第5章 组合数据类型及其应用

本章学习目标

- 理解序列、映射、数据可变与不可变的概念
- 熟练掌握元组数据类型的使用方法
- 熟练掌握列表数据类型的使用方法
- 熟练掌握字典数据类型的使用方法
- 掌握集合数据类型的使用方法

前4章讨论的都是一些相对简单的问题，程序中存储和处理的数据比较少，而且数据与数据之间都相对独立。对于复杂的问题来说，需要处理的数据量一般都比较大，而且数据之间往往存在关联。举个例子来说，如果要处理一个学校开设的课程情况，就要把表示课程信息的数据，包括课程编号、课程名称、学分等相互关联的数据项组织到一起处理，类似这样的问题就要借助组合数据类型来实现。本章研究组合数据类型及应用。主要介绍元组、列表、字典、集合4种组合数据类型及应用。

5.1 元组及其应用


扫一扫

5.1.1 元组概述

1. 元组的含义

元组是使用一对圆括号括起来的0个或多个数据元素的序列。如()、(1,2,3)、("a","b","c")都是元组。

对元组有以下4点说明。

(1) 元组中的数据元素可以是包括元组在内的任意数据类型。

(2) 元组中元素的个数叫作元组的长度。

(3) 含有 0 个元素的元组(一对空的圆括号)叫作空元组。

(4) 若元组中只有一个元素,书写时元素的后面必须跟一个逗号。

2. 定义元组

定义元组的一般格式如下。

```
变量名 = 元组
```

在 IDLE 命令交互方式下定义元组的程序示例如图 5-1 所示。在该程序中,第 1 个变量定义了空元组 $t1$。第 2 个变量定义了只有一个整数元素 1 的元组 $t2$。该条语句里 1 后面的逗号是必须要有的,否则系统会把 1 当作为普通的整数处理。第 3 个变量定义了含三个整数元素的元组 $t3$,存了某个人的三科考试成绩。第 4 个变量定义了含三个浮点数元素的元组 $t4$,存了三个季度的 GDP 增长率。最后一个变量定义了元组 $t5$,存了一个学生的姓名、年龄和三科考试成绩。它含有三个不同数据类型的元素,自左向右依次是字符串类型、整数类型、元组类型。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> #元组数据类型
>>>
>>> t1=() #空元组
>>> t1
()
>>> t2=(1,) #只有一个元素
>>> t2
(1,)
>>> t3=(75,66,87) #含三个整数元素
>>> t3
(75, 66, 87)
>>> t4=(0.032,0.027,0.036) #含三个小数元素
>>> t4
(0.032, 0.027, 0.036)
>>> t5=("李梅",18,(75,87,62)) #含不同类型的元素
>>> t5
('李梅', 18, (75, 87, 62))
>>>
```

图 5-1 定义元组的程序示例

3. 一维和多维元组

可以通过圆括号的层数来确定元组的维数。只含一层圆括号的元组叫作一维元组。含有两层圆括号的元组叫作二维元组。同样的道理,含有三层圆括号的元组叫作三维元组,含有四层圆括号的元组叫作四维元组……依此类推。

如:

(1,2,3)是一维元组。

(1,2,(3))是二维元组。

(1,2,(3,(4)))是三维元组。

二维及二维以上的元组叫作多维元组。

5.1.2 元组处理

1. 访问元素

和字符串一样,元组中的元素也是使用序号来表示其在序列中的位置,且同样拥有非负

序号和负序号两种格式。

一维元组 $t1=(1,2,3)$中元素的序号如图 5-2 所示。其中，非负序号自左向右是 0～2，负序号自左向右是－3～－1。

对于多维元组来说，每一层圆括号中的元素都采用相同的方法确定序号。二维元组 $t2=(1,(2,3))$中元素的序号如图 5-3 所示。其中，第一层圆括号中的元素 1 和(2,3)，其非负序号自左向右是 0～1，负序号自左向右是－2～－1。第二层圆括号中的元素 2 和 3，非负序号自左向右也是 0～1，负序号自左向右也是－2～－1。

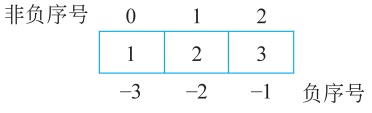

图 5-2　一维元组中元素的序号

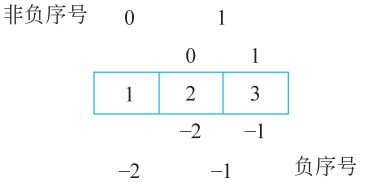

图 5-3　二维元组中元素的序号

和检索字符串中字符的方法一样，可以通过序号来访问元组中的元素。

访问元组中元素的一般格式如下。

