第 3 章
组合数据类型

本章学习目标

- 熟悉组合数据类型的特点和分类
- 熟练掌握字符串、列表和元组类型的创建方法、操作符、操作方法
- 掌握集合的创建方法、操作符、操作方法
- 熟练掌握字典的创建方法、操作符、操作方法
- 运用组合数据类型解决数据存储问题


本章源代码

本章首先介绍组合数据类型的特点和分类，然后分别介绍字符串类型、列表类型、元组类型、集合类型和字典类型的常用操作和常用方法，同时介绍各数据类型的应用。其次介绍正则表达式库，并通过两个医学实践案例介绍组合数据类型的应用。最后，提出两个探索问题，引导读者思考。

3.1 组合数据类型概述

简单数据类型能够存储单个数据，当要存储一系列数据时，则需要定义多个简单数据类型变量。另外，数值类型仅能存放数值型数据，无法存放文本数据。在 Python 中，使用组合数据类型可以将多个数据存储在一个变量中。

按照数据的组织方式，组合数据类型可以分为序列类型、集合类型和映射类型。序列类型常见的有字符串、元组和列表；集合类型常见的有集合；映射类型常见的有字典。根据数据是否可修改，又可将组合数据类型分为可变类型和不可变类型，如表 3.1 所示。

表 3.1 组合数据类型分类

	分 类	细 类	特 点
组合数据类型	序列类型	字符串	不可变类型
		元组	不可变类型
		列表	可变类型
	集合类型	集合	可变类型
	映射类型	字典	可变类型

不可变类型的变量在第一次赋值声明时，会在内存中开辟一块空间，用来存储这个变量被赋予的值，此时变量的值就与开辟的内存空间绑定了，开发者不能修改存储在内存中的值。如果给变量重新赋值，就会开辟一块新的内存空间存储新的值。因此，不可变数据类型的值发生变化，地址也会变，如例 3.1 所示。

【例 3.1】　不可变类型的变量前后赋值的地址变化。

```
#例 3.1
a=2
b=2
print("a 的原地址是：{},b 的原地址是：{}".format(id(a),id(b)))
a=3
b=5
print("a 的新地址是：{},b 的新地址是：{}".format(id(a),id(b)))
str1="abc"
str2="abc"
print("str1 的原地址是：{},str2 的原地址是：{}".format(id(str1),id(str2)))
str1="abdf"
str2="abff"
print("str1 的新地址是：{},str2 的新地址是：{}".format(id(str1),id(str2)))
```

代码执行结果如下：

```
a 的原地址是：140714581008816,b 的原地址是：140714581008816
a 的新地址是：140714581008848,b 的新地址是：140714581008912
str1 的原地址是：2492572533424,str2 的原地址是：2492572533424
str1 的新地址是：2492640969648,str2 的新地址是：2492640968880
```

本例中，两个整数变量 a 和 b，当值相同时，地址是一样的，当分别被重新赋不同的值后，地址都发生了变化，不再相同；两个字符串变量 str1 和 str2，当最初赋值相同时，地址是相同的，当分别被重新赋不同的值后，地址发生了变化，不再相同。

可变类型的变量在第一次赋值声明时，也会在内存中开辟一块空间，用来存储这个变量被赋予的值。开发者能修改存储在内存中的值，当该变量的值发生了改变，它对应的内存地址不发生改变。可变数据类型变量中的值发生变化，地址不会变。若对变量进行重新赋值，则变量的地址也会改变，如例 3.2 所示。

【例 3.2】　定义两个列表，追加新元素，观察列表前后地址变化。

```
#例 3.2
lst1=["abc",1]
lst2=["abc",1]
print("lst1 的原地址是：{},lst2 的原地址是{}".format(id(lst1),id(lst2)))
lst1.append(5)          #在列表 lst1 中追加一个新元素 5
lst2.append(7)          #在列表 lst1 中追加一个新元素 7
print("lst1 追加元素后的地址是：{},lst2 追加元素后的地址是：
{}".format(id(lst1),id(lst2)))
lst1=["abc",1,"ddd"]
lst2=["abc",2,"fff"]
print("lst1 重新赋值后的地址是：{},lst2 重新赋值后的地址是：
{}".format(id(lst1),id(lst2)))
```

代码执行结果如下：

```
lst1 的原地址是：2492641197896,lst2 的原地址是：2492641197832
lst1 追加元素后的地址是：2492641197896,lst2 追加元素后的地址是：2492641197832
lst1 重新赋值后的地址是：2492640309064,lst2 重新赋值后的地址是：2492640307848
```

本例中，即使两个列表变量 lst1 和 lst2 赋的初值是相同的，地址却是不一样的，当分别追加了新元素，地址都没有发生变化；当分别被重新赋值后，相当于又重新开辟了内存单元（内存单元有新的内存单元地址），两个列表变量的新地址和原地址不再相同。

3.1.1 序列类型

序列类型按照先后顺序组织元素。序列中，每个元素都有自己的编号（索引号），可以通过索引号来获取元素。索引号有两类编号方式：从左向右按正向递增的顺序，起始序号为 0；从右向左按反向递减的顺序，起始序号为－1，如表 3.2 所示。

表 3.2 序列类型索引

序号	0	1	2	3	4	5	6	7	8	9	10	11	12
元素	中	华	人	民	共	和	国	，	首	都	北	京	。
序号	－13	－12	－11	－10	－9	－8	－7	－6	－5	－4	－3	－2	－1

序列类型有一些通用的操作，如索引、切片、运算、判断成员，也可以使用 Python 的内置函数和序列类型的方法来完成复杂的操作。

1. 序列的索引

序列类型按序号索引，获取字符串变量 zg＝“中华人民共和国，首都北京。”，从左向右的索引号从 0 开始，最大索引号是 12，从右向左的索引号从－1 开始，最小索引号是－13。对字符串 zg 内的单个字符索引，可以使用索引号来标识。例如字符“首”，可以使用 zg[8]或 zg[－5]索引。

2. 序列的切片

序列类型通过切片可以访问一定范围内的元素，可以是连续的元素，也可以是按指定步长的不连续的元素，并生成一个新的序列。序列切片方式，有以下几种情况，如表 3.3 所示。

