第八章　迭代器与解析表达式

本章导读

迭代器是Python语言的重要特征。

学习目标：

1. 掌握Python中迭代器的使用；
2. 掌握生成器的使用；

本章目录

第一节　快速了解

1、可迭代对象

for-in是Python的重要循环语句，常用于list、tuple、string、range等序列以及dict、set等。for-in的使用前提必须是可迭代对象，也就是说list、tuple等都是可迭代对象，例程8-1的Odd是自定义类，由于其中定义了__iter__()和__next__()方法，其实例化对象为可迭代对象。

例程8-1

```
第1行  #!/usr/bin/python
第2行
第3行  class Odd():
第4行      def __init__(self):
第5行          self.__nowNum=1
第6行      def __iter__(self):
第7行          return self
第8行
第9行      def __next__(self):
第10行         if self.__nowNum>100:
第11行             raise StopIteration
第12行
第13行         rtnVal=self.__nowNum
第14行         self.__nowNum+=2
第15行         return rtnVal
第16行
第17行 for i in Odd():
第18行     print(i,end="\t")#输出：1-100之间的奇数
第19行
第20行 print(Odd().__next__())#输出：1
第21行 print(Odd().__next__())#输出：1
第22行
第23行 myOdd=Odd()
第24行 print(myOdd.__next__())#输出：1
第25行 print(myOdd.__next__())#输出：3
第26行
```

```
for i in myOdd:
    print(i,end="\t")#输出：5-99之间的奇数

#print(myOdd.__next__())#输出：跑输异常，StopIteration

iterOdd=iter(Odd())
print(type(iterOdd))#输出：<class '__main__.Odd'>

for i in iterOdd:
    print(i,end="\t")#输出：1到100之间的奇数

#eof
```

判断一个对象是否可以迭代，可用例程8-2所示代码。

例程8-2

```
#!/usr/bin/python

from collections import Iterable
print(isinstance('123',Iterable))

#eof
```

2、生成器

例程8-1的功能是生成奇数序列，功能并不复杂，但代码量大，Python提供了一种使用yield的替代解决方案，其所在函数常被成为生成器，如例程8-3所示。观察第9-10行会发现，函数Test()能使用for-in循环，虽然没有__next__()方法，但也能在第13-15行使用，且遍历到终点后，能抛出异常StopIteration(如第16行所示，为了能让其后代码执行，此处注释掉)。如同第18、19行所示，iter()函数同样能作用于Test()。值得注意：type(Test)其返回值为<class 'function'>，而type(Test())的返回值为<class 'generator'>。

例程8-3

```
#探索yield使用

def Test():
    yield 1
    yield 3
    yield 5

myTest=Test()
for i in myTest:
    print(i,end="\t")#输出：1 3 5

hisTest=Test()
print(hisTest.__next__())#输出：1
print(hisTest.__next__())#输出：3
print(hisTest.__next__())#输出：5
#print(hisTest.__next__())#输出：StopIteration

iterTest=iter(Test())
print(next(iterTest))#输出：1

print(type(Test))#输出：<class 'function'>
print(type(Test()))#输出：<class 'generator'>

```

```
print(type(iterTest))#输出：<class 'generator'>
第24行
第25行 #eof
```

如前所述，yield所在函数为生成器(generator)。当函数执行时，当第1次遇到yield时，即将yield后的表达式作为函数的返回值。第2次调用时，从上一次yield执行后的位置开始执行，并根据上一次的场景(如：变量值等等)得到第2个yield后表达式的值。依次类推，当所有yield都执行完毕后，自动抛出StopIteration。另外，如第21行所示，函数名Test的类型依然是functon，而Test()则是generator(生成器)，如第22行所示。

例程8-4和例程8-5的功能都是生成质数列表，分别采用class和yield方式，可以明显看出，yield生成器模式更简洁。注意：例程8-5没有raise StopIteration，而例程8-4必须有raise StopIteration。对于生成器，yield关键字是关键。在Python语言中，当一个yield执行完毕后，该函数将保持当时执行环境，包括：局部变量、指令指针、内部堆栈、异常处理等等。当再次执行时，将从上一个yield后开始执行。

例程8-4

```
第1行 #!/usr/bin/python
第2行 #!/usr/bin/python
第3行
第4行 class Prime():
第5行     def __init__(self,N):
第6行         self.__lastNum=N
第7行         self.__nowNum=2
第8行     def __iter__(self):
第9行         return self
第10行    def __next__(self):
第11行        for i in range(self.__nowNum,self.__lastNum+1):
第12行            for j in range(2,i//2+1):
第13行                if i%j==0:
第14行                    break
第15行            else:
第16行                self.__nowNum=i+1
第17行                return i
第18行        raise StopIteration
第19行
第20行 for i in Prime(100):
第21行     print(i,end="\t")
第22行
第23行 #eof
```

对于例程8-5，for-in循环到2到N+1之间(第5行代码)，当循环完毕后，将自动执行隐含的raise StopIteration。在for-in循环体内，如果外层循环变量i(第5行代码)能被内层循环变量n(第六行代码)整除，则表明不是质数，中断内层循环，外层循环继续。如果不能被内层循环的所有数整除，则执行第8行代码，执行其中的yield i，该数由于不能被内层循环的所有数整除，因此是质数。

例程8-5

```
第1行 #!/usr/bin/python
第2行
第3行 def Prime(N):
第4行     if N<2:raise StopIteration
第5行     for i in range(2,N+1):
第6行         for n in range(2,i//2+1):
第7行             if i%n==0:break;
第8行         else:yield i
第9行
第10行 for i in Prime(97):
第11行     print(i,end="\t")
```

```
第12行
第13行  #eof
```

