第 5 章

变 量 类 型

5.1 整数

Python 中的整数跟平常所认知的整数一样，包括正整数、负整数和零，即没有小数点的数字。在 Python 中也可以用二进制、八进制或十六进制表示十进制整数，其中，二进制以 0b 或 0B 为前缀，八进制以 0o 或 0O 为前缀，十六进制以 0x 或 0X 为前缀。另外，需要注意的是，Python 可以处理任意大小的整数，示例代码如下：

```
#资源包\Code\chapter5\5.1\0501.py
num1 = 10
#输出结果为 10
print(num1)
#type()函数用于返回数据类型，整数的输出结果为<class 'int'>
print(type(num1))
num2 = 0b110
#输出结果为十进制的 6
print(num2)
#Python 可以处理任意大小的整数
num3 = 12345678901234567890
#输出结果为 12345678901234567890
print(num3)
```

5.2 浮点数

浮点数就是小数，即带有小数点的数，示例代码如下：

```
#资源包\Code\chapter5\5.2\0502.py
num = 0.55
#输出结果为 0.55
print(num)
#type()函数用于返回数据类型，浮点数的输出结果为<class 'float'>
print(type(num))
```

这里有一点非常重要，读者必须注意，即由于计算机的内存、CPU 寄存器等硬件单元都是有限的，只能表示有限位数的二进制，因此存储的二进制小数就会和实际转换而成的二进

制数有一定的误差，所以整数运算永远是精确的，而浮点数运算则可能是不精确的，示例代码如下：

```
#资源包\Code\chapter5\5.2\0503.py
num1 = 0.55
num2 = 0.4
#输出结果为 0.9500000000000001
print(num1 + num2)
```

为了解决浮点数运算不精确的问题，可以通过 decimal 模块中的 Decimal 类进行精确运算，示例代码如下：

```
#资源包\Code\chapter5\5.2\0504.py
from decimal import Decimal
num1 = Decimal('0.55')
num2 = Decimal('0.4')
#输出结果为 0.95
print(num1 + num2)
```

5.3 复数

在 Python 中，把形如 $z=a+b\mathrm{j}$(a，b 均为实数)的数称为复数，其中，a 称为实部，b 称为虚部，j 称为虚数单位。当 z 的虚部等于零时，称 z 为实数；当 z 的虚部不等于零，但实部等于零时，称 z 为纯虚数，示例代码如下：

```
#资源包\Code\chapter5\5.3\0505.py
z1 = 2 + 3j
z2 = 1 + 2j
#输出结果为(3+5j)
print(z1 + z2)
#type()函数用于返回数据类型，复数的输出结果为<class 'complex'>
print(type(z1 + z2))
```

5.4 布尔值

在 Python 中，可以直接使用 True 或 False 来表示布尔值。需要注意的是，布尔值的首字母一定要大写，否则会报错，示例代码如下：

```
#资源包\Code\chapter5\5.4\0506.py
bl1 = True
#输出结果为 True
print(bl1)
#type()函数用于返回数据类型，布尔值的输出结果为<class 'bool'>
```

```
print(type(bl1))
#报错,首字母必须大写
bl2 = false
```

5.5 空值

空值是 Python 里的一个特殊值,用 None 来表示。None 不能表示为 0,也不能表示为空字符串等,因为后者是有意义的,而 None 表示没有值,也就是空值,示例代码如下:

```
#资源包\Code\chapter5\5.5\0507.py
val = None
#输出结果为 None
print(None)
#type()函数用于返回数据类型,None 的输出结果为<class 'NoneType'>
print(type(None))
```

5.6 字符串

5.6.1 创建字符串

可以通过单引号或双引号包裹字符来创建字符串。在 Python 3.x 版本中,字符串是以 Unicode 编码的,也就是说,Python 的字符串支持多种语言,示例代码如下:

```
#资源包\Code\chapter5\5.6\0508.py
str1 = 'hello Python'
print(str1)
str2 = '你好,Python'
print(str2)
#type()函数用于返回数据类型,字符串的输出结果为<class 'str'>
print(type(str2))
```

如果想让字符串中的所有字符(如反斜线、元字符和换行符等具有特殊含义的字符)都直接按照其字面的意思来使用,有两种方式:一是创建原始字符串,即在字符串的第 1 个引号之前加上字母 r;二是在具有特殊含义的字符前添加反斜线。建议读者使用第 1 种方式,示例代码如下:

```
#资源包\Code\chapter5\5.6\0509.py
#原始字符串
str1 = r'hello \n world'
#输出结果为 hello \n world
print(str1)
#在具有特殊含义的字符之前添加反斜线
str2 = 'hello \\n world'
#输出结果为 hello \n world
print(str2)
```

此外，如果书写的字符串非常长，即需要跨多行时，则需要使用一对 3 个单引号（或双引号，建议读者使用单引号）代替一对单引号（或双引号）来创建字符串。注意，此种方式可以在一对 3 个单引号之间同时使用单引号或双引号，而不需要使用反斜线进行转义，示例代码如下：

```
#资源包\Code\chapter5\5.6\0510.py
str = '''name-'夏正东'
age-35
teach-"Python"'''
print(str)
```

5.6.2 访问字符串中的值

1. 索引访问

字符串是字符的有序集合，可以通过其位置来获得具体的元素。在 Python 中，字符串中的字符是通过索引来提取的，索引从 0 开始，第 1 个元素的索引为 0，第 2 个元素的索引为 1，也可以取负值，表示从末尾提取，最后一个元素的索引为－1，倒数第 2 个元素的索引为－2。

