属性。  
还有一种类中普通的方法（没有self参数)，这种方法既可以被类直接调用也可以被类的实例对象调用，但是被实例对象调用的时候，要求方法至少有一个参数，因为调用时会将实例对象本身传给第一个参数。staticmethod函数功能就是将这种方法定义成类的静态方法，正确的方法是使用 @staticmethod装饰器，这样在实例对象调用的时候，不会把实例对象本身传入静态方法的第一个参数了。

## 构造方法





总结：  
1) 如果子类没有定义构造函数，将默认使用父类构造函数；  
2) 如果一个子类从多个父类派生，而子类又没有自己的构造函数,则按顺序继承，  
哪个父类在最前面且它又有自己的构造函数，就继承它的构造函数；如果最前面第一个父类没有构造函数，则继承第2个的构造函数，第2个没有的话，再往后找，以此类推。  
3) 如果子类有自己的构造函数，不会自动调用父类的构造函数，只会调用自己的构造函数  
4) 如果需要用到父类的构造函数，则需要在子类的构造函数中显式的调用  
5) 显式调用父类构造函数有两种方法：super以及 通过父类直接调用\_\_init\_\_函数；

## 多态



总结：  
1) 如果父类方法的功能不能满足需求，可以在子类重写父类的方法。实例对象调用方法时会调用其对应子类的重写后的方法

## 继承

总结：  
1) 定义：class 派生类名（基类名)  
2) 特点：  
I: 在继承中基类的构造（\_\_init\_\_()方法）不会被自动调用，它需要在其派生类的构造中亲自专门调用。使用super().\_\_init\_\_()或parentClassName.\_\_init\_\_()  
II: 在调用基类的方法时，需要加上基类的类名前缀，且需要带上self参数变量。区别于在类中调用普通函数时并不需要带上self参数  
III: 总是首先查找对应类型的方法，如果它不能在派生类中找到对应的方法，它才开始到基类中逐个查找。（先在本类中查找调用的方法，找不到才去基类中找）。

## 封装

# 高级特性

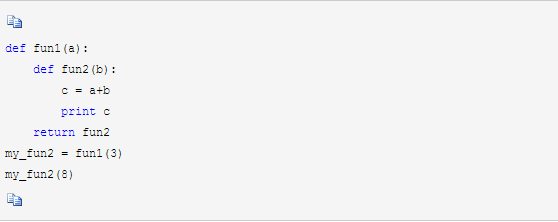
## 6,1 闭包

闭包需满足三个条件： 1. 是嵌套函数；2. 内部函数使用了外部函数的变量；3. 外部函数返回内部函数

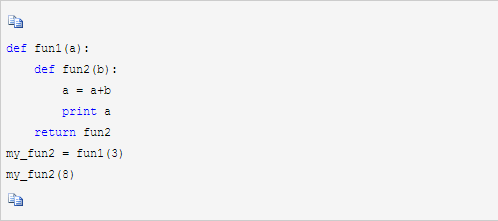
例如1：



例如2：



例如3（注意点）：

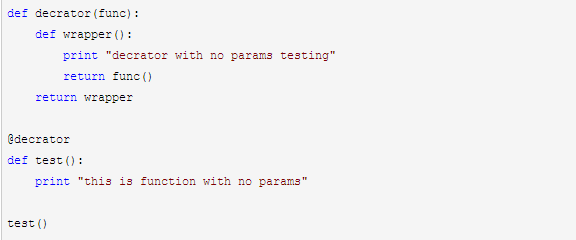


这段程序的本意是要通过在每次调用闭包函数时都对变量a进行递增的操作，但是会出现找不到变量a的错误。

## 装饰器

### 6.2.1被装饰对象为函数，函数作为装饰器

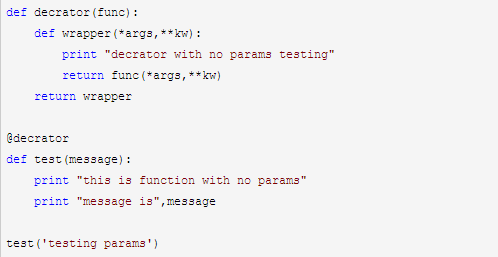
* + 被装饰函数无参数，装饰器函数无参数



Result:

decrator with no params testing  
this is function with no params  
[Finished in 0.1s]

