第5章

组合数据类型

拓展阅读

学习目标

- 了解组合数据类型，能够说出序列类型、集合类型和映射类型各自的特点
- 掌握列表的操作，能够创建列表并访问、添加、排列、删除列表的元素
- 掌握列表推导式，能够通过列表推导式创建列表
- 掌握元组的操作，能够独立创建元组并访问元组的元素
- 熟悉集合的操作，能够熟练地创建集合并操作集合的元素
- 掌握字典的操作，能够独立创建字典并操作字典的元素
- 掌握字典推导式，能够通过字典推导式创建字典
- 掌握运算符在组合数据类型中的使用规则，能够通过+、*、in、not in 这几个运算符对组合数据类型进行操作

程序中有时不仅要处理数字、字符串这些基础类型的数据，还需要处理一些混合数据。为此，Python 定义了可以表示和记录混合数据的组合数据类型。使用组合数据类型表示和记录数据，不仅能使数据表示得更为清晰，还能极大简化开发人员的开发工作，提高开发效率。本章将对 Python 中的组合数据类型进行详细讲解。

5.1 认识组合数据类型

组合数据类型可将多个相同类型或不同类型的数据组织为一个整体。根据数据组织方式的不同，Python 的组合数据类型可分成 3 类，分别是序列类型、集合类型和映射类型。关于它们的介绍如下。

1. 序列类型

序列类型来源于数学概念中的数列。数列是按一定顺序排成一列的一组数，每个数称为数列的项，每项不是在其他项之前，就是在其他项之后。具有 n 个项的数列$\{a_n\}$的定义如下：

$$\{a_n\} = a_0, a_1, a_2, \cdots, a_{n-1}$$

需要注意的是，数列的索引是从 0 开始依次递增的。通过索引 i 可以访问数列中的第 i+1 个项。例如通过索引 1 可以获取数列$\{a_n\}$中的第 2 个项 a_1。

序列类型在数列的基础上进行了扩展，Python 中的序列类型支持正向索引和反向索引，如图 5-1 所示。

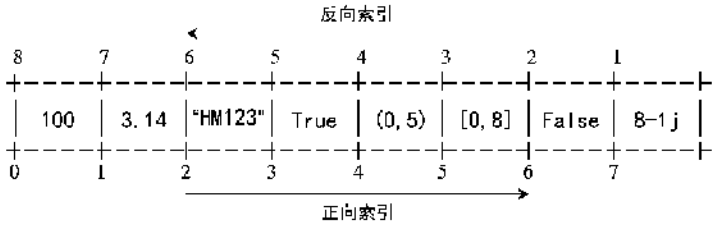

图5-1　序列的索引

正向索引从左向右依次递增，从左数的第 1 个元素的索引为 0，第 2 个元素的索引为 1，以此类推；反向索引从右向左依次递减，从右数的第 1 个元素的索引为-1，第 2 个元素的索引为-2，以此类推。这种双向索引可以方便用户快速访问序列中任意位置的元素，尤其是开头位置和结尾位置的元素。

Python 中常用的序列类型主要有 3 种，分别为字符串（str）、列表（list）和元组（tuple），其中字符串和元组是不可变的序列类型，一旦创建以后其内部的元素无法被修改，而列表是可变的序列类型。第 4 章已经详细介绍了字符串，本章将在 5.2 节和 5.3 节分别介绍列表和元组。

2. 集合类型

数学中的集合是指由具有某种特定性质的对象汇总而成的集体，其中组成集合的对象称为集合的元素。例如，成年人集合的每一个元素都是已满 18 周岁的人。集合中的元素具有以下 3 个特性。

- 确定性：对于给定的一个集合，其每个元素要么属于该集合，要么不属于该集合，不允许有模棱两可的情况出现。
- 互异性：集合中的元素必须是唯一的，不允许重复。如果尝试向集合中添加已经存在的元素，则会自动忽略添加元素的操作，不对集合进行任何修改，也就是说既不会添加元素，也不会覆盖已经存在的元素。
- 无序性：集合中的元素没有顺序，若多个集合中的元素仅顺序不同，那么这些集合本质上是同一集合。

Python 中的集合类型与数学中的集合概念一致，也具备以上 3 个特性。它用于存储一组元素，元素必须唯一，元素可以是无序的。另外，Python 要求放入集合中的元素必须是不可变数据类型的，例如整型、浮点型、复数类型、布尔类型、字符串类型和元组类型，列表、字典及集合类型都属于可变的数据类型，这些类型的数据都不能存放到集合中。

3. 映射类型

映射类型以键值对的形式存储元素，键值对中的键与值之间存在映射关系。在数学中，设 A、B 是两个非空集合，若按某个确定的对应法则 f，可使集合 A 中的任意一个元素 x 在集合 B 中都有唯一确定的对应元素 y，则称 f 为从集合 A 到集合 B 的一个映射。映射关系示例如图 5-2 所示。

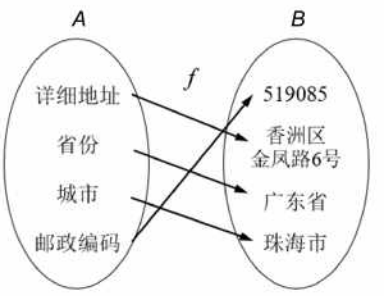

图5-2 映射关系示例

字典是 Python 中唯一的映射类型，字典的键必须遵循以下两个原则。

（1）每个键只能对应一个值，不允许重复出现。

（2）字典中的键是不可变数据类型的。

本章将在 5.6 节对 Python 中的字典进行详细介绍。

5.2 列表

列表是 Python 中最灵活的序列类型之一，它没有长度的限制，可以保存任意个任意类型的元素，用户可以自由地对列表中的元素进行各种操作，包括访问、添加、排序、删除等。本节将针对列表的相关操作进行讲解。

