**第五章 序列**

在Python内置的类型中，有一种类型是有序排列的，其可以通过下标的方式来标注一个 或者若干个成员,这种类型叫作序列(sequence )。本章我们将针对元组(tuple )、列表(list) 和range这几种序列进行详细讲解。

*1* 袱谀俳么是序殂

在Python中,序列是一组有顺序的元素集合,严格来说,是对象的集合，但鉴于我们还 没有引入“对象”的概念,暂且说是元素。序列可以包含一个或者多个元素，也可以没有任何 元素。之前,我们学过的数据类型,基本都可以作为序列的元素。

由于序列是一个有序集合,所以我们可以通过其位置获得他们的元素。在Python中,序 列中的元素是通过索引提取的。

假设某个序列N有7个元素，那么这7个元素对应的索引如图54所示。

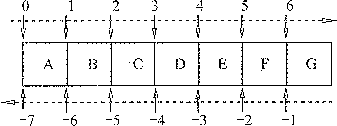


图5』序列元素的索引

从图54中可以看出,元素的索引可以是正数也可以是负数。

如果是正数索引，那么元素A对应的索引是0,元素B对应的索引是1，元素C对应的索 引是丄 依此类推，元素G对应的索引是6。

如果是负数索引,那么元素G对应的索引是元素F对应的索引是-2,元素E对应的 索引是-3,依此类推，元素A对应的索引是-7。

为了大家更好地理解元素的索引，我们可以将序列中的每一个元素看作是一个盒子，索引 看作是盒子的入口。不管索引是正数还是负数，盒子的入口都是位于盒子左侧，只有打开盒子 左侧的入口，才能拿到盒子里面存储的东西。

*凯2*通膈序列操作

所有序列类型都可以进行某些特定的操作。这些操作包括索引、切片、序列相加、乘以及 检查某个元素是否属于序列的成员。除此之外，Python还提供了计算序列长度、找出最大元素 和最小元素等内建函数。

5.2J ([])

序列的元素都有索引，索引从0开始递增。如果使用负数索引，Pythoii会从右开始计算, 并且T代表的是最右边的元素索引。

下面是索引的示例代码:

# string 是一个字符 串

>>> sting = nitcast"

>>> sting[0]

>>> sting[1]

p

»> sting [ -1 ]

»> sting [ -2 ]

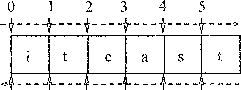
! s s

#获取字符串string中的第1个元素

#获取字符串string中的第2个元素

#获取字符串string的最后1个元素

#获取字符串string的倒数第2个元素

上述代码中,sting是一个字符串,我们通过索引的方式, 依次获取出字符串中的每个字符，由此说明，字符串是一个由字 符组成的序列，如图5・2所示。

-6 -5 -4 -3 -2 -1

艮2卷 （ [i：j]） 图缶2使用索引获取序列元素

除了可以使用索引访问某个元素外，还可以使用切片操作来访问一定范围内元素，切片操 作需要两个索引来作为边界,例如，假设S是一个序列,那么S[i：j]获取到的是索引从i到j-l 的元素。

下面是使用切片的示例代码：

»> seque = [1^2,3^4,51

»> seque [2:4]

[3, 4]

上述代码中，seque是一个列表,它是一个序列类型,当使用切片useque[2:4]J，获取一定 范围内的元素时，得到的结果是索引从2到4的元素。

示例图如图5-3所示。

同理，使用负数索引来获取最后三个元素的方式如下所示。

»> seque = [ 1,2 7 3,4,5 ]

*〉〉）*seque[-4:-1]

[2, 3, 4]

示例图如图5-4所示。

[开始位置：结束位置] 索引值指的是刀要在哪里“切下‘‘

图 5-3 seque[2:4]操作

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | 2 | 3 | 4 | |
|  |  |  |  |
|  | 1 | *槌曇京* |  | |  | 5 |
|  |  |  |  | |  |  |
|  | -5 - | -4 - | -3 - | | -2 -1 | |

[开始位置：结束位置]

索引值指的是刀要在哪里“切下'

图 5-4 seque[-4:-l]操作

如果切片中最左边的索引比它右边的晚出现在序列中，结果就是一个空的序列。例如：

>>> seque = [1,2,3,4Z 5] # seque是一个列表,属于序列类型

»> seque [2:1] #由于左边的索引出现的晚于右边的序列,结果为空序列

[]

»> seque [-1: -4] #由于左边的索引出现的晚于右边的序列,结果为空序列

[]

如果希望使用切片获取的元素包含最后一个元素，可以将对应切片的结束索引设置为空,

示例如下:

»> seque = [lz 3^ 5]

>>> seque[3:]

[4,5]

示例图如图5-5所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *d* | 1 | 2 2 | J *1* | 4 |
|  | 1 | 2 | 3 | *W蘇* | 一…1 |
| 奇 |  |  |  |  |  |
|  | 5 - | 4 - | 3 - | -2 - | [1 |

#获取索引为3到最后的元素

[开始位置：结束位置] 索引值指的是刀要在哪里“切下"

图 5-5 seque[3:]操作

同理，如果从第一个元素开始切片9那么将对应切片的开始索

引置为空，示例如下:

>>> seque = [ 1,2,3,4,5]

»> seque [ : 3 ] #获取前三个元素

[1, 2, 3]

示例图如图缶6所示。

当然，如果切片的开始索引和结束索引都为空，则表示包含整 个序列,示例如下:

>>> seque = [ 1,2,3,4,5]

>>> seque [ : ] #获取整个序列

丄 2, 3； 4, 5]

示例图如图缶7所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ~1 |  |  | 界一 r |  |  |
|  | *1曇h* | *饗蒙* |  | 4 | 5 |
|  |  |  |  |  |  |

[开始位置：结束位置] 索引值指的是刀要在哪里“切下'

0 12 3 4

-5 -4 -3 -2 -1

图 5-6 seque[:3]操作

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | ―一 r | 1 r ~ | | *j ►* |
|  | *号鶴；* |  | 3 | *?繼§* |  |
|  | 1 7 |  |  |  |  |

[开始位置：结束位置] 索引值指的是刀要在哪里“切下” 0 12 3 4

-5 -4 -3 -2 -1

[気"小提示： 图5・7 seque[:]操作

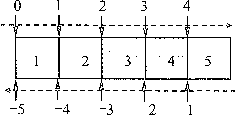
在Python中，有一种叫作“语法糖”的东西，它通常指的是一种便捷的写法，可以给我

们的开发带来方便。编译器不仅会帮我们做相关的转换，而且可以提高编码效率，在性能上也

不会有损失。例如，上述切片的三种简写方式，就是Python序列进行切片的语法糖。

5a2D3 费援（[i：j：k]）

在进行切片的时候开始和结束点需要进行指定(显式或隐式)，而另一个参数——步长 (Mplength)通常都是隐式设置的，默认值为1。具体格式如下所示。



[起始索引：结束索引：步长]

上述格式中，如果步长为1,切片操作就是按照这个步长逐个遍历序列的元素彝然后返回 开始和结束点之间的所有元素。例如，下面几个表达式输出的值是相等的： #获取索引0-5 （不包含5）的元素，步长为1

>» seque = [ 17 2z 3f 4f 5] »> seque [0:5:1]

[lz 2, 3, 4, 5]

>>> seque[0:5]

[lz 2r 3, 4, 5]

»> seque [::]

[lz 2, 3, 4, 5]

#获取索引。〜5 （不包含5）的元素，步长默认为1

#默认步长默认为1

如果步长被设置为比1大的数，那么就会跳过某些元素。示例如下

>>> seque = [12 7 3 7 4 7 5]

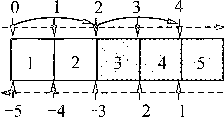
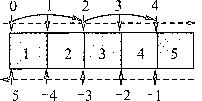
>>> seque [ : : 2 ] #获取步长为2

图 5-8 seque[0::2]和 seque[::2]操作

丄 3, 5]

示例图如图5-8所示。

如果步长为负数，表示序列从右向左获取元素，此时, 两个边界的意义实际上发生了反转。例如,seque[4:l:-2]以 反转的顺序获取了索引从2到4的元素。示例如下:

»> seque = [17 2’ 3r 47 5]

»> seque [4 :1: -1 ] #逆序获取序列元素

[5, 4； 3]

示例图如图5・9所示。

图 5-9 seque[4:l:-l]操作

小提示:

使用步长操作序列时，步长值可以是正数，也可以是负数，但是，步长值不能为0。因为 一旦步长为0,那么序列操作是无法往下进行的。

5.2.4刿新巢命冠鬣暴齎屬著麝測

Python提供了两个成员运算符in和not in,它们可以判断某个元素是否属于序列。如果使 用运算符in来判断元素，存在则结果为True,否则为Falseo反之，如果使用运算符not in来 判断元素，不存在则判断结果为True，否则判断结果为False。使用成员运算符判断序列的格 式如下所示：

元素[not] in 序列

接下来，看一个例子：

»> sting = "hello itheima"

»> nh° in sting#因为字符串sting中存在h,所以使用in运算符的判断结果为True

True

»> not in sting #因为字符串sting中不存在w,所以使用not in的判断结果为True True

其实，hi和硕血都属于成员运算符，它们的功能是一样的，并且其返回值只能是True或 者False,大家选择其中一个使用即可。

E.2.5 屬魏髓li" 「 •

在Pythoti中，可以使用加号+对两个类型相同的序列进行连接操作，语法格式如下旨 序列1+序列2

例如：

»> [lr 2, 3] + [4f 5, 6]

丄 2, 3, 4, 5, 6]

»> \* hello ? + ' itheimaf

\* helloitheimaJ

需要注意的是，使用加号将两个相同序列连接后，生成的是一个新的序列。相加的两个序 列必须是同一种类型，比如，字符串和列表是不可以使用加号连接的，否则会出现下列错误。

»> [1,2,3] + !456s

Traceback (most recent call last):

File "<stdin>", line 1f in <module>

TypeError: can only concatenate list (not "str") to list

5.2.®麝飆尊簽享帼藜 •

使用数字花乘以一个序列会生成新的序列，在新的序列中，原来的序列将被重复殒次。语 法格式如下：

序列\*n

II必须是一个整数,像连接操作符一样,该操作符返回一个新的包含多份原对象拷贝的对象。 例如，使用数字3乘以一个长度为3的序列，结果生成的新序列长度会变为9O

>>> demo\_list = [1,2,3] \*3

>>> demo\_list

[1,2,3,1,2,3,1,2,3]

亂ZJ it篡療列煖麗 ,

Python提供了一个len()函数用于计算序列的长度，并且返回值是一个整数。使用1印()函 数十十算序列长度的方式非常简单，示例如下：

>>> demo\_list = [1f 2f 3, 4r 5]

