**[深入Python(1): 字典排序 关于sort()、reversed()、sorted()](http://www.cnblogs.com/BeginMan/p/3193081.html)**

一、Python的排序

**1、reversed()**

这个很好理解，reversed英文意思就是：adj. 颠倒的；相反的；（判决等）撤销的

print list(reversed(['dream','a','have','I']))

#['I', 'have', 'a', 'dream']

**2、让人糊涂的sort()与sorted()**

在Python 中sorted是内建函数(BIF),而sort()是列表类型的内建函数list.sort()。

**sorted()**

sorted(iterable[, cmp[, key[, reverse]]])

Return a new sorted list from the items in iterable.

The optional arguments(可选参数) cmp, key, and reverse have the same meaning as those for the list.sort() method (described in section [Mutable Sequence Types](http://docs.python.org/2/library/stdtypes.html#typesseq-mutable)).

cmp specifies（指定） a custom comparison function of two arguments (iterable(可迭代的) elements) which should return a negative（复数）, zero or positive（正数） number depending on whether the first argument is considered smaller than, equal to, or larger than the second argument: cmp=lambda x,y: cmp(x.lower(), y.lower()). The default value is None.

key specifies a function of one argument that is used to extract a comparison key from each list element: key=str.lower. The default value is None (compare the elements directly).

reverse is a boolean value. If set to True, then the list elements are sorted as if each comparison were reversed.

#字符串排序使用是字典序，而非字母序

"""sorted()按照字典序排序"""

lis = ['a','c','z','E','T','C','b','A','Good','Tack']

print sorted(lis) #['A', 'C', 'E', 'Good', 'T', 'Tack', 'a', 'b', 'c', 'z']

关于字典序：

[可参考百度百科](http://baike.baidu.com/view/4670107.htm)。http://baike.baidu.com/view/4670107.htm

根据ASCII排，具体如下：  
0-9（对应数值48-59）；  
A-Z（对应数值65-90）；  
a-z（对应数值97-122）；

------------

标准序: 短在前,长在后,等长的依次比字母,  
如to < up < cap < cat < too < two <boat < boot < card  
字典序: 依次比字母,   
如boat < boot <cap < card < cat < to < too< two < up

更有甚者说字典序就是字典的排序，像字典一样。我一直没有找到权威的说明，什么是字典序？？？？求答案！！

**sort()**

s.sort([cmp[, key[, reverse]]])

三、Python的字典排序

**1、关于Python字典的一些特征**

无序：

字典，形如 dic = {'a':1 , 'b':2 , 'c': 3},字典中的元素没有顺序，所以dic[0]是有语法错误的。

无重：

不可以有重复的键值，所以 dic.add['c'] = 4后，字典变成 {'a':1 , 'b':2 , 'c': 4}.

**2、根据“键”或“键值”进行不同顺序的排序**

函数原型：sorted(dic,value,reverse)

**解释：**dic为比较函数，value 为排序的对象（这里指键或键值），

reverse：注明升序还是降序，True--降序，False--升序（默认）

**3、例子：**

dic = {'a':31, 'bc':5, 'c':3, 'asd':4, '33':56, 'd':0}  
想把dic的value按照从大到小排序（value都是整数）。

dic = {'a':31, 'bc':5, 'c':3, 'asd':4, '33':56, 'd':0}

print sorted(dic.iteritems(), key=lambda d:d[1], reverse = False )

#[('d', 0), ('c', 3), ('asd', 4), ('bc', 5), ('a', 31), ('33', 56)]

**解释如下：**

（1）、dic.iteritems()，返回字典键值对的元祖集合

[复制代码](javascript:void(0);)

print dic.iteritems() #<dictionary-itemiterator object at 0x00B71A80>

for obj in dic.iteritems():

print obj,obj[0],obj[1]

#('a', 31) a 31

#('c', 3) c 3

#('d', 0) d 0

#('bc', 5) bc 5

#('33', 56) 33 56

#('asd', 4) asd 4

[复制代码](javascript:void(0);)