5.2.1 创建列表

Python 中有多种创建列表的方式，既可以直接使用中括号“[]”创建列表，也可以使用 list()函数创建列表，具体介绍如下。

1. 使用中括号“[]”创建列表

使用中括号“[]”可以直接创建一个空列表，示例代码如下：

```
list_one = []                                    # 创建空列表，没有任何元素
```

在中括号中可以添加一个或多个元素，多个元素使用英文逗号分隔。列表的元素不仅可以为整型、浮点型等数字类型的数据，而且可以是字符串、列表、元组、字典等组合数据类型的数据，还可以是用户自定义的数据类型的数据。示例代码如下：

```
list_two = ['p', 'y', 't', 'h', 'o', 'n']          # 列表中的元素类型均为字符串
list_three = [1, 'a', '&', 2.3]                    # 列表中的元素类型不同
list_four = [1, 'a', '&', 2.3, list_three]         # 列表中嵌套了另一个列表
```

2. 使用 list()函数创建列表

list()函数需要接收一个可迭代对象，并根据可迭代对象创建一个列表。如果该函数没有接收任何可迭代对象，那么会创建一个空列表。示例代码如下：

```
# 创建空列表，结果为[]
li_one = list()
# 根据字符串创建列表，结果为['p', 'y', 't', 'h', 'o', 'n']
li_two = list('python')
# 根据另一个列表创建列表，结果为[1, 'python']
li_three = list([1, 'python'])
```

多学一招：可迭代对象

支持通过 for-in-语句迭代获取其内部元素的对象就是可迭代对象。我们已经学习了字符串和列表这两种数据类型，这两种数据类型的数据都是可迭代对象。同时，后续将学习的集合、字典的数据也是可迭代对象。

若希望知道一个目标对象是否可迭代，则可以使用 isinstance()函数，示例代码如下：

```
# 从 collections.abc 模块中导入 Iterable 类
from collections.abc import Iterable
ls = [3, 4, 5]
print(isinstance(ls, Iterable))
```

运行代码，结果如下所示：

```
True
```

由以上运行结果可知，列表 ls 是一个可迭代对象。

5.2.2　访问列表元素

Python 中既可以通过索引和切片这两种方式访问列表中的元素，也可以通过循环依次访问列表中的每个元素，下面分别介绍这 3 种访问列表元素的方式。

1. 通过索引访问列表元素

索引就像图书的目录，阅读时我们可以借助目录快速定位指定内容。类似地，每个列表元素都有唯一的索引来标识其在列表中的位置，通过索引可以快速定位列表中的元素，而无须遍历整个列表。通过索引访问列表元素的语法格式如下：

```
list[n]
```

以上语法格式中，list 表示列表或列表类型的变量，n 表示索引，其作用是访问列表 list 中索引为 n 的元素。

Python 中的序列类型支持双向索引，其中正向索引从 0 开始，自左至右递增；反向索引从–1 开始，自右向左递减。可分别通过正向索引和反向索引访问列表中的同一个元素，示例代码如下：

```
list_demo01 = ["Java", "C#", "Python", "PHP"]
print(list_demo01[1])                    # 通过正向索引访问列表元素
print(list_demo01[-3])                   # 通过反向索引访问列表元素
```

运行代码，结果如下所示：

```
C#
C#
```

2. 通过切片访问列表元素

切片是一种操作序列类型的方法，用于截取序列中的部分元素，从而得到一个新的子序列。下面以列表为例介绍如何通过切片访问列表的元素。通过切片访问列表元素的语法格式如下：

```
list[m:n:step]
```

以上语法格式表示按步长 step 获取列表 list 中索引 m～n–1 对应的元素，不包括索引 n 对应的元素。step 默认为 1；m 和 n 可以省略，若 m 省略，表示切片从列表头部开始，若 n 省略，表示切片到列表末尾结束。示例代码如下：

```
li_one = ['p', 'y', 't', 'h', 'o', 'n']
print(li_one[1:4:2])      # 按步长 2 获取 li_one 中索引 1～3 对应的元素
```

```
print(li_one[2:])          # 获取 li_one 中索引 2 到末尾对应的元素
print(li_one[:3])          # 获取 li_one 中索引 0～2 对应的元素
print(li_one[:])           # 获取 li_one 中的所有元素
```

运行代码，结果如下所示：

```
['y', 'h']
['t', 'h', 'o', 'n']
['p', 'y', 't']
['p', 'y', 't', 'h', 'o', 'n']
```

3. 通过循环依次访问列表元素

列表是一个可迭代对象，这意味着用户可以对列表进行迭代处理，逐个访问列表中的元素。Python 中使用 for 循环来遍历列表中的元素，示例代码如下：

```
li_one = ['p', 'y', 't', 'h', 'o', 'n']
for li in li_one:
    print(li, end=' ')
```

运行代码，结果如下所示：

```
p y t h o n
```

5.2.3 添加列表元素

添加列表元素是一种常见的列表操作，Python 中提供了 append()、extend()和 insert()这几个方法实现添加列表元素的操作，以满足向列表中动态添加元素的需求。关于这些方法的具体介绍如下。

1. append()方法

append()方法用于在列表末尾添加一个新元素，该方法需要接收一个参数，即要添加的新元素，新元素的类型是任意的，可以是整型、字符串、列表、元组、字典等。append()方法会将参数作为整体添加到列表的末尾，而不会将参数内部的多个元素逐个添加。示例代码如下：

```
list_one = [1, 2, 3, 4]
list_one.append(5)                    # 在列表末尾添加元素 5
print(list_one)
list_one.append(['论语', '诗经'])      # 继续在列表末尾添加另一个列表
print(list_one)
```

