第 3 章
组合数据类型

为满足程序中复杂的数据表示，Python 提供了组合数据类型。组合数据类型可以将多个基本数据类型或组合数据类型作为一个整体进行操作，能够更清晰地反映数据之间的关系，也能更加方便地管理和操作数据。

在 Python 中，常用的组合数据类型有以下四种。

(1) 列表(list)，是一种有序、可更改的集合，允许重复的成员。

(2) 元组(tuple)，是一种有序且不可更改的集合，允许重复的成员。

(3) 字典(dictionary)，是一个无序、可变、有索引的集合，没有重复的成员。

(4) 集合(set)，是一个无序、可变、无索引的集合，没有重复的成员。

3.1 列表

列表是由一组任意类型的数据组合而成的序列，具有可变对象、可变长度、异构和任意嵌套的特点。列表中的每一个数据称为元素，列表将数据元素放在一对方括号内并以逗号分隔。一个列表中的数据元素可以是基本数据类型，也可以是组合数据类型或自定义数据类型的数据，并且 Python 允许同一个列表中元素的数据类型不同。例如下面的列表都是合法的列表对象：

```
[1,2, 3, 4, 5]
[1, "Hello", 3.14]
[1, "北京", "010", True]
["日期", "中国", [2020, 8, 1]]
```

3.1.1 创建列表

创建列表有两种方法。一种是使用方括号，在方括号内把每一个列表元素用逗号进行分隔，并用赋值运算符将一个列表赋值给变量。具体格式如下：

```
<列表名>=[<元素 1>,<元素 2>,<元素 3>,…,<元素 n>]
```

例如：

```
list1=[]
list2=[1, 2, 3, 4]
list3=["red", "green", "blue"]
```

另一种创建列表的方法是使用 list()函数，例如：

```
list4=list("贵州大学")     #每个字符作为列表中的一个数据元素,即['贵','州','大','学']
list5=list("Hello", "World")
list6=list(range(2, 4))    #将 range()函数产生的序列变为列表,即[2,3,4]
```

列表可以包括不同类型的元素，例如：

```
list7=list(1, "Hello", 3.14)
```

但通常建议列表中元素最好使用相同的数据类型。

列表可以嵌套使用，例如：

```
list8=[list2, list3]
```

运行结果如下：

```
>>>list2=[1, 2, 3, 4]
>>>list3=["red", "green", "blue"]
>>>list8=[list2, list3]          #创建一个嵌套列表
>>>print(list8)
[[1, 2, 3, 4], ['red', 'green', 'blue']]
```

3.1.2　访问列表

1. 使用下标

列表是一个有序序列，可以通过序号或下标来访问列表中的元素。格式如下：

```
<列表名>[<index>]
```

其中列表名为一个列表的名称，index 为访问的列表元素下标或索引。index 的值从0 开始，它可以是一个正数，也可以是一个负数，为正数时表示正向访问列表，为负数时表示逆向访问列表，如图 3-1 所示。

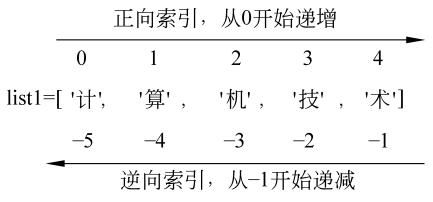

图 3-1　列表的索引序号

如果一个列表的长度是 N，则合法的下标在 0 到 N－1 之间或者在－1 到－N 之间。例如，定义列表 list1 如下：

```
list1=['a', 'b', 'c', 'd', 'e', 'f']
```

则 list1[0]的值为'a'；list1[3]的值为'd'；而 list1[－2]的值为'e'，表示从列表的右侧开始

倒数第 2 个的元素。如果在程序中试图访问下标大于 5 的元素，那么会导致一个运行的错误 IndexError，如下所示：

```
>>>list1=['a', 'b', 'c', 'd', 'e', 'f']
>>>list1[0]                          #访问列表 list1 中正向索引序号为 0 的列表元素
'a'
>>>list1[3]                          #访问列表 list1 中正向索引序号为 3 的列表元素
'd'
>>>list1[-2]                         #访问列表 list1 中逆向索引序号为-2 的列表元素
'e'
>>>list1[8]                          #如果访问的列表序号不存在，则返回索引序号错误
…
IndexError: list index out of range   #如果访问的列表序号不存在，则返回索引序号错误
```