>>> len (demo\_list) #使用丄en函数计算序列的长度

5

1皿()函数是PythCMl内置的一个函数，其用法虽然简单，但是在后面序列的操作中会被经 常用到，比如使用while遍历序列时，可以通过1皿()函数计算序列长度歹从而决定wh论循环 遍历的次数。

5.2.8護W刿餉憂失冠囊俺靂辨冠繁’•

min()函数和max()函数也是Python的内置函数，它们用在序列中，分别用于返回序列中 最大和最小的元素。

示例如下:

>>> number = [23, *12 r* 100^ 432 f 234]

>>> max (number) #使用max函数计算序列的最大元素

432

»> min (number) #使用min函数计算序列的最小元素

12

亂2⑨ 賣懿冠薰尷麝刿軍證勰» 置

index。函数可以查找某个值在序列中第一次岀现的位置，例如：

s . index (x [, i [, j ] ] ) #这里的方括号表示i和j是可选的

上述代码中，s表示的一个序列，X表示的是要查找的元素。可选参数i、j表示序列查找 的起始索引和结束索引。如果在序列中查找到指定的元素，返回的是元素第一次岀现的位置。 示例如下：

>>> demo\_list = [1f 2 7 3 r 4 r 5 ]

>>> demo\_list» index(37 05)

2

但是，如果在序列demo list中找不到x元素,index会抛出ValueError异常。

»> demo\_\_list = [1Z2Z37475]

>>> demo\_list.index(67 07 5)

Traceback (most recent call last):

File "<stdin>° r line 1z in <module>

ValueError: 6 is not in list

5.2J©毓谤漏鬣種療飒闢噩HIS痍灘 .

如果希望统计序列中某个元素出现的次数，可以使用coirntO函数来实现，例如，统计x 在序列s中出现的总次数，代码如下：

s.count(x)

又如，统计列表demo\_list中出现3的个数，示例如下：

»> demo\_list=[l7273^17372z3]

>>> demo\_list.count(3)

3

5O 3可变序列——殂表(1st)

「毙鼠1緋忽暴刿震' 「 ’ " 「 '

在Python中，列表是一个可变序列；它通常用于存储相同类型的数据，当然也可以存储 不同类型的数据。

Python中的列表表现形式有点像其他语言中的数组，列表中的元素使用方括号包含，元素 之间使用逗号分隔，示例如下：、

>» ['次, 顷,Jc\ M\* , ?e!]

[\* a \* z ?b ! , s c ! J d5 \* e ?]

»> [1, 2, 3, 4, 5, ?hello?] ,

[1, 2, 3, 4, 5Z ^elloM '

在前面的例子中,我们已经多次使用到了列表,它的强大之处不言而喻。之所以列表是可 变的，是因为列表中存储的内容是可以改变的。在后面的小节，我们会针对列表的创建、列表 的操作进行讲解。

艮乳2测褰餉劎篷商就

Python提供了多种列表创建方式，具体如下:

牝値用』对有齬号創建『个窒的刻表 一 • .

示例如下:

»> 口

[]

2使用方捅号，願逗畳分隔箱秦 ― ’

示例如下:

»> [ Taf z fb\ f cr ]

[Ja\ 5bf, \*c?] '

1. 使用类型构造函数创建列暴

Python提供了 list函数可以创建列表，语法格式如下：

list ( [iterable] ) #这里的方括号表示iterable是可选项

上述格式中，参数iterable是可选的，如果没有指定参数，那么使用廉函数创建的列表将 会是一个空的列表。

1. 使用列表發导式創建列畿 .

通过列表推导式，我们可以直接创建一个列表。具体格式如下：

[x for x in it㊀rable]

在第3种和第4种创建方式中，iterable表示可迭代的对象，这个对象可以是序列、支持 迭代的容器或者一个可迭代对象。如果迭代的iterable已经是一个列表，那么会复制一份并返回， 类似于进行iterable[:]操作。

接下来，使用上述四种列表的创建方式创建4个列表，具体代码如下：

#使用一对方括号创建一个空列表

»> list\_01 = 口

»> list\_01

[]

#使用方括号创建列表，元素使用逗号分隔

»> list\_02 = [ 1 北京! , ?天津? . 5 河北,]

>>> list\_02

[J北京f , ?天津?*，* \*河北']

#使用丄ist函数创建列表，函数的参数为空

>>> list\_03 = list ()

>>> list\_03

[J

#使用list函数创建列表，函数的参数为一个字符串

>>> list\_04 = list(? abc ?)

>>> list\_04

[,aS fb?z fc?]

#使用列表推导式创建列表

»> li=[l,273,4]

>>> list\_05 = [x\*2 for x in li]

»> list\_05

[2, 4, 6,8]

需要注意的是，使用列表推导式创建列表时，由于受到内存限制，列表容量有限，而且， 假如创建一个包含100万个元素的列表，如果我们仅仅访问前面几个元素，后面绝大多数元素 占用的空间都白白浪费了*，*会造成存储空间的极大浪费。在后面的学习中，我们会介绍一种机制, 叫作生成器，它可以不必创建完整的序列，而是在循环的过程中不断推算后续的元素，从而节 省大量的空间。

5.3.3巍骚藻傕籍■ •

如果想改变列表中某个元素的值，只需要通过索引对这个元素重新赋值即可。重新赋值后 的列表，其包含的元素会发生变化，示例如下：

>» list\_demo = [lr 2,3,4,5, ? a \* , !b \* ]

>>> list\_demo[5] = !aa J

>>> list\_demo

[lz 2, 3, 4, 5, ?aa% ?b!]