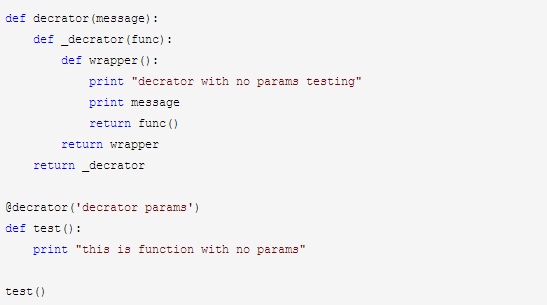
* + 被装饰函数有参数，装饰器函数无参数



Result:

decrator with no params testing  
this is function with no params  
message is testing params  
[Finished in 0.1s]

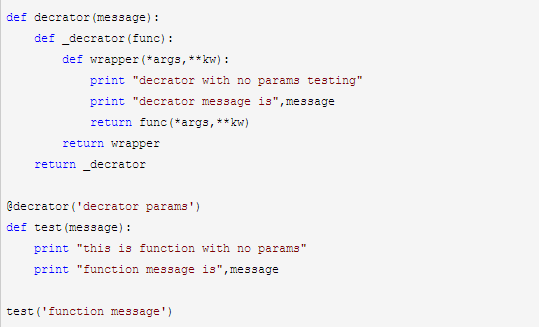
* + 被装饰函数无参数，装饰器函数有参数



Result:

decrator with no params testing  
decrator params  
this is function with no params  
[Finished in 0.1s]

* + 被装饰函数有参数，装饰器函数有参数

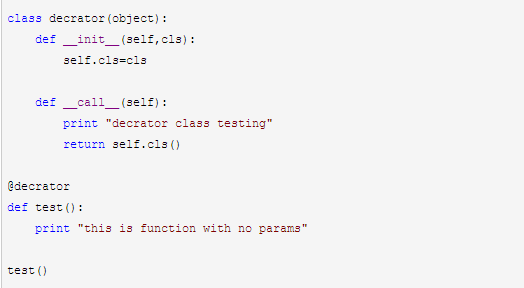


Result:

decrator with no params testing  
decrator message is decrator params  
this is function with no params  
function message is function message  
[Finished in 0.1s]

### 6.2.2被装饰对象为函数，类作为装饰器

* + 被装饰函数无参数，装饰器类无参数



Result:

decrator class testing  
this is function with no params  
[Finished in 0.1s]

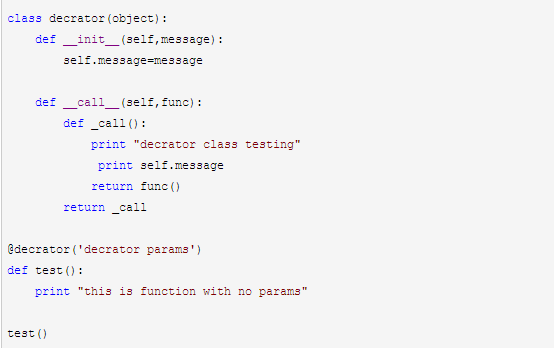
* + 被装饰函数有参数，装饰器类无参数



Result:

decrator class testing  
this is function with no params  
function messages is function params  
[Finished in 0.1s]

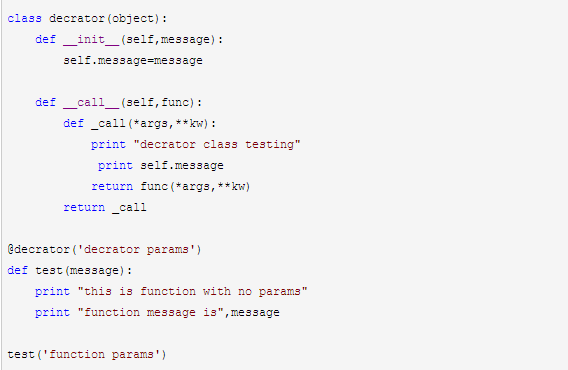
* + 被装饰函数无参数，装饰器类有参数



Result:

decrator class testing  
decrator params  
this is function with no params  
[Finished in 0.1s]

* + 被装饰函数有参数，装饰器类有参数

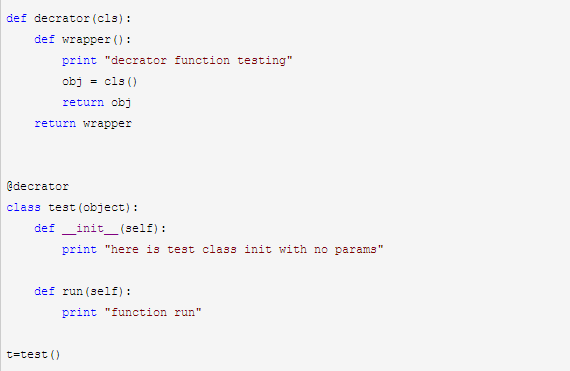


Result:

decrator class testing  
decrator params  
this is function with no params  
function message is function params  
[Finished in 0.1s]