```
元组变量名[序号]
```

对于一维元组 $t1=(1,2,3)$来说，$t1[1]$和 $t1[-2]$访问的都是左边第 2 个元素 2。对于二维元组 $t2=(1,(2,3))$来说，$t2[1]$和 $t2[-1]$访问的都是左边第 2 个元素(2,3)，它是一个一维元组。

访问多维元组里元素的格式如下。

```
元组变量名[序号 1][序号 2]…[序号 n]
```

其中，序号 1、序号 2、……、序号 n 分别用来表示该元素所处的第 1 层、第 2 层、……、第 n 层圆括号里的序号。很显然，访问的元素处于第几层圆括号里，就要带几个序号。

对于二维元组 $t2=(1,(2,3))$来说，如果要访问元素 2，就要带 2 个序号，可以使用 $t2[1][0]$、$t2[1][-2]$、$t2[-1][0]$、$t2[-1][-2]$表示。

2. 元组切片

和字符串一样，元组也可以进行切片。元组切片的方法及注意事项与 2.4.2 节中介绍的字符串切片完全一样，这里不再赘述。

字符串切片与元组切片的不同如下。

(1) 字符串切片的结果是字符串类型。

(2) 元组切片的结果是元组类型。

在 IDLE 命令交互方式下对元组进行切片的程序示例如图 5-4 所示。可以对照图 5-4(b)所示的存储情况加以理解。

3. 元组运算

和字符串类似，Python 中用于元组的运算符如表 5-1 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> ##元组切片操作，结果是元组
>>> t=(1,2,3,4,5)
>>> t[1:3];t[1:5:2];t[-3:-1]
(2, 3)
(2, 4)
(3, 4)
>>> t[:3];t[::2];t[::-1]
(1, 2, 3)
(1, 3, 5)
(5, 4, 3, 2, 1)
>>> t[3:1];t[-1:-3]
()
()
>>>
```

(a) 程序代码

t=(1,2,3,4,5)

非负序号	0	1	2	3	4	
	1	2	3	4	5	
	−5	−4	−3	−2	−1	负序号

(b) 存储结构

图 5-4　元组切片程序示例

表 5-1　元组的运算符和表达式

运　算　符	实 施 运 算	表　达　式
+	合并运算	$t1+t2$
*	复制运算	$t*n$
in	包含运算	$t1$ in t

其中，合并运算(+)用来将运算符右边元组中的元素按原有顺序合并到左边元组最后一个元素后面，生成一个新的元组。复制运算(*)用来生成原元组的若干副本，并将其合并为一个新的元组。包含运算(in)用来判断一个对象是否是元组的元素。

