第5章 字符串、列表和元组

本章重点内容：字符串、列表和元组的基本概念，字符串、列表和元组的特性，字符串、列表和元组的操作，字符串、列表和元组的常用内建函数。

本章学习要求：通过本章的学习，理解和掌握 Python 中最为常用和基础的三种数据类型，可以使用三种数据类型并配合语句解决一些问题。

5.1 单词本 wordbook 的构建

【例 5-1】 在英语的学习过程中，遇到不认识的单词，可以将其添加到单词本中，用于后续复习使用。要求：打开一个英文文本，过滤掉介词、代词等停用词（去掉长度小于 5 的单词）后，用该文本中的剩余词汇构建初始单词本 wordbook.txt。对于给定文本，遍历其中的词汇，对于长度大于 5 的单词，若其不在当前的单词本中，则把它添加到单词本中。

观看视频

参考代码：

```
file = open('DeclarationofIndependent.txt','r').read()#将文本读出成字符串格式
file.replace(',', '')
file.replace('.', '')
filelist = file.split()                          #以字符串中的所有单词为元素转换为列表
filelist1 = []                                   #构建空列表
for word in filelist:                            #遍历列表中的元素
    if len(word)> 5:                             #如果单词的长度大于 5
        cleanword = word.lower().strip('.,')     #将大写全部转换为小写并去除标点符号
        filelist1.append(cleanword)              #将清洗过的单词添加到列表中
#单词本初始化
wordbook = open('wordbook.txt','w')              #以读的方式打开文本文件
for x in filelist1:                              #对已经清洗过单词元素的列表做遍历
    wordbook.write(x)                            #把单词元素写入单词本
    wordbook.write('\n')                         #每写一个单词的同时,写入一个换行
wordbook.close()                                 #关闭单词本
#更新单词本
s = 'Now we are engaged in a great civil war, testing whether that nation, or any nation so
conceived and so dedicated, can long endure. We are met on a great battle - field of that war. We
have come to dedicate a portion of that field, as a final resting place for those who here gave
their lives that nation might live. It is altogether fitting and proper that we should do this. '
#给定短文本字符串
s.replace(',', '')
```

```
s.replace('.', '')
list_s = s.split()                          #split()将给定的文本字符串转换为列表
clean_s = []
for word in list_s:                         #对列表做遍历,清洗单词
    if len(word)> 5:
        cleanword = word.lower().strip('.,')
        clean_s.append(cleanword)

bookfile = open('wordbook.txt','r').read()  #打开原有单词本文件
wordbookfile = bookfile.split()
print(len(wordbookfile))                    #输出原有单词本的单词数量
for word in clean_s:
    if word not in wordbookfile:
        wordbookfile.append(word)
for w in wordbookfile:
    bookfile.write(w)
bookfile.close()
print(len(wordbookfile))                    #输出新的单词本的单词数量
```

【分析】 本例中使用 open()函数实现对文本文件的打开操作,使用 read()方法将读出的文本文件转换为字符串。单词本的创建和更新过程中,以单词为编辑对象,因此又将字符串以单词为单位作为元素,放入列表序列中进行处理(包括清洗等操作)。列表和字符串是 Python 中重要的序列类型,文件的读写等操作请参见后续章节。

5.2 序列

5.2.1 什么是序列

序列是 Python 中最基本的数据结构。一组元素按照一定的顺序组合在一起则称为序列。序列中每一个元素都用一个数字来表示它的位置,这个数字称为索引。Python 中的索引和 C 或 Java 语言一样,都是从 0 开始,第二个是 1,以此类推。通过索引,序列的每一个元素都可以被访问到。故序列也是 Python 的一种访问模式。

Python 中包含五种内建的序列,包括字符串、列表、元组、buffer 对象和 xrange 对象。其中前三种是最常用的,也是本章的重点。

与普通的数据类型不同,序列不仅可以使用标准类型的操作符,还可以使用自己独特的序列类型操作符。

【例 5-2】 简单的序列操作。

参考代码:

```
>>> astring = 'hello world'                    #字符串序列
>>> alist = list(astring)                      #将字符串转换为列表序列
>>> alist
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']   #每个字母就构成列表的元素
>>> atuple = tuple(alist)                      #将列表转换为元组
>>> atuple
('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')   #元组以小括号为标志
```

```
>>> alist.clear()                                  #清除元素
>>> alist
[]
>>> alist = [1, 5, 6, 8] + [4, 5, 7, 9]            #加号'+'连接序列,而非加法操作
>>> alist
[1, 5, 6, 8, 4, 5, 7, 9]
>>> alist[4:7]                                     #列表的切片操作,左开右闭原则
[4, 5, 7]
>>> astring[0]                                     #输出字符串序列的第0号元素
'h'
>>> alist.index(7)                                 #输出列表中元素7的索引
6
>>> alist[7]                                       #输出列表序列中索引为7的元素
9
>>> sorted(alist)                                  #对列表序列进行排序
[1, 4, 5, 5, 6, 7, 8, 9]
>>> sorted(astring)
[' ', 'd', 'e', 'h', 'l', 'l', 'l', 'o', 'o', 'r', 'w']   #对字符串元素进行排序
>>>
```

