第 3 章

组合数据类型

Python 之所以简单易学、使用方便，其中一个重要支持技术就是组合数据类型。通过各类组合数据类型，提供了能满足各种应用需要的数据定义形式，极大方便了程序的开发，缩减了代码量。因此，要想学好 Python，一定要掌握组合数据类型的定义与使用。本章从组合数据类型概述入手，讲解列表、元组、字典和集合等常用的组合数据类型。读者应该重点掌握列表、字符串和字典的使用，以及相互间的类型转换。

3.1 组合数据类型概述

在 Python 程序设计中，基本数据类型主要处理简单的数据计算，但对于复杂数据计算，例如一组数据的统计计算，就需要组合数据类型进行处理。组合数据类型包括列表、元组、字符串、字典、集合等。其中字符串可以看成是单一字符的有序组合，属于组合数据类型，也可以看成是一串字符，也属于基本数据类型。

在组合数据中，一个重要的特性就是数据的存储是否有序。如果是有序的，就可以利用存储的坐标索引来获得数据。例如，字符串就是有序组合数据，设字符串 s="Python"，利用 s[2]就可以获得字符"t"。而对于无序数据，要获得数据，只能通过数据标识的关键值(key)以字典方式获得，或者通过循环迭代的方式逐一随机获得(例如集合)。

在组合数据中，另外一个重要特性，就是组合数据中的值是否可以被修改，即组合数据类型是否“可变”。例如，字符串就是一个不可修改的数据类型，列表就是可修改类型。根据以上描述，总结如表 3.1 所示的常用组合数据类型分类。

表 3.1 组合数据类型分类

组合数据类型	有序类型	列表(list,[ ]),可变
		元组(tuple,[ ]),不可变
		字符串(str," "),不可变
	无序类型	字典(dict,{key: value}),可变
		集合(set,{ }),可变

3.2 列　　表

3.2.1 列表的定义与赋值

1. 列表定义与元素访问

列表是常用的组合数据类型，它是包含0个或多个元素的有序序列。列表的基本形式为：

```
[<元素 1>, <元素 2>,…, <元素 n>]
```

或

```
[ ]
```

多个元素之间用逗号分隔，元素个数无限制。各元素可以是不同的任意数据类型，包括组合数据类型。0个元素的列表为空列表[ ]。列表中元素的索引编号，与字符串中的字符编号方法相同，也是从左到右，从0开始递增；或从右到左，从－1开始递减。可以通过索引编号访问元素，语法格式为：

```
<列表名>[索引编号]
```

示例如下：

```
>>>x=[1,2,3,4,5]
>>>x
    [1, 2, 3, 4, 5]
>>>x[0]
    1
>>>x[1]
    2
>>>x[-1]
    5
>>>x[-4]
    2
>>>x=[]                          #空列表
>>>x
    []
>>>y=["华北科技学院",17000,"北京东燕郊"]
>>>y[2]
    '北京东燕郊'
>>>y[3]                          #索引 3 超出范围,提示错误
    Traceback (most recent call last):
        File "<pyshell#151>", line 1, in <module>
            y[3]
    IndexError: list index out of range
```

列表 y 有 3 个元素，索引为 0、1、2，所以 y[3]中的索引编号 3 超出了范围，程序提示错误。

也可以直接以列表形式输入数据，例如：

```
>>>x=eval(input("请输入数据:"))
    请输入数据:[1,3,5,7]
>>>x
    [1, 3, 5, 7]
```

2. 使用列表为多变量赋值

可以使用列表同时为多个变量赋值，通常使用 input()函数配合 split()等方法来创建列表，从而实现同时为多个变量赋值。例如：

```
>>>x,y,z=[1,2,3]
>>>print(x,y,z)
    1 2 3
>>>a,b,c=input("请输入数据:").split()
    请输入数据:11 22 33
>>>a
    '11'
>>>b
    '22'
>>>c
    '33'
>>>a,b,c=input("请输入数据:").split(",")
    请输入数据:11,22,33
>>>a
    '11'
>>>b
    '22'
>>>c
    '33'
```

使用 input()输入的数据都被视为字符串，split()将字符串分隔成列表，列表中的每个元素仍是字符串。若要处理数字类型，可以使用 map()函数来配合实现，详见 3.2.4 节。

```
>>>x,y,z=map(int,input("请输入数据:").split())
```

