第 3 章

Python 函数及组合数据类型

本章导读

本章介绍函数和组合数据类型。函数部分包括函数的定义与调用、函数的参数传递、函数递归三个方面的内容。组合数据类型部分包括序列类型、集合类型、映射类型等。

本章重点介绍函数的定义与调用，以及列表、集合和字典类型等。

学习目标

- 能描述函数的作用。
- 会定义函数并进行调用。
- 能描述函数递归的使用场景及条件。
- 能描述各种组合数据类型的特点及作用。
- 会熟练使用列表类型的常用操作。
- 会熟练使用字典类型的常用操作。

各例题知识要点

例 3.1 定义函数求阶乘(函数的定义与调用)

例 3.2 定义函数输出固定字符串(无参数函数)

例 3.3 定义函数求三个数之和(可选参数)

例 3.4 定义函数输出学生信息(可变参数)

例 3.5 计算 1 到 n 的总和及平均值(返回多个值)

例 3.6 求阶乘(局部变量的作用范围)

例 3.7 求阶乘(使用 global 保留字)

例 3.8 追加成绩(组合数据类型为全局变量)

例 3.9 追加成绩(组合数据类型为局部变量)

例 3.10 计算 x 的 y 次方的值(lambda 函数)

例 3.11 求阶乘(函数递归)

例 3.12 求斐波那契数列(函数递归)

例 3.13 定义颜色元组(元组的定义与索引)

例 3.14 输入成绩(定义列表并追加元素)

例 3.15 删除不合理成绩(删除列表元素)

例 3.16 输出前三位的成绩(列表排序)

例 3.17 不同类型元素的集合(集合与转换)

例 3.18 去除重复的姓名(集合去重复)

例 3.19 国家首都映射(字典初始化)
例 3.20 根据键取出相应的值(字典元素的引用)
例 3.21 字典分析与元素删除(字典常用函数和方法)
例 3.22 查询城市的区号(字典的使用)
例 3.23 统计词频(jieba 库的使用)

3.1 函　　数

函数是一段具有特定功能的、可重用的代码。函数是功能的抽象,一般来说,每个函数表达特定的功能。

使用函数的主要目的有两个:

(1) 降低编程难度。

(2) 代码复用。

如果将数学公式的计算过程定义为函数,则只要调用函数即可使用该计算过程。例如,数学中的组合数计算,即从 n 个元素中不重复地选取 m 个元素,如计算公式 3.1 所示。

$$C_n^m = \frac{A_n^m}{m!} = \frac{n!}{m!\ (n-m)!} \tag{3.1}$$

从公式可以看出,计算组合数,需要多次计算阶乘。如果要编程来计算组合数的话,可以把求阶乘的代码抽象出来,定义为一个函数,然后在计算组合数的过程中多次使用求阶乘的函数。使用函数既可以降低编程难度,又能多次复用代码,提高代码效率。

函数需要先定义后调用。本节将围绕函数的定义与调用、函数的参数传递、函数的递归三方面的内容展开介绍。

3.1.1 函数的定义与调用

1. 函数的定义

函数的定义形式如下:

```
def <函数名> (<0 个或多个形式参数>):
    <函数体>
    return <返回值>
```

在函数的定义中,函数名是函数的标识。函数名后面是用圆括号括起来的形式参数列表,形式参数列表可以包含 0 个或多个形式参数。形式参数用来接收外部传递给函数的数据,如果函数的运行不需要提供参数,则可以不要形式参数,此时圆括号中是空的。但是,不管圆括号内的参数是否为空,圆括号本身都不能省略。

函数体是实现函数功能的,需要若干条语句。注意函数体要缩进。

如果函数需要给外界带回运算结果的话,可以使用 return 语句。return 后可以列出 0 个或多个数据。return 后面没有数据时,return 的作用是返回到主调语句,不带回任何

数据。当return后面有一个数据时,将该数据作为结果带回。当return后面有多个数据时,将把多个数据带回。

【例3.1】 定义求阶乘的函数,计算n!。

【分析】 先定义一个函数fact(),求阶乘,然后提供实际参数调用函数,并输出函数返回值。

程序代码如下:

```
def fact(n):
    s = 1
    for i in range(1, n+1):
        s *= i
        return s
m = fact(8)
print(m)
```

在本例中,变量n是形式参数,计算得到的阶乘值存放在变量s中,通过return返回s的值。

定义函数需要注意以下几点:

- 定义函数时,参数是输入、函数体是处理、return返回的结果是输出,即遵循IPO流程。
- 定义函数时,圆括号中的参数是形式参数;调用函数时,圆括号中的参数是实际参数。
- 函数调用后得到返回值。
- 定义函数后,必须调用,才能真正运行函数中的形式代码。实际参数替换定义中的形式参数。
- 函数定义后,如果不调用,则不会被执行。