5.2.2　标准类型的操作符

标准类型的操作符共分为三种：值比较、对象身份比较和布尔运算(逻辑运算)。

1. 值比较

关系运算符中的值比较运算也可以用于序列对象中。列表的关系运算是按序进行比较的,即从序列的第一个元素开始,逐元素进行比较。只有在元素值相同,顺序也相同的情况下,两个序列才相等,否则以元素先后的大小关系来确定序列的大小关系。常见的值比较运算有>(大于)、<(小于)、>=(大于或等于)、<=(小于或等于)、==(等于)和!=(不等于)。例如：

```
>>> 'father'>'children'
True
>>> [1,2,3,4] == [13,1,3,4,4]
False
>>> (1,2,3) == (1,2,3)
True
>>> [1,2,3,4]> = [1,3,5,6]
False
```

2. 对象身份比较

对象身份比较运算的运算符有 is(是)和 is not(不是)。与等于和不等于的区别是,对象身份比较运算中,比较的是两个序列是否为同一个对象,而等于和不等于比较的是值是否相等。例如：

```
>>> a = [1, 2, 3]
>>> b = [1 ,2 , 3]
>>> a == b
True
>>> a is b
```

```
False
>>> c = [1, 2, 3]
>>> d = c
>>> c is d
True
```

3. 布尔运算

连接序列对象的常见的布尔运算有 not(非)、and(与)和 or(或),其返回值为真或假。例如:

```
>>> ('father' > 'children') and ([1, 2, 3, 4] == [13, 1, 3, 4, 4])
False
>>> ('father' > 'children') or ([1, 2, 3, 4] == [13, 1, 3, 4, 4])
True
>>> not 'father' > 'children'
False
```

5.2.3 序列类型的操作符

序列类型的操作符是指序列所特有的操作符和在序列上产生与其他数据结构不同效果的操作符。根据操作符的优先级从高到低,分别为索引、切片、重复、连接和成员检查。

1. 索引([])

seq[index]的作用是获取序列 seq 中下标为 index 的元素。例如:

```
>>> a = [1,2,3]
>>> a[1]
    1
```

2. 切片([index1: index2],[index1: index2: length])

seq[index1: index2]的作用是获取下标从 index1 开始到 index2(不包括)为止的元素集合。seq[index1: index2: length]的作用是获取下标从 index1 开始到 index2(不包括)为止步长为 length 的元素集合。例如:

```
>>> a = [1, 2, 3, 4, 5, 6]
>>> a[0:2]
[1, 2]
>>> a[0:6:2]
[1, 3, 5]
```

3. 重复(*)

seq * Integer 的作用是获得 seq 序列的 Integer(整型数)次重复。当需要获得一个序列的多份副本时,可以使用重复操作符来简化计算,即将 seq 中的元素重复 Integer 次。例如:

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

4. 连接(+)

seq1+seq2 的作用是连接序列 1 和序列 2,以获得一个按顺序包含两个序列全部元素的新序列,同时两个序列是同一个类型的。例如:

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> [1,2,3] + 'abc'                                    # 连接两个不同序列对象时,会报错
Traceback (most recent call last):
  File "< pyshell # 0 >", line 1, in < module >
    [1,2,3] + 'abc'
TypeError: can only concatenate list (not "str") to list
>>>
```

5. 成员检查(in 和 not in)

object in seq 的作用是判断元素 object 是否在序列 seq 中,若在则返回 True,反之则返回 False。而 object not in seq 作用是判断元素 object 是否不在序列 seq 中,若在则返回 False,反之则返回 True。例如:

```
>>> 1 in [1, 2, 3]
True
>>> 4 in [1, 2, 3]
False
>>> 1 not in [1, 2, 3]
False
>>> 4 not in [1, 2, 3]
True
```

5.2.4　内建函数

Python 中序列有内建的函数,便于精简代码和简化操作。接下来介绍最为常用的几个内建函数(方法)。

1. 类型转换函数

list()、str()和 tuple()三个类型转换函数被用在各种序列类型之间转换。例如:

```
>>> a = 'abcdef'
>>> b = [1, 2 ,3]
>>> c = (1, 2, 3)
>>> type(a)
< class 'str'>
>>> type(b)
< class 'list'>
>>> type(c)
< class 'tuple'>
>>> type(list(a))
< class 'list'>
>>> list(a)
['a', 'b', 'c', 'd', 'e', 'f']
>>> str(b)
'[1, 2, 3]'
>>> type(str(c))
< class 'str'>
```

2. len()函数

len()函数接受一个序列作为参数,返回这个序列的长度(元素的个数)。例如:

```
>>> len('abcdefghijk')
11
>>> len([1, 2, 3, 4, 5])
5
```

3. max()、min()和 sum()函数

这三个函数都是接受一个序列作为参数,返回该序列中最大值、最小值和元素的和,前提是该序列中各个元素之间是可比较的或可计算的。例如:

```
>>> max([1, 32, 12, 14])
32
>>> min([3, 55, 2, 13])
2
>>> min('jihbgyr')
'b'
>>> sum([1,2,3])
6
>>> sum('asd')
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    sum('asd')
TypeError: unsupported operand type(s) for + : 'int' and 'str'
```

4. reversed()函数

reversed()函数接收一个序列作为参数,返回一个以逆序访问的迭代器,需要 for 循环,将迭代器中的元素逐个输出。例如:

```
>>> for item in reversed([1 , 2, 3, 4, 5]):
print(item)
5
4
3
2
1
>>> reversed('sdfgh')
<reversed object at 0x0000021A79DA4D90>
>>>
```

5. sorted()函数

sorted()函数接收一个序列,返回一个有序的列表。其中序列中各个元素必须是两两可比的。其语法规则为:sorted(seq, key=None, reverse=False)。其中,seq 为用于被排序的序列,key 为指定的排序元素,reverse 指定排序规则,reverse=True 时降序,reverse=False 时升序(默认)。例如:

```
>>> sorted([4, 1, 66, 43])
[1, 4, 43, 66]
>>> sorted([4, 1, 66, 43], reverse = True)
[66, 43, 4, 1]
>>>
>>> sorted(['a', 'k', 'e', 'q'])
['a', 'e', 'k', 'q']
```

```
>>> example_list = [5, 0, 6, 1, 2, 7, 3, 4]
>>> result_list = sorted(example_list, key = lambda x: x * 2)
>>> result_list
[0, 1, 2, 3, 4, 5, 6, 7]
>>> students = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10),]
>>> sorted(students, key = lambda student : student[2])              #按照学生年龄排序
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
>>> sorted(students, key = lambda student : student[2], reverse = True)#按照学生年龄逆序排列
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
>>>
```

6. enumerate()函数

enumerate()接受一个序列作为参数，返回一个 enumerate 对象，该对象生成由序列每个元素的 index(索引值)和 item(元素值)组成的元组。例如：

```
>>> a = ['a', 'b', 'c', 'd', 'e']
>>> for item in enumerate(a):
            print(item)
(0, 'a')
(1, 'b')
(2, 'c')
(3, 'd')
(4, 'e')
```

7. zip()函数

zip()函数接受多个序列，返回一个列表。这个列表的长度为多个序列中最短序列的长度，第一个元素为各个序列第一个元素组成的元组，其他元素以此类推。例如：

```
>>> for item in zip([5, 6, 3, 7], 'adkvjdfff', (2, 3, 4, 2, 5)):
        print(item)
(5, 'a', 2)
(6, 'd', 3)
(3, 'k', 4)
(7, 'v', 2)
```

8. seq.index(x[,i[,j]])函数

这个内建函数的作用是，得到序列 seq 中从 i 开始到 j 位置中第一次出现元素 x 的索引。例如：

```
>>> a = ['a', 'd', 'g', 'f', 'g']
>>> a.index('g')
2
>>> a.index('g', 3)
4
```

9. seq.count(x)函数

这个函数的作用是计算序列 seq 中出现元素 x 的总次数。例如：

```
>>> a = ['a', 'd', 'g', 'f', 'g']
>>> a.count('g')
2
```

5.3 字符串

5.3.1 字符串类型

字符串类型是 Python 中最常见的类型之一,可以通过 str()内建函数将其他序列转换为字符串,或者可以简单地在引号中包含字符的方式来创建。与 C 或者 Java 语言不同,Python 中没有字符,而是用一个长度为 1 的字符串来表示字符。同时,在 Python 中单引号、双引号和三引号(三引号有差异,可用于多行注释)所起到的作用是一样的,即'String'、"String"、'''String'''代表同一个字符串。

在 Python 3 中,所有的字符串都用 Unicode 表示(16 位)。在 Python 2 内部存储为 8 位 ASCII 码,因此需要附加'u'使其成为 Unicode,而现在不再需要了。

与 C 语言的字符串是可变的不同,Python 的字符串和 Java 语言的一样,一旦被创建就不可以被改变。这个特性会在 5.3.3 节中讨论。

5.3.2 字符串的操作

字符串是一种序列,在 5.2 节中所提到的关于序列的操作符和内建函数都可以用来操作字符串。接下来看几个例子,了解怎样操作字符串。

1. 字符串的创建和赋值

```
>>> aString = 'abcdefg'        #将一个创建好的字符串赋值给变量,就完成了字符串的赋值
>>> bString = "abcdefg"
>>> cString = str([1,2,3,45,66,6])
>>> cString
'[1, 2, 3, 45, 66, 6]'
>>> aString
'abcdefg'
>>> bString
'abcdefg'
>>> aString is bString
True
```

2. 字符串的索引和切片

```
>>> aString = 'hello world'
>>> aString[:5]
'hello'
>>> aString[-1]
'd'
>>> aString[::-1]
'dlrow olleh'
```

在 Python 中可以用负数来索引序列,-1 代表最后一位,-2 代表最后第二位,以此类推。在切片操作中,seq[i: j]中若 i 省略则代表从第一位开始,若 j 省略则代表直至最后一位。同时 seq[::-1]相当于将 seq 序列进行翻转。这两个切片操作适用于所有序列类型而不仅仅是字符串。

3. “改变”字符串

```
>>> aString = 'hello world'
>>> aString
'hello world'
>>> aString = aString[:4] + aString[5:]
>>> aString
'hell world'
```

这个例子看似删去了字符串中的一个字符 o，但其实是 aString[:4] + aString[5:]创建了一个新的字符串'hell world'，再将其赋值给 aString。新字符串由原字符串的子串构成，跟原字符串并没有直接关系。因此字符串类型仍然是不可变对象。

4. 使用 Python 的 string 模块中预定义的字符串