**（2）、关于排序对象**

上述已经说过，value(或key)为排序的对象（这里指键或键值），然而为什么使用lambda函数呢，这里请参阅：**[点击阅读](http://www.cnblogs.com/BeginMan/p/3178103.html" \t "_blank)**

key=lambda d:d[1] 是将键值(value)作为排序对象。

key = lambda d:d[1]

for i in dic.iteritems():

print key(i), #输出31 3 0 5 56 4，这些都是字典dic的值

如果选择 key = lambda d:d[0]，则选择【键Key】作为排序对象。

**(3)、reverse**

reverse 是否反向，reverse=Ture表示反向。

**(4)、注意：**

sorted(dic.iteritems(), key=lambda d:d[1], reverse = False )将每一项dic.iteritems()键值对的元祖进行迭代，每一项都作为参数传入key()函数（我说的是这个：key=lambda d:d[1],）中。

**4、回顾**

lis = ['a','bc','c','asd','33','d']

print sorted(lis) #['33', 'a', 'asd', 'bc', 'c', 'd']

依次比字母, 如boat < boot <cap < card < cat < to < too< two < up

**5.问题**

具体实例可参考：[\*\*python的排序函数sort,sorted在列表排序和字典排序中的应用详解和举例\*\*](http://wangwei007.blog.51cto.com/68019/1100742)

现在有这种情况，排序中排序。如大题号排序，然后大题对应的小题号也排序，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | lis = [{'Big':3, 'small':2},{'Big':3, 'small':4},{'Big':2, 'small':2}, {'Big':3, 'small':1},{'Big':2, 'small':1},{'Big':1, 'small':1}]    # 大题号排序  li = sorted(lis, key=lambda s: s['Big'])    # 输出：  #[{'small': 1, 'Big': 1}, {'small': 2, 'Big': 2}, {'small': 1, 'Big': 2}, {'small': 2, 'Big': 3}, {'s  mall': 4, 'Big': 3}, {'small': 1, 'Big': 3}]    # 小题号排序：  sort\_ff = []  no = set([i['Big'] for i in li])  for obj in no:  li\_ = []  for i in ff:  if i['Big'] == obj:  li\_.append(i)  l = sorted(li\_, key=lambda s: s['small'])  for j in l:  sort\_ff.append(j)    # 输出结果：  [{'small': 1, 'Big': 1}, {'small': 1, 'Big': 2}, {'small': 2, 'Big': 2}, {'small': 1, 'Big': 3}, {'small': 2, 'Big': 3}, {'small': 4, 'Big': 3}] |

**善用sort() 或 sorted(), a.sort() 已改变其结构，b = a.sort() 是错误的写法! 而 sorted(a, ...)并没有改变a的结构。**

# [深入Python(2): \_\_init\_\_.py 用法](http://www.cnblogs.com/BeginMan/p/3183629.html)

原文：http://www.2cto.com/kf/201204/129388.html

python的每个模块的包中，都有一个\_\_init\_\_.py文件，有了这个文件，我们才能导入这个目录下的module。  
那么，\_\_init\_\_.py还有什么别的功能呢？  
其实，\_\_init\_\_.py里面还是可以有内容的，我们在导入一个包时，实际上导入了它的\_\_init\_\_.py文件。  
我们可以再\_\_init\_\_.py文件中再导入其他的包，或者模块。  
[python]  
import readers   
import writers   
import commands   
import users   
import meta   
import auth   
import admin

这样，当我们导入这个包的时候，\_\_init\_\_.py文件自动运行。帮我们导入了这么多个模块，我们就不需要将所有的import语句写在一个文件里了，也可以减少代码量。  
不需要一个个去导入module了。  
\_\_init\_\_.py 中还有一个重要的变量，叫做 \_\_all\_\_。我们有时会使出一招“全部导入”，也就是这样：  
from PackageName import \*  
这时 import 就会把注册在包 \_\_init\_\_.py 文件中 \_\_all\_\_ 列表中的子模块和子包导入到当前作用域中来。比如：  
#文件 \_\_init\_\_.py

