第3章 列表和元组

刘　卉

huiliu@fudan.edu.cn

3.0 序列

一系列相关的连续值(对象), 按顺序排列.

序列成员称为“元素”.

程序设计中常用的数据存储方式

- 几乎每种程序设计语言都提供了列表数据结构, 如: C和Basic的一维、多维数组.

Python提供的序列类型在所有程序设计语言中最丰富, 最灵活, 功能最强大.

- 常用的序列结构: 字符串, 列表, 元组, 字典*, 集合*, range对象……

	可变	不可变
有序	列表	元组, 字符串
无序	字典, 集合	

  - 有序序列(列表/元组/字符串)支持双向索引:

    正向索引: 第一个元素的下标为0, 第二个元素的下标为1, ……;

    反向索引: 最后一个元素的下标为-1, 倒数第二个元素的下标为-2, ……

```
>>> s='hello'
>>> s[0]
'h'
>>> s[-1]
'o'
```

* 在有些表述中, 字典和集合不属于序列.

3.1 列表: Python的"苦力"

3.1.1 创建与删除

3.1.2 元素访问与计数

3.1.3 成员资格判断

- for循环

3.1.4 元素的增加

3.1.5 元素的删除

3.1.6 切片操作

3.1.7 排序

3.1.8 用于序列操作的常用内置函数

3.1.9 列表推导式

列表简介

- Python内置的可变序列, 若干元素的有序集合.
  - 所有元素放在一对“[]”中, 元素之间以“,”分隔
  - 各元素的类型可相同/不同: 整数/实数/字符串, 列表/元组/字典/集合, 其它自定义类型的对象……

    e.g. [10, 20, 30, 40]

    ['crunchy frog', 'ram bladder', 'lark vomit']

    ['spam', 2.0, 5, [10, 20]]

    [['file1', 200, 7], ['file2', 260, 9]]

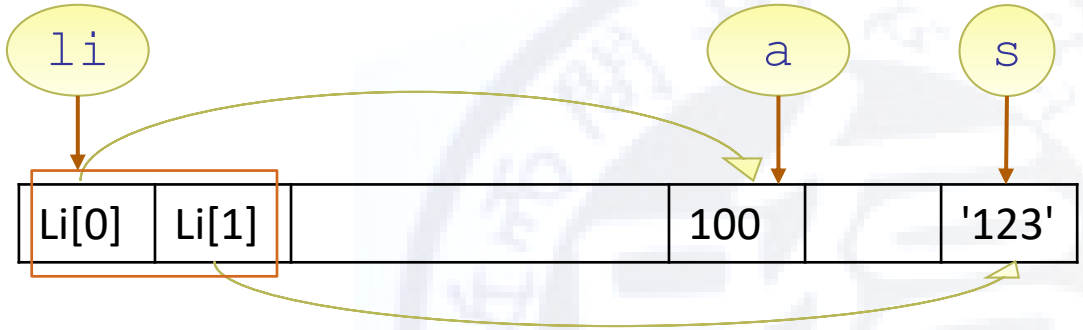
列表元素的访问——列表名[元素索引]

- 列表是有序序列⇨通过下标访问列表中某个元素.
- 正向下标: 0, 1, ……; 反向下标: -1, -2, ……
- 下标越界→程序报错.

```
>>> aList = [10, 20, 30, 40]
>>> aList[0]
10
>>> aList[4]
Traceback (most recent call
last):
  File "<pyshell#48>", line 1,
in <module>
    aList[4]
IndexError:  list index out
of range
```

```
# 二维数组的访问:
>>> aList = [[1, 2, 3], [4,
5, 6]]
>>> aList[0]
[1, 2, 3]
>>> aList[0][0]
1
>>> aList[1][2]
6
```

列表在内存中的存放




```
>>> li=[100, "123"]
>>> id(li)
30558248
>>> id(li[0])
1486708544
>>> id(li[1])
31144288
```

- 列表元素*vs*元素值: 列表元素为元素值的引用, 而非元素值本身.
- 增加/删除列表元素时, 列表自动扩展/收缩, 保证元素在内存中连续存放, 但元素值并非连续存放.

  e.g. li[0]和li[1]连续存放, 但100和"123"并不在一起.

+ 延伸阅读: Python中list的实现 Copy&Run
http: //www.jianshu.com/p/J4U6rR

列表的修改

- 修改变量值时，并非直接修改变量的值，而是使变量指向新的值→同样适用于列表.

```
>>> a = [1, 2, 3]
>>> id(a)
20230752
>>> a = [1, 2]   # a指向新的列表
>>> id(a)
20338208
```

列表元素的修改

1) 通过下标修改列表元素的值.

2) 通过列表自身提供的方法增加/删除元素.

- 列表在内存中的起始地址不变, 但元素地址发生变化.

```
>>> li = [100, "123"]
>>> id(li)
33277432
>>> id(li[0])
1503813440
>>> li[0]=200
>>> li
[200, '123']
>>> id(li)
33277432
>>> id(li[0])
1503815040
```

列表方法(操作)	说 明(lst为列表名,所有操作均针对lst进行)
lst.append(x)	将元素x添加至列表lst尾部(原地添加)
lst.extend(L)	将列表L中所有元素添加至列表lst尾部
lst.insert(index, x)	在lst指定位置index处添加元素x
lst.remove(x)	在lst中删除首次出现的元素x
lst.pop([index])	删除并返回lst指定位置的元素,默认删除列表尾部元素
lst.clear()	删除列lst中所有元素,但保留列表对象(Python2不支持)
lst.index(x)	返回lst中值为x的首个元素的下标
lst.count(x)	返回指定元素x在lst中的出现次数
lst.reverse()	对lst的所有元素进行原地逆序(修改lst本身)
lst.sort()	对lst的所有元素进行原地排序(修改lst本身)
lst.copy()	返回lst的拷贝(Python2不支持)

3.1.1 列表的创建与删除

1. 使用"="直接将一个列表赋值给变量

```
>>> a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
>>> a_list = []      # 创建空列表
```

2. 使用list()BIF将可迭代对象(元组, range对象, 字符串, ...)转换为列表.

```
>>> a_list = list((3, 5, 7, 9, 11))  # 将元组转换为列表
>>> a_list
[3, 5, 7, 9, 11]
>>> list(range(1, 10, 2))  # 将range对象转换为列表
[1, 3, 5, 7, 9]
>>> list('hello world')    # 将字符串转换为列表
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> x = list() #创建空列表
>>> x
[]
```

内置函数range()

```
range([start,] stop[, step])   # 函数参数中的[]代表可选参数
```

- start: 起始值(默认为0)
- stop: 终止值+1(即, 结果中不包括stop)
- step: 步长(默认为1)
- 函数返回一个range对象(可迭代)
- 可用list()函数将range对象转化为列表

```
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
```

- 不再使用时, 可用del命令删除整个列表
  - 若列表对象所指向的值不再有其它对象引用，Python将同时删除该值.

```
>>> del a_list
>>> a_list
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    a_list
NameError: name 'a_list' is not defined
```

  - 删除列表对象a_list之后，该对象就不存在了，再次访问时将抛出异常"NameError".