2. 函数的调用

在例3.1中,函数定义里的变量n是形式参数。在函数定义后,通过语句m=fact(8)调用函数,这里的8为实际参数。

整个调用过程中数据的传递情况如图3-1所示。在函数调用过程中,实际参数8传递给形式参数n,经过计算得到结果s=40320,通过return将s的值40320返回,赋值给m并输出。

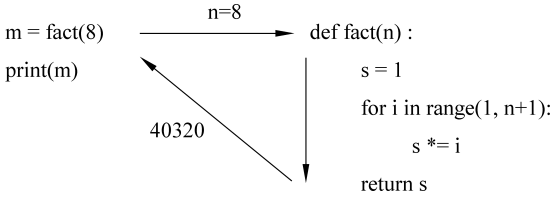


图3-1 函数的调用过程图

3.1.2　函数的参数传递

参数传递形式有无参数传递、可选参数传递、可变参数传递等几种情况。

1. 无参数传递

无参数传递的函数定义形式如下：

```
def <函数名>() :
    <函数体>
    return <返回值>
```

注意：函数可以有参数，也可以没有参数，但即使没有参数，也必须保留圆括号。

【例 3.2】 定义一个无参数函数，输出“This is a function”。

【分析】 无参数函数即函数名后的括号内不加任何参数，注意括号不能省略。

程序代码如下：

```
def func() :
    print("This is a function")
func()
```

程序说明：这是一个无参数的函数形式，调用时没有传入参数值。

2. 可选参数传递

可选参数就是在定义函数时就指定默认值的参数。可选参数函数的定义形式如下：

```
def <函数名>(<非可选参数>, <可选参数>=<默认值>) :
    <函数体>
    return <返回值>
```

注意：可选参数必须放在非可选参数的后面。

【例 3.3】 定义求三个数之和的函数，包含 3 个参数，其中 2 个是可选参数，默认值分别是 3 和 5。

【分析】 函数参数包含可选参数时，可选参数要放在非可选参数之后。

程序代码如下：

```
def Sum(a,b=3,c=5):
    return a+b+c
print(Sum(8))
print(Sum(8,2))
```

程序运行结果为：

```
16
15
```

程序说明：

(1) 在本例中，函数 Sum 共有 3 个参数，分别是 a、b、c。其中 b 和 c 是可选参数，在定义时给定了默认值，分别是 3 和 5。

(2) 通过 print(Sum(8))调用时，只给了一个参数 8，赋给 a，因此 b 和 c 取默认值。而在通过 print(Sum(8,2))调用时，给了两个参数，则 c 会取默认值 5。

3. 可变参数传递

当定义函数时，如果参数数量不确定，则可以将形式参数指定为可变参数。

可变参数传递函数形式如下：

```
def <函数名>(<参数>, * b) :
    <函数体>
    return <返回值>
```

其中，带星号 * 的参数即为可变参数，可变参数只能出现在参数列表的最后。一个可变参数代表一个元组。有关元组的知识将会在本章后续讲解。

【例 3.4】 定义一个可变参数函数，输出学生的学号、姓名和喜爱的运动项目。

【分析】 每个学生喜爱的运动项目可能有多个，因此，运动项目考虑用可变参数存储。

程序代码如下：

```
def stu ( num , name, * sports):
    print("num:", num, "name:", name , "sports:" , sports)
stu("2020001", "Wang")
stu("2020002", "Li", "running")
stu("2020003", "Zhao", "running", "skating")
```

输出结果为：

```
num: 2020001 name: Wang sports: ( )
num: 2020002 name: Li sports: ('running',)
num: 2020003 name: Zhao sports: ('running', 'skating')
```

程序说明：

(1) 参数 sports 前面加星号，表示 sports 是可变参数。

(2) 调用 stu 函数时，可以不提供值给 sports 参数；也可以为其指定一个参数值，比如“running”；还可以为 sports 提供两个或多个参数值，如“running”和“skating”两个值。

3.1.3 函数的返回值

函数可以返回 0 个或多个结果，传递返回值用 return 保留字。如果不需要返回值，则可以没有 return 语句。return 可以传递 0 个返回值，也可以传递任意多个返回值。

下面举例说明如何利用 return 传递多个返回值。

【例 3.5】 编写函数，计算 1 到 n 的总和及平均值，并返回总和值及平均值。

【分析】 函数有两个返回值，分别是总和值和平均值。使用 return 语句，逗号分隔两个返回值。

程序代码如下：

```
def func(n,m):
    s=0
    for i in range(1,n+1):
        s+=i
    return s,s/m
s,ave=func(10,10)
print(s, ave)
```

运行结果为：

```
55 5.5
```

程序说明：

(1) 在定义 func()函数时，首先接收两个值分别存放到形式参数 n 和 m 中，然后通过 for 循环求得 1 至 n 的和，并存放在变量 s 中，最后函数返回 s 的值及 s 除以 m 的商。程序通过一个 return 返回了两个值。