```
>>> import string
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.digits
'0123456789'
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

string 模块中预定义的字符串，可以配合正则表达式使用，也可以用于清洗数据，实现简单、快速的代码编写。

5.3.3　字符串的独特特性

和其他的序列有所不同，字符串有着自己独特的特性。这些特性是其他序列类型所不具备的。

1. 不可改变性

在 5.3.2 节的第三个例子“改变”字符串中可以看到，想要改变一个字符串只能通过创建一个新字符串的方法，抛弃老字符串，达到改变的效果。若强行给字符串中某一元素赋值以试图改变字符串，就会报错：

```
>>> aString = 'hello world'
>>> aString[1] = 'c'
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    aString[1] = 'c'
TypeError: 'str' object does not support item assignment
```

2. 转义字符

在 Python 中，字符串通常放在引号中作为标志。例如：

```
>>> s = 'This is Python'
```

若在字符串中,本身就存在单引号需要输出的情况,也会报错。例如:

```
>>> s = 'I'm fine'
SyntaxError: invalid syntax      #报错
```

因此需要在字符中使用特殊字符(如引号)时,Python 和其他高级语言一样,使用反斜杠(\)来转义字符。一个反斜线加一个单一字符可以表示一个特殊字符。

```
>>> s = 'I\'m fine'
>>>
```

添加转义字符后,单引号被认为是一个字符串中的特殊字符,而不是字符串的标志符号,程序便不会报错。

常见的转义字符如表 5-1 所示。

表 5-1　常见的转义字符

转义字符	描　述	转义字符	描　述
\(在行尾时)	续行符	\n	换行
\\	反斜杠符号	\v	纵向制表符
\'	单引号	\t	横向制表符
\"	双引号	\r	回车
\a	响铃	\f	换页
\b	退格(Backspace)	\oyy	八进制数,yy 代表的字符,例如:\o12 代表换行
\e	转义	\xyy	十六进制数,yy 代表的字符,例如:\x0a 代表换行
\000	空	\other	其他的字符以普通格式输出

有时并不想让转义字符生效,只是想显示字符串原来的意思。除了用\\来代表反斜杠符,使转义失效外,Python 提供了一种更简单的方法,就要用 r 和 R 来定义原始字符串。例如:

```
>>> print('\thello')
    hello
>>> print('\\thello')
\thello
>>> print(r'hello')
hello
>>> print(r'\thello')
\thello
```

3. 格式化字符串

格式化字符串是一些程序设计语言在格式化输出 API 函数中用于指定输出参数的格式与相对位置的字符串参数。例如,“今天我想要买 x 个 y”这句话中的 x 和 y 可以根据变量的不同,输出不同的内容。在 Python 中,采用的格式化方式和 C 语言是一致的,用%实现。

字符串格式化符号如表 5-2 所示。

表 5-2　字符串格式化符号

符　　号	描　　述
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整数
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数(大写)
%f	格式化浮点数,可指定小数点后的精度
%e	用科学记数法格式化浮点数
%E	作用同%e,用科学记数法格式化浮点数
%g	%f 和%e 的简写
%G	%f 和%E 的简写
%p	用十六进制数格式化变量的地址

格式化操作辅助指令如表 5-3 所示。

表 5-3　格式化操作辅助指令

指　　令	功　　能
*	定义宽度或者小数点精度
—	用作左对齐
+	在正数前面显示加号(+)
<sp>	在正数前面显示空格
#	在八进制数前面显示零('0'),在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格
%	'%%'输出一个单一的'%'
(var)	映射变量(字典参数)
m.n.	m 是显示的最小总宽度,n 是小数点后的位数

例如：

```
>>> '今天我想买%d个%s' % (5,'苹果')
'今天我想买5个苹果'
>>> '%.2f' % 3.1415926
'3.14'
>>> '%E' % 1234.567890
'1.234568E+03' #科学记数法
```

从 Python 2.6 开始,新增了一种格式化字符串的函数 str.format(),它增强了字符串格式化的功能。其基本语法是通过{}和：来代替以前的%。format()函数可以接受无限个参数,位置可以不按顺序。例如：

```
>>> "{} {}".format("hello", "world")
'hello world'
```

```
>>> "{1} {0} {1}".format("hello", "world")
'world hello world'
```

4. 三引号(也支持做注释)

Python 中三引号允许一个字符串跨多行(跨行时解释器默认在换行处加入\n),字符串中可以包含换行符、制表符以及其他特殊字符。使用三引号使编程更方便,输入什么便输出什么,不用在字符串中写转义字符表示换行等。例如:

```
>>> a= '''这是第一行
换一行写
'''
>>> print(a)
这是第一行
换一行写
>>> a
'这是第一行\n换一行写\n'
>>> a = 'ssz-\
ECUPL'
>>> a
'ssz-ECUPL'
```

5.3.4 字符串的内建函数

字符串是序列,所以在 5.2.4 节中的序列的一些内建函数都适用于字符串。而字符串独有的内建函数也有很多,在这里篇幅有限,所以只介绍最为常用的几个内建函数(方法)。

1. string.find(str, beg=0, end=len(string))

检测 str 是否包含在 string 中,如果 beg 和 end 指定范围,则检查是否包含在指定范围内,如果找到 str,则返回首次匹配到的目标字符串的起始字符的索引值,否则返回-1。这个和序列的 index()方法类似,但是 index()方法找不到 str 时会返回一个异常,而 find()方法则返回-1。例如:

```
>>> 'abcdefghij'.find('c')
2
>>> 'abcdefghij'.find('k')
-1
>>> 'abcdefghij'.index('k')
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    'abcdefghij'.index('k')
ValueError: substring not found
```

2. string.split(str, num=string.count(str))