索引访问分为指定索引访问和分片索引访问。其中，分片索引获取的字符串包括开始索引的元素，但不包括结束索引的元素，示例代码如下：

```
#资源包\Code\chapter5\5.6\0511.py
str = 'www.oldxia.com'
#通过指定索引访问，获取字符串中的第 5 个元素，即 o
print(str[4])
#通过指定索引访问，获取字符串中的最后一个元素，即 m
print(str[-1])
#通过分片索引访问，获取从索引 1 开始到索引 3 的元素，即 ww.
print(str[1:4])
#通过分片索引访问，获取从索引 4 开始一直到最后的元素，即 oldxia.com
print(str[4:])
#通过分片索引访问，获取从索引 0 开始一直到索引 6 的元素，即 www.old
print(str[:7])
#通过分片索引访问，获取全部字符串，即 www.oldxia.com
print(str[:])
```

2. 循环遍历

可以通过 for 循环获取字符串中的每个元素，示例代码如下：

```
#资源包\Code\chapter5\5.6\0512.py
str = 'www.oldxia.com'
for s in str:
    print(s)
```

3. 拆包访问

对于可迭代的对象，都可以进行拆包访问。拆包是指将一个结构中的数据拆分为多个

单独的变量的值。

拆包的方式有两种，一种用单独的变量接收；另一种用“*”号接收，该种方式返回的是一个列表，示例代码如下：

```
#资源包\Code\chapter5\5.6\0513.py
str = 'www.oldxia.com'
a, b, c, *s = str
#输出的结果为 w w w ['.', 'o', 'l', 'd', 'x', 'i', 'a', '.', 'c', 'o', 'm']
print(a, b, c, s)
```

5.6.3　字符串的相关操作

1. 连接

通过“+”号将多个字符串连接到一起，示例代码如下：

```
#资源包\Code\chapter5\5.6\0514.py
str1 = '夏正东'
str2 = 'teach'
str3 = 'Python'
print(str1 + '' + str2 + '' + str3)
```

2. 拼接

通过 join()方法将字符串中的元素以指定的分隔符拼接成一个新的字符串并返回。注意，该方法不修改原字符串，其语法格式如下：

```
join(str)
```

其中，参数 str 表示需要拼接的字符串，示例代码如下：

```
#资源包\Code\chapter5\5.6\0515.py
str = 'www.oldxia.com'
#seq 表示分隔符
seq = '-'
#输出结果为 w-w-w-.-o-l-d-x-i-a-.-c-o-m
print(seq.join(str))
#不修改原字符串
print(str)
```

3. 计数

通过 count()方法统计字符串中指定字符出现的次数并返回，其语法格式如下：

```
count(sub[, start, end])
```

其中，参数 sub 表示需要统计的子字符串；参数 start 为可选参数，表示搜索的开始索引，如果省略该参数，则默认值为 0；参数 end 为可选参数，表示搜索的结束索引，如果省略该参数，则默认值为需要统计的子字符串的长度，注意，统计范围不包括结束索引对应的值，

示例代码如下：

```
#资源包\Code\chapter5\5.6\0516.py
str1 = 'http://www.oldxia.com'
print(str1.count('w'))
str2 = 'http://www.oldxia.com'
#输出结果为2,如果为str.count('w',0,10),则结果为3
print(str2.count('w', 0, 9))
```

4. 去除空格

通过strip()方法移除字符串开头和结尾指定的字符(默认为空格或换行符)并返回，并且该方法只能删除开头或结尾的指定字符，不能删除中间部分的字符。注意，该方法不修改原字符串，其语法格式如下：

```
strip([chars])
```

其中，参数chars为可选参数，表示需要移除的字符，如果省略该参数，则默认值为空格或换行符，示例代码如下：

```
#资源包\Code\chapter5\5.6\0517.py
str1 = '夏正东 '
#移除默认值空格
print(str1.strip())
#不修改原字符串
print(str1)
str2 = '123Python123'
#移除指定字符123
print(str2.strip('123'))
#不修改原字符串
print(str2)
```

5. 替换

通过replace()方法可以把字符串中的字符串替换成指定的字符串并返回。注意，该方法不修改原字符串，其语法格式如下：

```
replace(old, new[, max])
```

其中，参数old表示需要被替换的子字符串；参数new表示用于替换的子字符串；参数max为可选参数，表示最多替换的次数，如果省略该参数，则默认为全部替换，示例代码如下：

```
#资源包\Code\chapter5\5.6\0518.py
str = '夏正东 teach Python'
print(str.replace('Python', 'Linux'))
#不修改原字符串
print(str)
```

```
str = 'this is Python this is PHP this is Java'
#指定第 3 个参数,替换 2 次
print(str.replace('this', 'that', 2))
```

6. 查找

通过 find()方法查找字符串中是否包含指定的字符或字符串,如果包含,则返回该字符对应的索引或该字符串的开始索引,否则返回－1。注意,该方法不修改原字符串,其语法格式如下:

```
find(str[, start, end])
```

其中,参数 str 表示需要查找的字符或字符串;参数 start 为可选参数,表示开始索引,如果省略该参数,则默认值为 0;参数 end 为可选参数,表示结束索引,如果省略该参数,则默认值为需要查找的字符或字符串的长度。注意,查找范围不包括结束索引对应的值,示例代码如下:

```
#资源包\Code\chapter5\5.6\0519.py
str = 'this is Python this is PHP this is Java'
#查找字符对应的索引为 1
print(str.find('h'))
#查找字符串开始的索引为 2
print(str.find('is'))
#没有查找到,所以返回 - 1
print(str.find('Linux'))
#不修改原字符串
print(str)
str = 'this is Python this is PHP this is Java'
#没有查找到,因为查找范围不包括结束索引 3 对应的值
print(str.find('this', 0, 3))
#查找字符串开始的索引为 0
print(str.find('this', 0, 4))
```

