第 3 章　Python 容器数据类型

除整型、浮点型和布尔型等基本类型外，Python 还提供列表、元组、字典和集合等容器数据类型，容器数据可以容纳批量数据。这些容器数据类型又可分为序列类型、映射类型和集合类型。序列类型包括字符串、列表和元组等，它们的元素都是有序排列的，可以通过索引下标访问它们的元素。字典属于映射类型。映射类型的元素由键（key）和值（value）组成，简称“键值对”。在一个映射型的变量中，键是唯一的。集合类型变量中的元素是没有顺序的，且不允许出现重复的元素，类似数学中的集合。

3.1　列表

列表（list）是 Python 中最常用的序列类型。它由一系列元素（又称数据项）组成，所有元素都被包含在一对方括号“[]”中。列表中的元素可以是任何类型的数据，并且不需要所有元素具有相同的类型。列表具有如下特性：

（1）列表中的元素类型可以相同，也可以不同。

（2）每个列表元素都有索引和值两个属性，索引用于标识元素在列表中的位置，值指的是元素本身的值。

序列类型的元素索引分为正索引和负索引两种情况（见图 3.1）。正索引的索引值从 0 开始，在列表中从左向右依次递增 1，因此最后一个元素的索引为元素个数 − 1。负索引的索引值从-1 开始，在列表中从右向左依次递减 1。一般情况下列表的索引是正索引。

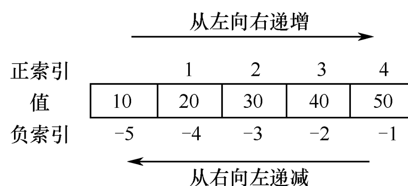

图 3.1　元素的正索引和负索引

3.1.1　创建列表和存取列表元素

可以使用方括号“[]”创建一个列表对象，也可以使用函数 list()创建一个列表对象。创建列表的代码格式是

```
[元素 1, 元素 2, … ]
```

或

```
list(元素 1, 元素 2, …)
```

【例】列表的创建。

```
In: lst = [ ]                       # 创建空列表
In: lst
Out: []

In: list()                          # 创建空列表
Out: []

In: [12, 3.0, 'python']             # 列表元素的数据类型不相同
Out: [12, 3.0, 'python']
```

```
In: list('python')              # 字符串中的每个字符成为列表中的一个元素
Out: ['p', 'y', 't', 'h', 'o', 'n']
```

3.1.2 列表基本操作

1. 访问列表元素

可以使用索引访问列表中的元素，其格式如下：

```
列表对象[index]
```

index：元素索引。

【例】访问列表中的元素。

（1）获取 lst 列表中索引为 1 的元素

```
In: lst = [1, 2, 3]
In: lst[1]
Out: 2
```

（2）修改 lst 列表中索引为 1 的元素的值

```
In: lst = [1, 2, 3]
In: lst[1] = 5                  # 将索引为 1 的元素的值修改为 5
In: lst
Out: [1, 5, 3]
```

2. 列表合并

可以使用运算符“+”将两个列表合并在一起。

【例】合并两个列表。

```
In: [1, 2] + [3, 'abc']
Out: [1, 2, 3, 'abc']
```

3. 重复

可以使用运算符“*”创建具有重复元素的列表。

【例】创建元素“1, 2”重复 3 次的列表。

```
In: [1, 2] * 3
Out: [1, 2, 1, 2, 1, 2]
```

4. 迭代

迭代操作可用于遍历列表中的元素。

【例】迭代输出列表 lst 中的元素。

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

5. 成员判断

使用“in”操作符判断对象是否属于列表。

【例】判断对象是否属于列表。

```
In: 2 in [1, 2, 3]
Out: True
In: 5 in [1, 2, 3]
Out: False
```

3.1.3　列表常用函数

Python 提供了一系列处理列表对象的函数，可以对列表进行元素添加、删除、排序和反转等操作。表 3.1 列出了列表的常用函数。

表 3.1　列表的常用函数

函　数　名	说　　明
append(x)	在尾部添加新元素 x
index(x)	返回 x 在列表中的索引位置，不含 x 将返回一个运行时错误
count(x)	统计指定元素值 x 出现的次数，不含 x 将返回 0
del 对象[index]	删除列表对象中索引为 index 的元素
extend(lst)	添加新列表
insert(index, x)	将 x 插入到列表的 index 位置
len(lst)	求列表长度
pop()	删除元素，并返回删除对象
remove(x)	删除元素 x
reverse()	逆排列
sort()	排序

【例】列表常用函数的应用。

（1）在 lst 列表末尾添加元素

```
In: lst = [1, 2, 3]
In: lst.append(5)       # 末尾插入
In: lst
Out: [1, 2, 3, 5]
In: lst.index(1)        # 数据 1 在列表的第 0 个位置
Out:0
In: lst.index(4)        # 将报错，列表中不包含数据 4
ValueError: 4 is not in list
```

