第3章 Python序列化数据及推导式

Python 中的序列化数据是一种重要的数据类型，主要包括前面章节中见到的字符串 string 数据，以及列表 list 数据、元组 turple 数据、字典 dict 数据、集合 set 数据等。本章将引导同学们了解上述几种常见的序列化数据类型的特点、常见的针对序列化数据的处理方法等，并简介推导式及其用法。

学习本章内容时，要求：

- 理解有序序列和无序序列；对于有序序列，掌握对其进行排序的相关方法；
- 理解可变序列（不可哈希）和不可变序列（可哈希）的概念；对于可变序列，掌握对其元素进行赋值等的相关方法；
- 掌握字符串类型数据的概念、特点及基本操作方法；
- 掌握列表类型数据的概念、特点及基本操作方法；掌握元组类型数据的概念、特点及基本操作方法；掌握列表和元组的异同点；
- 掌握字典类型数据的概念、特点及基本操作方法；掌握集合类型数据的概念、特点及基本操作方法；掌握字典和集合的异同点；
- 掌握列表推导式、元组推导式、字典推导式、集合推导式的用法。

3.1 概　　述

3.1.1 序列化数据

在程序设计中，除了常见的基本数值类型（如整型 int、浮点型 float、布尔型 bool 等）外，有时还要处理序列化数据。顾名思义，“序列”指按特定顺序依次排列的一组数据，它们可以占用一块连续的内存空间，也可分散到多块空间中。本章将讨论序列化数据中的字符串、列表、元组、字典、集合。

有的同学可能会有疑问：基本数据类型就能解决很多问题了，为什么还要学习使用序列化数据呢？这是因为在很多场合中，仅使用基本数据类型并不方便。很多时候，当需要保存一批数据时，使用序列

化数据类型是十分方便的。例如，在如下几种情况中，单纯使用数值型数据是不合适的。

- 全班同学的姓名（如晓明、小红、小兵……）适合用序列化数据中的字符串类型数据表示，适合用序列化数据中的列表、元组等存储。例如定义一个列表 a[“晓明”,“小红”, “小兵”]，可以通过类似于学号的索引号找到某个同学的姓名，如 a[0]中存储“晓明”，a[1]代表“小红”，a[－1]代表“小兵”等；如果没有重名的同学且存储是无序的，则使用序列化数据中的集合类型也可以表示。
- 全班同学期末考试的平均分数，如“晓明”考了 89 分，“小红”考了 92 分，“小兵”考了 90 分，适合用键-值对（key-value）数据类型表示，其中姓名作为键（key），分数作为值（value），适合用序列化数据中的字典存储，例如：{“晓明”：89，“小红” ：92，“小兵”：90}。
- 期末考试的科目（如数学、物理、化学、语文）适合用序列化数据中的元组表示，因为科目名是不能被修改的；若要求有序，可以用元组表示；若要求无序、无重复科目，也可用集合存储这些科目。

序列化数据在日常生活中其实是很常见的，高中数学课上学到的各种数列（如等差数列、等比数列等）就是一种序列化数据。可见，对于上面的需求，无法用基本数据类型中的 int、float、bool 等表示。此时，就需要使用序列化数据中的字符串、列表、元组、字典、集合等序列化的数据类型了。

3.1.2 推导式

推导式（或称为生成式，本书后续不再区分这两种称呼）是指可以从一个数据序列构建另一个新的数据序列的高效的结构体，推导式的使用也是 Python 这门语言的一大特色，Python 能用简洁的推导式完成其他语言需要很多行代码才能实现的功能。

Python 有几种常用的推导式，如生成器推导式（生成结果是生成器对象）、列表推导式（生成结果是列表对象）、元组推导式（生成结果是元组对象）、字典推导式（生成结果是字典对象）、集合推导式（生成结果是集合对象）。例如，列表推导式是形如“[expr for val in collection if condition]”的形式简洁、内涵复杂的具有 Python 特色的表达式。由于其定界符是方括号，因此这是一个列表推导式，它表达的含义是可构造这样一个列表：用变量 val 依次遍历可迭代对象 collection，找出满足条件 condition 的元素后，经 expr 表达式或相关函数的变化后，输出以列表表示的结果。例如，从键盘输入一句英文句子，将单词长度大于 3 的所有单词转换为大写字符输出，可以非常简洁地用图 3.1 所示的一条语句（第 2 行代码）完成任务。这里的 expr 是 upper()方法，collection 是输入的英文句子中的各个单词，condition 是 len()＞ 3。

```
1 mystring = input("请输入用一句英文句子").split()
2 [x.upper() for x in mystring if len(x) >3]
```

```
请输入用一句英文句子China launches two space experiment satellites
['CHINA', 'LAUNCHES', 'SPACE', 'EXPERIMENT', 'SATELLITES']
```

图 3.1 列表推导式示例

3.2 序列化数据的主要特点和常用内置函数

3.2.1 主要特点

除字符串外，序列化数据中的列表、元组、字典、集合的相同之处是其中的各个元素之间都是用逗号隔开的。不同之处有不少，最直观的是它们的定界符是不一样的：列表用方括号[ ]作为定界符，元组用圆括

号()作为定界符,字典和集合均用大括号{ }作为定界符;字典中的元素是用冒号:隔开的键-值对,其中键(key)可以理解为类别(如一个人的姓名或一个科目名称),而值(value)可以理解为这个类别的一个取值(如某人的身高为180cm)或一组取值(如某科目的全班同学的考试成绩[90, 80, 88, 92])等。

课堂练习

请你分别定义一个列表x和一个元组y(注意列表元素是用中括号括起来的,而元组元素是用圆括号括起来的)。用type(x)和type(y)方法分别显示它们的类型。

序列化数据分为**有序**(如字符串"你吃"和"吃你"是不一样的)和**无序**(如集合中的诸多元素之间就是无序的)数据。对于诸如字符串、列表、元组等有序数据来说,其数据存储是一个接一个地有序排列,每个元素都拥有一个对应的值,代表它存储在序列中的某个位置,可以用**索引**定位其中的某个元素,而对无序数据(如集合中的数据)是不能用索引号访问的。打个比方来说,钢琴一共有88个有序排列的琴键(琴键相当于数据),这88个有序的琴键能在五线谱上找到它们一一对应的确切位置,五线谱中的高低位置就相当于索引号。因此,当看到五线谱某个位置上(索引)的符号后,就能够准确找到对应的琴键(数据)。对于**有序**数据来说,它们大多支持**双向索引**(如字符串、列表、元组等都支持双向索引),即它的第一个元素的索引(下标)为0,第二个元素的索引(下标)为1,最后一个元素的索引(下标)为-1,倒数第二个元素的索引(下标)为-2,以此类推。

图3.2所示的例子显示了字符串的**有序性**,其中len()可获取字符串的长度,其索引从0开始;也可使用enumerate()方法,它用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列,同时列出数据下标和数据本身,因此图3.2中的第4行代码的迭代变量是idx和u两个变量。

```
1  data = "Python"
2  for index1 in range(len(data)):
3      print(index1, data[index1])
4  for idx, u in enumerate(data):
5      print ('#%d: %s' % (idx+1, data[idx]))
```

图3.2 字符串的索引与有序性示例

可以用方括号、圆括号、花括号分别定义列表、元组、字典、集合;使用list()、tuple()、dict()、set()也可将数据转换为列表、元组、字典(需要键-值对数据)、集合,但此时使用圆括号。请注意:①没有list[ ]、dict{ }、set{ }的用法;②在进行索引(切片)时,列表、元组、集合元素都使用方括号,不使用圆括号和花括号。