3.1.2 列表元素的访问与计数

- 下标→列表元素
  - 若指定下标不存在→抛出异常IndexError

```
>>> aList=[3,4,5,6,7,9,11,13,15,17]
>>> aList[3]
6
>>> aList[3]=5.5
>>> aList
[3,4,5,5.5,7,9,11,13,15,17]
```

```
>>> aList[15]
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in
   <module>
    aList[15]
IndexError: list index out of
  range
```

Copy&Run

指定元素→下标

- 列表的index方法: 获取指定元素首次出现的下标

```
L.index(value, [start, [stop]]) # return first index of value.
                  # Raises ValueError if the value is not present.
>>> aList
[3, 4, 5, 5.5, 7, 9, 11, 13, 15, 17]
>>> aList.index(7)
4
>>> aList.index(100)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    aList.index(100)
ValueError:  100 is not in list
```

统计指定元素在列表中出现的次数: 列表的count方法

```
>>> aList
[3, 4, 5, 5.5, 7, 9, 11, 13, 15, 7]
>>> aList.count(7)
2
>>> aList.count(11)
1
>>> aList.count(8)
0
```

3.1.3 成员资格判断

判断列表中是否存在指定值

- 1) 使用count()方法: 返回大于0的数→成员存在, 返回0→不存在.
- 2) 使用“in”关键字判断某个值是否存在于列表中, 返回True/False.
  - 适用于所有可迭代对象: 元组, 字典, range对象, 字符串, 集合, ……
  - 通常在循环中使用: 对可迭代对象的元素进行遍历.

```
>>> aList
[3, 4, 5, 5.5, 7, 9, 11, 13, 15, 17]
>>> 3 in aList
True
>>> 18 in aList
False
>>> bList = [[1], [2], [3]]
>>> 3 in bList
False
>>> 3 not in bList
True
>>> [3] in bList
True
```

遍历列表——for循环语法

```
for 变量 in 序列/迭代对象:
        循环体
```

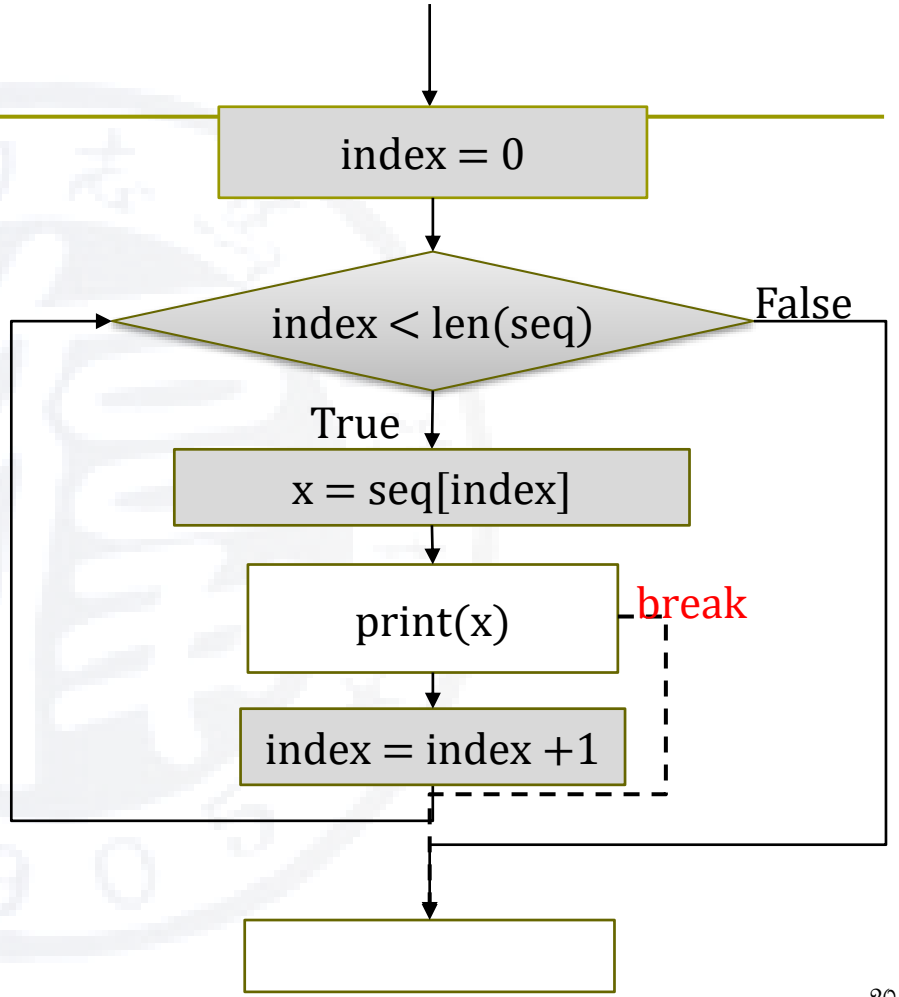
```
for 变量 in 序列/迭代对象:
        循环体
else:
        语句块
```

当循环自然结束(不是因为执行了break而结束)时，执行else结构中的语句.

for循环流程(1)

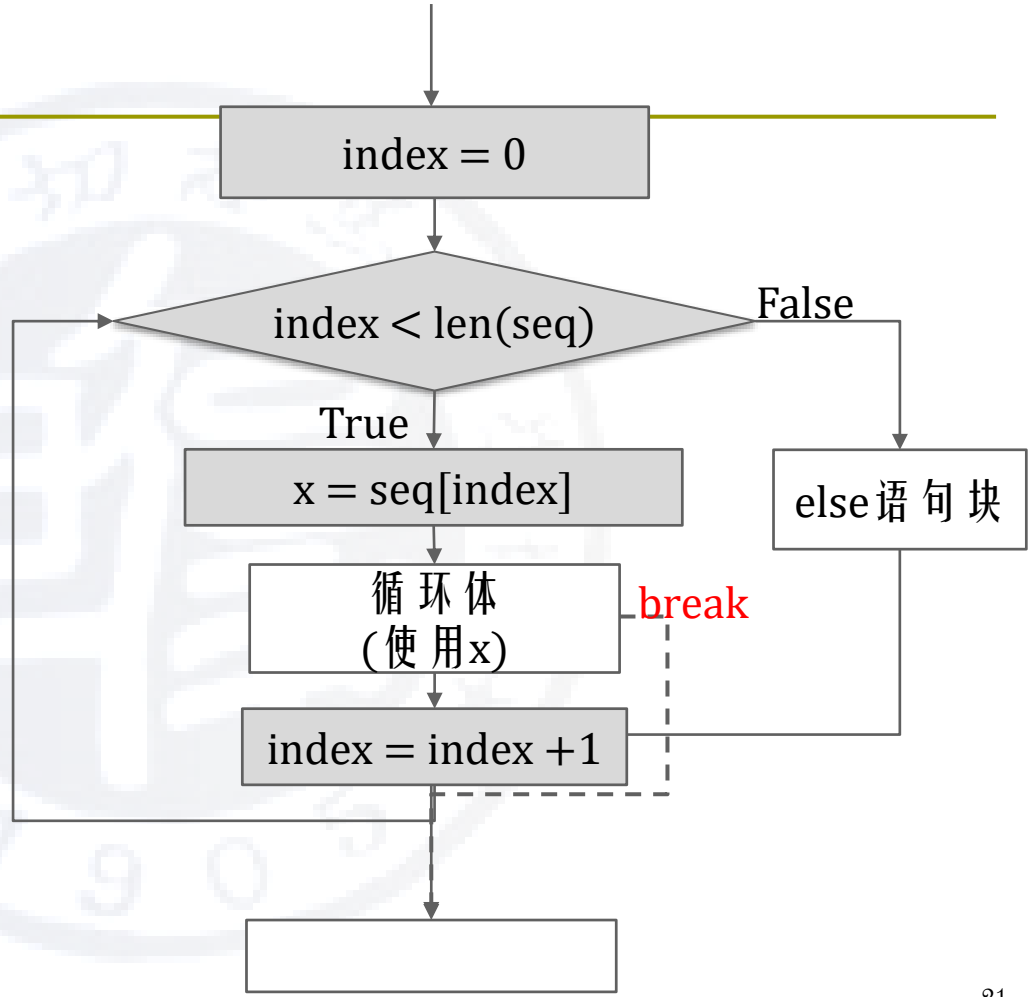
```
[例]seq = [1, 2, 3, 4, 5]
    for x in seq:
        print(x)
```