以 str 为分隔符切片 string,并返回一个列表,如果 num 有指定值,则仅切除前 num 个 str 并返回。例如:

```
>>> 'dafagahajaka'.split('a')
['d', 'f', 'g', 'h', 'j', 'k', '']
>>> 'dafagahajaka'.split('a',5)
['d', 'f', 'g', 'h', 'j', 'ka']
```

3. string.upper()和 string.lower()

前者将字符串中的所有字母转换为大写字母，后者将字符串中所有字母转换为小写字母。例如：

```
>>> 'abcDEFghi12Kl'.upper()
'ABCDEFGHI12KL'
>>> 'abcDEFghi12Kl'.lower()
'abcdefghi12kl'
```

4. string.startswith(obj,beg=0,end=len(string))和 string.endswith(obj, beg=0, end=len(string))

前者检查字符串是否是以 obj 开头，若是则返回 True，否则返回 False。如果 beg 和 end 指定值，则在指定范围内检查。后者检查是否以 obj 结尾。例如：

```
>>> 'acbdefghij'.startswith('acb')
True
>>> 'acbdefghij'.startswith('ab')
False
>>> 'acbdefghij'.endswith('hij')
True
```

5. string.strip([obj])

删除 string 字符串前后的 obj，如果不传参数，删除前后空格。例如：

```
>>> '    hello world!     '.strip()
'hello world!'
>>> '@@@@#@hello world!@#@@'.strip('@')
'#@hello world!@#'
>>> 'hello world!'.strip('hello')
' world!'
>>>
```

6. string.join(seq)

以 string 作为分隔符，将 seq 中所有的元素（用字符串表示）合并为一个新的字符串。例如：

```
>>> '&'.join('abcdefg')
'a&b&c&d&e&f&g'
>>> '@'.join(['hello', 'world', '!'])
'hello@world@!'
>>>
```

7. string.replace(old, new [, max])

将字符串 string 中的所有子串 old 替换成新字符串 new，如果 max 指定，则替换不超过 max 次。例如：

```
>>> 'hello@world@!'.replace('@','#')
'hello#world#!'
>>> 'hello@world@!'.replace('@','#',1)
'hello#world@!'
```

8. string.partition(str)

从左往右匹配 str 片段,以第一个匹配项为分隔符,返回头、分隔符、尾三部分的三元组。如果没有找到分隔符,就返回 string 本身和两个空元素组成三元组。例如:

```
>>> a = 'hello world I am coming'
>>> a.partition('I am')
('hello world ', 'I am', ' coming')
>>> a.partition('abc')
('hello world I am coming', '', '')
```

【例 5-3】 判断是否是一个正确的变量名。编写一个小程序,判断变量名是否符合 Python 变量的命名规则。

参考代码:

```
import string
import keyword
alphas = string.ascii_letters + '_'
nums = string.digits
inp = input('请输入一个变量名:')
#判断输入的是否是 Python 关键字
if inp in keyword.kwlist:
    print('变量名不可以是 Python 中的关键字')
elif len(inp) > 0:
    #判断是否以字母或者_开头
    if inp[0] not in alphas:
        print('不合法!变量名必须以字母或者_开头')
    else:
        s = True
        for otherChar in inp[1:]:
        #判断是否只出现了字母、数字或_
        if otherChar not in alphas + nums:
            print('不合法,变量名里只能出现字母、数字或_')
            s = False
            Break
        if s:
            print('对的,是合法的变量名')
else:
    print('不能什么也不输入')
```

5.4 列表

5.4.1 列表的定义

和字符串一样,列表也是 Python 中最常用的数据类型之一。它们都是序列,有很多相同之处,例如在 5.2 节中提到的序列操作符,也都适用于列表。二者不同的是:字符串只能由字符组成,且是不可变的(不能单独改变它的某个值),而列表是可变的,不仅长度可变,而且其中的元素也可变。Python 中的列表很像 C 语言或者 Java 语言中的数组,但是比它们

都要灵活。在C语言或者Java语言中的数组中的元素必须是同一数据类型，而Python中的列表并不是。而且列表中包含的元素甚至也可以是序列，以及自己创建的数据类型对象。同时，相比于C和Java语言中的数组的长度固定性，列表可以任意地添加或者减少元素，还可以跟其他的列表结合或者把一个列表分成几个。列表可以用一对大括号([])来创建。例如：

```
>>> alist = [1,2,3,4,5]
```

列表是一个有序集合，索引从0开始，列表的索引值可以为负(－n即倒数第n个元素的索引)。例如：

```
>>> alist[1]
2
>>> alist[ - 1]
5
```

5.4.2　列表的操作

列表的元素是可以变动的，如增加、删除、修改，不过需要注意的是，列表的元素不是基本数据类型，而是一个个标识符引用对象。列表常见的操作如下。

1. 创建列表和赋值

```
>>> alist = [1, '22', 'hello', 2.33]
>>> alist
[1, '22', 'hello', 2.33]
>>> blist = list(alist[2])
>>> blist
['h', 'e', 'l', 'l', 'o']
>>> alist[2] = blist
>>> alist                          #列表中的元素不需要同一类型
[1, '22', ['h', 'e', 'l', 'l', 'o'], 2.33]
```

2. 列表的索引和切片

列表和字符串一样，也有索引和切片操作。

```
>>> alist = ['hello', 'world','!', 'I ', 'am', 'coming']
>>> alist[2]
'!'
>>> alist[ - 1]
'coming'
>>> alist[:: - 2]
['coming', 'I ', 'world']
>>> alist[2:5:2]
['!', 'am']
```