需要注意的是，在通过索引为元素赋值的时候，必须保证列表的索引是有效的。即索引不 能超过(列表长度-1),否则会报hidexError异常,表示索引超出了列表的边界。

>>> list\_demo[10] = 10

Traceback (most recent call last):

File o<stdin>°, line 1, in <module>

IndexError: list assignment index out of range

2. W^[:J .'

切片的功能非常强大，它可以对列表的元素进行大范围操作，例如，使用切片对列表多个 元素进行替换，示例如下：

>>> name = list(\* peter J)

>>> name

[,p \ s e ? 7 \* t17 \* r !]

>>> name[1:] = list(1ython \*)

>>> name

[?o\ "]

上述代码使用切片对原先的列表元素进行了大范围替换。可能有的人会认为，逐个使用索 引也可以对这些元素进行替换。只不过麻烦一些。其实，切片还有其他很多功能，例如，我们 可以使用切片对原有的列表元素进行替换，示例如下：

»> number = [ 1, *3f 5r* 7 z 9]

»> number [: : ] = [2, *4, 6f* 8, 10]

>» number

[2, 4, 6, 8, 10]

感兴趣的读者，可以自己尝试使用步长大于1或者负数索引来感受切片在列表中的强大之 处吧。

3.成员运算符(inu not in)

我们可以检查一个对象是否是一个列表中的元素。检查方式比较简单，只需要使用也或者 硕也来判断即可。接下来，我们看一段代码：

>>> demo\_list = [12, [1, 5 a \*]z \* color \*]

>>> \* color1 in demo\_list

True

>>> \* a1 in demo\_list

False

>>> [1, 5 a1] in demo\_list

True

大家注意看上述代码，其中\*并不属于demojisto因为M本身不是demojist的一个成员， 而是demo\_list[l]的一个成员,demojist [1]也是一个歹!j表类型。

连接操作符允许我们把多个列表对象合并在一起，只要这些要合并的列表属于同一种类型 即可。示例如下：

»> numl = [100, 5 age 1, [ sa \ 'b\* ]]

>» num2 = [ 1 name ' z 2 + 5 j , [ 1, ' c 1 ]]

>>> num = numl+num2

>>> num

[100, \*ageS [ ?a\ fb! ] , \* namef, (2+5j) , [1,"]]

上述代码中，使用连接操作符+将列表numl和num2合并,返回的num是一个新的列表。 需要指出的是，使用连接操作符+只能将一个列表的内容添加到另外一个列表中，它不能将元 素直接添加到列表中，例如，下面的代码会报错。

>> num + 12

Traceback (most recent call last): File "<stdin>n, line 1r in <module>

TypeError: can only concatenate list (not nint") to list

当试图将12添加到列表num中，程序会报TypeError0这是连接操作符+左右两边类型不 一致导致的。如果希望在列表中添加新的元素，可以使用append或者印tend方法实现，这两 个方法将会在下面的小节讲解。

氮重复繰作膂(勺 ’

重复操作符可能更多地应用在字符串类型中。不过，列表和元组跟字符串同属序列类型， 这里，我们有必要讲解一下重复操作符在列表中的使用。

下面通过代码来演示重复操作符在列表中的使用效果。

#空的列表

>>> number1 = [] \* 3

>>> number1

[]

#包含一个元素的列表

»> number2 = [ 1 ] \* 3

>» number2

[1, lz 1]

#包含多个元素的列表

>>> number3 = [1f *2f* 3] \* 3

>>> number3

[1. 2, 3, lz 2, 3, 17 2g 3]

#嵌套列表

»> number4 = [ [ ?a\ ?b! ] , [ \*c\ ，d']] \* 3

>>> number4

[[5aS !bf] z !dJ ] , [Ja\ ['c\ ?d? ] , ['a' ,b! ] , [!c\ 9']]

上述代码中，使用重复操作符操作应用在一个列表后，返回的仍然是一个列表，不同的是, 新的列表元素个数增加了，并且增加的元素都是对原先列表元素的重复。

5.3.4刿鶴饗鑿緡美圖纖 . \ "

列表作为序列的一种类型，它支持序列的通用操作。接下来，针对列表支持的函数进行逐一讲解。

1 ten(). . . . ： •

对字符串来说，len(s)返回字符串的长度，就是字符串包含的字符个数。对于列表来说， 知()返回的是列表的元素个数。示例如下：

>>> number\_list = [ 1z 2,3,4 7 5]

>>> len(number\_list)

5

>>> a = list ( \* abcdef ?) '

>>> len (a)

6

2h max() IH mm()

max(s)和min(s)在字符串中作用不大，因为这两个函数只能找出字符串中“最大”和“最 小”的字符。这里的“最大”和"最小”是按照词典顺序排列的。对于列表来说，使用max() 和min()函数判断列表中的最大值或最小值时，该列表中的元素必须是相同的类型，否则，对 于混合类型元素的列表，这两个函数是无法准确返回最大值和最小值的。

使用max().和min()函数操作元素是相同类型的列表。

»> list\_str = ['bed' !abcf , ! adef ]

>>> max(list\_str)

\* bed \*

>>> min(a)

s abc!