- index: 内部隐含变量; 灰色框: 内部隐含操作;
- 每次比较时(每轮循环), 重新计算len(seq);



for循环流程(2)

```
for x in seq:
    循环体
else:
    语句块
```



示例1 for语句

```
>>>
This is Line 1
This is Line 2
This is Line 3
This is Line 4
This is Line 5
This is Line 6
This is Line 7
This is Line 8
This is Line 9
This is Line 10
>>>
```

```
for lineNumber in range(10):
    print('This is line', lineNumber + 1)
```

对照

```
lineNumber = 0

while lineNumber < 10:
    print('This is line', lineNumber+1)
    lineNumber += 1
```

示例2

- 输入一个字符串，输出以空格间隔的字符序列

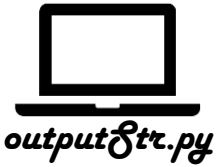
```
>>>
请输入一串字符：FudanUniversity
F u d a n U n i v e r s i t y
>>>
```

for 1

```
s = input('Enter a string: ')
dis = ''
for each in s:
    dis += each + ' '
print(dis)
```

for 2

```
s = input('Enter a string: ')
for each in s:
    print(each, end = ' ')
print()
```



示例3

- 实现列表的index()方法, 即, 在列表中查找指定值首次出现的下标.

```
from random import sample
listRandom = sample(range(100), 10)#random.sample()返回列表
print('The list is: ', listRandom)

key = eval(input('Enter an integer: '))
i = 0
while i < len(listRandom):
    if listRandom[i] == key:
        print('The index of %d is %d.' % (key, i))
        break # 找到就终止循环
    i += 1
else: # 没有找到
    print(key, 'is not in the list.')
```

Copy&Run

```
from random import sample

listRandom = sample(range(100), 10)
print('The list is: ', listRandom)

key = eval(input('Enter an integer: '))
for i in range(len(listRandom)):  # range对象保存列表的索引
    if listRandom[i] == key:
        print('The index of %d is %d.' % (key, i))
        break
else:
    print(key, 'is not in the list.')
```

for语句的循环变量自动变化

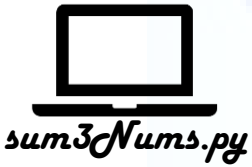
```
# 输出10以内奇数
i = 0
while i < 10:
    if i % 2 == 0:
        continue
    print(i, end=' ')
    i += 1
```

```
# i=seq[index],index每轮循环自动增1
for i in range(10):
    if i % 2 == 0:
        i += 1   # 并不会影响i的取值
        continue
    print(i, end=' ')
```

- 当前循环的i与下一轮循环的i不同⇨修改其值并不影响循环的执行.

比较两段代码的功能差异

```
i = s = 0
while i < 3:
    x = int(input('Enter an integer: '))
    if not x:
        continue  # x为0则回到循环头部
    s += x
    i += 1
print('The sum is:', s)
```




```
i = s = 0
for i in range(3):
    x = int(input('Enter an integer: '))
    if not x:
        continue
    s += x
print('The sum is:', s)
```

Copy&Run

```
i = s = 0   # while语句的循环变量由程序语句控制其变化
while i < 3:
    x = int(input())
    if not x:
        continue
    s += x
    i += 1
print('The sum is:', s)
```

```
7
0
0
2
0
6
The sum is: 15
```

```
i = s = 0   # for语句的循环变量自动变化
for i in range(3):
    x = int(input())
    if not x:
        continue
    s += x
print('The sum is:', s)
```

```
7
0
0
The sum is: 7
```

课堂练习

- 下面的程序输出什么?

```
s = 0
for i in range(10):
    if i%2:
        continue
    s += i
print(s)
```

```
20
```

for循环与while循环

两种基本的循环结构语句

- while语句: 用于循环次数难以提前确定的情况.
- for语句: 用于循环次数可提前确定的情况, 尤其适用于枚举序列/迭代对象中的元素.

循环结构可互相嵌套, 实现复杂的逻辑.

3.1.4 增加列表元素

1) 使用“+”运算符将两个列表合并, 生成一个新列表

```
>>> aList = [3, 4, 5]
>>> aList = aList + [7]
>>> aList
[3, 4, 5, 7]
```

+ 思考题: 如何判定是否创建了新列表?

- 将两个列表中的元素依次复制到新列表.
- 由于涉及大量元素的复制, "+"操作速度较慢⇨添加大量元素时不建议使用该方法. (P36程序: 增加列表元素的不同方法的性能比较)

2) 使用列表对象的append()方法, 原地修改列表

- 在列表尾部添加一个元素, 速度较快, 推荐使用.

```
>>> aList.append(9)
>>> aList
[3, 4, 5, 7, 9]
```

+ 思考题: 如何判定是原地修改列表?

3) 使用列表对象的extend()方法，将另一个迭代对象的所有元素添加至该列表尾部.

- 原地操作.

```
>>> a = [5, 2, 4]
>>> id(a)
54728008
>>> a.extend([7, 8, 9])
>>> a
[5, 2, 4, 7, 8, 9]
>>> id(a)
54728008
```

与a = a + [7, 8, 9]效果相同，但extend效率高.

4) 使用列表对象的insert()方法将元素添加至指定位置.

```
L.insert(index, object)  # insert object before index
>>> aList = [3, 4, 5, 7, 9]
>>> aList.insert(3, 6)
>>> aList
[3, 4, 5, 6, 7, 9]
```

- insert()方法涉及插入位置后所有元素的移动⇨影响处理速度(P38例程: insert与append方法的性能比较).
- 列表删除方法remove()和pop()弹出非尾部元素时, 也有类似问题.
- 建议：除非有必要，应尽量避免在列表中间位置插入/删除元素，优先使用append()和pop()在列表尾部增删元素.

5) 使用乘法扩展列表对象：列表与整数相乘，生成新列表，新列表是原列表的重复.

```
>>> aList = [1, 2, 3]
>>> aList = aList*3
>>> aList
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- 不是原地修改，也适用于字符串/元组，并具有相同特点.

- 若列表的元素是列表，使用*扩展列表➡创建多维数组
  - 此时，不复制元素的值，而是复制值的引用⇨每个重复元素指向相同值.