按序列中的元素是否可变,又分为**可变序列**和**不可变序列**两类。如字符串就是一个不可变序列,一旦定义后,其内容是不可改变的。请注意,字符串也是有替换方法replace()可以使用的,只不过完成替换操作后,要记得将其赋予另外一个新的字符串变量,原始字符串即使使用了replace(),它也是不变的,如图3.3所示。列表中的元素是可以修改的,所以它是可变序列。

表3.1给出了对常见序列化数据的说明。其中的"可哈希"是指可以使用Python内置函数hash()得出其哈希值,即对于一个对象a,如果hash()返回一个整型值(哈希值),则a就是可哈希的。限于本书的科普性质,这里先不介绍哈希函数,只需要知道列表、字典、集合这些可以增加元素、删除元素、修改元素的可变对象属于不可哈希对象;元组、字符串这些不可变对象属于可哈希的对象。可以使用内置函数hash()计算一个对象的哈希值,如图3.4所示。

```
1 mystring = "Schools can establish positions for sports coaches"
2 newstring = mystring.replace("Schools", "Universities")
3 print("原始字符串:%s\n替换后的新字符串:%s" %(mystring, newstring))
```

```
原始字符串:Schools can establish positions for sports coaches
替换后的新字符串:Universities can establish positions for sports coaches
```

图 3.3　字符串的不可替换性

```
hash('string')
```

```
988744190731951637
```

```
hash((1, 2, (2, 3)))
```

```
-9209053662355515447
```

```
hash(1, 2, [2, 3]) # 失败，因为list是可变的
```

图 3.4　使用 hash()判断是否可哈希

表 3.1　字符串、列表、元组、字典、集合数据的主要特点

	字符串 str	列表 list	元组 tuple	字典 dict	集合 set
定界与分隔符	单引号如 a = ‘x y z’ 双引号如 a = “x y z” 三引号如 a = ‘‘‘x y’’’ 元素间无分隔符	中括号如[x, y, z] 元素间用逗号隔开	小括号如(x, y, z) 元素间用逗号隔开	大括号，如{x: v1, y: v2, z: v3} 元素间用逗号隔开	大括号，如{x, y, z} 元素间用逗号隔开
有序？	有序 a[0]　a[−1]	有序 l[0]　l[−1]	有序 t[0]　t[−1]	无序	无序
可变？	不可变 （可哈希）	可变 （不可哈希）	不可变 （可哈希）	不可变 （可哈希）	可变 （不可哈希）
可重复？	是	是	是	键不可重复 值可以重复	否

根据表 3.1，序列化数据的特点分类如图 3.5 所示。列表和元组都按顺序保存元素，所有元素占用一块连续的内存空间，因此每个元素都有自己的索引，可以通过索引访问；列表和元组的区别在于列表元素是可以修改的，而元组是不可修改的。字典和集合存储的数据都是无序的，其中字典元素以键-值对的形式保存。

例 3.1　分别定义列表、元组、字典、集合类型数据。使用 type()方法显示序列化数据的类型；使用索引访问有序序列的某些元素；对于可变对象，修改或增加其元素值；对于不可变象，通过 hash()显示其哈希值。

【提示与说明】　图 3.6 所示为代码实现。请注意以下几点：①定义列表时可以直接用方括号定义，列表中的元素类型可以混杂存在，可以在列表中再嵌套列表、元组等，如第 1 行代码是在列表中又嵌套的列表、元组；第 2 行代码定义元组时也是在内部嵌套了列表、元组、字符串，这四种序列化数据的类型可通过 type()方法得到；②有序序列(如列表、元组等)可通过索引访问其中的有序元素，如第 8、9 行代码所示；③对于可变序列，可以增加新的元素、修改已有的元素，请注意在列表、集合、字典中增加

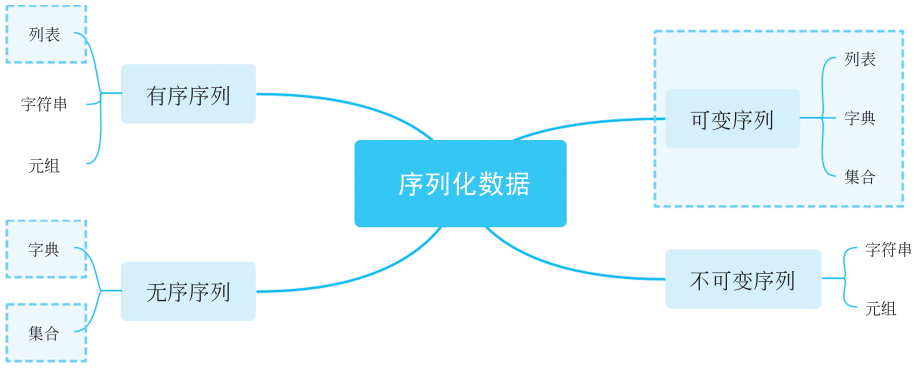

图 3.5　序列化数据的特点分类

新元素的方法——列表用 append()方法、集合用 add()方法，如第 12、13 行代码所示；④对字典中的元素可以直接增加新值(如第 14 行代码所示)，也可以修改键-值对结果；⑤列表、集合这些可以增、删、改元素的对象，属于不可哈希对象；元组、字符串这些不可变对象属于可哈希对象，可以使用内置函数 hash()计算一个对象的哈希值，但如果试图计算不可哈希对象的哈希值，则会抛出异常。

```
1  a= [1,2,[3,4,"my"],(5,6,"hello"),"music","sport"]  #定义整数和字符串的混合列表
2  b = (3.14,[5.68, "world"],("piano","football"),"OK") #定义嵌套的元组
3  mydict = {"a":1,"b":2,"c":3,"d":4} #字典
4  myset = {"数学","语文", 1,2}        #集合
5  #下面显示各自的类型
6  print(str(type(a))+str(type(b))+str(type(mydict))+str(type(myset)))
7  #对于有序序列，是可以用索引访问的
8  print("列表的首元素是："+str(a[0]))#str()用于将结果转换为字符串显示
9  print("元组的末元素是："+str(b[-1]))
10 #对于可变序列，是可以增加、修改其原始值的
11 a[0] = 5 #修改列表中首元素值
12 a.append(100) #在列表追加新元素
13 myset.add(5) #在集合中追加新元素
14 mydict["e"]=5   #字典新增加键(用引号括起来)，等号右侧为其值
15 mydict["a"]=100 #字典修改键(用引号括起来)，等号右侧为其新值
16 #分别显示。看看上述操作的结果
17 print(a)
18 print(mydict)
19 #对于不可变对象，是可以显示其哈希值的
20 print(hash("Hello, world"))
```

```
<class 'list'><class 'tuple'><class 'dict'><class 'set'>
列表的首元素是：1
元组的末元素是：OK
[5, 2, [3, 4, 'my'], (5, 6, 'hello'), 'music', 'sport', 100]    列表和字典值发生变化
{'a': 100, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
2729330529633989697 哈希值
```

图 3.6　4 种序列化数据的比较

另外，序列化数据之间是可以相互转换的。例如一个字符串可以方便地转换为列表、元组、字典、集合——可以用 list()、tuple()、dict()、set()实现相应数据类型的转换。需要注意的是，字典元素是键-值对，单纯的“键”是不能成为字典元素的，因此需要“值”和它匹配，此时可以使用 zip()函数，相关示例详见例 3.2。