7. 分割

通过 split()方法将字符串按照指定分隔符进行分割,并返回分割后的字符串列表。注意,该方法不修改原字符串,其语法格式如下:

```
split(str[, num])
```

其中,参数 str 表示分隔符;参数 num 为可选参数,表示分割的次数,如果省略该参数,则默认为全部分割,示例代码如下:

```
#资源包\Code\chapter5\5.6\0520.py
str = 'www.oldxia.com'
print(str.split('.'))
#分割 1 次
print(str.split('.', 1))
```

5.6.4 字符串格式化

在 Python 中有 3 种方式可以对字符串中的全部或部分字符串进行格式化，分别是%格式符、format()函数和 f-string。

在进行格式化时，字符串中的待格式化字符串需要使用占位符进行代替，并且需要在占位符中使用字符串格式化符号，如表 5-1 所示，用于以指定格式输出。

表 5-1 字符串格式化符号

格式化符号	描　　述
c	格式化字符
s	格式化字符串
d	格式化整数
u	格式化无符号整数
o	格式化无符号八进制数
x	格式化无符号十六进制数
X	格式化无符号十六进制数(十六进制字符大写)
f	格式化浮点数字，可指定小数点后的精度
e	用科学记数法格式化浮点数
E	用科学记数法格式化浮点数(科学记数法字符大写)
g	%f 和%e 的简写
G	%F 和%E 的简写
%	输出%

1. %格式符

当使用%格式符进行格式化时，其占位符为“%”，其字符串格式化符号需要在占位符之后，并且%格式符右侧的待格式化字符串必须与占位符一一对应，示例代码如下：

```
#资源包\Code\chapter5\5.6\0521.py
name = '夏正东'
age = 35
# %s 表示格式化字符串, %o 表示格式化无符号八进制数
print('name: %s age: %o' % (name, age))
salary = 12366.7893
#指定小数精度,保留小数点后两位
print('月薪为: %.2f' % (salary))
```

2. format()函数

当使用 format()函数进行格式化时，其占位符为“{}”，并且有两种方式表示待格式化字符串，即位置映射和关键字映射。此外，占位符中的字符串格式化符号需要在位置映射和关键字映射之后，并且两者之间需要使用冒号进行分隔，示例代码如下：

```
#资源包\Code\chapter5\5.6\0522.py
name = '夏正东'
age = 35
#位置映射
```

```
print('name:{1} age:{0}'.format(age, name))
#关键字映射
print('name:{name} age:{age}'.format(name = '夏正东', age = 35))
salary = 12366.7893
#指定小数精度,保留小数点后两位
print('月薪为:{0:.2f}'.format(salary))
```

3. f-string

f-string 是 Python 3.6 新引入的一种字符串格式化方式，主要目的是使格式化字符串的操作更加简便。

f-string 在形式上是以 f 或 F 修饰符开头的字符串，其占位符同样为“{}”。需要注意的是，f-string 的占位符表示待格式化字符串的方式不同于 format()函数，后者使用映射的方式表示待格式化字符串，而 f-string 的占位符可以直接为整数、浮点数、字符串或函数，甚至可以是对象中的属性或方法。此外，f-string 占位符中的字符串格式化符号同样需要在待格式化的字符串之后，并且其中间需要使用冒号进行分隔。

综上所述，f-string 在功能方面不逊于传统的%格式符和 format()函数，同时性能又优于二者，并且使用起来也更加简洁明了，因此对于 Python 3.6 及以后的版本，推荐读者使用 f-string 进行字符串格式化，示例代码如下：

```
#资源包\Code\chapter5\5.6\0523.py
name = '夏正东'
age = 35
def getsalary():
    return 10329.788
#占位符类型可以为函数
str = f"名字: {name},年龄: {age}岁,薪水: {getsalary()}元"
print(str)
class Person:
    def __init__(self):
        self.name = '夏正东'
        self.age = 35
        self.salary = 10329.788
    def getsalary(self):
        return self.salary
person = Person()
#占位符类型可以为对象中的属性或方法,并且可以指定小数的精度
str = f"名字: {person.name},年龄: {person.age}岁,薪水: {person.getsalary():.2f}元"
print(str)
```

5.7　列表

5.7.1　创建列表

通过中括号将元素包裹，并且元素之间使用逗号分隔，即可完成列表的创建，示例代码如下：

```
#资源包\Code\chapter5\5.7\0524.py
lt = [1, 6.66, 'name', None]
print(lt)
#type()函数用于返回数据类型,列表的输出结果为<class 'list'>
print(type(lt))
```

5.7.2 访问列表中的值

1. 索引访问

与字符串的索引一样,列表的索引也是从 0 开始的,第 1 个元素的索引为 0,第 2 个元素的索引为 1;同样也可以取负值,表示从末尾提取,最后一个元素的索引为－1,倒数第 2 个元素的索引为－2。

索引访问分为指定索引访问和分片索引访问。其中,分片索引获取的列表包括开始索引的元素,但不包括结束索引的元素,示例代码如下:

```
#资源包\Code\chapter5\5.7\0525.py
lt = [1, 2, 3, 4, 5, 6, 7, 8, 9]
#通过指定索引访问,获取列表中的第 4 个元素,即 4
print(lt[3])
#通过指定索引访问,获取列表中的倒数第 2 个元素,即 8
print(lt[ - 2])
#通过分片索引访问,获取从索引 1 开始到索引 4 的元素,即[2, 3, 4, 5]
print(lt[1:5])
#通过分片索引访问,获取从索引 3 开始一直到最后的元素,即[4, 5, 6, 7, 8, 9]
print(lt[3:])
#通过分片索引访问,获取从索引 0 开始到索引 5 的元素,即[1, 2, 3, 4, 5, 6]
print(lt[:6])
#通过分片索引访问,获取列表中的全部元素,即[1, 2, 3, 4, 5, 6, 7, 8, 9]
print(lt[:])
```