```
>>> x = [[1, 2, 3]]*3
>>> x
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>> x[0][0] = 10 # x[0][0], x[1][0], x[2][0]引用相同值
>>> x
[[10, 2, 3], [10, 2, 3], [10, 2, 3]]
>>> x = [[None] * 2] * 3   # None: 空类型
>>> x
[[None, None], [None, None], [None, None]]
>>> x[0][0] = 5
>>> x
[[5, None], [5, None], [5, None]]
```

各种增加列表元素方法的比较

	列表元素增加方法	别名	效果
1	+	拼接	新建列表
2	lst.append	追加	原地修改
3	lst.extend	扩展	原地修改
4	lst.insert	插入	原地修改
5	*	复制	新建列表

3.1.4 删除列表元素

1. del命令: 删除列表指定位置元素(原地操作)或整个列表

```
>>> a_list = [3, 5, 7, 9, 11]
>>> del a_list[1]
>>> a_list
[3, 7, 9, 11]
```

2. 列表的pop(index = -1)方法

- 删除并返回指定位置(默认为最后一个)元素.
- 若给定索引超出列表范围, 则抛出异常.

```
>>> a_list = list((3,5,7,9,11)) #将元组(3,5,7,9,11)转换为列表
>>> a_list.pop()      # 默认删除最后一个元素
11   # pop方法返回值：删除的元素
>>> a_list
[3, 5, 7, 9]
>>> a_list.pop(1)
5
>>> a_list
[3, 7, 9]
```

3. 列表的remove(value)方法

- 删除首次出现的指定元素，若列表中不存在要删除的元素，则抛出异常.

```
>>> a_list = [3, 5, 7, 9, 7, 11]
>>> a_list.remove(7)   # 删除第一个7
>>> a_list
[3, 5, 9, 7, 11]
```

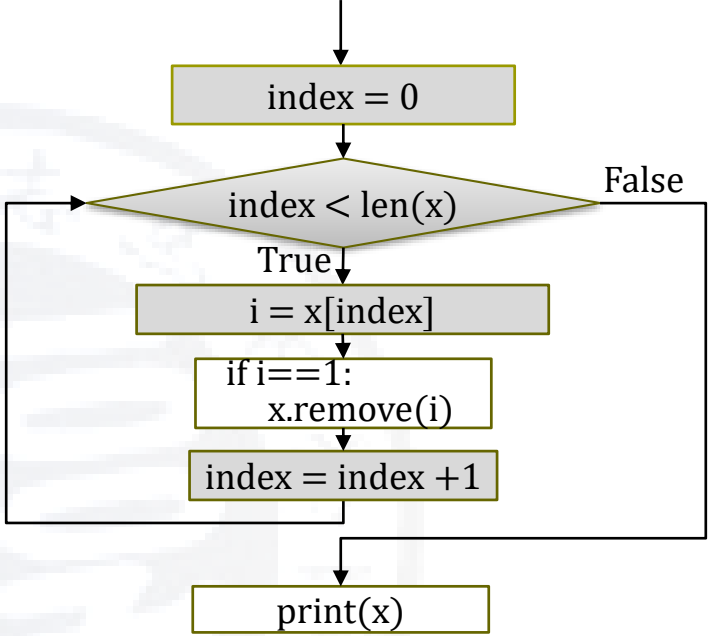
使用循环删除列表元素

- 使用"循环+remove()"方法, 逐个删除列表中的指定元素.

  e.g. 删除列表x中所有的1.

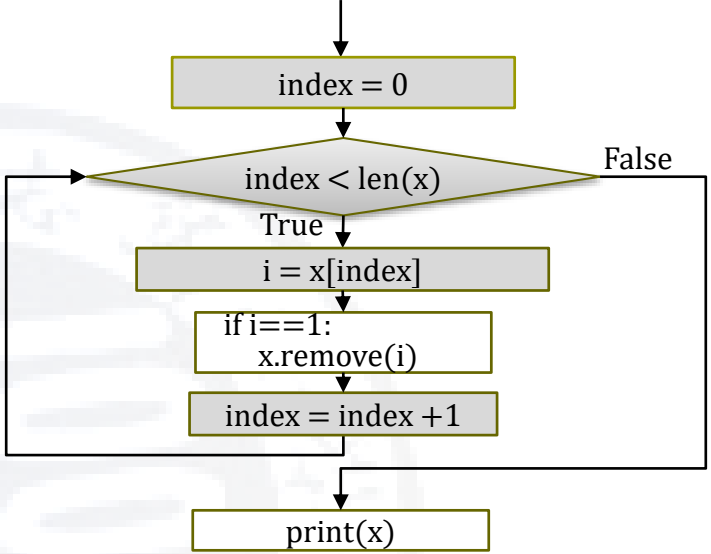
```
x = [1, 2, 1, 2, 1, 2, 1, 2, 1]
#这段代码的逻辑是错误的,尽管执行结果正确
for i in x:
    if i == 1:
        x.remove(i)
print(x)
[2, 2, 2, 2]
```

```
x = [1, 2, 1, 2, 1, 1, 1]
for i in x: # i: x[index]
    if i == 1:
        x.remove(i)
print(x)
[2, 2, 1]
```



- 为什么?
  - 列表 x 随着元素被删除而变化→每轮循环处理的列表不一样.
  - 每插入/删除一个元素，该元素位置后面所有元素的索引都发生改变.
  - for循环隐含的index不管列表是否变化, 每次均自动增加1

```
x = [1, 2, 1, 2, 1, 1, 1]
for i in x: # i: x[index]
    if i == 1:
        x.remove(i)
print(x)
```



列表x(每轮循环开始)	隐含下标index	index < len(x)	i: x[index]	列表x(每轮循环结束)
[1, 2, 1, 2, 1, 1, 1]	0	0 < 7⇨True	1	[2, 1, 2, 1, 1, 1]
[2, 1, 2, 1, 1, 1]	1	1 < 6⇨True	1	[2, 2, 1, 1, 1]
[2, 2, 1, 1, 1]	2	2 < 5⇨True	1	[2, 2, 1, 1]
[2, 2, 1, 1]	3	3 < 4⇨True	1	[2, 2, 1]
[2, 2, 1]	4	4 < 3⇨False		循环结束

- 怎么办?
  - 方法1: 使用while循环, 用程序语句控制循环变量的变化

```
x = [1, 2, 1, 2, 1, 1, 1]
i = 0
while i < len(x):
    if x[i] == 1:
        x.pop(i)   # 下一轮待考察元素还是x[i]
    else:
        i = i + 1
print(x)
```

```
x = [1, 2, 1, 2, 1, 1, 1]
i = 0
while i < len(x):
    if x[i] == 1:
        x.pop(i)   # 下一轮待考察元素还是x[i]
    else:
        i = i + 1
print(x)
```

列表x(每轮循环开始)	下标i	i < len(x)	x[i]	列表x(每轮循环结束)
[1, 2, 1, 2, 1, 1, 1]	0	0 < 7⇨True	1	[2, 1, 2, 1, 1, 1]
[2, 1, 2, 1, 1, 1]	0	0 < 6⇨True	2	[2, 1, 2, 1, 1, 1]
[2, 1, 2, 1, 1, 1]	1	1 < 6⇨True	1	[2, 2, 1, 1, 1]
[2, 2, 1, 1, 1]	1	1 < 5⇨True	2	[2, 2, 1, 1, 1]
[2, 2, 1, 1, 1]	2	2 < 5⇨True	1	[2, 2, 1, 1]
[2, 2, 1, 1]	2	2 < 4⇨True	1	[2, 2, 1]
[2, 2, 1]	2	2 < 3⇨True	1	[2, 2]
[2, 2]	2	2 < 2⇨False		循环结束

方法2: 从后往前遍历列表.