2. 列表切片

列表切片可以从列表中取得多个元素并组成一个新列表。格式如下：

```
<列表名>[<start>: <end>: <step>]
```

说明：

(1) 切片就是在序列中划定一个区间(start：end)，并按步长 step 选取元素，但不包括 end 下标指示的元素；

(2) 步长的默认值为 1，即不指定步长，就是获取指定区间中的每个元素，但不包括终止下标指示的元素；

(3) 起始下标 start 和终止下标 end 缺省或表示为 None，分别默认为列表起点和终点；

(4) 起始在左、终止在右时，步长应为正；起始在右、终止在左时，步长应为负，否则切片为空。

切片操作举例如下：

```
>>>newList=['a', 'b', 'c', 'd', 'e', 'f']
>>>newList[2: 4]      #访问列表 newList 中正向索引序号从 2 到 4(不包括)的列表元素
['c', 'd']
>>>newList[3: ]       #访问列表 newList 中正向索引序号从 3 到最后的列表元素
['d', 'e', 'f']
>>>newList[: -3]      #访问列表 newList 中从开始到逆向索引序号为-3(不包括)的列表元素
['a', 'b', 'c']
>>>newList[: : 2]     #访问列表 newList 中正向索引序号从开始到最后且步长为 2 的列表元素
['a', 'c', 'e']
>>>newList[-4: -2]    #访问列表 newList 中反向索引序号从-4 到-3 的列表元素
['c', 'd']
>>>newList[4: 1: -2]  #访问列表 newList 中正向索引序号从 4 到 2 且步长为-2 的列表元素
['e', 'c']
>>>newList[-1:: -1]   #访问列表 newList 中反向索引序号从-1 到-6 的列表元素，即逆序列表
['f', 'e', 'd', 'c', 'b', 'a']
>>>newList[4: 1: 2]   #正向索引：起始在右、终止在左，步长为正，则返回空列表
[]
```

3.1.3　更新列表

列表是一种可变的数据类型，列表的长度和列表元素的值都是可以更改的。更新列表主要有修改列表元素、添加列表元素和删除列表元素等操作。

1. 修改列表元素

修改列表元素只需索引需要修改的元素并对其赋新值即可。具体格式如下：

```
列表名[<index>]=<值>
```

其中，列表名为一个已经存在的列表，index 为该列表的正向或逆向索引序号，值为任意数据值。例如：

```
>>>color=['red', 'green', 'blue', 'white']
>>>color
['red', 'green', 'blue', 'white']
>>>color[1]=['绿色']              #将列表 color 中索引序号为 1 的元素的值改为'绿色'
>>>color
['red', '绿色', 'blue', 'white']
>>>color[0], color[2]='红色', '蓝色'
                          #将列表 color 中下标为 0 和 2 的元素值分别改为'红色'和'蓝色'
>>>color
['红色', '绿色', '蓝色', 'white']
>>>color[0: 2]=['R', 'G']         #当索引为一个范围时，值也需是列表
>>>color
['R', 'G', '蓝色', 'white']
```

在修改列表元素时，当列表序号范围和赋值列表长度不相等时，可以增加或删除列表。例如：

```
>>>list=[1, 2, 3, 4, 5]
>>>list
[1, 2, 3, 4, 5]
>>>list[0: 2]=['one', 'two', 'three', 'four']   #用 4 个列表元素替换选定的两个列表元素
>>>list
['one', 'two', 'three', 'four', 3, 4, 5]
>>>list[-3: ]=['five']                          #用一个列表元素替换选定的 3 个列表元素
>>>list
['one', 'two', 'three', 'four', 'five']
```

2. 添加列表元素

虽然可以通过对列表赋值来添加列表元素，但通常还是使用专门的内置方法对列表进行添加元素的操作。常用添加元素的方法有多个，如表 3-1 所示。

表 3-1　添加元素方法

方　法	说　明
列表名.append(obj)	在列表末尾添加元素
列表名.insert(index, obj)	在列表中的 index 索引序号处插入元素
列表名.extend(序列)	在列表末尾一次性添加另一个序列中的多个元素

例如：

```
>>>color=['白', '黑', '红']
>>>color.append('蓝')
>>>color
['白', '黑', '红', '蓝']
>>>color.insert(0, '绿')
>>>color
['绿', '白', '黑', '红', '蓝']
>>>color.extend(['黄', '紫'])
>>>color
['绿', '白', '黑', '红', '蓝', '黄', '紫']
```

注意：使用 append()和 insert()方法每次只能插入一个列表元素。

3. 删除列表元素

删除列表元素，可以使用 del 语句。格式如下：

```
del <列表名>[<索引>]或者 del <列表名>
```

其中，列表名为一个已经存在的列表名称，索引为列表索引序号，此时 del 语句表示将删除列表中对应索引序号的列表元素。如果 del 语句后省略了索引序号，则表示将删除整个列表。

```
>>>color=['白', '黑', '红', '蓝', '黄']
>>>del color[2]          #删除列表中索引序号为 2 的列表元素
>>>color
['白', '黑', '蓝', '黄']
>>>del color[: 2]        #删除列表中 0,1 索引序号的列表元素
>>>color
['蓝', '黄']
>>>del color             #删除列表 color
>>>color                 #列表删除后,再次使用列表,将出现"未定义"错误
…
NameError: name 'color' is not defined
```

