迭代器：

1.迭代器只能进行一次迭代，迭代对象可重复迭代。

2.所有的可迭代对象可以通过iter()函数变成迭代器。

3.迭代器的特点是可以通过next()内置函数或者调用自身的\_\_next\_\_()方法来获取元素，当元素获取完则会抛出StopIteration异常。所以迭代器的元素只能使用一次，在迭代之后元素会被销毁，这也正是迭代器的一大魅力。

生成器函数：generator

生成器函数中return后面如果有值，那么这个值为StopIteration异常的说明，不是程序的返回值

Python3里面的range是一个类  
Python2里面的range是列表  
列表可以用iter，range不可以

所以python3要先转化 list(range(5))

>>> x = iter([0,1,2])

>>> for i in x: #迭代器只能进行一次迭代

print (i)

0

1

2

>>> for i in x: #第二次则无效

print (i)

>>>

通过next()内置函数、\_\_next\_\_()方法来依次访问里面的元素

>>> x.\_\_next\_\_()

0

>>> next(x)

1

>>> x.\_\_next\_\_()

2

>>> next(x)

Traceback (most recent call last):

File "<pyshell#70>", line 1, in <module>

x.\_\_next\_\_()

StopIteration

列表生成式:

列表推导式

>>> [x for x in range(10)]

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

生成器表达式：通列表解析语法，把列表解析的[]换成（），返回迭代器

>>> l = (x\*\*2 for x in range(3))

>>> l

<generator object <genexpr> at 0x04224B70> #生成器函数：generator

>>> l.\_\_next\_\_()

0

>>> x.\_\_next\_\_()

1

>>> x.\_\_next\_\_()

4

Traceback (most recent call last):

File "<pyshell#70>", line 1, in <module>

x.\_\_next\_\_()

StopIteration

在函数中如果出现了yield关键字，那么这个函数就是生成器函数

生成器函数里面在没有return时会执行到没有值时返回的StopIteration的异常。

return后面是StopIteration异常的说明。

生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个迭代器。

>>> def l():

a = 1

while a<4:

yield a

a +=1

return '异常'

>>> l

<function l at 0x03BD01E0> #function函数

>>> l()

<generator object l at 0x03BC0AE0> #generator生成器函数

>>> a = l()

>>> next(a)

1

>>> next(a)

2

>>> next(a)

3

>>> next(a)

Traceback (most recent call last):

File "<pyshell#94>", line 1, in <module>

a.\_\_next\_\_()

StopIteration 异常

生成器支持的方法

close() 手动关闭生成器函数，后面的调用会直接返回StopIteration异常

send() 生成器函数最大的特点是可以接受外部传入的一个变量，

并根据变量内容计算结果后返回

首先要通过g.send(None)或者next启动生成器函数，

并执行到第一个yield语句结束的位置

throw() 用来向生成器函数送入一个异常，可以结束系统定义的异常，或自定义的异常