```
x = [1, 2, 1, 2, 1, 1, 1]
for i in range(len(x)-1, -1, -1):
    if x[i] == 1:
        del x[i]     # x.pop(i)
```

列表x(每轮循环开始)	i	x[i]	列表x(每轮循环结束)
[1, 2, 1, 2, 1, 1, 1]	6	1	[1, 2, 1, 2, 1, 1]
[1, 2, 1, 2, 1, 1]	5	1	[1, 2, 1, 2, 1]
[1, 2, 1, 2, 1]	4	1	[1, 2, 1, 2]
[1, 2, 1, 2]	3	2	[1, 2, 1, 2]
[1, 2, 1, 2]	2	1	[1, 2, 2]
[1, 2, 2]	1	2	[1, 2, 2]
[1, 2, 2]	0	1	[2, 2]
[2, 2]	-1		循环结束

3.1.6 切片操作

切片是Python序列的重要操作之一, 适用于列表/元组/字符串/range对象

- 语法格式: lst[d1:d2:d3]
- d1: 切片的开始位置(默认为0)
- d2: 切片的截止(但不包含)位置(默认为列表长度)
- d3: 切片的步长(默认为1), 若省略则第二个冒号亦可省略.
- 返回值: 列表元素的拷贝.

e.g. lst = [1, 2, 3, 4, 5]

lst[::], lst[:] 二者等价

赋值 vs 切片

```
>>> aList = [3, 5, 7]
>>> bList = aList # bList与aList
>>> bList          是同一列表
[3, 5, 7]
>>> bList[1] = 8
>>> aList
[3, 8, 7]
>>> aList == bList
True
>>> aList is bList
True
>>> id(aList)
19061816
>>> id(bList)
19061816
```

```
>>> aList = [3, 5, 7]
>>> bList = aList[:]
>>> aList == bList
True   # 包含的元素相同
>>> aList is bList
False  # 不同列表
>>> id(aList) == id(bList)
False
>>> bList[1] = 8
>>> bList
[3, 8, 7]
>>> aList
[3, 5, 7]
```

切片的功能

- 功能1: 截取列表中的任何部分, 得到一个新列表.

```
>>> aList = [3, 4, 5, 6, 7, 9, 11,
13, 15, 17]
>>> aList[: : ]
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> aList[: : -1]   # 反向切片
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
>>> aList[: : 2]
[3, 5, 7, 11, 15]
>>> aList[1: : 2]
[4, 6, 9, 13, 17]
>>> aList[3: : ]
[6, 7, 9, 11, 13, 15, 17]
```

```
>>> aList[3: 6]
[6, 7, 9]
>>> aList[3: 6: 1]
[6, 7, 9]
>>> aList[0: 100: 1]
[3, 4, 5, 6, 7, 9, 11,
13, 15, 17]
>>> a[100: ]
[]
```

- 与使用下标访问列表元素的方法不同, 切片操作不会因下标越界而抛出异常: 只是在列表尾部截断或返回空列表, 代码具有更强的健壮性.

功能2: 使用切片原地修改列表内容(修改/删除/增加列表元素)

```
>>> aList = [3, 5, 7]
>>> aList[len(aList): ]
[]
>>> aList[len(aList): ] = [9]
#增加列表元素
>>> aList
[3, 5, 7, 9]
>>> aList[: 3] = [1, 2, 3]
#修改列表元素
>>> aList
[1, 2, 3, 9]
>>> aList[: 3] = []    #删除
>>> aList
[9]
```

将列表中的某些元素直接替换（右侧替换左侧
ATTENTION ： 原地修改

```
>>> aList = list(range(10))
>>> aList
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> aList[: : 2] = [0]*(len(aList)//2)
>>> aList
[0, 1, 0, 3, 0, 5, 0, 7, 0, 9]
```

把切片作为左值使用(赋值运算符左边), 会对原列表进行修改.

功能3: 使用del与切片结合, 删除多个列表元素

```
>>> aList = [3, 5, 7, 9, 11]
>>> del aList[: 3]  # aList[:3] = []
>>> aList
[9, 11]
```

[例] 删除列表x中所有的1——使用for循环正向遍历列表

- 方法3：在循环的判断条件中，使用列表的切片替代原始列表→即使原始列表因元素增删而变化，切片不变.

```
x = [1, 2, 1, 2, 1, 1, 1]
for i in x[:]:    # 循环考察原列表的切片
    if i == 1:
        x.remove(i)    # 修改原列表，不会影响切片
```

```
x = [1, 2, 1, 2, 1, 1, 1]
for i in x[:]:
    if i == 1:
        x.remove(i)
```

Q: 如下写法对吗?

```
x_copy = x
for i in x_copy:
    if i == 1:
        x.remove(i)
```

x[:](在循环中不变)	隐含下标index	index < len(x[:])	i: x[:][index]	列表x(每轮循环结束)
[1, 2, 1, 2, 1, 1, 1]	0	0 < 7⇨True	1	[2, 1, 2, 1, 1, 1]
	1	1 < 7⇨True	2	[2, 1, 2, 1, 1, 1]
	2	2 < 7⇨True	1	[2, 2, 1, 1, 1]
	3	True	2	[2, 2, 1, 1, 1]
	4	True	1	[2, 2, 1, 1]
	5	True	1	[2, 2, 1]
	6	True	1	[2, 2]
	7	7 < 7⇨False		循环结束

[例] 编写程序, 计算今天是今年的第几天.

```
import time
date = time.localtime()
year = date[0]
month = date[1]
day = date[2]
day_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

# 判断year是否为闰年
if(year % 4 == 0 and year % 100 != 0) or year % 400 == 0:
    day_month[1] = 29
if month == 1:
    print(day)
else:
    print(sum(day_month[: month-1])+day)
```

3.1.7 列表排序

1. 使用列表的sort方法进行原地排序——支持多种排序

```
>>> aList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> import random
>>> random.shuffle(aList)   # 打乱顺序
>>> aList
[11, 3, 13, 15, 6, 17, 5, 9, 4, 7]
>>> aList.sort()   # 默认为升序排列
>>> aList
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]   # aList发生改变
>>> aList.sort(reverse = True)   # 降序排列
>>> aList
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
```

Question: 以下print函数将输出什么?

```
>>> print(aList.sort())
```

Copy&Run

2. 使用BIF sorted(...)对可迭代对象排序并返回排好序的新列表

```
>>> aList = [9, 7, 6, 5, 4, 3, 17, 15, 13, 11]
>>> sorted(aList)
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]   # sorted(aList)的返回值
>>> aList
[9, 7, 6, 5, 4, 3, 17, 15, 13, 11]   # aList没变
>>> bList = sorted(aList)   # 定义新列表把排序结果保存下来
>>> bList
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
>>> bList = sorted(aList, reverse=True)
>>> bList
[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]
```

Question: 以下print函数将输出什么?