运行代码，结果如下所示：

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, ['论语', '诗经']]
```

2. extend()方法

extend()方法用于将另一个序列中的每个元素逐个添加到列表的末尾，实现对原列表的扩展。示例代码如下：

```
list_str = ['a', 'b', 'c']
list_num = [1, 2, 3]
list_str.extend(list_num)      # 将 list_num 的所有元素添加到 list_str 的末尾
print(list_num)
print(list_str)
```

运行代码，结果如下所示：

```
[1, 2, 3]
['a', 'b', 'c', 1, 2, 3]
```

3. insert()方法

insert()方法用于在列表中的指定位置插入一个新元素，插入位置之后的元素会依次向后移动。如果新元素是列表、元组、字典，则会将其视为一个整体插入列表的指定位置。需要注意的是，如果指定位置超出了列表的长度，则 insert()方法会将元素插入列表的开头或者结尾。示例代码如下：

```
names = ['小明', '小红', '小兰']
# 在列表 names 中索引为 2 的位置插入新元素'小白'
names.insert(2, '小白')
print(names)
# 在列表 names 中索引为 1 的位置插入新元素('张三', '李四')
names.insert(1, ('张三', '李四'))
print(names)
names.insert(10, '王五')      # 在列表 names 的末尾插入新元素'王五'
print(names)
names.insert(-10, '王五')     # 在列表 names 的开头插入新元素'王五'
print(names)
```

运行代码，结果如下所示：

```
['小明', '小红', '小白', '小兰']
['小明', ('张三', '李四'), '小红', '小白', '小兰']
['小明', ('张三', '李四'), '小红', '小白', '小兰', '王五']
['王五', '小明', ('张三', '李四'), '小红', '小白', '小兰', '王五']
```

5.2.4　列表元素排序

列表元素排序指的是对列表中的元素按照一定规则进行重新排列的操作。Python 中列表元素排序常用的方法有 sort()、sorted()、reverse()，下面分别介绍这些方法。

1. sort()方法

sort()方法用于按特定顺序对列表中的所有元素进行排序，该方法的语法格式如下：

```
sort(key=None, reverse=False)
```

以上语法格式中各参数的含义如下。

- key：用于指定排序规则，该参数的取值可以是列表支持的函数或者匿名函数。例如，key=len 表示按元素的长度进行排序。该参数的默认值为 None，表示将按照元素的值进行排序。如果元素的类型是数字类型，则会按照数字的大小进行排序；如果元素的类型是字符串，则会按照字母顺序进行排序。
- reverse：用于控制列表元素排序的方式，该参数值可以取 True 或者 False，其中 True 表示降序排列，False（默认值）表示升序排列。

使用 sort()方法对列表元素排序后，排序后的元素会覆盖列表原有的元素，不产生新列表，示例代码如下：

```
li_one = [6, 2, 5, 3]
li_two = [7, 3, 5, 4]
li_three = ['python', 'java', 'php']
li_one.sort()                          # 采用升序的方式对列表中的元素进行排序
li_two.sort(reverse=True)              # 采用降序的方式对列表中的元素进行排序
li_three.sort(key=len)                 # 按照元素的长度对列表中的字符串进行排序
print(li_one)
```

```
print(li_two)
print(li_three)
```

以上代码首先创建了 3 个列表，即 li_one、li_two 和 li_three，然后使用 sort()方法分别对这 3 个列表进行排序，其中列表 li_one 采用默认的升序方式重新排列其内部的元素；列表 li_two 采用降序的方式重新排列其内部的元素；列表 li_three 按照元素的长度，并且采用升序的方式重新排列其内部的元素。

运行代码，结果如下所示：

```
[2, 3, 5, 6]
[7, 5, 4, 3]
['php', 'java', 'python']
```

2. sorted()函数

sorted()函数用于按升序的方式排列列表元素，该函数的返回值是升序排列后的新列表，排序操作不会对原列表产生影响。示例代码如下：

```
li_one = [4, 3, 2, 1]
li_two = sorted(li_one)        # 采用升序的方式对列表 li_one 的元素进行排序
print(li_one)
print(li_two)
```

运行代码，结果如下所示：

```
[4, 3, 2, 1]
[1, 2, 3, 4]
```

从上述结果可以看出，原列表中的元素没有任何变化，而新列表中的元素已经按照从小到大的顺序进行排列。

3. reverse()方法

reverse()方法用于逆置列表，即把原列表中的元素从右至左依次排列。示例代码如下：

```
li_one = ['a', 'b', 'c', 'd']
li_one.reverse()
print(li_one)
```

运行代码，结果如下所示：

```
['d', 'c', 'b', 'a']
```

5.2.5 删除列表元素

删除列表元素是一种常见的列表操作，它是指从一个列表中删除特定的元素或者所有的元素，使得列表中不再包含某元素或者任何元素。Python 中常用的删除列表元素的方式有 del 语句、remove()方法、pop()方法和 clear()方法，具体介绍如下。

1. del 语句

del 语句用于删除列表中指定位置的元素，示例代码如下：

```
names = ['小明', '小红', '小兰']
del names[0]                                    # 从列表中删除索引为 0 的元素
print(names)
```

运行代码，结果如下所示：