例 3.2　对于给定的字符串以及由 range()产生的一系列数字，分别将其分别转换为列表、元组、字典、集合数据类型。

【提示与说明】　本题是使用 list、tuple、dict、set 等方法将相应数据转换为列表、元组、字典、集合等

的示例，代码实现如图 3.7 所示。其中，第 2、3 行代码分别演示 list()、tuple()转换为列表、元组；第 4、5 行代码是使用 dict()转换为字典的情况，注意此例中 zip()的第一个变量是键，第二个变量是值，由于第 4、5 行代码的 zip()的两个变量是不一样的，因此最后结果也是不同的。第 6 行代码是使用 set()转换为集合，注意转换为集合后去掉了重复元素“l”且结果已变得无序，这也印证了表 3.1 中集合的不可重复且无序的特性。另外，range()对象也可以转换为特定类型的列表、元组、字典、集合等，进行类型转换时可以使用 map()方法，详见第 11～14 行代码。

```
1  mystr = "Hello"  #定义字符串
2  mylist = list(mystr)#字符串转换为列表
3  mytuple = tuple(mystr)#字符串转换为元组
4  mydict1 = dict(zip(mystr,range(5)))#通过zip（）组合键值对生成字典
5  mydict2 = dict(zip(range(5), 'cdefgab')) #同时，zip第一个参数是键，第二个参数是值
6  myset = set(mystr)#字符串转换为集合
7  print("字符串转换为列表：",mylist)
8  print("字符串转换为元组：",mytuple)
9  print("字典结果1：",mydict,"字典结果2：",mydict2)
10 print("字符串转换为集合：",myset)
11 print("range结果转换为整数列表：",list(map(int, range(5))))#生成器range()结果转为整型列表的元素
12 print("range结果转换为整数元组：",tuple(map(int, range(5))))#生成器range()结果转为整型元组的元素
13 print("range结果转换为字符串元组：",tuple(map(str, range(5))))#生成器range()结果转为字符串型元组的元素
14 print("range结果转换为字符串集合：",set(map(str, range(5))))#生成器range()结果转为字符串型集合的元素
```

```
字符串转换为列表： ['H', 'e', 'l', 'l', 'o']
字符串转换为元组： ('H', 'e', 'l', 'l', 'o')
字典结果1： {'H': 0, 'e': 1, 'l': 3, 'o': 4} 字典结果2： {0: 'c', 1: 'd', 2: 'e', 3: 'f', 4: 'g'}
字符串转换为集合： {'l', 'e', 'H', 'o'}
range结果转换为整数列表： [0, 1, 2, 3, 4]
range结果转换为整数元组： (0, 1, 2, 3, 4)
range结果转换为字符串元组： ('0', '1', '2', '3', '4')
range结果转换为字符串集合： {'0', '3', '4', '1', '2'}
```

集合的无序性

图 3.7　字符串向其他序列化数据类型的转换

课堂练习

请仿照上面的例子，使用 tuple()、str()将一个列表中的内容转换为元组、字符串。

例 3.3　列表中的元素可以是各种不同类型的数据。请利用 list[ ]方法定义一个含有多个不同类型数据的列表；利用 for 循环分别显示这个列表中的各项内容。

【提示与说明】　此题是为下面将要介绍的列表进行预习，目的是了解列表中的元素可以是异构型的。for 循环的可迭代变量序列可以由 list 承担，代码实现如图 3.8 所示。从图中可看到，列表中可以嵌套由方括号表示的列表、由圆括号表示的元组、由花括号表示的集合等各种类型的数据。通过 for 循环遍历时，会将其中的各种数据元素分别列出，请注意最后一行输出的集合元素与原始定义时的不同，说明集合中的元素是无序的。

```
1  univs = ['Hebei', 3.14, [1,2,"OK"], ("Hello", 34.65,[4,5,6,7]), {"北京","上海"}]
2  for x in univs:
3      print (x)
```

```
Hebei
3.14
[1, 2, 'OK']
('Hello', 34.65, [4, 5, 6, 7])
{'上海', '北京'}
```

图 3.8　使用 for 循环遍历列表中的各个元素

例 3.4 定义一个内容为字符串的列表并修改其中的某个字符内容。类似地，试着定义一项内容为字符串的元组，看看能修改它的值吗？再定义一个字符串，看看能修改它的某个值吗？

【提示与说明】 分别定义一个列表和一个元组。由于它们都是有序的，因此按照其中某个元素的索引，可以对列表中某个元素的值进行修改操作，实际运行效果如图 3.9 所示。可见，我们可以随意更改列表中的元素，但无法对元组中的内容进行修改，这也印证了表 3.1 中元组不可变的特性。

```
1 a= ["北京","上海","青岛","武汉"] #列表
2 a[-1] ="苏州"
3 print(a)
```

```
['北京', '上海', '青岛', '苏州']
```

```
1 b= ("北京","上海","青岛","武汉") #元组
2 b[-1] ="苏州"
3 print(b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-516c5b6d83df> in <module>()
      1 b= ("北京","上海","青岛","武汉") #元组
----> 2 b[-1] ="苏州"
      3 print(b)

TypeError: 'tuple' object does not support item assignment
```

元组对象不支持对其项目的(重新)指派

图 3.9 可变序列(列表)和不可变序列(元组)

3.2.2 常用内置函数

综上所述，序列化数据是一组存于连续内存或不连续内存区域的一组序列化的数据，一些内置函数可应用于这些不同的序列化数据。表 3.2 给出了一些常用的内置函数。

表 3.2 序列化数据的部分常用内置函数

函　数	说　明
range(start, end, step)	返回从 start(含)开始到 end(不含)为止的一组以 step 为步长的序列化数
str(seq)	将数据 seq 转换为字符串
list(seq)	将数据 seq 转换为列表
tuple(seq)	将数据 seq 转换为元组
dict(seq)	将以键-值对表示的数据 seq 转换为字典
set(seq)	将数据 seq 转换为集合
sorted(seq)	对数据 seq 进行顺序排序
reversed(seq)	对数据 seq 进行逆序排序
len(seq)	求数据 seq 的长度
enumerate(seq)	迭代显示数据 seq 的索引号及其内容

例如，表 3.2 中的 list()是将其他类型的数据转换为列表数据的方法，使用示例如图 3.10 所示。请注意由于字典是键-值对数据，因此在转换为列表时，直接将字典变量作为 list()的参数是将键-值对中

的键转换为列表元素；字典变量的 values()方法是将键-值对中的值转换为列表元素，items()方法是完整的键-值对数据。关于字典序列化数据的特点，本章后续会介绍。

```
list(range(1,10,2)) #将range()结果转换为列表元素
print(list('Hello')) #将字符串转换为列表元素
print(list({1,3,5})) #将集合元素转换为列表元素
dict={"a":3,"b":4,"c":0}
print(list(dict)) #将字典中的键转换为列表元素
print(list(dict.values()))#将字典中的值转换为列表元素
print(list(dict.items())) #将字典的键值对转换为列表元素
```

```
['H', 'e', 'l', 'l', 'o']
[1, 3, 5]
['a', 'b', 'c']
[3, 4, 0]
[('a', 3), ('b', 4), ('c', 0)]
```

图 3.10　将其他序列化数据转换为列表数据

例 3.5　针对表 3.2 中函数的使用方法，验证使用 str()、list()、tuple()、dict()、set()进行数据类型转化的方法；使用 sorted()、reversed()排序并迭代输出排序后的效果；针对列表和元组，完成切片操作。