```
>>> print(sorted(aList))
```

list.sort(...) *vs* BIF sorted(...)

相同之处

- 可以对列表排序.

不同之处

- 调用方式不同: 前者属于列表的方法(仅对列表有效); 后者属于内置函数, 可以对包含列表在内的多种可迭代对象排序.
- lst.sort(...) 修改lst, 返回None, 即, 无返回值.
  - BIF sorted(lst, ...)不修改lst, 而是返回排序后的新列表

```
bList = aList.sort() ✘
bList = sorted(aList)
```

3. 列表对象的reverse方法——原地逆序

```
>>> aList = [4, 11, 7, 5, 15, 6, 17, 3, 9, 13]
>>> aList.reverse()
>>> aList
[13, 9, 3, 17, 6, 15, 5, 7, 11, 4]
```

4. BIF reversed(...): 将列表元素逆序并返回迭代器(iterator)对象

```
>>> aList = [4, 11, 7, 5, 15, 6, 17, 3, 9, 13]
>>> c = reversed(aList)
>>> c
<list_reverseiterator object at 0x01FD0A50>
>>> list(c)      # 将迭代对象转换为列表
[13, 9, 3, 17, 6, 15, 5, 7, 11, 4]
>>> list(c):
[]  # 无输出内容，迭代对象已遍历结束，需重新创建迭代对象
>>> c = list(reversed(aList))
>>> c
[13, 9, 3, 17, 6, 15, 5, 7, 11, 4]
```

3.1.8 用于序列操作的常用内置函数

- 序列大小比较: 关系运算符, 序列的__le__()及相关方法.

```
>>> [1,2,3] < [1,2,4]
True
>>> 'ABC' < 'C' < 'Pascal' < 'Python'
True
>>> (1,2,3,4) < (1,2,4)
True
>>> (1,2) < (1,2,-1)
True
>>> (1,2,3) == (1.0,2.0,3.0)
True
>>> (1,2,('aa','ab'))<(1,2,('abc','a'),4)
True
>>> 'a' > 'A'
True
```

```
>>> a = [1, 2]
>>> b = [1, 2, 3]
>>> a.__le__(b)
True
>>> a.__gt__(b)
False
>>> a > b
False
>>> a < b
True
```

Copy&Run

len(列表)

- 返回列表中的元素个数, 还适用于元组/字典/字符串/range, ...

max(列表), min(列表)

- 返回列表中的最大或最小元素, 还适用于元组/range.

sum(列表)

- 对数值型列表的元素进行求和运算, 对非数值型列表运算出错, 同样适用于元组/range.

- zip(列表1, 列表2, ...)
  - 将多个列表对应位置元素组合为元组, 返回包含这些元组的zip对象.
  - Zip对象可用list()函数转换为列表.
  - 该函数对元组/字符串对象同样有效.

```
>>> aList = [1, 2, 3]
>>> bList = [4, 5, 6]
>>> cList = zip(aList, bList)
>>> cList
<zip object at 0x0000000003728908>
>>> list(cList)
[(1, 4), (2, 5), (3, 6)]      # 列表的元素是元组
```

- enumerate(列表)
  - 枚举列表元素, 返回枚举对象: 每个元素为包含下标和值的元组.
  - 枚举对象可用list()函数转换为列表.
  - 该函数对元组/字符串/zip对象同样有效.

遍历元素

```
>>> dList = [5, 6, 7]
>>> for item in enumerate(dList):
        print(item)
(0, 5)
(1, 6)
(2, 7)
```

```
>>> keys = ['a', 'b', 'c', 'd']
>>> values = [1, 2, 3, 4]
>>> for k, v in zip(keys, values): # k→keys[index], v→values[index]
        print(k, v)


a 1
b 2
c 3
d 4
>>> aList = [1, 2, 3]
>>> bList = [4, 5, 6]
>>> cList = [7, 8, 9]
>>> dList = zip(aList, bList, cList)
>>> for index, value in enumerate(dList):
        print(index, ': ', value)


0 :  (1, 4, 7)
1 :  (2, 5, 8)
2 :  (3, 6, 9)
```

将一组值直接用元组赋值

示例3(P25)

- 实现列表的index()方法, 即, 在列表中查找指定值首次出现的下标.

```
from random import sample

listRandom = sample(range(100), 10)
print('The list is: ', listRandom)

key = eval(input('Enter an integer: '))
for i, value in enumerate(listRandom):
    if value == key:
        print('The index of %d is %d.' % (key, i))
        break
else:
    print(key, 'is not in the list.')
```

枚举函数的主要用途是联系下标和对应元素

Copy&Run

3.1.9 列表推导式——轻量级循环

- 利用其它列表创建新列表的一种方法.

```
[表达式 for 变量 in 列表 (if 条件)] # "if 条件"可选
[expr for value in seq (if condition)] # 列表元素：expr的运算结果
```

```
>>> aList = [x*x for x in range(10)]
>>> aList
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

相当于:

```
>>> aList = []
>>> for x in range(10):
        aList.append(x*x)
```

依次添加

- 过滤不符合条件的元素——if子句

```
>>> aList = [-1, -4, 6, 7.5, -2.3, 9, -11]
>>> [i for i in aList if i > 0]   # 过滤掉列表中的非正数
[6, 7.5, 9]
```

- 使用列表推导式实现嵌套列表的平铺——for子句嵌套

```
>>> vec = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

相当于

```
>>> v = []
>>> for elem in vec:
        for num in elem:
            v.append(num)
```

[例] 用户输入一个正整数n, 求从1~n各数的平方和(Chap2. P23)

```
n = int(input('Enter an integer: '))
```

```
i = 1
s = 0
while i <= n:
    s += i * i
    i += 1
```

```
s = 0
for i in range(1, n+1):
    s += i * i
```

```
x = [i * i for i in range(1, n+1)]
s = sum(x)
```

```
print('Sum of squares(1~%d): %d' % (n, s))
```

- 在列表推导式中使用多个循环，可实现多序列元素的任意组合，还可结合条件语句过滤特定元素.

```
>>> [(x, y) for x in range(3) for y in range(3)]
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]

>>> v = []
>>> for x in range(3):
        for y in range(3):
                v.append((x,y))

>>> [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

>>> v = []
>>> for x in [1, 2, 3]:
        for y in [3, 1, 4]:
                if x!= y:
                        v.append((x, y))
```

列表推导式中可使用函数或复杂表达式

```
>>> [sum(item) for item in [(1, 2, 3), (4, 5, 6), (7, 8, 9)]]
[6, 15, 24]

>>> [v**2 if v % 2 == 0 else v+1 for v in [2, 3, 4, -1] if v > 0] #复杂表达式
[4, 4, 16]
```

两维列表&循环嵌套

- 表格/矩阵中的数据可用两维列表存储.

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1,375	967	1,087
Boston	983	0	214	1,102	1,505	1,723	1,842
New York	787	214	0	888	1,549	1,548	1,627
Atlanta	714	1,102	888	0	661	781	810
Miami	1,375	1,505	1,549	661	0	1,426	1,187
Dallas	967	1,723	1,548	781	1,426	0	239
Houston	1,087	1,842	1,627	810	1,187	239	0

```
distances = [[0, 983, 787, 714, 1375, 967, 1087],
             [983, 0, 214, 1102, 1505, 1723, 1842],
             [787, 214, 0, 888, 1549, 1548, 1627],
             [714, 1102, 888, 0, 661, 781, 810],
             [1375, 1505, 1549, 661, 0, 1426, 1187],
             [967, 1723, 1548, 781, 1426, 0, 239],
             [1087, 1842, 1627, 810, 1187, 239, 0]]
```

两维列表

- 两维列表中的值: 通过行下标&列下标访问.

  e.g. 列表matrix中的每个值可用matrix[i][j]来访问, i、j分别是行下标和列下标.