2. 循环遍历

可以通过 for 循环获取列表中的每个元素,示例代码如下:

```
#资源包\Code\chapter5\5.7\0526.py
lt = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for val in lt:
   print(val)
```

3. 拆包访问

对于可迭代的对象,都可以进行拆包访问。拆包是指将一个结构中的数据拆分为多个单独的变量的值。

拆包的方式有两种,一种用单独的变量接收;另一种用“*”号接收,该种方式返回的是一个列表,示例代码如下:

```
#资源包\Code\chapter5\5.7\0527.py
lt = [1, 2, 3, 4, 5, 6, 7]
n1, n2, n3, *n = lt
#输出的结果为 1 2 3 [4, 5, 6, 7]
print(n1, n2, n3, n)
```

5.7.3　列表的特性

列表中可以存储任意数据类型的值，并且列表中的元素值可以进行修改，示例代码如下：

```
#资源包\Code\chapter5\5.7\0528.py
lt = [1, 2, 3, 4, 5, 6, 7, 8, 9]
lt[0] = 100
#输出结果为[100, 2, 3, 4, 5, 6, 7, 8, 9]
print(lt)
```

5.7.4　列表的相关操作

1. 插入

通过 insert()方法可以将指定的元素插入列表中的指定位置。注意，该方法无返回值，但会修改原列表，其语法格式如下：

```
insert(index, obj)
```

其中，参数 index 表示索引；参数 obj 表示元素，示例代码如下：

```
#资源包\Code\chapter5\5.7\0529.py
lt = ['PHP', 'Python', 'Java']
#在索引 1 的位置插入元素
lt.insert(1, 'C++')
#修改原列表
print(lt)
```

2. 追加

通过 append()方法可以在列表的末尾添加新的元素。注意，该方法无返回值，但会修改原列表，其语法格式如下：

```
append(obj)
```

其中，参数 obj 表示元素，示例代码如下：

```
#资源包\Code\chapter5\5.7\0530.py
lt = ['PHP', 'Python', 'Java']
lt.append('C++')
#修改原列表
print(lt)
```

3. 删除

删除列表中的元素有两种方式。

1）remove()方法

该方法用于删除列表中第一次出现的指定元素。注意，该方法无返回值，但会修改原列表，其语法格式如下：

```
remove(obj)
```

其中，参数 obj 表示元素，示例代码如下：

```
#资源包\Code\chapter5\5.7\0531.py
lt = ['PHP', 'Python', 'PHP', 'Java']
lt.remove('PHP')
#删除第 1 个 PHP,输出结果为['Python', 'PHP', 'Java'],修改原列表
print(lt)
```

2）pop()方法

该方法用于删除列表中指定的元素。注意，该方法的返回值为被删除元素，并会修改原列表，其语法格式如下：

```
pop([index])
```

其中，参数 index 为可选参数，表示索引，如果省略该参数，则默认删除最后一个元素，示例代码如下：

```
#资源包\Code\chapter5\5.7\0532.py
lt1 = ['PHP', 'Python', 'Java']
#删除索引为 0 的元素
val1 = lt1.pop(0)
#返回被删除元素的值
print(val1)
#修改原列表
print(lt1)
lt2 = ['PHP', 'Python', 'Java']
#删除最后一位元素
val2 = lt2.pop()
print(val2)
#输出结果为['PHP', 'Python']
print(lt2)
```

4. 倒置

通过 reverse()方法可以倒置列表中的元素。注意，该方法无返回值，但会修改原列表，其语法格式如下：

```
reverse()
```

示例代码如下：

```
#资源包\Code\chapter5\5.7\0533.py
lt = ['PHP', 'Python', 'Java']
lt.reverse()
#修改原列表
print(lt)
```

5. 复制

通过 copy()方法可以返回一个列表的浅复制。注意，该方法返回复制之后的新列表，不会修改原列表，其语法格式如下：

```
copy()
```

示例代码如下：

```
#资源包\Code\chapter5\5.7\0534.py
lt = ['PHP', 'Python', 'Java']
new_lt = lt.copy()
print(new_lt)
```

下面，详细介绍直接赋值、浅复制和深复制之间的区别。

首先，来看一段程序，示例代码如下：

```
#资源包\Code\chapter5\5.7\0535.py
a = [1, 2, 3, 4, [1, 2, 3]]
b = a
print('a 的内存地址: {0}'.format(id(a)))
print('b 的内存地址: {0}'.format(id(b)))
a.append(100)
a[4].append(100)
#输出结果为[1, 2, 3, 4, [1, 2, 3, 100], 100]
print('列表 a: {0}'.format(a))
#输出结果为[1, 2, 3, 4, [1, 2, 3, 100], 100]
print('列表 b: {0}'.format(b))
```

上面的代码使用的是直接赋值。由于列表本身就是一个对象，所以直接赋值其本质就是对象的引用，即起别名，所以列表 a 和列表 b 的内存地址是一致的，并且由于列表 a 和列表 b 都指向同一个引用，所以列表 a 和列表 b 中的元素（包括子对象[1,2,3]）也会被同时修改，如图 5-1 所示。

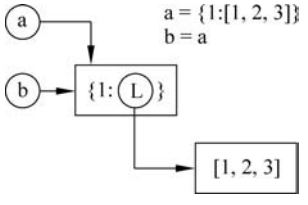

图 5-1　直接赋值

然后，再来看一段程序，示例代码如下：

```
#资源包\Code\chapter5\5.7\0536.py
a = [1, 2, 3, 4, [1, 2, 3]]
b = a.copy()
```

```
print('a 的内存地址: {0}'.format(id(a)))
print('b 的内存地址: {0}'.format(id(b)))
a.append(100)
a[4].append(100)
#输出结果为[1, 2, 3, 4, [1, 2, 3, 100], 100]
print('列表 a: {0}'.format(a))
#输出结果为列表 b: [1, 2, 3, 4, [1, 2, 3, 100]]
print('列表 b: {0}'.format(b))
```