（2）统计 lst 列表中元素值 2 出现的次数

```
In: lst = [1, 2, 2, 2, 3]
In: lst.count(2)        # 2 出现了 3 次
Out: 3
In: lst.count(4)        # 列表中不含数据 4，返回 0 次
Out: 0
```

（3）删除 lst 列表索引为 1 的元素

```
In: lst = [1, 2, 3]
In: del lst[1]
In: lst
Out: [1, 3]
```

（4）将列表 x 中的元素添加到 lst 列表中

```
In: lst = [1, 2, 3]
In: x = [4, 5]
In: lst.extend(x)
In: lst
Out: [1, 2, 3, 4, 5]
```

（5）在 lst 列表索引为 1 的位置插入值为 5 的元素

```
In: lst = [1, 2, 3]
In: lst.insert(1, 5)
In: lst
Out: [1, 5, 2, 3]
```

（6）获取 lst 列表的长度（即列表元素的个数）

```
In: lst = [1, 2, 3]
In: len(lst)
Out: 3
```

（7）删除 lst 列表指定索引位置的元素，并返回该删除元素；若没有指定索引，则删除列表末尾的元素

```
In: lst = [1, 2, 3]
In: lst.pop(1)
Out: 2
In: lst
Out: [1, 3]
```

（8）删除 lst 列表中值为 2 的元素，如果有多个，那么只删除第一个

```
In: lst = [1, 2, 2, 3]
In: lst.remove(2)
In: lst
Out: [1, 2, 3]
```

（9）将 lst 列表中的元素反转

```
In: lst = [1, 2, 5, 4]
In: lst.reverse()
In: lst
Out: [4, 5, 2, 1]
```

（10）对 lst 列表中的元素进行排序，若是数值型数据则按从小到大排序，若是字符串则按字典顺序排序。sort()函数的参数 reverse=True 时为逆序排序，默认为 reverse=False 升序排序

```
In: lst = [1, 4, 3, 5, 2]
In: lst.sort()                      # 默认升序
In: lst
Out: [1, 2, 3, 4, 5]
In: lst = [1, 4, 3, 5, 2]
In: lst.sort(reverse=True)          # 参数 reverse=True 表示进行逆序排序
In: lst
Out: [5, 4, 3, 2, 1]
```

注意上式不能写为 lst=lst.sort()，因为 lst.sort()执行后未返回新列表（返回值为 None），若将 None 值赋给 lst，则 lst 原来的数据就会丢失。

lst.sort()方法直接改变原列表的顺序，如果排序时希望原列表不变，返回一个新的有序列表，那么可以借助 sorted()函数。

```
In: lst2 = sorted(lst)              # 执行后 lst 不变，得到新的有序列表 lst2
```

3.1.4　切片

列表的切片操作可以获得列表的多个元素。切片操作的基本格式如下：

```
列表对象[start : end : step]
```

start：索引开始位置，可以省略，默认为 0。

end：索引结束位置（不包括 end），可以省略，默认为列表长度。

step：步长，表示截取数据的步长，正数表示从左向右截取，负数表示从右向左截取，可以省略但不能为 0，默认为 1。

【例】列表切片应用。

```
In: lst = [1, 2, 3, 4, 5]
In: lst[1:4]            # 截取列表 lst 中从索引 1 到索引 3 的元素
Out: [2, 3, 4]
In: lst[1:]             # 截取列表 lst 中从索引 1 到末尾的元素
Out: [2, 3, 4, 5]
In: lst[:4]             # 截取列表 lst 中从开始到索引 3 的元素
Out: [1, 2, 3, 4]
In: lst[1:4:2]          # 截取列表 lst 中从索引 1 到索引 3，步长为 2 的元素
Out: [2, 4]
In: lst[-4:-1]          # 截取列表 lst 中从索引-4 到索引-2 的元素
Out: [2, 3, 4]
In: lst[3:0:-1]         # 步长为负，从右向左截取列表 lst 中从索引 3 到索引 1 的元素
Out: [4, 3, 2]
In: lst[-2:-5:-1]       # 步长为负，从右向左截取列表 lst 中从索引-2 到索引-4 的元素
Out: [4, 3, 2]
```

3.1.5　列表生成方式

1．产生一个数值递增列表

使用 range()函数可以产生一个数值递增且可迭代操作的对象，基本格式如下：

```
range(start, end, step)
```

start：起始元素值，可以省略，默认为 0。

end：结束元素值（不包括 end），不能省略。

step：步长，正数表示数值递增，负数表示数值递减，可以省略但不能为 0，默认为 1。

注意 range(100)并未在内存中立即产生 100 个数，只是产生了一个可迭代对象，具体数据将在后续迭代过程中逐一产生，这样有利于节省内存。将 range()函数产生的迭代对象作为列表创建函数 list()的参数，可以产生一个数值递增的列表。