```
['小红', '小兰']
```

此外，del 语句也可以删除整个列表，示例代码如下：

```
del names          # 删除列表 names
print(names)
```

以上程序运行后会出现错误，错误信息具体如下：

```
Traceback (most recent call last):
  File "E:\FastPrograms3\Chapter05\code02.py", line 5, in <module>
    print(names)
          ^^^^^
NameError: name 'names' is not defined
```

从最后一条错误信息可以看出，程序没有找到名为 names 的变量。之所以出现这种情况，是因为 Python 底层已经从命名空间中删除了变量 names，解除了变量 names 与列表的关联关系，并且触发了垃圾回收机制，将不再使用的列表当作垃圾进行回收，以后无法再访问变量 names 了。

2. remove()方法

remove()方法用于删除列表中的某个元素，若列表中有多个匹配的元素，remove()只删除匹配到的第一个元素，示例代码如下：

```
chars = ['h', 'e', 'l', 'l', 'e']
chars.remove('e')                                  # 删除匹配到的第一个'e'
print(chars)
```

运行代码，结果如下所示：

```
['h', 'l', 'l', 'e']
```

3. pop()方法

pop()方法用于删除列表中指定位置的元素，并返回被删除的元素。若未指定具体元素，则该方法会删除列表中的最后一个元素。示例代码如下：

```
numbers = [1, 2, 3, 4, 5]
print(numbers.pop())        # 删除列表中的最后一个元素
print(numbers.pop(1))       # 删除列表中索引为 1 的元素
print(numbers)
```

运行代码，结果如下所示：

```
5
2
[1, 3, 4]
```

注意，如果指定位置超出了列表的长度，则会导致程序出错。

4. clear()方法

clear()方法用于清空列表中的所有元素，将列表变为空列表，示例代码如下：

```
names = [1, 2, 3]
names.clear()               # 清空列表中的所有元素
print(names)
```

运行代码，结果如下所示：

```
[]
```

从上述结果可以看出，names 列表中没有任何元素。

5.2.6　列表推导式

列表推导式是符合 Python 语法规则的复合表达式，用于以简洁的方式根据已有的列表构建满足特定需求的列表。基本列表推导式的语法格式如下：

```
[表达式 for 临时变量 in 目标对象]
```

以上语法格式由表达式及其后面的 for 语句组成，其中 for 语句用于遍历目标对象，并将每次遍历到的元素赋给临时变量，目标对象必须是一个可迭代对象，例如列表、字符串、元组等；表达式用于在每次循环中对临时变量进行处理或者计算，它可以是任何有效的、包含运算符的表达式，也可以是变量或者常量。

当程序执行列表推导式时，首先会创建一个空列表，然后执行 for 语句遍历目标对象的元素，在每次循环中将遍历到的元素赋给临时变量，之后执行表达式，将表达式的处理或计算结果添加到列表中，最后返回生成的新列表。

例如，使用列表推导式创建一个列表，该列表中的每个元素的值是另一个列表中每个元素的值的平方，示例代码如下：

```
ls = [1, 2, 3, 4, 5, 6, 7, 8]
ls = [data * data for data in ls]
print(ls)
```

运行代码，结果如下所示：

```
[1, 4, 9, 16, 25, 36, 49, 64]
```

除了上面介绍的语法格式外，还可以结合 if、if-else 语句或 for 循环嵌套构成比较复杂的列表推导式，进而更灵活地生成列表。下面对一些复杂的列表推导式进行介绍。

1. 带 if 语句的列表推导式

在基本列表推导式的 for 语句之后添加一个 if 语句，就组成了带 if 语句的列表推导式，其语法格式如下：

```
[表达式 for 临时变量 in 目标对象 if 判断条件]
```

以上列表推导式的执行过程是，先遍历目标对象，然后将遍历到的元素赋给临时变量，若临时变量的值符合判断条件，则按表达式对其进行处理或计算，并将处理或计算后的结果添加到新列表中。

例如，通过带 if 语句的列表推导式构建一个新列表，新列表中只保留列表 ls 中大于 4 的元素，具体代码如下：

```
new_ls = [temp for temp in ls if temp > 4]
print(new_ls)
```

运行代码，结果如下所示：

```
[9, 16, 25, 36, 49, 64]
```

2. 带 if-else 语句的列表推导式

在基本列表推导式的 for 语句之前添加一个 if-else 语句，就组成了带 if-else 语句的列表推导式，其语法格式如下：

```
[表达式 1 if 判断条件 else 表达式 2 for 临时变量 in 目标对象]
```

以上列表推导式的执行过程是，先遍历目标对象，然后将遍历到的元素赋给临时变量，若临时变量的值符合判断条件，则按表达式 1 对其进行处理或计算，否则按表达式 2 对其进行处理或计算，并将处理或计算后的结果添加到新列表中。

例如，通过带 if-else 语句的列表推导式构建一个新列表，新列表中保留列表 ls 中值为偶数的元素，以及值为奇数的元素加 1 的结果，具体代码如下：

```
new_ls = [temp if temp % 2 == 0 else temp + 1 for temp in ls]
print(new_ls)
```

运行代码，结果如下所示：

```
[2, 4, 10, 16, 26, 36, 50, 64]
```

3. 带 for 循环嵌套的列表推导式

在基本列表推导式的 for 语句之后添加一个 for 语句，就组成了带 for 循环嵌套的列表推导式，其语法格式如下：

```
[表达式 for 临时变量 1 in 目标对象 1 for 临时变量 2 in 目标对象 2]
```

以上语法格式中的 for 语句按从左至右的顺序分别是外层循环和内层循环。利用上述列表推导式可以根据两个目标对象快速生成一个新的列表。例如，将列表 ls_one 和列表 ls_two 中相同位置的元素相加后的结果作为列表 ls_three 的元素，示例代码如下：

```
ls_one = [1, 2, 3]
ls_two = [3, 4, 5]
ls_three = [x + y for x in ls_one for y in ls_two]
print(ls_three)
```

运行代码，结果如下所示：