上例中通过 map()函数把 x、y 和 z 都转换为整型数据。

3. 不可变与可变数据类型

Python 数据类型分为不可变数据类型和可变数据类型。不可变类型的对象一旦创建，其值就不能被修改了，如果修改就会为其分配新的内存空间。可变类型的对象值可以被修改。不可变类型包括数字型、字符串、布尔型、元组，可变类型包括列表、字典、集合。列表的元素个数和内容都是可变的。

```
>>>x=3
>>>id(x)
    1922990256
>>>x=x+1
>>>id(x)                          #整数 x 的值变化,内存地址变化
    1922990288
>>>x=[1,2,[3,4],5]
>>>id(x)
    1905953032264
>>>x[1]=7                         #列表元素 x[1]赋值为 7
>>>x
     [1, 7, [3, 4], 5]
>>>id(x)                          #列表 x 的值变化,内存地址没变
    1905953032264
>>>x[2]                           #x[2]为一个列表元素
     [3, 4]
>>>x[2][0]                        #x[2]中的第 0 个元素
    3
>>>x[2][1]=9                      #x[2]中的第 1 个元素赋值为 9
>>>x
     [1, 7, [3, 9], 5]
```

x[2][1]是将 x[2]看作一个整体,它是列表 x 的一个列表元素,但它又是一个列表,所以它的第一个元素就表示为 x[2][1]。可以采用这种多级索引方式,方便逐级访问元素的信息。

3.2.2 列表的基本操作

列表的基本操作符与字符串的操作符功能类似。列表的基本操作符如表 3.2 所示,可与字符串操作符比较来理解。这里假设 s=[1,2,3,4,5],t=['a','b'],x=3。

表 3.2 列表基本操作符

操作符	功 能 说 明	示例	结 果
+	列表连接	s+t	[1, 2, 3, 4, 5, 'a', 'b']
*	列表重复	t * 2	['a', 'b', 'a', 'b']
[N]	索引,返回列表的第 N 个元素	t[0]	'a'
[M:N]	切片,返回列表中第 M 到 N 个元素的子序列,不包含第 N 个元素	s[1:4]	[2, 3, 4]
[M:N:K]	步长切片,返回列表中第 M 到 N 个元素以 K 为步长的子序列,K 为正负整数	s[1:4:2] s[4:1:-2]	[2, 4] [5, 3]
in	元素是否在列表中,是则返回 True,否则返回 False	x in s	True

续表

操作符	功能说明	示例	结果
not in	元素不在列表中，是则返回 True，否则返回 False	x not in s	False
del	删除列表中的元素	del t[0]	#输出 t 的结果 ['b']

```
>>>t=['a','b']
>>>2*t
    ['a', 'b', 'a', 'b']
>>>s=[1,2,3,4,5]
>>>del s[1:3:2]                    #删除索引值从 1 到 3(不包括 3)步长为 2 的元素
>>>s
    [1, 3, 4, 5]
>>>s[::-1]                         #列表翻转
    [5,4,3,1]
```

3.2.3 列表的内置方法

列表的内置方法如表 3.3 所示，这里假设 s=[1,2,3,4,5]，t=['a','b']，x=3。

表 3.3 列表内置方法

方法	功能说明	示例	输出 s 结果
count(x)	返回 x 在列表中的出现次数	s.count(2)	1
index(x,[M,[N]])	返回列表中第一个值为 x 的元素的索引，若不存在，则抛出异常	s.index(5)	4
append(x)	将 x 追加至列表尾部	s.append(x)	[1, 2, 3, 4, 5, 3]
extend(t)	将列表 t 所有元素追加至列表尾部	s.extend(t)	[1, 2, 3, 4, 5, 'a', 'b']
insert(i,x)	在列表第 i 位置前插入 x	s.insert(1, 7)	[1, 7, 2, 3, 4, 5]
remove(x)	在列表中删除第一个值为 x 的元素	s.remove(2)	[1, 3, 4, 5]
pop([i])	删除并返回列表中下标为 i 的元素，若省略 i，则 i 默认为−1，弹出最后一个元素	s.pop(2)	3
clear()	列表清空，删除列表中所有元素，保留列表对象	s.clear()	[ ]
reverse()	列表翻转	s=[1,3,5,4,2] s.reverse()	[2, 4, 5, 3, 1]
sort([key=None, reverse=False])	列表排序，key 用来指定排序规则，reverse 为 False 则升序，True 则降序	s=[1,3,5,4,2] s.sort()	[1, 2, 3, 4, 5]
copy()	列表浅复制	s1=s.copy()	#输出 s1 的结果 [1, 2, 3, 4, 5]

