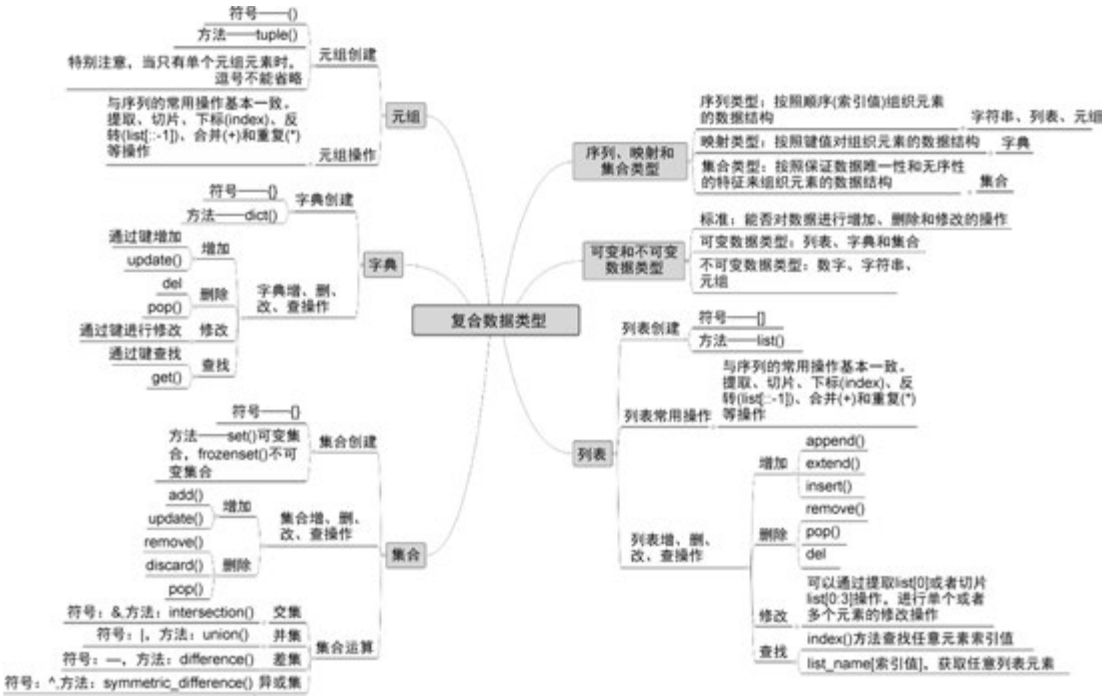
第5章 复合数据类型

思想引领　　视频讲解

学习目标

- 理解数据类型分类。
- 掌握序列类型的基本操作方法。
- 熟练掌握列表类型、元组类型的操作方法。
- 熟练掌握字典类型的组成方式和操作方法。
- 理解集合类型的特征，熟悉集合类型的运算方式。



在数据量庞大且复杂的信息时代，数字或字符串这些基本数据类型还远远满足不了实际数据处理的需求，因此，Python 提供了复合数据类型。复合数据类型主要用于在不同的场景下，为复杂的数据提供不同的组织处理方式和存储方式。这样不仅减少了程序开发人员的工作量，还大大提高了程序的运行效率。本章将对列表、元组、字典和集合这 4 类复合数据类型进行详细介绍。

5.1 数据类型分类

5.1.1 序列、映射和集合类型

根据数据组织方式的不同，可以将 Python 中的字符串、列表、元组、字典和集合数据类型分为 3 类：序列类型、映射类型和集合类型，如图 5-1 所示。

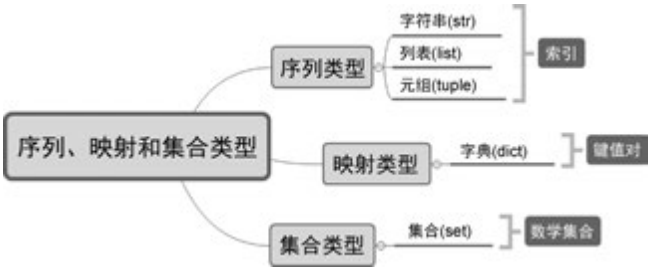


图 5-1 序列、映射和集合类型

1. 序列类型

序列类型指将元素按照固定索引值有序地组织在一起的数据结构，可以通过索引值查找指定元素。所以字符串、列表和元组这 3 种数据类型有很多类似的操作方法，如提取和切片操作等。

2. 映射类型

映射类型指将元素按照键值对的方式组织在一起的数据结构，可以通过键值对中的键查找对应的值。字典的元素就是由键值对构成的，所以增、删、改、查操作与其他数据类型区别较大，学习时需要注意，避免犯错。

3. 集合类型

集合类型指将元素按照互异的且无序的方式组织在一起的数据结构，可以进行类似于数学集合的运算，如交集、并集等。

5.1.2 可变和不可变数据类型

按照能否对数据进行增、删、改操作，可以将数字、字符串、列表、元组、字典和集合分为可变和不可变数据类型，如图 5-2 所示。

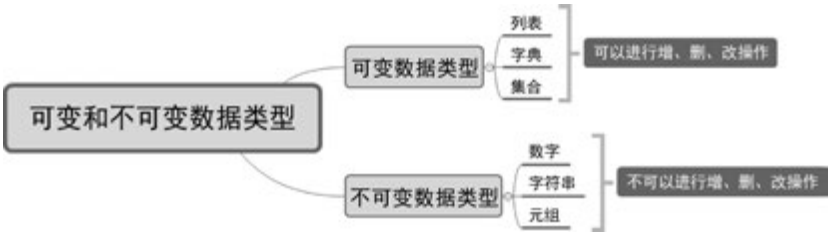


图 5-2 可变和不可变数据类型

可变数据类型：可以对值进行增加、删除和修改操作的数据类型。从存储角度来讲，当

改变其值时，存储空间的地址不会发生改变。属于可变数据类型的有列表、字典和集合。

不可变数据类型：不可以对值进行增加、删除和修改操作的数据类型。从存储角度来讲，当改变其值时，存储空间的地址会发生改变。属于不可变数据类型的有数字、字符串和元组。

简单地理解可变和不可变数据类型，前者不仅有访问、查看的权限，还有可编辑的权限；而后者有只读的权限，不能修改。

5.2 列表

列表是 Python 中最为常用的一种可变数据类型，列表元素和长度都是可以变化的，开发环境中内置有相应的增加、删除和修改方法；列表中可以存储任何其他数据类型；而且列表属于序列类型，所以列表元素是有序的，每一个列表元素都有对应的索引值。