(2) 在调用 func()函数时，实际参数 10 和 10 分别传递给形式参数 n 和 m。func 函数计算并返回和值 55 及商 5.5。最后，输出带回的总和值及平均值。

3.1.4　局部变量和全局变量

根据变量的作用范围，变量可以分为局部变量和全局变量。

全局变量在整个程序范围内均有效，而局部变量仅在本函数内有效，如图 3-2 所示。

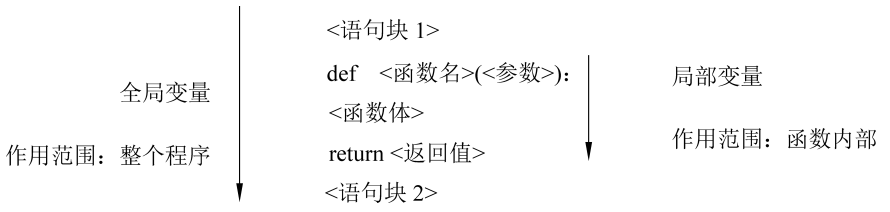


图 3-2　局部变量和全局变量的作用范围

【例 3.6】 编写函数计算 n 的阶乘，并调用该函数求 8!，在函数外和函数内有同名变量 s，在函数外 s 初始化为 10，在函数内用于返回求得的阶乘值。

【分析】 求阶乘的函数，存放结果的变量应初始化为 1。

程序代码如下：

```
n, s = 8, 10                #n 和 s 是全局变量
def fact(n) :               #fact()函数中的 n 和 s 是局部变量
```

```
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
print(fact(n), s)     #此处的 n 和 s 是全局变量
```

运行结果为：

```
40320 10
```

程序说明：

(1) 在函数 fact()内，参数 n 以及用到的 s 均为 fact()函数的局部变量。该函数计算得到 n! 的值，并将结果通过 s 返回。函数运行结束后，fact()函数中的局部变量 n 和 s 将会被释放。

(2) 在 fact()函数外，print()输出参数中出现的 n 和 s 是全局变量。全局变量 n 为 8，全局变量 s 的值始终为 10。

注意：局部变量是函数内部的变量，有可能与全局变量重名，但它们代表不同的存储空间。

函数外部出现的变量是全局变量，函数内部出现的变量是局部变量。这一点类似于现实生活中，两栋楼均存在 201 号房间，虽然两个房间的名称一样，但是，两个 201 房间代表的是不同的实体。而且，局部变量和全局变量的存在周期不同。全局变量所占用的存储空间在程序的整个运行阶段均存在。但是，函数内部的局部变量只有在开始调用函数时才分配存储空间，函数调用结束后，局部变量所占的存储空间就会被释放掉。

1. 局部变量和全局变量是不同变量

如果想在函数内部使用全局变量，可以使用 global 保留字。例 3.7 是一个使用 global 保留字的示例。

【例 3.7】 修改例 3.6，利用保留字 global 修饰变量 s，观察输出的结果 s 有何不同。

【分析】 在函数内使用 global 关键字。

程序代码如下：

```
n, s = 8, 10              #n 和 s 是全局变量
def fact(n) :             #fact()函数中的 n 是局部变量，s 是全局变量
    global s
    for i in range(1, n+1):
        s *= i
    return s
print(fact(n), s)         #n 和 s 是全局变量
```

运行结果为：

```
403200 403200
```

程序说明：在本例中，函数 fact()内部的变量 s 前面增加了保留字 global，表明 s 为全局变量，此处的 s 与函数 fact()外部的 s 为同一个变量。因此，在退出函数之后，函数 fact()内部对 s 的修改依然保留。

2. 若局部变量为组合数据类型且未创建，则该变量等同于全局变量

Python 语言有列表、集合、元组、字典等组合类型。具体在 3.2 节介绍，在此先了解其在作为全局变量和局部变量上的差别。

【例 3.8】 使用列表存储一组成绩数据，向列表中追加一个成绩，追加用函数实现。

【分析】 定义函数向列表中追加一个元素。

程序代码如下：

```
ls = [90,88,69,92]
def func(a):
    ls.append(a)
    return
func(78)
print(ls)
```

运行结果为：

```
[90, 88, 69, 92, 78]
```

程序说明：在本例中，首先创建一个列表 ls，ls 为全局变量。在函数 func()内部没有创建 ls，向列表 ls 中追加元素值 78。这种情况下，函数内部的 ls 就等同于全局变量 ls。调用该函数，即将 78 增加到 ls 中，输出结果表示列表 ls 在函数 func()的内部从四个元素增加为五个元素。

【例 3.9】 改写例 3.8，在函数内部创建列表后追加。

程序代码如下：

```
ls = [90,88,69,92]
def func(a):
    ls=[]
    ls.append(a)
    print("函数内 ls 为:",ls)
    return
func(78)
print("函数外 ls 为:",ls)
```

