**Python的可迭代对象（iterable）**

有\_\_iter\_\_方法的对象，都是可迭代对象.可迭代对象的意思通过调用\_\_iter\_\_方法,他们可以每次循环返回一个元素。

Python内置有以下6种可迭代的对象：Python内置str、list、tuple、dict、set、file都是可迭代对象。可迭代的对象执行\_\_iter\_\_方法得到的返回值是迭代器对象。

可迭代对象并不总是内存友好的，例如可迭代对象的list类型，它会在内存空间中直接开辟足够容纳所有数据的空间，而同为可迭代对象的迭代器(iterator)，对内存空间是友好的。

可迭代对象并不总能用下标index访问。例如可迭代对象list, str, tuple, set类型，他们可以用下标index进行访问。而同为可迭代对象的迭代器，生成器， 或者Python内置类型字典，是不能用下标进行访问的。

**Python的迭代器(iterator)对象**

迭代器对象指的是即内置有\_\_iter\_\_又内置有\_\_next\_\_方法的对象。迭代器有一个特点就是可以被next()函数调用并不断返回下一个值的对象，并且每次只能迭代它们一次。原因很简单,因为它们不是全部存在内存里,它们只在要调用的时候在内存里生成。

迭代器对象由于一定还有\_\_iter\_\_方法,所以其一定是可迭代对象。而可迭代对象只能保证其中含有\_\_iter\_\_方法，并不一定含有\_\_next\_\_方法，所以可迭代对象不一定是迭代器对象。

**Python的生成器(Generator)**

**Python的生成器**本质是带有yield关键字的函数。在函数中使用 yield关键字，则函数执行到yield时，将跳出函数返回元素. 当跳出函数返回元素时候，yield关键字之前的状态都会被保存（相比之下return会结束该方法不保存任何状态）。当下次再次调用时，将从yield所在地方开始。Python内置函数range()便是如此。

例如自定义一个生成器:

Python内置函数range(x,y)则表达包括x开始但是不包括y.我们可以自定义一个生成器，使其包括y:

def custom\_range(start, end):

starting\_point = start  
 while end >= starting\_point:  
 yield starting\_point  
 starting\_point += 1  
  
  
generator = custom\_range(5, 10)  
for i in generator:  
 print(i)

Note：虽然上面的方法没有定义\_\_iter\_\_以及\_\_next\_\_方法。但是其方法内使用yield关键字，是生成器，所以也是迭代器。

**关于迭代器和和生成器的区别**

生成器是一种特殊的迭代器，生成器实现了迭代器协议\_\_iter\_\_以及\_\_next\_\_.生成器是可以改变迭代的值的，然而迭代器随意改值会有问题.

**关于迭代器和list的区别**

list直接把所有数据加载到内存， 而迭代器是一个一个取值，在需要下一个值的时候才回去计算取出这个值到内存（可以把迭代器想象成一个生成器的代码，一次next，运行代码，然后返回值）

**For…in… 语句使用：**

For…in…语句的语法要求为for… in … <iterable对象>. 要求对一个iterable对象进行遍历。由于是iterable对象，所以可以为Python内置的str、list、tuple、dict、set、file， 也可以为任意的迭代器对象（因为所有的迭代器对象都有\_\_iter\_\_方法都是可迭代对象）， 也可以为任意的生成器。因为生成器必定为迭代器，也就必定为可迭代对象。