表 3.3 序列类型切片方式

类 别	方 法	说 明	举 例	结 果
两边切片	[:n]	从左侧开始到第 n－1 个索引号	zg[:4]或者 zg[:－9]	“中华人民”
	[n:]	从左侧第 n 个索引号开始到序列最后	zg[10:]或者 zg[－3:]	“北京。”
中间切片	[n:m]	从左侧第 n 个索引号开始到第 m－1 个索引号	zg[4:7]或者 zg[－9:－6]	“共和国”
	[n:m:k]	从左侧第 n 个索引号开始到第 m－1 个索引号，以 k 为步长	zg[1:10:2]或者 zg[－12:－3:2]	“华民和，都”

3. 序列的加法运算

序列类型的加法运算，是序列和序列之间的连接，从而生成一个新的序列，用“＋”运算

符实现。如，"abc"＋"def"表达式运算的结果是"abcdef"。这种连接运算，只支持相同类型的序列之间进行。

4. 序列的乘法运算

序列类型的乘法运算，是序列的若干次重复，从而生成一个新的序列，用“＊”运算符实现。如，"abc"＊3 表达式运算的结果是"abcabcabc"。

5. 判断序列成员

判断一个元素是不是序列的成员，可以使用 in 和 not in 保留字来实现。如表 3.2 的序列中，判断“共和国”是不是序列的元素，可以用表达式“共和国” in “中华人民共和国，首都北京。”表示，表达式的结果为逻辑值真(True)；表达式“共和国” not in “中华人民共和国，首都北京。”的结果为逻辑值假(False)。

6. 内置函数

Python 提供了一些内置函数，如表 3.4 所示。

表 3.4　序列类型内置函数

类别	函　数	功　能	举　例	结　果
统计函数	len(x)	返回序列 x 的长度即元素个数	len("ab;c")	4
	min(x)	返回序列 x 中最小的元素	min("abc")	"a"
	max(x)	返回序列 x 中最大的元素	max("abc")	"c"
	sum(x)	对元素都是数值类型的序列 x 求和	sum([1,2,3])	6
修改函数	sorted(x[, reverse＝False])	对序列元素排序，返回一个有序列表	sorted(['b','c','a'])	['a', 'b', 'c']
	reversed(x)	返回一个以逆序访问的迭代器	reversed(['a','b','c'])	迭代器
生成函数	zip(x_1[, x_2, x_3, …, x_n])	返回一个 zip 对象，可生成一个迭代器，迭代器的第 n 个元素是由所有参数序列第 n 个元素组成的 n 元组	zip('abc',(1,2,3))	迭代器
	enumerate(x)	返回一个 enumerate 对象，可生成迭代器，该迭代器元素是由序列 x 元素的索引和值组成的元组	enumerate('abc')	迭代器

3.1.2　集合类型

数学中有集合的概念，而在 Python 中，集合类型用来保存不重复的元素，也就是说集合中的元素是唯一的。因此，集合元素有以下几个特性：①无序性，集合中的元素是没有前后顺序的；②多样性，集合中可以保存多种数据类型的元素；③唯一性，集合中的元素是唯一存在的，不会重复出现。

集合类型的操作包括数学集合中的并、交、差、补运算，如果集合 A 的元素包含集合 B 的元素，则集合 A 是集合 B 的父集，集合 B 是集合 A 的子集，相应地，集合的运算也包括判断集合之间的包含关系。

由于集合类型的特点，集合的应用通常用于数据的去重、元素共现、数据包含等场景中。

3.1.3 映射类型

映射是指两个事物之间的对应关系，两个事物间的映射关系在生活中大量存在。例如，学生的编号对应唯一的一个学生、医院内的患者编号对应唯一的一个患者。类似学生编号和学生、患者编号和患者的这种对应关系就是映射中的一对一关系。

除了一对一关系外，映射也有一对多和多对多关系。例如，一种疾病可能对应多种症状、同一种症状可能会对应多种疾病。当具有一对多或多对多关系的多个事物出现交集时，就要考虑去重机制。

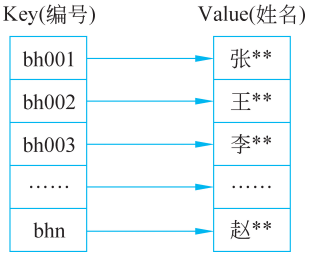

图 3.1 编号姓名映射关系图

在计算机中，映射(key-value)是一种关联式的容器类型，它存储了对象与对象之间的映射关系。Python 也提供了映射类型，字典是 Python 语言中唯一的映射类型。字典有两个属性，一个属性是 key(也称为键)，另一个属性是 value(也称为值)，key 和 value 统称为键值对，一个 key 可以对应一个值，也可以对应多个值。通过 key 可以获取到 value。例如，可以把患者编号和患者姓名以字典方式存储起来，患者编号存储到 key 中，患者姓名存储到 value 中。这样就可以通过患者编号很容易找到某位患者了，如图 3.1 所示。

Python 中，字典是典型的映射类型。

3.2 字符串

Python 中，字符串是最常使用的数据类型，由 0 个或多个字符组成。字符串属于序列类型。

3.2.1 字符串的创建

字符串的标识符是一对单引号''、一对双引号""或一对三引号"""，可以标识一个字符串常量。例如，'中国'，"北京"，'''健康中国'''。

3.2.2 转义字符

Python 中，程序里的字符串中的一些特殊字符，使用键盘无法直接输入，如 Tab 制表位、回车等，可以用斜线\加一个字符来代替，通常称为转义字符。常见的转义字符如表 3.5 所示，用法如例 3.3 所示。

表 3.5 常见转义字符

转义字符	说　明	转义字符	说　明
\n	换行，将当前位置移到下一行开头	\\	代表反斜线字符\'
\t	水平制表，跳到下一个 Tab 位置	\'	代表一个单引号字符
\b	退格，将当前位置移到前一列	\"	代表一个双引号字符
\r	回车，将当前位置移到本行开头		

【例 3.3】　创建一个患者就诊信息字符串，并将内容分行，按格式打印输出。

```
#例 3.3
st= "患者：\t 张**\n 性别：\t 男\n 年龄：\t40\n 出院描述：生命体征正常，伤口无疼痛，无发热乏力，食欲好，二便如常。"
print(st)
```

代码执行结果如下：

```
患　　者：　　张**
性　　别：　　男
年　　龄：　　40
出院描述：生命体征正常，伤口无疼痛，无发热乏力，食欲好，二便如常。
```