1. 查找数据：index(x,[M,[N]])方法的使用

在列表第 M 到 N 个元素内查找第一个值为 x 的元素，返回其索引，不包括第 N 个元素。若不指定 M 和 N，则在整个列表中查找。若不指定 N，则从第 M 个到最后查找。若不存在 x，则抛出异常。

```
>>>s=[1,3,2,3,4,5]
>>>s.count(3)
   2
>>>s.count(7)                    #列表中 7 出现次数为 0
   0
>>>s.index(3)
   1
>>>s.index(3,2)                  #省略 N,则从第 2 个到最后
   3
>>>s.index(3,2,4)
   3
>>>s.index(3,2,3)                #从第 2 个到第 3 个,不包括第 3 个
   Traceback (most recent call last):
      File "<pyshell#18>", line 1, in <module>
         s.index(3,2,3)
   ValueError: 3 is not in list
>>>s.index(6)
   Traceback (most recent call last):
      File "<pyshell#87>", line 1, in <module>
         s.index(6)
   ValueError: 6 is not in list
```

2. 插入数据：insert(i，x)方法的使用

在列表的第 i 位置前插入 x，该位置后面的所有元素后移并且在列表中的索引加 1，如果 i 为正数且大于列表长度，则在列表尾部追加 x，如果 i 为负数且小于列表长度的相反数，则在列表头部插入元素 x。

```
>>>s=[1,2,3,4,5]
>>>s.insert(0, 7)
>>>s
   [7, 1, 2, 3, 4, 5]
>>>s.insert(6, 8)
>>>s
   [7, 1, 2, 3, 4, 5, 8]
>>>s.insert(-1, 9)
>>>s
   [7, 1, 2, 3, 4, 5, 9, 8]
>>>s.insert(-8, 0)
>>>s
```

```
    [0, 7, 1, 2, 3, 4, 5, 9, 8]
```

3. 按值删除数据：remove(x)方法的使用

在列表中删除第一个值为 x 的元素，该元素之后所有元素前移并且索引减 1，如果列表中不存在 x，则抛出异常。

```
>>>s=[1,3,2,3,4,5]
>>>s.remove(3)                              #删除第一个 3
>>>s
    [1, 2, 3, 4, 5]
>>>s.remove(7)                              #删除 7,提示错误
Traceback (most recent call last):
       File "<pyshell#45>", line 1, in <module>
           s.remove(7)
   ValueError: list.remove(x): x not in list
```

4. 按索引删除数据：pop([i])方法的使用

删除并返回列表中下标为 i 的元素。若省略 i，则 i 默认为－1，弹出最后一个元素。如果弹出中间位置的元素，则后面的元素索引减 1。如果 i 不是索引范围内的整数，则抛出异常。

```
>>>s=[1,2,3,4,5]
>>>s.pop()
   5
>>>s
    [1, 2, 3, 4]
>>>s.pop(-2)
   3
>>>s
    [1, 2, 4]
>>>s.pop(3)
   Traceback (most recent call last):
       File "<pyshell#80>", line 1, in <module>
           s.pop(3)
   IndexError: pop index out of range
```

5. 列表排序：sort([key，reverse])方法的使用

key 用来指定排序规则，默认按照数值排序。reverse 默认值为 False，表示升序。两个参数可以省略一个，也可以都省略。

```
>>>s=[1,3,0,5,12,4,2]
>>>s.sort(key=str, reverse=True)            #按照字符串类型,降序排列
>>>s
    [5, 4, 3, 2, 12, 1, 0]
>>>s.sort(key=str)                          #按照字符串类型,升序排列
>>>s
```

```
    [0, 1, 12, 2, 3, 4, 5]
>>>s.sort(reverse=True)                    #按照数值类型,降序排列
>>>s
    [12, 5, 4, 3, 2, 1, 0]
>>>s.sort()                                #按照数值类型,升序排列
>>>s
    [0, 1, 2, 3, 4, 5, 12]
```

6. 列表浅复制：copy()方法的使用

Python 程序设计中的复制,分为浅复制和深复制。浅复制是将原列表的引用复制到一个新列表。原列表与新列表中的不可变类型数据变化时,互不影响;若是可变类型数据变化时,则互相影响。深复制是将原列表的数值复制到一个新列表,两个列表互相独立,互不影响。可以使用标准库 copy 中的 deepcopy()函数来实现深复制。

```
>>>s=[1,2,[3,4],5]
>>>s1=s.copy()
>>>s1
    [1, 2, [3, 4], 5]
>>>id(s)
   2223554802312
>>>id(s1)
   2223554802824
>>>s[0]=0                                  #s[0]、s[1]为整型,修改后不会相互影响
>>>s1[1]=7
>>>s
    [0, 2, [3, 4], 5]
>>>s1
    [1, 7, [3, 4], 5]
>>>s[2][0]=0                               #s[2]为列表类型,修改后会相互影响
>>>s1[2][1]=9
>>>s
    [0, 2, [0, 9], 5]
>>>s1
    [1, 7, [0, 9], 5]
>>>s2=s1                                   #赋值语句,s1 和 s2 指向同一个列表对象
>>>s2
    [1, 7, [0, 9], 5]
>>>id(s1)                                  #内存地址相同,s1 和 s2 所有修改互相影响
   2223554802824
>>>id(s2)
   2223554802824
```