使用max()和min()函数操作混合类型元素的列表。

>>> demo\_list = [!bcd? 7 1abc v f !ade1f 123]

>>> min(demo\_list)

Traceback (most recent

File °<stdin>nF line

TypeError: unorderable

call last):

17 in <module>

types: int () < str()

3. Iist()

list(seq)函数用于将一个可迭代对象(比如另一个序列)转为列表。该函数将可迭代对象 作为参数，通过浅拷贝的方式创建一个新的列表。例如，将字符串转为列表的方式如下所示。

>>> word = , hello ,

#将word转为列表list\_l和列表list\_2

>>> list\_l = list(word)

>>> list\_2 = list(word)

#判断列表list\_l和list\_2的值是否相等

>>> == list\_2

True

#判断列表和lis\_2是否是同一个列表

>>> list\_l is list\_2

False

从上述代码可以看出，执行list()函数将字符串转为两个列表后，虽然两个列表的值是相 等的，但两个列表却不是同一个列表。

VS注意：在列表的相关函数中，Python 2存在一个cmp()函数，该函数用于比较两个列表

的元素大小。但是，在Python 3中，cnip()函数已经被废弃，如果我们要比较两个列表元素大小， 可以使用表达式(a > b ) - ( a< B )当作cmp(a,b)。

53.5魂裹羹醴帼美商1 憲

前面讲解了列表类型的相关函数，接下来，我们针对列表类型的相关方法来讲解。列表 类型方法的使用，与列表函数非常类似歹只不过列表方法需要通过列表对象来调用，调用方 式如下：

列表对象•方法(参数)

列表类型的相关方法如下：

1 Hstappend(obj)

在列表末尾添加新的对象，示例如下：

>>> listl = [1r 2, 3f 4]

»> listl. append (5)

>>> listl

[lz 2, 3, 4, 5]

» list2 = [ [lb [2] , [3]]

>>> list2[0].append(3)

>>> list2

[[lz 3L [2L [3]]

上述代码中,列表listl使用append方法在末尾添加了一个新元素,列表list2使用append 方法对第一个元素添加新对象，添加的新对象同样在末尾。

接下来，再看一个例子，代码如下：

>>> demo\_list = [[]]

>>> demo\_list

[口， [], 口]

#使用操作符\*3对列表进行重复操作

92 w Pythoo靈戯飜新 毆馨孝Pjthon

>>> demo\_list [ 0 ] . append (3) #对列表第1个元素后追加新对象

>» demo\_list

[[3] , [3] , [3]]

上述代码中,大家可能会产生疑惑,为什么最后使用"demo\_list[0].append(3)n追加元素后, 列表 demojist 的结果为“回,[3], [3]]” ,而不是"[[3],[],[]]"呢？

这里要给大家说明的是，当使用重复操作符\*对列表增加元素后，列表中的元素是同一个。 这点可以使用id函数来验证。

>>> demo\_list = [[]] \* 3

>>> id(demo\_list[0])

4320231560

>>> id(demo\_list[1])

4320231560

>>> id(demo\_list[2])

4320231560

从上述代码可以看出，列表中的每个元素的id都是4320231560,这样我们就没法实现多 个列表分开存储的需求了。

2. listxountfobj)

count方法用于统计某个元素在列表中出现的次数o示例如下:

»> demo\_list = *[1,2 r 2] [[1 r 2] f 2]]*

»> demo\_list. count (1)

1

从上述代码可以看出，虽然在列表demojist中有很多1,但列表的元素值为1的只有一个。

& list.extendfseq)

extend方法可以在列表末尾一次性追加另一个序列中的多个值。换句话说,该方法实现的 是使用新列表扩展原来的列表。例如:

»> list\_str = [ J a1 , 5 b ' 7 \*c\*]

>>> list\_num = [ 1? 2 f 3]

>>> list\_str.extend(list\_num)

>>> list\_\_str

[sa\ b, !c\ 1, 2, 3]

上述操作看起来像是将两个列表进行了合并，效果与使用“+”相同。实际上歹两者是有 区别的。extend方法修改了被扩充的列表,也就是列表list\_str被扩充了，但是,如果使用“+” 合并列表，合并后的列表是一个新的列表，参与合并的两个列表值是不会发生变化的。

»> list\_str = H ?b! f e ]

>>> list\_\_num = [ 1 f 2 z 3 ]

>>> list\_str + list\_num

1, 2, 3]

>» list\_str

[5a\ ,b\ 'c，]

»> list\_num

[1, 2, 3]

4D fetJndex(obj)

index方法用于从列表中找出某个值第一个匹配项的索引位置。 示例如下:

>>> list\_str = [\* you \*r \* Hello \* f \* good \*, \* What5 f \* itheima\*] >>> list\_str.index(\* itheima ?)

4

>>> list\_str.index(1itcast1 )

Traceback (most recent call last):

File "<stdin>"r line 1f in <module> ValueError: Jitcast\* is not in list

上述代码中，当使用index方法寻找到指定元素时，会返回该元素的索引位置，反之，如 果没有找到匹配的值，则会报VahieErnr异常，提示没有在列表中找到这个值。

5- listJnsert(index5 obj)

insert方法用于将对象插入列表的指定位置。示例如下:

>>> names = [\* baby1f \* Lucy? z \* Alise1]

>>> names.insert*(2 r* 1 Peter')

>>> names

[v babys z \* Lucy? e ? Peter 5 z \* Alise \*]

上述代码中，使用insert方法将一个新元素!peterJ添加到索引为2的位置，相当于在列表 中新增了一个元素。

6» lisLpop(obj=listFH)