本例中，字符串 st 包含的字符有普通文本字符，也有特殊字符。其中，\t 代表 1 个制表位，\n 代表换行。打印输出时，实现格式化打印结果。

3.2.3　字符串内建函数

对字符串的操作，除了使用序列类型通用的操作符外，还有字符串的内建函数，也称为字符串的操作方法，可以对字符串数据进行处理。字符串的内建函数很多，常用的如表 3.6 所示，用法如例 3.4 所示。

表 3.6　常见字符串内建函数

类　型	方　法	描　述
字符串转换函数	string.capitalize()	将 string 的第一个字符转换为大写
	string.lower()	将 string 中所有大写字符转换为小写
	string.upper()	将 string 中所有小写字符转换为大写
	string.swapcase()	将 string 中所有字符大小写翻转
	string.center(width)	将 string 字符串扩充至 width 长度，原字符串居中，两边填充空格
	string.format(str)	将 str 格式化为指定的 string 模式
字符串处理函数	string.strip(str)	将 string 的左右两边指定字符去掉
	string.replace(str1,str2)	将 string 中的 str1 替换成 str2
	string.split(str="")	将 string 切片，str 为分隔符
	string.join(seq)	以 string 作为分隔符，将 seq 中所有的元素（字符串表示）合并为一个新的字符串
字符串判断函数	string.find(str)	检测 str 是否包含在 string 中，如果是返回开始的索引值，否则返回－1
	string.isalnum()	如果 string 至少有一个字符，并且所有字符都是字母或数字，则返回 True，否则返回 False
	string.isalpha()	如果 string 至少有一个字符，并且所有字符都是字母，则返回 True，否则返回 False
	string.isdecimal()	如果 string 只包含十进制数字，则返回 True，否则返回 False
	string.isnumeric()	如果 string 中只包含数字字符，则返回 True，否则返回 False
	string.startswith(obj)	检查字符串是否以 obj 开头，如果是，返回 True，否则返回 False
	string.endswith(obj)	检查字符串是否以 obj 结束，如果是，返回 True，否则返回 False
	string.count(str)	返回 str 在 string 中出现的次数

为了便于理解，可以把内建函数按功能分类。字符串转换函数是对原有字符串的元素进行修改；字符串处理函数功能是对字符串进行清洗，处理掉不需要的元素；字符串判断函数是检查字符串中的元素是否符合某个特征。

不论哪种字符串内建函数，都不能在原字符串上修改，而是生成一个新的字符串副本。例如，string.lower()函数，将原字符串中所有的大写字母转换为小写字母，并返回一个新的字符串。

【例 3.4】 中药方剂字符串，包含中药、剂量及炮制信息。请对字符串做预处理，提取中药和剂量信息，并将内容打印输出。

```
#例 3.4
st="茯苓 3 两,芍药 3 两,生姜 3 两(切),白术 2 两,附子 1 枚(炮、去皮、破 8 片)"
lst=st.split(",")
print(lst)
for item in lst:
    if "(" in item:
        n=item.find("(")
        item=item[:n]
    for c in item:
        if c.isdecimal():
            m=item.find(c)
            break
    yao=item[:m]
    jl=item[m:]
    print(yao,jl)
```

代码执行结果如下：

```
['茯苓 3 两', '芍药 3 两', '生姜 3 两(切)', '白术 2 两', '附子 1 枚(炮、去皮、破 8 片)']
茯苓 3 两
芍药 3 两
生姜 3 两
白术 2 两
附子 1 枚
```

本例中，字符串 st 以英文逗号分隔，由中药、剂量和炮制方法组成，炮制方法在括号中，目标是提取中药和剂量信息。首先，利用 split()函数对字符串进行切分，返回单个中药信息组成的列表；接着，对列表进行循环遍历，如果字符串里包含字符左括号“(”，则利用 find()函数找到“(”字符所在索引位置，并对字符串进行切片，去掉字符“(”以后的所有的信息，即去掉炮制信息；预处理后，再遍历该字符串中的每个字符，如果字符是数字，则记住该数字所在位置，跳出循环；最后，切片获取数字左边的子字符串保存到 yao 变量中，切片获取数字右边的子字符串保存到 jl 变量中，并打印输出。

3.2.4 字符串格式化

字符串格式化用于解决字符串和变量同时输出时的格式安排问题。字符串是程序向控制台、网络、文件等介质输出运算结果的主要形式之一，为了能提供更好的可读性和灵活性，字符串类型的格式化是运用字符串类型的重要内容之一。Python 语言同时支持两种字符

串格式化方法，format 方式是在 Python 3 引入的一个新的字符串格式化的方法，通过访问字符串的内建函数 format()实现。

字符串 format()方法的基本使用格式是：＜模版字符串＞.format(＜逗号分隔的参数＞)，其中，＜模版字符串＞由一系列的槽和其他固定字符组成，槽用来控制修改字符串中嵌入值出现的位置，用{}表示，其基本思想是将 format()方法的＜逗号分隔的参数＞中的参数按照序号关系替换到＜模板字符串＞的槽中。

＜模版字符串＞的槽内可以设置参数序号(如果缺省序号时，默认从 0 开始排序)，还可以设置格式控制信息。槽的内部样式格式是{＜参数序号＞:＜格式控制标记＞}，＜格式控制标记＞包括：＜填充＞＜对齐＞＜宽度＞＜,＞＜精度＞＜类型＞6 个字段，可以组合使用，具体说明如表 3.7 所示，用法如例 3.5 所示。

表 3.7　格式控制标记说明

格式控制标记	说　明
填充	＜宽度＞内除了参数外的字符采用什么方式表示，默认采用空格，可以通过＜填充＞更换
对齐	在＜宽度＞内输出时的对齐方式，分别使用＜、＞和^ 3 个符号表示左对齐、右对齐和居中对齐
宽度	当前槽的设定输出字符宽度。如果该槽对应的 format()函数参数长度比＜宽度＞设定值大，则使用参数实际长度；如果该值的实际位数小于指定宽度，则位数将被默认以空格字符补充
精度	表示两个含义，由小数点(.)开头。对于浮点数，精度表示小数部分输出的有效位数；对于字符串，精度表示输出的最大长度
,	用于显示数字的千位分隔符
类型	表示输出整数和浮点数类型的格式规则。对于整数类型，输出格式包括 6 种：b(二进制)、c(unicode 字符)、d(十进制方式)、o(八进制方式)、x(小写十六进制方式)、X(大写十六进制方式)。对于浮点数类型，输出格式包括 4 种：e(小写字母 e 的指数形式)、E(大写字母 E 的指数形式)、f(标准浮点形式)、%(百分形式)