【提示与说明】　代码实现如图 3.11 所示。序列化数据及其可迭代对象的数据对象均可使用 list()函数转换为列表。请注意：①len()方法不仅可用于字符串，其他序列化数据均可使用；②在使用 dict()函数将序列化数据转换为字典类型数据时，数据要以键-值对的形式出现，此时使用 zip()函数是常用方法，但要注意“键”不能重复，且无法被匹配上的数据将会舍弃，注意字符串 a 的长度是 11，去重后的键只有 8 个(空格也算一个)，因此 mydict 的长度是 8；若调用 zip()时将两个变量的顺序调换(参见第 5

```
a = "Hello World"
mylist = list(a)
mytuple = tuple(a)
mydict = dict(zip(a,range(30)))
mydict2 = dict(zip(range(30),a))
myset = set(a)
mystring = str(myset)
a_sorted = sorted(a)
a_reversed = reversed(mytuple)
print("列表：",mylist,"切片：",mylist[-1],"长度：",len(mylist))
print("元组：",mytuple,"切片：", mytuple[0],"长度",len(mytuple)) #注意元组的切片也是方括号
print("字典1：",mydict,"长度：",len(mydict)) #注意键的不重复性，无法通过zip配对的键-值对，将舍掉
print("字典2：",mydict2,"长度：",len(mydict2))
print("集合：",myset,"长度：",len(myset)) #注意结果的无序、不重复性
print("排序后的结果：",a_sorted)
for i,j in enumerate(a_reversed):
    print("索引：",i,"  字符：",j,end = "  ")
print("\n原始字符串：",a)
print("gbk编码最大值：",max(a.encode("gbk")),"原始最大值：", max(a))
print("去重和无序后的字符串：", mystring,"类型是：", type(mystring))
```

```
列表： ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'] 切片： d 长度： 11
元组： ('H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd') 切片： H 长度 11
字典1： {'H': 0, 'e': 1, 'l': 9, 'o': 7, ' ': 5, 'W': 6, 'r': 8, 'd': 10} 长度： 8
字典2： {0: 'H', 1: 'e', 2: 'l', 3: 'l', 4: 'o', 5: ' ', 6: 'W', 7: 'o', 8: 'r', 9: 'l', 10: 'd'} 长度：
集合： {'d', ' ', 'H', 'o', 'e', 'W', 'l', 'r'} 长度： 8
排序后的结果： [' ', 'H', 'W', 'd', 'e', 'l', 'l', 'l', 'o', 'o', 'r']
索引： 0  字符： d  索引： 1  字符： l  索引： 2  字符： r  索引： 3  字符： o  索引： 4  字符： W
字符： o  索引： 7  字符： l  索引： 8  字符： l  索引： 9  字符： e  索引： 10  字符： H
原始字符串： Hello World
gbk编码最大值： 114 原始最大值： r
去重和无序后的字符串： {'d', ' ', 'H', 'o', 'e', 'W', 'l', 'r'} 类型是： <class 'str'>
```

图 3.11　针对序列化数据的部分常用内置函数使用示例

行代码),则键-值对就完成了互换,mydict2 的长度是 11,此时键已经变成 0、1、2 等数字了,也就不再有重复的键了;③集合元素是不能重复且无序的,参见集合处理后的输出结果;④第 9 行代码的 reversed()可完成对原始字符串的逆序排序,但这并不影响原始字符串本身,注意第 15 行代码的输出结果是排序后的结果,而第 18 行代码则是原始结果;⑤enumerate()函数返回两个结果,分别是索引号及其对应的内容,因此在第 16 行代码的 for 循环中需要两个循环变量;⑥使用 max()、min()函数计算最大值和最小值时,若自变量是字符串等非数值型数据,则按字符本身的编码值进行计算,不同编码方式,结果可能不同。

例 3.6 分别将给定的 range 对象,字符串,集合,字典中的键、值、键-值对转换为对应的列表。定义一个列表,使用 del 删除它。

【提示与说明】 可以使用内置方法 list()完成其他数据向列表的转换;可以使用 del()删除列表。注意:list()没有返回值,可用 print()语句将其显示出来。实际运行效果如图 3.12 所示。列表被删除后不能再继续使用了,参见图 3.12 中最后的出错信息。

```
print(list((3,5,7,9,11)))                    #将元组转换为列表
print(list(range(1, 10, 2)))                 #将range对象转换为列表
print(list('hello world'))                   #将字符串转换为列表
print(list({3,7,5}))                         #将集合转换为列表
print(list({'北大':1, '清华':2, '交大':3}))#将字典中的key转换为列表
print(list({'北大':1, '清华':2, '交大':3}.values()))##将字典中的values转换为列表,
print(list({'北大':1, '清华':2, '交大':3}.items())) #将字典中的键值对 转换为列表
x = list()                                   #产生一个新的空列表
x = [1, 2, 3]                                #赋值
del x                                        #删除该列表x
print(x)
```

```
[3, 5, 7, 9, 11]
[1, 3, 5, 7, 9]
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
[3, 5, 7]
['北大', '清华', '交大']
[1, 2, 3]
[('北大', 1), ('清华', 2), ('交大', 3)]

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 11>()
      9 x = [1, 2, 3]                                #赋值
     10 del x                                        #删除该列表x
---> 11 print(x)

NameError: name 'x' is not defined
```

图 3.12 使用 list()完成对其他序列化数据的列表转换示例

3.3 字 符 串

3.3.1 基本特性

我们已经在第 1、2 章见过很多有关字符串的操作。除了支持序列化数据的通用操作(如双向索

引、比较大小、计算长度、元素访问、切片、成员测试等)外,字符串类型数据还支持一些特有的操作方法,如在 print()、format()中使用过的字符串格式化、字符串的查找和替换操作等。字符串属于不可变(可哈希)序列,所以不能直接对字符串对象进行修改操作。字符串支持切片操作,切片操作也只能访问其中的元素而无法修改其中的字符,即使字符串对象提供了更新方法如 replace(),但这也不是对原字符串直接进行"原地"修改和替换(因为它是不能修改原值的),而是返回一个新字符串作为替换后的结果,原来的字符串保持不变。

下面通过一些例题和实际操作介绍字符串的特性。

例 3.7 对于给定的字符串,将其转换为列表后看看能否修改其值,再看看能否直接修改原字符串中的某个字符。

【提示与说明】 字符串可通过 list()方法转换为列表,之后可以对列表内容进行修改,但字符串本身的内容是不能修改的。图 3.13 给出了针对上述要求的结果。第 2 行代码计算其哈希值,说明它是不可变的数据类型,因此第 9 行代码会出错(请参见图 3.13 中最后一行的出错提示,这也是表 3.1 体现的字符串不可变特性的一个示例);但用第 3 行代码将其转换为列表后,就可以用第 5 行代码的方法修改其中的某个值了。

```
1  mystr = "北京外国语大学" #给定字符串
2  print(hash(mystr))#不可变序列是可以计算哈希值的
3  mylist = list(mystr)#字符串转换为列表
4
5  mylist[0] = "南"#修改列表中的元素
6  for j in mylist: #看修改是否有效果?
7      print (j)
8
9  mystr[0] = "南" #不允许修改字符串内容
```

```
-5564135343321052023
南
京
外
国
语
大
学

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-5b797fc511ea> in <module>()
      7     print (j)
      8
----> 9 mystr[0] = "南" #不允许修改字符串内容

TypeError: 'str' object does not support item assignment
```

异常信息:字符串不支持修改内容

图 3.13 字符串的内容不可变、不可修改特性

例 3.8 字符串是有序序列,因此它支持双向索引、切片、计算长度、元素访问、成员测试等。请设计一个字符串,通过切片显示子串;通过 for 循环有序地显示各位置上的字符;判断某个字符是否在字符串中。