3.2.4 列表的内置函数

可操作列表的内置函数如表 3.4 所示,这里假设 s=[1,2,3,4,5],t=['a','b']。

表 3.4 可操作列表的内置函数

函 数	功能说明	示 例	结 果
list([x])	将字符串或元组 x 转换为列表,若省略 x,则创建空列表	list("Python 程序") list((1,2,3))	['P', 'y', 't', 'h', 'o', 'n', '程', '序'] [1, 2, 3]
len(s)	列表 s 的元素个数(长度)	len(s)	5
min(s)	列表 s 中的最小元素	min(s)	1
max(s)	列表 s 中的最大元素	max(t)	'b'
sum(s[,start])	列表 s 中元素求和,可设起始值 start,若省略,start 默认为 0	sum(s)	15
sorted(s[,key=None,reverse=False])	列表排序,参数含义同 sort() 方法	s=[1,3,5,4,2] sorted(s)	#输出 s 的结果 [1, 2, 3, 4, 5]
map(fun,iterable)	函数 fun 依次作用在 iterable 的每个元素上,得到一个新的迭代对象并返回	x,y=map(int,['1','2'])	#输出 x 的结果 1 #输出 y 的结果 2

常用内置函数的使用示例如下:

```
>>>s=list()                          #创建一个空列表
>>>s
    []
>>>s=[1,2,[3,4],5]
>>>len(s)
    4
>>>s=[1,2,3,4,5]
>>>sum(s,10)                         #相当于 10+sum(s)
    25
>>>x,y,z=map(int,input("请输入数据:").split())
    请输入数据:1 2 3                  #看似整数,实际被视为'1 2 3'
>>>x
    1
>>>y
    2
>>>z
    3
>>>'1 2 3'.split()                   #相当于 input("请输入数据:").split()
    ['1', '2', '3']
>>>type(map(int,['1', '2', '3']))
    <class 'map'>                     #map 类型
>>>x,y,z=map(int,['1', '2', '3'])    #对['1', '2', '3']中每个元素都调用 int()函数
>>>x                                 #转换为整型后,分别赋值给 x,y,z
```

```
    1
>>>y
    2
>>>z
    3
```

【例 3-1】 利用列表建立一个存储学生的学号、姓名和成绩的列表结构，存储两门课的成绩，并输出。

编写程序如下：

```
1   id=["1001","1002","1003"]
2   name=["张一","张二","张三"]          #存储姓名
3   value1=[91,92,93]                   #存储课程 1 的成绩
4   value2=[81,82,83]                   #存储课程 2 的成绩
5   data=[id,name,value1,value2]        #组合存储
6   print(data)
```

运行结果：

```
[['1001', '1002', '1003'], ['张一', '张二', '张三'], [91, 92, 93], [81, 82, 83]]
```

3.2.5 range()函数的使用

列表的产生需要编写一定量的代码，能否自动生成符合一定规则的列表呢？答案是肯定的。例如，可以实现存储 1～100 的等差序列列表[1,2,…,100]。

range()函数就是可以迭代产生指定范围内的数字序列，其语法格式为：

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

range()函数返回从 start 到 end、步长为 step 的整型数据序列，不包括 end 的值。start、end 和 step 必须为整型数据。start 默认值为 0，step 默认值为 1，且不可为 0。可以只省略 step，或者同时省略 start 和 step。注意，该函数返回的是 range 类型。要想获得 list 类型数据，需要使用 list()函数进行强制类型转换。例如：

```
>>>range(5)
    range(0, 5)                         #start 默认为 0
>>>type(range(-5,5))
    <class 'range'>                     #range 类型
>>>list(range(2,10,2))
     [2, 4, 6, 8]
>>>list(range(10,5,-2))                #步长可以为负数，此时要求 end>start
     [10, 8, 6]
>>>for i in range(10,20):              #for 循环结构
    print(i,end=" ")                    #每输出一个 i 的值，后跟一个空格" "
    10 11 12 13 14 15 16 17 18 19       #for 循环的输出结果
```