【例 3.5】　按格式打印九九乘法表。

```
#例 3.5
for i in range(1,10):
    a = 1
    while a <= i:
        print("{0}*{1}={2:>2}".format(a,i,a*i),end="\t")
        a +=1
    print()
```

代码执行结果如下：

```
1*1= 1
1*2= 2  2*2= 4
1*3= 3  2*3= 6  3*3= 9
1*4= 4  2*4= 8  3*4=12  4*4=16
1*5= 5  2*5=10  3*5=15  4*5=20  5*5=25
1*6= 6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
```

```
1 * 7= 7   2 * 7=14   3 * 7=21   4 * 7=28   5 * 7=35   6 * 7=42   7 * 7=49
1 * 8= 8   2 * 8=16   3 * 8=24   4 * 8=32   5 * 8=40   6 * 8=48   7 * 8=56   8 * 8=64
1 * 9= 9   2 * 9=18   3 * 9=27   4 * 9=36   5 * 9=45   6 * 9=54   7 * 9=63   8 * 9=72   9 * 9=81
```

本例中，外循环 for 循环执行 9 次，循环变量 i 代表行号；内循环 while 循环执行 i 次，循环变量 a 代表列号；内循环每循环一次，按格式打印当前行当前列的一个乘法等式（列号 * 行号＝a * i），行内以\t 制表位分隔；每执行完一个 for 循环，调用 print()函数实现换行。

3.3 列表和元组

Python 的序列类型中，列表是最常用的 Python 数据类型，它可以同时保存多个不同类似的数据。创建一个列表，只要把用逗号分隔的不同的元素使用方括号[ ]括起来即可。列表的元素不需要具有相同的类型，可以是任意类型，每个元素都有位置对应的索引值（也可称为下标）。

列表的元素可以增加、删除、修改，也可以改变元素的顺序。对列表的操作可以通过列表的操作符和内建函数实现。

元组和列表较相似，都是序列类型，符合序列类型的特征，很多操作都是相似的；但二者又有区别，最大的区别是，列表的元素是可变的，而元组的元素是不可变的。

3.3.1 列表的创建

1. 直接创建

创建一个列表，只要把用逗号分隔的不同的元素使用方括号[]括起来即可，如赋值语句 list1＝ ["关冲", "劳宫", "前谷", 123]；赋值语句 list2＝[]，则创建一个空列表。

2. 使用 list()函数创建

使用 list()函数可以将字符串、元组、字典等其他组合数据类型转换为列表，如例 3.6 所示。

【例 3.6】 创建列表。

```
#例 3.6
list1=[]
list2=["关冲", "劳宫", "前谷", 123]
print(list2)
print("字符串'This is Python'转换为列表: ",list("This is Python"))
```

代码执行结果是：

```
['关冲', '劳宫', '前谷', 123]
字符串'This is Python'转换为列表: ['T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'P', 'y',
't', 'h', 'o', 'n']
```

本例中，list1 为空列表，list2 是直接赋值的非空列表，利用 list()函数将字符串转换为列表。值得注意的是，在原字符串“This is Python”中，字符串中的每个元素转换为列表中的每个元素，顺序不变。

3.3.2 列表操作符

列表的常用操作包括索引、切片、加、乘、检查成员。索引操作是通过索引号的方式获取

列表中的单个数据项；切片操作是通过起止索引号获取连续或按步长增长的不连续的数据项；列表加运算是将两个列表的数据项首尾相连生成一个新的列表；列表乘运算是将列表复制 n 次并连接生成新的列表；检查成员操作是判断某个元素是不是列表的数据项，如例 3.7 所示。

【例 3.7】　创建一个由感冒的症状组成的列表，并执行索引、切片、加、乘、检查成员等操作。

```
#例 3.7
感冒=["打喷嚏","流鼻涕","发烧 37.5 以上","嗓子疼","怕冷","咳嗽"]
新冠肺炎=["发热","干咳","乏力","肌肉痛","关节痛"]
print(感冒[0])
print(感冒[1:3])
print(感冒[0:5:2])
print(感冒+新冠肺炎)
print(感冒 * 2)
print("发烧" in 新冠肺炎)
```

代码执行结果如下：

```
打喷嚏
['流鼻涕', '发烧 37.5 以上']
['打喷嚏', '发烧 37.5 以上', '怕冷']
['打喷嚏', '流鼻涕', '发烧 37.5 以上', '嗓子疼', '怕冷', '咳嗽', '发热', '干咳', '乏力', '肌肉痛', '关节痛']
['打喷嚏', '流鼻涕', '发烧 37.5 以上', '嗓子疼', '怕冷', '咳嗽', '打喷嚏', '流鼻涕', '发烧 37.5以上', '嗓子疼', '怕冷', '咳嗽']
False
```

本例中，新建两个列表变量：感冒和新冠肺炎。通过索引号“感冒[0]”获取第 0 个数据项；通过“感冒[1:3]”获取第 1 个和第 2 个数据项(不包括 3)；通过“感冒[0:5:2]”获取从 0 开始到第 4 个数据项(不包括 5)，以 2 为步长，即第 0 个、第 2 个、第 4 个数据项；通过“感冒＋新冠肺炎”将两个列表首尾相连生成新的列表；通过“感冒 * 2”使列表复制 1 次元素生成新的列表；通过“"发烧" in 新冠肺炎”判断发烧是不是新冠肺炎的数据项。

3.3.3　列表内建函数

列表对自身元素的操作，还可以通过内建函数来完成。列表是可变的，可以对元素增加、删除、修改，可以统计和索引元素。常用内建函数如表 3.8 所示。

表 3.8　列表类型内建函数

类别	函　数	功　能
增加函数	list.append(x)	将元素 x 追加到列表末尾
	list.insert(n,x)	在指定位置 n 添加元素 x
	list.extend(x)	将列表 x 追加到原列表后面，返回合并后的列表
删除函数	list.pop([n])	删除列表位置 n 的元素，n 缺省时，删除列表末尾元素，返回删除的元素
	list.remove(x)	删除元素 x，如果有多个元素 x，删除的是第一次出现的元素，如果元素不存在会报错
	list.clear()	清除所有元素，返回空列表