【提示与说明】 代码实现如图 3.14 所示。注意第 2~8 行的切片方法,它也支持双向索引。可以

利用字符串的有序性，通过 for 循环顺序显示各个字符，可以使用 in 操作符进行成员测试。详见图 3.14 中的代码注释。

课堂练习

你能仿照图 3.14 中的例子，把给定的字符串通过 for 循环反向显示出来吗？

```
# 1 切片操作
name = "北京外国语大学"
print(name[0:]) #从头开始至字符串结尾
print(name[0:1]) #只要头一个字符
print(name[0:2])
print(name[:len(name)]) #从头开始至字符串结尾
print(name[-3:]) #从倒数第三个字符开始至结尾
print(name[::-1],end ='&') #直接通过逆向索引不间断地完成倒置输出
print("\n")
# 2 有序性
mystr = "附属石家庄外国语学校" #给定字符串
for i in mystr:#迭代显示字符串中各个位置上的内容
    print(i,end = "/")
print("\n")
for j in range(0,len(mystr)):
    print(mystr[j],end ="@") #通过字符串的正向索引显示其内容
print("\n")

for m in list(mystr): #转换为列表后显示
    print(m,end ="*")
print("\n")

# 3 成员测试
if name[-1] in mystr:
    print("含")
else:
    print("不含")
```

```
北京外国语大学
北
北京
北京外国语大学
语大学
学大语国外京北&

附/属/石/家/庄/外/国/语/学/校/

附@属@石@家@庄@外@国@语@学@校@

附*属*石*家*庄*外*国*语*学*校*

含
```

图 3.14　字符串的有序、双向索引等特性

3.3.2 常用的字符串内置方法

下面介绍字符串常用内置方法的使用。假设给定字符串变量 mystr，表 3.3 中的各个内置方法可以完成相应的操作。

表 3.3 字符串的查找、索引、计数、编码、解码等方法

方　法	说　明
mystr.find(givenstr, beg=0, end=len(string)) 以及 mystr.rfind(givenstr, beg=0, end=len(string))	find：从 mystr 左侧开始检测指定的字符串 givenstr 是否包含在其中，如果包含，则返回其开始时的索引位置，否则返回－1；如果指定了从 beg 开始到 end 结束的范围，则在这个范围内进行检测 rfind：从 mystr 右侧开始检测，方法同 find
mystr.index(givenstr, beg=0, end=len(string)) 以及 mystr.rindex(givenstr, beg=0, end=len(string))	index：从 mystr 左侧开始检测指定的字符串 givenstr 是否包含在其中 rindex：与 index 用法类似，只不过是从尾部(右侧)开始检测
mystr.count(givenstr, beg=0,end=len(string))	返回指定的字符串 givenstr 在给定的 mystr 中出现的次数。如果有 beg 或者 end 区间参数，则返回在指定区间内出现的次数
mystr.encode(encoding='UTF-8',errors='strict')	以 encoding 参数指定的编码格式编码 mystr 字符串；errors 是设置不同错误的处理方案，默认为 'strict'
mystr.decode(encoding='UTF-8',errors='strict')	以 encoding 参数指定的编码格式解码 mystr 字符串；errors 是设置不同错误的处理方案，默认为 'strict'

例 3.9 针对表 3.3 中的字符串方法设计字符串，对某个字符进行查找、统计某个字符在特定范围内出现的次数；分别将其按 UTF-8、GBK 进行编码和解码，观察一下按不同编码方式进行编码后的效果。

【提示与说明】 可以先通过 sys 包中的 getdefaultencoding()方法显示系统默认的字符编码方式(注意：某个方法是在对象后使用点操作符调用的)。图 3.15 展示了上面几种方法的示例。第 5、6 行

```
1  import sys
2  print(sys.getdefaultencoding())#显示系统默认编码
3  term = input("请输入要查询的文字：")
4  mystr = '北京外国语大学附属石家庄外国语学校'#给定字符串
5  print("该字符%s首次出现的位置%d："%(term,mystr.find(term))) #返回它首次出现的位置
6  print("该字符%s最末次出现的位置%d："%(term,mystr.rfind(term)))
7  print("该字符%s一共出现%d次：" %(term,mystr.count(term)))#显示出现的次数
8  print("在指定范围内出现的次数",mystr.count(term,8,len(mystr)))#在指定范围出现次数
9  mystr_utf8 = mystr.encode("UTF-8")#按UTF-8对给定字符串进行编码
10 mystr_gbk = mystr.encode("GBK")#按GBK对给定字符串进行编码
11 print("按UTF-8进行编码为：", mystr_utf8)
12 print("按GBK进行编码为：", mystr_gbk)
13 print("UTF-8 解码：", mystr_utf8.decode('UTF-8','strict'))#解码
14 print("GBK 解码：", mystr_gbk.decode('GBK','strict'))
```

```
utf-8
请输入要查询的文字：学
该字符学首次出现的位置6：
该字符学最末次出现的位置15：
该字符学一共出现2次：
在指定范围内出现的次数 1
按UTF-8进行编码为： b'\xe5\x8c\x97\xe4\xba\xac\xe5\xa4\x96\xe5\x9b\xbd\xe8\xaf\xad\xe5
\xa4\xa7\xe5\xad\xa6\xe9\x99\x84\xe5\xb1\x9e\xe7\x9f\xb3\xe5\xae\xb6\xe5\xba\x84\xe5\xa
4\x96\xe5\x9b\xbd\xe8\xaf\xad\xe5\xad\xa6\xe6\xa0\xa1'
按GBK进行编码为： b'\xb1\xb1\xbe\xa9\xcd\xe2\xb9\xfa\xd3\xef\xb4\xf3\xd1\xa7\xb8\xbd\xc
a\xf4\xca\xaf\xbc\xd2\xd7\xaf\xcd\xe2\xb9\xfa\xd3\xef\xd1\xa7\xd0\xa3'
UTF-8 解码： 北京外国语大学附属石家庄外国语学校
GBK 解码： 北京外国语大学附属石家庄外国语学校
```

图 3.15 字符串编码、查找、计数等方法的使用

代码是关于 find()、rfind()的用法，即分别从左侧和右侧查找指定的字符；在 count()计数方法中，是可以指定计数范围的(第 8 行代码，终点即字符串长度，可以使用求字符串的长度 len()方法实现)；第 9、10 行代码是字符串编码的方法。

表 3.4　字符串的大小写、拼接等方法

方　法	说　明
mystr.islower() 以及 mystr.isupper()	islower 返回字符串是否为小写。如果字符串中包含至少一个区分大小写的字符且所有区分大小写的字符都是小写，则返回逻辑真值 True，否则返回逻辑假值 False isupper 返回字符串是否为大写。如果字符串中包含至少一个区分大小写的字符且所有区分大小写的字符都是大写，则返回逻辑真值 True，否则返回逻辑假值 False
lower() 以及 upper()	转换字符串中所有字符为小写 转换字符串中所有字符为大写
mystr.join(seq)	该方法接收一个序列参数 seq，seq 是要连接的元素序列、字符串等，即指定字符串 mystr 作为分隔符，将其散播到 seq 所有的元素间

例 3.10　针对表 3.4 中的字符串方法，设计字符串判断它是否均为小写字符；若不是小写字符，则将其转换为小写字符。以指定字符串作为分隔符，将它散播到另一个字符串中并显示这个新构成的字符串。统计某个字符中的最大值、最小值、长度等。