3. 列表的更新

```
>>> alist = ['a', 'b', 'c', 'd', 'e','f', 'g']
>>> alist[2] = 'H'                 #更改列表的元素
>>> alist
```

```
['a', 'b', 'H', 'd', 'e', 'f', 'g']
>>> alist[1:4] = ['3', '4']
>>> alist
['a', '3', '4', 'e', 'f', 'g']
>>> alist.append('hahaha')      #在列表末尾追加元素
>>> alist
['a', '3', '4', 'e', 'f', 'g', 'hahaha']
>>> alist.insert(1,'hhh')       #在列表内部插入元素,插入后的索引为1
>>> alist
['a', 'hhh', '3', '4', 'e', 'f', 'g', 'hahaha']
```

可以对列表中的指定元素进行赋值和更新。而且这些指定元素可以是通过切片得到的序列,将这几个元素一起替换为新的内容。同时列表有一个内建函数 append(),可以将新元素追加到列表末尾。而 insert()可以指定索引位置的插入操作。

4. 列表的删除

列表的删除操作可以是指定元素删除,也可以是指定位置删除,还可以使用 pop()方法进行末尾元素的删除与返回操作。

```
>>> alist
['hello', 'world', '!', 'I ', 'am', 'coming']
>>> del alist[2]
>>> alist
['hello', 'world', 'I ', 'am', 'coming']
>>> alist.remove('am')
>>> alist
['hello', 'world', 'I ', 'coming']
>>> alist.pop()
'coming'
>>> alist
['hello', 'world', 'I ']
>>> alist.pop(0)
'hello'
>>> alist
['world', 'I ']
```

当明确地知道要删除元素在列表中的位置时,可用 del()来删除;当并不明确要删除元素的位置时,可用 remove()方法删除特定的元素;而 pop()方法也有删除列表元素的作用,并返回删除的元素,未传入位置参数时,pop()默认删除列表中最后一个元素。

【例 5-4】 创建一个列表,元素是 100 以内的自然数,然后删除该列表中的后 50 个元素。

参考代码:

```
L = []
for x in range(1,101):
    L.append(x)
print(L)
for y in range(50):
    L.pop()        #删除末尾元素
print(L)
```

注意，在第二次循环中，删除的语句也可以写成 L.pop(50)，即每次循环，都删除列表中的第 51 号元素，每次删除操作之后，从 51 号位置开始向后的每个元素的索引都向前进 1。

【例 5-5】 列表 wordlist 中的元素是单词本中的单词，从该列表中随机选出某一元素，询问用户：认识该单词吗？如果认识，则从键盘输入 Y，否则输入 N。当用户输入 Y 时，从列表中删除该元素，否则什么也不做。循环操作 5 次后结束单词本的更新，提示“今日更新完毕。”

参考代码：

```
wordlist = ['unanimous', 'declaration', 'thirteen', 'united', 'states', 'america','when', 'course',
'human', 'events', 'becomes', 'necessary', 'people', 'dissolve','political', 'bonds']
import random
for x in range(5):
    word = random.choice(wordlist)
    print(word)
    answer = input('认识该单词吗?如果认识,从键盘输入 Y,否则输入 N')
    if answer == 'Y':
        wordlist.remove(word)
    else:
        continue
print('今日更新完毕。')
```

运行结果如图 5-1 所示。

```
when
认识该元素吗？如果认识，从键盘输入Y，否则输入NY
becomes
认识该元素吗？如果认识，从键盘输入Y，否则输入NN
course
认识该元素吗？如果认识，从键盘输入Y，否则输入NY
becomes
认识该元素吗？如果认识，从键盘输入Y，否则输入NY
events
认识该元素吗？如果认识，从键盘输入Y，否则输入NY
今日更新完毕。
>>>
```

图 5-1　例 5-5 运行结果

5.4.3　列表的常用内建函数

和字符串一样，关于列表，Python 也内建了很多函数（方法），便于程序员调用。除了 5.4.2 节中提到的与列表更新、删除有关的内建函数 del()、append()、insert()、remove()、pop()和 5.2.4 节中提到的关于序列的内建函数外，列表还有其他常用的内建函数（方法），具体如下。

1. list.extend(seq)

extend()方法在列表末尾一次性追加另一个序列中的多个值，也就是将 seq 中的元素一个一个地加入列表的末尾，而不是像 append()方法一样将整个序列当成一个元素加在列表末尾。extend()的作用和连接（+）类似，但效率较高。例如：

```
>>> alist = [1,2,3]
>>> blist = [4,5,6]
>>> alist.extend(blist)
>>> alist
```

```
[1, 2, 3, 4, 5, 6]
>>> alist[3:] = []
>>> alist
[1, 2, 3, []]
>>> alist.append(blist)
>>> alist
[1, 2, 3, [4, 5, 6]]
```

2. list.sort()和 list.reverse()

sort()和 reverse()方法的作用和 5.2.4 节中提到的 sorted()和 reversed()函数类似,都是将列表(序列)中的元素排序和翻转。但是 sorted()和 reversed()是接受一个序列,即以一个序列作为参数,返回排好序和翻转后的新序列,对原序列不产生影响,而 sort()和 reverse()方法是直接作用在原有列表上,不会产生新的列表。例如:

```
>>> alist = [1, 4, 6, 3, 5 ,9, 8]
>>> sorted(alist)
[1, 3, 4, 5, 6, 8, 9]
>>> alist
[1, 4, 6, 3, 5, 9, 8]
>>> alist.sort()
>>> alist
[1, 3, 4, 5, 6, 8, 9]
```

