

Iterator and Generator

迭代器

生成器

Iterables

元组,字典,列表,字符串,集合 是可迭代对象,可以用for循环遍历其中的元素
使用Python内置的 iter() 函数可以根据上述对象创建迭代器(Iterator)

a=[1,2,3,4,5]

a_iterator=iter(a)

对迭代器执行 next() 即可进行遍历.

next(a_iterator) >>> 1

没有剩余的元素时 调用next()会报错.

next(a_iterator) >>> 2

对迭代器执行for循环时,相当于不断调用next()直至没有元素

next(a_iterator) >>> 3

因此如果提前调用了next(),则for循环会从剩余部分开始执行

...

调用for循环遍历时 实际上进行的就是上面的过程.

一些函数实际上也返回迭代器

reversed(sequence) 返回 sequence 反序之后对应的迭代器.

reversed() 仅记住传入时的序列长度,超出长度就不管了.

a=[1,2,3]

输出结果为 3 2 4

b=reversed(a)

迭代器在跟踪列表

a[0]=4;a.append(5)

for _ in b:

print(_,end='')

zip(*iterables) 返回由各个输入对象按序组成的元组组成的迭代器.

a,b=[1,2,3],[4,5,6]

输出为 (1,4) (2,5) (3,6)

z=zip(a,b)

当然也可以写 for x,y in zip(a,b): ...

for _ in z:

print(_,end='')

zip() 停在最短元素对应的位置

如果 a=[1,2,3,4,5] 结果仍不变.

map(func, iterable) 与 [func(_) for _ in iterable] 类似

filter(func, iterable) 与 [_ for _ in iterable if func(_)] 类似

意为“过滤”

Generators

现在我们来看看生成器.

"A generator is a type of iterator which yields results from a generator function."

一个简单的例子如下.

```
def evens():
    num=0
    while num<10:
        yield num
        num+=2
evengen=evens()
```

此时的 evengen 就是一个 generator
每对 evengen 调用一次 next(), 便会执行直至第一次执行完 yield 语句
然后程序暂停, 直至下次调用 next()

关于 yield 还有一个写法 for item in a: 可以简写为 yield from a
 yield item

即 yield from 中包含了一个 for 循环