\_\_all\_\_ = ["Module1", "Module2", "subPackage1", "subPackage2"]

如：在一个包里有foo.py、\_\_init\_\_.py

#\_\_init\_\_.py

import os

import datetime

#foo.py

from \_\_init\_\_ import \*

print datetime.datetime.now()

则输出：2013-07-11 11:34:41.250000

[**深入Python(3): and、or以及and-or**](http://www.cnblogs.com/BeginMan/p/3197123.html)

一、and:

在Python 中，and 和 or 执行布尔逻辑演算，如你所期待的一样，但是它们并不返回布尔值；而是，返回它们实际进行比较的值之一。

[复制代码](javascript:void(0);)

>>> 'a' and 'b'

'b'

>>> '' and 'b'

''

>>> 'a' and 'b' and 'c'

'c'

[复制代码](javascript:void(0);)

在布尔上下文中从左到右演算表达式的值，如果布尔上下文中的所有值都为真，那么 and 返回最后一个值。

如果布尔上下文中的某个值为假，则 and 返回第一个假值

二、or:

[复制代码](javascript:void(0);)

>>> 'a' or 'b'

'a'

>>> '' or 'b'

'b'

>>> '' or [] or {}

{}

>>> 0 or 'a' or 'c'

'a'

[复制代码](javascript:void(0);)

使用 or 时，在布尔上下文中从左到右演算值，就像 and 一样。如果有一个值为真，or 立刻返回该值

如果所有的值都为假，or 返回最后一个假值

注意 or 在布尔上下文中会一直进行表达式演算直到找到第一个真值，然后就会忽略剩余的比较值

三、and-or:

and-or 结合了前面的两种语法，推理即可。

[复制代码](javascript:void(0);)

>>> a='first'

>>> b='second'

>>> 1 and a or b

'first'

>>> (1 and a) or b

'first'

>>> 0 and a or b

'second'

>>> (0 and a) or b

'second'

>>>

[复制代码](javascript:void(0);)

这个语法看起来类似于 C 语言中的 *bool* ? a : b 表达式。整个表达式从左到右进行演算，所以先进行 and 表达式的演算。 1 and 'first' 演算值为 'first'，然后 'first' or 'second' 的演算值为 'first'。

0 and 'first' 演算值为 False，然后 0 or 'second' 演算值为 'second'。

and-or主要是用来模仿 三目运算符 bool?a:b的，即当表达式bool为真，则取a否则取b。

and-or 技巧，*bool* and a or b 表达式，当 a 在布尔上下文中的值为假时，不会像 C 语言表达式 *bool* ? a : b 那样工作。

四、安全使用and-or

[复制代码](javascript:void(0);)

>>> a=""

>>> b="second"

>>> (1 and [a] or [b])

['']

>>> (1 and [a] or [b])[0]

''

>>>

[复制代码](javascript:void(0);)

由于 [a] 是一个非空列表，所以它决不会为假。即使 a 是 0 或者 '' 或者其它假值，列表 [a] 也为真，因为它有一个元素。

一个负责的程序员应该将 and-or 技巧封装成一个函数：

def choose(bool,a,b):

return (bool and [a] or [b])[0]

print choose(1,'','second') #''

[**深入Python(4):深拷贝和浅拷贝**](http://www.cnblogs.com/BeginMan/p/3197649.html)