在 IDLE 命令交互方式下使用元组运算的程序示例如图 5-5 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> #元组运算
>>> t1=(1,2,3);t2=("a","b","c")
>>> t1+t2
(1, 2, 3, 'a', 'b', 'c')
>>> t1*3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> (1,2) in t1
False
>>> (1,2) in ((1,2),(2,3))
True
>>>
```

图 5-5　元组运算程序示例

在这个程序中，先定义了含 3 个整数 1、2、3 的元组 $t1$ 和含 3 个字符串"a""b""c"的元组 $t2$，之后进行 $t1+t2$ 运算，结果是生成了一个含 6 个元素的元组(1,2,3,'a','b','c')。之后进行 $t1*3$ 运算，结果是将 $t1$ 复制了 3 次，并合并成了一个含 9 个元素的元组(1,2,3,1,2,3,1,2,3)。之后判断(1,2)是否为 $t1$ 的元素，结果是 False。最后判断(1,2)是否为元组((1,2),(2,3))的元素，结果为 True。

4. 常用函数与方法

Python 提供了 4 个用于元组处理的内置函数，如表 5-2 所示。

表 5-2 内置元组处理函数

函　数	功　能
len(*t*)	求元组 *t* 的长度，即元素个数
max(*t*)	求元组 *t* 中元素的最大值
min(*t*)	求元组 *t* 中元素的最小值
tuple(seq)	将 seq 转换为元组类型

在 IDLE 命令交互方式中使用几个函数的程序示例如图 5-6 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> ##元组处理函数
>>> s="abc"
>>> t=(1,2,5.2,4)
>>> t1=("ba","ab","oo")
>>> t2=(1j,2j,3j)
>>> len(t)
4
>>> max(t)
5.2
>>> min(t1)
'ab'
>>> max(t2)
Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    max(t2)
TypeError: '>' not supported between instances of 'complex' and 'complex'
>>> tuple(s)
('a', 'b', 'c')
>>>
```

图 5-6 使用元组处理函数的程序示例

在这个程序中，先定义了一个字符串变量 *s* 和 3 个元组变量 *t*、*t*1、*t*2。*t*、*t*1、*t*2 分别存了含 4 个数值型元素、3 个字符串元素、4 个复数元素。之后执行 len(*t*)，求 *t* 的元素个数，结果是 4。执行 max(*t*)，求 *t* 中元素的最大值，结果是 5.2。执行 min(*t*1)，求 *t*1 中元素的最小值，结果是字符串"*ab*"。执行 max(*t*2)，求 *t*2 中元素的最大值，程序运行出错，因为在 3.1.1 节中介绍过复数类型的数据无法比较大小。最后执行 tuple(*s*)，结果是把字符串"abc"转换成了含三个元素的元组('a'、'b'、'c')。

除了内置函数，还有 2 个重要的元组处理方法，如表 5-3 所示。

表 5-3 2 个元组处理方法

方　法	功　能
t.count(value)	返回元组 *t* 中值为 value 的元素个数
t.index(value)	返回元组 *t* 中值为 value 的元素首次出现的序号

在 IDLE 命令交互方式下使用两个方法的程序示例如图 5-7 所示。

在上面的程序中，首先定义了含 8 个元素的元组 *t*，之后使用 count 函数统计值为 1 的元素个数，结果是 3。使用 index 函数求值为 3 的元素首次出现的序号，结果是 2。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> #元组处理方法
>>>
>>> t=(1, 2, 3, 2, 3, 2, 1, 1)
>>>
>>> t.count(1)
3
>>> t.index(3)
2
>>>
```

图 5-7 使用元组处理方法的程序示例

5. 元组的特性

元组是一种不可变的数据类型。这种不可变性体现在元组一旦被定义，就只能作为一个整体进行赋值，不可以对单个元素进行赋值。

有关元组不可变的程序示例如图 5-8 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> #元组的不可变性
>>>
>>> t1 = ("滑雪","游泳","狩猎")
>>>
>>> #t1可以整体被操作
>>> t1 = ("滑雪","游泳","狩猎","购物")
>>>
>>> #不可以修改元素值
>>> t1[1]="冲浪"
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    t1[1]="冲浪"
TypeError: 'tuple' object does not support item assignment
>>>
```