5.2.1 列表的创建

(1) 通过一对方括号([])可以创建列表，方括号内部的元素使用英文逗号进行分隔，示例如下：

```
list1 = [ ]
list2 = [1, 2, 3, 4, 5]
list3 = [1, "python", 7, [3, 2, 8]]
```

list1 中没有列表元素，即空列表。从 list2 和 list3 中可以看出，列表除了可以存储同一种类型的数据，还可以同时存储多种不同类型的数据。

(2) 通过 list()函数也可以创建列表，通常是将元组或字符串转换为列表类型，代码如例 5-1 所示。

【例 5-1】 list()函数创建列表的代码示例。

```
str1 = "python"
list_str = list(str1)
print(list_str)

tuple1 = (2, 4, "python", [1, 2, 3])
list_tuple = list(tuple1)
print(list_tuple)
```

输出结果如下：

```
['p', 'y', 't', 'h', 'o', 'n']
[2, 4, 'python', [1, 2, 3]]
```

用 list()函数将字符串转换为列表时，字符串的每个字符会单独成为一个列表元素。将元组转换为列表时，从形式上看，相当于将元组的圆括号替换成了方括号，其他的不变；从内容上看，就是将所有的元组元素转换为列表元素，从不可变数据类型转为了可变数据类型。

5.2.2 列表的操作

1. 提取和切片操作

列表属于序列类型，所以可以进行单个列表元素的提取操作和多个列表元素的切片操作。

提取操作的语法规范与字符串类型一样，通过正、负索引值都可以进行提取，而且索引值不能越界，代码如例 5-2 所示。

【例 5-2】 列表的提取操作的代码示例。

```
list_test = [1, "python", 7, [3, 2, 8]]
# 正索引提取元素
print(list_test[0])
print(list_test[1])
# 负索引提取元素
print(list_test[-1])
# 索引越界，会产生错误
print(list_test[10])
```

输出结果如下：

```
1
python
[3, 2, 8]
IndexError: list index out of range
```

切片操作的语法规范与字符串相同，分为切片起始位置(start)、切片终止位置(end)和步长(step)，且用冒号分隔。这三个位置的数字都可以省略，省略起始位置时，默认从第一个字符开始切片；省略终止位置时，默认从起始位置开始切片，直到切完为止；省略步长时，默认为 1。另外，越界时不会报错，起始位置越界默认返回空列表，终止位置越界默认切完为止，代码如例 5-3 所示。

【例 5-3】 列表的切片操作的代码示例。

```
list3 = [1, "python", 7, [3, 2, 8]]
print(list3[0:3:1])              # 完整的切片操作
print(list3[0:3])                # 一般步长可以省略，默认为 1
print(list3[:3])                 # 省略起始位置
print(list3[0:])                 # 省略终止位置
print(list3[:])                  # 全部省略
print(list3[9:])                 # 起始位置越界
print(list3[1:10])               # 终止位置越界
```

输出结果如下：

```
[1, 'python', 7]
[1, 'python', 7]
[1, 'python', 7]
[1, 'python', 7, [3, 2, 8]]
[1, 'python', 7, [3, 2, 8]]
```

```
[ ]
['python', 7, [3, 2, 8]]
```

2. 运算符操作

列表可以进行加法(+)的合并操作、乘法(*)的重复操作,还可以运用成员运算符(in或 not in)判断元素是否存在,代码如例 5-4 所示。

【例 5-4】 列表的基本运算符操作的代码示例。

```
print([1,2,3] + [4,5,6])    # 合并两个列表
print([1,2,3] * 3)          # 重复列表 n 次
print(4 in [1,2,3])         # in 的规则是元素在列表中则返回 True,不在则返回 False;not in
                            # 运算符规则与之相反
print(3 in [1,2,3])
```

输出结果如下:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
False
True
```

3. 列表的增、删、改、查操作

列表有增加、删除、修改和查找的操作方法,接下来分别介绍每种操作方法。

(1) 增加操作。列表的增加操作的方法有 append()、extend()和 insert() 3 种,代码如例 5-5 所示。

【例 5-5】 列表的增加操作代码示例。

```
>>> student = ["小赵", "小钱", "小孙"]
>>> new_stu = ['小李', "小周", "小吴"]

# 第一,可以使用 append()方法一次向末尾增加一个列表元素
>>> student.append('小郑')
>>> student
['小赵', '小钱', '小孙', '小郑']

# 第二,可以使用 extend()方法一次向末尾增加多个列表元素
>>> student.extend(new_stu)
>>> student
['小赵', '小钱', '小孙', '小郑', '小李', '小周', '小吴']

# 第三,可以使用 insert()方法向指定位置增加一个列表元素
>>> student.insert(1, '小王')
>>> student
['小赵', '小王', '小钱', '小孙', '小郑', '小李', '小周', '小吴']
```

(2) 删除操作。列表的删除操作的方法有 del 关键字法、remove()和 pop()方法,代码

如例 5-6 所示。

【例 5-6】 列表的删除操作的代码示例。

```
# 第一,可以使用 remove()方法,根据需要删除的元素的值来进行删除
>>> student = ['小赵', '小王', '小钱', '小孙', '小郑', '小李', '小周', '小吴']
>>> student.remove('小王')          # 等效于 student.remove(student[1])
>>> student
['小赵', '小钱', '小孙', '小郑', '小李', '小周', '小吴']

# 第二,可以使用 pop()方法,根据需要删除的元素的索引值来进行删除
>>> student.pop(0)                  # (默认值 - 1)有返回值
'小赵'
>>> student
['小钱', '小孙', '小郑', '小李', '小周', '小吴']

# 第三,可以使用 del 关键字,根据单个元素提取操作或者多个元素的切片操作,删除单个或者同时
# 删除多个列表元素
>>> del student[0]                  # 可删除某一个
>>> student
[ '小孙', '小郑', '小李', '小周', '小吴']
>>> del student[0:2]                # 可删除多个
>>> student
['小李', '小周', '小吴']
```

(3) 修改操作。列表可以通过索引值来进行相应的修改操作,代码如例 5-7 所示。

【例 5-7】 列表的修改操作的代码示例。

