在Python的循环中，注意行的缩进，循环根据行的缩进来判断循环体的范围。

for x in range(101):

sum = sum + x

print("sum = ", sum)

print("sum = ", sum)只会执行1次

for x in range(101):

sum = sum + x

print("sum = ", sum)

print("sum = ", sum)会执行101次

在Python中，可以在同一py文件中定义两个完全相同的函数，不过没什么意义，也可以定义名称相同，但参数不同的函数， 与C++中的函数重载不同的是：之前定义的函数会被之后定义的同名函数覆盖掉，不能再使用了。

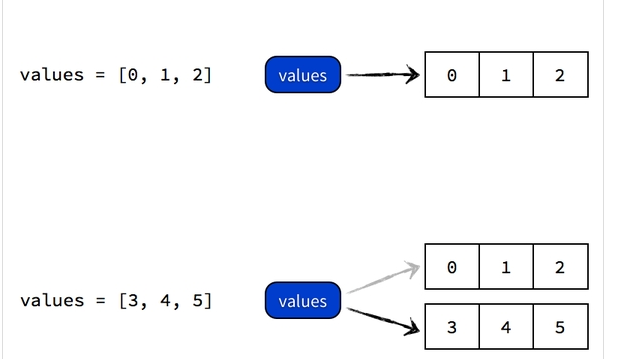
参考：<http://my.oschina.net/leejun2005/blog/145911?fromerr=9N0CIDGg>

<http://xianglong.me/article/python-variable-quote-copy-and-scope/>

Python中的“变量”：变量只是“标签”，它是对象的引用，变量无类型，而对象有类型。

values = [0, 1, 2]，先创建一个列表对象[0, 1, 2]，然后贴上标签values

values = [3, 4, 5]，创建另一个列表对象[3, 4, 5]，然后把标签values从前面的列表对象[0, 1, 2]上撕下来，贴到[3, 4, 5]对象上。



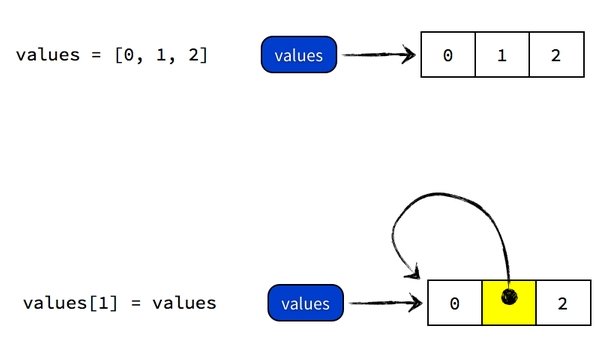
执行下面的pyton代码：

object\_reference.py

values = [0, 1, 2]

values[1] = values

print(values) #赋值无限次

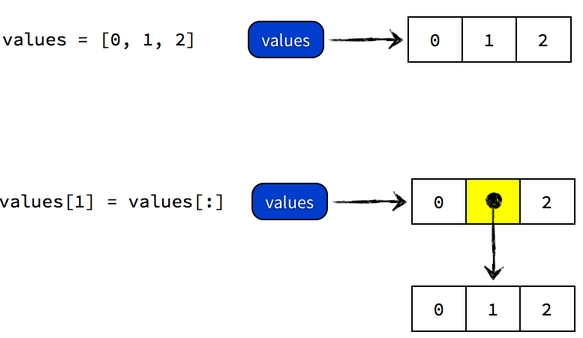


values = [0, 1, 2]

values[1] = values[:]

print(values) #[0,[0, 1, 2], 2]

values[:]生成对象的拷贝，得到一个新的对象。



values[:]是浅拷贝，有时需要使用深拷贝。

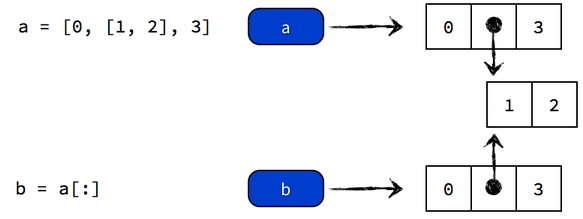
a = [0, [1, 2], 3]

b = a[:]

a[0] = 8

a[1][1] = 9

a为[8, [1, 9], 3]，b为[0, [1, 9], 3]，可以发现b的第2个元素也发生的改变。原因如下图：



修改a[1][1]就会同时修改b[1][1]

深拷贝：

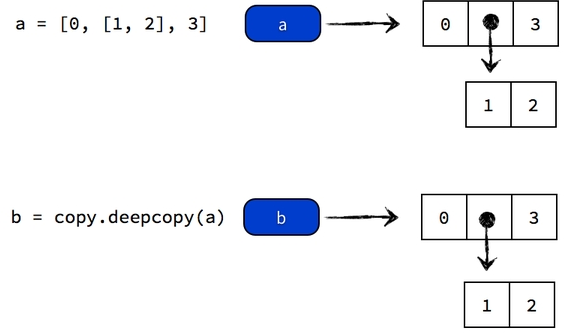
import copy

a = [0, [1, 2], 3]

b = copy.deepcopy(a)

a[0] = 8

a[1][1] = 9



deepcopy本质上是递归的浅拷贝

Python中的可变对象和不可变对象

不可变对象：int,float,long,str,tuple等

可变对象：list,set,dict等

immutable\_mutable\_object1.py

a = "test"

a\_list = [1, 2]

print(id(a))# 2083942140536

print(id(a\_list)) #2083942573256

a = "abc"

a\_list = [1, 2]

print(id(a)) # 2083942007952

print(id(a\_list)) # 2083942573384

可以看出无论是不可变对象还是可变对象，使用赋值运算符=改变值时，都会创建新的对象。

对于可变对象而言，可以修改其值，而对于不可变对象而言，其值是不可变的。

Python中的可变对象有点类似于C++中的引用.

b\_list = a\_list

b\_list[0] = 4

print(a\_list) # [4, 2]

print(b\_list) # [4, 2]

a\_list += [5, 6]

print(a\_list) # [4, 2, 5, 6]

print(b\_list) # [4, 2, 5, 6]

a\_list = a\_list + [7, 8] #创建了新的对象

print(a\_list) # [4, 2, 5, 6, 7, 8]

print(b\_list) # [4, 2, 5, 6]

immutable\_mutable\_object.py

def func\_int(a):

a+= 4

def func\_list(a\_list):

a\_list[0] = 4

t = 0

func\_int(t)

print(t) # t = 0

t\_list = [1, 2, 3]

func\_list(t\_list)

print(t\_list) # t\_list = [4, 2, 3]

对于可变对象，对象的操作不会重建对象，对于不可变对象，每次操作就重建新的对象。

在函数参数传递时，Python把参数里传入的变量对应的对象的引用依次赋给对应的函数内部变量。func\_int中的局部变量a其实是全局变量t所指向对象的另一个引用，由于整数对象是不可变的，所以func\_int对变量a进行修改时，实际是将局部变量a指向到了整数对象“4”。而对于函数func\_list来说，其修改的是一个可变对象，局部变量a\_list和全局变量t\_list指向的是同一个变量。

可变对象list的赋值运算符=和append/extend差别

list\_a = []

def func\_list1():

list\_a = [1]

func\_list1()

print(list\_a) #[]

print("=======================")

list\_b = []

def func\_list2():

list\_b.append(1)

func\_list2()

print(list\_b) # [1]

“=”创建了局部变量，而append/extend重用了全局变量。这有点类似C++中的指针，在函数中改变指针本身不会影响外部传入的指针，但在函数中改变指针所指向的值则会影响外部传入的指针所指向的值。