图 5-8　元组不可变的程序示例

在该程序中，元组 $t1$ 开始存储了某旅游公司开出的 3 个旅游项目("滑雪","游泳","狩猎")，后来公司发展，把项目扩展到了 4 个("滑雪","游泳","狩猎","购物")，所以对元组 $t1$ 作为一个整体重新进行了赋值，这样做是可以的。后面试图把第 2 个元素 $t1[1]$修改为"冲浪"，结果引发了错误。系统提示，元组类型不支持对元素赋值操作。

元组的这种不可变特性可以保护其内部的数据，确保数据的安全。

扫一扫

5.1.3　2 个程序设计实例

1. 输出星期信息

【实例 5-1】　编程实现，输入一个 1 到 7 之间的整数，输出与输入数字对应的“星期一”至“星期日”的信息。

经过分析，“星期一”至“星期日”是 7 个固定不变的字符串，符合元组不可变的特点，可以使用一维元组 weeks 来存储。为了操作方便，可以令它含 8 个元素，第 1 个元素存空串("")，这样，只要令输入的数字 n 作元组 weeks 的序号，取到的恰好就是与该数字相对应的星期几的信息。

一维元组 weeks 的存储结构简图如图 5-9 所示。

0	1	2	3	4	5	6	7
""	"星期一"	"星期二"	"星期三"	"星期四"	"星期五"	"星期六"	"星期日"

图 5-9　一维元组 weeks 的存储结构简图

根据上面的分析，确定了程序的数据结构，需要一个一维元组 weeks 和一个整型变量 n，分别用来存星期几的信息和输入的数据。

程序的算法流程图如图 5-10 所示。

程序的完整代码和运行效果如图 5-11 所示。第 4～5 行是定义元组 weeks 的代码，它实质上是一条语句，可以直接写成多行，不需要使用反斜杠(\)续行。weeks 里含 8 个元素，

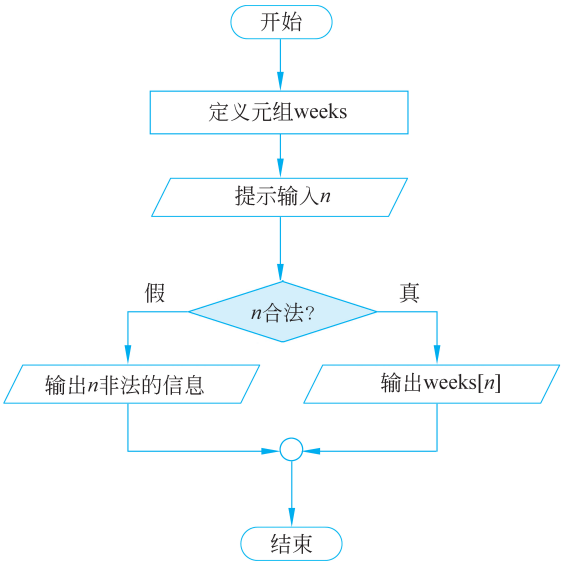

图 5-10 【实例 5-1】的算法流程图

序号为 0 的是空串，尽管它在编程时没有用，但是加入了它，使得后面其他 7 个元素的序号是几，对应的元素就是字符串“星期几”，这就给问题处理带来了很大的方便。第 10～14 行是一个双路分支结构，用来控制输入 1～7 的数时就输出“今天是星期几”的信息，否则就输出“输入的不是 1 至 7 之间的数字!”的信息。图 5-11(b)是单独运行程序两次的运行效果截图。

```
File  Edit  Format  Run  Options  Window  Help
 1 #元组应用一，输出对应数字的星期信息
 2
 3 #使用一维元组存星期一至星期日的信息
 4 weeks=("","星期一","星期二","星期三","星期四",
 5         "星期五","星期六","星期日")
 6
 7 n = eval(input("\n请输入1～7的数字:  "))
 8
 9 #两路分支结构
10 if 1<=n<=7:
11     #输出以n为序号的元素
12     print("\n今天是{}\n".format(weeks[n]))
13 else:
14     print("\n{}不是1～7的数字!\n".format(n))
15
```