删除列表元素还可以使用内置方法，如表 3-2 所示。

表 3-2　删除元素方法

方　法	说　明
列表名.remove(obj)	删除列表中第一个和 obj 值相等的元素。如果列表中有 1 个以上相同的列表元素，则需多次使用 remove()方法
列表名.pop(index)	删除列表中索引序号为 index(默认值为－1)的元素，并且返回该元素的值
列表名.clear()	删除列表中所有元素

例如：

```
>>>color=['白', '蓝', '绿', '黑', '黄', '蓝']
>>>color.remove('蓝')              #remove()方法使用一次只能删除一个"蓝"
>>>color
['白', '绿', '黑', '黄', '蓝']
>>>color.pop(2)                    #删除列表 color 中下标为 2 的元素并返回该元素的值
'黑'
>>>print(color)
['白', '绿', '黄', '蓝']
>>>color.clear()                   #删除列表中的所有列表元素
>>>color
[]
```

3.1.4　列表常用的其他操作

1. 列表的拼接和复制

在 Python 中，可以使用运算符＋来连接两个列表，并返回一个新列表。例如：

```
>>>list1=[1, 2]
>>>list2=[3, 4]
>>>list3=list1+list2
>>>list3
[1, 2, 3, 4]
```

复制列表可以使用 copy()方法，例如：

```
>>>list1=[1, 2, 3, 4]
>>>list2=list1.copy()              #复制列表 list1 并返回一个新的列表
>>>print(list1,list2)
[1, 2, 3, 4][1, 2, 3, 4]
```

使用运算符“＊”可以将一个列表复制若干次后形成一个新的列表。例如：

```
>>>list1=[1, 2]
>>>list2=list1＊2
>>>list2
[1, 2, 1, 2]
```

list1＊2 和 2＊list1 相同。

注意：列表也可以像变量之间的赋值那样，将一个列表的值赋给另一个列表，但是和基本变量赋值不同的是，列表的赋值只是将实际数据的地址引用进行了赋值，而不是将实际数据赋值一份给新的列表。例如：

```
>>>x1=100
>>>x2=x1
>>>x1, x2
(100, 100)                          #两个变量的值相等
>>>x2=200
>>>x1, x2
```

```
(100, 200)                          #更改一个变量的值，另一个不会变
>>>list1=[1, 2, 3, 4]
>>>list2=list1                      #将 list1 赋值给 list2
>>>list2[0]=5                       #更改 list2 中的某些元素值
>>>list2[1]=6
>>>list1, list2
([5, 6, 3, 4], [5, 6, 3, 4])       #list1 和 list2 同时改变。可以看出，list2 使用了 list1 的
                                     存储地址。当改变 list2 的元素，list1 的元素同时改变
>>>list2=[7, 8, 9, 10]             #对 list2 进行实际数据的赋值，list2 会使用独立的存储地
                                     址，不再和 list1 关联
>>>list1, list2
([5, 6, 3, 4], [7, 8, 9, 10])
```

2. 列表的遍历

对列表中的每个元素均做一次访问称为对列表的一次遍历。通过 while 循环依次访问列表的各个下标就可以实现对列表的一次访问。例如：

```
>>>list=[1, 2, 3]
>>>i=0
>>>while(i<len(list)):
        print (list[i])
        i+=1
1
2
3
```

其中，len()函数是 Python 的内置函数，其功能是返回列表的元素个数。

也可以使用 for 循环实现列表的遍历，这样就可以在不使用下标变量的情况下顺序遍历列表。例如：

```
>>>list=[1, 2, 3]
>>>for ele in list:
        print(ele)
1
2
3
```

通过灵活地运用循环结构，可以以不同的方式来访问列表中的元素。例如，下面的代码输出列表中所有偶数下标的元素：

```
>>>list=[1, 2, 3]
>>>for i in range(0, len(list), 2):
        print(list[i])
1
3
```

3. in/not in 运算符

使用 in/not in 运算符可以判断一个元素是否在列表中。例如，列表 list 包含的元素为

[1,3,5,7],变量 i 的值为 4,那么表达式 i in list 的返回值为 False,而表达式 i not in list 的返回值为 True,具体代码如下:

```
>>>list=[1, 3, 5, 7]
>>>i=4
>>>print(i in list)
False
>>>print(i not in list)
True
```

4. 列表的比较

对列表的比较可以使用关系运算符(<、>、==、<=、>=、!=)。两个列表的比较规则如下:比较两个列表的第一个元素,如果元素相同,则继续比较下面两个元素;如果两个元素不同,则返回两个元素的比较结果;一直重复这个过程直到有不同的元素或比较完所有的元素为止。例如:

```
>>>list1=["vb", "java", "C++"]
>>>list2=["C++", "vb", "java"]
>>>print(list1>list2)
True
```

说明:字符串之间的比较是按正向下标,从 0 开始,以对应字符的码值(如 ASCII 码值)作为依据进行的,直到对应字符不同,或所有字符都相同,才能决定大小,或是否相等。

