第3章 组合数据类型

在Python语言编程中，既需要独立的变量来保存一份数据，也需要组合数据类型来保存大量数据。组合数据类型是指将多个数据有效组织起来并统一表示的数据类型。Python语言中的组合数据类型可以分为序列类（字符串、列表、元组）、集合类（集合）、映射类（字典），其中字符串和元组是不可变序列，其余均是可变序列。不可变序列在内存中是不会改变的，如果对不可变序列做出改变，则会将该序列存储在新的地址空间中，而原来的序列将会被系统回收。可变序列发生改变时，是不会改变地址的，而是改变内存中的值，或者是扩展内存空间。

3.1 序列类——列表与元组

序列（sequence）是Python语言中常用组合数据类型。它是一块可存放多个数据的连续内存空间，通常这些数据按一定顺序排列，可以通过每个数据所在位置的编号（称为索引）进行访问。序列就好比是一家旅馆，旅馆中的每个房间就如同序列存储数据的一个个内存空间，每个房间所特有的房间号就相当于索引值。正如可以通过房间号找到这家旅馆中的每个房间，访问到房间内居住的旅客一样，也可以通过索引找到序列中的每个内存空间，获取其中存储的数据。第2章中学习过的字符串就是一种序列，可以直接通过索引访问字符串内的字符。

Python语言中的序列支持几种通用的操作，包括索引、切片、相加、相乘、检查是否包含某个元素等。

3.1.1 序列的基本操作

1. 序列索引

如前所述，序列中的每个元素都有属于自己的编号，即索引。Python语言支持正、负两种索引值标注索引的方式。正索引值是从起始元素开始，第一个索引值是0，第二个索引值是1，以此类推，直到最后一个元素，如图3-1所示。

	元素1	元素2	元素3	…	元素n-1	元素n
索引（下标）	0	1	2	…	n-2	n-1

图3-1 序列索引值示意图

此外，Python语言还支持索引值是负数的方式。此类索引是从右向左计数，也就是说，从最后一个元素开始计数，倒数第一个元素的索引值是－1，倒数第二个索引值是－2，以此

类推直到第一个元素，如图 3-2 所示。

	元素1	元素2	元素3	…	元素n-1	元素n
索引（下标）	$-n$	$-(n-1)$	$-(n-2)$	…	-2	-1

图 3-2　序列负索引值示意图

无论是正索引值还是负索引值，都可以非常方便地用来访问序列中的每个元素。不过要注意，使用正值作为序列各元素的索引值时，是从 0 开始计数的，而使用负值作为序列索引值时，是从－1 开始计数的。

【例 3-1】 以字符串类型的序列为例，访问序列的首尾元素。

【参考代码】

```
str = "Hello Python"              # 定义字符串
print(str[0], " = ", str[-12]) # 访问第一个元素
print(str[11], " = ", str[-1]) # 访问最后一个元素
```

程序执行结果：

```
H = H
n = n
```

2. 序列切片

利用索引操作可以访问序列中的一个元素，而通过切片操作可以一次性访问序列中的若干元素，生成一个新序列。

【语法格式】

```
seqname[start : end : step]
```

【说明】

完成序列的切片，其中各个参数的含义分别如下。

（1）seqname：表示序列的名称。

（2）start：表示切片开始的索引位置（包括该位置），此参数也可以不指定，会默认为 0，也就是从序列的开头进行切片。

（3）end：表示切片结束的索引位置（不包括该位置），如果不指定，则默认为序列的长度，即序列尾。

（4）step：当 step 大于 0 时，表示从左向右取字符；当 step 小于 0 时，表示从右向左取字符。step 的绝对值减 1，表示每次取字符的间隔数，即当 step 的值不为 1 时，在进行切片取序列元素时，会“跳跃式”地取元素。如果省略设置 step 的值，则最后一个冒号可以省略。

【演示】 对字符串"1234567890"进行切片。

```
>>> str = "1234567890"
>>> str[0:5]          # 取索引区间为 0～5(不包括索引值 5)的字符的字符串
    '12345'
>>> str[:5]           # 默认初始为 0,相当于 str[0:5]
    '12345'
```

```
>>> str[5:]          # 取索引区间从 5 开始,到字符串尾之间的字符串
    '67890'
>>> str[:]           # 输出整个字符串
    '1234567890'
>>> str[0:5:1]       # 在索引值为 0～5(不包括索引值 5)的字符中,正向取每个字符
    '12345'
>>> str[5:0:-1]      # 在索引值为 0～5(不包括索引值 0)的字符中,反向取每个字符
    '654321'
>>> str[0:5:-1]      # 在索引值为 0～5 的字符中,反向每 1 个字符取 1 个字符
    ''                # 因为索引值开始小于结束,无法反向取,所以输出为空
>>> str[::3]         # 在整个字符串中,正向每 3 个字符取第 1 个字符
    '1470'
>>> str[::-3]        # 在整个字符串中,反向每 3 个字符取第 1 个字符
    '0741'
>>> str[::-1]        # 在整个字符串中,反向取每个字符
    '0987654321'
```

这里通过 s[::－1]可以方便地求出指定序列的逆序排列。

【例 3-2】 利用切片方法,根据输入的 1～7 的整数值,输出对应的星期,其中 7 对应星期日。

【分析】

将所有星期缩写组成一个序列,计算数字对应星期缩写字符的位置,利用切片进行提取。

【参考代码】

```
week1 = "MonTueWesThuFriSatSun"
week2 = "星期一星期二星期三星期四星期五星期六星期日"
w = int(input("请输入 1～7 的整数:"))
weekday = (w－1) * 3
print(week1[weekday:weekday + 3],week2[weekday:weekday + 3])
```

程序执行结果:

```
请输入 1～7 的整数:7
Sun 星期日
```

3. 序列相加

Python 语言支持两个类型相同的序列使用“＋”运算符实现两个序列的相加操作,它完成两个序列的拼接,若采用多个“＋”可以连接多个字符串。

【演示】 采用“＋”运算符连接字符串。

```
>>> classname = "Python 语言程序设计"        # 变量赋值
>>> print("课程名称:" + classname)           # 利用" + "运算符输出
    课程名称:Python 语言程序设计
```

4. 序列相乘

Python 语言还可以使用数字 n 与一个序列相乘获得一个新的序列,新序列的内容为原来序列元素被重复 n 次的结果。

【演示】 输出字符串 classname 重复 3 次后的结果。

```
>>> classname = 'Python 语言程序设计'            # 变量赋值
>>> print(classname * 3)
    Python 语言程序设计 Python 语言程序设计 Python 语言程序设计
```

5. 序列元素的查找

如果需要检查一个序列中是否包含某个元素，那么可以使用运算符 in 来实现。

【语法格式】

```
value in sequence
```

【说明】

value 表示要检查的元素，sequence 表示序列，根据查找结果返回 True 或者 False。

【演示】 为了检查字符串"Hello Python"中是否包含字符'e'，可以执行如下代码：

```
>>> str = "Hello Python"
>>> print('e' in str)
    True
```

和 in 功能恰好相反的运算符是 not in，它用来检查某个元素是否不包含在指定的序列中，例如：

```
>>> str = "Hello Python"
>>> print('e' not in str)
    False
```

6. 序列内置函数

除了上述序列操作外，Python 语言提供了一些内置函数以支持更多的序列操作，如表 3-1 所示，其中包括计算序列长度以及获取序列中的最大或最小元素的方法等。

表 3-1 常用内置序列处理函数

函　数	描　述
sum()	求序列中所有值的和
max()	求序列中的最大值
min()	求序列中的最小值
len()	求序列的长度
str()	把序列格式转换为字符串
list()	把序列格式转换为列表
reversed()	把序列中的所有元素进行逆序
sorted()	把序列中的所有元素进行排序
enumerate()	把序列组合成一个索引序列，一般在 for 循环中

【例 3-3】 序列内置函数的运用。

【参考代码】

```
str = "Python"
print(len(str))          # len()返回序列长度 6
```

```
print(max(str))          # max()返回序列中最大的元素 y
print(min(str))          # min()返回序列中最小的元素 P
print(sorted(str))       # sorted()对序列中的元素进行排序,返回列表
```

程序执行结果：

```
6
y
P
['P', 'h', 'n', 'o', 't', 'y']
```

3.1.2 列表

列表(list)是一种常见的序列,它是一种有序的集合,可以随时添加或删除其中的元素。

1. 列表的创建

使用方括号创建一个列表,只要将用逗号分隔的不同数据项括起来即可。

【语法格式】

```
listname = [[value1][,value2][,value3]…]
```

【说明】

listname 为列表名,命名符合标识符规则。方括号中的每个数据项 value 都称为列表元素,元素之间用“,”进行分隔。元素的个数称为列表的长度。因为列表存储的是多个数据,通常 listname 采用复数形式。若列表表示单个对象的多类信息时,也考虑采用单数命名。当然也可以创建空列表。