(a) 程序代码

```
请输入1～7的数字:  5
今天是星期五
>>>
```

```
请输入1～7的数字:  8
8不是1～7的数字!
>>>
```

(b) 运行效果

图 5-11 【实例 5-1】的程序代码和运行效果

2. 求已过去的天数

【实例 5-2】 编程实现，输入 1980 年 1 月 1 日之后的年、月、日，输出该年度经过了的天数。

经过分析，12 个月的天数是固定不变的整数，符合元组不可变的特点，可以使用一个含两个一维元组的二维元组 month_days 来存储。

二维元组 month_days 的存储结构简图如图 5-12 所示。

在图 5-12 中，第 1 行的序号为 0，存了不是闰年时各月份的天数；第 2 行的序号为 1，存

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	31	28	31	30	31	30	31	31	30	31	30	31
1	0	31	29	31	30	31	30	31	31	30	31	30	31

图 5-12　二维元组 month_days 存储结构简图

了闰年时各月份的天数。为了处理方便，可以令每行含 13 个元素，序号为 0 的元素存 0，其他序号的元素存储该序号对应月份的天数。

程序运行时，根据输入的年是否为闰年决定从哪一行中取值——闰年从第 2 行取值，不是闰年从第 1 行取值。然后遍历 1 至输入的月份减 1，累计每个月的天数之和。遍历结束时加上最后一个月的天数（也就是输入的天数），即可求得结果。

这里可以定义一个变量 leap，用它存是否是闰年的结果。它的值是逻辑值 True 或 False，2.2.3 节介绍过，逻辑值是整数的子集，False 对应的是 0，True 对应的是 1，如果让 leap 做 month_days 的序号，就可以解决根据是否是闰年从二维元组中准确取值的问题。

基于上述分析确定了程序的数据结构，如表 5-4 所示。

表 5-4　【实例 5-2】数据结构信息

变　量　名	数 据 类 型	作　　用
month_days	tuple	存储不是闰年和闰年每个月的天数
year	int	存输入的年
month	int	存输入的月
day	int	存输入的日
leap	bool	存是否是闰年的信息
i	int	控制遍历月份
days	int	累计经历的天数

程序的算法流程图如图 5-13 所示。

程序的完整代码及运行效果如图 5-14 所示。

在该程序中，第 5～8 行是二维元组的定义部分。第 15 行把输入的非整月的天数存到了 days 中。第 18 行是判断闰年的语句，用到了 3.1.2 节中介绍过的判断闰年表达式。第 21～22 行是一个遍历循环结构，用于累计整月的天数。请注意，第 5～8 行二维元组 month_days 的定义写成了多行，且进行了对齐处理；第 11～13 行输入语句里也进行了对齐处理，这都是好的编程习惯，因为这样的代码结构更清晰，更容易阅读。第 22 行用到了 5.1.2 节介绍的使用双序号访问二维元组元素的方法。

图 5-14(b)给出了两次运行程序，分别输入 1980 年 5 月 1 日和 2021 年 5 月 1 日的运行效果，因为前者是闰年，后者不是闰年，所以结果差了一天。