```
>>> student = ["小赵", "小钱", "小孙"]
>>> student[0] = "小陈"              # 修改某一个
>>> student[1:3] = ['小冯', '小蒋']  # 修改多个
>>> student
['小陈', '小冯', '小蒋']
```

(4) 查找操作。列表可以通过 index()方法查找索引值,代码如例 5-8 所示。

【例 5-8】 列表的查找操作的代码示例。

```
>>> student = ["小赵", "小钱", "小孙"]
>>> student.index("小钱")
1
```

Python 中常用的列表操作函数和方法分别如表 5-1 和表 5-2 所示。

表 5-1 Python 中常用的列表操作函数

序　号	函数及描述
1	len(list) 返回列表元素个数

续表

序　号	函数及描述
2	max(list) 返回列表元素最大值
3	min(list) 返回列表元素最小值
4	sorted(list) 返回排序后的列表，默认为升序

表 5-2　Python 中常用的列表操作方法

序　号	方　法
1	list.append(obj) 在列表末尾添加新的元素
2	list.count(obj) 统计某个元素在列表中出现的次数
3	list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值(用新列表扩展原来的列表)
4	list.index(obj) 从列表中找出某个值第一个匹配项的索引位置
5	list.insert(index,obj) 将对象插入列表
6	list.pop([index=-1]) 移除列表中的一个元素(默认为最后一个元素)，并且返回该元素的值
7	list.remove(obj) 移除列表中某个值的第一个匹配项
8	list.reverse() 反向列表中的元素
9	list.sort(key=None,reverse=False) 对原列表进行排序，默认为升序
10	list.clear() 清空列表
11	list.copy() 复制列表

5.2.3　列表的综合应用案例

1. 列表和 for 循环结合应用案例——杨辉三角形

杨辉三角形是二项式系数在三角形中的一种几何排列，在中国南宋数学家杨辉 1261 年所著的《详解九章算法》一书中出现。在欧洲，帕斯卡(1623—1662 年)于 1654 年发现这一规律，所以杨辉三角形又被称为帕斯卡三角形。帕斯卡的发现比杨辉要迟 393 年，比贾宪迟 600 年。杨辉三角形是中国数学史上的一个伟大成就。杨辉三角形的特征如下。

(1) 从第 3 行起，除首尾数字外，每个数等于它上方两数之和。

(2) 每行数字左右对称，且从 1 增加至最大值再减少到 1。

（3）第 n 行的数字有 n 项。

将 n 取 8，算法思路如下所示。

（1）嵌套循环外层控制行数 i，内层控制列数 j。

（2）第 1 列(j==0)和对角线(i==j)全为 1。

（3）剩下的数 a[i][j]=a[i-1][j-1]+a[i-1][j]。

（4）每列输出换行即可。输出效果如图 5-3 所示，代码如例 5-9 所示。

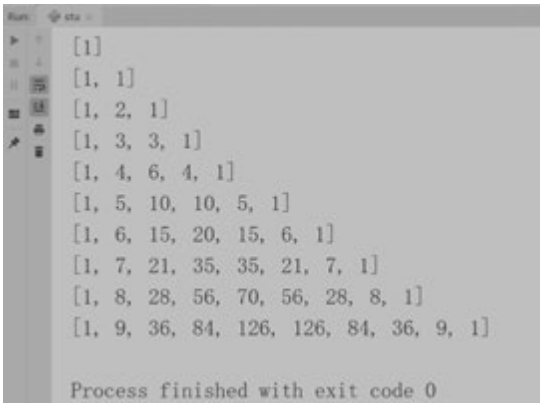


图 5-3　杨辉三角形

【例 5-9】 列表实现杨辉三角形的代码示例。

```
yh = [ ]
for i in range(8):
    yh.append([ ])
    for j in range(i + 1):
        if j == 0 or j == i:
            yh[i].append(1)
        else:
            yh[i].append(yh[i - 1][j - 1] + yh[i - 1][j])
    print(yh[i])
```

2. 嵌套列表应用案例——篮球运动员数据处理

篮球运动员(以下简称“球员”)信息如表 5-3 所示，代码如例 5-10 所示。

表 5-3　球员信息表　　单位：分

球　员	场均得分	场均篮板	场均助攻
球员 1	25.00	7.70	7.80
球员 2	21.80	7.90	3.10
球员 3	26.40	5.12	6.08
球员 4	28.00	6.82	5.38
球员 5	30.40	5.40	8.75
球员 6	18.60	5.40	7.88

【例 5-10】 嵌套列表综合应用代码示例。

（1）使用二维列表存储球员信息，代码示例如下：

```
nba_data = [
["球员", "场均得分", "场均篮板", "场均助攻"],
    ["球员 1", 25.00, 7.70, 7.80],
    ["球员 2", 21.80, 7.90, 3.10],
    ["球员 3", 26.40, 5.12, 6.08],
    ["球员 4", 28, 6.82, 5.38],
    ["球员 5", 30.4, 5.40, 8.75],
    ["球员 6", 18.60, 5.40, 7.88]
]
```

(2) 查找球员信息。查找 nba_data 中的相关信息,并输出对应内容,如图 5-4 所示。

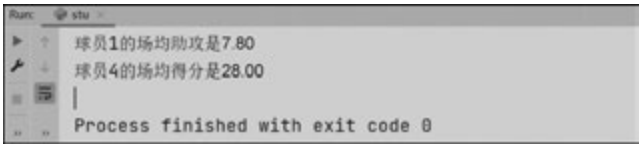


图 5-4　查找球员信息

实现嵌套列表查找操作的代码示例如下:

```
data1 = nba_data[1][3]              # 球员 1 的场均助攻
data2 = nba_data[4][1]              # 球员 4 的场均得分
print("{}的{}是{}".format(nba_data[1][0], nba_data[0][3], data1))
print("{}的{}是{}".format(nba_data[4][0], nba_data[0][1], data2))
```

(3) 增加球员信息。

① 按照相同数据结构追加到 nba_data 列表末尾。增加球员 7 的信息(球员:球员 7;场均得分:28.8;场均篮板:8.2;场均助攻:7.5)。

② 按照相同数据结构在球员 5 和球员 6 中间插入球员 8 的信息(球员:球员 8;场均得分:23.1;场均篮板:3.9;场均助攻:4.7)。

实现嵌套列表增加操作的代码示例如下:

```
player1 = ["球员 7", 28.80, 8.20, 7.50]
player2 = ["球员 8", 23.10, 3.90, 4.70]
nba_data.append(player1)              # 增加到末尾用 append()方法
nba_data.insert(6, player2)           # 插入某个位置用 insert()方法
print(nba_data)
```

(4) 删除球员信息。

① 删除球员 2 的所有信息,代码示例如下:

```
# 以下三种删除方式任选其一
del nba_data[2]
nba_data.remove(nba_data[2])
nba_data.pop(2)
```

② 删除球员 5 的场均篮板数,代码示例如下:

```
# 以下三种删除方式任选其一,注意前面增删操作对索引值的影响
del nba_data[4][2]
nba_data[4].remove(nba_data[4][2])
nba_data[4].pop(2)
```

(5) 修改球员信息。将球员 6 改为球员 0,代码示例如下：

```
nba_data[6][0] = "球员 0"
```

5.3 元组

元组类型与列表类型有很多相似的特征,首先,元组内部的元组元素也可以是任意其他数据类型；其次,元组元素也是有序的,每个元素都有对应的索引值,可以进行查找访问。不同的是,元组属于不可变数据类型,元组元素以及元组长度无法修改,无法进行增、删、改操作。所以在需要保证数据不能被随意修改的场景中,经常使用元组来达到禁止修改数据的目的。

5.3.1 元组的创建

(1) 通过一对圆括号(())可以创建元组,圆括号内部的元素使用英文逗号进行分隔。其中,非空元组可以省略括号,示例如下：

```
Tuple1 = ()                          # 创建一个空元组
Tuple2 = 1,                          # 由逗号结尾表示元组
Tuple3 = (1, )                       # 单个元素的元组
Tuple4 = (1, 2, [3, 4],"Python")     # 包含多个元素的元组
```

(2) 通过 tuple()函数也可以创建元组,通常是将列表或字符串转换为元组类型,代码如例 5-11 所示。

【例 5-11】 tuple()函数创建元组的代码示例。

```
List = [1, 2, [3, 4], "python"]
print(tuple(List))
Str = "Python"
print(tuple(Str))
```

输出结果如下：

```
(1, 2, [3, 4], 'python')
('P', 'y', 't', 'h', 'o', 'n')
```

5.3.2 元组的操作

1. 提取和切片操作

元组也属于序列类型,所以也可以进行单个元组元素的提取操作和多个元组元素的切

片操作，代码如例 5-12 所示。

【例 5-12】　元素的提取操作和切片操作的代码示例。

```
Tuple = (1, 2, [3, 4], "python")
print(Tuple[0])              # 正索引提取第一个元素
print(Tuple[0:3])            # 切片第一个到第三个元素
print(Tuple[-1])             # 负索引提取最后一个元素
```

输出结果如下：

```
1
(1, 2, [3, 4])
'python'
```

2. 运算符操作

元组可以进行加法(+)的合并操作、乘法(*)的重复操作，还可以运用成员运算符(in 或 not in)判断元素是否存在，代码如例 5-13 所示。

【例 5-13】　元组的基本运算符操作的代码示例。

```
print((1, "python", [3, 4]) + ('1', '3'))
print((1,2,3) * 2)
print("python" in (1, "python", [3, 4]) + ('1', '3'))
```

输出结果如下：

```
(1, 'python', [3, 4], '1', '3')
(1, 2, 3, 1, 2, 3)
True
```

Python 中包含的元组操作函数如表 5-4 所示。

表 5-4　Python 中包含的元组操作函数

序　号	函数及描述	序　号	函数及描述
1	len(tuple) 返回元组元素个数	3	min(tuple) 返回元组元素最小值
2	max(tuple) 返回元组元素最大值	4	sorted(tuple) 返回排序后的列表，默认为升序

5.3.3　元组的综合应用案例

元组和循环结构的综合应用案例——简易菜单查询系统。

根据用户输入的数字，返回相应的菜单信息，如表 5-5 所示。简易菜单查询系统的设计思路如下。

表 5-5 菜单信息

汉 堡 类	小 食 类	饮 料 类
香辣鸡腿堡 15.00 元	薯条 7.00 元	可口可乐 7.00 元
劲脆鸡腿堡 15.00 元	黄金鸡块 9.00 元	九珍果汁 15.00 元
新奥尔良烤鸡腿堡 16.00 元	香甜粟米棒 15.00 元	经典咖啡 15.00 元
半鸡半虾堡 20.00 元	鸡肉卷 10.00 元	雪碧 7.00 元

(1) 将菜单信息存储成元组。

(2) 通过字符串提示用户输入操作数字。

① 查询汉堡类菜单请输入 1。

② 查询小食类菜单请输入 2。

③ 查询饮料类菜单请输入 3。

④ 若不进行任何查询操作,则请输入 0。

(3) 当用户输入相应的数字时,程序要输出相应的详细食物菜单。只有当输入 0 后,才能退出系统。

代码如例 5-14 所示。

【例 5-14】 元组和循环结构的综合应用实例的代码示例。

```
print("""
*****************************************
            欢迎进入菜单查询系统

         查询汉堡类菜单请输入:1
         查询小食类菜单请输入:2
         查询饮料类菜单请输入:3
         若不进行任何查询操作,则请输入:0
*****************************************
""")
menu = ("【您已退出,感谢您的使用!】",
        "【香辣鸡腿堡 15.00 元 劲脆鸡腿堡 15.00 元 新奥尔良烤鸡腿堡 16.00 元 半鸡半虾堡
20.00 元】",
        "【薯条 7.00 元 黄金鸡块 9.00 元 香甜粟米棒 15.00 元 鸡肉卷 10.00 元】",
        "【可口可乐 7.00 元 九珍果汁 15.00 元 经典咖啡 15.00 元 雪碧 7.00 元】")
while True:
    number = int(input("请选择需要进行操作的对应数字:"))
    if number == 0:
        print(menu[number])
        break
print("您选择的菜单详情:\n" + menu[number])
```

输出结果如下：