POP方法可以移除列表中的一个元素9默认移除的是最后一个元素，并且返回该元素的值。 示例如下:

>>> numbers = [1r 2, 3f 4f 5]

>>> numbers„pop()

5

>>> numbers

[lz 2, 3, 4]

>>> numbers.pop(1)

2

>>> numbers

[1, 3, 4]

从上述代码可以看出，如果pop方法不指定参数，那么列表的最后一个元素会被移除，如 果指定了要移除的元素的索引,那么该索引对应的元素则会被移除。

*7.* SistBremove(obj)

remove方法同样用于移除列表中某个值的第一个匹配项。示例如下：

>>> chars = [sh% \ ,l5f feM

>>> chars.remove(\* e \*)

>>> chars

[fh\*, P, 'I', ?e\*]

>>> chars.remove(? y \*)

Traceback (most recent call last):

File "<stdin>"f line 1, in <module>

ValueError: list.remove(x): x not in list

从上述代码可以看出，使用方法移除列表元素时，如果列表中有多个匹配的元素，只 会移除第一个匹配的。但是，如果使用remove方法移除的值不存在，那么程序会报ValueError, 提示列表中不存在要移除的值。

8a Ifet.reverseO

reverse方法用于反向列表中元素,也就是对列表的元素进行逆序,该方法只改变列表元素 的顺序，不返回任何值。

示例如下:

>» numbers = [1,2,3’4’5]

>>> numbers.reverse()

>>> numbers

[5, 4, 3, 2, 1]

9b list.sortO

sort方法用于对列表进行排序,该方法是列表特有的,其语法格式如下:

sort(\*, key=None, reverse=None)

在上述格式中,sort方法接收两个参数,其中参数key用于指定每个列表元素的比较键, 例如,key=str.lower,如果不指定或者为None,则表示不计算键的值,直接对列表元素排序; 参数reverse是一个布尔值，默认情况下，参数reverse的值为False。如果设置为True,列表元 素会进行排序，并对排序后的结果进行颠倒，其效果类似于先对列表进行sort,后进行此 操作。

>>> num = [23^12 z 37 6711,8]

>>> num.sort()

>>> num

[3, 6偿京 11, 12, 23]

>>> numo sort(reverse=True)

>>> num

[23, 12, 11, 8, 6, 3]

需要注意的是，使用sort排序后的列表会覆盖原列表(也就是说，排序前后的列表是同一 个id),该方法不会返回一个新列表。示例如下:

»> num = [23,12,3,6,11,8]

»> id (num)

4320231624

>>> num.sort()

>>> num

[3, 6, 8, 11, 12, 23]

»> id (num)

4320231624

实际上，还有一个sorted函数可以排序，与sort方法相比，该函数会返回一个新的列表， 示例如下：

»> num = [23,12,3, 6,11,8]

>>> new\_num = sorted(num)

>>> new\_num

[3, 6, 8, 11, 12, 23]

I”多学一招：可变序列支持的操作

..

在通用序列操作中，大多数操作都是“查询”类的操作。这里，我们来讲一下可变序列支 持的“增” “删”.改”操作。

为了方便讲解可变序列支持的操作，这里，我们先预定义几个符号，这些符号及相关说明 如表54所示。

|  |  |
| --- | --- |
| 表5-1定义的符号及其含义 | |
| 符 号 | 表示的含义 |
| S | 表示一个可变序列类型实例 |
| t | 表示任何一个可迭代对象 |
| X | 表示序列S中满足条件的任何类型的元素(例如，bytearray只接受满足0 <= x <=255约束的整型值) |
| 可变序列支持的操作及其相关结果的相关说明，如表5.2所示。  表5-2可变类型支持的操作及相关结果说明 | |
| 操 作 | 结 衆 |
| s[i] = *x* | 将序列$中小标为i的元素用x替换 |
|  | 将序列s中从i至!]j的切片用可迭代对象t的内容替换 |
| s[i:j:k]=t | s[i：j：k]中的元素用可迭代对象t的内容替换 |
| s \*= n | 更新序列*s*为s的n次重复的结果 |
| dels[i:j] | 删除序列s中从i到j的切片，等价于s[i:j]=[] |
| del s[i:j:k] | 从序列s中删除s[i:j药中的元素 |
| del s[i:j:k] | 从序列s中删除s[i:j:k]中的元素 |
| s.pop() /s.pop(i) | 获取序列s中下标为i的元素，并从序列s中删除该元素;i默认为-1,即默认删除并返回序列s的 最后一个元素 |
| s.remove(x) | 从序列S中移除第一个等于x(即s[i] — X)的元素§如果x在序列s中不存在，则会抛出ValueError |
| s.clear() | 移除序列s中的所有元素，等价于del s[:] |
| s.append(x) | 将x追加到序列s的末尾，等价于s[len(s):len(s)=闵] |
| s.extend(t) or s+=t | 将可迭代对象t中的元素拼接到序列s的末尾，大部分时候等价于s[len(s):len(s)] = t |
| s.insert(i,x) | 在序列s中下标为i的位置插入X |
| s.copy() | 创建一个序列s的浅拷贝，等价于s[:] |
| s.reverse() | 反转序列*s*中元素的位置，该方法直接对序列s本身做修改(可以节约空间)，不会返回被反转后的序列 |
| 5« 4不可变序列 死缰 | |

元组同列表一样，都属于序列。与列表相比，元组的内容是不可修改的，也就是说，元组 是一个不可变序列。

96 w Pjfthoo褻纖爨鹳 财霉寥Pjthon

5.4J売噩IM篷薦就

创建元组的语法比较简单歹具体如下：

1険願『瓣圜齬蜀創繼『命窒苑缱 •.