5. 列表推导式

利用列表推导式可以使用非常简捷的方式生成满足特定需要的列表。语法格式如下:

```
[<表达式>for <变量>in <序列>]
```

例如:

```
>>>list1=[x * x for x in range(1, 10)]
>>>list1
[1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>list2=[i for i in list1 if i%2==0]
>>>list2
[4, 16, 36, 64]
```

说明:

(1) list1 是由 10 以内的自然数的平方组成的列表;

(2) list2 是由 list1 中的偶数组成的列表。

列表推导式还可以在第一个 for 子句后面添加 for 子句或 if 子句。例如:

```
>>>list3=[1, 2]
>>>list4=[2, 4]
>>>list5=[x+y for x in list3 for y in list4 if x!=y]
>>>list5
[3, 5, 6]
```

这里的 list5 将包含 list3 和 list4 列表中所有不相等的元素的和。

6. 列表元素排序

可以使用内置函数 sorted()返回一个元素排序后的列表。该函数的格式如下：

```
sorted(<列表名>[, <key>=<排序属性>][, <reverse>=False/True])
```

说明：

(1) 排序的前提是元素间可以相互比较，用术语 iterable(可迭代)表示。若一个序列中有不可相互比较的元素，就不可排序。

(2) 一个列表中的元素对象可以有许多属性，要用 key 指定按照哪个属性排序。例如对字符串可以指定 str.lower，即按小写字母表顺序(不区分大小写)排序。通常，对于字符串元素以及数值型元素对象，key 项可以缺省，默认按照数值排序。对于字符串对象，按照编码值(如 ASCII 码值)排序。

(3) sorted()函数默认按照升序排序，但可以用 reverse 的取值为 True/False，决定是否反转，reverse 默认值为 False。

(4) sorted()返回一个列表。

例如：

```
>>>list1=['hello', 'World', "Ok", "abc"]
>>>list2=[4, 2, 8, 3]
>>>list3=["e", 1, "a", "n", 2]
>>>sorted(list1, key=str.lower)       #将 list1 列表元素按小写字母表顺序升序排序
['abc', 'hello', 'Ok', 'World']
>>>sorted(list2)                      #将 list2 列表元素按升序排序
[2, 3, 4, 8]
>>>sorted(list2, reverse=True)        #将 list2 列表元素按降序排序
[8, 4, 3, 2]
>>>sorted(list3)                      #错误，list3 中含有不可比较的元素
Traceback (most recent call last):
  File "<pyshell #16>", line 1, in <module>
    sorted(list2)
TypeError: '<' not supported between instances of 'int' and 'str'
```

3.1.5 列表的内置函数与其他方法

1. 列表相关的内置函数

Python 中的一些内置函数为列表的使用提供了便利。这些内置函数有 len()、max()、min()和 sorted()等，如表 3-3 所示。

表 3-3 列表的常用内置函数

函 数	说 明
len(list1)	返回 list1 中列表元素的个数
max(list1)	返回 list1 列表中元素的最大值，要求 list1 中列表元素类型相同

续表

函　数	说　　明
min(list1)	返回 list1 列表中元素的最小值，要求 list1 中列表元素类型相同
reversed(list1)	返回 1 个新列表，新列表将 list1 翻转，即第 1 个元素与最后一个对换；第 2 个与倒数第 2 个对换，以此类推
sum(list1)	如果 list1 中所有列表元素都是数字，则函数返回列表元素之和
list(tuple)	将元组 tuple 转换为列表

注意：以上函数不仅能应用到列表上，还能应用于所有可迭代对象，如后面要介绍的元组、字典、集合等。

2. 列表的其他方法

列表常用的其他方法有 index()、count()等，如表 3-4 所示。

表 3-4　列表的其他方法

方　法	说　　明
list1.index(x)	返回列表 list1 中第一个值为 x 的元素的索引(下标)，如果没有这样的元素则会报错
list1.count(x)	返回列表 list1 中 x 出现的次数
list1.sort()	将列表 list1 进行升序排序。如果需要对列表进行降序排序，则可加参数 reverse＝True，例如：list1.sort(reverse＝True)
list1.reverse()	将翻转列表 list1 中的所有元素位置
list1.copy()	复制列表

3.1.6　二维列表

列表的元素可以是任何类型的对象，当然也可以是列表。如果一个列表中的列表元素也是由列表构成，那就构成了类似矩阵的二维列表。二维列表可以理解为一个由行组成的列表。例如，图 3-2 中定义的名为 list1 的列表是一个长度为 3 的列表，其中的每一个元素又是一个列表。

```
list1=[
  [1, 2, 3, 4, 5]
  [6, 7, 8, 9, 10]
  [11, 12, 13, 14, 15]
  ]
```

	[0]	[1]	[2]	[3]	[4]
[0]	1	2	3	4	5
[1]	6	7	8	9	10
[2]	11	12	13	14	15

图 3-2　二维列表

二维列表的每一行可以使用索引号或下标来访问，称为行下标。例如图 3-2 中的二维列表 list1 中，list1[0]的值即是列表[1，2，3，4，5]。每一行中的值又可以通过列下标来访问。