```
[4, 5, 6, 5, 6, 7, 6, 7, 8]
```

5.3　元组

元组的表现形式为一组包含在小括号“()”中、由英文逗号分隔的元素，元素的个数、类型不受限制。使用小括号可以直接创建元组，示例代码如下：

```
t1 = ()                          # 创建空元组，结果为()
t2 = (1,)                        # 创建包含单个元素的元组，结果为(1,)
t3 = (1, 2, 3)                   # 创建包含多个元素的元组，结果为(1, 2, 3)
t4 = (1, 'c', ('e',2))           # 元组嵌套，结果为(1, 'c', ('e', 2))
```

需要注意的是，若元组中只有一个元素，则该元素之后的英文逗号是不能省略的。

使用内置函数 tuple()也可以创建元组。当该函数接收的参数为空时会创建一个空元组，当该函数接收的参数为可迭代对象时会创建非空元组，示例代码如下：

```
t1 = tuple()                     # 创建空元组，结果为()
t2 = tuple([1, 2, 3])            # 利用列表创建元组，结果为(1, 2, 3)
# 利用字符串创建元组，结果为('p', 'y', 't', 'h', 'o', 'n')
t3 = tuple('python')
t4 = tuple(range(5))             # 利用可迭代对象创建元组，结果为(0, 1, 2, 3, 4)
```

Python 中支持通过索引与切片的方式访问元组的元素，也支持通过循环依次访问元组的元素，示例代码如下：

```
print(t2[1])                     # 以索引的方式访问元组元素
print(t3[2:5])                   # 以切片的方式访问元组元素
for data in t3:                  # 通过循环遍历元组的元素
    print(data,end=' ')
```

运行代码，结果如下所示：

```
2
('t', 'h', 'o')
p y t h o n
```

需要注意的是，元组是不可变的数据类型，元组创建以后其内部的元素不能被修改，因此元组不支持添加元素、删除元素和元素排序等一些会修改元素的操作。

5.4 实训案例

5.4.1 成语接龙


成语接龙

成语接龙是中华民族传统的文字游戏。它拥有悠久的历史和广泛的社会基础，是我国文字、文明的一个缩影，是老少皆宜的民间娱乐活动。成语接龙游戏的规则如下。

（1）成语必须由 4 个字组成。

（2）除了第 1 个成语外，其余成语的第一个字，都是上一个成语的最后一个字，例如，“叶公好龙”“龙马精神”“神采飞扬”“扬眉吐气”“气壮山河”。

（3）每轮成语不得有重复的。

现有一组成语，分别是“万事如意”“发愤图强”“笑容满面”“意气风发”“强颜欢笑”，本案例要求编写程序，从“万事如意”这个成语开始，完成其余成语的自动接龙。

5.4.2 中文数字对照表


中文数字对照表

阿拉伯数字因其具有简单易写、方便使用的特点成为最流行的数字书写方式之一，但在使用阿拉伯数字记数时，可以将某些数字不留痕迹地修改成其他数字，例如，将数字“1”修改为数字“7”，将数字“3”修改为数字“8”。为了避免引起麻烦，可以使用中文大写数字壹、贰、叁、肆等替换对应的阿拉伯数字，替换规则如图 5-3 所示。

零	壹	贰	叁	肆	伍	陆	柒	捌	玖
0	1	2	3	4	5	6	7	8	9

图5-3　中文大写数字与阿拉伯数字的替换规则

本案例要求编写程序，实现将输入的阿拉伯数字转换为对应的中文大写数字的功能。

5.5 集合

Python 的集合本身是可变类型，但集合中的元素必须是不可变类型的。与列表和元组相比，集合的特点是元素无序且必须唯一。下面分别介绍创建集合和集合的常见操作。

1. 创建集合

集合的表现形式为一组包含在大括号“{}”中、由英文逗号“,”分隔的元素。使用“{}”可以直接创建集合，示例代码如下：

```
s1 = {1}                                    # 创建包含一个元素的集合
s2 = {1, 'b', (2,5)}                        # 创建包含多个元素的集合
```

使用内置函数 set()也可以创建集合，如果该函数的参数列表为空，则创建的是一个空集合，示例代码如下：

```
s = set()
```

需要注意的是，使用{}不能创建空集合，空集合只能利用 set()函数创建。

若使用 set()函数创建非空集合，需为该函数传入一个可迭代对象，示例代码如下：

```
s1 = set([1, 2, 3])                    # 根据列表创建集合
s2 = set((2, 3, 4))                    # 根据元组创建集合
s3 = set('python')                     # 根据字符串创建集合
s4 = set(range(5))                     # 根据 range()函数返回的结果创建集合
```

2. 集合的常见操作

集合是可变的，集合中的元素可以动态增加或删除。Python 提供了一些内置方法来操作集合，操作集合的常见方法如表 5-1 所示。

表 5-1　操作集合的常见方法

方法	功能说明
add(x)	向集合中添加元素 x，x 已存在时不做处理
remove(x)	删除集合中的元素 x，若 x 不存在则抛出 KeyError 异常
discard(x)	删除集合中的元素 x，若 x 不存在不做处理
pop()	随机返回集合中的一个元素，同时将该元素从集合中删除。若集合为空，抛出 KeyError 异常
clear()	清空集合
copy()	复制集合，返回值为集合
isdisjoint(T)	判断当前集合和集合 T 是否包含相同的元素，如果不包含则返回 True，否则返回 False

使用表 5-1 中的方法操作本节创建的集合，示例代码如下：