【演示】 利用列表保存班里同学的姓名。

```
>>> classmates = [ ]                              # 创建一个空列表
>>> classmates
    [ ]
>>> classmates = ['LiMing', 'WangFang', 'QianWei']  # 创建一个包含 3 个元素的列表
>>> classmates
    ['LiMing', 'WangFang', 'QianWei']
```

这里变量 classmates 就代表了一个列表,其中的元素均是字符串。

另外,还可以使用 list()函数来创建列表,它可以将其他类型的数据转换为列表类型。

【演示】 list()函数创建列表。

```
>>> list()        # 创建一个空列表
    [ ]
>>> list("Python")    # 将字符串转换为列表
    ['P', 'y', 't', 'h', 'o', 'n']
```

事实上列表中的元素并不需要是相同的类型,可以是任意 Python 语言中的基本数据类型或者是自定义的数据类型。

【演示】 多种类型的列表元素。

```
>>> student = ['LiMing', '男', 20, 2022]
>>> student
    ['LiMing', '男', 20, 2022]
```


列表 student 中的元素包含了字符串和整数两种类型的数据，甚至可以在列表中嵌套另一个列表。

【演示】 列表嵌套。

```
>>> student1 = ['LiMing', '男', 20, 2022]
>>> student2 = ['WangFang', '女', 19, 2022]
>>> student = [student1, student2]
>>> student
    [['LiMing', '男', 20, 2022],['WangFang', '女', 19, 2022]]
```

2. 基本操作

列表作为一种序列，序列的所有特性、操作和内置函数对于列表都是成立的。同时列表还具有自身特殊的操作。

1）访问列表

列表中的每个元素都对应一个位置编号，这个位置编号称为元素的索引。列表也可以通过索引和切片来访问列表中的元素。

【语法格式】

```
listname[Index]
```

【说明】

listname 为列表名，符合标识符命名规则。Index 为索引值，若 Len 为列表长度，则 Index 的取值范围为 0～Len－1。注意，当 Index＞Len－1 时，会出现下标越界错误，即下标超出了可表示的范围。

【演示】 列表的访问。

```
>>> classmates = ['LiMing', 'WangFang', 'QianWei', 'ZhaoQian']
>>> classmates[1]          # 通过索引读取第二个元素
    WangFang
>>> classmates[1:-1]       # 通过切片截取从第二个到倒数第一位(不包含该位)元素
    ['WangFang', 'QianWei']
```

嵌套列表即列表中含有多个子列表，列表的元素为子列表，可以通过多层索引来访问各列表中的元素，采用双中括号表示。

【语法格式】

```
listname[Index1][Index2]
```

【说明】

listname 为列表名，Index1 代表嵌套列表中的第 Index1 个子列表，Index2 代表第 Index1 个子列表中的第 Index2 个元素，Index1 和 Index2 均从 0 开始。

【演示】 嵌套列表元素的访问

```
>>> student = [['LiMing', '男', 20, 2022],['WangFang', '女', 19, 2022]]
>>> student[0]
    ['LiMing', '男', 20, 2022]
>>> student[1][0]
    'WangFang'
```

2）更新列表

和字符串不同的是，列表中的元素是可以随时修改的，因此还可以直接对列表的某一个元素赋值，完成更改列表元素的值。

【语法格式】

```
listname[Index] = Value
```

【说明】

listname 为列表名，Index 为索引值。Index 取值范围是 0～len－1。

【演示】 更新列表中单个元素的元素值。

```
>>> classmates = ['LiMing', 'WangFang', 'QianWei', 'ZhaoQian']
>>> classmates[1] = 'SunLiang'     # 修改第二个元素
>>> classmates
    ['LiMing', 'SunLiang', 'QianWei', 'ZhaoQian']
```

除了给一个元素赋值外，Python 语言还支持通过切片语法给一组元素赋值，即将切片获取的一组字符串用新值替换。在进行这种操作时，如果不指定步长（step 参数），Python 语言就不要求新赋值的元素个数与原来的元素个数相同。这意味着该操作既可以向列表中添加元素，也可以删除列表中的元素。

【演示】 更新列表中多个元素的元素值。

```
>>> classmates = ['LiMing', 'WangFang', 'QianWei', 'ZhaoQian', 'ZhouWei', 'WuYue']
>>> classmates[1:4] = ['SunLiang', 'ZhaoLin']     # 切片方式修改一组数据
>>> classmates
    ['LiMing', 'SunLiang', 'ZhaoLin', 'ZhouWei', 'WuYue']
```

如果对空切片赋值，就相当于插入一组新元素。

【演示】 插入元素值。

```
>>> classmates = ['LiMing', 'WangFang', 'QianWei', 'ZhaoQian']
>>> classmates[2:2] = ['SunLiang','ZhaoLin']   # 在索引值为 2 的位置插入元素
>>> classmates
    ['LiMing', 'WangFang', 'SunLiang', 'ZhaoLin', 'QianWei', 'ZhaoQian']
```

注意，使用切片语法赋值时，Python 语言不支持赋单个值。

【演示】

```
>>> scores = [89, 90, 76, 87]
>>> scores[2:2] = [67, 75]
>>> scores
    [89, 90, 67, 75, 76, 87]
```

```
>>> scores[2:2] = 82          # 赋单个值出错
    Traceback (most recent call last):
     File "< pyshell # 120 >", line 1, in < module >
       scores[2:2] = 82
    TypeError: can only assign an iterable
```

但是若使用字符串赋值，Python 语言会自动把字符串转换为序列，其中的每个字符都是一个元素。

【演示】

```
>>> classmates = ['LiMing', 'WangFang', 'QianWei']
>>> classmates[2:2] = 'SunLiang'
>>> classmates
    ['LiMing', 'WangFang', 'S', 'u', 'n', 'L', 'i', 'a', 'n', 'g', 'QianWei']
```

使用切片语法时也可以通过 step 参数指定步长，但这时就要求所赋值的新元素个数与原有元素个数相同。

【演示】

```
>>> classmates = ['LiMing', 'WangFang', 'QianWei', 'ZhaoQian', 'ZhouWei', 'WuYue']
>>> classmates[1:6:2] = ['SunLiang', 'ZhengYue', 'FengYan']      # 为第 1,3,5 元素赋值
>>> classmates
    ['LiMing', 'SunLiang', 'QianWei', 'ZhengYue', 'ZhouWei', 'FengYan']
```

3）删除列表或列表元素

可以使用 del 语句来删除整个列表或列表中的指定元素。

【语法格式】

```
del listname[< Index >]
```

【说明】

删除列表中指定的元素，当 Index 省略时，表示删除整个列表。

【演示】 删除列表或列表元素。

```
>>> classmates = ['LiMing', 'WangFang', 'QianWei','ZhaoQian','ZhouWei','WuYue']
>>> classmates
    ['LiMing', 'WangFang', 'QianWei', 'ZhaoQian', 'ZhouWei', 'WuYue']
>>> del classmates[2]          # 删除索引值为 2 的第三个元素
>>> classmates
    ['LiMing', 'WangFang', 'ZhaoQian', 'ZhouWei', 'WuYue']
>>> del classmates             # 删除整个列表
>>> classmates
    Traceback (most recent call last):
     File "< pyshell # 24 >", line 1, in < module >
     classmates
    NameError: name 'classmates' is not defined. Did you mean: 'classmethod'?
```

注意，无论是对元素进行访问、赋值还是删除，都不能超过列表的索引范围，否则程序会出现 IndexError 异常。

4）列表运算

列表也可以应用序列中的相加、相乘、检查元素的功能。列表相加即列表组合，完成多个列表元素取出重组获取新列表的过程。列表相乘完成的是列表重复的过程。利用成员运算符 in 可以判断指定元素是否在列表中。

列表的主要运算有：

（1）加法。使用加号运算符可以实现列表的连接操作，操作结果生成一个新列表，其列表元素来源于原来的两个列表。

（2）乘法。用整数 n 乘以一个列表可以生成一个新列表，即原来列表的每个元素在新列表中重复 n 次。

（3）比较。使用关系运算符可以对两个列表进行比较。比较的规则如下：首先比较两个列表的第一个元素，如果这两个元素相等，则继续比较下面两个元素；如果这两个元素不相等，则返回这两个元素的比较结果；重复这个过程，直至出现不相等的元素或比较完所有元素为止。

（4）检查元素。使用 in 运算符可以判断一个值是否包含在列表中。

（5）遍历列表。通过 while 循环或 for 循环实现访问列表中的每个元素。

【演示】 列表运算示例。

```
>>> classmates1 = ['LiMing', 'WangFang', 'QianWei']
>>> classmates2 = ['ZhouWei', 'ZhaoQian','WuYue']
>>> classmates = classmates1 + classmates2   # 列表相加
>>> classmates
    ['LiMing', 'WangFang', 'QianWei', 'ZhouWei', 'ZhaoQian', 'WuYue']
>>> classmates1 * 2                          # 列表相乘
    ['LiMing', 'WangFang', 'QianWei', 'LiMing', 'WangFang', 'QianWei']
>>> 'ZhouWei' in classmates                  # 判断元素'ZhouWei'是否在列表 classmates 中
    True
>>> for c in classmates: print(c, end = " ")   # 遍历列表元素
    LiMing WangFang QianWei ZhouWei ZhaoQian WuYue
```