```
matrix = [
    [1, 2, 3, 4, 5],
    [6, 7, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [1, 0, 0, 0, 8],
    [0, 0, 9, 0, 3],
]
```

	[0]	[1]	[2]	[3]	[4]
[0]	1	2	3	4	5
[1]	6	7	0	0	0
[2]	0	1	0	0	0
[3]	1	0	0	0	8
[4]	0	0	9	0	3

```
matrix[0] is [1, 2, 3, 4, 5]
matrix[1] is [6, 7, 0, 0, 0]
matrix[2] is [0, 1, 0, 0, 0]
matrix[3] is [1, 0, 0, 0, 8]
matrix[4] is [0, 0, 9, 0, 3]

matrix[0][0] is 1
matrix[4][4] is 3
```

使用输入值来构建二维列表

- 用户输入矩阵的行数和列数, 然后输入矩阵内每个元素.

```
matrix = []       # 创建新列表
rows = eval(input('Enter the number of rows: '))
cols = eval(input('Enter the number of cols: '))
for row in range(rows):
    matrix.append([])    # 在矩阵中添加新行
    for col in range(cols):
        ele = eval(input('input an element and press Enter: '))
        matrix[row].append(ele)
print(matrix)
```

使用随机数来构建二维列表

- 矩阵的每个元素都是0~99之间的随机整数.

```
matrix = []
rows = eval(input('Enter the number of rows: '))
cols = eval(input('Enter the number of cols: '))
for row in range(rows):
    matrix.append([])
    for col in range(cols):
        matrix[row].append(random.randint(0, 99))
print(matrix)
```

Copy&Run

输出矩阵

```
print('The matrix is:')
for row in range(len(matrix)):
    for col in range(len(matrix[row])):
        print('%-6d' % matrix[row][col], end='')
    print()  # 换行
```

行数

列数

或

```
for row in matrix:
    for value in row:
        print('%-6d' % value, end='')
    print()
```

对矩阵所有元素求和

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

total = 0
for row in matrix:
    for value in row:
        total += value

print("Total is", total) # Print the total
```

```
for row in matrix:
    total += sum(row)
```

- 也可用一层循环，循环内对每行调用sum()

按列求和

- 输出矩阵各列之和

列数

```
for col in range(len(matrix[0])): # 外层循环是列循环
    total = 0
    for row in range(len(matrix)): # 内层循环是行循环
        total += matrix[row][col]
    print('The sum of column %d is %d.' % (col+1, total))
```

行数

找出和最大的行

- maxRow: 最大和, maxIndex: 对应的行下标

```
maxSum = 0   # 将最大和的初值置为0
for i, row in enumerate(matrix):
    if maxSum < sum(row):   # 如果当前行的和大于maxSum
        maxSum = sum(row)   # 更新maxSum
        maxIndex = i   # 更新maxSum的行下标
print('Row %d has the maximum sum of %d.' %
      (maxIndex+1, maxSum))
```

打擂台

打乱矩阵

```
from random import randint

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(matrix)

rows, cols = len(matrix), len(matrix[0])
for row in range(rows):
    for col in range(cols):
        i = randint(0, rows-1)  # 随机生成行下标
        j = randint(0, cols-1)  # 随机生成列下标
        matrix[row][col], matrix[i][j] = \
            matrix[i][j], matrix[row][col]  # 交换两个元素

print(matrix)
```

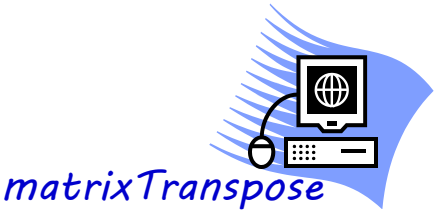
矩阵转置

1. 使用列表推导式

```
>>> matrix = [ [1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> [[ row[i] for row in matrix ] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

2. 使用内置函数

```
>>>list(zip(*matrix))    # *matrix: 序列解包
 [(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```



Copy&Run

二维列表排序

- sort()方法可以对二维列表排序.
  - 通过每行的第一个元素进行排序;
  - 对于第一个元素相同的行, 则通过它们的第二个元素进行排序;
  - 如果前二个元素都相同, 则利用第三个元素进行排序;
  - 依此类推……

```
points = [[4, 2], [1, 7], [4, 5], [1, 2], [1, 1], [4, 1]]
points.sort()
print(points)
```

```
[[1, 1], [1, 2], [1, 7], [4, 1], [4, 2], [4, 5]]
```

示例 —— 给学生答案评分

- 假设有8名学生和10道选择题, 他们的答案存储在一个表格中, 每一行记录了一位学生对这些问题的答案, 如下图所示: 每题1分, 程序显示评分结果.

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

标准答案:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

代码(1)

```
answers = [['A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'],
           ['D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'],
           ['E', 'D', 'D', 'A', 'C', 'B', 'A', 'E', 'A', 'D'],
           ['C', 'B', 'A', 'E', 'D', 'C', 'D', 'E', 'A', 'D'],
           ['A', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'],
           ['B', 'B', 'E', 'C', 'C', 'D', 'A', 'A', 'C', 'D'],
           ['B', 'B', 'A', 'C', 'C', 'D', 'D', 'C', 'C', 'D'],
           ['E', 'B', 'E', 'C', 'C', 'A', 'B', 'C', 'D', 'A']]
keys = ['D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D']

print("The Students' correct counts are as follow (10 in all): ")
for i in range(len(answers)):
    correct = 0
    for j in range(len(answers[i])):
        if answers[i][j] == keys[j]:
            correct += 1
    print('Student %d : %d' % (i, correct))
```

代码(2)

- 不使用下标, 程序可读性更强.

```
for index, record in enumerate(answers):
    correct = 0
    for ans, key in zip(record, keys):
        if ans == key:
            correct += 1
    print('Student %d : %d' % (index+1, correct))
```

3.2 元组

- 3.2.1 元组的创建与删除
- 3.2.2 元组与列表的区别
- 3.2.3 序列解包
- 3.2.4 生成器推导式

3.2.0 元组

和列表类似, 但属于不可变序列.

一旦创建, 用任何方法都不能修改其元素.