```
s1.add('s')                            # 向集合 s1 中添加元素 s
print(s1)
s2.remove(3)                           # 删除集合 s2 中的元素 3
print(s2)
s3.discard('p')                        # 删除集合 s3 中的元素 p
print(s3)
data = s4.pop()                        # 随机返回集合 s4 中的一个元素
print(data)
s3.clear()                             # 清空集合 s3
print(s3)
s5 = s2.copy()                         # 复制集合 s2 并赋给 s5
print(s5)
result = s4.isdisjoint(s2)             # 判断集合 s4 和 s2 是否有相同的元素
print(result)
```

运行代码，结果如下所示：

```
{1, 2, 3, 's'}
{2, 4}
{'h', 'y', 't', 'n', 'o'}
0
set()
{2, 4}
False
```

3. 集合推导式

集合推导式与列表推导式的格式相似，区别在于集合推导式外侧为大括号“{}”，其语法格式如下所示：

```
{表达式 for 临时变量 in 目标对象 if 判断条件}
```

以上语法格式中遍历的目标对象可以是集合或其他可迭代对象。通过集合推导式在列表 ls 的基础上生成只包含值为偶数的元素的新集合，示例代码如下：

```
ls = [1, 2, 3, 4, 5, 6, 7, 8]
s = {data for data in ls if data%2==0}
print(s)
```

运行代码，结果如下所示：

```
{8, 2, 4, 6}
```

集合推导式的更多语法格式可通过列表推导式类比，此处不赘述。

5.6 字典

“字典”这个词相信读者都不会陌生，在碰到不认识的字时，可以使用字典的汉字部首表查找对应的汉字。Python 中的字典也具备类似的功能，它以键值对的形式组织数据，可以通过“键”快速查找其对应的“值”。本节将对 Python 中的字典进行介绍。

5.6.1 创建字典

在 Python 中，字典的表现形式为一组包含在大括号“{}”中的键值对，每个键值对为一个字典元素。不同键值对使用英文逗号“,”分隔，键和值之间使用“:”分隔，语法格式如下：

```
{键 1:值 1, 键 2:值 2, …, 键 N:值 N}
```

字典的值可以是任意类型的数据，键可以是任意不可变类型的对象，如字符串、元组等。字典像集合一样使用“{}”包裹元素，字典中的元素无序，且键必须唯一。

使用“{}”可以直接创建字典，示例代码如下：

```
d1 = {}                                          # 创建空字典
d2 = {'A': '123', 'B': '135', 'C': '680'}        # 创建字典，键的类型都是字符串
d3 = {'A': 123, 12: 'python'}                    # 创建字典，键的类型不同
```

使用内置函数 dict()也可以创建字典，示例代码如下：

```
d4 = dict()                                      # 创建空字典
d5 = dict({'A': '123', 'B': '135'})              # 创建非空字典
```

5.6.2 字典的访问

Python 中可以使用字典的键访问其对应的值，具体的语法格式如下：

```
字典变量[键]
```

通过以上语法格式访问 5.6.1 小节创建的字典中的元素，示例代码如下：

```
print(d2['A'])
print(d3[12])
```

运行代码，结果如下所示：

```
123
python
```

Python 提供了内置方法 get()，该方法可以根据键从字典中获取对应的值，若指定的键不存在则返回指定的默认值。get()方法的语法格式如下：

```
d.get(key[, default])
```

在上述语法格式中，key 表示要获取值的键；default 是可选的参数，表示键不存在时返回的默认值。如果指定的键存在于字典中，则返回与该键关联的值；如果指定的键不存在，则返回指定的默认值；如果没有指定默认值，则返回 None。

示例代码如下：

```
print(d2.get('A'))
print(d3.get(12))
```

运行代码，结果如下所示：

```
123
python
```

除了利用键访问值的 get()外，Python 还提供了分别用于访问字典中所有键、值和元素的内置方法 keys()、values()和 items()，这些方法的示例代码如下：

```
dic = {'name': '小明', 'age':23, 'height':185}
print(dic.keys())                                    # 利用 keys()获取所有键
print(dic.values())                                  # 利用 values()获取所有值
print(dic.items())                                   # 利用 items()获取所有元素
```

运行代码，结果如下所示：

```
dict_keys(['name', 'age', 'height'])
dict_values(['小明', 23, 185])
dict_items([('name', '小明'), ('age', 23), ('height', 185)])
```

内置方法 keys()、values()、items()的返回值都是可迭代对象，利用循环可以遍历这些对象。以遍历 keys()的返回值为例，示例代码如下：

```
for key in dic.keys():
    print(key)
```

运行代码，结果如下所示：

```
name
age
height
```

5.6.3　字典元素的添加和修改

字典支持通过为指定的键赋值或使用 update()方法添加和修改元素，下面分别介绍如何添加和修改字典元素。

1. 字典元素的添加

当字典中不存在某个键时，利用如下语法格式可在字典中添加一个元素：

```
字典变量[键] = 值
```

例如，通过上述语法格式在字典中添加一个元素，具体代码如下：

```
add_dict = {'name': '小明', 'age':23, 'height':185}
add_dict['sco'] = 98                                 # 添加元素
print(add_dict)
```

以上代码通过为指定的键赋值实现了字典元素的添加。

运行代码，结果如下所示：

```
{'name': '小明', 'age': 23, 'height': 185, 'sco': 98}
```

当字典中不存在某个键时，使用 update()方法同样可以实现元素的添加。update()方法

不仅能给字典添加一个元素，还可以一次性给字典添加多个元素。update()方法的语法格式如下：

```
update([other])
```

以上语法格式中，参数 other 是可选的，表示要添加的元素，它可以是一个字典，例如{'b': 3, 'c': 4}，也可以是一个由键值对元组组成的可迭代对象，例如[('b', 3), ('c', 4)]，还可以是形如“键 1=值 1，键 2=值 2，…”的值，例如 b=3, c=4。

示例代码如下：