列表在进行乘法运算时，还可以实现初始化指定长度列表的功能。

【演示】 创建一个长度为 5 的空列表。

```
>>> emtylist = [None] * 5
>>> emtylist
    [None, None, None, None, None]
```

5）内置函数

序列内置函数均适用于列表类型，包括 len()、max()、min()、sorted()等。

【演示】 内置函数 len()。

```
>>> len(['LiMing', 'WangFang', 'QianWei'])   # 返回列表长度
    3
```

同时，对列表还可以使用 sum()内置函数获取所有元素之和。这在统计任务中非常常见。但是要注意，做求和操作的列表元素必须都是数字，也就是序列中的元素类型都必须是数字，而不能是字符或字符串型，否则该函数运行时将产生异常，因为解释器无法判定是要

做求和操作还是要做连接操作(“+”运算符可以连接两个序列)。

【演示】 内置函数 sum()。

```
>>> scorelist = [90,89,92,93,78]
>>> sum(scorelist)          # 返回列表所有元素和
    442
>>> classmates = ['LiMing', 'WangFang', 'QianWei']
>>> sum(classmates)         # 字符串列表 sum()运行出错
    Traceback (most recent call last):
      File "<pyshell# 158>", line 1, in <module>
        sum(classmates)
    TypeError: unsupported operand type(s) for + : 'int' and 'str'
```

注意,内置函数还可用于后面介绍的元组、字典和集合类型,用法与此相同。

3. 列表方法

Python 语言为列表类型设计了丰富的方法,能够支持不同列表的各种功能。若有列表 listname,则其常用方法如表 3-2 所示。

表 3-2 列表的常用方法

方 法	描 述
listname.append(obj)	在列表 listname 末尾添加新的对象 obj
listname.extend(seq)	将另一个序列 seq 中的所有元素依次添加到 listname 的末尾
listname.insert(index,obj)	将对象 obj 插入列表 listname 的 index 索引位置上
listname.pop([index=-1])	删除列表 listname 中索引值为 index 的元素,默认删除索引值为-1 的元素,即最后一个元素,并返回该元素值
listname.remove(obj)	移除列表 listname 中值为 obj 的第一个元素
listname.index(obj[,start[,end]])	从列表 listname 的指定范围(从 start 到 end)中,找出值为 obj 的第一个元素的位置
listname.count(obj)	统计 obj 在列表 listname 中出现的次数
listname.reverse()	将列表 listname 中的所有元素反向排序
listname.sort(key=None,reverse=False)	对列表 listname 中的元素按照 key 进行排序,reverse 指定是否为降序,默认为升序排列
listname.clear()	清空列表 listname
listname.copy()	复制列表 listname

1) append()方法

append()方法用于在列表的末尾追加新的元素。

【语法格式】

```
listname.append(Object)
```

【说明】

listname 为列表名,Object 可以是普通元素也可以是列表,无论其是哪种类型都将作为一个元素追加到列表中。

【例 3-4】 利用 append()方法追加列表元素。

【参考代码】

```
cities = ['Beijing', 'Shanghai']            # 创建列表 cities
cities.append('Guangzhou')                  # 追加一个字符串类型的元素
print(cities)
cities.append(['Suzhou', 'Hangzhou'])       # 追加列表,整个列表被当成一个元素
print(cities)
```

程序执行结果：

```
['Beijing', 'Shanghai', 'Guangzhou']
['Beijing', 'Shanghai', 'Guangzhou', ['Suzhou', 'Hangzhou']]
```

可以看到,当用 append()方法传递列表时,此方法会将其视为一个整体,作为单个元素添加到列表中,从而形成嵌套列表。

2) extend()方法

extend()方法将另一个序列 seq 中的所有元素依次添加到列表 listname 的末尾。extend()方法和 append()方法一样都能够在列表末尾添加元素。

【语法格式】

```
listname.extend(seq)
```

【说明】

listname 为列表名,seq 可以是列表、字符串、元组等序列类型,但不能是单个数字。extend()方法不会把参数视为一个整体,将其作为单个元素进行添加,而是把参数包含的所有元素逐个添加到列表中。

【例 3-5】 利用 extend()方法添加列表元素。

【参考代码】

```
cities = ['Wuhan', 'Fuzhou']
cities.extend('Zhengzhou')          # 添加元素 Zhengzhou
print(cities)
others = ['Hefei', 'Jinan']
cities.extend(others)               # 添加列表,列表被拆分成多个元素
print(cities)
cities.extend("Xian")               # 添加字符串,字符串的每个字符均形成一个新元素
print(cities)
```

程序执行结果：

```
['Wuhan', 'Fuzhou', 'Z', 'h', 'e', 'n', 'g', 'z', 'h', 'o', 'u']
['Wuhan', 'Fuzhou', 'Z', 'h', 'e', 'n', 'g', 'z', 'h', 'o', 'u', 'Hefei', 'Jinan']
['Wuhan', 'Fuzhou', 'Z', 'h', 'e', 'n', 'g', 'z', 'h', 'o', 'u', 'Hefei', 'Jinan', 'X', 'i', 'a', 'n']
```

3) insert()方法

append()方法和 extend()方法只能在列表末尾插入元素,有时候希望在列表中间某个位置插入元素,那么就可以使用 insert()方法。

【语法格式】

```
listname.insert(index,obj)
```

【说明】

listname 为列表名，参数 index 是列表的索引值，执行时在此处插入元素 obj，不覆盖原本的数据项，原数据项向后顺延。参数 obj 可以是元素也可以为列表。

当插入列表等容器对象时，insert()方法也是将其视为一个整体，作为单个元素插入列表中，这一点和 append()方法的应用是一样的。

【例 3-6】 利用 insert()方法添加列表元素。

【参考代码】

```
cities = ['Wuhan', 'Fuzhou']
cities.insert(1, 'Zhengzhou')                # 在索引值为 1 的位置插入元素
print(cities)
cities.insert(3, ['Hefei', 'Jinan'])         # 插入列表,整个列表被当成一个元素
print(cities)
# 在索引值为 0 的位置插入字符串,整个字符串被当成一个元素
cities.insert(0, 'Xian')
print(cities)
```

程序输出结果：

```
['Wuhan', 'Zhengzhou', 'Fuzhou']
['Wuhan', 'Zhengzhou', 'Fuzhou', ['Hefei', 'Jinan']]
['Xian', 'Wuhan', 'Zhengzhou', 'Fuzhou', ['Hefei', 'Jinan']]
```

insert()方法主要用来在列表的中间位置插入元素，如果仅仅需要在列表的末尾添加元素，建议使用 append()方法或 extend()方法，以免索引值的设置出错。

4）pop()方法

在前面已经学习过可以通过 del 语句来删除列表中的某个元素。除此之外，Python 语言还定义了多种内置函数来删除元素。

在列表中删除元素主要分为如下 3 种场景：

(1) 根据目标元素所在位置的索引进行删除，可以使用 del 语句或者 pop()方法；

(2) 根据元素本身的值进行删除，可使用 remove()方法；

(3) 将列表中所有元素全部删除，可使用 clear()方法。

【语法格式】

```
listname.pop([index = -1])
```

【说明】

参数 index 为列表索引值。与 del 语句的作用相似，pop()方法用来删除列表中指定位置 index 的元素，默认删除列表的最后一个元素，并返回删除的数据。这类似于数据结构中的“出栈”操作。注意，如果指定的参数 index 越界，则会出现 IndexError 错误。

【例 3-7】 利用 pop()方法删除指定列表元素。

【参考代码】

```
cities = ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei', 'Jinan', 'Xian']
print("原始列表:",cities)
cities.pop(3) # 删除索引值为 3 的第四个元素
print("删除第四个元素:",cities)
cities.pop() # 默认删除最后一个元素
print("删除最后一个元素:",cities)
```

程序执行结果：

```
原始列表: ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei', 'Jinan', 'Xian']
删除第四个元素: ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Jinan', 'Xian']
删除最后一个元素: ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Jinan']
```

不少编程语言提供了和 pop()方法相应的 push()方法。该方法也可用来将元素添加到列表的尾部，这类似于数据结构中的“入栈”操作。但是 Python 语言是个例外，Python 语言并没有提供 push()方法，因为完全可以使用 append()来代替 push()的功能。

5) remove()方法

如果要根据元素本身的值来删除元素，那么可以使用 remove()方法。

【语法格式】

```
listname.remove(obj)
```

【说明】

参数 obj 为列表元素值。完成删除列表中的某个元素第一个匹配结果。注意，remove()方法只会删除第一个和指定值 obj 相同的元素。同时也要注意，必须保证该元素是存在的，否则会引发 ValueError 错误。