一、前奏：熟悉Python内存管理

在Python中，变量在第一次赋值时自动声明，在创建---也就是赋值的时候，解释器会根据语法和右侧的操作数来决定新对象的类型。

引用计数器：一个内部跟踪变量

引用计数：每一个对象各有多少个引用

当对象被创建并(将其引用)赋值给变量时，该对象的引用计数就被设置为 1

>>> x = 3.14

语句 x=3.14，创建一个浮点型对象并将其引用赋值给了x,x是第一个引用，该对象的引用计数为1

当一个对象(的引用)又被赋值到其他变量，或做参数传递等，该对象的一个新的引用(或叫别名)被创建，则该对象的引用计数自动+1。

以下都会增加引用计数：

y = x #做别名

foo(x) #做参数传递

mylis = [1,2,x,'a'] #成为容器对象的一个元素

以下都会减少引用计数：

[复制代码](javascript:void(0);)

del x #del显式销毁

bar = x

x = True #对象的一个别名被赋值给其他对象

mylis.remove(x) #对象被从窗口对象中移除

del mylis #窗口对象本身被销毁

[复制代码](javascript:void(0);)

二、Python的复制

从上面可见，**对象的赋值实际上是对象的引用**。当创建一个对象，然后把它赋给另一个变量的时候，python并没有拷贝这个对象，而只是拷贝了这个对象的引用。

当你对一个对象赋值的时候(做为参数传递,或者做为返回值),Python和Java一样,总是传递原始对象的引用,而不是一个副本。

[复制代码](javascript:void(0);)

"""传递原始对象的引用,而不是一个副本"""

a = [1,2,3]

b = a

b.append(100)

print b #[1, 2, 3, 100]

print a #[1, 2, 3, 100]

print id(a) #11530368

print id(b) #11530368

[复制代码](javascript:void(0);)

如 果你想修改一个对象,而且想让原始的对象不受影响,那你就需要对象复制。

可以 使用copy.copy(),它可以进行对象的浅复制(shallow copy),它复制了对象,但对于对象中的元素,依然使用引用.

(1)、使用切片[:]操作进行拷贝

(2)、使用工厂函数(如list/dir/set)等进行拷贝

(3)、copy.copy()

[复制代码](javascript:void(0);)

>>> jack = ['jack',['age',20]]

>>> tom = jack[:]

>>> anny = list(jack)

>>> jack

['jack', ['age', 20]]

>>> tom

['jack', ['age', 20]]

>>> anny

['jack', ['age', 20]]

>>> print id(jack),id(tom),id(anny)

13457088 18487376 18489136

[复制代码](javascript:void(0);)

接下来修改上面例子，对姓名和年级进行修改：

[复制代码](javascript:void(0);)

>>> tom[0]='tom'

>>> anny[0]='anny'

>>> print tom

['tom', ['age', 20]]

>>> print anny

['anny', ['age', 20]]

>>> anny[1][1]

20

>>> anny[1][1]= 18

>>> anny[1][1]

18

>>> print jack,tom,anny

['jack', ['age', 18]] ['tom', ['age', 18]] ['anny', ['age', 18]]

[复制代码](javascript:void(0);)

发现，虽然姓名都对号了，但是年龄却都变成了18.这是为什么呢？

我们看看它们元素的id

[复制代码](javascript:void(0);)

>>> [id(x) for x in jack]

[13463040, 13456608]

>>> [id(x) for x in tom]

[13463424, 13456608]

>>> [id(x) for x in anny]

[18501664, 13456608]

[复制代码](javascript:void(0);)

发现，其中列表中  姓名字符串  id都不一样，但是 年龄列表id却都相同。

这是因为：python中字符串不可以修改，所以在为tom和anny重新命名的时候，会重新创建一个’tom’和’anny’对象，替换旧的’jack’对象。

**这就说明了，浅复制(shallow copy),它复制了对象,但对于对象中的元素,依然使用引用.**

[复制代码](javascript:void(0);)

"""浅copy"""

import copy

aa = [1,2,3]

bb = copy.copy(aa)

print id(aa) #11533088

print id(bb) #12014776

bb[0] =100

print bb #[100, 2, 3]

print aa #[1,2,3]

#由于数字不可变，修改的时候会替换旧的对象

print [id(x) for x in bb] #[10247196, 10246388, 10246376]

print [id(y) for y in aa] #[10246400, 10246388, 10246376]

[复制代码](javascript:void(0);)