上面的代码使用的是浅复制。由于列表本身就是一个对象，经过浅复制之后，列表 a 和列表 b 都是占有内存空间的独立对象，所以列表 a 和列表 b 的内存地址是不一致的，进而导致当修改列表 a 内的元素时，无法影响列表 b，但是，它们内部元素中的子对象[1,2,3]还是指向统一的引用，不会因为浅复制而改变，所以列表 a 和列表 b 内部元素中的子对象[1,2,3]会被同时修改，如图 5-2 所示。

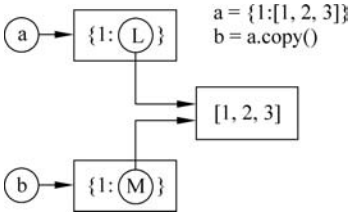

图 5-2　浅复制

最后，再来看一段程序，示例代码如下：

```
#资源包\Code\chapter5\5.7\0537.py
import copy
a = [1, 2, 3, 4, [1, 2, 3]]
b = copy.deepcopy(a)
print('a 的内存地址: {0}'.format(id(a)))
print('b 的内存地址: {0}'.format(id(b)))
a.append(100)
a[4].append(100)
#输出结果为[1, 2, 3, 4, [1, 2, 3, 100], 100]
print('列表 a: {0}'.format(a))
#输出结果为列表 b: [1, 2, 3, 4, [1, 2, 3]]
print('列表 b: {0}'.format(b))
```

上面的代码使用的是深复制，需要使用 copy 模块。由于列表本身就是一个对象，经过深复制之后，列表 a 和列表 b，以及其内部元素（包括子对象[1,2,3]）都是占有内存空间的独立对象，两个列表属于完全独立的两个对象，所以列表 a 和列表 b 的内存地址是不一致的，并且其内部元素（包括子对象[1,2,3]）也不会被同时修改，如图 5-3 所示。

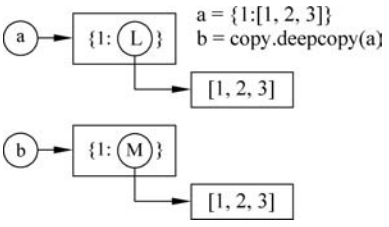

图 5-3　深复制

6. 清空

通过 clear()方法可以将列表中的元素清空。注意，该方法无返回值，但会修改原列表，其语法格式如下：

```
clear()
```

示例代码如下：

```
#资源包\Code\chapter5\5.7\0538.py
lt = ['PHP', 'Python', 'Java']
lt.clear()
#修改原列表
print(lt)
```

7. 计数

通过 count()方法可以统计在列表中指定元素出现的次数并返回，其语法格式如下：

```
count(obj)
```

其中，参数 obj 表示元素，示例代码如下：

```
#资源包\Code\chapter5\5.7\0539.py
lt = ['PHP', 'Python', 'Java','python','Python']
#区分大小写
num = lt.count('Python')
#输出结果为 2
print(num)
```

8. 查找索引

通过 index()方法查找指定元素第一次出现的索引并返回，其语法格式如下：

```
index(x[, start, end])
```

其中，参数 x 表示元素；参数 start 为可选参数，表示开始索引，如果省略该参数，则默认值为 0；参数 end 为可选参数，表示结束索引，如果省略该参数，则默认值为需要查找的字符或字符串的长度，注意，查找范围不包括结束索引对应的值，示例代码如下：

```
#资源包\Code\chapter5\5.7\0540.py
lt1 = ['PHP', 'python', 'Java', 'Python', 'Python']
#区分大小写
print(lt1.index('Python'))
lt2 = ['PHP', 'python', 'Java', 'Python', 'JavaScript', 'C++', 'Python', 'C']
#不包含结束索引对应的值,输出结果为 6
print(lt2.index('Python', 4, 7))
```

5.7.5　列表推导式

列表推导式提供了一种简明扼要的方法来创建列表，其语法格式如下：

```
[ 表达式 for 语句 [, if 语句 [, for 语句 ...] ] ]
```

列表推导式的结构是在一个中括号里包含一个表达式，之后是一个 for 语句，然后是 0

个或多个 if 语句或 for 语句，最后在这个以 if 语句和 for 语句为上下文的表达式运行完成之后，返回一个新的列表。其执行顺序为左边第 2 条语句是最外层，依次往右进一层，左边第 1 条语句是最后一层，示例代码如下：

```
#资源包\Code\chapter5\5.7\0541.py
#列表推导式的方式
lt1 = [x * y for x in range(1, 5) if x > 2 for y in range(1, 4) if y < 3]
#输出结果为[3, 6, 4, 8]
print(lt1)
#列表推导式的执行顺序
lt2 = []
for x in range(1, 5):
   if x > 2:
      for y in range(1, 4):
            if y < 3:
               lt2.append(x * y)
#输出结果同样为[3, 6, 4, 8]
print(lt2)
```

5.8 元组

5.8.1 创建元组

创建元组有两种方式，一种使用逗号将元素分隔；另一种使用小括号将元素包裹，并且元素之间使用逗号分隔。强烈推荐读者使用第 2 种方式，示例代码如下：

```
#资源包\Code\chapter5\5.8\0542.py
#使用逗号将数据分隔
tp1 = 1, 6.66, 'name', None
print(tp1)
#type()函数用来返回数据类型,元组的输出结果为<class 'tuple'>
print(type(tp1))
#使用小括号将数据包裹,并且数据之间使用逗号分隔
tp2 = (1, 6.66, 'name', None)
print(tp2)
#type()函数用于返回数据类型,元组的输出结果为<class 'tuple'>
print(type(tp2))
```