访问二维列表的格式如下：

```
<列表名>[<索引 1>][<索引 2>]
```

其中，索引 1 为二维列表的元素索引号，即行下标，索引 2 为二维列中索引 1 指向的列表元素中的元素索引号，即列下标。例如：

```
>>>a=[1, 2, 3, 4, 5]
>>>b=[6, 7, 8, 9, 10]
>>>c=[11, 12, 13, 14, 15]
>>>list1=[a, b, c]
>>>list1[0]
[1, 2, 3, 4, 5]
>>>list1[1][3]
9
```

语句 list1[1][3]中，[1]中的 1 代表访问的是 list1 列表的索引序号为 1 的列表元素，即列表 b；3 代表访问列表 b 中索引序号为 3 的列表元素，即数字 9。

要遍历一个二维列表，一般需要使用两层嵌套的循环结构来实现。其中，外层循环遍历每一行，内层循环遍历一行的每个元素，如程序 L3.1 所示。

【例 3.1】 遍历二维列表。

```
#L3.1 例 3.1
stu1=['2001', '李华', '男', '19']
stu2=['2002', '王燕', '女', '18']
stu3=['2003', '陈强', '男', '20']
stu4=['2004', '赵晓红', '女', '19']
list=[stu1, stu2, stu3, stu4]
for i in range(len(list)):
    for j in range(len(list[i])):
        print(list[i][j], end=" ")        #end=""负责输出每项数据后加个空格
    print()                               #输出一行数据后换行
```

程序运行结果如下：

```
2001 李华 男 19
2002 王燕 女 18
2003 陈强 男 20
2004 赵晓红 女 19
```

说明：

(1) 列表 list1 到 list4 为 4 名同学的基本信息，列表 list 是包含了以上 4 个列表的二维列表；

(2) len(list)用来求列表 list 的列表元素个数，range(len(list))可以返回 list 中列表元素的正向索引序号；

(3) 在双重循环中，外层循环遍历列表 list 中的每个列表元素，内层循环遍历 list[i]中的每个列表元素；

(4) print()作用为输出一行数据后换行。

在 Python 中也可以建立更高维度的列表。三维列表就需要三个下标来确定一个元素,也需要三层嵌套的循环结构来遍历整个列表。

3.1.7　列表应用举例

【例 3.2】 输入一行字符,分别统计其中英文字母、空格、数字和其他字符的个数。

【解析】 首先是输入一个字符串,根据字符串中每个字符的 ASCII 码值判断其类型。数字 0～9 对应的 ASCII 码值为 48～57,大写字母 A～Z 对应的 ASCII 码值为 65～90,小写字母 a～z 对应的 ASCII 码值为 97～122。使用 ord()函数将字符转换为 ASCII 码值。可以先找出各类型的字符,放到不同的列表中,再分别计算各个列表的长度,代码如下。

【例 3.2】 统计各类字符的个数。

```
#L3.2 例 3.2
a_list =list(input('请输入一行字符: '))
letter=[]
space=[]
numb=[]
other=[]
for i in range(len(a_list)):
    if ord(a_list[i]) in range(65, 91) or ord(a_list[i]) in range(97,123):
        letter.append(a_list[i])
    elif a_list[i] =='':
        space.append(' ')
    elif ord(a_list[i]) in range(48, 58):
        numb.append(a_list[i])
    else:
        other.append(a_list[i])
print('英文字母个数: %s' %len(letter))
print('空格个数: %s' %len(space))
print('数字个数: %s' %len(numb))
print('其他字符个数: %s' %len(other))
```

程序运行结果如下:

```
请输入一行字符: dfu12 * &^jif
英文字母个数: 6
空格个数: 2
数字个数: 2
其他字符个数: 3
```

目前,很多网站都引入了验证码技术,以有效地防止用户利用机器人自动注册、登录、刷票等。验证码一般是包含一串随机产生的数字或符号以及一些干扰元素(如数字直线、若干圆点、背景图片等)的图片。用户通过肉眼观察验证码,输入其中的数字或符号并提交给网站验证。

常见的 6 位验证码: Kk64ul、eOGpUz、4Gfs81。

【例 3.3】 随机生成一组 6 位验证码,验证码的每个字符可以是大写字母、小写字母或

数字，有且只能是这三种类型的一种，具体生成哪种类型的字符是随机的。

【解析】 6 位验证码功能需随机生成 6 个字符，可以将每个字符临时存放在列表中，因为列表是可变且有顺序的。通过列表实现 6 位验证码功能的基本实现思路是：

(1) 创建一个空列表；

(2) 生成 6 个随机字符逐个添加到列表中；

(3) 将列表元素拼接成字符串。

上述思路的步骤(2)是验证码功能的核心。为确保每次生成的字符类型只能是大写字母、小写字母和数字的任一种，可使用 1、2、3 分别代表这三种类型：若产生随机数 1，表示生成大写字母；若产生随机数 2，表示生成小写字母；若产生随机数 3，表示生成数字。