```
add_dict.update(weight=98)                    # 添加一个元素
print(add_dict)
add_dict.update(stu_id=1, address='北京')      # 添加多个元素
print(add_dict)
```

运行代码，结果如下所示：

```
{'name': '小明', 'age': 23, 'height': 185, 'sco': 98, 'weight': 98}
{'name': '小明', 'age': 23, 'height': 185, 'sco': 98, 'weight': 98, 'stu_id': 1,
'address': '北京'}
```

2. 字典元素的修改

修改字典元素的本质是通过键获取值，并重新对元素进行赋值。修改元素的操作与添加元素的操作基本相同，示例代码如下：

```
modify_dict = {'stu1': '小明', 'stu2': '小刚', 'stu3': '小兰'}
modify_dict.update(stu2='小强')        # 使用 update()方法修改元素
modify_dict['stu3'] = '小婷'           # 通过指定键修改元素
print(modify_dict)
```

以上代码通过 update()方法将 stu2 的值修改为“小强”，通过指定键将 stu3 的值修改为“小婷”。

运行代码，结果如下所示：

```
{'stu1': '小明', 'stu2': '小强', 'stu3': '小婷'}
```

5.6.4 字典元素的删除

Python 支持通过 pop()、popitem()和 clear()方法删除字典中的元素，下面分别介绍这几个方法。

1. pop()

pop()方法可根据指定键删除字典中的指定元素，若删除成功，该方法返回被删除的元素的值。示例代码如下：

```
per_info = {'001': '张三', '002': '李四',
            '003': '王五', '004': '赵六'}
print(per_info.pop('001'))          # 使用 pop()删除指定键为 001 的元素
print(per_info)
```

运行代码，结果如下所示：

```
张三
{'002': '李四', '003': '王五', '004': '赵六'}
```

由以上输出结果可知，元素“'001': '张三'”被成功删除。

2. popitem()

使用 popitem()方法可以随机删除字典中的一个元素。实际上 popitem()之所以能随机删除元素，是因为字典元素本身无序。若删除成功，popitem()方法会返回被删除的元素，示例代码如下：

```
per_info = {'001': '张三', '002': '李四',
            '003': '王五', '004': '赵六'}
print(per_info.popitem())                    # 使用 popitem()方法随机删除元素
print(per_info)
```

运行代码，结果如下所示：

```
('004', '赵六')
{'001': '张三', '002': '李四', '003': '王五'}
```

3. clear()方法

clear()方法用于清空字典中的元素，示例代码如下：

```
per_info = {'001': '张三', '002': '李四',
            '003': '王五', '004': '赵六'}
per_info.clear()                        # 使用 clear()方法清空字典中的元素
print(per_info)
```

运行代码，结果如下所示：

```
{}
```

由以上运行结果可知，字典 per_info 被清空，成为空字典。

5.6.5　字典推导式

使用字典推导式是一种快速构建字典的方法，它可以根据一定的规则从一个可迭代对象中创建字典。字典推导式与列表推导式的格式类似，区别在于字典推导式使用大括号包裹，且大括号内部的表达式需要包含键和值两个部分，其语法格式如下：

```
{键的表达式:值的表达式 for 临时变量 in 目标对象}
```

上述语法格式中总共包含两个部分，分别是键值对表达式和 for 语句，其中键值对表达式用于生成字典的键值对，键的表达式和值的表达式既可以是任何有效的包含运算符的表达式，也可以是变量或者常量；for 语句用于遍历目标对象中的元素，并将元素赋给临时变量。注意，临时变量的个数和目录对象的结构是匹配的，比如目标对象为字典时，由于字典里面的元素是一个键值对，它包括键和值两个部分，所以可以使用两个临时变量分别存储键和值。

当程序执行字典推导式时，首先会创建一个空字典，然后执行 for 语句遍历目标对象的元素，在每次循环中将遍历到的元素赋给临时变量，接着根据给定的表达式计算或处理键和值，将新的键值对添加到字典中，最后返回生成的新字典。

利用字典推导式可快速交换字典中的键和值，示例代码如下：

```
old_dict = {'name': '小明', 'age':23, 'height':185}
new_dict = {value:key for key, value in old_dict.items()}
print(new_dict)
```

运行代码，结果如下所示：