3. list.copy()

copy()方法返回一个列表的副本,用于复制列表。直接赋值所达到的复制效果是浅拷贝,只是让两个变量指向同一块内存空间中的值,两列表会同步变化,如图 5-2 所示。而 copy()方法是新开辟了一块内存空间,两者只是值相同的两个不同列表,如图 5-3 所示。例如:

```
>>> alist = [1, 2, 3, 4]
>>> blist = alist
>>> alist, blist
([1, 2, 3, 4], [1, 2, 3, 4])
>>> alist[1] = 'a'
>>> alist, blist
([1, 'a', 3, 4], [1, 'a', 3, 4])
>>> blist = alist.copy()
>>> alist,blist
([1, 'a', 3, 4], [1, 'a', 3, 4])
>>> alist[1] = 2
>>> alist, blist
([1, 2, 3, 4], [1, 'a', 3, 4])
```

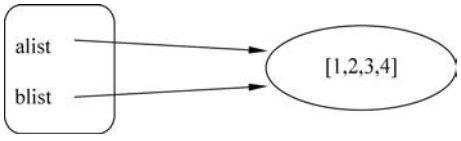

图 5-2　列表的浅拷贝

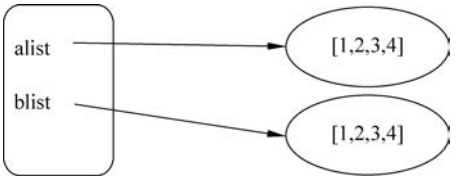

图 5-3　值相同的两个不同列表

4. list.clear()

这个方法的作用是将列表清空，使其变为一个空列表。例如：

```
>>> alist = [1, 2, 3, 4, 5, 6]
>>> alist
[1, 2, 3, 4, 5, 6]
>>> alist.clear()
>>> alist
[]
```

【例 5-6】 合并列表 lst1＝[32，42，12，5，14，4，1]和 lst2 ＝ [199，22，324，89，2]，并且保持合并后的列表有序(从小到大)。

【分析】 合并两个简单列表，并且保持合并后列表有序，最简单快速的方法是先连接两个列表，再对新列表进行排序，两三行代码就可以解决。但是若规定不可以用＋(连接操作符)或 extend()方法连接两个列表，还有什么方法可以做到？

参考代码：

```
lst1 = [32, 42, 12, 5, 14, 4, 1]
lst2 = [199, 22, 324, 89, 2]
lst3 = []
#先分别给两个列表排序
lst1.sort()
lst2.sort()
i = 0
j = 0
len1 = len(lst1)
len2 = len(lst2)
'''将排好序的两个列表中未插入新列表的最小项相互比较,小的插入新的列表中
直至一个列表的所有元素都插入新列表中后,再将另一个列表的剩余元素依次插入'''
while i < len1 and j < len2:
    if lst1[i]< lst2[j]:
        lst3.append(lst1[i])
        i += 1
    else:
        lst3.append(lst2[j])
        j += 1
#将未全部插入新列表的那个列表中的剩余元素依次插入新列表中
if i < len1:
    for l in lst1[i:]:
        Lst3.append(l)
else:
    for l in lst2[j:]:
        Lst3.append(l)
#输出两个列表合并后按从小到大排序的新列表
print(lst3)
```

5.5 元组

5.5.1 什么是元组

元组是和列表十分相近的另一种容器类型。元组和列表看起来不同的一点是元组用的是小括号而列表用的是中括号。但是元组一旦被创建,就不可以被改变。元组和字符串一样,都有不可改变性,它是不可变类型。在功能上,正是因为元组是不可变类型,它可以成为字典(Python 的另一种数据结构)的键,而列表不行。另外,在处理一组对象时,这个组默认是元组类型。例如:

```
>>> 1, 'hahah', 2.1
(1, 'hahah', 2.1)
```

5.5.2 元组的操作

元组的操作和列表很类似,但是元组是不可变类型,所以列表中关于更新和删除的操作都无法使用在元组上。

1. 元组的创建和赋值

```
>>> atuple = (None, 'hello', 'world', 1)    #元组的创建,标识符是小括号"()"
>>> atuple
(None, 'hello', 'world', 1)
>>> ctuple = tuple([1, 2, 3, 4])           #用 tuple()函数进行类型的转换,转换为元组
>>> ctuple
(1, 2, 3, 4)
>>> btuple = (1)
>>> btuple
1
>>> type(buple)
<class 'int'>
>>> btuple = (1,)
>>> btuple
(1,)
```

元组的创建和赋值与列表还有字符串类似,直接用()创建或者使用内建函数 tuple()创建。需要注意的是,在创建单个元素元组时,在元组分隔符里面加一个逗号(,),否则标识符小括号容易和数学运算表达式中的小括号混淆。

2. 元组的“可变性”

```
>>> alist = ['hello', 'world']
>>> atuple = (1, 'hahaha',alist)
>>> atuple
(1, 'hahaha', ['hello', 'world'])
>>> alist[1] = 'father'
>>> atuple
(1, 'hahaha', ['hello', 'father'])
```