【例 3-8】 利用 remove()方法删除指定元素。

【参考代码】

```
cities = ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei', 'Jinan', 'Xian']
print("原始列表:", cities)
cities.remove('Zhengzhou') # 删除元素值为'Zhengzhou'的元素
print("删除'Zhengzhou'元素:", cities)
# 删除元素值为'Zhengzhou'的元素,因元素已删除,列表中不存在该元素,所以出错
cities.remove('Zhengzhou')
print("删除'Zhengzhou'元素:",cities)
```

程序执行结果：

```
原始列表: ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei', 'Jinan', 'Xian']
删除'Zhengzhou'元素: ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Xian']
Traceback (most recent call last):
  File "D:\Python\3 - 8.py", line 11, in < module >
    cities.remove('Zhengzhou')
ValueError: list.remove(x): x not in list
```

最后一次删除时，因为元素值为'Zhengzhou'的元素已删除，即该元素不存在，所以导致程序执行报错。因此在使用 remove()删除指定元素时，需要确保元素存在。

6) index()方法

Python 语言中列表提供了 index()方法和 count()方法，它们都可以用来查找元素。其中，index()方法用来查找某个元素在列表中出现的位置，返回元素的索引值。

【语法格式】

```
listname.index(obj[, start][, end])
```

【说明】

参数中 obj 为查找对象即指定的列表元素。第二个参数 start 和第三个参数 end 指定查找范围，表示查找 start 和 end 位置之间的元素，返回该元素的索引值。如果只设置 start 而未设置 end，那么表示查找从 start 到列表末尾的元素；如果 start 和 end 均未设置，那么默认查找整个列表。注意，如果该元素不存在，则会出现 ValueError 异常。

【例 3-9】 利用 index()方法查找列表元素。

【参考代码】

```
cities = ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei', 'Jinan', 'Xian']
print("原始列表:",cities)
print(cities.index( 'Zhengzhou') )          # 在列表中的所有元素中查找元素 'Zhengzhou'
print(cities.index( 'Zhengzhou', 1, 3) )    # 在索引值为 1～3 的元素中查找元素 'Zhengzhou'
print(cities.index( 'Zhengzhou', 1) )       # 在索引值 1 之后的元素中查找元素 'Zhengzhou'
print(cities.index('Nanjing') )             # 查找一个不存在的元素 'Nanjing'
```

程序执行结果：

```
原始列表: ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei', 'Jinan', 'Xian']
2
2
2
Traceback (most recent call last):
  File "D:\Python\3 - 9.py", line 10, in < module >
    print(cities.index('Nanjing') )    # 查找一个不存在的元素 'Nanjing'
ValueError: 'Nanjing' is not in list
```

7) count()方法

若不确定元素是否存在，则利用 index()方法来查找指定元素会有出现异常的可能，因此在查找元素之前也可以使用 count()方法判断某个元素在列表中出现的次数。

【语法格式】

```
listname.count(obj)
```

【说明】

参数 obj 为统计对象即指定的列表元素。该方法完成查看元素 obj 在列表中出现的次数。如果 count()方法返回 0，则表示列表中不存在该元素。

【例 3-10】 利用 count()方法统计指定的列表元素。

【参考代码】

```
cities = ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei', 'Jinan', 'Zhengzhou', 'Xian']
print("'Zhengzhou'出现了 %d 次" %cities.count('Zhengzhou'))    # 统计元素出现的次数
# 判断一个元素是否存在
if cities.count('Nanjing'):
    print("列表中存在'Nanjing'")
else:
    print("列表中不存在'Nanjing'")
```

程序执行结果：

```
'Zhengzhou'出现了 2 次
列表中不存在'Nanjing'
```

8) reverse()方法

reverse()方法用于将列表中的所有元素反向排序。

【语法格式】

```
listname.reverse()
```

【说明】

该方法将对原列表的元素进行反向排序，操作在原列表上进行，不返回新的列表，即没有返回值。

【演示】 利用 reverse 方法对原列表进行反向排序。

```
>>> cities = ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei']
>>> cities.reverse()              # 列表反向
>>> print(cities)
    ['Hefei', 'Zhengzhou', 'Fuzhou', 'Wuhan']
```

9) sort()方法

sort()方法用于对列表进行排序。

【语法格式】

```
listname.sort([key = None][,reverse = False])
```

【说明】

使用该方法可以对列表 listname 进行排序，其中各个参数的含义与内置函数 sorted()相同。利用关键字 key 指定一个比较函数参数，实现自定义排序，默认为 None；关键字 reverse 用于指定排序规则参数，设置为 True 则按降序排序，默认为 False，表示按升序排序。使用该方法将会修改原列表。该方法没有返回值，若需要返回一个新的列表，需使用内置函数 sorted()。

【例 3-11】 利用 sort()方法对列表元素进行排序。

【参考代码】

```
cities = ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Xian']
print("初始列表:", cities)
```

```
# 简单使用列表排序,默认为升序
cities.sort()
print("升序排列:", cities)
# 降序排序
cities.sort(reverse = True)
print("降序排列:", cities)
# 通过设置 key 参数来指定比较规则
def takeSecond(elem):                    # 定义比较函数
    return elem[1]                       # 返回列表索引值为 1 的元素
cities = [['Nanjing',4],['Beijing',1],['Guangzhou',3],['Shanghai',2]]
print("\n 待排序列表:\n", cities)
cities.sort(key = takeSecond) # 指定按照第二个元素的大小进行排序
print ('根据 ID 大小排序列表:\n', cities)
```

程序执行结果：

```
初始列表: ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Xian']
升序排列: ['Fuzhou', 'Hefei', 'Jinan', 'Wuhan', 'Xian']
降序排列: ['Xian', 'Wuhan', 'Jinan', 'Hefei', 'Fuzhou']
待排序列表:
[['Nanjing', 4], ['Beijing', 1], ['Guangzhou', 3], ['Shanghai', 2]]
根据 ID 大小排序列表:
[['Beijing', 1], ['Shanghai', 2], ['Guangzhou', 3], ['Nanjing', 4]]
```

10）clear()方法

clear()方法用来删除列表中的所有元素。

【语法格式】

```
listname.clear()
```

【说明】

与 del listname[:]相当，即清空列表。

【演示】 清空列表。

```
>>> cities = ['Wuhan', 'Fuzhou', 'Zhengzhou', 'Hefei']
>>> cities.clear()    # 清空列表
>>> print(cities)
    []
```

11）copy()方法

copy()方法用于复制列表，类似于 list[:]操作。

【语法格式】

```
listname.copy()
```

【说明】

该方法用于复制列表 listname，返回列表的一个副本，即复制后的新列表。

【例 3-12】 利用 copy()方法对列表进行复制。

【参考代码】

```
cities1 = ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Xian']
print("初始列表 cities1:", cities1)
cities2 = cities1.copy()       # 复制列表 cities1
print("复制列表 cities2:",cities2)
print("更新列表 cities2 中最后一个元素: ")
cities2[-1] = 'Nanjing'        # 更新列表 cities 中索引值为 -1 的元素值
print("cities1: ",cities1)
print("cities2: ",cities2)
```

程序执行结果：

```
初始列表 cities1: ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Xian']
复制列表 cities2: ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Xian']
更新列表 cities2 中最后一个元素:
cities1: ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Xian']
cities2: ['Wuhan', 'Fuzhou', 'Hefei', 'Jinan', 'Nanjing']
```

可见，新列表后续的赋值更新将不会影响原列表。

3.1.3 元组

元组(tuple)和列表类似，也是 Python 语言中一个重要的序列类型。元组由一系列按特定顺序排序的元素组成。不过，元组和列表的不同之处在于：列表的元素是可以更改的，包括插入、删除和更新元素，所以列表是可变序列；而元组一旦被创建，它的元素就不可更改了，所以元组是不可变序列。因此，通常用元组来保存无须修改的数据序列。

1. 元组的创建

【语法格式】

```
(element1, element2, …, elementn)
```

【说明】

element1～elementn 表示元组中的各个元素。从形式上看，元组的所有元素都放在一对小括号“()”中，相邻元素之间用逗号“,”分隔。从存储内容上看，和列表一样，元组可以存储整型、实型、字符串型、列表、元组等任何类型的数据，并且一个元组中各个元素的类型可以不同。

【演示】 创建元组。

```
>>> ('Michael', 'Bob', 2022, 2000, ('a', 'b'))
```

可以看出，这个元组中的元素有字符串型和整型等，类型并不统一。

创建列表的方式有使用“[]”直接创建和使用内置函数 list()创建两种方式。类似地，创建元组的方式也有使用“()”直接创建和使用内置函数 tuple()创建两种。不过 tuple()内置函数只接收可以转换为元组的数据，包括字符串、列表等。