运行结果为：

```
函数内 ls 为: [78]
函数外 ls 为: [90, 88, 69, 92]
```

程序说明：

(1) 在本例中，在函数 func()内部创建一个列表 ls，且初始化为空列表。这种情况下，函数内部的 ls 为局部变量。

(2) 从程序的运行结果可以看出，第一次在函数内输出的是局部变量 ls，第二次在函数外输出的是全局变量 ls。两个变量虽然名称相同，但是不同的变量，所占的存储空间不同，存储内容也不相同。

简单总结局部变量和全局变量的使用规则如下：

- 对于基本数据类型，无论是否重名，局部变量和全局变量是不同的。
- 如果想要在函数内部使用全局变量，可以使用 global 保留字。
- 对于组合数据类型，只要函数内局部变量没有真正创建，就默认为全局变量。

3.1.5 lambda 函数

lambda 函数是一种匿名函数。定义时需要使用 lambda 保留字，且直接用函数名返回结果。lambda 函数一般用于定义能够在一行内表示的简单函数。

lambda 函数形式如下：

```
<函数名> = lambda <参数>: <表达式>
```

lambda 函数可以等价替换成 def 定义的函数。冒号 : 之前可以有 0 个或多个参数。即上面的语句等价于下面的函数定义。

```
def <函数名>(<参数>) :
    <函数体>
    return <返回值>
```

【例 3.10】 定义 lambda 函数，计算 x 的 y 次方的值。

【分析】 按照 lambda 函数的定义形式，包括两个参数，分别是 x 和 y。

程序代码如下：

```
>>> f = lambda x, y : x ** y
>>> f(2, 3)
```

输出结果为：

```
8
```

程序说明：

(1) 定义函数 f，包括两个参数 x 和 y。函数功能是计算 x 的 y 次方。调用并指定参数分别为 2 和 3，函数返回结果是 8。如果改写成 def 形式，如图 3-3 所示。

(2) lambda 函数主要用于一些特定函数或方法的参数，有一些固定使用方式，建议逐步掌握。一般情况下，建议使用 def 定义的普通函数。注意谨慎使用 lambda 函数。

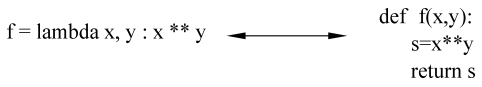


图 3-3　lambda 函数的等价形式

3.1.6　函数递归

递归是数学归纳法思维的编程体现。本节将从递归的定义、递归的实现、递归的调用过程、递归应用举例四个方面进行介绍。

1. 递归的定义

递归即函数定义中调用函数自身的方式。

递归有两个关键特征，即链条和基例。递归过程需要递归链条，同时也需要一个或多个不需要再次递归的基例。

2. 递归的实现

按照递归的两个关键特征，采用函数和分支语句来实现。

因为递归是函数调用自身的方式，因此，递归本身就应该是函数，必须用函数定义方式来描述。函数内部需要两个关键特征，即基例和链条。递归的实现方法是，判断输入参数是否是基例，如果是则直接给出结果，否则按照链条给出对应的调用自身的代码。

【例 3.11】　定义递归函数，求 $n!$。

【分析】　求阶乘的过程可以理解为递归过程。

$$n!=\begin{cases}1, & n=0\\ n(n-1)!, & \text{其他}\end{cases} \tag{3.2}$$

当 n 为 0 时，结果为 1，否则返回 $n-1$ 作为参数调用自身，并返回 n 与 $(n-1)!$ 的乘积。

程序代码如下：

```
def fact(n):
    if n==0:
        return 1
    else:
        return n * fact(n-1)
```