【例】产生一个数值递增列表。

```
In: it = range(1, 5)        # 产生一个数值从 1 到 4 的迭代对象 it
In: lst = list(it)          # 以迭代对象为参数创建列表
In: lst
Out: [1, 2, 3, 4]
```

2．产生多维列表

多维列表可视为列表中嵌套的列表。例如，二维列表可视为每个元素都是一维列表的列表，三维列表可视为每个元素都是二维列表的列表。

【例】创建一个二维列表。

```
In: lst = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

此时，二维列表 lst 的内容如下所示，每行是一个一维列表：

1	2	3	4
5	6	7	8
9	10	11	12

多维列表的维数是从 0 开始的。例如，二维列表有第 0 维和第 1 维，分别对应于行和列。访问多维列表中的元素时，使用的命令格式如下：

```
列表对象[row][col]…
```

其中第一个“[]”为列表的第 0 维，第二个“[]”为第 1 维，以此类推。

【例】多维列表的访问。

```
In: lst = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
In: lst[1][2]              # 获取第 0 维索引为 1，第 1 维索引为 2 的数据
Out: 7
In: lst[1]                 # 获取第 0 维索引为 1 的数据，是一个一维列表
Out: [5, 6, 7, 8]
```

3. 列表生成式

Python 的列表还可以用其特有的列表生成式语法产生。列表生成式的一般格式如下：

```
新列表对象=[ 表达式  for  变量  in  可迭代对象 <if 条件> ]
```

它表示从可迭代对象中取得变量的值，将变量代入表达式计算得到新值，由这些新值构成新的列表。<if 条件>是可选的，如果设有筛选条件，那么满足条件的变量值才能加入新列表。

```
In: lst = list(range(5))
In: lst
Out: [0, 1, 2, 3, 4]
In: lst2 = [x**2 for x in lst]
      # 列表生成式，将 lst 列表中的每个元素平方，得到列表 lst2
In: lst2
Out: [0, 1, 4, 9, 16]
In: lst3 = [ x for x in lst2 if x not in lst ]
      # 抽取在 lst2 中但不在 lst 中的元素得到列表 lst3
In: lst3
Out: [9, 16]
In: [ x for x in range(20) if x%3==0 ]  # [0, 20]内的 3 的倍数
Out:[0, 3, 6, 9, 12, 15, 18]
```

3.2 元组

元组（tuple）是用一对圆括号“()”定义的序列。元组中的元素值自元组创建后就不能修改。元组中的元素数据类型可以相同，也可以不相同。

3.2.1 创建元组和存取元组元素

1. 创建元组

可以使用圆括号“()”定义创建一个元组对象，也可以使用函数 tuple()创建一个元组对象。

元组创建的格式如下：

```
    (元素 1，元素 2，…)          # ()可省略
```

或

```
    tuple(可迭代对象)            # 可迭代对象：字符串、列表、元组、字典、集合等
```

【例】创建三个元素的元组。

```
In: tpl = (1, 2, 3)             # 使用“()”创建元组
In: tpl
Out: (1, 2, 3)

In: tpl2 = tuple([1, 2, 3])     # 使用 tuple()函数创建元组，参数为列表对象
In: tpl2
Out: (1, 2, 3)

In: it = range(1, 5)            # 产生可迭代对象
In: tpl = tuple(it)             # 使用 tuple()函数创建元组，参数为可迭代对象
In: tpl
Out: (1, 2, 3, 4)
```

2．索引和切片

元组的索引和切片操作与列表的索引和切片操作非常相似。

【例】元组的索引操作和切片操作。

（1）访问元组 tpl 中索引为 1 的元素

```
In: tpl = (1, 2, 3)
In: tpl[1]
Out: 2
```

（2）截取元组 tpl 中从索引位置 1 到索引位置 3 的元素

```
In: tpl = (1, 2, 3, 4, 5)
In: tpl[1:4]
Out: (2, 3, 4)
```

3．其他操作

在 Python 中，元组的其他操作，如求元组的长度、合并、重复、迭代、成员判断等，也与列表的对应操作类似。

【例】元组的操作。

（1）元组求长度

```
In: tpl = (1, 2, 3)
In: len(tpl)
Out: 3
```

（2）合并两个元组

```
In: (1, 2, 3) + (4, 5)
Out: (1, 2, 3, 4, 5)
```

需要注意的是，由于元组中的元素是不能增删的，因此合并元组不是在已有的元组中添加元素，而是产生一个新元组，新元组与已有元组在内存中是独立的。

（3）重复，创建一个元素“1，2”重复 3 次的元组