【例 3-13】 创建元组的合法方式。

【参考代码】

```
tuple1 = ('SQL', 'Ruby', 'Java', 'Go', 'C++')            # 使用"()"直接创建
tuple2 = tuple(['SQL', 'Ruby', 'Java', 'Go', 'C++'])     # 将列表转换为元组
tuple3 = tuple("hello")                                  # 将字符串转换为元组
tuple4 = tuple(range(1, 6))                              # 将区间转换为元组
print("tuple1: ",tuple1)
print("tuple2: ",tuple2)
print("tuple3: ",tuple3)
print("tuple4: ",tuple4)
```

程序执行结果：

```
tuple1: ('SQL', 'Ruby', 'Java', 'Go', 'C++')
tuple2: ('SQL', 'Ruby', 'Java', 'Go', 'C++')
tuple3: ('h', 'e', 'l', 'l', 'o')
tuple4: (1, 2, 3, 4, 5)
```

注意，当创建的元组中只有一个元素时，该元素后面必须添加一个逗号，否则 Python 语言解释器会将括号当作算术运算符使用。

【例 3-14】 创建元组中的括号。

【参考代码】

```
a = ("hello", )          # 最后加上逗号
print(type(a))
print(a)
b = ("hello")            # 最后不加逗号
print(type(b))
print(b)
```

程序执行结果：

```
<class 'tuple'>
('hello',)
<class 'str'>
hello
```

有时候在将数据打包成元组时还可以省略小括号。

【演示】 元组打包。

```
>>> something = 'Bob', 2022, ('a', 'b')
>>> type(something)
    <class 'tuple'>
```

上例中值'Bob'、2022 和('a', 'b')一起被打包进元组。对此的逆操作也可以。

【演示】 若有上述打包过程，则可进行序列解包。

```
>>> s, i, t = something
>>> s
    'Bob'
>>> i
```

```
    2022
>>> t
    ('a', 'b')
```

序列解包适用于左侧变量与右侧序列元素数量相等的情况。所以,多重赋值其实只是元组打包和序列解包的组合。

2. 基本操作

序列的所有基本操作和内置函数对于元组几乎都是成立的。例如,和列表一样,可以使用索引访问元组中的某个元素,也可以使用切片访问元组中的若干元素。

【例 3-15】 元组的基本操作。

【参考代码】

```
hi = tuple("Hello Python")
# 使用索引访问元组中的某个元素
print(hi[3])                          # 使用正数索引
print(hi[-4])                         # 使用负数索引
# 使用切片访问元组中的一组元素
print(hi[2:5])                        # 使用正数切片
print(hi[-6: -1])                     # 使用负数切片
print(hi[1: 8: 3])                    # 指定步长
```

程序执行结果:

```
l
t
('l', 'l', 'o')
('P', 'y', 't', 'h', 'o')
('e', 'o', 'y')
```

但是,元组是不可变序列,元组中的元素不能被修改。所以元组的元素只能出现在赋值号的右边,而不能出现在赋值号的左边。在某些操作中,只能创建一个新的元组去替代旧的元组,而非原地更新元组中元素的值。

【例 3-16】 元组重新赋值。

【参考代码】

```
languages = ('Go', 'C++')
languages = ('Python', 'Ruby', 'C')     # 对元组进行重新赋值
print(languages)
new = ('Go', 'C++')
print(languages + new)
print(new * 3)
print(languages)
print(new)
```

程序执行结果:

```
('Python', 'Ruby', 'C')
('Python', 'Ruby', 'C', 'Go', 'C++')
('Go', 'C++', 'Go', 'C++', 'Go', 'C++')
```

```
('Python', 'Ruby', 'C')
('Go', 'C++')
```


可以看出，使用元组的相加和相乘操作以后，参与运算的元组内容没有发生改变，这说明运算结果返回的是一个新元组。

3. 相关方法

由于元组不可修改，因此列表方法中修改原对象的方法均不能应用于元组，例如append()方法、insert()方法、remove()方法等。del()方法也不能用来删除元组中的某个元素，只能删除整个元组。但有些列表方法对于元组也是适用的，例如 index()方法和 count()方法。

【演示】 元组的常用方法。

```
>>> tup = tuple("abandon")
>>> tup
    ('a', 'b', 'a', 'n', 'd', 'o', 'n')
>>> tup.index('n')
    3
>>> tup.count('a')
    2
```

3.2 映射类——字典

列表和元组都是有序的序列，它们的元素在内存中是连续存放的。相对应地，Python 语言还有一种组合类型字典(dict)，它是一种无序的、可变的数据集合，每个元素以“键值对”(key-value)的形式在内存中进行存储。

字典类型也是 Python 语言中唯一的映射类型。“映射”是数学中的术语，简单理解，就是元素之间存在相互对应的关系，即通过一个元素可以唯一地找到另一个元素，正如学生的学号可以唯一地对应一个学生名字。字典中，习惯将各元素对应的索引称为键(key)，各个键对应的元素称为值(value)，键及其关联的值即为“键值对”。字典类型很像一本字典，通过字典中的索引表可以快速找到想要查找的汉字，其中索引表就相当于字典类型中的键，而索引表对应的汉字则相当于键对应的值。

字典和列表、元组等序列相比，有其独特的特点：

(1) 字典通过键而不是通过索引来读取元素；

(2) 列表和元组按照索引值大小存储各个元素，而字典中的元素是无序的。

3.2.1 字典的创建

在形式上，字典的每个键值对用冒号分隔，键值对之间用逗号分隔，整个字典包含在大括号中。

【语法格式】

```
{key1: value1, key2: value2, …, keyN: valueN}
```

【说明】

key1:value1～keyN:valueN 表示各个元素的键值对。需要注意的是,同一字典中的各个键必须唯一,不能重复。同时,字典的键必须是不可变的,所以只能使用数字、字符串或者元组等不可变类型,不能使用列表。字典的值可以是 Python 语言支持的任意数据类型。并且,字典和列表、元组一样可以任意嵌套。

创建字典也有多种方式,下面将一一加以介绍。

1. 使用“{}”创建字典

最常见的创建字典的方法是直接使用大括号“{}”,这也是最基本的方法。在创建字典时,键和值之间使用冒号“:”分隔,相邻元素之间使用逗号“,”分隔,所有元素放在大括号“{}”中。因为字典是可变对象,所以开始创建时并不用将所有数据都填充进去,可以先创建一个空字典,待有数据时再更新字典。

【例 3-17】 创建字典。

【参考代码】

```
emptyDict = {}                                    # 使用大括号来创建空字典
scores = {'数学': 95, '英语': 92, '语文': 84}     # 创建包含 3 组键值对的字典
print(scores)                                     # 输出字典
print("Length:", len(scores))                     # 用 len()内置函数查看字典元素的数量
print(type(scores))                               # 用 type()内置函数查看变量
```

程序执行结果:

```
{'数学': 95, '英语': 92, '语文': 84}
Length: 3
<class 'dict'>
```

2. 使用 dict()函数创建字典

如果已有键值数据可以使用内置函数 dict()直接通过列表创建字典。通过 dict()函数创建字典有多种写法。

【演示】 使用 dict()函数创建字典。

```
>>> scores = dict(数学 = 95,英语 = 92,语文 = 84)          # 注意,字符串作为键时不能写引号
>>> scores = dict([('数学',95),('英语',92),('语文',84)]) # 列表和元组混合使用
>>> scores = dict((['数学',95],['英语',92],['语文',84])) # 列表和元组混合使用
>>> scores = dict([['数学',95],['英语',92],['语文',84]]) # 使用嵌套列表
>>> scores = dict((('数学',95),('英语',92),('语文',84))) # 使用嵌套元组
>>> scores
    {'数学': 95, '英语': 92, '语文': 84}
```

上述语句均等价地创建了一个字典 scores:{'数学':95,'英语':92,'语文':84}。

如果 dict()函数没有任何参数,则代表创建一个空字典。

```
>>> scores = dict()      # 不带参数表示生成一个空字典
>>> scores
    { }
```

3. 使用 fromkeys()方法创建字典

Python 语言为字典类型提供了 fromkeys()内置方法创建带有默认值的字典。

【语法格式】

```
dict.fromkeys(seq[, value = None])
```

【说明】

seq 参数表示字典中所有键的序列，value 参数表示创建字典中各元素的初始值，默认为空值 None。该方法返回一个新字典。

【例 3-18】 采用 fromkeys()方法创建字典。

【参考代码】

```
classes = ['语文', '数学', '英语']
scores = dict.fromkeys(classes)            # 省略,默认值为 None
print(scores)
scores = dict.fromkeys(classes, 60)        # 指定默认值为 60
print(scores)
```

程序执行结果：

```
{'语文': None, '数学': None, '英语': None}
{'语文': 60, '数学': 60, '英语': 60}
```

可以看到，classes 列表中的元素全部作为 scores 字典的键，而各个键对应的值都是指定的默认值。这种创建方式通常用于初始化字典，设置各个键对应的初始值。