下面试试对象中可变元素：

[复制代码](javascript:void(0);)

lis = [['a'],[1,2],['z',23]]

copyLis = copy.copy(lis)

copyLis[1].append('bar')

print copyLis #[['a'], [1, 2, 'bar'], ['z', 23]]

print lis #[['a'], [1, 2, 'bar'], ['z', 23]]

[复制代码](javascript:void(0);)

**如果希望复制一个容器对象,以及它里面的所有元素(包含元素的子元素),使用copy.deepcopy,这个方法会消耗一些时间和空间,不过,如果你需要完全复制,这是唯一的方法.**

[复制代码](javascript:void(0);)

"""深copy"""

deepLis = copy.deepcopy(lis)

deepLis[1].append('foo')

print deepLis #[['a'], [1, 2,'foo'], ['z', 23]]

print lis #[['a'], [1, 2], ['z', 23]]

[复制代码](javascript:void(0);)

注意：

1、对于非容器类型(如数字、字符串、和其他‘原子’类型的对象)没有被拷贝一说。

2、如果元祖变量只包含原子类型对象，则不能深copy。

参考：http://blog.csdn.net/sharkw/article/details/1934090

http://www.01happy.com/python-shallow-copy-and-deep-copy/

[**深入Python(5):random与range**](http://www.cnblogs.com/BeginMan/p/3203267.html)

一、range()内建函数

range(*start*, *stop*[, *step*])

返回列表

[复制代码](javascript:void(0);)

>>> range(10)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(1, 11)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> range(0, 30, 5)

[0, 5, 10, 15, 20, 25]

>>> range(0, 10, 3)

[0, 3, 6, 9]

>>> range(0, -10, -1)

[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

>>> range(0)

[]

>>> range(1, 0)

[]

[复制代码](javascript:void(0);)

 下面一个实例实现冒泡排序：

[复制代码](javascript:void(0);)

"""实现冒泡排序"""

lis = [15,10,23,6,11,105,70,2,54,21,8]

for i in range(len(lis)-1,0,-1):

for j in range(0,i):

if lis[j]>lis[j+1]:

lis[j],lis[j+1] = lis[j+1],lis[j]

print lis #[2, 6, 8, 10, 11, 15, 21, 23, 54, 70, 105]

[复制代码](javascript:void(0);)

当然比起强大的Python内建函数，冒泡排序就显得臃肿了，可以使用sort()来实现上面的排序：

[复制代码](javascript:void(0);)

"""sort()的实现效果"""

lis2 = [15,10,23,6,11,105,70,2,54,21,8]

lis2.sort()

print lis2 #[2, 6, 8, 10, 11, 15, 21, 23, 54, 70, 105]

[复制代码](javascript:void(0);)

在我另一篇博文：**[深入Python(1): 字典排序 关于sort()、reversed()、sorted()、cmp()等](http://www.cnblogs.com/BeginMan/p/3193081.html)**，有详细的讲解。

关于range()这里不再累述，官网文档讲的很详细。

[**推荐阅读**](http://www.cnblogs.com/buro79xxd/archive/2011/05/23/2054493.html)：http://www.cnblogs.com/buro79xxd/archive/2011/05/23/2054493.html

二、random模块

Python中的random模块用于生成随机数。

这里不再啰嗦了，最好看看文档，不过这里**[推荐阅读](http://www.cnblogs.com/yd1227/archive/2011/03/18/1988015.html" \t "_blank)**：http://www.cnblogs.com/yd1227/archive/2011/03/18/1988015.html

这里做个小实验，关于range()和random的随机算命小程序：

[复制代码](javascript:void(0);)