```
In: (1, 2) * 3
Out: (1, 2, 1, 2, 1, 2)
```

（4）迭代元组中的所有元素

```
In: tpl = (1, 2, 3)
In: for x in tpl:
      print(x)
1
2
3
```

（5）判断对象是否为元组中的元素

```
In: 2 in (1, 2, 3)
Out: True
In: 5 in (1, 2, 3)
Out: False
```

（6）统计 tpl 元组中元素值 2 出现的次数

```
In: tpl = (1, 2, 2, 2, 3)
In: tpl.count(2)
Out: 3
```

3.2.2 元组和列表的差异

元组一旦创建，其中的元素就不可修改。元组也不能增加或删除元素。因为元组结构简单，所以元组占用的内存比列表占用的内存少，如果待处理的数据多用于查询，无须修改，那么可以考虑用元组存储。另外，由于元组是不可修改的，在函数调用或返回函数值时可考虑传递元组，以避免参数被误修改。

```
In: tpl = (1, 2, 3)
In: tpl[0] = 100            # 将报错，元组中的元素不可修改
TypeError: 'tuple' object does not support item assignment
In: tpl.append(4)           # 将报错，元组不支持 append()方法
AttributeError: 'tuple' object has no attribute 'append'
```

虽然不可以直接修改元组中的元素，但如果需要改变元组中的元素，那么可以先将元组转换为列表，然后对列表进行修改，再将修改好的列表转换为新的元组。列表和元组相互转换的函数是 tuple(lst)和 list(tpl)。

【例】将 tpl 元组中索引位置 1 的值由 2 改为 5。

```
In: tpl = (1, 2, 3)
In: lst = list(tpl)              # 将元组转换为列表
In: lst[1] = 5
In: tpl = tuple(lst)             # 将列表转换为元组
In: tpl
Out: (1, 5, 3)
```

3.2.3 序列操作函数

针对列表和元组等序列类型，Python 提供了一些实用的序列操作函数，如表 3.2 所示。

表 3.2　常用的序列操作函数

函　数　名	功　　能
all(seq)	判断序列对象的每个元素是否都为逻辑真值，返回 True 或 False
any(seq)	判断序列对象是否有逻辑真值的元素，返回 True 或 False
range(start,stop[, step])	返回从 start（默认值为 0）开始，到 stop 结束（不包括 stop）的可迭代对象，step 为步长（默认值为 1）
reversed()	反转序列，返回可迭代对象
sorted(iter)	对序列对象 iter 进行排序，返回一个有序列表
zip(iter1[, iter2,…])	将多个迭代对象相同位置的元素聚合成元组，返回一个以元组为元素的可迭代对象

【例】常用的序列操作函数。

```
In: all([1, 2, 'Python'])        # 非 0 元素对应布尔值 True，本例所有元素都对应 True
Out: True
In: all([0, 1, 2, 'Python'])     # 0 对应的布尔值为 False
Out: False
In: any([0, 1, 2, 'Python'])
Out: True
In: lst = [3, 2, 5, 4]
In: sorted(lst)                  # 排序
Out: [2, 3, 4, 5]
In: r = reversed(lst)            # 反转，生成可迭代对象
In: list(r)                      # 由可迭代对象创建列表
Out: [4, 5, 2, 3]
In: z = zip(['a', 'b', 'c'], [1, 2, 3])  # 产生一个以两个列表元素聚合成元组的新迭代对象
In: list(z)                      # 由可迭代对象创建列表
Out: [('a', 1), ('b', 2), ('c', 3)]
```

3.3　字典

字典（dict）可视为键值对构成的数据容器。搜索字典中的元素时，首先查找键，然后根据找到的键获取对应的值。这是一种高效、快速的查找方法。字典中的键必须是唯一的。

3.3.1　创建字典和存取键值对

1．创建字典

可以使用标记“{}”创建一个字典对象，也可以使用函数 dict()创建一个字典对象。字典中的每个元素都包含键和值两部分内容，创建字典时，键和值用冒号“:”隔开，字典的元素之间用逗号“,”分开。创建字典的格式如下：

```
{键 1:值 1, 键 2:值 2, …}
```

或

```
dict(键 1=值 1, 键 2=值 2, …)
```

【例】创建字典。

```
In: di = {}        # 空字典
In: di
```

```
Out: {}
In: dict()          # 空字典
Out: {}
In: di = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: di
Out: {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
```

2. 字典的访问

字典中元素的访问是通过键获取相应的值。访问字典元素的格式如下：