为确保每次生成的是大写字母、小写字母或数字类型的字符，可以根据数值范围或字符的 ASCII 码控制每个类型中包含的所有字符：数字对应的数值范围为 0～9；大写字母对应的 ASCII 码范围为 65～90；小写字母对应的 ASCII 码为 97～122。之后再从这些字符中随机选择一个字符即可，具体代码如下。

【例 3.3】 随机生成 6 位验证码。

```
#L3.3 例 3.3
import random                              #导入随机模块
code_list=[]
for i in range(6):                         #控制验证码的位数
    state=random.randint(1,3)              #随机生成字符分类
    if state==1:
        kind1=random.randint(65,90)
        uppercase=chr(kind1)               #随机生成大写字母
        code_list.append(uppercase)
    elif state==2:
        kind2=random.randint(97,122)
        lowercase=chr(kind2)               #随机生成小写字母
        code_list.append(lowercase)
    elif state==3:
        numb=random.randint(0,9)           #随机生成数字
        code_list.append(str(numb))
code="".join(code_list)                    #将列表元素连接成字符串
print(code)
```

程序运行结果如下：

```
T90eWg
```

3.2 元组

Python 中的元组与列表非常相似，不同的是元组中的元素是不可变的，即元组一旦创建，其元素不可修改，也不能添加或者删除元素。元组由用逗号分隔的若干元素组成。一个元组中的数据元素可以是基本数据类型，也可以是组合数据类型或自定义数据类型的数据。例如，下面的元组都是合法的：

```
(1, 2, 3, 4, 5)
("a", 3.5, True)
()
(("red", "green", "blue"), (1, 2, 3))
```

3.2.1　创建元组

创建元组可以使用一对小括号，在小括号内用逗号分隔元组元素，具体格式如下：

```
<元组名>=(<元素 1>, <元素 2>, <元素 3>, …, <元素 n>)
```

例如：

```
tuple1=(1, 2, 3, 4)
tuple2=()                #创建一个空元组
tuple3=('a', 'b', 'c')
tuple4=(True, )          #创建只有一个元素的元组
```

注意：只含有一个元素的元组，元素后面一定要有逗号，否则就是一个表达式，而不是元组了；含有两个或者两个以上元素的元组，最后一个元素后可以有逗号，也可以没有。

使用小括号创建元组时，小括号也可以省略，例如：

```
tuple5=1, 2, 3, 4, 5
```

另一种创建元组的方法是使用 tuple()函数。例如：

```
tuple6=tuple()               #产生一个空元组，等价于 tuple6=()
tuple7=tuple(range(1, 8, 2)) #将 range 函数产生的序列变为元组，即(1,3,5,7)
tuple8=tuple("贵州大学")     #每字符作为元组中的一个数据元素，即('贵','州','大','学')
```

3.2.2　访问元组

访问一个元组和访问列表的方法类似，可以使用索引序号即下标来访问元组元素，也可以使用切片访问多个元素。例如：

```
>>>tuple1=('a', 'b', 'c', 'd', 'e', 'f')
>>>tuple1[3]          #访问元组 tuple1 中正向索引序号为 3 的元素
'd'
>>>tuple1[-2]         #访问元组 tuple1 中逆向索引序号为-2 的元素
'e'
>>>tuple1[2: 4]       #访问元组 tuple1 中正向索引序号从 2 到 3 的元素
('c', 'd')
>>>tuple1[1: -3]      #访问元组 tuple1 中从正向索引序号为 1 到逆向索引序号为-4 的元素
('b', 'c')
```

注意：元组属于不可变序列，一旦创建，元组中的值就固定不可变了，也无法为元组增加元素或删除元素，所以不能通过索引下标或切片修改元组中的元素。

另外元组的切片还是元组，就像列表的切片还是列表一样。

3.2.3 元组的常用操作

1. 元组的连接

和列表类似，元组可以使用运算符＋来连接两个元组，并返回一个新元组。例如：

```
>>>t1=(1, 2)
>>>t2=(3, 4)
>>>t3=t1+t2
>>>t3
(1, 2, 3, 4)
```

使用运算符 * 可以将一个元组复制若干次后形成一个新的元组。例如：

```
>>>t4=(1, 2)
>>>t5=t4 * 2
>>>t5
(1, 2, 1, 2)
```

注意：元组不支持 copy()方法。

2. 元组的遍历

元组的遍历和列表的遍历类似，可以通过 for 循环或 while 循环实现。例如：

```
>>>t1=(1, 2, 3)
>>>for i in t1:
        print(i)
1
2
3
```

3. 删除元组

由于元组中的元素是不可变的，也就是不允许删除元素，但可以使用 del 语句删除整个元组。格式如下：

```
del <元组名>
```

例如：

```
>>>t1=(1, 2, 3)
>>>del t1
```

4. 元组与列表的转换

使用 list()函数可以将元组转换为列表，而使用 tuple()函数可以将列表转换为元组。例如：

```
>>>list1=[1, 2, 3]
>>>t1=tuple(list1)
>>>print(t1)
(1, 2, 3)
```