### 6.2.3被装饰对象为类，函数作为装饰器

* + 被装饰类无参数，装饰器函数无参数



Result:

decrator function testing  
here is test class init with no params  
[Finished in 0.1s]

* + 被装饰类有参数，装饰器函数无参数



Result:

decrator function testing  
here is test class init with params  
class params  
[Finished in 0.1s]

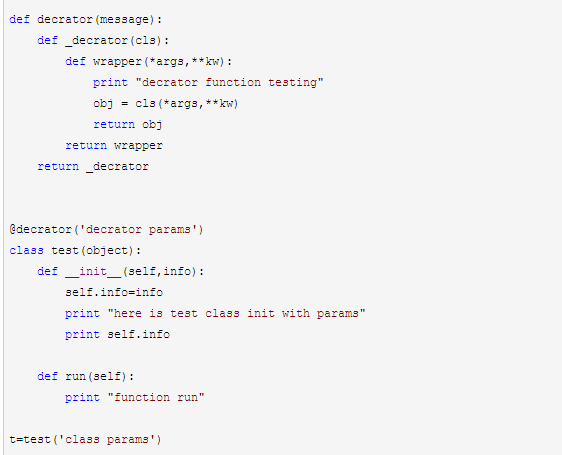
* + 被装饰类无参数，装饰器函数有参数



Result:

decrator function testing  
here is test class init with params  
[Finished in 0.1s]

* + 被装饰类有参数，装饰器函数有参数

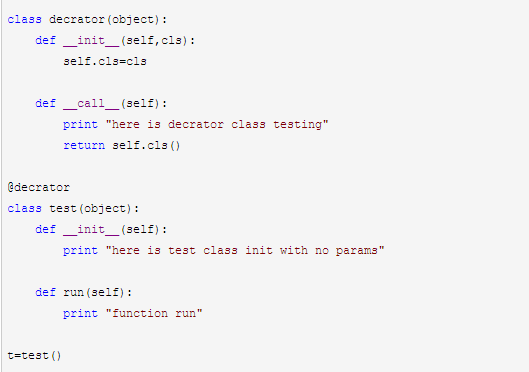


Reslut:

decrator function testing  
here is test class init with params  
class params  
[Finished in 0.1s]

### 6.2.4被装饰对象为类，类作为装饰器

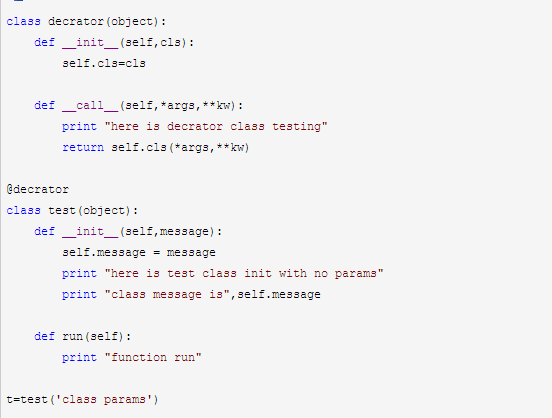
* + 被装饰类无参数，装饰器类无参数



Result:

here is decrator class testing  
here is test class init with no params  
[Finished in 0.1s]

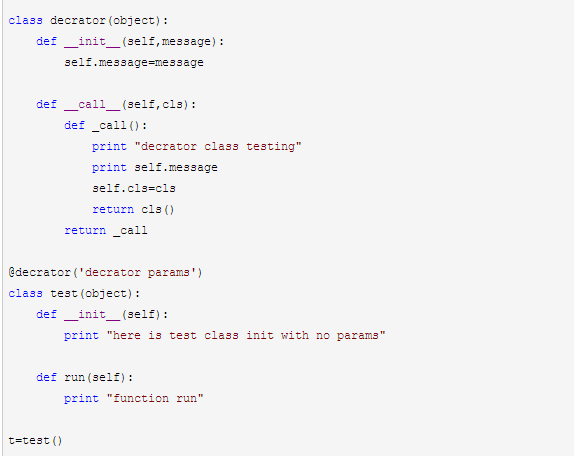
* + 被装饰类有参数，装饰器类无参数



Result:

here is decrator class testing  
here is test class init with no params  
class message is class params  
[Finished in 0.1s]

