1.切片

2.迭代

可以看出，Python的for循环抽象程度要高于C的for循环，因为Python的for循环不仅可以用在list或tuple上，还可以作用在其他可迭代对象上。

所以，当我们使用for循环时，只要作用于一个可迭代对象，for循环就可以正常运行，而我们不太关心该对象究竟是list还是其他数据类型。

如何判断一个对象是可迭代对象呢？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

默认情况下，dict迭代的是key。如果要迭代value，可以用for value in d.values()，如果要同时迭代key和value，可以用for k, v in d.items()。

如果要对list实现类似Java那样的下标循环怎么办？Python内置的enumerate函数可以把一个list变成索引-元素对，这样就可以在for循环中同时迭代索引和元素本身：

>>> **for** i, value **in** enumerate(['A', 'B', 'C']):  
... print(i, value)  
...  
0 A  
1 B  
2 C

3.列表生成式

>>> [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]

还可以使用两层循环，可以生成全排列：

>>> [m + n **for** m **in** 'ABC' **for** n **in** 'XYZ']  
['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']

>>> d = {'x': 'A', 'y': 'B', 'z': 'C' }  
>>> [k + '=' + v **for** k, v **in** d.items()]  
['y=B', 'x=A', 'z=C']

最后把一个list中所有的字符串变成小写：

>>> L = ['Hello', 'World', 'IBM', 'Apple']  
>>> [s.lower() **for** s **in** L]  
['hello', 'world', 'ibm', 'apple']

4.生成器

在Python中，这种一边循环一边计算的机制，称为生成器：generator。

要创建一个generator，有很多种方法。第一种方法很简单，只要把一个列表生成式的[]改成()，就创建了一个generator：

>>> L = [x \* x **for** x **in** range(10)]  
>>> L  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
>>> g = (x \* x **for** x **in** range(10))  
>>> g  
<generator object <genexpr> at 0x1022ef630>

如果要一个一个打印出来，可以通过next()函数获得generator的下一个返回值：

正确的方法是使用for循环，因为generator也是可迭代对象：

>>> g = (x \* x **for** x **in** range(10))  
>>> **for** n **in** g:  
... print(n)  
...   
0  
1  
4  
9  
16  
25  
36  
49  
64  
81

定义generator的另一种方法。如果一个函数定义中包含yield关键字，那么这个函数就不再是一个普通函数，而是一个generator：

**def fib**(max):  
 n, a, b = 0, 0, 1  
 **while** n < max:  
 **yield** b  
 a, b = b, a + b  
 n = n + 1  
 **return** 'done'

这里，最难理解的就是generator和函数的执行流程不一样。函数是顺序执行，遇到return语句或者最后一行函数语句就返回。而变成generator的函数，在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行。

5.迭代器

可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator。

Python的Iterator对象表示的是一个数据流，Iterator对象可以被next()函数调用并不断返回下一个数据，直到没有数据时抛出StopIteration错误。可以把这个数据流看做是一个有序序列，但我们却不能提前知道序列的长度，只能不断通过next()函数实现按需计算下一个数据，所以Iterator的计算是惰性的，只有在需要返回下一个数据时它才会计算。

集合数据类型如list、dict、str等是Iterable但不是Iterator，不过可以通过iter()函数获得一个Iterator对象。

例

*# 首先获得Iterator对象:*  
it = iter([1, 2, 3, 4, 5])  
*# 循环:*  
**while** True:  
 **try**:  
 *# 获得下一个值:*  
 x = next(it)  
 **except** StopIteration:  
 *# 遇到StopIteration就退出循环*  
 **break**