```
*****************************************
            欢迎进入菜单查询系统

         查询汉堡类菜单请输入:1
```

```
        查询小食类菜单请输入:2
        查询饮料类菜单请输入:3
        若不进行任何查询操作,则请输入:0
 ****************************************

请选择需要进行操作的对应数字:1
您选择的菜单详情:
【香辣鸡腿堡 15.00 元 劲脆鸡腿堡 15.00 元  新奥尔良烤鸡腿堡 16.00 元 半鸡半虾堡 20.00 元】
请选择需要进行操作的对应数字:3
您选择的菜单详情:
【可口可乐 7.00 元  九珍果汁 15.00 元  经典咖啡 15.00 元  雪碧 7.00 元】
请选择需要进行操作的对应数字:0
【您已退出,感谢您的使用!】
```

首先,输出简易欢迎界面,提示用户可进行的操作; 然后,将菜单信息依次以字符串类型存储成元组元素,元组元素的索引值分别是 0、1、2、3,刚好与菜单的查询系统输入的操作数一致。

最后,将 while 循环的条件设置为 True,即条件永远成立,所以循环体加入了分支结构,当输入 0 时,触发 break,跳出循环,退出系统,否则输出 menu[number],例如,输入 1 时输出的数据是 menu[1],对应的就是汉堡类菜单的详情。

5.4　字典

本节主要介绍字典类型。字典属于可变数据类型,所以可以进行增、删、改、查的操作。另外,字典不同于列表或元组等数据类型,它的元素是以键值对(key:value)的形式存在的。就好比用《新华字典》查汉字一样,通过拼音或者偏旁部首来查找对应的汉字,Python 中的字典元素则是通过每一个键值对中的键来查找对应的值。字典键值对具有独特的数据形式,特点如下。

(1) 键必须是唯一的,但值是可以重复的。

(2) 键的类型只能是不可变数据类型(字符串、数字和元组),值可以是任何其他数据类型。

(3) 字典键值对是无序的,需要用键来查找值。

5.4.1　字典的创建

(1) 通过一对花括号({})可以创建字典,花括号内部的元素由键值对组成,多个键值对用逗号隔开,键值对内部的键和值用冒号隔开,示例如下:

```
dict1 = {}
dict2 = {'A': 65, 'B': 66, 'C': 67}
dict3 = {'001':["小李",100], {'002':["小梁",98], {'003':["小江",98]}
```

(2) 通过 dict()方法也可以创建字典,通常是将列表或元组转换为字典类型。需要注意的是,列表和元组内部的每个元素需要成对出现。也可以直接通过赋值表达式形式进行

字典的创建，示例如下：

```
dict4 = dict((('a', 97), ('b', 98)))              # 将元组转换成字典
dict5 = dict([("小赵", 18), ["小钱", 19]])        # 将列表转换成字典
dict6 = dict(a = 97, b = 98, c = 99)              # 以赋值表达式的形式创建字典
```

5.4.2 字典的操作

（1）查找操作。查找、访问字典中的值通常需要通过键来实现，也可以通过 get()方法进行查询，代码如例 5-15 所示。

【例 5-15】 字典的查找操作的代码示例。

```
employee_infos = {"001": ["王一", 10000],
                  "002": ["李二", 5200],
                  "003": ["张三", 4700],
                  "004": ["赵四", 3860],
                  "005": ["伍佰", 1200],
                  "006": ["六子", 8500]}
print(employee_infos["001"])
dict_get1 = employee_infos.get("007", "此键不存在")
dict_get2 = employee_infos.get("006", "此键不存在")
print(dict_get1)
print(dict_get2)
```

输出结果如下：

```
['王一', 10000]
此键不存在
['六子', 8500]
```

（2）增加操作。第一，可以直接通过赋值的方法，第二，可以通过 update()方法，代码如例 5-16 所示。

【例 5-16】 字典的增加操作的代码示例。

```
employee_infos["007"] = ["小红",6000]
employee_infos.update({"008":["小李", 7000], "009":["小丽", 6800]})
print(employee_infos)
```

输出结果如下：

```
{'001': ['王一', 10000], '002': ['李二', 5200], '003': ['张三', 4700],
 '004': ['赵四', 3860], '005': ['伍佰', 1200], '006': ['六子', 8500],
 '007': ['小红', 6000], '008': ['小李', 7000], '009': ['小丽', 6800]}
```

（3）删除操作。第一，可以通过关键字 del 来进行删除操作；第二，可以通过 pop()方法进行删除操作，代码如例 5-17 所示。

【例 5-17】　字典的删除操作的代码示例。

```
del employee_infos["006"]
employee_infos.pop("007")
print(employee_infos)
```

输出结果如下：

```
{'001': ['王一', 10000], '002': ['李二', 5200], '003': ['张三', 4700],
'004': ['赵四', 3860], '005': ['伍佰', 1200], '008': ['小李', 7000], '009': ['小丽', 6800]}
```

(4) 修改操作。直接通过赋值的方式修改，代码如例 5-18 所示。

【例 5-18】　字典的修改操作的代码示例。

```
employee_infos["001"] = ["王一一",11000]
print(employee_infos["001"])
employee_infos["001"][1] = 12000
print(employee_infos["001"])
```

输出结果如下：

```
['王一一', 11000]
['王一一', 12000]
```

Python 中包含的字典操作方法如表 5-6 所示。

表 5-6　Python 中包含的字典操作方法

方　法	描　述
dict.clear()	删除字典内所有元素
dict.copy()	返回一个字典的浅拷贝
dict.get(key,default=None)	返回指定键的值，如果键不在字典中则返回 default 设置的默认值
dict.items()	返回包含字典所有键值元组对的可迭代对象，可以使用 list()来转换为列表
dict.keys()	返回包含字典所有的键的可迭代对象，可以使用 list()来转换为列表
dict.values()	返回包含字典所有的值的可迭代对象，可以使用 list()来转换为列表
dict.update(dict2)	把字典 dict2 的键值对更新到 dict 里
dict.pop(key[,default])	删除字典给定键所对应的值，返回值为被删除的值。key 值必须给出，否则返回 default 值

5.4.3　字典的应用

字典的综合应用案例——学生信息表。

学生信息如表 5-7 所示。

表 5-7 学生信息

学生姓名	年龄/岁	Python/分	大数据/分
赵哈哈	18	100	99
钱呵呵	18	93	89
孙嘿嘿	19	89	92
李嘻嘻	17	100	100
周哇哇	38	0	0

代码如例 5-19 所示。

【例 5-19】 字典的综合应用。

(1) 使用嵌套字典存储学生信息,代码示例如下:

```
dict_1 = {
    "200601": {"name": "赵哈哈", "age": 18, "score": [100, 99]},
    "200602": {"name": "钱呵呵", "age": 18, "score": [93, 89]},
    "200603": {"name": "孙嘿嘿", "age": 19, "score": [89, 92]},
    "123456": {"name": "周哇哇", "age": 38, "score": [0, 0]}
}
```

(2) 查找学生信息。

① 查找赵哈哈的年龄,代码示例如下:

```
stu1_age = dict_1["200601"]["age"]
print("赵哈哈的年龄是{}岁".format(stu1_age))
```

② 查找赵哈哈的 Python 成绩,代码示例如下:

```
stu1_python = dict_1["200601"]["score"][1]
print("赵哈哈的 python 成绩是{}分".format(stu1_python))
```

③ 增加学生信息。学生姓名:李嘻嘻;年龄:17;Python 成绩:100;大数据成绩:100。代码示例如下:

```
dict_1["200604"] = {"name": "李嘻嘻", "age": 17, "score": [100, 100]}
print(dict_1)              # 使用 update()方法也可以
```

(3) 删除学生信息。

① 删除周哇哇的所有信息,代码示例如下:

```
# 两种删除方式任选其一皆可
del dict_1["123456"]              # del 关键字删除
dict_1.pop("123456")              # pop()方法删除传入键即可
```

② 删除赵哈哈的所有分数，代码示例如下：

```
dict_1["200601"].pop("score")
```

③ 删除钱呵呵的 Python 成绩，代码示例如下：

```
# 注意需要删除的数据的数据类型，此处为列表所以列表的删除方法都可以使用，示例如下
dict_1["200602"]["score"].pop(0)
del dict_1["200602"]["score"][0]
```

(4) 修改学生信息。

① 将赵哈哈的年龄改成 20，代码示例如下：

```
dict_1["200601"]["age"] = 20
```

② 将李嘻嘻的大数据成绩减少 2 分，代码示例如下：

```
dict_1["200604"]["score"][1] -= 2
```

5.5　集合

Python 中的集合类型与高中数学中的集合有很多相似的特征，在学习集合类型时可以进行对比学习。集合类型可以细分为可变集合(set)和不可变集合(frozenset)，一般没有特别强调时，介绍的都是可变集合，所以也会有对应的增、删、改、查操作。而集合的特征如下：

(1) 集合中的元素是无序的。

(2) 集合中的元素是唯一的、不重复的。可以利用集合的这一特性进行去重操作。

(3) 集合的元素必须是不可变数据类型，即由数字、字符串和元组组成。

5.5.1　集合的创建

(1) 通过一对花括号{}可以创建集合，集合元素用逗号进行分隔。但是需要特别注意，空集合不能用{}来创建，因为{}创建出来的是空字典。示例如下：

```
set1 = {100, 'word', 10.5, True}
```

(2) 通过 set()方法也可以创建集合，空集合就需要用 set()方法创建。示例如下：

```
set_one = set('tuple')
set_two = set((13, 15, 17, 19))
```

frozenset()用于创建不可变集合。

5.5.2　集合的操作

(1) 增加操作。第一，可以使用 add()方法增加一个集合元素；第二，可以使用 update()方

法增加多个集合元素，代码如例 5-20 所示。

【例 5-20】 集合的增加操作代码示例。

```
food = {"鱼香肉丝","米饭","鱼香肉丝","水煮牛肉","米饭","葱爆羊肉","蛋炒饭"}
new_food = {"番茄炒蛋","小鸡炖蘑菇"}
food.add("土豆丝")
food.update(new_food)
print("a 店当日销售的菜品种类:")
print(food)
```

输出结果如下：

```
a 店当日销售的菜品种类:
{'水煮牛肉', '土豆丝', '葱爆羊肉', '蛋炒饭', '小鸡炖蘑菇', '鱼香肉丝', '米饭', '番茄炒蛋'}
```

（2）删除操作。第一，可以使用 remove()方法删除指定元素；第二，可以使用 discard()方法删除指定元素；第三，可以使用 pop()方法随机删除某个元素，代码如例 5-21 所示。

【例 5-21】 集合的删除操作代码示例。

```
food.remove("米饭")              # 删除不存在集合元素,会报错
food.discard("米饭")             # 删除不存在集合元素,不会报错
food.pop()                       # 随机删除
print(food)
food.clear()                     # 清空
print(foid)
```

输出结果如下：

```
{'水煮牛肉', '蛋炒饭', '葱爆羊肉', '鱼香肉丝', '土豆丝', '小鸡炖蘑菇'}
set()              # 空集合
```

5.5.3 集合的运算

高中数学中的集合有交集、并集等集合的运算，而 Python 中的集合类型也有相应的运算。接下来一一介绍。

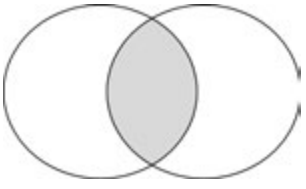


图 5-5 交集的维恩图

（1）交集运算，符号是 &，对应方法是 intersection()。交集的维恩图如图 5-5 所示。

两个集合的交集运算，得到的结果是由两个集合共有的集合元素组成的集合，代码如例 5-22 所示。

【例 5-22】 集合的交集操作的代码示例。

```
set1 = {'太阳','地球', '金星','火星', '土星'}
set2 = {'木星','地球','火星','水星'}
print(set1 & set2)                    # 使用符号 & 获取交集
print(set1.intersection(set2))        # 使用集合方法 intersection()获取交集
```

输出结果如下：