"""如题，输入5项自己的兴趣爱好，然后随机抽取20下，统计随机出现次数的最大数，依据此作为最后结果"""

def result():

love\_list = [] #爱好列表

count = 5 #列表元素个数(同时作为循环计数)

while count>0:

love = raw\_input(u'请选择你5项喜欢的做的事情:(还剩%s项)'%count)

love\_list.append(love.decode('utf-8')) #解码

count-=1

random.shuffle(love\_list) #元素打乱，重新洗牌

print '\*\*\*\*随机统计，通过随机出现次数最多者为依据，结果在于人品\*\*\*\*'

i = 0 #循环计数

lis = [] #循环后兴趣爱好列表

while i<20:

s= random.choice(love\_list) #选取范围

lis.append(s)

i+=1

max=0 #列表元素出现最多的次数

what = u'' #列表元素出现最多的次数时对应的元素

for obj in lis:

if lis.count(obj)>max:

max = lis.count(obj)

what = obj

if max:

print '综上：你最适合:%s' %(what.decode('UTF-8').encode('GBK') ) #下一步：Python经典问题：中文乱码的研究

result()

[复制代码](javascript:void(0);)

今天人品很好，抽了个让我满意的答案：



三、感悟：关于学完就忘

相信很多人和我一样，对于知识的学习，尤其是一门编程语言，总是学完就忘，十分苦恼。

不过这里，通过我多次的跌倒总结如下，或许能够巩固知识：

(1)、好记性不如烂笔头，多写技术博客，多总结，贵在总结

(2)、烂笔头不如学而时习之，但是很多人不想时习之，因为大多数对于学过的东西就不再那么有好奇心了，但是如果转换一个观念，或许就能激发兴趣去复习。如：

　　(2.1):以待面试，给自己施压  
　　(2.2):尝试从中抽离没有挖掘的价值知识  
　　(2.3):以”入门--深入--精通“ 来划分阶段，实现阶段性学习与复习

(3)、最好能学以致用，结合到自己工作中，或者写个小demo

还记得”**一万小时定律**“吗？

要成为某个领域的专家，需要10000小时，按比例计算就是：如果每天工作四个小时，一周工作五天，那么成为一个领域的专家至少需要十年。这就是所谓的一万小时定律。



勤奋努力的学习加上自己不断总结、自省，一套好的学习方法，我想要不得1万个小时吧？

在天朝，最好不要等十年去造就一个伪专家！

标准库random实现了一个随机数生成器

[**深入Python(5):递归**](http://www.cnblogs.com/BeginMan/p/3223356.html)

如果函数包含了对其自身的调用，该函数就是递归的。如下：

[复制代码](javascript:void(0);)

"""非递归方式呈现"""

sum = 0

for obj in range(1,101):

sum+=obj

print sum

""""1+2+3+...+100"""

def foo(n):

if n>0:return n+foo(n-1)

if n<=0:return 0

print foo(100)

"""阶乘"""

def fac(n):

if n==0 or n==1:

return 1

else:

return n\*fac(n-1)

print fac(10)

[复制代码](javascript:void(0);)

注意：

如果在使用递归的时候，出现条件判断，一定要把所有可能出现的条件都进行相应的处理。例如累加，如果写成这样：

""""1+2+3+...+100"""

def foo(n):

if n>0:return n+foo(n-1)

print foo(100)

**则会出错，可能有人会说，我只要满足条件就执行，不满足就不执行，但是不要忘了，如果一个Python函数被设计成不返回任何东西，它会返回None。出错如下：**

Traceback (most recent call last):  
  File "<stdin>", line 1, in ?  
  File "<stdin>", line 2, in foo  
  File "<stdin>", line 2, in foo  
  ...  
  File "<stdin>", line 2, in foo  
TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'

这里插入一些关于递归的网方解释，因为我是从网上搜到的这些内容：  
（1）递归就是在过程或函数里调用自身；  
（2）在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

递归算法一般用于解决三类问题：  
（1）数据的定义是按递归定义的。（比如Fibonacci函数）  
（2）问题解法按递归算法实现。（回溯）  
（3）数据的结构形式是按递归定义的。（比如树的遍历，图的搜索）

递归的缺点：递归算法解题的运行效率较低。在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。

参考：http://www.cnblogs.com/balian/archive/2011/02/11/1951054.html