3.2.2 基本操作

字典的基本操作与列表和元组的操作略有不同之处。

1. 访问字典

1）通过键访问字典

列表和元组是通过下标来访问元素的，而字典则通过键来访问键的对应值。因为字典中的元素是无序的，每个元素的位置都不固定，所以字典也不能像列表和元组那样采用切片的方式一次性访问多个元素。

【语法格式】

```
dictname[obj]
```

【说明】

dictname 为字典名，参数 obj 为键。注意，键必须是存在的，否则访问会发生异常。

【演示】 通过键访问字典。

```
>>> scores = dict(数学 = 95,英语 = 92,语文 = 84)      # 注意,字符串作为键时不能写引号
>>> print(scores['数学'])                              # 键存在
    95
>>> print(scores['物理'])                              # 键不存在
```

```
Traceback (most recent call last):
 File "<pyshell# 164>", line 1, in <module>
    print(scores['物理'])
KeyError: '物理'
```

如果字典中键的值本身也是字典,则可以使用多个键来访问字典元素。

```
>>> person = {'birthday':{"year":2000,"month":3,"day":23}}
>>> person["birthday"]["day"]
    23
```

2) 通过 get()方法访问字典

除了上面这种方式外,Python 语言更推荐使用字典类型提供的 get()方法来获取指定键的对应值。与上面方法不同的是,当指定的键不存在时,get()方法不会发生异常。

【语法格式】

```
dictname.get(key[, default = None])
```

【说明】

dictname 表示字典变量名,key 表示指定的键,default 用于指定要查询的键不存在时的返回信息,默认为 None。

【演示】 通过 get()方法访问字典。

```
>>> scores = {'数学':95,'英语':92,'语文':84}
>>> print(scores.get('数学'))                          # 键存在
    95
>>> print(scores.get('物理'))                          # 键不存在,返回默认值 None
    None
>>> print(scores.get('物理','该键不存在'))             # 键不存在,返回提示信息
    该键不存在
```

和 get()方法相类似的还有一个 setdefault()方法,两者唯一的区别是:如果键不存在于字典中,setdefault()方法将会添加键并将值设为 default 参数指定的值(默认为 None)。

【例 3-19】 setdefault()方法的使用。

【参考代码】

```
scores = {'数学': 95,'英语': 92,'语文': 84}
print("初始字典:",scores)
scores.setdefault('数学')        # 键存在
scores.setdefault('物理')        # 键不存在,设置'物理'值为 None
scores.setdefault('化学',60)     # 键不存在,设置值为 60
print("新字典:",scores)
```

程序执行结果:

```
初始字典: {'数学': 95, '英语': 92, '语文': 84}
新字典: {'数学': 95, '英语': 92, '语文': 84, '物理': None, '化学': 60}
```

2. 更新字典

添加和更新字典元素都可以通过赋值语句来实现，即直接通过键来设置值。

【语法格式】

```
dictname[obj] = value
```

【说明】

dictname 表示字典变量名，如果字典未包含指定的键 obj，则在字典中增加一个新元素，其键为 obj 值为 value；如果指定的键 obj 已经存在于字典中，则将键 obj 对应的值更新为新值 value。

【演示】 通过键添加和更新字典元素。

```
>>> scores = dict(数学 = 95,英语 = 92,语文 = 84)
>>> scores['物理'] = 88      # 添加新键"物理",其值为 88
>>> scores
    {'数学': 95, '英语': 92, '语文': 84, '物理': 88}
```

不过 Python 语言不允许同一个键出现两次。所以如果同一个键被赋值两次，后值将覆盖前值。

【演示】 多次赋值同一个键。

```
>>> scores = dict(数学 = 95,英语 = 92,语文 = 84)
>>> scores['英语'] = 88    # 相当于重新赋值
>>> scores
    {'数学': 95, '英语': 88, '语文': 84}
```

3. 删除字典

和列表一样，删除字典使用 del 关键字，既可以删除整个字典，也可以仅删除字典中的某一键值对。

【演示】 字典的删除。

```
>>> scores = dict(数学 = 95,英语 = 92,语文 = 84)
>>> del scores['数学']
>>> scores
    {'英语': 92, '语文': 84}
>>> del scores
>>> scores
    Traceback (most recent call last):
     File "<pyshell# 191>", line 1, in <module>
        scores
    NameError: name 'scores' is not defined
```

4. 键的检测

因为字典中的每个元素由键及其对应值组成，所以对字典元素操作之前，可以使用运算符 in 检测该键是否存在于字典中，如果存在则返回 True，否则返回 False。而运算符 not in 刚好相反，如果键在字典中则返回 False，否则返回 True。

【演示】 检测字典中的键。

```
>>> scores = dict(数学 = 95,英语 = 92,语文 = 84)
>>> '英语' in scores
    True
>>> '物理' not in scores
    True
```

5. 内置函数

字典的内置函数主要针对字典的键进行操作。例如，可以对字典执行内置函数 list()，返回该字典中所有键的列表，也可以使用 len()、max()、min()、sum()、sorted()等内置函数对字典中的键计算规模、最大值、最小值、总和以及对键进行排序。

【演示】 字典内置函数的使用。

```
>>> students = {2: '张悦', 3: '李明', 1: '王琳'}
>>> list(students)
    [2, 3, 1]
>>> len(students)
    3
>>> max(students)
    3
>>> min(students)
    1
>>> sum(students)
    6
>>> sorted(students)
    [1, 2, 3]
```

3.2.3 相关方法

作为 Python 语言最重要的类型之一，字典拥有诸多内置方法。若 dict 为字典名，则常用方法如表 3-3 所示。

表 3-3 字典的常用方法

方 法	描 述
dict. get(key,default=None)	返回字典 dict 指定键 key 的值。如果键不在字典中则返回 default 参数值，默认为 None
dict. setdefault(key,default=None)	和 get()类似，但如果键 key 不在字典中，将会添加键并将对应值设为 default
dict. items()	返回字典 dict 的所有键值对组成的对象
dict. keys()	返回字典 dict 的所有键组成的对象
dict. values()	返回字典 dict 的所有值组成的对象
dict1. update(dict2)	把字典 dict2 的键值对更新到字典 dict1 里
dict. pop(key[,default])	删除并返回字典 dict 中键 key 所对应的值，如果不存在键则返回 default 值
dict. popitem()	随机删除并返回字典 dict 中的一组键值对
dict. clear()	清空字典
dict. copy()	复制字典

前面已经介绍过 get()方法和 setdefault()方法，而 clear()方法和 copy()方法的使用与列表一致。接下来介绍其余内置方法的使用。

1. items()、keys()和 values()方法

这三种方法用来获取字典中的特定数据，分别返回键值对、键和值组成的对象。dict.items()方法用于获取包含字典 dict 中所有(键，值)元组的列表；dict.keys()方法用于获取包含字典 dict 中所有键的列表；dict.values()方法用于获取包含字典 dict 中所有值的列表。

【例 3-20】 获取字典中的特定数据。

【参考代码】

```
scores = {'数学': 95, '语文': 89, '英语': 90}
print(scores.items())              # 获取字典中的键值对
print(scores.keys())               # 获取字典中的键
print(scores.values())             # 获取字典中的值
```

程序执行结果：

```
dict.items([('数学', 95), ('语文', 89), ('英语', 90)])
dict.keys(['数学', '语文', '英语'])
dict.values([95, 89, 90])
```

可以发现，方法 items()、keys()和 values()返回值的类型分别为 dict_items、dict_keys 和 dict_values。这些类型并不是常见的列表或者元组类型，因为 Python 语言不希望用户直接操作这几种方法的返回值。

如果想使用这三种内置方法返回的对象，一般有如下两种方案：

(1) 使用 list()函数将返回的数据转换为列表；

(2) 使用 for in 循环遍历它们的返回值。

【例 3-21】 获取字典的键和值。

【参考代码】

```
scores = {'数学': 95, '语文': 89, '英语': 90}      # 创建字典
classes = list(scores.keys())                      # 类型转换为列表
print(classes)
for v in scores.values():                          # 遍历值
    print(v, end = ' ')
print("\n" + '-'*25)
for k,v in scores.items():                         # 遍历键值对
    print("key:", k, " value:", v)
```

程序执行结果：

```
['数学', '语文', '英语']
95 89 90
-------------------------
key: 数学 value: 95
key: 语文 value: 89
key: 英语 value: 90
```

2. update()方法

update()方法可以使用另一个字典所包含的键值对来更新现有的字典。在执行 update()方法时,如果被更新的字典中已包含对应的键值对,那么原 value 值会被覆盖;如果被更新的字典中不包含对应的键值对,则在原字典中添加该键值对。

【例 3-22】 更新现有的字典。

【参考代码】

```
scores = {'数学': 95, '语文': 89, '英语': 90}
scores.update({'语文': 83, '物理': 93, '化学': 92})
print(scores)
```