```
字典对象[key]          # key：字典中元素的键
```

【例】访问字典中的元素。

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct['姓名']         # 通过键取值
Out: 'zhangsan'
In: dct['001']          # 将报错，不能直接取出值。字典中没有名为'001'的键
KeyError: '001'
```

3. 添加和修改字典元素

添加和修改字典元素的格式如下：

```
字典对象[key] = value
```

key：元素的键。

value：元素的值。

如果字典中不存在 key 键，那么在字典中添加一个 key：value 的元素，如果字典中存在 key 键，那么将字典中 key 键对应的值修改为 value 值。

【例】添加、修改字典元素。

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct['性别'] = '男'           # 在字典中添加键为"性别"，值为"男"的数据
In: dct
Out: {'学号': '001', '姓名': 'zhangsan', '年龄': 19, '性别': '男'}

In: dct['年龄'] = 20             # 修改字典中"年龄"的值为 20
In: dct
Out: {'学号': '001', '姓名': 'zhangsan', '年龄': 20, '性别': '男'}
```

3.3.2　字典的常用方法

Python 提供了许多专门处理字典对象的函数，表 3.3 列出了其中比较常用的函数。

表 3.3　字典常用函数

函　数　名	说　　明
clear()	清除字典中的所有元素
copy()	复制整个字典
del 对象[key]	删除字典中键为 key 的元素
get(key, default)	返回键 key 对应的值，如果字典中不存在 key 键，那么返回 default
items()	返回所有键值对

（续表）

函　数　名	说　　明
keys()	返回所有键
values()	返回所有值
update(dct)	使用 dct 字典数据更新当前字典
pop(key, default)	返回键 key 对应的值，并删除该元素，若字典中不存在 key 键，则返回 default

【例】字典常用函数的应用。

（1）清除字典对象中的所有元素

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct.clear()
In: dct
Out: {}
```

（2）复制字典对象 dct

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: d = dct.copy()
In: d
Out: {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
```

（3）删除字典对象中的元素

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: del dct['年龄']
In: dct
Out: {'学号': '001', '姓名': 'zhangsan'}
```

（4）获得字典对象中键对应的值

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct.get('年龄')                    # 获取字典中键为“年龄”的值
Out: 19
In: dct.get('性别', '男')              # 字典中无“性别”键，默认返回“男”
Out: '男'
```

（5）获得字典对象中键对应的值，并删除该元素

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct.pop('年龄')                    # 返回字典中键为“年龄”的值，并删除该元素
Out: 19
In: dct
Out: {'学号': '001', '姓名': 'zhangsan'}
```

（6）获得字典对象中的所有键

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct.keys()                         # 获取字典中的所有键
Out: dict_keys(['学号', '姓名', '年龄'])
```

（7）获得字典对象中的所有值

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct.values()                       # 获取字典中的所有值
dict_values(['001', 'zhangsan', 19])
```

（8）获得字典对象中的所有键值对

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: dct.items()                        # 获取字典中的所有键值对
```

```
Out: dict_items([('学号', '001'), ('姓名', 'zhangsan'), ('年龄', 19)])
```

（9）更新字典 dct 中的元素

```
In: dct = {'学号': '001', '姓名': 'zhangsan', '年龄': 19}
In: d = {'学号': '002', '年龄': 20}
In: dct.update(d)                    # 根据字典 d 中的数据修改字典 dct
In: dct
Out: {'学号': '002', '姓名': 'zhangsan', '年龄': 20}
```

3.4 集合

集合（set）是由 0 个或多个元素构成的无序组合，集合中的元素不允许重复。集合是可变的，即可以添加或删除集合中的元素。集合不支持索引操作。

3.4.1 创建集合

可以使用标记“{}”创建一个集合对象，也可以使用函数 set()创建一个集合对象。集合中的元素用逗号“,”分隔。与字典中的每个元素都包含键和值两部分内容不同，集合中的每个元素只有值，没有键。创建集合的格式如下：

```
{元素 1, 元素 2, …}
```

或

```
set(可迭代对象)
```

【例】创建集合。

```
In: set()                           # 创建空集合
Out: set()

In: {'C++', 'Java', 'Python'}       # 创建 3 个元素的集合
Out: {'Java', 'C++', 'Python'}      # 集合元素是无序的，此处的显示顺序和创建顺序不一致

In: set(['C++', 'Java', 'Python'])    # 以列表为参数，创建 3 个元素的集合
Out: {'Java', 'C++', 'Python'}
In: set([1, 1, 3, 3, 5, 5, 5])        # 集合不允许重复元素，所以可以利用集合快速去重
Out:{1, 3, 5}
```

创建空集合只能使用 set()函数而不能使用“{}”方式，因为“{}”创建的是空字典而不是空集合。

3.4.2 遍历集合

由于集合中的元素没有索引的属性，同时也没有像字典中“键值对”的对应关系，因此无法直接获取集合中的指定元素，只能通过迭代来遍历集合中的所有元素。

【例】遍历集合 s。