与列表不同的是，元组是使用圆括号来包含元素的，创建一个空的元组的方式如下所示。

() #这是一个空的元组

2.創建包念~个遞素的晃缱 • •

在单个元素后跟逗号表示单个元素的元组，下面两行代码创建的都是只包含一个元素的元组。 a, #这是单个元素的元组

(aj #这是单个元素的元组

*3O*創建包倉爹令施素的氟襲

如果元组包含多个元素，那么，多个元素之间需要使用逗号分隔，示例如下：

*aFbf* c #这是一个元组，包含3个元素

(a, bz c) #这是一个元组，包含3个元素

4.使闔襲篷橱造函鐵劎建蠢缱

可以使用tuple()或者tuple(iterable)构造元组*，*其中，参数iterable是一个迭代类型，如果 使用tuple。函数创建元组的时候，不传入参数，那么创建的元组是一个空元组。示例如下：

* 使用 tuple (iterable)函数创建元组 tuple\_str

>>> tuple\_str = tuple ( \*abc1)

>>> tuple\_str

(!a\ •bS P )

* 使用 tuple (iterable)函数建元组 tuple\_list

»> tuple\_list = tuple ( )

>>> tuple\_list

(1, 2, 3)

#使用tuple ()函数创建元组tuple\_null

»> tuple\_null = tuple ()

>>> tuple\_null

0

S.4.2 「

在前面的小节中，我们已经介绍了序列的通用操作，这些通用操作不仅适用列表，同时也 适用于元组。为了避免太多重复信息，我们这里不对元组的每个操作进行讲解，这里只简单演 示一下元组的不可变性和其他独特的特性。

1。不兔许更新死組的元素

由于元组的不可变性，元组是不允许被修改的，否则会报错。

»> number = (1Z2Z3Z4Z5,6)

»> number [2] = 10

Traceback (most recent call last):

File "<stdin>,!r line 1 f in <module>

TypeError: \* tuple 1 obj ect does not support item assignment

2.不灸许删除寿缱聂秦

由于元组的不可变性，删除元组的某个元素是不可能的。当然，我们可以通过重组元组的 方式来去除丢弃的元组元素。

如果希望删除整个元组，可以使用del语句来实现。

»> demo\_tuple = tuple ( \* abc 1 )

>>> del demo\_tuple

>» demo\_tuple

Traceback (most recent call last):

File H<stdin>"f line 1f in <module>

NameError: name , demo\_tuple# is not defined

在上述代码中身adel demo\_\_tuple,?用于删除一个元组。删除成功后，元组本身整个都不存 在了.此时如果再访问，会提示访问的元组未定义。

5d4d3 「

我们讲过创建元组的方式,对于初学Python的同学可能无法理解,为什么单个元素的元 组要在后面加个逗号呢?例如,tuple\_stH!123\)o下面,我们带大家看一个例子:

>>> tuple\_l = (1)

>>> tuple\_2 = (2z 3)

>>> tuple\_l+tuple\_2

Traceback (most recent call last):

File "<stdin>"f line 1f in <module>

TypeError: unsupported operand type(s) for +: 1int\* and \* tuple 1

大家仔细看上述错误信息:"TypeError: unsupported operand type(s) for +: "inf and "tuple"o 原来是Python解释器会把这个元素当作一个算术表达式来处理。例如:

>>> tuple\_num = (1)

»> type (tuple\_num)

<class ? int!>

>>> tuple\_letter = (5 a 1)

>>> type(tuple\_letter)

<class ? str 1>

从上述代码可以看出，如果想声明只有一个元素的元组，如果不加都逗号，Pythoti会 以为我们只是声明了一个int类型的值或者是字符串。为了区分这种情况，Python规定在元 组单个元素后面要添加一个逗号，例如，tuple\_nuni=(l9)9这样，Python解释器才会明白这 是一个元组M

»> tuple\_letter = (! 1 S )

>>> type(tuple\_letter)

<class \* tuple \*>

>>> tuple\_num = (2f)

>>> type(tuple\_num)

<class 1 tuple ? >

前面我们一直强调元组是一种不可变的序列，但是，在特定的情形下，元组也是可变的，

看一段代码：

>>> tuple\_char = (fa?z ,b \ [\*x \ !yj])

>>> tuple\_char[2] [0] = \* c J

>>> tuple\_char[2][1] = !d1

>>> tuple\_char

(!a\ 顿'?d?])