```
{'火星', '地球'}
{'火星', '地球'}
```

（2）并集运算，符号是|，对应方法是 union()。并集的维恩图如图 5-6 所示。

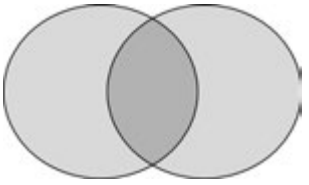


图 5-6　并集的维恩图

两个集合的并集运算，得到的结果是两个集合所有的集合元素组成的集合，因为集合元素的唯一性，所以相同的元素会自动去重，代码如例 5-23 所示。

【例 5-23】 集合的并集操作的代码示例。

```
set1 = {'太阳','地球', '金星','火星', '土星'}
set2 = {'木星','地球','火星','水星'}
print(set1|set2)              # 使用符号|获取并集
print(set1.union(set2))       # 使用集合方法 union()获取并集
```

输出结果如下：

```
{'金星', '地球', '土星', '木星', '火星', '太阳', '水星'}
{'金星', '地球', '土星', '木星', '火星', '太阳', '水星'}
```

（3）差集运算，符号是－，对应方法是 difference()。差集的维恩图如图 5-7 所示。

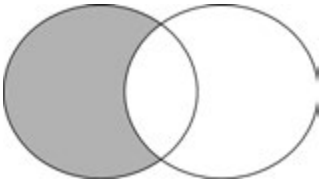


图 5-7　差集的维恩图

两个集合的差集运算。例如，set1－set2 得到的结果是 set1 相对于 set2 独有的集合元素组成的集合，代码如例 5-24 所示。

【例 5-24】 集合的差集操作的代码示例。

```
set1 = {'太阳','地球', '金星','火星', '土星'}
set2 = {'木星','地球','火星','水星'}
print(set1 - set2)              # 使用 - 来获取差集
print(set2.difference(set1))    # 使用集合方法 difference()获取差集
```

输出结果如下：

```
{'金星', '太阳', '土星'}              # set1 对 set2 的差集
{'木星', '水星'}                      # set2 对 set1 的差集
```

（4）异或集运算，符号是^，对应方法是 symmetric_difference()。异或集的维恩图如图 5-8 所示。

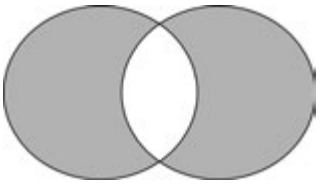


图 5-8　异或集的维恩图

两个集合的异或集（又被称为对称补集）运算，得到的结果是 set1 和 set2 各自独有的集合元素共同组成的集合，代码如例 5-25 所示。

【例 5-25】 集合的异或集操作的代码示例。

```
set1 = {'太阳','地球', '金星','火星', '土星'}
set2 = {'木星','地球','火星','水星'}
print(set1^set2)                          # 获取异或集
print(set1.symmetric_difference(set2))   # 使用集合方法 symmetric_difference()获取异或集
```

输出结果如下：

```
{'太阳', '土星', '木星', '金星', '水星'}
{'太阳', '土星', '木星', '金星', '水星'}
```

另外，高中数学中有子集、真子集的概念，而 Python 中也有相应的运算符和方法来进行判断。

（1）运用比较运算符＜和＜＝来判断真子集和子集，例如有集合 A、B，A＜B 用于判断 A 集合中所有元素是否都包含在 B 集合中，不包括 A、B 集合相等的情况；A＜＝B 用于判断 A 集合中所有元素是否都包含在 B 集合中，包括 A、B 集合相等的情况。代码如例 5-26 所示。

【例 5-26】 集合的子集和真子集操作的代码示例。

```
set1 = {"武汉", "长沙", "郑州", "南昌", "北京"}
set2 = {"武汉", "长沙", "南昌", "北京"}
print(set1.issubset(set2))              # 子集对应方法
print(set1 <= set2)                     # 子集对应运算符
print(set1 < set2)                      # 真子集
```

输出结果如下：

```
False
False
False
```

（2）运用比较运算符＞和＞＝来判断真超集和超集，例如有集合 A、B，A＞B 用于判断 A 集合是否包含所有 B 集合的元素，不包括 A、B 集合相等的情况；A＞＝B 用于判断 A 集合是否包含所有 B 集合的元素，包括 A、B 集合相等的情况。代码如例 5-27 所示。

【例 5-27】 集合的超集和真超集操作的代码示例。

```
set1 = {"武汉", "长沙", "郑州", "南昌", "北京"}
set2 = {"武汉", "长沙", "南昌", "北京"}
print(set1.issuperset(set2))              # 超集对应方法
print(set1 >= set2)                       # 超集对应运算符
print(set1 > set2)                        # 真超集
```

输出结果如下：

```
True
True
True
```

Python 中集合的相关操作方法如表 5-8 所示。

表 5-8 集合的操作方法

方 法	描 述
Set.add()	给集合添加一个元素
Set.update()	给集合添加多个元素
Set.discard()	删除集合中指定的元素
Set.pop()	随机删除元素
Set.remove()	删除指定元素
Set.clear()	删除集合中的所有元素
Set1.union(Set2)	返回集合 Set1 和 Set2 的并集
Set1.intersection(Set2)	返回集合 Set1 和 Set2 的交集
Set1.difference(Set2)	返回集合 Set1 和 Set2 的差集
Set1.symmetric_difference(Set2)	返回两个集合中不重复的元素集合,即异或集
Set1.isdisjoint(Set2)	判断两个集合是否包含相同的元素,如果没有包含则返回 True,否则返回 False
Set.copy()	复制一个集合
Set1.issubset(Set2)	判断指定集合是否为该方法参数集合的子集
Set1.issuperset(Set2)	判断该方法的参数集合是否为指定集合的超集

5.5.4 集合的应用

下面介绍集合的综合应用案例——最大公约数和最小公倍数。

最大公约数,也称为最大公因数、最大公因子,指两个或多个整数共有约数中最大的一个约数。

两个或多个整数公有的倍数叫作它们的公倍数,其中除 0 以外最小的一个公倍数就叫作这几个整数的最小公倍数。

求最大公约数有多种方法,常见的有质因数分解法、短除法、辗转相除法、更相减损术。数学定理已经证明两个数的最大公约数与最小公倍数的乘积等于这两个数的乘积。本案例中则不适合采用常规的算法来求解最大公约数和最小公倍数,而是运用集合的相关操作实现。

这里给定的两个数为 24 和 36,获取两个给定数的最大公约数和最小公倍数,并分别提取出它们各自独有的约数和倍数集合。实现如下功能。

(1) 创建 24 和 36 的约数和倍数集合。

(2) 24 的约数集合{1,2,3,4,6,8,12,24},36 的约数集合{1,2,3,4,6,9,12,18,36}。

(3) 各自的倍数集合分别只取到 5 倍,如 24 的倍数集合{24,48,72,96,120}。

(4) 运用 Python 中的交集运算求出 24 和 36 的最大公约数与最小公倍数集合。

(5) 运用 max()和 min()函数分别获取各集合中的最大值和最小值,即最大公约数和最小公倍数。

(6) 运用差集运算分别获取并输出 24 和 36 各自独有的约数集合。

代码如例 5-28 所示。

【例 5-28】 集合的差集操作的代码示例。