续表

类别	函　数	功　能
排序函数	list.sort([key=lambda x: x[m], reverse = False)	对列表中的元素排序，默认升序排序，如果元素是元组，可以以元组中的第 m 个元素作为排序关键字进行排序，reverse=False，为升序排序，reverse=True，为降序排序，返回排序后列表
	list.reverse()	将列表进行翻转，返回翻转后的列表
统计函数	list.count(x)	返回的是元素 x 在列表里面的个数
	list.index(x)	返回的是元素 x 在列表中的第一个位置

为了便于记忆，将内建函数分为 4 类，分别是增加函数、删除函数、排序函数、统计函数。增加函数，既可以在列表末尾，也可以在指定位置增加元素，还可以将一个列表整体追加到原列表中。append()函数可以将新的元素追加到列表的末尾，返回一个新的列表；insert()函数可以在指定位置添加元素，原位置元素按顺序向后移动；extend()函数可以将另一个列表与原列表合并起来组成新列表。删除函数，既可以逐个删除元素，也可以一次性删除全部元素。pop()函数可以删除指定位置的元素，如果需要保存被删除的元素，可以存储到一个变量中；而 remove()函数则是删除指定元素，如果元素不存在，则会报错；clear()函数可以将元素一次性全部删除，在做数据处理时，应谨慎使用。排序函数可以使用 reverse()函数直接对列表翻转，也可以使用 sort()函数进行升序或降序排序，在使用 sort()函数排序时，还可以指定某个元素作为排序关键字。统计函数中，利用 count()函数可以统计某个元素在列表中出现的次数，利用 index()函数获取某个元素在列表中第一次出现的位置。

3.3.4　元组的创建和使用

Python 的序列类型中，元组是有序且不可更改的序列。创建一个元组，只要把用逗号分隔的不同的元素使用圆括号()括起来即可。例如赋值语句 tuple1=()，可以创建一个空元组；赋值语句 tuple2=("关冲", "劳宫", "前谷", 123)可以创建一个非空元组，每个元素都有位置对应的索引值(也可称为下标)。

元组的元素不能增加、删除、修改，也不能改变元素的顺序。对元组的操作除了可以索引、切片、加、乘、判断成员外，元组的操作函数不包括增加、删除、修改类别，如果需要对元组进行修改，就要将元组先转换为列表，再做修改。因此，元组的内建函数，只有统计类型的函数与列表相似，利用 count()函数可以统计某个元素在元组中出现的次数，利用 index()函数可以获取某个元素在元组中第一次出现的位置，说明如表 3.8 所示，用法如例 3.8 所示。

【例 3.8】 创建一个由风寒感冒、风热感冒的症状组成的列表，创建一个由姓名和体温为二元组的元素组成的体温列表，执行追加、插入、排序、弹出等操作，并使用 for 循环按格式打印输出体温元素和类型。

```
#例 3.8
风寒感冒=["打喷嚏","流清鼻涕","发烧 37.5 以上","嗓子疼","怕冷","咳嗽"]
风热感冒=["发热","咽痛","乏力","肌肉痛","关节痛","黄痰","浓黄鼻涕","咳嗽"]
体温=[("小红",40),("小名",38),("小亮",36),("小颖",38.5),("小花",39),("小良",40)]
风寒感冒.append("恶风")
print(风寒感冒)
```

```
风热感冒.insert(1,"头疼")
print(风热感冒)
体温.sort(key=lambda x:x[1],reverse=True)
print(体温)
print(风寒感冒.pop(5))
for name in 体温:
    print("{}:{}".format(name[0],name[1]))
    print(name,type(name))
```

代码执行结果如下：

```
['打喷嚏', '流清鼻涕', '发烧 37.5 以上', '嗓子疼', '怕冷', '咳嗽', '恶风']
['发热', '头疼', '咽痛', '乏力', '肌肉痛', '关节痛', '黄痰', '浓黄鼻涕', '咳嗽']
[('小红', 40), ('小良', 40), ('小花', 39), ('小颖', 38.5), ('小名', 38), ('小亮', 36)]
咳嗽
小红:40
('小红', 40) <class 'tuple'>
小良:40
('小良', 40) <class 'tuple'>
小花:39
('小花', 39) <class 'tuple'>
小颖:38.5
('小颖', 38.5) <class 'tuple'>
小名:38
('小名', 38) <class 'tuple'>
小亮:36
('小亮', 36) <class 'tuple'>
```

本例中，风寒感冒和风热感冒的列表，是由症状字符串组成的；体温列表是由姓名和体温的元组为元素组成的。利用 append()函数，向风寒感冒列表的最后追加症状“恶风”；利用 insert()函数，向风热感冒列表的索引位置 1 处插入症状“头疼”；利用 sort()函数对体温列表排序，排序关键字是列表元素的元组第 2 个数据项（即体温），reverse＝True 为降序排序。

3.4　集合

Python 中，集合通常用来存放唯一值的一系列数据，即包含 0 个或多个数据项的无序组合。

需要注意以下几点。

（1）集合的元素只能是固定数据类型，如整数、浮点数、字符串、元组；不能是可变数据类型，如列表、字典。

（2）集合的元素是无序的，因此，不能对集合做索引、切片、排序、修改、插入等操作，集合的输出顺序和定义顺序可以不一致。

（3）集合的元素是唯一的，因此，任何包含重复数据的添加都是无效的，使用集合类型能够过滤掉重复元素。

3.4.1 集合的创建

1. 直接创建

创建一个非空集合，只要把用逗号分隔的不同的元素使用大括号{}括起来即可，如赋值语句 set1={"关冲", "劳宫", "前谷", 123}。

2. 使用 set()函数创建

使用 set()函数可以将字符串、列表、元组等其他类型转换为集合，如果 set()函数参数为空，则创建一个空集合，如例 3.9 所示。

【例 3.9】 分别创建空集合，非空集合。

```
#例 3.9
set1=set()
print("空集合: ",set1)
set2={"关冲", "劳宫", "前谷", 123}
print("非空集合: ",set2)
set3=set("This is Python")
print("字符串'This is Python'转换为集合: ",set3)
```

代码执行结果是：

```
空集合: set()
非空集合: {'前谷', '关冲', 123, '劳宫'}
字符串'This is Python'转换为集合: {'t', 's', 'y', 'P', ' ', 'n', 'T', 'o', 'h', 'i'}
```