【提示与说明】　在字符串后面跟一个点运算符再接相应的方法即可使用。图 3.16 展示了上面几个字符串常用内置方法的结果。islower()、isuper()的用法参见图 3.16 中的第 4、7 行代码；join()方法可用于散播指定的字符到原始字符串中，用法参见图 3.16 中的第 9 行代码；max()、min()、len()方法不是用点操作符，而是在其后的参数括号中填写字符串，用法参见第 11 行、第 13 行代码。注意此例显示出的字符串的不可变性。

```
1  mystr1 = "AbCdEfG"
2  mystr2 = "6712345"
3  mystr3 = "&"
4  if mystr1.islower():
5      print("都是小写字符")
6  else:
7      print("不都是小写字符，字符串mystr1小写后的结果的："+mystr1.lower())
8
9  mystr4 = mystr3.join(mystr2) #用mystr3中内容作为分隔符依次插入（连接）到mystr2中
10 print("字符串mytr2是："+mystr2+"。可见它没有被join改变本身内容哦。")
11 print("mystr2中最大的字符是："+max(mystr2)+"。最小的字符是：" +min(mystr2))
12
13 print("字符串mystr3："+mystr3+" ，长度："+str(len(mystr3))+"。没被join改变本身内容")
14
15 print("通过join新生成的字符串是："+mystr4)
```

```
不都是小写字符，字符串mystr1小写后的结果的：abcdefg
字符串mytr2是：6712345。可见它没有被join改变本身内容
mystr2中最大的字符是：7。最小的字符是：1
字符串mystr3：& ，长度：1。没被join改变本身内容
通过join新生成的字符串是：6&7&1&2&3&4&5
```

图 3.16　字符串部分内置方法使用示例

表 3.5 字符串的替换、分隔、开始或结束字符、字符串清洗等方法说明

方 法	说 明
str.replace(old, new [, max])	把字符串 str 中的 old 子串替换成新串 new。如果指定了替换次数 max，则替换次数最大不超过 max 次
str.split(s1="", num)	以子串 s1 为分隔符分割字符串 str(此例中 s1 是空格)；如果给定了 num 值，则仅截取 num+1 个子串
str.startswith(substr, beg=0,end=?) 以及 str.endwith(substr, beg=0,end=?))	startwith()检查字符串 str 是否以指定子串 substr 开头：若是则返回 True，否则返回 False；如果指定了区间范围 beg 和 end 的值，则在指定区间范围内进行检查 endwith()的用法与 startwith 的用法类似，只不过它是检测 str 是否以指定的子串 substr 结尾
strip() lstrip() rstrip()	字符串清洗方法，即删除字符串头尾指定字符(默认删除头尾空格、回车符、换行符、Tab 制表符等，也可指定删除特定字符)。strip 方法去掉原字符串左右两边的空白字符后返回新的字符串；rstrip 和 lstrip 分别去掉字符串右边和左边的空白字符后返回新的字符串

例 3.11 给定一个字符串。针对表 3.5 中的字符串方法，完成如下任务：①将其中某些字符替换为新的设定字符，并比较一下“指定次数的替换”和“不限次数的全替换”有什么区别；②分别以空格和指定的某个字符为分隔符分隔原始的字符串；③判断某个字符串是否以某个特定字符开头；④完成数据清洗。

【提示与说明】 上述方法均使用点操作符，代码实现如图 3.17 所示，主要代码后有说明文字，可以参考。使用 replace()方法替换时可以指定次数，也可不指定；使用 split()方法时，若不写分隔的参数，则默认以空格作为分隔符；将字符串迭代遍历并追加到列表的方法如第 18～20 行代码所示。需要注意的是，strip()、lstrip()、rstrip()方法均不对原字符串进行改变，而只是返回清洗后的结果，若需要保留清洗后的结果，往往需要将其赋值给其他变量。

```
1  #replace()的用法示例
2  oldstr = "basic php javascript python c c++ c# object-c" #原始字符串
3  new_with_time = oldstr.replace("c", "@", 2) #只更新指定次数
4  new_without_time = oldstr.replace("c", "@") #不限更新次数
5  print("指定更新次数后的结果是："+new_with_time)
6  print("不指定次数后的更新结果是："+new_without_time)
7  #字符串分割
8  print ("以空格为分隔符返回列表",oldstr.split())
9  print ("以空格为分隔符分隔成2+1=3个子串返回列表",oldstr.split(" ",2))
10 print ("以字符p为分隔符分隔并返回列表",oldstr.split("p"))
11 #判断某个字符串是否以某个特定字符开头
12 if oldstr.startswith("b") or oldstr.endswith("c"):
13     print("OK")
14 else:
15     print("NO")
16 #数据清洗与通过迭代方式依次替换
17 new = []
18 for i in oldstr.lstrip():
19     i = i.replace('c', 'C') #小写c替换成大写字符C
20     new.append(i)
21 print(new)
```

图 3.17 替换、分割、开始或结束字符、字符串清洗等方法的使用示例

例 3.12 从键盘输入一个英文句子，统计这句话中有多少英文单词，并计算这句话中每个单词的平均长度。例如，输入“this is an example”，需要输出单词数为 4，这 4 个单词的平均字母数是 3.75。

【提示与说明】 由于英文单词都是以空格分隔的，因此可以使用 split()方法完成单词分隔并统计数量(分隔字符就是空格)；通过 len()方法能迭代计算出每个单词的长度，通过计算，就能得到每个单词的平均字母数。代码实现如图 3.18 所示。

```
sentence = input("请输入一个英文句子：")
listofwords = sentence.split()#以空格为分隔符
print("共有：%d个单词" %len(listofwords))
sum = 0
for word in listofwords:
    sum +=len(word) #累加每个单词的长度
print("每个单词中平均含有：%4.2f" %(sum/len(listofwords))+"个字母")
```

```
请输入一个英文句子：this is an example
共有：4个单词
每个单词中平均含有：3.75个字母
```

图 3.18 split()方法使用示例

3.4 列表和元组

3.4.1 列表和元组的主要异同点

列表的使用非常广泛，它是升级版的“数组”；元组可以看成“轻量级”列表，虽然它有和列表不一样的特点(如元素的不可变性等)，但从表面看，列表、元组内的各个元素都是以逗号隔开的。元组除了定界符是圆括号外，好像与列表很相似。其实，元组与列表还是有区别的，主要区别是列表是动态的，可增、删、改其元素；但元组是静态的，内容不可变。因此，如果定义了一系列常量值且仅是对它们进行遍历而不能对其元素进行任何修改，可以使用元组而不用列表存储。在某些情况下(如在要求不能改变数据本身等应用场合下)使用元组更合适、更安全。由于列表和元组有一些异同点，为方便同学们更好地理解和区分二者的操作，这里将列表和元组放在一起介绍。

首先看看相同点：列表和元组都是 Python 内置的用来存储一连串元素的序列化容器，相同点如下。

(1) 都属于有序序列，从最左边第一个元素开始往右排序，序号分别是 0，1，2，…；从最右边第一个开始往左排序，序号分别是－1、－2、－3，…，详见图 3.11 的切片操作。但对于无序的集合来说是无法进行切片操作的，这是因为集合的所有元素均没有索引，切片也就无从谈起。

(2) 使用内置函数 len()统计元素个数，max()求最大值，min()求最小值，sum()求数字元素和，运算符 in 测试是否包含某个元素(返回一个逻辑值)，count()统计指定元素的出现次数，index()获取指定元素首次出现的位置；元组也可以像列表那样作为 map()、filter()、zip()、reduce()等函数的参数。

(3) 列表和元组中的元素可重复(这点与集合元素不同)；元素可以是同一类型的，也可以是不同类型的，例如可分别为整数、实数、字符串等基本类型，甚至可以是列表、元组、字典、集合以及其他自定义类型的混合对象；可以在列表及元组中再嵌套列表、元组、字典、集合等序列化数据；可以分别使用 list()和 tuple()方法分别将其他类型的数据转换为列表和元组，请参考图 3.6 和图 3.11 中的代码。