第二节　itertools库

itertools是Python标准库，其简单用法如例程8-6所示，使用itertools的前提是import该库，如第3行代码所示。itertools.count()是itertools库中的一个方法(第8-9行)，能产生无限多个以第一个参数为起点第二个参数为每次增长量(步长)的数，如果省略第二个参数，则默认为1。例程第5-6行代码range()也能产生以某个数为起点，每次间隔一定量，但必须有终点。和itertools.count()相比，更大的不同是，itertools.count()每次临时生成需要量，而range()是一次性产生一个序列保存在内存之中，占据相应内存空间。itertools中的方法都有类似特点。

例程8-6

```
第1行  #!/usr/bin/python
第2行
第3行  import itertools
第4行
第5行  for i in range(5,2000,5):
第6行      print(i,end="\t")
第7行
第8行  for i in itertools.count(5,5):
第9行      print(i,end="\t")
第10行
第11行 #eof
```

表8-1：itertools

Iterator	描述	Results	示例
count(start[,step])	无限计数，start是起点，step是不长，可省略，省略则默认为1	start+0*step, start+1*step, start+2*step, ...	count(10)-->10 11 12 13 14 ...
cycle(p)	p无限循环	p0, p1, ... plast, p0, p1, ...	cycle('ABCD')-->A B C D A B C D...
repeat(elem[,n])	如省略n则无限循环elem，如指定n值，则循环n次	elem, elem, elem, ... endlessly or up to n times	repeat(10,3)-->10 10 10
chain(p,q[,r,...])	将两个或多个组合在一起	p0, p1, ... plast, q0, q1, ...	chain('ABC', 'DEF') -->A B C D E F
compress(d,s)	根据s序列压缩d序列	(d[0] if s[0]), (d[1] if s[1]), ...	compress('ABCDEF',[1,0,1,0,1,1]) --> A C I
dropwhile(pred,seq)	根据pred删减seq序列，返回第一个及其后满足条件的子序列	seq[n], seq[n+1], starting when pred fails	dropwhile(lambda x: x<5, [1,4,6,4,1]) -->
groupby(iter[, keyFun])	按keyFun对iter数据排序	sub-iterators grouped by value of keyfunc(v)	如例程8-7第3-13行代码所示。使用groupby前，需对数据排序。
ifilter(pred,seq)	按pred返回值为True对seq过滤	elements of seq where pred(elem) is true	ifilter(lambda x: x%2, range(10)) --> 1 3 5
ifilterfalse(pred, seq)	按pred返回值为False对seq过滤	elements of seq where pred(elem) is false	ifilterfalse(lambda x: x%2, range(10)) --> 6 8
islice(seq, [start,] stop [, step])	切片	elements from seq[start:stop:step]	islice('ABCDEFG', 2, None) --> C D E F G
imap(func, p, q, ...)	依次执行func(p0,q0),func(p1,q1)，……	func(p0, q0), func(p1, q1), ...	imap(pow, (2,3,10), (5,2,3)) --> 32 9 1000

Iterator	描述	Results	示例
starmap(Fx, seq)	形如：Fx(*seq[0]), func(*seq[1]), ...	Fx(*seq[0]),Fx(*seq[1]), ...	starmap(pow, [(2,5), (3,2), (10,3)]) #out: 3 1000
tee(iter, n)	复制n份iter	iter1, iter2, ... itern	
takewhile(Fx,Seq)	获取子序列，知道Fx值为False	seq[0], seq[1], until pred fails	takewhile(lambda x: x<5, [1,4,6,4,1]) --> 1
izip(p, q, ...)	对应组合	(p[0], q[0]), (p[1], q[1]), ...	izip('ABCD', 'xy') --> Ax By
izip_longest(p, q, ...,fillvalue=' ')	对应组合，如不足，填入fillvalue	(p[0], q[0]), (p[1], q[1]), ...	izip_longest('ABCD', 'xy', fillvalue='-') --> C- D-
product(p, q, ... [repeat=1])		cartesian product, equivalent to a nested for-loop	product('ABCD',repeat=2)#out: AA AB AC BA BB BC BD CA CB CC CD DA DB DC DD
permutations(p[, r])	排列，如省略r则默认为2	r-length tuples, all possible orderings, no repeated elements	permutations('ABCD',2)#out: AB AC AD B BD CA CB CD DA DB DC
combinations(p[,r])	组合，如省略r默认为2	r-length tuples, in sorted order, no repeated elements	combinations('ABCD',2)#out: AB AC AD B CD
combinations_with_replacement(p, r)		r-length tuples, in sorted order, with repeated elements	combinations_with_replacement('ABCD',2 AA AB AC AD BB BC BD CC CD DD

例程8-7

```
第1行  #!/usr/bin/python
第2行
第3行  import itertools
第4行  def score_level(Score):
第5行      if Score>=85:return "BEST"
第6行      if Score<85 and Score>=75:return "BETTER"
第7行      if Score<75 and Score>=60:return "GOOD"
第8行      if Score<60:return "BAD"
第9行  scoreList=[87,76,67,56,98,84,99,78,67,89]
第10行 scoreList=sorted(scoreList)
第11行 for i,m in itertools.groupby(scoreList,key=score_level):
第12行     print(i,list(m),end=" ",sep="=")
第13行 #上行输出：BAD=[56] GOOD=[67, 67] BETTER=[76, 78, 84] BEST=[87, 89, 98, 99]
第14行
第15行 #eof
```

第三节　小结