元组本身是不可变的，在本例中，从表面上看，元组 atuple 的元素确实变了，但其实变的不是 atuple 的元素，而是元组中的元素列表 alist 的元素。atuple 一开始指向的 alist 并没有改成别的 alist。所以，元组不可变意思是元组中的每个元素指向永远不变。

3. 元组的删除和“更新”

```
>>> atuple = (1, 2, 3, 'hello', 'world')
>>> atuple
(1, 2, 3, 'hello', 'world')
>>> atuple = atuple + ('!',)
>>> atuple
(1, 2, 3, 'hello', 'world', '!')
>>> del atuple
```

因为元组和字符串一样是不可变的，所以要更新元组，只能用＋(连接操作符)创建一个新元组来实现“更新”的功能。而因为不可变性，无法删除元组中的某个元素，除非用 del()函数删除整个元组。

5.5.3　元组的操作符和内建函数

元组的操作符和内建函数与列表相比，没什么大区别，就是缺少了可以改变自身的内建函数。相关的操作符和内建函数都已经在前面介绍过，接下来用几个例子来具体了解元组的操作符和内建函数。

1. 元组的运算符

```
>>> len((1, 2, 3))
3
>>> ('a', 'b', 'c') + ('d', 'e', 'f')
('a', 'b', 'c', 'd', 'e', 'f')
>>> ('Ha!',) * 5
('Ha!', 'Ha!', 'Ha!', 'Ha!', 'Ha!')
>>> 'a' in (1, 2, 'a', 'b')
True
>>> (6, 7) > (4,8)
True
```

2. 元组的索引和切片

```
>>> atuple = ('hello', 'world', '!')
>>> atuple[ - 2]
'world'
>>> atuple[ : : - 1]
('!', 'world', 'hello')
>>> atuple[1:]
('world', '!')
```

3. 元组的内建函数

```
>>> atuple = (1, 33, 4, 4.6, - 5)
>>> max(atuple)
33
```

```
>>> min(atuple)
-5
>>> atuple.index(4)
2
>>> atuple.count(4.6)
1
>>> sorted(atuple)
[-5, 1, 4, 4.6, 33]
>>> type(sorted(atuple))
<class 'list'>
>>> atuple
(1, 33, 4, 4.6, -5)
```

从上述例子中可以看出,在使用 sorted()函数排序元组时,返回的是一个排好序的列表,并且对元组不产生影响。而 sort()方法会对调用它的对象产生改变,故元组无 sort()方法。

5.6 练习

1. 编写代码,有如下变量 name = " aleX",按照要求实现每个功能。

(1) 移除 name 变量对应的值两边的空格,并输出移除后的内容。

(2) 判断 name 变量对应的值是否以"al"开头,并输出结果。

(3) 判断 name 变量对应的值是否以"X"结尾,并输出结果。

(4) 将 name 变量对应的值中的"l"替换为"p",并输出结果。

(5) 将 name 变量对应的值根据"l"分隔,并输出结果。

(6) 第(5)题分隔之后得到的值是什么类型?

(7) 将 name 变量对应的值变大写,并输出结果。

(8) 输出 name 变量对应的值的后 2 个字符。

(9) 输出 name 变量对应的值中"e"所在索引位置。

2. 编写代码,有列表 li=['alex', 'eric', 'rain'],按照要求实现每一个功能。

(1) 计算列表长度并输出。

(2) 列表中追加元素"seven",并输出添加后的列表。

(3) 修改列表第 2 个位置的元素为"Kelly",并输出修改后的列表。

(4) 删除列表中的元素"eric",并输出修改后的列表。

(5) 将列表内的元素顺序反转,并输出反转后的列表。

(6) 使用 for 循环输出列表的所有元素。

3. 编写代码,有如下元组 tu=('alex','eric','rain'),按照要求实现每一个功能。

(1) 获取元组的第 1 个和第 2 个元素,并输出。

(2) 使用 for、len、range 输出元组的索引。

(3) 使用 enumerate 输出元组元素和序号。

4. 欲从 s='Hello world'字符串中切片出子串'Hlwl',正确的切片表达式为(　　)。

A. s[::3]　　B. s[:3:2]　　C. s[3:11:3]　　D. s[3::3]

5. 若列表 score＝[60,70,60,60,70,90],则执行操作 score.remove(score[－2])后 score 的值是（　　）。

A. [60, 60, 60, 70, 90]　　B. [60, 70, 60, 60, 70]

C. [70, 60, 60, 70, 90]　　D. [60, 70, 60, 60, 90]

6. 若 Tup＝(2,(2,1),(2,(2,1)),(2,(2,1),(2,(2,1)))),则下列叙述正确的是(　　)。

A. Tup[3][2]的值为(2,1)　　B. 元组 Tup 的长度为 5

C. Tup[3]的值是(2,(2,1),(2,(2,1)))　　D. Tup[2]的值是(2,1)

7. 已知 x ＝ 'abcdefg',若要得到'defgabc'结果,应使用表达式(　　)。

A. x[3:] ＋ x[:3]　　B. x[:3] ＋ x[:3]

C. x[3:] ＋ x[3:]　　D. x[:3] ＋ x[3:]

8. 执行结果为[1, 2, 3, 1, 2, 3, 1, 2, 3]的表达式是(　　)。

A. [1,2,3]＋ [1,2,3]

B. ['1','2','3']＋ ['1','2','3']＋ ['1','2','3']

C. [1, 2, 3] ** 3

D. [1, 2, 3] * 3