程序说明：当 n 等于 0 时，阶乘值返回 1，否则通过 n 和(n－1)的阶乘的乘积得到结果，因此需要调用函数求(n－1)的阶乘。

有关递归的实现，需要注意以下几点：

- 递归本身是一个函数，需要函数定义方式描述。
- 递归的实现需要函数 ＋ 分支语句。
- 在函数内部，采用分支语句对输入参数进行判断。

3. 递归的调用过程

下面通过调用求阶乘的递归函数，说明递归函数的执行过程。

求 5 的阶乘，即用 5 作为实际参数调用函数 fact()，调用过程以及参数值的传递情况如图 3-4 所示。

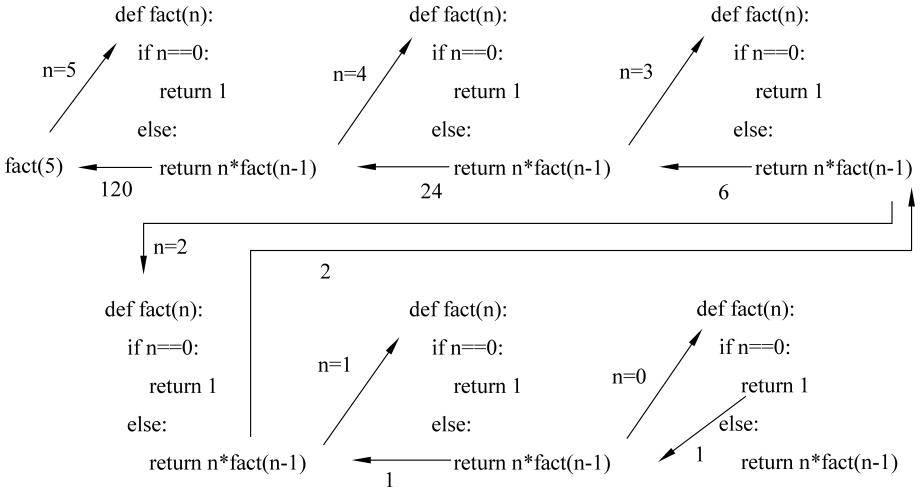


图 3-4　递归函数中数据的传递过程

从 fact(5)开始调用 fact()函数，直到 n＝0 时，到达基例后开始返回，逐级返回上级调用位置，并带回计算结果。

4. 递归应用举例

【例 3.12】 利用递归方法，求斐波那契数列第 10 项。

【分析】 斐波那契数列是一个典型的可以使用递归实现的示例。斐波那契数列为：1,1,2,3,5,8,13,21,34,55,…。数列各项计算公式如下：

$$F(n)=\begin{cases}1, & n=1\\ 1, & n=2\\ F(n-1)+F(n-2), & \text{其他}\end{cases} \tag{3.3}$$

斐波那契数列的基例为：当 n＝1 或 n＝2 时，F(n)的值为 1。其他情况下，使用递归链条，调用自身，得到 F(n－1)与 F(n－2)的和，作为 F(n)的值。

程序代码如下：

```
def fib(n):
    if n==1 or n==2:
        return 1
    else:
        return fib(n-1)+fib(n-2)
print(fib(10))
```

输出结果为：

```
55
```

3.2　组合数据类型

组合数据类型是与基本数据类型相对的，基本数据类型的数据是单一的数，而组合数据类型的数据是一组数。

组合数据类型包括序列类型、集合类型、映射类型等多种类型。

3.2.1　序列类型

序列类型，顾名思义，是具有先后顺序的一组元素。序列类型相当于其他编程语言中的一维数组。序列类型与数组的区别在于，Python 中序列的各元素可以属于不同类型，而数组的各元素必须是属于同一种类型。

序列类型数据通过序号来表示元素的先后顺序，并通过序号来访问每个元素。序列是基类，序列包括字符串类型、元组类型、列表类型等形式的子类，如图 3-5 所示。其中，字符串类型已在前面基本数据类型部分进行了介绍。

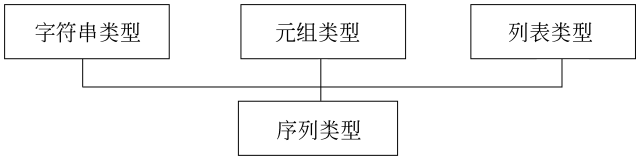


图 3-5　序列类型的子类

1. 序号的定义

序列类型的序号包括正向递增序号和反向递减序号两种。其中，正向递增序号从 0 开始自左向右递增。反向递减序号从－1 开始自右向左递减。如图 3-6 所示，为列表["China", 3.1415,1024,(2,3),[95,83,100]]中各个元素的值以及序号情况。

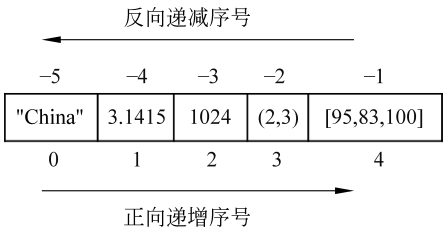


图 3-6　序列类型的两种序号体系

2. 序列类型通用操作符

序列类型的通用操作符如表 3-1 所示，表中也列出了序列常用的函数。

表 3-1　序列类型的通用操作符和函数表

操作符及应用	描　　述
x in s	如果 x 是序列 s 的元素，则返回 True，否则返回 False。例如： >>>(2,3) in ["China", 3.1415, 1024, (2,3), [95,83,100]] True
x not in s	如果 x 是序列 s 的元素，则返回 False，否则返回 True。例如： >>>2 not in ["China", 3.1415, 1024, (2,3), [95,83,100]] True

