■str是不可变对象，list是可变对象

# str是不变对象  
a = 'abc'  
b = a.replace('a', 'A')  
print(b)  
# 输出：Abc  
print(a)  
# 输出： abc  
  
# list是可变对象  
c = [2, 3, 1]  
print(c)  
# 输出： [2, 3, 1]  
c.sort()  
print(c)  
# 输出：[1, 2, 3]

■数组list

# list就是数组  
a = [1, 'b', 3, 'd', 'ee']  
  
# len()方法返回数组的长度  
print(len(a))  
# 输出： 5  
  
# 通过索引访问数组中的元素，-1代表最后一位  
print(a[-1])  
# 输出： ee  
  
# 以此类推，倒数第二个是-2，倒数第三个是-3  
print(a[-2])  
# 输出： d  
print(a[-3])  
# 输出： 3

数组的方法

a = [1, 'b', 3, 'd', 'ee']  
  
# append()方法在数组末尾追加元素  
a.append(6)  
print(a)  
# 输出：[1, 'b', 3, 'd', 'ee', 6]  
  
# insert()方法在数组指定位置插入元素  
a.insert(2, 'haha')  
print(a)  
# 输出： [1, 'b', 'haha', 3, 'd', 'ee', 6]  
  
# pop()在数组末尾删除一个元素  
a.pop()  
print(a)  
# 输出：[1, 'b', 'haha', 3, 'd', 'ee']  
  
# pop()输入索引参数，在索引位置删除一个元素  
a.pop(3)  
print(a)  
# 输出： [1, 'b', 'haha', 'd', 'ee']  
  
# 可以把指定位置的元素直接赋值  
a[1] = 'cc'  
print(a)  
# 输出： [1, 'cc', 'haha', 'd', 'ee']  
  
# 嵌套数组  
b = [1, 2, [3, 4, 5]]  
print(b[2][0])  
# 输出： 3  
  
# 空数组的长度是0  
c = []  
print(len(c))  
# 输出： 0

■元组tuple

# 元组具有不可变性，它也没有append()，insert()这样的方法  
a = ('a', 'b', 3)  
print(a)  
  
# 如果元祖里只有一个元素1，则需要在1后面加逗号，因为如果不加逗号，系统会识别为是括号1  
b = (1,)  
print(b)  
  
# 表面上看，tuple的元素确实变了，但其实变的不是tuple的元素，而是list的元素。  
# tuple一开始指向的list并没有改成别的list，所以，tuple所谓的“不变”是说，tuple的每个元素，指向永远不变。  
# 即指向'a'，就不能改成指向'b'，指向一个list，就不能改成指向其他对象，但指向的这个list本身是可变的！  
# 理解了“指向不变”后，要创建一个内容也不变的tuple怎么做？那就必须保证tuple的每一个元素本身也不能变。  
c = (1, 2, [3, 4])  
c[2][1] = 5  
print(c)  
# 输出： (1, 2, [3, 5])

■切片

# slice切片  
a = [1, 2, 3, 4, 5, 6, 7]  
  
# 从第一位（索引0）取到第五位（索引4）  
print(a[0:5])  
# 输出：[1, 2, 3, 4, 5]  
  
# 从第一位取可以省略0  
print(a[:5])  
# 输出：[1, 2, 3, 4, 5]  
  
# 取最后一位用-1  
print(a[-1])  
# 输出： 7  
  
# 从头取到尾（应该是a[0:-1]这里省略0和-1）  
print(a[:])  
# 输出：[1, 2, 3, 4, 5, 6]  
  
# 最后一位-1也可以省略，如取倒数三个数  
print(a[-3:])  
# 输出： [5, 6, 7]  
  
# 取所有元素，每2个取一个  
print(a[::2])  
# 输出：[1, 3, 5, 7]  
  
# 取偶数  
print(a[1::2])  
# 输出：[2, 4, 6]

■列表生成式

# 列表生成式  
# 要生成list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]可以用list(range(1, 11))：  
print(list(range(1, 11)))  
# 输出： [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
# 但如果要生成[1x1, 2x2, 3x3, ..., 10x10]怎么做？方法一是循环：  
L = []  
for x in range(1, 11):  
 L.append(x \* x)  
print(L)  
# 输出： [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]  
  
# 而列表生成式则可以用一行语句代替循环生成上面的list：  
# 写列表生成式时，把要生成的元素x \* x放到前面，后面跟for循环，就可以把list创建出来  
print([x \* x for x in range(1, 11)])  
# 输出： [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]  
  
# 最后把一个list中所有的字符串变成小写：  
L1 = ['Hello', 'World', 'IBM', 'Apple']  
print( [s.lower() for s in L1])  
# 输出： ['hello', 'world', 'ibm', 'apple']

■生成器generator

通过列表生成式，我们可以直接创建一个列表。但是，受到内存限制，列表容量肯定是有限的。而且，创建一个包含100万个元素的列表，不仅占用很大的存储空间，如果我们仅仅需要访问前面几个元素，那后面绝大多数元素占用的空间都白白浪费了。

所以，如果列表元素可以按照某种算法推算出来，那我们是否可以在循环的过程中不断推算出后续的元素呢？这样就不必创建完整的list，从而节省大量的空间。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：  
  
# 使用列表生成式  
L = [x \* x for x in range(10)]  
print(L)  
# 输出： [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
  
# 使用生成器generator  
g = (x \* x for x in range(10))  
print(g)  
# 输出： <generator object <genexpr> at 0x0000023E04A35570>  
# g是一个generator对象，可以用for来遍历  
for x in g:  
 print(x)  
输出：  
# 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'  
  
  
f = fib(6)  
print(f)  
# 输出： <generator object fib at 0x000002C9FA7D5570>

■可迭代对象Iterable和 迭代器Iterator

# 我们已经知道，可以直接作用于for循环的数据类型有以下几种：  
# 一类是集合数据类型，如list、tuple、dict、set、str等；  
# 一类是generator，包括生成器和带yield的generator function。  
# 这些可以直接作用于for循环的对象统称为可迭代对象：Iterable。  
# 可以使用isinstance()判断一个对象是否是Iterable对象：  
from collections import Iterable  
print(isinstance([], Iterable))  
# 输出： True  
  
# 而生成器不但可以作用于for循环，还可以被next()函数不断调用并返回下一个值，  
# 直到最后抛出StopIteration错误表示无法继续返回下一个值了。  
# 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。  
# 可以使用isinstance()判断一个对象是否是Iterator对象：  
from collections import Iterator  
print(isinstance((x for x in range(10)), Iterator))  
# 输出： True