程序执行结果:

```
{'数学': 95, '语文': 83, '英语': 90, '物理': 93, '化学': 92}
```

从执行结果可以看出,由于被更新的字典中已包含键为"语文"的键值对,因此更新时该键值对的值将被改写;而被更新的字典中不包含键为"物理"和"化学"的键值对,所以更新时会在原字典中增加这两个键值对。

3. pop()和 popitem()方法

pop()和 popitem()方法都用来删除指定字典中的键值对,不同的是,pop()方法用来删除指定的键值对,而 popitem()方法用来随机删除一个键值对。

【例 3-23】 删除字典中的键值对。

【参考代码】

```
scores = {'数学': 95, '语文': 89, '英语': 90, '化学': 83, '生物': 98, '物理': 89}
print(scores)
print(scores.pop('化学'))       # 删除并返回键对应的值 83
print(scores)
print(scores.popitem())         # 删除并返回一个随机的键值对
print(scores)
```

程序执行结果:

```
{'数学': 95, '语文': 89, '英语': 90, '化学': 83, '生物': 98, '物理': 89}
83
{'数学': 95, '语文': 89, '英语': 90, '生物': 98, '物理': 89}
('物理', 89)
{'数学': 95, '语文': 89, '英语': 90, '生物': 98}
```

其实,popitem()方法并非随机删除一个键值对,虽然字典是无序的,但键值对在内存中的存储也是有顺序的,popitem()方法总是弹出内存中的最后一个键值对,这和列表的 pop()方法类似,都实现了数据结构中"出栈"的操作。

3.3 集合类——集合

Python 语言中的集合(set)和数学中的集合概念一样,用来保存多个不重复的元素,即集合中的元素都是唯一的,互不相同。集合相当于一个无序的无重复元素序列。

集合分为可变集合和不可变集合。与列表和元组等有序序列不同，集合并不记录元素的位置，因此对集合不能进行索引和切片等操作。不过，用于序列的一些操作和函数也能够用于集合。此外集合还可以进行交集、并集、差集等集合运算。

3.3.1 集合的创建

使用大括号创建一个集合，只要将用逗号分隔的不同数据项括起来即可。

【语法格式】

```
{element1, element2, …, elementN}
```

【说明】

element1～elementN 表示集合中的元素，元素的个数没有限制。

从形式上看，集合和字典类似，它将所有元素放在一对大括号“{}”中，相邻元素间用“,”分隔。

从内容上看，同一集合中可以存储不可变的数据类型，包括整型、浮点型、字符串型、元组等，但无法存储列表、字典、集合等这些可变的数据类型，否则 Python 语言解释器会报错。例如，{[1,2,3]}、{{'a':1}}、{{1,2,3}}会抛出 TypeError 异常。集合中的数据必须保证唯一。

Python 语言同样提供了两种创建集合的方法，分别是使用“{}”直接创建和使用 set()内置函数将列表、元组等类型数据转换为集合。

【例 3-24】 使用“{}”创建集合。

【参考代码】

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)
set1 = set("apple")                    # 将字符串类型转换为集合
set2 = set([1, 2, 3, 2, 1])            # 将列表类型转换为集合
set3 = set((1, 2, 3, 2, 1))            # 将元组类型转换为集合
print("set1:",set1)
print("set2:",set2)
print("set3:",set3)
```

程序执行结果：

```
{'pear', 'orange', 'banana', 'apple'}
set1: {'l', 'a', 'p', 'e'}
set2: {1, 2, 3}
set3: {1, 2, 3}
```

可见创建出的集合内元素是无序且非重复的。注意，创建一个空集合只能用 set()而不是“{}”，因为“{}”是用来创建一个空字典而非空集合。

【演示】 集合输出。

```
>>> {1,2,1,(1,2,3),'c','c'}
    {1, 2, 'c', (1, 2, 3)}
```

由于集合是无序的，因此每次输出时元素的排序顺序可能都不相同。

3.3.2 基本操作

在 Python 语言中，不仅支持集合元素的访问、更新和删除，还支持一些运算。

1. 访问集合

由于集合中的元素是无序的，因此无法像列表那样使用索引访问元素。Python 语言中，访问集合元素最常用的方法是使用循环结构，将集合中的数据逐一读取出来。

【例 3-25】 访问集合。

【参考代码】

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
for fruit in basket:
    print(fruit, end = ' ')
```

程序执行结果：

```
apple orange banana pear
```

由于目前尚未学习循环结构，以上代码只需关注结果。

2. 更新集合

由于集合中的元素无法通过索引值或者键来访问，因此无法指定某个元素进行重新赋值。更新集合的方式只能是添加新元素和删除特定值的元素。当需要向一个集合中添加元素时，可以使用 Python 语言为集合类型内置的 add()方法和 update()方法。add()方法接收一个参数，并将该参数值作为新元素添加到集合中。而 update()方法接收的参数可以有多个，用逗号隔开。各参数可以是集合、列表、元组或字典等。执行方法可将其中包含的所有元素分别添加到集合中(字典类型参数则将字典的键加入集合中)。以上过程中，如果要添加的元素已存在则不进行任何操作。

【例 3-26】 更新集合。

【参考代码】

```
basket = {'apple', 'orange'}
basket.add('banana')                # 添加新元素'banana'
print(basket)
basket.update({'orange', 'pear'})   # 添加新元素'orange'和'pear',其中'orange'已存在
print(basket)
# 添加三个新元素
basket.update(['lemon', 'coconut'], {'cherry': 'meaningless'})
print(basket)
```

程序执行结果：

```
{'banana', 'apple', 'orange'}
{'apple', 'orange', 'pear', 'banana'}
{'coconut', 'apple', 'orange', 'lemon', 'cherry', 'pear', 'banana'}
```

3. 删除集合

当需要删除集合中的一个元素时，可以通过内置方法 remove()和 discard()来实现，它

们均接收一个参数来指定想要删除的元素值，两者的区别在于：如果被删除的元素不存在，则 remove()方法执行时则会发生错误，而 discard()方法则不会。

【演示】 删除集合中的元素。

```
>>> basket = set(('orange', 'pear', 'apple', 'banana'))     #创建集合
>>> basket.remove("apple")                                  # 删除"apple"元素
>>> basket
    {'pear', 'orange', 'banana'}
>>> basket.remove("lemon")                                  # 被删除元素不存在,产生错误
    KeyError: 'lemon'
>>> basket.discard("lemon")    # 对于 discard()方法,被删除元素不存在,执行不产生错误
>>> basket
    {'pear', 'orange', 'banana'}
```

当然，也可以使用 pop()内置方法随机删除集合中的一个元素。

【演示】 使用 pop()内置方法随机删除集合中元素。

```
>>> basket.pop()
    'pear'
>>> basket
    {'orange', 'banana'}
```

4. 集合运算

与列表、元组、字典一样，集合也可以使用 in 或 not in 运算符来检查是否包含某个元素，也可以使用 len()、max()、min()、sum()和 sorted()等内置函数。除此之外，集合最常用的操作是集合之间完成交集、并集、差集以及对称差集等集合运算。集合运算操作如表 3-4 所示。

表 3-4 集合运算操作

运算操作	运算符	含义
交集	&	取两集合公共的元素
并集	|	取两集合全部的元素
差集	-	取一个集合中有另一集合没有的元素
对称差集	^	取两个集合全部元素中不属于公共元素的元素

【演示】 集合运算。

```
>>> a = {1, 2, 3}          # 创建两个集合
>>> b = {3, 4, 5}
>>> a & b                  # 交集,即集合 a 和 b 中都包含的元素
    {3}
>>> a | b                  # 并集,即集合 a 或 b 中包含的所有元素
    {1, 2, 3, 4, 5}
>>> a - b                  # 差集,即集合 a 中包含而集合 b 中不包含的元素
    {1, 2}
>>> a ^ b                  # 对称差集,并集元素减交集元素,即不同时包含于 a 和 b 的元素
    {1, 2, 4, 5}
```

3.3.3 相关方法

Python 语言为集合类型提供的常用内置方法如表 3-5 所示，其中 set 为集合。

表 3-5 集合的常用方法

方　法	描　述
set.add(e)	为集合 set 添加元素 e
set.update(s)	给集合 set 添加参数 s 中的各元素，s 可以是一个或多个对象
set.remove(e)	从集合 set 中移除指定元素 e，移除一个不存在的元素时会发生错误
set.discard(e)	从集合 set 中移除指定元素 e，移除一个不存在的元素时不会发生错误
set.pop()	从集合 set 中随机移除一个元素
set.intersection(s)	返回 set 和 s 的交集，s 可以是一个或多个对象
set.intersection_update(s)	将 set 更新为与 s 的交集，s 可以是一个或多个参数值
set.union(s)	返回 set 和 s 的并集，s 可以是一个或多个对象
set.difference(s)	返回 set 和 s 的差集
set.difference_update(s)	将 set 更新为与 s 的差集
set.symmetric_difference(s)	返回 set 和 s 的对称差集
set.symmetric_difference_update(s)	将 set 更新为与 s 的对称差集
set.isdisjoint(s)	判断集合 set 和 s 是否包含相同的元素，如果没有则返回 True，否则返回 False
set.issubset(s)	判断集合 set 是否为 s 的子集
set.issuperset(s)	判断集合 set 是否为 s 的超集
set.clear()	移除集合中的所有元素
set.copy()	复制一个集合