3.3　元　　组

3.3.1　元组的定义与赋值

元组可以看作不可变的、只读版的列表，一旦创建就不能被修改。元组的基本形式为：

```
(<元素 1>, <元素 2>,…, <元素 n>)
```

或

```
( )
```

其中，小括号可以省略。当元组只有一个元素时，逗号不能省略。

tuple()函数与 list()函数类似，可以转换或创建成一个元组。

例如：

```
>>>s=(1,2,3,4,5)
>>>s
    (1, 2, 3, 4, 5)
>>>type(s)
   <class 'tuple'>
>>>s=tuple("Python 程序")
>>>s
    ('P', 'y', 't', 'h', 'o', 'n', '程', '序')
>>>s=tuple([1,2,3])
>>>s
    (1, 2, 3)
>>>s=(5)                                  #(5)不能表示一个元素的元组，这是整数 5
>>>type(s)
   <class 'int'>
>>>s=(5,)                                 #逗号不能省略
>>>type(s)
   <class 'tuple'>
>>>s=()                                   #空元组
>>>s
    ()
>>>s=tuple()                              #空元组
>>>s
    ()
```

3.3.2　元组的基本操作

在列表的基本操作、内置方法和内置函数中，那些不会改变元素值的基本都适用于元

组，在此不再累述。不适用于元组的方法和函数有 append()、extend()、insert()、remove()、pop()等。请读者参照列表学习内容，扩展学习元组。

元组就是不变的列表，正是元组不可修改的特点，使得它在某些场合是不可替代的。很多内置函数和序列类型方法的返回值为元组类型。元组可以用作字典的键，也可以作为集合的元素，而列表则不能。元组比列表的访问和处理速度更快，因此在不需要修改元素的操作时，建议使用元组。

3.4 字　　典

3.4.1 字典的定义与赋值

字典是键和值的映射关系，每个键对应一个值，它是键值对的无序可变序列，也是常用组合数据类型之一。字典的基本形式为：

```
{<键 1>: <值 1>, <键 2>: <值 2>, … , <键 n>: <值 n>}
```

或

```
{ }
```

注：Python 3.5(含)以前，字典为无序，3.6(含)以后版本为有序，提升访问效率。

字典的键只能使用不可变的类型，但值可以是可变的或者不可变的类型。键是唯一的，不能重复，值可以重复。字典的多个键值对之间是无序的，所以打印输出的顺序与开始创建的顺序可能不同。可以通过键访问获得字典中该键对应的值，语法格式为：

```
<字典名>[<键>]
```

也可以为键赋给新的值，来修改原有键对应的值。若键不存在，则添加一个新元素。

1. 字典的创建

dict()函数可以创建一个字典。其使用方法如下：

```
>>>x={"学校":"华北科技学院","学生数":17000,"地址":"北京东燕郊"}
    {'学校': '华北科技学院', '学生数': 17000, '地址': '北京东燕郊'}
>>>d={}                                    #空字典
>>>d
    {}
>>>type(d)
    <class 'dict'>
>>>d=dict()                                #空字典
>>>d
    {}
>>>d=dict(学校='华北科技学院', 学生数=17000, 地址='北京东燕郊')
>>>d
    {'学校': '华北科技学院', '学生数': 17000, '地址': '北京东燕郊'}
```

2. 字典取值与赋值

```
>>>d["地址"]                                    #访问键"地址"的值
    '北京东燕郊'
>>>d["学生数"]=18000                            #修改键"学生数"的值
>>>d["电话"]                                    #访问不存在的键"电话",输出错误提示
    Traceback (most recent call last):
        File "<pyshell#30>", line 1, in <module>
            d["电话"]
    KeyError: '电话'
>>>d["电话"]="01061591417"                      #键"电话"不存在,则添加键值对
>>>d
    {'学校': '华北科技学院', '学生数': 18000, '地址': '北京东燕郊', '电话':
'01061591417'}
```

3.4.2 字典的基本操作

字典的常用函数和方法如表3.5所示,这里假设d={"学校":"华北科技学院","学生数":17000,"地址":"北京东燕郊"},t={'学生数':18000}。

表3.5 字典常用函数和方法

函数和方法	功能说明	示例	结果
keys()	返回所有的键信息	d.keys()	dict_keys(['学校', '学生数', '地址'])
values()	返回所有的值信息	d.values()	dict_values(['华北科技学院', 17000, '北京东燕郊'])
items()	返回所有的键值对	d.items()	dict_items([('学校', '华北科技学院'), ('学生数', 17000), ('地址', '北京东燕郊')])
get(key[,default])	返回键对应的值,若键不存在,则返回默认值	d.get("地址")	'北京东燕郊'
setdefault(key[,default])	返回键对应的值,若键不存在,则添加该键值对	d.setdefault("地址")	'北京东燕郊'
pop(key[,default])	返回键对应的值,并删除该键值对,若键不存在,则返回默认值	d.pop("地址")	'北京东燕郊'
popitem()	随机返回一个键值对,并删除该键值对	d.popitem()	('学生数', 17000)
clear()	清除所有的键值对	d.clear()	#输出d的结果{}
update(t)	修改键对应的值,若键不存在,则添加该键值对	d.update(t)	#输出d的结果 {'学校': '华北科技学院', '学生数': 18000, '地址': '北京东燕郊'}