下面介绍列表和元组的切片方法。切片会返回列表或元组的某个元素“子集”。假设有一个名为 x 的列表(或元组)，x[start：end[：step]]会返回从 start 开始到 end 结束的、以 step 为步长的一个列表“子集”。x[start：end[：step]]外层的中括号表示切片符，即使是元组，切片也用方括号而不用圆括

号；而其内部的 step 步长的中括号【 】是可选项，表示 step 这个参数可省略不写，当步长 step 为空时，默认 step 为 1；其实，这里的起始 start、终止 end 和步长 step 参数也都可以为空：当起始参数 start 为空时，默认从列表（或元组）头开始（首位置的索引为 0），但当起始位置 start 大于列表（或元组）总长度时会返回空；当终止参数 end 为空时，默认到列表（或元组）的尾部，但当终止参数 end 大于原列表（或元组）的总长度时，会返回列表的全部数据。可见，基于 x[start：end【：step】]的列表（或元组）切片操作可以方便地对列表（或元组）x 中的数据进行各种抽取、反转等切片操作。

例 3.13 给定一个含有一组自然数的列表或元组。请完成如下操作：

① 显示列表或元组的长度；

② 分别只返回奇数位置和偶数位置的列表或元组的子元素；

③ 使用列表或元组的切片操作，分别显示列表或元组的总长度、全部原始数据、倒序的原始数据、奇数位置的数据、偶数位置的数据、指定范围内的数据、切片列表前 n－1 个元素后的结果。

【提示与说明】 ①使用 len(x)方法可以显示列表或元组 x 的长度。②若返回奇数位置和偶数位置的数据，只需分别把首位置变更一下即可。③若在 print()中将说明性的字符和列表或元组拼接在一起显示，则需要将列表或元组 x 通过 str(x)方法完成其由列表或元组类型到字符串类型的转换。切片列表结果如图 3.19 所示。

```
myList = [3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
mytuple = ("hello", "ok", 3.14, ["a","b","c"], 7.68, "world")
print("该列表总长度是："+str(len(myList)))
print("该元组总长度是："+str(len(mytuple)))
print("列表全部数据是："+str(myList[::])+"元组全部数据是："+str(mytuple[::]))
print("列表倒排结果是："+str(myList[::-1])+"元组倒排结果是："+str(mytuple[::-1]))
print("列表奇数位置的数据是："+str(myList[::2]))#起始位置为0，返回奇数位置的数据
print("元组偶数位置的数据是："+str((mytuple[1::2])))#起始位置为1，返回偶数位置的数据
print("列表指定范围结果："+str(myList[3:6])+"元组指定范围结果："+str(mytuple[0:3]))
print("列表删除了最后一位后的结果是：",myList[:-1])
print(myList[0:100]) #当终止位置大于总长度时，返回全部
print(mytuple[100:]) #当起始位置大于总长度时，返回空
```

```
该列表总长度是：10
该元组总长度是：6
列表全部数据是：[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]元组全部数据是：('hello', 'ok', 3.14, ['a', 'b', 'c'], 7.68, 'world')
列表倒排结果是：[17, 15, 13, 11, 9, 7, 6, 5, 4, 3]元组倒排结果是：('world', 7.68, ['a', 'b', 'c'], 3.14, 'ok', 'hello')
列表奇数位置的数据是：[3, 5, 7, 11, 15]
元组偶数位置的数据是：('ok', ['a', 'b', 'c'], 'world')
列表指定范围内的结果是：[6, 7, 9]元组指定范围内的结果是：('hello', 'ok', 3.14)
列表删除了最后一位后的结果是： [3, 4, 5, 6, 7, 9, 11, 13, 15]
[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]
()
```

图 3.19 列表（元组）的切片操作使用示例

课堂练习

你试一试，看如何能把 range()对象转换为一个列表并分别显示倒数第一个、倒数第三个元素？

列表和元组的主要不同点如下。

（1）列表定界符是一对方括号[ ]，使用[ ]表示一个空列表；元组定界符是一对圆括号()，使用()表示一个空列表。虽然元组的元素访问和列表一样都是使用方括号，但元组返回的仍然是元组对象。

（2）列表是可变的序列，当列表的元素增加或删除时，列表对象会自动进行扩展或收缩；列表中的元素值可以修改，也可以在其尾部追加新元素。但元组是不可变的，因此没有增加元素、修改元素、删除元素的方法，如果要对元组进行排序，可使用内置函数 sorted(元组对象)并生成新的列表对象(这是因为原始元组是不可变的)。

（3）列表、元组均支持切片操作；对于元组而言，只能通过切片访问元组中的元素，不允许使用切片修改元组中元素的值。

（4）元组、列表都支持运算符＋。对于元组来说，如果只有一个元素，也要加一个逗号，如(3,)，仅有一个整型或实数元素后无逗号的元组相当于直接定义为非元组的整型或实数，因此，若定义为元组，即使只有一个元素，后面也要加上逗号。但对于列表来说则没有这种特性。如图 3.20 所示，注意第 2 行代码和第 5 行代码的区别。另外，请注意出错信息。

```
1 x=(3.14) #仅有一个整型或实数元素后无逗号的元组，相当于直接定义为非元组的整型或实数
2 y=(12,) #若定义为元组，即使只有一个元素，后面也要加上逗号
3 print(type(x),type(y))
4 a = [3.14]
5 b = [12]
6 print(type(a), type(b))
7 merge_tuple = x+y
8 merge_list = a+b
9 print(merge_tuple, merge_list)
```

```
<class 'float'> <class 'tuple'>
<class 'list'> <class 'list'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [33], in <cell line: 7>()
      5 b = [12]
      6 print(type(a), type(b))
----> 7 merge_tuple = x+y
      8 merge_list = a+b
      9 print(merge_tuple, merge_list)

TypeError: unsupported operand type(s) for +: 'float' and 'tuple'
```

图 3.20　元组元素的定义

课堂练习

1. 请你把 range()对象转换的列表元素的倒数第二个元素赋予一个新值，再使用列表的 append()方法在末尾追加一个新元素。

2. 由 range()对象结果生成一个列表。要求：(1)生成其中从第 3 个元素到最后的切片序列；(2)生成其中从头到第 3 个元素的切片序列；(3)生成最后三个子字符串。

3.4.2　列表和元组的常用方法

列表和元组有一些方法是可以共用的。可以通过 Python 提供的一些方法完成对列表中元素的增、删、改、弹出、返回索引、逆序、排序等操作。当然，有些方法元组是无法使用的，如增、删、改操作等，表 3.6 中列出的方法对于元组而言均不可用。

1. 增、删、改列表数据的部分常用方法

增、删、改列表数据的部分常用方法如表 3.6 所示。

表 3.6 增、删、改列表数据的部分常用方法说明

方　法	说　明
append(x)	将某个元素 x 添加至列表的尾部
extend(A)	将列表 A 中的所有元素添加至当前列表的尾部
insert(index, x)	在列表指定位置 index 的前面添加元素 x,从这个 index 位置往后的所有元素均后移(索引号增加)
remove(x)	在列表中删除首次出现的 x 元素,删除后该元素之后的所有元素前移一个位置,若同一值在序列中多次出现,只移除第一个
pop([index])	弹出(删除并返回)列表中下标为 index 的元素,如果缺省 index 参数,则默认其为-1,即弹出列表的最后一个元素
clear()	删除列表中的所有元素,但保留列表对象(原列表成为空列表)

在增加列表数据的方法中,append(x)用于向列表尾部追加元素 x;extend(A)用于将另一个列表 A 中的所有元素都追加至当前列表的尾部;insert(index, x)用于向列表的指定位置 index 的前面插入元素 x。在删除数据的方法中,remove(x)用于删除列表中第一个与指定值相等的元素;pop([index])用于删除并返回指定位置(默认是最后一个)的元素;clear()用于清空列表,另外,可以使用 del()方法删除列表中指定位置的元素。