```
{'小明': 'name', 23: 'age', 185: 'height'}
```

字典推导式也支持 if 语句和 for 循环嵌套，此处不再讲解，有兴趣的读者可自行学习。

5.7 实训案例

5.7.1 词频统计


词频统计

词频统计是一种统计文本中单词出现次数的方法，它可以帮助我们了解文本中的哪些单词出现得最频繁，或者在某个特定的上下文中，某些单词的使用情况。本案例要求将用户输入的一段英文文本转变为小写形式后统计并输出每个单词的出现次数。

5.7.2 手机通讯录


手机通讯录

手机通讯录用于记录联系人的联系方式和基本信息，人们在手机通讯录中通过姓名可以方便地查看对应联系人的手机号、电子邮箱、联系地址等信息，也可以自由编辑联系人信息，包括新增、修改、删除联系人等。通讯录通常包含 6 个功能，每个功能都对应一个序号，用户选择序号执行相应的操作。手机通讯录中各功能的介绍如下。

（1）添加联系人：用户根据提示分别输入联系人的姓名、手机号、电子邮箱和联系地址，输入完成后输出保存成功的提示信息。注意，若输入的用户信息为空，会提示用户输入正确的信息。

（2）查看通讯录：如果通讯录不为空，按照固定的格式展示通讯录中每个联系人的信息。如果通讯录为空，则会直接提示通讯录无信息。

（3）删除联系人：如果通讯录不为空，则用户根据提示输入联系人的姓名，若该联系人在通讯录中，则删除该联系人，提示删除成功，否则提示该联系人不在通讯录中。如果通讯录为空，则会直接提示通讯录无信息。

（4）修改联系人：如果通讯录不为空，用户根据提示输入要修改的联系人的姓名，之后按照提示分别输入该联系人的新姓名、新手机号、新电子邮箱、新联系地址，并输出修改成功的提示信息。如果通讯录为空，则会直接提示通讯录无信息。

（5）查找联系人：如果通讯录不为空，用户根据提示输入联系人的姓名，若该联系人在通讯录中，则输出该联系人的所有信息，否则输出该联系人不在通讯录中的提示信息。如果通讯录为空，则会直接提示通讯录无信息。

（6）退出：退出手机通讯录。如果用户不主动选择退出，那么可以一直使用手机通讯录。

本案例要求编写程序，实现具有如上功能的手机通讯录。

5.8 组合数据类型使用运算符

2.6 节介绍的针对数字类型的运算符，对组合数据类型同样适用，但考虑到组合数据类型与数字类型之间存在差异，本节将说明“+”“*”“in”“not in”这几个运算符在组合数据类型中的使用规则。

1.“+”运算符

Python 的字符串、列表和元组支持“+”运算符，与数字类型不同，组合数据类型的变量相加时不进行数值的累加，而是进行变量的拼接。示例代码如下：

```
str_one = "hello "
str_two = "world"
print(str_one + str_two)
list_one = [1, 2, 3]
list_two = [4, 5, 6]
print(list_one + list_two)
tuple_one = (1, 2, 3)
tuple_two = (3, 4, 5)
print(tuple_one + tuple_two)
```

运行代码，结果如下所示：

```
hello world
[1, 2, 3, 4, 5, 6]
(1, 2, 3, 3, 4, 5)
```

2. “*”运算符

“*”运算符的运算规则与“+”运算符的类似，字符串、列表和元组可以和整数进行乘法运算，运算之后产生的新变量为原变量重复整数次的结果。以列表类型的变量为例，示例代码如下：

```
list_one = [1, 2, 3]
print(list_one * 3)
```

运行代码，结果如下所示：

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

3. “in”“not in”运算符

“in”“not in”运算符称为成员运算符，用于判断某个元素是否属于某个变量。Python 的字符串、列表、元组、集合和字典都支持成员运算符。以列表为例，示例代码如下：

```
list_one = [1, 2, 3]
print(1 in list_one)
print(1 not in list_one)
```

运行代码，结果如下所示：

```
True
False
```

5.9　本章小结

本章首先带领读者简单认识了 Python 中的组合数据类型；然后分别介绍了 Python 中常用的组合数据类型——列表、元组、集合、字典的创建和使用，并结合实训案例帮助读者巩固这些数据类型知识；最后介绍了组合数据类型使用运算符的规则。通过对本章的学习，读者能掌握并熟练运用 Python 中的组合数据类型。

5.10　习题

一、填空题

1. 使用内置的_____函数可创建一个列表。
2. Python 中列表的元素可通过索引或_____两种方式访问。
3. 使用内置的_____函数可创建一个元组。

4. 字典元素由_____和_____组成。
5. 字典中的键具有_____性。

二、判断题

1. 列表只能存储同一类型的数据。（　　）
2. 元组支持添加、删除和修改元素的操作。（　　）
3. 列表的索引从 1 开始递增。（　　）
4. 字典中的键唯一。（　　）
5. 字典中的元素可通过索引的方式访问。（　　）

三、选择题

1. 下列方法中，可以对列表元素排序的是（　　）。

A. sort()　　B. reverse()　　C. max()　　D. list()

2. 阅读下面的程序：

```
li_one = [2, 1, 5, 6]
print(sorted(li_one[:2]))
```

运行程序，输出结果是（　　）。

A. [1 ,2]　　B. [2 ,1]　　C. [1 ,2 ,5 ,6]　　D. [6 ,5 ,2 ,1]

3. 下列方法中，默认删除列表中最后一个元素的是（　　）。

A. del　　B. remove()　　C. pop()　　D. extend()

4. 阅读下面的程序：

```
lan_info = {'01': 'Python', '02': 'Java', '03': 'PHP'}
lan_info.update({'03': 'C++'})
print(lan_info)
```

运行程序，输出结果是（　　）。

A. {'01': 'Python', '02': 'Java', '03': 'PHP'}　　B. {'01': 'Python', '02': 'Java', '03': 'C++'}

C. {'03': 'C++','01': 'Python', '02': 'Java'}　　D. {'01': 'Python', '02': 'Java'}

5. 阅读下面的程序：

```
set_01 = {'a', 'c', 'b', 'a'}
set_01.add('d')
print(len(set_01))
```

运行程序，输出结果是（　　）。

A. 5　　B. 3　　C. 4　　D. 2

四、简答题

1. 列举 Python 中常用的组合数据类型，并简单说明它们的异同。
2. 简单介绍删除字典元素的几种方式。

五、编程题

1. 已知列表 li_num1 = [4, 5, 2, 7]和 li_num2 = [3, 6]，请将这两个列表合并为一个列表，并将合并后的列表中的元素按降序排列。
2. 已知元组 tu_num1 = ('p', 'y', 't', ['o', 'n'])，请向元组的最后一个列表中添加新元素“h”。
3. 已知字符串 str_demo= 'skdaskerkjsalkj'，请统计该字符串中各字母出现的次数。
4. 已知列表 li_one = [1,2,1,2,3,5,4,3,5,7,4,7,8]，编写程序实现删除列表 li_one 中重复数据的功能。