有一点需要重点强调，即当创建的元组中只有一个数据的时候，无论使用上述哪种方式创建，只要该数据之后没有逗号，则创建的不是元组，而是一个整数，所以，读者在创建只包含一个数据的元组时，一定要注意在该数据之后加上逗号，示例代码如下：

```
#资源包\Code\chapter5\5.8\0543.py
tp = 1,
#此种方式创建的是元组,输出结果为<class 'tuple'> (1,)
print(type(tp), tp)
```

```
tp = (1,)
#此种方式创建的是元组,输出结果为<class 'tuple'> (1,)
print(type(tp), tp)
tp = (1)
#此种方式创建的是整数,输出结果为<class 'int'> 1
print(type(tp), tp)
```

5.8.2　访问元组中的值

1. 索引访问

与字符串的索引一样,元组的索引也是从 0 开始,第 1 个元素的索引为 0,第 2 个元素的索引为 1;同样也可以取负值,表示从末尾提取,最后一个元素的索引为－1,倒数第 2 个元素的索引为－2。

索引访问分为指定索引访问和分片索引访问。其中,分片索引获取的元组包括开始索引的元素,但不包括结束索引的元素,示例代码如下:

```
#资源包\Code\chapter5\5.8\0544.py
tp = (1, 2, 3, 4, 5, 6, 7, 8, 9)
#通过指定索引访问,获取元组中的第 4 个元素,即 4
print(tp[3])
#通过指定索引访问,获取元组中的倒数第 2 个元素,即 8
print(tp[-2])
#通过分片索引访问,获取从索引 1 开始到索引 4 的元素,即(2, 3, 4, 5)
print(tp[1:5])
#通过分片索引访问,获取从索引 3 开始一直到最后的元素,即(4, 5, 6, 7, 8, 9)
print(tp[3:])
#通过分片索引访问,获取从索引 0 开始到索引 5 的元素,即(1, 2, 3, 4, 5, 6)
print(tp[:6])
#通过分片索引访问,获取元组中的全部元素,即(1, 2, 3, 4, 5, 6, 7, 8, 9)
print(tp[:])
```

2. 循环遍历

可以通过 for 循环获取元组中的每个元素,示例代码如下:

```
#资源包\Code\chapter5\5.8\0545.py
tp = (1, 2, 3, 4, 5, 6, 7, 8, 9)
for val in tp:
    print(val)
```

3. 拆包访问

对于可迭代的对象,都可以进行拆包访问。拆包是指将一个结构中的数据拆分为多个单独的变量的值。

拆包的方式有两种,一种用单独的变量接收;另一种用“*”号接收,该种方式返回的是一个列表,示例代码如下:

```
#资源包\Code\chapter5\5.8\0546.py
tp = (1, 2, 3, 4, 5, 6, 7)
n1, n2, n3, *n = tp
#输出结果为 1 2 3 [4, 5, 6, 7]
print(n1, n2, n3, n)
```

5.8.3 元组的特性

元组属于不可变的数据类型，其元素值不允许修改，所以元组与列表不一样，元组不具有添加、删除和修改其内部元素的方法，示例代码如下：

```
#资源包\Code\chapter5\5.8\0547.py
tp = (1, 2, 3, 4, 5, 6, 7)
#报错
tp[0] = 100
```

5.8.4 元组的相关操作

1. 连接

元组中的元素值是不允许修改的，但可以对元组进行连接，组合成一个新的元组，示例代码如下：

```
#资源包\Code\chapter5\5.8\0548.py
tp1 = (1, 2, 3)
tp2 = ('a', 'b', 'c')
tp = tp1 + tp2
#输出结果为(1, 2, 3, 'a', 'b', 'c')
print(tp)
```

2. 删除

元组中的元素值是不允许删除的，但可以使用 del 语句来删除整个元组，示例代码如下：

```
#资源包\Code\chapter5\5.8\0549.py
tp = (1, 2, 3)
del tp
#由于元组已经被删除，所以会报错，提示 tp 未定义
print(tp)
```

5.8.5 元组推导式

元组推导式提供了一种简明扼要的方法来创建元组，其语法格式如下：

```
( 表达式 for 语句 [, if 语句 [, for 语句 ...] ] )
```

元组推导式的结构是在一个小括号里包含一个表达式，之后是一个 for 语句，然后是 0 个或多个 if 语句或 for 语句，注意，与列表推导式不同，最后在这个以 if 语句和 for 语句为上下文的表达式运行完成之后，返回的是一个生成器对象，需要使用 tuple()函数来获得新元组。其执行顺序为左边第 2 条语句是最外层，依次往右进一层，左边第一条语句是最后一层，示例代码如下：

```
#资源包\Code\chapter5\5.8\0550.py
#元组推导式的方式
tp1 = (x * y for x in range(1, 5) if x > 2 for y in range(1, 4) if y < 3)
#输出结果为一个生成器对象
print(tp1)
#输出结果为(3, 6, 4, 8)
print(tuple(tp1))
#元组推导式的执行顺序
lt = []
for x in range(1, 5):
    if x > 2:
        for y in range(1, 4):
            if y < 3:
                lt.append(x * y)
#输出结果同样为(3, 6, 4, 8)
print(tuple(lt))
```