元组的其他常见操作与列表类似，具体可参考本书 3.1.4 节，这里不再赘述。

5. 元组的常用函数和方法

元组的常用函数和方法与列表的基本一致，具体用法可参考本书 3.1.5 节中的表 3-3 和表 3-4。需要注意的是，元组不支持 sort()、reverse()和 copy()方法。

3.2.4　元组与列表的比较

元组和列表都属于序列，两者的创建和使用的方法非常类似。元组和列表的不同之处主要体现在以下几个方面。

(1) 列表属于可变序列，可以随意修改列表中元素的值以及对列表进行增加和删除元素操作；而元组则属于不可变序列，元组中的元素一旦定义就不允许进行增加、删除和替换操作。因此，tuple 类没有提供 append()、insert()、remove()等成员函数。在使用下标访问或切片操作时，也只允许读取元组中的值而不能对其进行修改。

(2) 元组的访问速度和处理速度比列表更快。如果所需要定义的序列内容不会对其进行修改，这种情况一般使用元组。

(3) 使用元组可以使元素在实现上无法被修改，从而使代码更加安全。

(4) 作为不可变序列，元组可以用作字典的键，而列表不可以充当字典键，因为列表是可变的。

3.3　字典

在 Python 中，字典属于映射类型，由键值对组成，通过键来实现对元素的存取。在字典中“键”的作用就是索引，就像列表用整数做索引一样。列表的一个索引对应列表中的一个数据，字典中的一个“键”对应着字典中的一个数据。一个键值对就形成字典中的一个条目。

在一个字典结构中，键是唯一的，但值不唯一。这就好比在一个学校里，一个学生只能有唯一的学号，但是不同学号的学生的姓名有可能是相同的。字典将键值对放在一对大括号内，并使用逗号作为分隔，每个键值对内部用冒号分隔。例如下面的字典都是合法的：

```
{'a': 1, 'b': 2, 'c': 3}
{}
{'203001': '张三', '203002': '李四', '203003': '张三'}
```

3.3.1　创建字典

创建字典可以采用以下几种方法。

1. 使用赋值语句创建字典

使用赋值语句创建字典的格式如下：

```
字典名={<键 1>: <值 1>, <键 2>: <值 2>, <键 3>: <值 3>, …, <键 n>: <值 n>}
```

其中，字典的键可以是任何不可变类型(如数字、字符串、元组等)，列表、字典、集合等可变类型不能作为键，值可以是任意类型。

例如：

```
>>>d1={}                    #创建一个空字典
>>>d1
{}
>>>d2={'A': 90, 'B': 80, 'C': 70, 'D': 60}
>>>d2
{'A': 90, 'B': 80, 'C': 70, 'D': 60}
```

在创建字典时，如果同一个键被两次赋值，那么第一个值无效，第二个值被认为是该键的值。例如：

```
>>>d3={'A': 90,'B': 80,'B': 70}
>>>d3
{'A': 90, 'B': 70}
```

这里的键 B 生效的值是 70。

2. 使用 dict()函数创建字典

使用 dict()函数可以将键值对形式的列表创建为字典，也可以将键值对形式的元组创建为字典。例如：

```
>>>item1=[('A', 90), ('B', 80), ('C', 70), ('D', 60)]
                                          #定义 item1 为一个列表
>>>d4=dict(item1)                        #通过 dict()函数将列表 item1 创建为字典
>>>d4
{'A': 90, 'B': 80, 'C': 70, 'D': 60}
>>>item2=(('A', 90), ('B', 80), ('C', 70), ('D', 60))   #定义 item2 为一个元组
>>>d5=dict(item2)                        #通过 dict()函数将元组 item2 创建为字典
>>>d5
{'A': 90, 'B': 80, 'C': 70, 'D': 60}
```

使用 dict()函数还可以通过设置关键字参数创建字典，例如：

```
>>>d6=dict(A=90, B=80, C=70)
>>>d6
{'A': 90, 'B': 80, 'C': 70}
```

3. 调用 fromkeys()方法创建字典

通过调用 fromkeys()方法可以创建值相同的字典。例如：

```
>>>d7={}.fromkeys(['A', 'B', 'C', 'D'], "大于 60 分")
>>>d7
{'A': '大于 60 分', 'B': '大于 60 分', 'C': '大于 60 分', 'D': '大于 60 分'}
```

调用 fromkeys()方法也可以不指定值，此时创建的字典默认为 None 空值。例如：

```
>>>d8={}.fromkeys(['A', 'B', 'C', 'D'])
>>>d8
{'A': None, 'B': None, 'C': None, 'D': None}
```