* + 被装饰类无参数，装饰器类有参数



Result:

decrator class testing  
decrator params  
here is test class init with no params  
[Finished in 0.1s]

* + 被装饰类有参数，装饰器类有参数



Result:

decrator class testing  
decrator params  
here is test class init with no params  
function params  
[Finished in 0.1s]

### 6.2.5装饰器参数传递给被装饰对象，作为被装饰对象参数使用



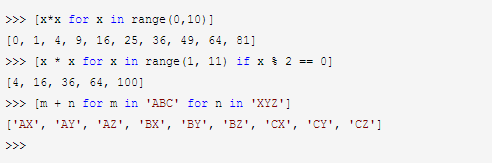
Result:

decrator function testing  
here is test class init with params  
function run  
class params  
0  
[Finished in 0.2s]

## 6.3 列表生成式，生成器，迭代器，可迭代对象

### 6.3.1 列表生成式

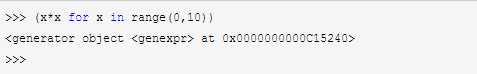
列表生成式即创建列表的方式,这里是中括号[]



### **6.3.2 生成器(Generator)**

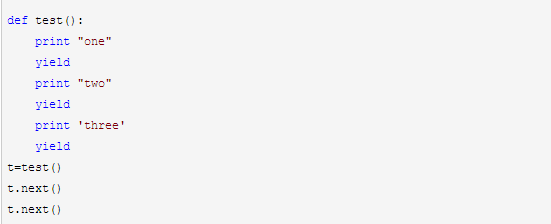
生成器有两种方法：第一个是通过生成器表达式来创建，另外一种是通过定义带有yield的函数来实现。

生成器, 这里是小括号()



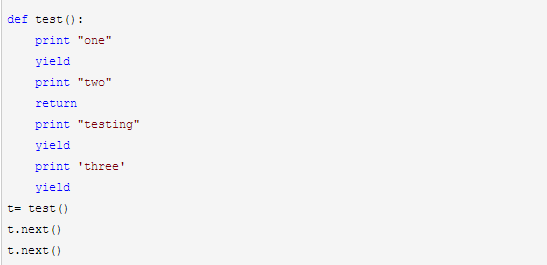
说明：从这个可以看出生成器和列表生成的区别：列表生成式直接返回了表达式的结果列表， 而生成器是一个对象，该对象包含了对表达式结果的计算引用。  
前者如果数据量太大时候会占用很大内存，而生成器保存的只是一个计算对象，不会占用很大内存。

如果上述算法比较复杂，不好写在一个()，那么可以使用定义一个包含yield的函数来实现，这个就是第二种实现方法。  
如果一个函数定义中包含了yield关键字，这个函数就不再是普通函数，而是一个generator object。  
要想调用这个函数，需要使用next()函数，并且遇到yield语句返回(可以把yield理解为return)。例如：

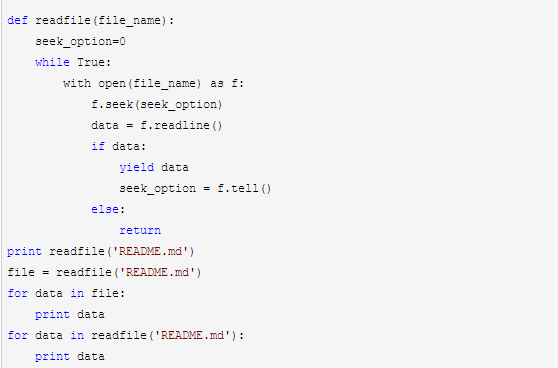


输出结果为：  
one  
two  
[Finished in 0.2s]

说明：首先创建生成器对象实例t，然后通过next方法调用函数，每一次执行到yield语句会被终止，并在下次该实例再次执行next方法时候继续执行。所以第一个next方法打印one，第二个next打印two.  
test().next()  
test().next()  
输出结果为：  
one  
one  
[Finished in 0.2s]  
说明：注意必须是同一个实例不断调用next方法，如上这种调用方法达不到想要的效果。