例 3.14 建立一个新列表并赋初值;分别通过 append()、extend()、insert()方法扩充原来列表的内容。之后,对扩充后的列表进行元素去重、排序操作。

【提示与说明】 代码实现如图 3.21 所示,可以从键盘输入一些数据以自定义一个列表,注意这些数据之间要用分隔符隔开,图 3.21 中所示是采用空格作为分隔符,因此用到了 split()方法;通过 x1.append()增加新的内容后,原来的 x1 列表末尾就追加了这些新元素;x1.extend(x2)是将 x2 列表中的内容全部顺序追加到 x1 列表的尾部,注意列表中的元素是可以重复的;由于列表内容是可变的,因此可以使用 insert()方法在特定位置追加新的元素。之后,利用 not in 操作判断重复元素以完成去重操作。对于不重复的元素使用 append()方法追加到新列表中并使用 sort()方法进行排序。

```
list1= list(map(int, input("请输入列表中的各个数值, 用空格隔开").split()))
list2 = [3,4,5,6,7,8,9]
list1.append(45)
list1.extend(list2)
list1.insert(-1, 100) #在最后元素的前面插入新的元素
print("扩充后的列表:",list1)
result = []
for item in list1:
        if item not in result:
            result.append(item)
            result.sort()
print("排序去重后的列表:",result)
```

```
请输入列表中的各个数值, 用空格隔开3 4 5 6 7 8 9
扩充后的列表: [3, 4, 5, 6, 7, 8, 9, 45, 3, 4, 5, 6, 7, 8, 100, 9]
排序去重后的列表: [3, 4, 5, 6, 7, 8, 9, 45, 100]
```

图 3.21 列表中插入元素、去重元素、排序元素使用示例

课堂练习

利用 range(20)方法生成 20 以内的数;通过 list()方法将其转换为列表并赋予变量 x;再通过 y = [ ]定义另一个列表 y;利用 for 循环依次遍历 x 中的各个元素,并对每次遍历到的元素加 5 后,追加到列表 y 中。

例 3.15 对于一个已经定义好的列表，逐次弹出排在队尾的元素。

【提示与说明】 可通过 len()方法求得原列表的长度；通过 for 循环，依次使用 pop()方法弹出队尾的元素，如图 3.22 所示(x1 列表中已经提前存储了字符串、浮点数、表达式、嵌套列表等多种元素)。从图 3.22 中可以看出，弹出所有元素后，原列表长度为 1；执行 clear()方法后，该列表长度才清零。

```
x1 = ["排球", "手球", "篮球", "北京大学", "清华大学", 3.14, 12*2, [1,2,"OK"]]
print("原始队列的长度是%d" %len(x1)) #len()为求其长度
for i in range(1,len(x1)):
    x = x1.pop()            #弹出（删除）排在列表最尾部的元素
    print("弹出的第%d个元素是：%s" %(i,x))
print("弹完后队列的长度是%d" %len(x1))
x1.clear()
print("清空队列后的长度是%d" %len(x1))
```

```
原始队列的长度是8
弹出的第1个元素是：[1, 2, 'OK']
弹出的第2个元素是：24
弹出的第3个元素是：3.14
弹出的第4个元素是：清华大学
弹出的第5个元素是：北京大学
弹出的第6个元素是：篮球
弹出的第7个元素是：手球
弹完后队列的长度是1
清空队列后的长度是0
```

图 3.22 依次弹出列表末尾元素

例 3.16 从键盘输入一些数字组成一个列表，再从键盘输入部分数字，将原列表中和这部分数据重复的数据删除(删除输入部分的数据)。

【提示与说明】 此题是练习 remove()的用法，代码实现如图 3.23 所示。

```
list1= list(map(int, input("请依次输入列表中的各个数值，用空格隔开").split()))
list2= list(map(int, input("请再输入你想删除的那几个数值，用空格隔开").split()))
for item in list2:
    list1.remove(item)
print(list1)
```

```
请依次输入列表中的各个数值，用空格隔开4 6 7 8 9 12
请再输入你想删除的那几个数值，用空格隔开7 9
[4, 6, 8, 12]
```

图 3.23 remove()方法使用示例

2. 和元素计数、排序、复制有关的部分常用方法

和元素计数、排序、复制有关的部分常用方法如表 3.7 所示。

表 3.7 和元素计数、排序、复制有关的方法说明

方 法	说 明
index(x)	返回列表或元组中第一个值为 x 的元素的下标；若不存在值为 x 的元素，则抛出异常信息
count(x)	返回指定元素 x 在列表或元组中出现的次数
x.reverse()	对列表 x 中的所有元素逆序排序(元组无此方法)

方　法	说　明
x.sort(key = None, reverse=False)	对列表 x 中的元素进行排序，这里的 key 是一个关于排序策略的函数；reverse=False 为升序，reverse=True 为降序 （元组无此方法）
copy()	返回复制的列表（元组无此方法）

对表 3.7 中部分方法的解释如下。

（1）index()和 count()两个方法都是和列表或元组的索引和位置有关的：index(x)用于返回指定元素 x 在列表或元组中首次出现的位置，如果该元素不在列表或元组中，则抛出异常；count(x)用于返回列表或元组中指定元素 x 出现的总次数。

（2）reverse()和 sort()方法都是和列表排序有关的：reverse()方法用于将列表所有元素逆序排序；sort()方法用于**按照指定排序函数 key** 对所有元素进行排序；reverse()和 sort()方法都是用处理后的数据替换原来的数据，原列表或元组地址不变且没有返回值。

例 3.17　不使用 reverse()方法，对输入的列表数据进行逆序输出，即第 0 个和第 n－1 个互换，第 2 个和第 n－2 个互换，以此类推，一直到中间的那个数据为止。

【提示与说明】 需要用到 for 循环，循环次数为元素个数的一半，循环体所做的事情就是对找到的首尾两个数据进行交换，为此需要找到中间的那个数据的索引。如图 3.24 所示，“//”为向下取整操作符。

```
1  mylist= list(map(int, input("请输入列表中的各个数值，用空格隔开").split()))
2  for i in range(0, (len(mylist)-1)//2+1):
3      temp =  mylist[i]
4      mylist[i] = mylist[len(mylist)-i-1]
5      mylist[len(mylist)-i-1] = temp
6  print(mylist)
7
8
```

```
请输入列表中的各个数值，用空格隔开3 5 1 5 8 9 2
[2, 9, 8, 5, 1, 5, 3]
```

图 3.24　数据排序与交换使用示例

请思考一下，图 3.24 中第 2 行代码 range()的上限为什么是(len(mylist)－1)// 2 ＋1？如果不加 1，会出现什么情况？

例 3.18　首先生成一个具有不同长度元素的列表（元组），并按数“位”的长度分别进行逆序（位数多的排在前面，位数少的排在后面）排序；其次将其逆序排序，即位数少的排在前面，位数多的排在后面；最后统计指定的某个元素首次出现的索引位置及其出现的总次数。

【提示与说明】 代码实现如图 3.25 所示。这里需要用到 index()、count()、sort()等方法。列表元素排序时需要得到每个元素的长度，因此可将其作为排序的 key 值，第 5 行代码使用了 lambda 匿名函数（将在第 4 章介绍）；第 6 行代码显示的是重新排序后的列表，说明原始列表中元素的顺序已经发生了变化，执行第 7 行代码的结果是对这个新顺序的逆序排序，这印证了 reverse()和 sort()方法都是用处理后的数据替换原来的数据，这也验证了方法 sort()的“原地”操作特性，即用处理后的数据替换原