续表

函数和方法	功能说明	示例	结果
copy()	浅复制字典	t=d.copy()	#输出t的结果 {'学校': '华北科技学院', '学生数': 17000, '地址': '北京东燕郊'}
del	删除字典中指定的键值对	del d["地址"]	#输出d的结果 {'学校': '华北科技学院', '学生数': 17000}
in	返回键是否在字典中，是返回True，否则返回False	"学校" in d	True
len(d)	返回字典的长度，即键值对个数	len(d)	3

常用函数举例如下。

get(<key>[,<default]>)函数返回键对应的值，若键不存在，则返回默认值，但不会将键和默认值添加到字典中。例如：

```
>>>d={"学校":"华北科技学院","学生数":17000,"地址":"北京东燕郊"}
>>>d.get("电话","01061591417")
    '01061591417'
>>>d
    {'学校': '华北科技学院', '学生数': 17000, '地址': '北京东燕郊'}
>>>d.setdefault("电话","01061591417")    #键"电话"不存在,添加键值对
    '01061591417'
>>>d
    {'学校': '华北科技学院', '学生数': 17000, '地址': '北京东燕郊', '电话':
'01061591417'}
>>>d.pop("地址")                         #返回键"地址"对应的值,并删除该键值对
    '北京东燕郊'
>>>d.pop("数据来源","华北科技学院")       #键"数据来源"不存在,返回默认值
    '华北科技学院'
>>>d.popitem()                           #随机返回一个键值对,并删除
     ('电话', '01061591417')
>>>d
    {'学校': '华北科技学院', '学生数': 17000}
>>>d1={"学生数":18000,"地址":"北京东燕郊"}
>>>d.update(d1)                          #更新键"学生数"的值,添加"地址"键值对
>>>d
    {'学校': '华北科技学院', '学生数': 18000, '地址': '北京东燕郊'}
>>>del d["电话"]                         #删除不存在的键"电话",输出错误提示
    Traceback (most recent call last):
        File "<pyshell#100>", line 1, in <module>
```

```
        del d["电话"]
KeyError: '电话'
```

【例 3-2】 修改例 3-1,采用字典形式存储学生学号、姓名和两门课成绩,其中 key 为学生的学号,value 为姓名和学生的成绩。

编写程序如下:

```
1  data={'1001':["张一",91,81],'1002':["张二",92,82],'1003':["张三",93,83]}
                                                    #组合存储
2  print(type(data),data)
3  print(type(data.items()),data.items())
4  print(type(data.keys()),data.keys())             #字典的所有 key
5  print(type(data.values()),data.values())         #字典的所有 value
6  mykey='1002'
7  print(mykey,data.get(mykey))                     #获得具体的值
```

运行结果:

```
<class 'dict'>{'1001': ['张一', 91, 81], '1002': ['张二', 92, 82], '1003': ['张三',
93, 83]}
<class 'dict_items'>dict_items([('1001', ['张一', 91, 81]), ('1002', ['张二',
92, 82]), ('1003', ['张三', 93, 83])])
<class 'dict_keys'>dict_keys(['1001', '1002', '1003'])
<class 'dict_values'>dict_values([['张一', 91, 81], ['张二', 92, 82], ['张三',
93, 83]])
1002 ['张二', 92, 82]
```

程序分析: 本例中第 1 行存储的是字典数据结构,key 为学号字符串,value 为列表。列表里面采用混合数据类型存储,姓名为字符串,成绩为整型数值。第 2～5 行为输出字典中不同的特征数据。第 7 行为利用 key 获得具体的值。

3.5 集　　合

3.5.1 集合的定义与赋值

集合类型与数学中集合的概念一致,即包含 0 个或多个元素的无序组合。集合中元素不可重复,元素类型只能是不可变数据类型。集合的基本形式为:

```
{<元素 1>, <元素 2>, … , <元素 n>}
```

集合中元素是无序的,它没有索引和位置的概念,元素打印输出的顺序与开始创建的顺序可能不同。由于集合元素不可重复,使用集合类型能过滤掉重复元素。

set()函数可以转换或创建成一个集合。

```
>>>s={1,2,3,4,5}
>>>s
```

```
    {1, 2, 3, 4, 5}
>>>s[0]                                  #集合不支持索引
    Traceback (most recent call last):
        File "<pyshell#23>", line 1, in <module>
            s[0]
    TypeError: 'set' object does not support indexing
>>>s=set([1,2,3,4,5])                   #set()函数将列表转换成集合
>>>s
    {1, 2, 3, 4, 5}
>>>s=set()                               #创建空集合
>>>s
    set()
>>>type(s)
    <class 'set'>
>>>s={}                                  #创建空字典,不是空集合
>>>type(s)
    <class 'dict'>
```