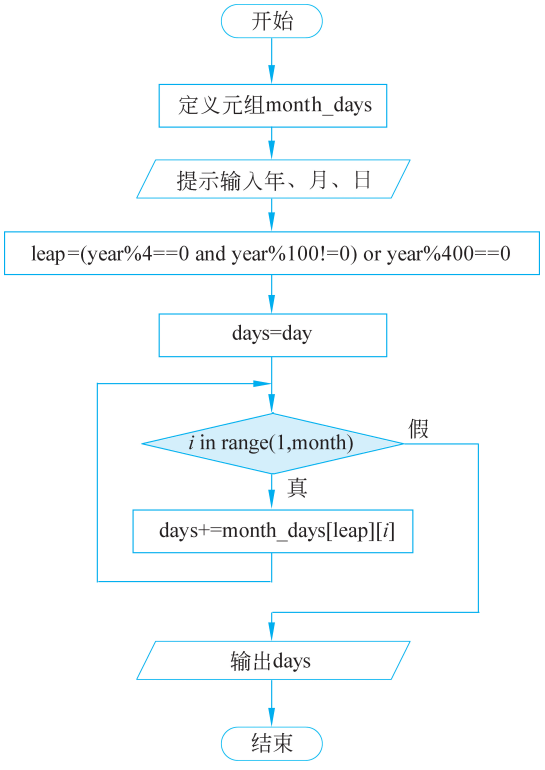

图 5-13 【实例 5-2】的算法流程图

```
File  Edit  Format  Run  Options  Window  Help
#元组应用二，输出对应日期过去的天数

#使用二维元组存每个月份的天数
#第一行对应不闰年，第二行对应闰年
month_days=(
    (0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
    (0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
    )

#输入年、月、日
year  = eval(input("\n请输入年:  "))
month = eval(input("\n请输入月:  "))
day   = eval(input("\n请输入日:  "))

days=day #把非整月的天数存到days中

#判闰年
leap=(year%4==0 and year%100!=0)or year%400==0

#遍历循环结构，统计天数
for i in range(1,month):
    days+=month_days[leap][i]

#输出结果
print("\n已经过去了{}天\n".format(days))
```

(a) 程序代码

```
请输入年:  1980
请输入月:  5
请输入日:  1
已经过去了122天
>>>
```

```
请输入年:  2021
请输入月:  5
请输入日:  1
已经过去了121天
>>>
```

(b) 运行效果

图 5-14 【实例 5-2】的程序代码及运行效果

扫一扫

5.2 列表及其应用

5.2.1 列表概述

1. 列表的含义

列表是使用一对方括号括起来的 0 个或多个数据元素的序列。如[]、[1,2,3]、["a","b","c"]都是列表。

对列表有以下 3 点说明。

(1) 列表中的数据元素可以是包括列表在内的任意数据类型。

(2) 列表中元素的个数叫作列表的长度。

(3) 含有 0 个元素的列表(一对空的方括号)叫作空列表。

2. 定义列表

定义列表的一般格式如下。

```
变量名 = 列表
```

在 IDLE 命令交互方式下定义列表的程序示例如图 5-15 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> #列表数据类型
>>>
>>> l1=[] #空列表
>>> l1
[]
>>> l2=[1] #含1个元素
>>> l2
[1]
>>> l3=[75,66,87] #含三个整数元素
>>> l3
[75, 66, 87]
>>> l4=[0.032,0.027,0.036] #含三个小数元素
>>> l4
[0.032, 0.027, 0.036]
>>> l5=["Python",3.5,[75,87,62]] #含三个不同类型元素
>>> l5
['Python', 3.5, [75, 87, 62]]
>>>
```

图 5-15 定义列表程序示例

在该程序中,第 1 个变量定义了空列表 *l*1。第 2 个变量定义了只有一个整数元素的列表 *l*2。请注意,和元组不同,当列表只有一个元素时,元素后面不需要写逗号。第 3 个变量定义了含 3 个整数元素的列表 *l*3,存了 3 科考试成绩。第 4 个变量定义了含 3 个浮点数元素的列表 *l*4,存了 3 个季度的 GDP 增长率。最后一个变量定义了列表 *l*5,存了某学生选修的一门课程信息,包括课程名称、学分和 3 次考试成绩,3 个元素自左向右分别是字符串类型、浮点类型和列表类型。

3. 一维和多维列表

可以通过方括号的层数来确定列表的维数。只含一层方括号的列表叫作一维列表。含