续表

操作符及应用	描　　述
s+t	连接两个序列 s 和 t。例如： >>>[1,2,3]+[4,5,6] [1, 2, 3, 4, 5, 6]
s * n 或 n * s	将序列 s 复制 n 次(s 为字符串,n 为整数)。例如： >>>[1,2,3] * 2 [1, 2, 3, 1, 2, 3]
s[i]	索引,返回 s 中的第 i+1 个元素,i 是序列的序号。例如： >>>s=["China", 3.1415, 1024, (2,3)] >>>s[1] 3.1415
s[i: j]或 s[i: j: k]	切片,返回序列 s 中第 i+1 到 j 以 k 为步长的元素子序列。例如： >>>s=["China", 3.1415, 1024, (2,3)] >>>s[1:3] [3.1415, 1024]
len(s)	返回序列 s 的长度,即元素个数。例如： >>>s=["China", 3.1415, 1024, (2,3)] >>>len(s) 4
min(s)	返回序列 s 的最小元素,s 中元素应可比较。例如： >>>min([33,77,11,44]) 11
max(s)	返回序列 s 的最大元素,s 中元素应可比较。例如： >>>max([33,77,11,44]) 77
s.index(x)或 s.index(x, i, j)	返回序列 s,序号从 i 到 j 中,第一次出现元素 x 的位置。例如： >>>"banana".index("a") 1
s.count(x)	返回序列 s 中出现 x 的总次数。例如： >>>"banana".count("a") 3

对于常用的操作说明如下：

- in 与 not in 用于判断是否为序列中的元素,返回结果是布尔类型的数据,即 True 或 False。
- +用于连接两个序列,结果是一个更大的序列,序列内容是在第一个序列的所有元素后面追加第二个序列的所有元素。
- * 用于复制操作符,即将序列 s 复制 n 次,结果是一个更大的序列。
- []用于索引,s[i]返回其中序号为 i 的元素。
- s[i: j]或 s[i: j: k]用于切片,即截取序列的元素子序列。s[i: j]返回序列 s 中序

号从 i 到 j－1 的元素子序列，s[i：j：k] 返回序列 s 中序号从 i 到 j－1、以 k 为步长的元素构成的子序列。需要注意的是，序列中第 1 个元素的序号为 0，因此序号为 i 的元素是序列中的第 i＋1 个元素。

- len(s) 返回序列 s 的长度，即元素个数。
- min(s) 返回序列 s 的最小元素，s 中元素应可比较。
- max(s) 返回序列 s 的最大元素，s 中元素应可比较。
- s.index(x)或 s.index(x, i, j) 返回序列 s 从 i 开始到 j 位置第一次出现元素 x 的位置。
- s.count(x) 返回序列 s 中出现 x 的总次数。

3. 元组

元组是一种序列类型，一旦创建就不能被修改。创建元组可以使用小括号或 tuple() 函数，元素间用英文逗号分隔。元组类型的变量赋值时可以使用或不使用小括号。

可以使用如下代码定义名为“color”的元组，分别是黑白和彩色。

```
>>> color="bw","multicolor"
>>> print(color)
```

运行后的输出结果如下：

```
('bw', 'multicolor')
```

给元组赋值时，小括号可以省略，输出时会自动添加小括号。

【例 3.13】 已知有元组 color 如下：

```
color=("bw",multicolor)
```

将其中的 multicolor 定义为包含红、绿、蓝三种颜色的元组，并输出其中的绿色值。

【分析】 color 是一个元组，其中的元素 multicolor 也要定义为元组。

程序代码如下：

```
multicolor=("Red","Green","Blue")
color=("bw",multicolor)
print(color)
print(color[1][1])
```

运行结果为：

```
('bw', ('Red', 'Green', 'Blue'))
Green
```

程序说明：元组 color 的第二个元素为一个元组，访问元组 color 中的元素“Green”，需要首先使用索引为 1，定位到 color 的第 2 个元素；然后，进一步根据绿色所在的索引值

1,取出“Green”。

元组继承序列类型的全部通用操作。元组创建后不能修改,因此元组没有特殊操作。

4. 列表

列表是一种常用的序列类型,创建后可以被随意修改。使用方括号[]或 list()创建,元素间用英文逗号(,)分隔。列表中各元素的类型可以不同,且列表无长度限制。

列表的常用操作和函数如表 3-2 所示。

表 3-2 列表类型的操作符和函数

函数或方法	描 述
ls[i]=x	替换列表 ls 第 i+1 个元素为 x
ls[i:j:k]=lt	用列表 lt 替换 ls 切片后所对应元素子列表
del ls[i]	删除列表 ls 中第 i+1 个元素
del ls[i:j:k]	删除列表 ls 中序号从 i 到 j 以 k 为步长的元素
ls+=lt	更新列表 ls,将列表 lt 元素增加到列表 ls 中
ls*=n	更新列表 ls,其元素重复 n 次

列表的常用方法如表 3-3 所示。

表 3-3 列表的常用方法表

函数或方法	描 述
ls.append(x)	在列表 ls 最后增加一个元素 x
ls.clear()	删除列表 ls 中所有元素
ls.copy()	生成一个新列表,赋值 ls 中所有元素
ls.insert(i,x)	在列表 ls 的第 i+1 个位置增加元素 x
ls.pop(i)	将列表 ls 中第 i+1 个位置元素取出并删除该元素
ls.remove(x)	将列表 ls 中出现的第一个元素 x 删除
ls.reverse()	将列表 ls 中的元素反转
ls.sort()	将列表 ls 中的元素排序,使用 reverse 参数指定升序或降序