5.9 字典

5.9.1 创建字典

字典由多个键和其对应的值构成的键值对组成，键和值中间以冒号分隔，每个键值对之间使用逗号分隔，最后，由大括号将每个键值对包裹，即可完成字典的创建，示例代码如下：

```
#资源包\Code\chapter5\5.9\0551.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
#输出结果为{'name': '夏正东', 'age': 35, 'teach': 'Python'}
print(dt)
#type()函数用于返回数据类型,字典的输出结果为<class 'dict'>
print(type(dt))
```

5.9.2 访问字典中的键

1. keys()方法访问

该方法返回一个可迭代对象，对象中包含了字典里所有的键。该可迭代对象的类型为 dict_keys，可以使用数据类型转换函数将其转换为指定的数据类型，示例代码如下：

```
#资源包\Code\chapter5\5.9\0552.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
items = dt.keys()
#类型为<class 'dict_keys'>
print(type(items))
#使用 list()函数将其转换为列表,输出结果为['name', 'age', 'teach']
print(list(items))
#使用 tuple()函数将其转换为元组,输出结果为('name', 'age', 'teach')
print(tuple(items))
```

2. 循环遍历

可以使用 for 循环直接获取字典中的键,示例代码如下:

```
#资源包\Code\chapter5\5.9\0553.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
for key in dt:
    print(key)
```

5.9.3 访问字典中的值

1. 键值访问

通过字典中的键,即可访问字典中相对应的值,示例代码如下:

```
#资源包\Code\chapter5\5.9\0554.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
print(dt['name'])
```

2. get()方法访问

该方法返回指定键的值,如果键不在字典中,则返回默认值 None 或者设置的默认值,示例代码如下:

```
#资源包\Code\chapter5\5.9\0555.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
item = dt.get('name')
print(item)
#如果键值不存在,则返回设置的默认值: 身高是秘密!
item = dt.get('height','身高是秘密!')
print(item)
```

3. values()方法访问

该方法返回一个可迭代对象,对象中包含了字典里所有的值。该可迭代对象的类型为 dict_values,可以使用数据类型转换函数将其转换为指定的数据类型,示例代码如下:

```
#资源包\Code\chapter5\5.9\0556.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
items = dt.values()
```

```
#类型为<class 'dict_values'>
print(type(items))
#使用 list()函数将其转换为列表,输出结果为['夏正东', 35, 'Python']
print(list(items))
#使用 tuple()函数将其转换为元组,输出结果为('夏正东', 35, 'Python')
print(tuple(items))
```

4. 循环遍历

可以使用 for 循环获取字典中的键,然后使用键值访问,从而获取字典中的值,示例代码如下:

```
#资源包\Code\chapter5\5.9\0557.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
for key in dt:
    print(dt[key])
```

5.9.4　访问字典中的键和值

1. items()方法访问

该方法返回一个可迭代对象,对象中包含了字典里所有键值对组成的元组。该可迭代对象的类型为 dict_items,可以使用数据类型转换函数将其转换为指定的数据类型,示例代码如下:

```
#资源包\Code\chapter5\5.9\0558.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
items = dt.items()
#类型为<class 'dict_items'>
print(type(items))
#使用 list()函数将其转换为列表,输出结果为[('name', '夏正东'), ('age', 35), ('teach', 'Python')]
print(list(items))
#使用 tuple()函数将其转换为元组,输出结果为(('name', '夏正东'), ('age', 35), ('teach',
'Python'))
print(tuple(items))
```

2. 循环遍历

可以使用 for 循环直接获取字典中的键,然后使用键值访问,获取字典中的值,示例代码如下:

```
#资源包\Code\chapter5\5.9\0559.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
for key in dt:
    print(key, dt[key])
```

5.9.5　字典的特性

字典中的值可以是任何 Python 对象,既可以是标准的对象,也可以是用户定义的对

象，但键不行。另外，有两点需要注意，一是在字典中不允许同一个键出现两次，在创建字典时，如果同一个键被赋值两次，则前一个键值对会被后一个键值对覆盖；二是字典中的键必须是不可变的，所以可以用数字、字符串或元组等充当，但是，列表不可以作为键，因为列表是可变的数据类型，示例代码如下：

```
#资源包\Code\chapter5\5.9\0560.py
dt1 = {'name': '夏正东', 'age': 35, 'teach': 'Python', 'teach': 'PHP'}
#第 3 个键值对被第 4 个键值对覆盖
print(dt1)
dt2 = {1: 'one', 'name': '夏正东', ('tp1', 'tp2'): 'tuple'}
#不变的数据类型可以作为字典的键
print(dt2)
dt3 = {1: 'one', 'name': '夏正东', ['lt1', 'lt2']: 'list'}
#报错，可变的数据类型不可以作为字典的键
print(dt3)
```

5.9.6 字典的相关操作

1. 添加

通过向字典中增加新的键值对的方式，来向字典中添加新的内容，示例代码如下：

```
#资源包\Code\chapter5\5.9\0561.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python'}
dt['city'] = '大连'
#输出结果为{'name': '夏正东', 'age': 35, 'teach': 'Python', 'city': '大连'}
print(dt)
```

2. 修改

通过修改字典中键值对的方式，来改修字典中已经存在的内容，示例代码如下：

```
#资源包\Code\chapter5\5.9\0562.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python', 'city': '大连'}
dt['teach'] = 'Linux'
#输出结果为{'name': '夏正东', 'age': 35, 'teach': 'Linux', 'city': '大连'}
print(dt)
```

3. 删除

删除字典中的键值对有 3 种方式。

1）del 关键字

该关键字可以删除字典中指定的键值对，示例代码如下：

```
#资源包\Code\chapter5\5.9\0563.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python', 'city': '大连'}
del dt['city']
#输出结果为{'name': '夏正东', 'age': 35, 'teach': 'Python'}
print(dt)
```