```
In: s = {'C++','Java','Python'}
In: for x in s:
      print(x)          # 输出的顺序可能和创建时的顺序不一致
Python
Java
C++
```

3.4.3　集合操作函数

Python 提供了一系列对集合对象进行各种操作的函数。表 3.4 列出了常见的集合操作函数。

表 3.4　常见的集合操作函数

函　数　名	说　　明
add(x)	添加元素，若集合中不存在元素 x，则添加
clear ()	清除所有元素
copy()	复制整个集合
len(s)	返回集合 s 的元素个数
pop()	随机返回集合中的一个元素，并删除该元素
remove(x)	删除元素 x，若集合中不存在 x，则报错
discard(x)	删除元素 x，即使 x 不存在也不报错
update(s)	将另一个集合的元素添加进来

【例】集合操作的常用函数的应用。

（1）向集合中添加元素

```
In: s = {'C++', 'Java', 'Python'}
In: s.add('PHP')
In: s
Out: {'PHP', 'Java', 'C++', 'Python'}
```

（2）删除集合中的元素

```
In: s = {'C++', 'Java', 'Python'}
In: s.remove('Java')             # 使用 remove()函数删除元素
In: s
Out: {'C++', 'Python'}
In: s.remove('Java')             # 再次删除，集合中已无 Java 元素将报错
KeyError: 'Java'
In: s.discard('Java')            # 使用 discard()删除，无 Java 元素也不报错
```

（3）清除集合中的所有元素

```
In: s = {'C++', 'Java', 'Python'}
In: s.clear()                    # 使用 clear()函数清除元素后的集合为空集合
In: s
Out:set()
```

（4）复制整个集合

```
In: s = {'C++', 'Java', 'Python'}
In: x = s.copy()
In: x
Out: {'Java', 'C++', 'Python'}
```

（5）将集合 t 中的元素添加进来

```
In: s = {'C++', 'Java', 'Python'}
In: t = {'PHP', 'C++'}
In: s.update(t)                    # 根据集合 t 的数据更新集合 s 的元素
In: s
Out: {'PHP', 'C++', 'Java', 'Python'}
```

（6）随机返回集合中的一个元素

```
In: s = {'C++', 'Java', 'Python'}
In: s.pop()                          # 随机返回一个元素，并在集合中删除该元素
Out: 'Java'
In: s
Out: {'C++', 'Python'}
```

（7）获取集合元素个数

```
In: s = {'C++', 'Java', 'Python'}
In: len(s)
Out: 3
```

3.4.4 集合运算：并、交、差

与数学中的集合概念一样，Python 中的集合也支持两个集合的并集、交集、差集等各种运算。常见的集合运算符如表 3.5 所示。

表 3.5 常见的集合运算符

运 算 符	说 明
S & T	交集，返回一个由两个集合 S 和 T 中都存在的元素构成的集合
S | T	并集，返回一个包含了两个集合 S 和 T 中的所有元素的集合
S - T	差集，返回一个由在集合 S 中存在但在集合 T 不存在的元素构成的集合
S ^ T	对称差集，返回一个由不在集合 S 和集合 T 中同时存在的元素构成的集合
S == T	集合 S 和集合 T 中的元素相同返回 True，否则返回 False
S != T	集合 S 和集合 T 中的元素不相同返回 True，否则返回 False
S <= T	子集测试，集合 S 是集合 T 的子集返回 True，否则返回 False
S < T	真子集测试，集合 S 是集合 T 的真子集返回 True，否则返回 False
S >= T	超集测试，集合 S 是集合 T 的超集返回 True，否则返回 False
S > T	真超集测试，集合 S 是集合 T 的真超集返回 True，否则返回 False

【例】常见的集合运算符的应用。

```
In: s = {10, 20, 30}
In: t = {20, 30, 40}
In: s & t                        # 交集
Out: {20, 30}
In: s | t                        # 并集
Out: {20, 40, 10, 30}
In: s - t                        # 差集
Out: {10}
In: s ^ t                        # 对称差集
Out: {40, 10}
In: s == t                       # 判断集合元素是否相同
Out: False
In: s != t                       # 判断集合元素不相同
Out: True
In: s = {10, 20, 30}
In: t = {10, 20}
```

```
In: s > t                        # 真超集测试
Out: True
In: s < t                        # 真子集测试
Out: False
```

3.5　可变类型和不可变类型

程序中的数据必须以特定类型存放在内存的某个区域中。根据变量对应的内存区域是否可以修改，Python 将数据类型分为不可变类型和可变类型。如果变量的值发生改变后对应的内存地址也发生改变，那么这种数据类型称为不可变数据类型。如果变量的值发生改变后对应的内存地址保持不变，那么这种数据类型称为可变数据类型。

Python 中的整型、浮点型、字符串型和元组属于不可变数据类型，列表、字典和集合属于可变数据类型。

【例】不可变数据类型的应用。