说明：如果生成器中有return,在执行过程中 return，则直接抛出 StopIteration 终止迭代。



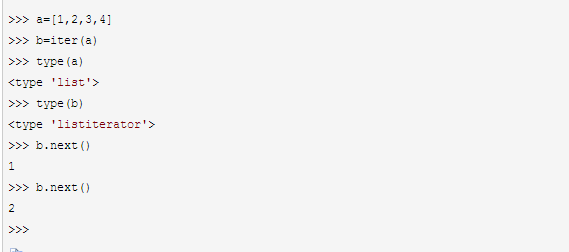
说明： 上述定义了一个读取文件的生成器。生成器一般可以和for循环配合使用，这样可以迭代生成器元素，而不使用next方法。

### **6.3.3 迭代器(Iterators)**

任何具有\_\_next\_\_()方法的对象都是迭代器。  
所以生成器是一种特殊的迭代器，任意一个生成器都属于迭代器。

### **6.3.4 可迭代对象**

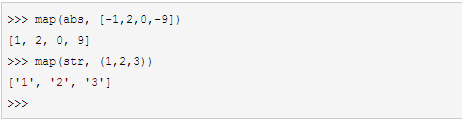
可迭代对象可以为任意对象，不一定非得是基本数据结构，只要这个对象可以返回一个iterator。  
可迭代对象通过iter函数可以转化为迭代器,如下例a是一个可迭代对象，而b是一个迭代器。  
实际在for循环的过程中也都是将可迭代对象首先转化成迭代器，然后通过next方法读取元素直到抛出异常。



## 内建函数

### 6.4.1 map

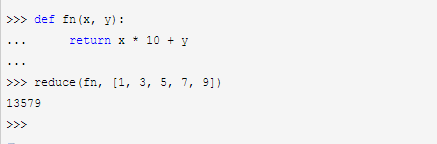
map(function,sequence)函数接收两个参数，一个是函数，一个是序列，map将传入的函数依次作用到序列的每个元素，并把结果作为新的list返回



### reduce

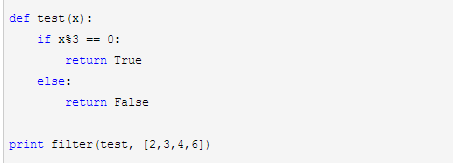
reduce把一个函数作用在一个序列[x1, x2, x3...]上，这个函数必须接收两个参数，reduce把结果继续和序列的下一个元素做累积计算，其效果就是：

reduce(f, [x1, x2, x3, x4]) = f(f(f(x1, x2), x3), x4)



### filter

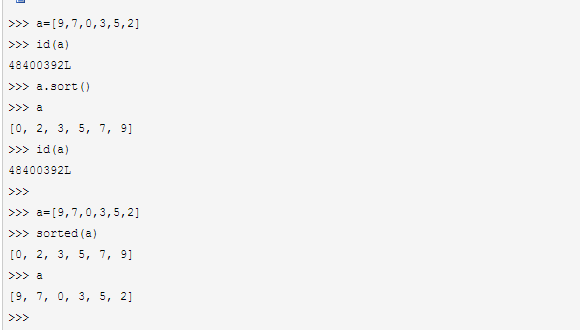
filter()也接收一个函数和一个序列。和map()不同的时，filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素。



### Sorted

Python list内置sort()方法用来排序, 但是list.sort()方法仅被定义在list中，只能作用于list。

而sorted()方法就是全局的，它对可迭代的序列排序生成新的序列。



从上面可以看出使用list.sort()方法来排序，此时list本身将被修改，而用sorted()方法会返回一个新的list，不影响原先list。

## 匿名函数

关键字lambda表示匿名函数。

lambda会创建一个函数对象，但不会把这个函数对象赋给一个变量，而def则会把函数对象赋值给一个变量。

另外lambda只是一个表达式，而def则是一个语句。简单的说在def中，用return可以返回的也可以放在lambda后面，不能用return返回的也不能定义在python lambda后面。因此，像if或for这种语句就不能用于lambda中，lambda一般只用来定义简单的函数。