定义方式和列表相同, 但所有元素放在一对'()'中, 而不是'[]'.

```
>>> TIOBE = ('Java', 'C',
   'Python',  'C++', C#')
>>> TIOBE
('Java', 'C', 'Python',  'C++',
   C#')
>>> x = ()   # 空元组
>>> x
()
>>> x=1,2,3 #不引起歧义时,括号可省
>>> x
(1,2,3)
```

注意: 若创建只有一个元素的元组, 需在元素后加上逗号.

```
>>> a_tuple = ('a', )
>>> a_tuple
('a',)
>>> a = 'a',
>>> a
('a',)
>>> a = ('a')
>>> a
'a'
```

3.2.1 元组的创建与删除

- tuple函数——将其它序列转换为元组并返回

```
>>> print(tuple('abcdefg'))   # 将字符串转换为元组
('a', 'b', 'c', 'd', 'e', 'f', 'g')
>>> aList = [-1, -4, 6, 7.5, -2.3, 9, -11]
>>> tuple(aList)   # 将列表转换为元组
(-1, -4, 6, 7.5, -2.3, 9, -11)
>>> s = tuple()   # 空元组
>>> s
()
```

- 使用del删除整个元组，不能删除元组元素

元组访问(补)

- 与list/str一样, 可以访问某个元素, 但不能赋值

1) 下标访问某个元素

```
>>> x = tuple([x**2 for x in range(12)])
>>> x
(0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121)
>>> x[5], x[-2]
(25, 100)
```

#把列表推导式生成的列表转换为元组

2） 切片访问多个元素

```
>>> x[: : -1]
(121, 100, 81, 64, 49, 36, 25, 16, 9, 4, 1, 0)
>>> x[: 10: 2]
(0, 4, 16, 36, 64)
```

- tuple的元素可以是可变对象
  - tuple的元素不可变, 但是元素指向的对象可变

```
>>> x = (1, 2, [4, 5])
>>> x[2] = [5, 5]
TypeError:  'tuple' object does not support item
  assignment
>>> x[2][0] = 5
>>> x
(1, 2, [5, 5])
```

- 除了索引(下标)和切片访问外, 还支持序列的其它基本操作
  - 连接和重复　e.g. `(1, 2) + ('a', 'b'), (1, 2) * 2`
  - 成员关系操作:
    - in/not in　e.g. `1 in (1, 2),'1' not in (1, 2)`
    - count(x): x出现的次数　e.g. `(1, 2).count(1)`
    - index(value, [start, [stop]])　value在指定范围第一次出现的下标, 找不到时抛出异常ValueError. e.g. `(1, 2).index(1)`
  - 比较运算: < <= == != >= >

内置函数对元组的操作

- sorted(iterable): 排序后返回新列表
- len(iterable), max(iterable), min(iterable): 长度, 最大值, 最小值
- sum(iterable): 数值元素的和
- enumerate(iterable): 返回枚举对象(包含下标和值的元组)
- zip(iter1, iter2…): 返回zip对象(多个可迭代对象中对应位置元素组成的元组)

3.2.2 元组 *vs* 列表

元组中的元素一旦定义就不允许更改

- 没有append()/extend()/insert()等方法, 无法向元组添加元素;
- 没有remove()/pop()方法, 无法对元组元素进行del操作.

tuple( )函数可接受一个列表参数, 返回一个包含同样元素的元组; 而list( )函数可接受一个元组参数并返回一个列表.

- 从效果上看, tuple( )冻结列表, 而list( )融化元组.

元组的元素不可变，但元素所指向的对象可变.

```
>>> x = ([1, 2], 3)
>>> x[0][0] = 5
>>> x
([5, 2], 3)
>>>x[0].append(8)
>>>x
([5, 2, 8], 3)
>>>x[0] = x[0] + [10]
TypeError:  'tuple' object does not support item
  assignment
```

错误说明：
列表的+运算会创建一个新的列表，
对x[0]的赋值就是尝试修改元组
元素的指向⇨不允许，报错！

元组的优点

- 元组的速度比列表快: 若定义了一系列常量值, 仅需对它们进行遍历(读操作)⇨使用元组较列表更佳.
- 元组对不允许改变的数据进行“写保护”, 使代码更加安全.
- 元组可用作字典键; 而列表不可, 因为可变.
- 很多内置函数的返回值是元组——必须对元组进行处理.

3.2.3 序列解包(sequence unpacking)

- 对多个对象引用(变量等)同时赋值 LHS = RHS
  - LHS可以是变量名, 或是通过下标/切片描述的多个list元素
  - LHS可通过圆括号、方括号组织, 通过逗号分割

```
>>> (x, y, z) = (False, 3.5, 'exp')
>>> x, y, z = (False, 3.5, 'exp')
>>> x, y, z = False, 3.5, 'exp'
>>> x, y, z
(False, 3.5, 'exp')
```

- RHS可以是任何可迭代对象，包括tuple/list/dict/range/str等⇨逐个取该序列的元素赋予左边对应位置的对象

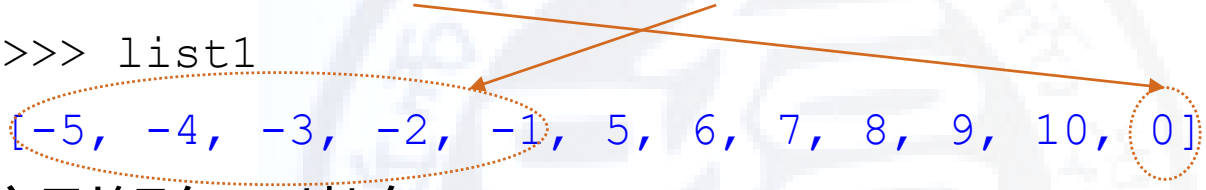
```
>>> a, b, c = [1, 2, 3]
>>> a, b, c
(1, 2, 3)
>>> a, b, c = 'abc'
>>> a, b, c
('a', 'b', 'c')
```

- 除了带星号的对象引用(*seq)外，要求RHS与LHS有相同数量的元素.
  - 带星号的对象最多出现一次，该对象前后的变量一一对应赋值后，剩余的值转换为list然后赋予该引用.

```
>>> a, *b, c = range(1, 7)
>>> a, b, c
(1, [2, 3, 4, 5], 6)
```

- 序列解包中, LHS可以是列表元素或者切片

```
>>> list1 = list(range(12)) # list1=[0,1,2,3,4,5,6,7,8,9,10,11]
>>> x, y, list1[-1], list1[0:5] = 3, 4, 0, range(-5, 0)
>>> list1
[-5, -4, -3, -2, -1, 5, 6, 7, 8, 9, 10, 0]
```



- 序列解包可嵌套

```
>>> a, [b, (c, d)] = 1, ['hello', ('Steve', 'Lee')]
>>> a, b, c, d
(1, 'hello', 'Steve', 'Lee')
```

3.2.4 生成器推导式

与列表推导式相似, 使用'()'而不是'[]'.

结果是一个生成器对象, 可转换为列表/元组

- 可使用生成器对象的__next__()方法进行遍历, 也可采用内置函数next(obj)
- 遍历时, 元素指针往前移动. 遍历完最后一个元素后, 下一个元素将抛出异常StopIteration.

```
>>> g = ((i+2)**2 for i in range(3))
>>> g
<generator object <genexpr> at 0x000001B3E1AD4518>
>>> tuple(g)
(4, 9, 16)
>>> tuple(g)
()

>>> g = ((i+2)**2 for i in range(3))
>>> next(g)
4
>>> next(g)
9
>>> next(g)
16
>>> next(g)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    next(g)
StopIteration
```

[例] 生成器推导式

- 毕达哥拉斯三元组: 存在{x, y, z}, 0<x<y<z, 使得x^2+y^2=z^2. 求前10个毕达哥拉斯三元组.

```
#方法一
pyt = [(x, y, z) for z in range(100) for y in range(1, z)
      for x in range(1, y) if x*x + y*y == z*z ]
firstN_pyt = pyt[:10]
print(firstN_pyt)
#方法二
pyt = ((x, y, z) for z in range(100) for y in range(1, z)
      for x in range(1, y) if x*x + y*y == z*z )
firstN_pyt = [next(pyt) for x in range(10) ] #依然需要使用列表推导式
print(firstN_pyt)
```