```
In: a = 10
In: id(a)                        # id()返回 a 的唯一标识符，等同于 a 的内存地址
Out: 262719776                   # 变量 a 的初始内存地址
In: a = 20                       # 改变 a 的值
In: id(a)
Out: 262719936                   # 变量 a 的内存地址发生改变
```

上面的代码中，语句"a = 20"并不是在变量 a 原来的内存区域（存放 10 的区域）上修改值，而是在另外一个内存区域存放"20"，因此变量 a 的内存地址发生了改变。注意，Python 的整型是不可变数据类型，并不是说整型变量赋值后就不可改变，而是说整型变量对应的内存区域不允许修改。为整型变量赋新值后，会使用新的内存空间存储新值，变量也指向新的内存区域。旧内存区域如果未被其他变量指向，那么会被废弃，并由 Python 自动回收内存空间。

【例】可变数据类型的应用。

```
In: lst = [1, 2, 3]
In: id(lst)
Out: 204715272                        # lst 的内存地址
In: lst.append(4)                     # lst 添加元素 4
In: lst[0] = 99                       # 修改 lst 中的第 0 个元素
In: lst
Out: [99, 2, 3, 4]
In: id(lst)
Out: 204715272                        # lst 的内存地址没有改变
```

上面的程序的执行结果表明，虽然使用 append()函数对列表 lst 添加了元素，也修改了第 0 个元素，但 lst 的内存地址并未改变，这表明是在原有内存中对旧值进行修改的。

【例】可变数据变量赋值时内存指向相同的问题。

```
In: lst = [1, 2, 3]
In: id(lst)
Out: 204715272                        # lst 的内存地址
In: lst2 = lst                        # 可变数据变量 lst 赋值给 lst2
In: id(lst2), id(lst)
Out: (204715272, 204715272)           # lst2 的内存地址与 lst 一样
```

```
In: lst2.append(4)                    # lst2 添加元素 4
In: lst2[0] = 99                      # 将 lst2 的第 0 个元素修改为 99
In: lst2
Out: [99, 2, 3, 4]
In: lst
Out: [99, 2, 3, 4]                    # lst 的元素也改变了
```

上面的程序显示，列表 lst 和 lst2 共享相同的内存地址，因此对 lst2 进行改变时，lst 也发生了改变，而这种改变可能并不是程序员所希望的。在对列表、字典这类可变数据对象进行操作时，要特别注意这种修改产生的副作用。

3.6 浅复制和深复制

为了解决可变变量赋值时两个对象指向同一内存区域从而产生修改时相互影响的问题，Python 提供了浅复制和深复制两种解决方法。

```
In: lst = [1, 2, 3]
In: lst2 = lst.copy()       # 浅复制，lst2 采用新存储空间保存赋值得到的 lst 的内容
In: id(lst),  id(lst2)      # 对比 lst 和 lst2 的内存地址
Out: (1979528444104, 1979527296520)      # 可见两个列表对象的内存地址不同
In: lst2.append(4)
In: lst2[0] = 99
In: lst2
Out: [99, 2, 3, 4]          # lst2 已变动
In: lst
Out: [1, 2, 3]              # 可见 lst 未变动

In: import  copy            # 引入 copy 模块
In: lst3 = copy.copy(lst)   # 也可利用 copy 模块中的 copy()函数完成浅复制
In: id(lst3), id(lst)
Out: (1979528442952, 1979528444104)  # 两个列表对象的内存区域不同
```

浅复制是在赋值时开辟新的存储空间来保存新对象。新旧对象由于在内存中是独立的，所以不会出现修改时相互影响的问题。浅复制初步解决了内存指向相同的问题，但如果被复制的对象内部还含有可变元素，那么浅复制时内部的可变元素还是会指向相同的地址，示例代码如下：

```
In: lst = [1, 5, ['a', 'b']]          # lst 的第 2 个元素是一个可变元素['a', 'b']
In: lst2 = lst.copy()                 # 浅复制
In: id(lst),  id(lst2)                # lst 和 lst2 的内存地址不同
Out: (1979528687752, 1979527288008)
In: lst2[0] = 99                      # 修改 lst2 的第 0 元素
In: lst2[2][1] = 'c'                  # 修改 lst2 中索引为 2 的元素（可变数据类型）
In: lst2
Out: [99, 5, ['a', 'c']]              # lst2 修改后的数据
In: lst
Out: [1, 5, ['a', 'c']]               # lst 的可变数据元素被改变，不可变数据元素未变
In: id(lst[2]),  id(lst2[2])          # 可见 lst 和 lst2 的第 2 个元素（列表）内存地址相同
Out: (1979527288904, 1979527288904)
```

从上面的程序的执行结果可以看出，浅复制的两个变量 lst 和 lst2 虽然内存地址不同，但

是如果变量中有可变数据元素，那么这些可变元素的内存地址还是相同的，改变一个可变元素的值，另一个可变元素也会受到影响。为了将两个对象的内存地址彻底分开，可借助 copy 模块中的 deepcopy()函数进行深复制。