注意：字典是一个无序序列，在内部存储时，不一定与创建字典的键值对顺序相同，所以在访问字典时，不需要特别关注显示的前后顺序。

字典与列表比较，有以下几个特点。

(1) 字典通过用空间来换取时间，其查找和插入的速度极快，运行时间不会随着键的增加而增加。

(2) 字典需要占用大量的内存。

(3) 字典是无序的对象集合，字典中的元素(即值)是通过键来存取的，而不是通过下标(索引)来存取的。

3.3.2　访问字典

访问字典中的值可以使用[]运算符，在[]中指定键，就能直接访问到对应的数据。格式如下：

```
<字典名>[<key>]
```

通过这个方式可以获得数据，也可以修改数据。例如：

```
>>>score={'张三': 87, '李四': 75}          #创建字典 score
>>>score
{'张三': 87, '李四': 75}
>>>score['李四']                           #访问 score 中键为'李四'的值
75
>>>score['李四']=90                        #将 score 中键为'李四'的值修改为 90
>>>score['李四']
90
```

对字典内的数据赋值时，如果那个键不存在，就直接给字典添加一个新的键对值元素。例如在上面的 score 字典中，加入如下代码：

```
>>>score['王五']=70
>>>score
{'张三': 87, '李四': 90, '王五': 70}
```

此时，score 中就有了三个条目。因此，字典可以直接使用[]运算符来添加元素。在添加元素过程中，如果加入的键是已经存在的，那么新的数据会将之前的数据覆盖。

有时不确定字典中是否存在某个键而又想访问该键对应的值，则可以通过 get()方法实现。格式如下：

```
<字典名>.get(<key>[, <default>])
```

其中，字典名为一个字典的名称，key 是字典中的关键字。这个方法将返回关键字 key 所对应的值，否则返回参数 default 的值。如果没有设置可选参数 default，则默认返回 None。例如：

```
>>>score={'张三': 96, '李四': 75, '王五': 62}
>>>score.get('李四', '键不存在')   #字典 score 中存在键'李四',则返回对应的值
75
>>>print(score.get('甲'))          #字典 score 中不存在键'甲',返回 None,而不是报错
None
>>>score.get('陈六', '键不存在')
                          #字典中不存在键'陈六',返回指定值'键不存在',即第二个参数键不存在
```

使用[]运算符和 get(key)方法不同的地方：如果字典中不存在键 key，[]会抛出 KeyError 异常，而 get(key)方法则返回一个特殊的 None 值。

3.3.3 更新字典

字典是一个可变对象，和列表类似，可以对字典进行修改元素、添加元素、删除元素和字典等操作。

1. 修改元素

可以使用赋值语句修改字典中元素的值。例如：

```
>>>score={'张三': 87, '李四': 75}
>>>score
{'张三': 87, '李四': 75}
>>>score['张三']=96         #将字典 score 中键为'张三'的值改为 96
>>>score
{'张三': 96, '李四': 75}
```

2. 添加元素

使用赋值语句也可以向字典中添加元素。例如：

```
>>>score={'张三': 87, '李四': 75}
>>>score
{'张三': 87, '李四': 75}
>>>score['王五']=62      #如果访问的键在原字典中不存在,则添加一个键值对元素
>>>score
{'张三': 87, '李四': 75, '王五': 62}
```

3. 删除元素和字典

删除字典元素或字典可以使用 del 语句，格式如下：

```
del <字典名>[<键>]或者 del <字典名>
```

其中，字典名为一个已经存在的字典名称，键为字典中存在的关键字，此时 del 语句表示将删除字典中对应键的字典元素。如果 del 语句后省略了键参数，则表示将删除整个字典。例如：

```
>>>score={'张三': 96, '李四': 75, '王五': 62}
>>>del score['张三']          #删除字典 score 中键为'张三'的键值对元素
>>>score
{'李四': 75, '王五': 62}
>>>del score                  #删除字典 score
```

删除字典元素还可以使用 pop()方法，格式如下：

```
<字典名>.pop(<key>[, <default>])
```

如果字典中存在键 key，则 pop()方法将删除 key 所在的条目并返回该条目的值，否则将返回参数 default 的值。如果没有设置可选参数 default 且 key 不在字典中，则提示一个 KeyError 异常。例如：

```
>>>score={'张三': 96, '李四': 75, '王五': 62}
>>>score.pop('张三')
96
>>>score.pop('陈六', '键不存在')
'键不存在'
>>>score.pop('陈六')
…
KeyError: '陈六'
```

3.3.4　字典常用的其他操作

1. 合并字典

使用 update()方法可以将一个字典中的元素添加到当前字典中，如果两个字典的键有重名，则用另一个字典中的值对当前字典的值进行更新，例如：

```
>>>d1={'a': 1,'b': 2}
>>>d2={'c': 3, 'd': 4}
>>>d3={'b': 11}
>>>d1.update(d2)      #如果 d2 和 d1 的键没有重复，则将 d2 中的所有元素添加到 d1
>>>d1
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>>d1.update(d3) #如果 d3 中有和 d1 重复的键，则用 d3 中重复键'b'的值 11 更新 d1 中键'b'的值
>>>d1
{'a': 1, 'b': 11, 'c': 3, 'd': 4}
```