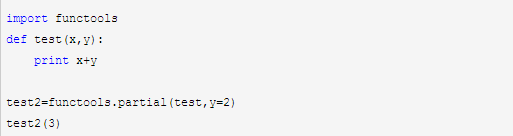


这里手动将lamda的匿名函数赋值给了变量f

## 偏函数

通过functools.partial帮助我们创建一个偏函数，作用就是把一个函数的某些参数给固定住（也就是设置默认值）返回一个新的函数，

调用这个新函数会更简单，而且不用更改原始函数。



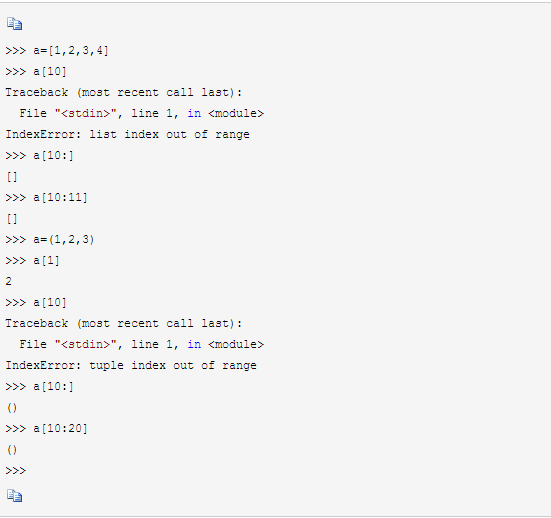
这里函数test定义了两个参数，如果某种情况下y一直是2的情况，那么我们可以重新生成一个新的函数test2将参数y给固定，

然后调用test2就相对简单一点，而且没有更改原先的函数。 当然也可以更改原先函数设置默认值即可，如下：

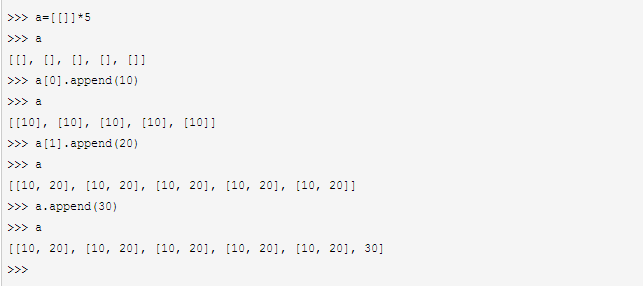


# FAQ：

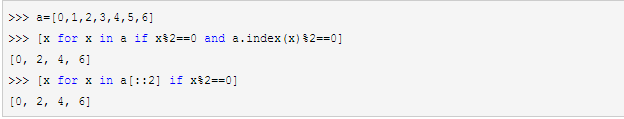
1. 超过列表成员个数的索引访问列表会出现IndexError错误，但是如果用切片去访问就不会报错，而是返回一个空列表。同样元组也是



1. a=[ [ ] ] \* 5的关键一点是它不是创造一个包含五个独立列表的列表，而是它是一个创建了包含对同一个列表五次引用的列表。



1. 给定一个含有N个数字的列表，取出偶数切片中的偶数值。以下两种方式，第二种先取出偶数切片的值再判断是否偶数。  
   a[::2]表示每两个元素取一个元素。



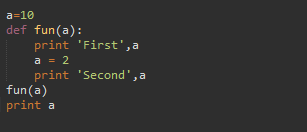
1. 下面为什么输出结果不一样？



官方说明：类型是属于对象的，而不是变量。而对象有两种,“可更改”（mutable）与“不可更改”（immutable）对象。在python中，strings, tuples, 和numbers是不可更改的对象，而list,dict等则是可以修改的对象。(这就是这个问题的重点)

当一个引用传递给函数的时候,函数自动复制一份引用,这个函数里的引用和外边的引用没有半毛关系了.所以第一个例子里函数把引用指向了一个不可变对象,当函数返回的时候,外面的引用没半毛感觉.而第二个例子就不一样了,函数内的引用指向的是可变对象,对它的操作就和定位了指针地址一样,在内存里进行修改.

自己解读:



Result:

First 10

Second 2

10

[Finished in 0.1s]

说明：1. 之前说过函数中不能对全局变量赋值？

def fun(a)和fun(a)，这两个a是不同的，第一个a是参数，第二个a是个全局变量。

所以这里不存在函数中引用了全局变量之后又对全局变量赋值。



Result:

[1]

[2]

[Finished in 0.1s]



Result:

Traceback (most recent call last):

File "D:\workspace\py\_demo\py\_demo\test2.py", line 89, in <module>

fun()

File "D:\workspace\py\_demo\py\_demo\test2.py", line 84, in fun

if a==[]:

UnboundLocalError: local variable 'a' referenced before assignment

[Finished in 0.1s]



Result:

[2, 1]

[2, 1]

[Finished in 0.1s]

这里说明如果全局变量是个可变对象，那么函数中是可以修改全局变量的值的，只是不能将变量引用赋值。

aVariable = aValue