上述元组tuple\_char在定义的时候有3个元素,分别是矶b和一个卩欢们。后来,通过 索引修改元组的元素后，元组的元素内容发生了变化。

下面，我们画图来描述这个元组是如何“变化的”，如图5J0所示。

表面上看，元组的元素确实变了，但其实变的不是元组的元素，而是列表的元素。元组一 开始指向的列表并没有改成别的列表，所以，元组所谓的“不变”是说，元组的每个元素，指 向永远不变。即指向'叭 就不能改成指向b,指向一个列表，就不能改成指向其他类型的对象, 但指向的这个列表本身是可变的！

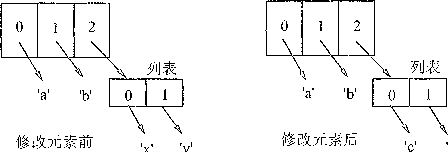


图5-10修改元素前后元组的情况

'鼠5元纸是兎筒号瑚象的戮讥类鹽

任意无符号的多个对象，以逗号隔开，默认为元组。我们看一段代码：

»> 12, 'ab?,34

(12, !ab\ 34)

»> x, y = 2/ 3

*»> xry*

(2, 3)

上述代码中，虽然这些对象都没有使用圆括号来显式表示为元组类型，但是返回的却是元 组类型，这由此可以证明元组是由逗号决定的，而不是圆括号。

5o6不同序刿的梱亙转擁

在前面的章节，我们学过一种类型叫作字符串，其实，字符串也属于一种序列类型，只不 过它专门用于文本操作*。*

在Python中,针对列表、元组和字符串提供了三个内建函数,分别是str(). list()和tuple(), 它们之间的相互转换使用这三个函数，具体示例如下：

>>> word = "abcdef'

#将字符串转为列表

>>> list(word)

[0, ?b\ ?c\ \*dv , #将字符串转为元组

>>> tuple(word)

(va\* , fbS \*cS

?f s)

#将列表转换元组

>>> tuple(list(word))

(fb\ vcS MS

#将元组转为列表

»> list (tuple (word))

[va\* F \*b\ ?cJ , !d? f

列表和元组转换为字符串则必须依靠joili函数，示例如下: #将元组转为字符串

>>> \* \*.join(tuple(word))

^abcdef7

#将列表转为字符串

>>> 'r .join(list(word))

5 abcdef 5

#使用str会将元组转为字符串内容包含

>>> str(tuple(word))

n (\* a \ ?bf , \*cS feJ, \* f!)"

So 7不可变序剜

Tamge

5-7J raog® Wig

range是一个不可变的序列，该序列的元素必须是整数，为什么这么说呢？我们先来看一 下range的使用方式：

range(stop);

range(starts stop[,step])

上述格式中，range的参数必须是整数，其中：

(1 ) start表示起始值,默认为0,例如range(5)等价于range(0? 5)；

(2)stop不可选，表示结束值，它的值不包含在范围内。例如range(09 5)表示的值只有。、 L 2, 3, 4,不包含5。

(3 ) step可选,表示步长,默认为1,例如range(0,5)等价于range(0, 5,1)。

®如果指定step为09则抛出一个ValueError异常。

•当step为正时,range的值由这个公式决定:r[i] = start + step\*i,而i >= 0并且r[i] < stopo

•当step为负时，range的值仍然由该公式决定:r[i] = start + step\*i,但是约束为i >= 0和 r[i] > stop0

这里需要说明的是，在Python 2和Python 3中，如果想输出range范围内的元素，两者是 有区别的O

Python 2 版本:

>>> range(0,5)

[0, 1, 2, 3, 4]

Python 3 版本:

»> range (0^5) range(0, 5)

上述情况的原因如下:

® Python2中,range返回的是一个列表类型。

® Python3中，iwge返回的是一个range类型，它需要通过list（）函数才能将列表中的值输出。 示例如下:

>>> list(range(5))

[0, 1, 2, 3, 4]

>>> list(range(0r 20z 2))

[0, 2, 4/ 6, 8, 10, 12, 14, 16,

>>> list(range(0, -10, -2))

[0, -2, f -6, -8]

>>> list(range(0))

[]

»> list (range (lz 0))

[]

#代表0~5 （不包含5）

#代表0〜20 （不包含20）,间隔为2

18]

#代表0~~10 （不包含T0）,间隔为2

#代表空的序列

#代表空的序列

#判断5是否在range中

#获取4在range中的索引位置

#计算range的长度

#获取range中的最大值

#获取range中的最小值

#统计range中值为1的元素个数

通常情况下，range。都会使用在循环中，用于控制循环的次数。此时，  
类似forninrange（l510）: 的语句中，不管是哪个版本，都是可以照常使用的。

如果将range用在

5b7q2 rang®

皿ge支持序列的通用操作，除了连接（+ ）和重复（\*）的操作，这是因为range对象只能 表示遵循严格模式的序列，而连接和重复操作会违反这个模式。

range支持的通用序列操作如下所示。

>>> demo\_range = range(0 710)

>>> 5 in demo\_range

True

>>> demo\_range.index(4)

4

>>> len(demo\_range)

10

>>> max(demo\_range)

9

>>> min(demo\_range)

0

»> demo\_range 8 count (1)

1

如果两个范围对象表示的值序列相同，则认为它们相等，可以使用==和!==对它们进行 比较。例如,range（0） = range（2, 1, 3）o

5d7d3 rang®壇刿褰S3施韻搦鰻 . ' .

range相对于列表和元组，最大的好处在于占用的内存固定，且较小，它仅存储start、stop 和step值，并且在需要的时候须才会计算每个条目的值。

如果两个range所代表的序列值一样，那么两个range是相等的。示例如下：

>>> range(0) == range*(2 f 1f* 3)

True

»> range (0 r 3, 2) == range (0, 4 f 2)

True

列表和元组则不尽其然,我们来验证一下:

比较两个列表：

»> list\_01 = [1,2,3]

>» list\_02 = [3f 2,1 ]

>>> list\_01 == list\_02

False

比较两个元组：

»> tuple\_01 = (f a \* z ?b \ \* c \*) >>> tuple\_02 = (\*b \ 1 c \* , J a 1 ) >>> tuple\_01 == tuple\_02 False