my\_abs()的函数定义保存为abstest.py文件了，那么，可以在该文件的当前目录下启动Python解释器，用**from abstest import my\_abs**来导入my\_abs()函数，注意abstest是文件名

**Python的函数返回多值其实就是返回一个tuple**

Python函数在定义的时候，默认参数L的值就被计算出来了，即[]，因为默认参数L也是一个变量，它指向对象[]，每次调用该函数，如果改变了L的内容，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。

**定义默认参数要牢记一点：默认参数必须指向不变对象！**

**可变参数**

**\*nums表示把nums这个list的所有元素作为可变参数传进去**

**关键字参数**

\*\*extra表示把extra这个dict的所有key-value用关键字参数传入到函数的\*\*kw参数，kw将获得一个dict，注意kw获得的dict是extra的一份拷贝，对kw的改动不会影响到函数外的extra。

**命名关键字参数**

关键字参数\*\*kw不同，命名关键字参数需要一个特殊分隔符\*，\*后面的参数被视为命名关键字参数

**必选参数、默认参数、可变参数、关键字参数和命名关键字参数**

def f1(a, b, c=0, \*args, \*\*kw):

print('a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw)

def f2(a, b, c=0, \*, d, \*\*kw):

print('a =', a, 'b =', b, 'c =', c, 'd =', d, 'kw =', kw)

>>> f1(1, 2)

a = 1 b = 2 c = 0 args = () kw = {}

>>> f1(1, 2, c=3)

a = 1 b = 2 c = 3 args = () kw = {}

>>> f1(1, 2, 3, 'a', 'b')

a = 1 b = 2 c = 3 args = ('a', 'b') kw = {}

>>> f1(1, 2, 3, 'a', 'b', x=99)

a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}

>>> f2(1, 2, d=99, ext=None)

a = 1 b = 2 c = 0 d = 99 kw = {'ext': None}

**递归函数**

递归函数的优点是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。

使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。

解决递归调用栈溢出的方法是通过**尾递归**优化，事实上尾递归和循环的效果是一样的，所以，把循环看成是一种特殊的尾递归函数也是可以的

**切片**

L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。即索引0，1，2，正好是3个元素。

如果第一个索引是0，还可以省略：

可迭代对象 方法是通过collections模块的Iterable类型判断：

from collections import Iterable

>>> isinstance('abc', Iterable) # str是否可迭代

True

>>> isinstance([1,2,3], Iterable) # list是否可迭代

True

>>> isinstance(123, Iterable) # 整数是否可迭代

False

#### 列表生成式

list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]可以用list(range(1, 11)) 生成

>>> list(range(1, 11))

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

>>> [x \* x for x in range(1, 11)]

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

>>> [x \* x for x in range(1, 11) if x % 2 == 0]

[4, 16, 36, 64, 100]

>>> [m + n for m in 'ABC' for n in 'XYZ']

['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']