本例中，利用 set()函数创建空集合 set1，通过直接赋值的方法创建非空集合 set2，利用 set()函数将字符串转换为集合 set3。值得注意的是，在原字符串"This is Python"中，字符 h、i、s 存在重复，生成集合后，只保留 1 个字母。

3.4.2 集合运算

集合有 4 种基本的操作符：交集(&)，并集(|)，差集(－)，补集(^)。利用这些操作符可以进行集合基本运算。另外，还可以判断集合的包含关系，如表 3.9 和例 3.10 所示。

表 3.9 集合类型操作符

操作符	示 例	描 述
&	S&T	交集，返回一个新集合，包括同时在 S 和 T 中的元素
|	S|T	并集，返回一个新集合，包括集合 S 和 T 中的所有元素
－	S－T	差集，返回一个新集合，包括在集合 S 中但不在集合 T 中的元素
^	S^T	补集，返回一个新集合，包括集合 S 和 T 中的元素，但不包括同时在集合 S 和 T 中的元素
<=	S<=T	如果 S 与 T 相同或者 S 是 T 的子集，则返回 True；否则返回 False。可以用 S<T 判断 S 是不是 T 的真子集
>=	S>=T	如果 S 与 T 相同或者 S 是 T 的父集，则返回 True；否则返回 False。可以用 S>T 判断 S 是不是 T 的真父集

【例 3.10】　集合的运算。

```
#例 3.10
风寒感冒=["打喷嚏","流清鼻涕","发热","怕冷","咳嗽","恶风","咳嗽"]
风热感冒=["发热","咽痛","乏力","肌肉痛","关节痛","黄痰","流浓黄鼻涕","咳嗽"]
set4=set(风寒感冒)
set5=set(风热感冒)
print("风寒感冒和风热感冒的共同症状是：",set4&set5)
print("风寒感冒和风热感冒的症状一共有：",set4|set5)
print("风寒感冒里包含的症状,在风热感冒没有的是：",set4-set5)
print("去掉风寒感冒和风热感冒共有症状后,其他症状是：",set4^set5)
```

代码执行结果是：

```
风寒感冒和风热感冒的共同症状是：{'发热', '咳嗽'}
风寒感冒和风热感冒的症状一共有：{'发热', '肌肉痛', '怕冷', '流清鼻涕', '关节痛', '打喷嚏', '恶风', '黄痰', '咽痛', '乏力', '流浓黄鼻涕', '咳嗽'}
风寒感冒里包含的症状,在风热感冒没有的是：{'恶风', '怕冷', '流清鼻涕', '打喷嚏'}
去掉风寒感冒和风热感冒共有症状后,其他症状是：{'肌肉痛', '怕冷', '流清鼻涕', '关节痛', '打喷嚏', '恶风', '黄痰', '咽痛', '乏力', '流浓黄鼻涕'}
```

集合的交、并、差、补运算应用在实际问题中，可以解决共现问题、统计去重后数据项个数、排除混杂数据等。在本例中，判断某人的感冒是风寒感冒还是风热感冒，可以通过症状作为参考，使用交运算找出共同症状；使用并运算找出所有症状；使用差运算排除风热感冒因素；使用补运算找出排除共同症状外的其他症状。

3.4.3　集合内建函数

与序列类型类似，Python 也提供了多种集合的内建函数用于元素的添加和删除，如表 3.10 和例 3.11 所示。

表 3.10　集合类型内建函数

类别	函　数	描　述
添加	S.add(x)	如果数据项 x 不在集合 S 中，将 x 添加到 S 中
	S.update(T)	合并集合 T 中的元素到当前集合 S 中，并自动去除重复元素
删除	S.pop()	随机删除并返回集合中的一个元素，如果集合为空则抛出异常
	S.remove(x)	如果 x 在集合 S 中，移除该元素；如果 x 不存在则抛出异常
	S.discard(x)	如果 x 在集合 S 中，移除该元素；如果 x 不存在不报错
	S.clear()	清空集合

【例 3.11】　集合的操作。

```
#例 3.11
风寒感冒={"打喷嚏","流清鼻涕","发热","怕冷","咳嗽","恶风"}
风热感冒={"发热","咽痛","乏力","肌肉痛","关节痛","黄痰","流浓黄鼻涕","咳嗽"}
风寒感冒.add("头疼")
print("风寒感冒添加头疼：",风寒感冒)
zhengz1=风寒感冒.pop()
```

```
print("随机删除风寒感冒 1 个症状并返回：",zhengz1,风寒感冒)
zhengz2=风寒感冒.discard("咳嗽")
print("随机删除风寒感冒 1 个症状不返回结果：",zhengz2,风寒感冒)
风寒感冒.update(风热感冒)
print("合并症状到风寒感冒并去重：",风寒感冒)
风寒感冒.clear()
print("清空集合",风寒感冒)
```

代码执行结果是：

```
风寒感冒添加头疼：{'打喷嚏', '咳嗽', '头疼', '怕冷', '流清鼻涕', '恶风', '发热'}
随机删除风寒感冒 1 个症状并返回：打喷嚏 {'咳嗽', '头疼', '怕冷', '流清鼻涕', '恶风',
'发热'}
随机删除风寒感冒 1 个症状不返回结果：None {'头疼', '怕冷', '流清鼻涕', '恶风', '发热'}
合并症状到风寒感冒并去重：{'关节痛', '流浓黄鼻涕', '肌肉痛', '咳嗽', '头疼', '怕冷',
'咽痛', '流清鼻涕', '黄痰', '恶风', '发热', '乏力'}
清空集合 set()
```

本例中，需要注意的是，因集合本身是无序的，所以，无论添加还是删除元素，都不能以序号作为索引。在删除元素时，需要关注几种内建函数的用法和注意事项。

3.5 字典

Python 的组合数据类型中，字典是典型的映射类型。字典不同于其他组合数据类型，它的数据项是由“键值对”的映射组成，键和值分别是两个集合，因此字典的元素也是无序的，如图 3.2 所示。