3.5.2 集合的基本操作

集合的常用函数和方法如表3.6所示,这里假设s={1,2,3,4,5},t={3,9}。另外,集合还提供了数学意义上的交集、并集、差集等运算。

表3.6 集合常用函数和方法

函数和方法	功能说明	示例	输出s结果
add(x)	添加元素x到集合中	s.add(6)	{1, 2, 3, 4, 5, 6}
pop()	随机返回一个元素,并删除该元素	s.pop()	1
discard(x)	删除元素x,若不存在x,不报错	s.discard(3)	{1, 2, 4, 5}
remove(x)	删除元素x,若不存在x,报错	s.remove(3)	{1, 2, 4, 5}
clear()	清除集合中所有元素	s.clear()	set()
update(t)	合并集合t到原集合中,并自动过滤重复元素	s.update(t)	{1, 2, 3, 4, 5, 9}
copy()	复制集合	t=s.copy()	#输出t的结果 {1, 2, 3, 4, 5}
isdisjoint(t)	若集合与t没有相同元素,返回True,否则返回False	s.isdisjoint(t)	#输出结果 False
len(s)	返回集合元素个数	len(s)	#输出结果 5
in	返回元素是否在集合中,是返回True,否则返回False	3 in s	#输出结果 True

```
>>>s={1,2,3,4,5}
>>>s.pop()
```

```
    1
>>>s
    {2, 3, 4, 5}
>>>s.discard(6)                          #删除不存在的 6,不报错
>>>s.remove(6)                           #删除不存在的 6,报错
    Traceback (most recent call last):
        File "<pyshell#58>", line 1, in <module>
            s.remove(6)
    KeyError: 6
```

3.6 列表与其他数据类型的转换

列表属于 Python 语言中最灵活的“胶水”式数据类型,它的特点是可以用来存储任意类型的数据。若在编程中遇到不能访问的数据类型,最简单的解决方案就是转换为列表,然后利用列表进行访问。通常,这种方式可以解决大量问题。因此,掌握列表和不同类型之间的转换,非常关键。

3.6.1 列表与字符串间的转换

1. 字符串转列表

字符串转列表就是按单个字符拆分,每个字符作为列表中的一个元素。例如:

```
>>>x=list("人生苦短、快学 Python")
>>>x
    ['人', '生', '苦', '短', '、', '快', '学', 'P', 'y', 't', 'h', 'o', 'n']
```

2. 列表转字符串

列表转字符串的思路,是把列表中的每个元素连接成一个整体的字符串。这里有一个前提条件,即列表中的每个元素都是字符串才可以进行连接。

```
>>>x1=['abc','A','B']
>>>x2="".join(x1)                       #""代表连接时采用的分隔符,没有字符表示分隔为空
>>>x2
    'abcAB'
```

如果列表中存在非字符串数据,需要先利用 map()函数把所有元素转换为字符串,然后再进行连接。

```
>>>x3=['a','b','c',123,5,6]
>>>x4="".join(x3)
    Traceback (most recent call last):
        File "<input>", line 1, in <module>
    TypeError: sequence item 3: expected str instance, int found
>>>x4="".join(map(str,x3))
>>>x4
```

```
'abc12356'
```

3.6.2 列表与字典间的转换

1. 字典转列表

字典转列表的关键点是 key 和 value 的采集与转换，两者不能同时进行转换，只能一次转换一种，具体方法示例如下：

```
>>>d1={"1001":"wang01","1002":"wang02","1003":"wang03"}
>>>d2=list(d1)
>>>d2
    ['1001', '1002', '1003']
>>>d3=list(d1.keys())
>>>d3
    ['1001', '1002', '1003']
>>>d3=list(d1.values())
>>>d3
    ['wang01', 'wang02', 'wang03']
```

2. 列表转字典

列表转字典的关键点在于确定列表中哪些数据作为 key，哪些数据作为 value。直接转换方式是实现不了的，需要借助于 zip()函数来实现，示例如下：

```
>>>y1=["a","b","c"]
>>>y2=[1,2,3]
>>>y3=zip(y1,y2)
>>>y3
   <zip object at 0x0000015EB70F2648>
>>>type(y3)
   <class 'zip'>                    #zip 类型
>>>y3=dict(y3)
>>>y3
   {'a': 1, 'b': 2, 'c': 3}
>>>y4=dict(zip(y2,y1))
>>>y4
   {1: 'a', 2: 'b', 3: 'c'}
```

zip(x,y)函数是对 x 和 y 中的数据重新进行打包组合，构成新的数据 zip 类型，利用 dict()函数就可以把这个数据转换为字典。

3.7 扩展：random 库

random 库是 Python 的标准函数库，用于产生伪随机数。random 库常用的函数如表 3.7 所示。这里假设已经使用 from random import * 语句引用了该库，且假设 s=[1,2,3,4,5]。