```
yue_set1 = {1, 2, 3, 4, 6, 8, 12, 24}
yue_set2 = {1, 2, 3, 4, 6, 9, 12, 18, 36}
bei_set1 = {24, 48, 72, 96, 120}
bei_set2 = {36, 72, 108, 144, 180}

yue_intersection = yue_set1 & yue_set2              # 交集得到公约数
bei_intersection = bei_set1 & bei_set2              # 交集得到公倍数
Max = max(yue_intersection)                         # 最大公约数
Min = min(bei_intersection)                         # 最小公倍数

yue1_difference = yue_set1 - yue_set2
yue2_difference = yue_set2 - yue_set1
print(Max, Min, yue1_difference, yue2_difference)
```

输出结果如下：

```
12 72 {8, 24} {9, 18, 36}
```

5.6 实践案例

1. 实践案例介绍

案例主题：学生成绩管理系统——复合数据类型。

场景背景：作为班级学习委员，你需要开发一个简单的学生成绩管理系统，使用复合数据类型存储和处理学生成绩数据。学生基本信息如下：

```
students = ( ("张明", "2023001", 85), ("李华", "2023002", 92),
             ("王芳", "2023003", 78), ("赵雷", "2023004", 65),
             ("陈晓", "2023005", 90), ("刘伟", "2023006", 55),
             ("杨帆", "2023007", 88), ("周婷", "2023008", 95),
             ("吴昊", "2023009", 82), ("郑琳", "2023010", 72))
```

系统需要实现以下功能：

(1) 存储学生的基本信息(姓名、学号和成绩)。

(2) 计算全班平均成绩。

(3) 查找最高分学生。

(4) 统计不及格学生(分数＜60)。

(5) 生成优秀学生名单(分数≥90)。

学生成绩排名(降序)。

涉及知识点：

- 分支、循环结构。
- 复合数据类型(元组、列表)数据处理。
- 字符串格式化。

2. 实践案例代码

```
# ===== 数据准备 =====
# 使用元组存储学生信息(姓名、学号、成绩)
students = (
    ("张明", "2023001", 85),
    ("李华", "2023002", 92),
    ("王芳", "2023003", 78),
    ("赵雷", "2023004", 65),
    ("陈晓", "2023005", 90),
    ("刘伟", "2023006", 55),
    ("杨帆", "2023007", 88),
    ("周婷", "2023008", 95),
    ("吴昊", "2023009", 82),
    ("郑琳", "2023010", 72)
)

# ===== 系统功能实现 =====

# 1. 计算全班平均成绩
total_score = 0
for student in students:
    total_score += student[2] # 索引 2 对应成绩
average_score = total_score / len(students)
print(f"\n=== 全班成绩统计 === ")
print(f"全班平均分: {average_score:.1f}")

# 2. 查找最高分学生
max_score = 0
top_student = None
for student in students:
    if student[2] > max_score:
        max_score = student[2]
        top_student = student
print(f"\n最高分学生: {top_student[0]}(学号:{top_student[1]}), 成绩:{top_student[2]}")

# 3. 统计不及格学生
fail_students = [ ]
for student in students:
    if student[2] < 60:
        fail_students.append(student)
print(f"\n不及格学生人数: {len(fail_students)}")
print("不及格学生名单:")
for student in fail_students:
    print(f" - {student[0]}(学号:{student[1]}), 成绩:{student[2]}")

# 4. 生成优秀学生名单(≥90 分)
excellent_students = [ ]
for student in students:
    if student[2] >= 90:
        excellent_students.append(student)
```

```
print("\n优秀学生名单(≥90 分):")
for student in excellent_students:
    print(f" - {student[0]}(学号:{student[1]}), 成绩:{student[2]}")

# 5. 按成绩降序排列学生
# 使用元组解包和排序算法
sorted_students = list(students) # 转换为列表以便修改

# 使用冒泡排序按成绩降序排列
n = len(sorted_students)
for i in range(n):
    for j in range(0, n - i - 1):
        if sorted_students[j][2] < sorted_students[j + 1][2]:
            # 交换位置
            sorted_students[j], sorted_students[j + 1] = sorted_students[j + 1], sorted_students
[j]

print("\n=== 学生成绩排名 ===")
for i, student in enumerate(sorted_students, 1):
    print(f"{i}. {student[0]} - {student[2]}分")
```

3. 实践案例结果展示

```
=== 全班成绩统计 ===
全班平均分: 80.2

最高分学生: 周婷(学号:2023008), 成绩:95

不及格学生人数: 1
不及格学生名单:
  - 刘伟(学号:2023006), 成绩:55

优秀学生名单(≥90 分):
  - 李华(学号:2023002), 成绩:92
  - 陈晓(学号:2023005), 成绩:90
  - 周婷(学号:2023008), 成绩:95

=== 学生成绩排名 ===
1. 周婷 - 95 分
2. 李华 - 92 分
3. 陈晓 - 90 分
4. 杨帆 - 88 分
5. 张明 - 85 分
6. 吴昊 - 82 分
7. 王芳 - 78 分
8. 郑琳 - 72 分
9. 赵雷 - 65 分
10. 刘伟 - 55 分
```

本章小结

本章先介绍数据类型的两种分类，其中列表和元组属于序列类型，字典属于映射类型，集合属于集合类型。然后介绍可变数据类型和不可变数据类型的定义。最后介绍复合数据类型（列表、元组、字典和集合类型）的操作方法、相应的运算和应用。

习题 5