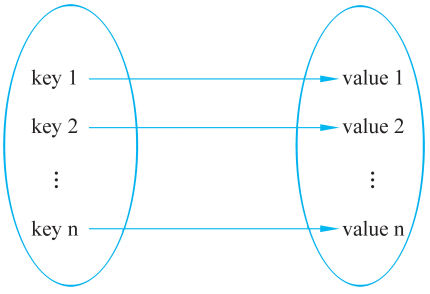

图 3.2 字典的键值对映射

在列表中，可以通过“下标数字”找到对应的对象。而在字典中，可以通过“键对象”找到对应的“值对象”。“键”是任意的不可变数据，例如整数、浮点数、字符串、元组。但是，列表、字典、集合这些可变对象不能作为“键”，并且“键”不可重复。“值”可以是任意的数据，并且可重复。

字典的元素可以增加、删除、修改。对字典的操作可以通过字典的操作符和内建函数实现。

3.5.1 字典的创建

创建一个字典，只要把用逗号分隔的不同的键值对使用花括号({})括起来即可。例如 dict1={"中国":"北京","俄罗斯":"莫斯科"}，创建一个非空字典；dict2={}创建一个空字典。也可以使用 dict()函数创建一个字典，如 dict4=dict()，具体用法如例 3.12 所示。

【例 3.12】 字典的创建的方法。

```
#例 3.12
dict1={'name':'张明明','age':30,'job':'数据分析师'}
print("dict1: ",dict1)
```

```
dict2=dict(name='张明明',age=30,job='数据分析师')
print("dict2: ",dict2)
dict3=dict([("name","张明明"),("age",30),("job",'数据分析师')])
print("dict3: ",dict3)
k=("name","age","job")
v=("张明明",30,'数据分析师')
dict4=dict(zip(k,v))
print("dict4: ",dict4)
dict5={ }                #空的字典对象
dict6=dict()             #空的字典对象
print(dict5,dict6)
```

代码执行结果是：

```
dict1: {'name': '张明明', 'age': 30, 'job': '数据分析师'}
dict2: {'name': '张明明', 'age': 30, 'job': '数据分析师'}
dict3: {'name': '张明明', 'age': 30, 'job': '数据分析师'}
dict4: {'name': '张明明', 'age': 30, 'job': '数据分析师'}
{} {}
```

本例中，第 1 行通过直接赋值的方法创建一个非空字典；第 3 行通过调用 dict()函数，使用按名称参数传递的方法创建一个非空字典；第 5 行，通过调用 dict()函数将由二元组元组组成的列表强制类型转换为字典；第 9 行，通过调用 zip()函数生成迭代器，再调用 dict()函数强制类型转换的方法创建字典；第 11、12 行使用两种方法创建空字典。

3.5.2　字典操作符

字典类型的操作符比较简单，常用的有赋值操作符等号(=)，判断成员的操作运算符 in，如 dict[name]="张明"，可以通过 name 键进行索引修改其对应的值为"张明"；语句 name in dict1，是用来判断 name 是不是字典 dict1 的键。

3.5.3　字典内建函数

字典的操作通常使用内建函数来完成，字典是可变的，可以对元素增加、删除、修改，但字典是无序的，如果要排序，需要先转换成列表才能完成。常用内建函数如表 3.11 所示，具体用法如例 3.13 所示。

表 3.11　字典类型内建函数

类别	函　数	描　述
获取	D.keys()	返回字典 D 键的可迭代对象
	D.values()	返回字典 D 值的可迭代对象
	D.items()	返回字典 D 键值对的可迭代对象
	D.get(key,v=None)	如果键存在，则返回其对应值；如果键不在字典中，则返回默认值 v
	D.copy()	返回字典 D 的副本
添加	D.update(D2)	将字典 D2 的键值对添加到字典 D 中

续表

类别	函　数	描　述
删除	D.pop(key[,d])	移除并且返回对应给定键或给定的默认值 D 的值
	D.popitem()	从 D 中移除任意一项，并将其作为(键,值)对返回
	D.clear()	清空字典

【例 3.13】 对字典数据进行遍历，获取数据项、键、值，并查找相关键值数据。

```
#例 3.13
dict1= {'name':'张明明','age':30,'job':'数据分析师'}
for item in dict1.items():
    print("键值对元组: ",item)
for key in dict1.keys():
    print("键: ",key)
for value in dict1.values():
    print("值: ",value)
print(dict1.get("name","无"))
print(dict1.get('bmi',0))
```

代码执行结果如下：

```
键值对元组: ('name', '张明明')
键值对元组: ('age', 30)
键值对元组: ('job', '数据分析师')
键: name
键: age
键: job
值: 张明明
值: 30
值: 数据分析师
张明明
0
```

本例中，dict1 是一个个人信息字典，第一个 for 循环遍历字典中所有的键值对，执行结果是每个键值对以二元组的形式显示；第二个 for 循环遍历字典中所有的键，执行结果是获取每个键；第三个 for 循环遍历字典中所有的值，执行结果是获取每个值；最后通过调用 get() 函数查找 name 和 age 键是否在字典中，如果键存在，则返回对应的值，因此 name 键存在返回值张明明，bmi 键不存在返回默认值 0。

3.6 正则表达式库

在医学中，有大量文本类数据，如医院电子病历系统中的医生诊断信息、患者主诉信息，医生开立的处方，医学期刊文献、典籍、科普文章，互联网医疗平台中的医患沟通数据等，要想对它们进行深入的挖掘和分析，经常需要从文本或字符串中抽取出想要的信息，用于进一步的语义理解或其他处理。在这些文本中，有时往往存在大量的信息符合某种特定的规律，

如何快速有效地提取这些相似规律的文本，是文本信息抽取的关键点。正则表达式就是一种从文本中抽取信息的有效手段，它一般通过搜索特定模式的语句实现信息的抽取。正则表达式应用范围非常广泛，如网页爬取、网页信息解析、文本预处理和信息抽取等，可以将非结构化、半结构化的文本转换成结构化的文本。本节通过介绍 Python 中正则表达式 re 库中的函数和元字符，演示正则表达式在文本信息处理中的应用。

3.6.1　正则表达式的概念

正则表达式(regular expression)是一种可以用于模式匹配和替换的工具，它是一种专用的编程语言。使用正则表达式，可以对指定文本实现匹配测试、内容查找、内容替换、字符串切分等功能。

非结构化文本通常有两类，一种是文本格式的文档；另一种是网页格式的文档。对于非结构化文本，正则表达式可以进行信息抽取，转换成结构化信息，方便进一步分析和挖掘；对于网页文档，往往含有大量的 html 标签，需要使用正则表达式将这些无用的标签去掉，抽取关键的文本信息，转换为自然语言文本。