【例 3.14】 输入 6 名学生的成绩并存入列表。

【分析】 循环输入,使用 append()方法追加到列表,最后输出。

程序代码如下:

```
score_list=[]
for i in range(6):
    score=eval(input())
    score_list.append(score)
print(score_list)
```

输出结果为：

```
输入：
78
90
89
65
80
98
输出：
[78, 90, 89, 65, 80, 98]
```

程序说明：score_list 为一个列表，每输入一个成绩，添加到 score_list 中。

【例 3.15】 删除不合理的成绩数据。

【分析】 通过取列表元素、列表合并、删除列表元素体会列表的操作方法。

按照索引取列表元素的程序代码如下：

```
score_list=[99, 87,103,89, 68, 83, 98,-32]
for i in score_list:
    if i<0 or i>100:
        score_list.remove(i)
print(score_list)
```

输出结果为：

```
[99, 87, 89, 68, 83, 98]
```

程序说明：删除不合理的成绩，按照百分制成绩的范围，将低于 0 分和大于 100 分的成绩删除，需要在整个成绩列表中进行循环遍历，满足条件的执行 remove()操作进行删除。

如果要进行两个列表的合并，可以使用如下的代码：

```
score_list=[99, 87, 89, 68, 83, 98]
new_list=[58,73,98]
score_list+=new_list
print(score_list)
```

输出结果为：

```
[99, 87, 89, 68, 83, 98, 58, 73, 98]
```

如果要删除第 5 位同学的成绩，可以使用 del，按照索引进行删除，在上述成绩列表的基础上，可以使用如下代码：

```
del score_list[4]            #删除列表中的索引为 4 的元素 83
```

得到新的列表：

```
[99, 87, 89, 68, 98, 58, 73, 98]
```

程序说明：列表元素的索引从 0 开始，两个列表可以使用“+”进行合并，即按照顺序连接成一个列表。

插入一个元素，使用 insert()方法，可以指定插入的具体位置。例如，要在上述成绩列表的基础上，在第 5 个位置上插入成绩 100，可以使用如下代码：

```
score_list.insert(4,100)
```

由于第 5 个位置，对应的索引号为 4，因此 insert()的第一个参数是 4，第二个参数是要插入的值。得到的新成绩列表：

```
[99, 87, 89, 68, 100, 98, 58, 73, 98]
```

【例 3.16】 已知成绩列表为[99，87，89，68，100，98，58，73，98]，按照从高到低，输出前三位的成绩。

程序代码如下：

```
score_list=[99, 87, 89, 68, 100, 98, 58, 73, 98]
score_list.sort(reverse=True)
print(score_list[:3])
```

输出结果为：

```
[100, 99, 98]
```

程序说明：列表的 sort()方法功能是将列表中的元素排序。排序默认是升序排列，如要实现降序排序，需指定参数 reverse 为 True，即形如 score_list.sort(reverse＝True)。输出前三名成绩，即输出排序后列表的前三个数，使用切片得到。

列表用于数据可改变的场景。因为列表元素本身也可以是组合数据类型，因此列表也经常用于存储多维数据。元组用于元素不改变的应用场景。如果要保护数据不被改变，可以将列表数据转换成元组类型。

3.2.2 集合类型

集合类型与数学中的集合概念一致。集合是多个元素的无序组合。集合元素之间无序，每个元素唯一，不存在相同元素。

创建集合类型数据用大括号{}或 set()，元素间用逗号分隔。若建立空集合类型，则必须使用 set()。

【例 3.17】 定义集合的示例。

【分析】 集合元素分别为字符、字符串、元组三种情况，输出集合元素，体会集合元素唯一和无序的特点。

程序代码如下：

```
A=set("python")
print(A)
B={"python","java","c++"}
print(B)
C={'bw',("red","green","blue")}
print(C)
```

输出结果为：

```
{'h', 'y', 't', 'n', 'p', 'o'}
{'java', 'c++', 'python'}
{('red', 'green', 'blue'), 'bw'}
```

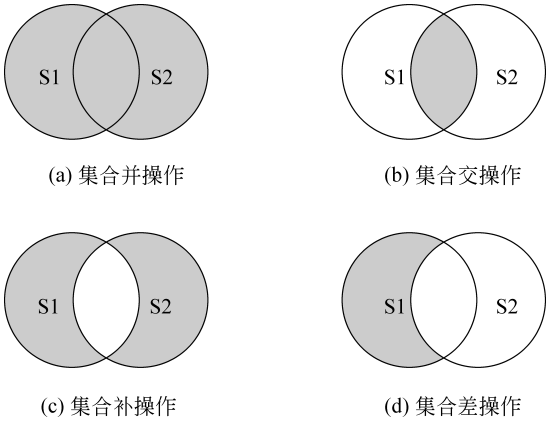
程序说明：

(1) 集合 A 通过函数 set()创建，将字符串转换为集合类型，即集合中包括每个字符，且无序。集合 B 通过大括号{}创建，包含三个元素，同样无序。集合 C 包含两个元素，其中一个元素是字符串，另一个元素是元组。