表 3.7 常用随机函数

函 数	功 能 说 明	示例	随机结果
random()	生成一个[0.0,1.0)之间的随机小数	random()	0.2340904626116812
uniform(M,N)	生成一个[M,N]之间的随机小数	uniform(1,10)	6.631482736972486
randint(M,N)	生成一个[M,N]之间的随机整数	randint(1,10)	4
randrange([M,]N[,K])	生成一个[M,N)之间,以K为步长的随机整数。M默认值为0,K默认值为1	randrange(1,10,2)	7
getrandbits(K)	生成一个K比特长度的随机整数	getrandbits(3)	6
choice(s)	从序列s中随机返回一个元素	choice(s)	3
sample(pop,K)	从pop类型中随机选取K个元素	sample(s,2)	[4, 3]
shuffle(s)	将序列s中元素随机排列,返回打乱后的序列	shuffle(s)	[5, 4, 2, 1, 3]

随机函数的用途之一是产生随机数供调试程序使用,这样可以省去人工输入数据的麻烦,同时提高工作效率。但每次运行程序产生的随机数都不相同,不便比较运行结果的正确性。所以若需要产生相同的随机数序列,可以使用seed()函数设置随机数种子,只要种子相同,则每次运行程序产生的随机数序列就相同。

seed(N)函数用于初始化随机数种子,若不指定N,则N的默认值为系统当前时间,且N一般为整数。

```
>>>seed(1)
>>>randint(1,10)
    3
>>>randint(1,10)
    10
>>>randint(1,10)
    2
```

当种子为1时,多次运行randint(1,10)产生的序列为3、10、2、5、2等。

当种子为2时,多次运行randint(1,10)产生的序列为1、2、2、6、3等。

【例3-3】 编写一个随机课堂考勤程序。设学生序号和姓名已存入字典中,请随机确定一个考勤学生,并输出学生序号和姓名。

编写程序如下:

```
1   import random
2   data={1:"张一",2:"张二",3:"张三",4:"张四"}
3   n=len(data)                              #n为学生人数
```

```
4   x=random.randint(1,n)                    #设学生的序号从1开始,且连续
5   print("随机选中的学生为:",x,data[x])
```

运行结果：

```
随机选中的学生为：2 张二
```

程序分析：代码中第2行利用字典存储了学生的信息，其中key为学生的序号，是整型数据。第4行为调用random库的randint()函数产生随机数。第5行利用这个随机数作为字典的key，获得对应的value值。

题目扩展：能否把本例的字典形式修改为列表形式来存储学生信息，并进行随机点名？**提示**：可以利用列表中有序结构的索引值作为查询依据。

实验三　组合数据类型的操作

一、实验目的

(1) 掌握列表的基本操作。

(2) 掌握列表的内置方法和函数的应用。

(3) 掌握字典的内置方法和函数的应用。

(4) 掌握元组、集合的应用。

(5) 掌握random库的使用。

二、实验内容

1. 计算100以内所有奇数的和，并输出结果。

2. 输入任意个整数，用逗号分隔，计算并输出它们的平均值，保留一位小数。

3. 输入10个正整数，将前5个升序排列，后5个降序排列，并输出结果。

4. 随机产生一个6位密码并输出。要求密码中只包含0～9数字和大写英文字母A～Z。例如，8LTXGD、OSXVTR、BKG64J都是随机产生的密码。

5. 使用字典存储学生的学号和姓名。请编程模拟以下过程：(1)设3个同学的信息已存入字典中，如d={1："张三"，2："李四"，3："王五"}。(2)输入一个新同学信息添加到字典中。(3)输入一个学号，查询对应学生的姓名并输出。(4)输入一个学号，修改对应学生的姓名。(5)输入一个学号，删除对应学生的信息。最后输出字典中的所有信息。

习　题　三

一、选择题

1. 下列(　　)类型数据是不可变化的。

A. 集合　　B. 字典　　C. 元组　　D. 列表

2. 下面不属于Python内置对象的是(　　)。

A. int　　B. list　　C. dict　　D. set

3. 已知x = [1, 2]和y = [3, 4]，那么x+y的结果是(　　)。