3.6.2　正则表达式的字符

正则表达式描述了一种字符串匹配的模式(pattern)，可以用来检查一个字符串是否含有某种子串、将匹配的子串替换或者从某个字符串中取出符合某个条件的子串等。正则表达式的模式串是由普通字符(如字符 a 到 z)以及特殊字符(称为“元字符”)组成的模式字符串，模式描述在搜索文本时要匹配的 1 个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

构造正则表达式的方法和创建数学表达式的方法一样，也就是用多种元字符与运算符可以将小的表达式结合在一起来创建更大的表达式。正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。

例如，模式字符串“pytho＋n”，可以匹配 python、pythoon、pythooon 等，＋号代表前面的字符必须至少出现 1 次；pytho * n，可以匹配 pythn、python、pythoon、pythooon 等，* 号代表前面的字符出现 0 到多次；pytho?n，可以匹配 pythn、python 等，? 号代表前面的字符出现 0 到 1 次。这里的＋、* 、? 都是元字符。

1. 普通字符

普通字符包括没有显示指定为元字符的所有字符，包括字母、数字、标点符号等，如表 3.12 所示。非打印字符也是正则表达式的一部分，如回车符、换行符等。另外，字符串中可以包含任何字符，如果待匹配的字符串中出现 $ 、[]等特殊字符，就会与正则表达式的特殊字符发生冲突。在 Python 中，解决这个问题的方式是使用转义字符\对字符串中的特殊字符进行转义，如\ $ 代表普通字符 $ ，如表 3.13 所示，具体用法如例 3.14 所示。

表 3.12　正则表达式中的普通字符

字　符	描　述	示　例
[ABC]	表示包含括号内的任意字母、汉字、数字标点符号组成的字符串	pat1＝"a[bd]c"，该模式可以匹配"abc"和"adc"字符串

续表

字　符	描　述	示　例
[^ABC]	表示不包含括号内的任意字母、汉字、数字标点符号组成的字符串	pat1＝"a[^bd]c"，该模式可以匹配除"abc"和"adc"等以 a 开头、以 c 结尾的长度为 3 的字符串
[A-Z]	匹配所有大写字母；[a-z]表示匹配所有小写字母	pat1＝"p[a-z]"，该模式可以匹配所有以 p 开头、第二个字符是任意小写字母的长度为 2 的字符串
[0-9]	匹配所有数字 0，1，2，…，9	pat1＝"p[0-9]"，该模式可以匹配所有以 p 开头、第二个字符是数字的字符串，如"p0"，"p1"，…，"p9"
[\u4E00-\u9FA5]	匹配所有中文汉字	pat1＝"[0-9]＋[\u4E00-\u9FA5]＋"，该模式可以匹配所有以任意数字开头后面是中文汉字的字符串

表 3.13　正则表达式中的非打印字符

字符	描　述	字符	描　述
\n	换行符	\d	匹配数字
\r	回车符	\D	匹配非数字
\t	制表符	\w	字、字母、数字
\v	垂直制表符	\W	与\w 相反，非(字、字母、数字)
\s	任何空白字符	\f	换页符
\S	匹配任何非空白字符，等价于[^\f\n\r\t\v]		

2. 元字符

元字符是特殊字符，就是一些有特殊含义的字符，若要匹配这些特殊字符，必须首先使字符“转义”，即将反斜线字符\放在它们前面。因此，元字符在正则表达式串中扮演着特殊的作用，如表 3.14 所示。

表 3.14　正则表达式中的元字符

字符	描　述	示　例
$	匹配输入字符串的结尾位置，要匹配 $ 字符本身，使用\ $	pat1＝" language. $ "，该模式可以匹配以 language.结尾的字符串
?	匹配前面的子表达式 0 次或 1 次	pat1＝pytho?n，该模式可以匹配 pythn、python 包含 0 个或 1 个 o 的字符串
*	匹配前面的子表达式 0 次或多次	pat1＝ pytho * n，该模式可以匹配 pythn、python、pythoon、pythooon 等包含 0 个或多个 o 的字符串
＋	匹配前面的子表达式 1 次或多次	pat1＝ python＋n，该模式可以匹配 python、pythoon、pythooon 等包含 1 个或多个 o 的字符串

续表

字符	描　　述	示　　例
{n}	匹配前面的子表达式 n 次	pat1＝pytho{3}n，该模式可以匹配 pythooon
{n,}	匹配前面的子表达式至少 n 次	pat1＝pytho{3,}n，该模式可以匹配 pythooon、pythoooon、……
{n,m}	匹配前面的子表达式至少 n 次，至多 m 次	pat1＝pytho{3,4}n，该模式可以匹配 pythooon、pythoooon
.	匹配除换行符 \n 之外的任何单字符	pat1＝"p."，该模式可以匹配以 p 开头，第二个字符除换行之外的任意长度为 2 的字符串
^	匹配输入字符串的开始位置，当该符号在方括号表达式中使用时，表示不接受该方括号表达式中的字符集合	pat1＝"^p"，该模式可以匹配以 p 开头的字符
|	指明两项之间任选其一	pat1＝"pa|b"等价于 pat1＝p[ab]，该模式可以匹配以 p 开头，第二个字符是 a 或 b 的长度为 2 的字符串
\	转义字符，用来对特殊字符进行转义	如\n 代表换行
[ ]	标记一个子表达式中任意单个字符的开始和结束位置	pat1＝"[is]"，该模式可以匹配原字符串中的所有 i 和 s 字符
( )	标记一个子字符串的开始和结束位置	pat1＝"(is)"，该模式可以匹配原字符串中的所有 is 子字符串

【例 3.14】　正则表达式的模式和字符。

```
#例 3.14
import re
str1="自然语言不是自动语言"
pat1="自[然动]语"
print(re.findall(pat1,str1))

str2="python is a language."
pat2="p[a-z]"
print(re.findall(pat2,str2))
pat3="language.$ "
print(re.findall(pat3,str2))

str3="python is a language. javascript is a language too."
pat4="(is)"
print(re.findall(pat4,str3))

str4="python is a language. paggage"
pat5="p."
print(re.findall(pat5,str4))
pat6="pa|b"
print(re.findall(pat6,str4))
pat7="^p"
print(re.findall(pat7,str4))
```

代码执行结果如下：