注意：集合元素不能是列表。

(2) 需要注意：由于集合的元素之间是无序的，因此，在不同的计算机中运行时，每次输出集合内容时，集合元素的顺序都不一样。

集合有四个基本操作，分别是并、差、交、补。与数学上的集合运算含义相同。各个操作结果如图 3-7 所示，其中有底纹的部分为各操作的结果。



(a) 集合并操作　(b) 集合交操作

(c) 集合补操作　(d) 集合差操作

图 3-7　集合操作符的运算结果

集合的 6 个操作符，如表 3-4 所示。

表 3-4 集合的操作符

操作符及应用	描　　述
S1|S2	并,返回一个新集合,包括在集合 S1 和 S2 中的所有元素
S1&. S2	交,返回一个新集合,包括同时在集合 S1 和 S2 中的元素
S1^S2	补,返回一个新集合,包括集合 S1 和 S2 中的非相同元素
S1－S2	差,返回一个新集合,包括在集合 S1 但不在集合 S2 中的元素
S1<＝S2 或 S1<S2	返回 True/False,判断 S1 和 S2 的子集关系
S1>＝S2 或 S1>S2	返回 True/False,判断 S1 和 S2 的包含关系

集合的 4 个增强操作符如表 3-5 所示,分别是并、差、交、补与赋值的结合,不仅执行并、差、交、补的操作,还会把运算结果存放到操作符左侧的集合变量 S 中。

表 3-5 集合的 4 个增强操作符

操作符及应用	描　　述
S|＝T	并,更新集合 S,包括在集合 S 和 T 中的所有元素
S－＝T	差,更新集合 S,包括在集合 S 但不在 T 中的元素
S&.＝T	交,更新集合 S,包括同时在集合 S 和 T 中的元素
S^＝T	补,更新集合 S,包括集合 S 和 T 中的非相同元素

集合的常用操作函数和方法如表 3-6 所示。

表 3-6 集合的常用操作函数和方法

操作函数或方法	描　　述
S.add(x)	如果 x 不在集合 S 中,将 x 增加到 S
S.discard(x)	移除 S 中元素 x,如果 x 不在集合 S 中,不报错
S.remove(x)	移除 S 中元素 x,如果 x 不在集合 S 中,产生 KeyError 异常
S.clear()	移除 S 中所有元素
S.pop()	随机返回 S 的一个元素并在 S 中删除这个元素,更新 S,若 S 为空产生 KeyError 异常
S.copy()	返回集合 S 的一个副本
len(S)	返回集合 S 的元素个数
x in S	判断 S 中元素 x,x 在集合 S 中,返回 True,否则返回 False
x not in S	判断 S 中元素 x,x 不在集合 S 中,返回 True,否则返回 False
set(x)	将其他类型变量 x 转变为集合类型

集合类型的最重要应用场景就是数据去重。

【例 3.18】 集合的去重示例。将列表中的姓名进行去重，重复的姓名只保留一个。

【分析】 转换为集合就可以自动去重。

程序代码如下：

```
name_list=['张飞','赵云','关羽','刘备','张飞','曹操','刘备','诸葛亮']
print(name_list)
name_set=set(name_list)
print(name_set)
```

输出结果为：

```
['张飞', '赵云', '关羽', '刘备', '张飞', '曹操', '刘备', '诸葛亮']
{'曹操', '诸葛亮', '赵云', '张飞', '刘备', '关羽'}
```

程序说明：因为列表中允许有多个相同的元素，所以列表 name_list 中可以包含两个'张飞'和两个'刘备'。在语句 name_set＝set(name_list)中，函数 set()根据 name_list 的元素得到集合 name_set，因为集合中的元素不能重复，因此集合 name_set 中只保留一个'张飞'和一个'刘备'。将其他类型数据转换为集合的过程，就会把相同的元素删除，起到自动去重的目的。

3.2.3　映射类型

映射即键值对，键是对数据索引的扩展，是通过键来索引值的过程。

字典是“映射”的体现：键和值一一对应。字典中的键不允许重复。若存在重复的键，则前面的项自动失效。而且，键必须是不可变类型。字典是键值对的集合，键值对之间无序。

创建字典可以采用大括号{}和函数 dict()，键值对用英文符号冒号“:”表示。下面是利用大括号创建字典的形式：

```
<字典变量> = {<键 1>:<值 1>, … , <键 n>:<值 n>}
```

【例 3.19】 定义字典的示例。键值对分别是三个国家的名称和首都。

【分析】 采用直接赋值方式，每个键值对用逗号分隔。

程序代码如下：

```
d={'China':'Beijing','France':'Paris','US':'Washington'}
print(d)
print(type(d))
```

输出结果为：

```
{'China': 'Beijing', 'France': 'Paris', 'US': 'Washington'}
<class 'dict'>
```