其中，add()、update()、remove()、discard()、pop()方法已在前文介绍过，另外 clear()方法和 copy()方法的使用与列表使用一致。

1. 交集、并集、差集、对称差集方法

除了并集相关的方法只有 union()外，交集、差集和对称差集均各自拥有两个内置方法。其中，交集相关方法是 intersection()和 intersection_update()；差集相关方法是 difference()和 difference_update()；对称差集相关方法是 symmetric_difference()和 symmetric_difference_update()。对于每种运算，两种方法的区别在于：前者返回一个新的集合；而后者是在原始的集合上进行操作，没有返回值。

【演示】 以交集为例，演示 intersection()和 intersection_update()内置方法的区别。

```
>>> basket1 = {"apple", "banana", "cherry"}
>>> basket2 = {"cherry", "pear"}
>>> basket1.intersection(basket2)          # 计算两集合的交集并返回
    {'cherry'}
>>> basket1                                # basket1 集合未发生变化
    {'cherry', 'apple', 'banana'}
>>> basket1.intersection_update(basket2)   # 在 basket1 上进行交集运算，无返回值
>>> basket1 # basket1 集合发生变化
    {'cherry'}
```

这些内置方法除了能够接收集合对象作为参数外，也可以接收范围、列表、元组、字典、字符串对象作为参数值。其中，交集和并集的相关方法还能接收多个参数，各参数之间使用逗号隔开，表示多个数据元素上的交集和并集计算。

【演示】 多种参数的交集和并集计算。

```
>>> words = {"a", "b", "c", "d"}
>>> words.intersection(["c", "d"], ("c", "e", "f", "d"))    # 多个列表、元组参数
    {'d', 'c'}
>>> words.union(range(1,10))                                 # 参数为一个范围
    {1, 'c', 2, 'b', 3, 4, 5, 6, 'd', 7, 8, 9, 'a'}
>>> words.difference("ab")                                  # 字符串参数将拆分各个字符作为元素
    {'c', 'd'}
```

而字典类型的参数使用键作为元素进行综合运算。

【演示】 字典类型的参数作为元素进行运算。

```
>>> words = {"a", "b", "c", "d"}                # 创建集合
>>> words.union({'a': 1, 'b': 2,'e': 5, 'f': 6})  # 字典参数将用键作为元素
    {'c', 'b', 'd', 'e', 'f', 'a'}
>>> words.intersection_update({'a':1, 'b':2,'e':5, 'f':6}, {'g':7}) # 产生空集
>>> words
    set()
```

2. 相交、超集、子集判断方法

集合支持用 isdisjoint()方法来判断一个集合与另一个集合是否非相交，分别用 issubset()和 issuperset()方法来判断一个集合是否是另一个集合的子集和超集。同样地，方法的参数可以是集合，也可以是范围、列表、元组、字典、字符串。

【演示】 isdisjoint()方法的使用。

```
>>> words = {"a", "b", "c", "d"}
>>> words.isdisjoint("e") # 非相交，返回 True
    True
>>> words.isdisjoint("ab") # 字符串参数拆分为各个字符元素。相交返回 False
    False
>>> words.issubset(("a", "b", "c", "d", "e", "f")) # 元组参数，前者是后者的子集
    True
>>> words.issuperset(["e", "a"]) # 列表参数，前者不是后者的超集
    False
```

3.4 推 导 式

推导式是一种独特的数据处理方式，可以从一个区间、字符串、列表、元组、字典和集合等数据类型中，快速生成一个满足指定需求的组合数据类型。Python 语言支持多种组合数据类型的推导式，包括列表推导式、元组推导式、字典推导式和集合推导式。

3.4.1 列表推导式

【语法格式】

```
[exp for item in collection]
```

或者

```
[exp for item in collection if condition]
```

【说明】

exp 是列表生成元素表达式，可以是有返回值的函数；for item in collection 表示迭代 collection 中的各个元素，每次将元素 item 传入 exp 表达式中；if condition 是条件，可以过滤 collection 元素中不符合条件的值。

【演示】 过滤字符串列表中长度小于或等于 3 的字符串，并将符合条件的字符串转换为大写字母。

```
>>> names = ('Bob','Tom','alice','Jerry','Wendy','Smith')
>>> new_names = [name.upper() for name in names if len(name)> 3]
>>> new_names
    ['ALICE', 'JERRY', 'WENDY', 'SMITH']
>>> type(new_names)
    <class 'list'>
```

3.4.2 元组推导式

【语法格式】

```
(exp for item in collection)
```

或者

```
(exp for item in collection if condition)
```

【说明】

各变量的含义与列表推导式一致。元组推导式和列表推导式的用法也完全相同，只是元组推导式是用小括号将各部分括起来，而列表推导式用的是中括号。注意，元组推导式返回的结果是一个生成器对象，而不是直接返回元组对象。

【演示】 生成一个包含数字 1～9 的元组。

```
>>> nums = (x for x in range(1,10))
>>> nums
    <generator object <genexpr> at 0x0000020B31FEDBC8>
>>> type(nums)
    <class 'generator'>
>>> tuple(nums)    # 使用 tuple()函数可以将生成器对象转换为元组
    (1, 2, 3, 4, 5, 6, 7, 8, 9)
```

3.4.3 字典推导式

【语法格式】

```
{key_expr: value_expr for item in collection}
```

或者

```
{ key_expr: value_expr for item in collection if condition}
```

【说明】

key_expr 和 value_expr 分别是字典的键和值的表达式；for item in collection 表示迭代 collection 中的各个元素，每次将元素 item 传入 key_expr 和 value_expr 表达式中；if condition 是条件，可以过滤 collection 元素中不符合条件的值。

【演示】 根据字符串及其长度创建字典。

```
>>> websites = ['Google', 'Chrome', 'Wiki']
>>> webdict = {key:len(key) for key in websites}
>>> webdict
    {'Google': 6, 'Chrome': 6, 'Wiki': 4}
>>> type(webdict)
    <class 'dict'>
```

上述代码将列表中各字符串值作为键，字符串的长度作为值，组成新字典的键值对。

也可以借助字典推导式生成乘法表。

【演示】 生成数字 3 的乘法表。

```
>>> dic = {x: x * 3 for x in range(1, 10)}
>>> dic
    {1: 3, 2: 6, 3: 9, 4: 12, 5: 15, 6: 18, 7: 21, 8: 24, 9: 27}
>>> dic[4]  # 读字典获取 4 * 3 的值
    12
```

3.4.4 集合推导式

集合推导式的基本格式和列表推导式、元组推导式类似。

【语法格式】

```
{exp for item in collection}
```

或者

```
{exp for item in collection if condition}
```

【说明】

exp 是集合生成元素表达式，可以是有返回值的函数；for item in collection 表示迭代 collection 中的各个元素，每次将元素 item 传入 exp 表达式；if condition 是条件，可以过滤

collection 元素中不符合条件的值。

【演示】 查找字符串中不是“a”“b”“c”的字母，并输出。

```
>>> words = {x for x in 'abandon' if x not in 'abc'}
>>> words
    {'d', 'n', 'o'}
>>> type(words)
    <class 'set'>
```

本章小结

列表和元组比较相似，都属于序列。它们都按顺序保存元素，所有的元素占用一块连续的内存，每个元素都有自己的索引，因此列表和元组的元素都可以通过索引来访问。它们的区别在于：列表是可以修改的，而元组是不可修改的。字典和集合存储的数据都是无序的，每个元素占用不同的内存，其中字典元素以键值对的形式保存，而集合的元素是非重复的、无序的。列表、元组、字典和集合都可以通过推导式来快速创建。

本章习题

（1）简述列表结构的基本特点。

（2）简述列表与元组的主要区别。

（3）索引号有什么概念？

（4）编程：建立字典 D={"数学":101,"语文":202,"英语":203,"物理":204,"生物":206}；完成向字典中添加键值对“"化学":205”；修改"数学"对应的键值为 201；删除"生物"对应的键值对；以输入为键，输出字典中对应的值，若键不存在，则输出“输入的键不存在!”。

（5）编程：输入一个列表和两个整数表示的列表索引号，输出列表中索引号为两数的元素组成的子列表。如输入[1,2,3,4,5,6]和 2,5，输出[3,6]。

（6）编程：根据输入的字符串，输出用逗号分隔每个字母后的字符串。

（7）编程：用户输入由数字或字符元素组成的两组字典，编写程序将两个字典合并为一个字典，如果两个字典中有相同的键，需将对应的值相加后作为字典中该键对应的新值，要求输出合并后的字典是按照值递增排序的，如果有值相同则再按键递增排序。