```
import copy                          # 导入 copy 模块
In: lst = [1, 5, ['a', 'b']]         # lst 的第 2 个元素是一个可变元素['a', 'b']
In: lst3 = copy.deepcopy(lst)        # 使用 copy 模块的 deepcopy()函数进行深复制
In: lst3[0] = 99                     # 修改 lst3 的第 0 个元素
In: lst3[2][1] = 'c'                 # 修改 lst3 中索引为 2 的元素，可变数据类型
In: lst3
Out: [99, 5, ['a', 'c']]             # lst3 修改后的数据
In: lst
Out: [1, 5, ['a', 'b']]              # lst 的数据不变
In: id(lst[2]),  id(lst3[2])         # 深复制时，lst 和 lst3 的第 2 个元素内存地址不同
Out: (1979528442056, 1979528427912)
```

从上面的代码的执行结果可以看出，深复制的两个变量 lst 和 lst3 是完全独立的，改变 lst3 中的可变元素，lst 不会随之发生改变。

3.7　本章小结

1. 序列类型数据包括字符串、列表和元组。
2. 列表是一种可变序列类型，使用“[]”或 list()函数创建列表对象。通过添加、修改和删除改变列表中元素的值，可以索引或切片访问。
3. 元组是一种不可变序列类型，使用“()”或 tuple()函数创建元组对象，元组中的元素不能增删，元素值不能修改，可以索引或切片访问。列表和元组相互转换的函数是 tuple()和 list()。
4. 字典是一种映射类型，由键和值组成，是可变数据类型，其元素没有固定的顺序，不能索引访问。使用“{}”创建字典对象，通过键访问值。
5. 集合是一种无序且不重复的元素集，使用“{}”创建集合对象。与字典不同，集合中的元素不是“键值对”形式的。
6. 在 Python 语言中，数据类型可分为可变数据类型和不可变数据类型。
7. 对可变数据类型，变量之间的赋值可借助浅复制或深复制实现内存指向的独立。

习题

一、单项选择题

1. 下列选项中，属于元组操作的函数是（　　）
   A. pop()　　B. sort()　　C. reverse()　　D. count()
2. 下列选项中，不属于字典操作的函数是（　　）
   A. clear()　　B. keys()　　C. update()　　D. sort()
3. 下列选项中，不能使用索引运算的是（　　）
   A. 列表　　B. 元组　　C. 集合　　D. 字符串
4. 下列字典定义正确的是（　　）
   A. a = ['a', 1, 'b', 2, 'c', 3]　　B. b = ['a':1, 'b':2, 'c':3]

C. c = {a:1, b:2, c:3}　　D. d = {'a':1, 'b':2, 'c':3}

5. 代码 type({'a',1,'b',2,'c',3})的运算结果是（　　）

A. <class 'list'>　　B. <class 'tuple'>

C. <class 'dict'>　　D. <class 'set'>

6. 代码 len({1,2,2,2,3,4,5})的运算结果是（　　）

A. 1　　B. 3　　C. 5　　D. 8

7. 下列关于列表的说法中，错误的是（　　）

A. list 是一个有序集合，可以添加或删除元素

B. list 是不可变的数据类型

C. list 可以存放任意类型的元素

D. 使用 list 时，索引可以是负数

二、简答题

1. 简述列表的特性。
2. 简述元组的特性。
3. 简述字典的特性。
4. 简述集合的特性。
5. 列表和元组两种序列结构有何区别？
6. 列表、元组、字典和集合分别用什么符号或函数创建？
7. 集合运算符&、|、-、^、>=和<=分别表示什么运算？

三、程序题

1. 输出列表 lst=[1,2,1,12,10,5,2,7,1,8]中不重复的元素，并统计数据个数。
2. 假设两个元组 x=(1,3,2)和 y=(5,9,4,7)，将两个元组的数据合并再并按从小到大的顺序排序。
3. 假设两个集合 a={1,2,3,4,5}和 b={2,4,6}，找出属于集合 a 但不属于集合 b 的元素。
4. 使用字典保存学生姓名和对应成绩，输出所有学生姓名，并找出某个学生的成绩。