2）pop()方法

该方法可以删除字典中指定的键值对。注意，该方法的返回值为被删除的键所对应的值，并修改原字典，其语法格式如下：

```
pop(key[, default])
```

其中，参数 key 表示键；参数 default 为可选参数，表示当删除的键不存在时返回的值，如果省略该参数，则默认无返回值，示例代码如下：

```
#资源包\Code\chapter5\5.9\0564.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python', 'city': '大连'}
val = dt.pop('city')
print(val)
#修改原字典
print(dt)
#给定默认值
val = dt.pop('phone','秘密!')
print(val)
print(dt)
```

3）popitem()方法

该方法可以删除字典中的最后一个键值对。注意，该方法返回被删除键值对组成的元组，并修改原字典，其语法格式如下：

```
popitem()
```

示例代码如下：

```
#资源包\Code\chapter5\5.9\0565.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python', 'city': '大连'}
val = dt.popitem()
#输出结果为('city', '大连')
print(val)
#修改原字典
print(dt)
```

4. 清空

通过 clear()方法删除字典内所有的键值对。注意，该方法无返回值，但会修改原字典，其语法格式如下：

```
clear()
```

示例代码如下：

```
#资源包\Code\chapter5\5.9\0566.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python', 'city': '大连'}
dt.clear()
#修改原字典
print(dt)
```

5. 复制

通过 copy()方法可以返回一个字典的浅复制(关于直接赋值、浅复制和深复制的区别，参考列表的相关操作中的复制)。注意，该方法返回复制之后的新字典，不会修改原字典，其语法格式如下：

```
copy()
```

示例代码如下：

```
#资源包\Code\chapter5\5.9\0567.py
dt = {'name': '夏正东', 'age': 35, 'teach': 'Python', 'city': '大连'}
new_dt = dt.copy()
print(new_dt)
```

5.9.7 字典推导式

字典推导式提供了一种简明扼要的方法来创建字典，其语法格式如下：

```
{ 表达式 for 语句 [, if 语句 [, for 语句 ...] ] }
```

字典推导式的结构是在一个大括号里包含一个表达式，之后是一个 for 语句，然后是 0 个或多个 if 语句或 for 语句，最后在这个以 if 语句和 for 语句为上下文的表达式运行完成之后，返回一个新的字典。其执行顺序为左边第 2 条语句是最外层，依次往右进一层，左边第 1 条语句是最后一层，示例代码如下：

```
#资源包\Code\chapter5\5.9\0568.py
input_dt = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
#字典推导式的方式
dt1 = {key: val for key, val in input_dt.items() if val % 2 == 0}
#输出结果为一个新的字典{'two': 2, 'four': 4}
print(dt1)
#字典推导式的执行顺序
input_dt = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
dt2 = {}
for key, val in input_dt.items():
    if val % 2 == 0:
        dt2[key] = val
#输出结果同样为一个新的字典{'two': 2, 'four': 4}
print(dt2)
```

5.10 集合

5.10.1 创建集合

创建可变集合有两种方式，一种通过大括号将元素包裹，并且元素之间使用逗号分隔；另一种通过 set()函数创建，该函数的参数为可迭代对象。

注意，集合中的元素不能为字典，否则会产生语法错误，示例代码如下：

```
#资源包\Code\chapter5\5.10\0569.py
st1 = {1, 6.66, 'name', None}
#创建可变集合,输出结果为{None, 1, 6.66, 'name'}
print(st1)
#type()函数用于返回数据类型,可变集合的输出结果为<class 'set'>
print(type(st1))
#创建可变集合,参数为可迭代对象字符串
st2 = set('abcdefg')
print(st2)
#创建可变集合,参数为可迭代对象列表
st3 = set([1, 6.66, 'name', None])
print(st3)
#创建可变集合,参数为可迭代对象集合
st4 = set({1, 6.66, 'name', None})
print(st4)
```

创建不可变集合可以使用 frozenset()函数，该函数的参数为可迭代对象，示例代码如下：

```
#资源包\Code\chapter5\5.10\0570.py
#创建不可变集合,参数为可迭代对象字符串
st1 = frozenset('abcdefg')
print(st1)
#type()函数用于返回数据类型,不可变集合的输出结果为<class 'frozenset'>
print(type(st1))
#创建不可变集合,参数为可迭代对象列表
st2 = frozenset([1, 6.66, 'name', None])
print(st2)
#创建不可变集合,参数为可迭代对象集合
st3 = frozenset({1, 6.66, 'name', None})
print(st3)
```

注意，创建空的可变集合或不可变集合，必须使用 set()函数或 frozenset()函数，因为{}表示的是空字典，示例代码如下：

```
#资源包\Code\chapter5\5.10\0571.py
#数据类型为<class 'set'>
print(type(set()))
#数据类型为<class 'frozenset'>
print(type(frozenset()))
#数据类型为<class 'dict'>
print(type({}))
```

5.10.2　访问集合中的值

在集合中，无论是可变集合，还是不可变集合，都无法通过索引访问元素，只能通过 for 循环，获取可变集合或不可变集合中的元素，示例代码如下：

```
#资源包\Code\chapter5\5.10\0572.py
st1 = set({1, 6.66, 'name', None})
#报错,提示集合不能通过索引访问
print(st1[0])
st2 = frozenset({1, 6.66, 'name', None})
#报错,提示集合不能通过索引访问
print(st2[0])
st3 = set({1, 6.66, 'name', None})
for val1 in st3:
   print(val1)
st4 = frozenset({1, 6.66, 'name', None})
for val2 in st4:
   print(val2)
```

5.10.3 集合的特性

集合是一个无序的数据类型,并且其内部的元素值不可重复,示例代码如下:

```
#资源包\Code\chapter5\5.10\0573.py
st1 = {1, 'name', 1, 'age', 6.66, 'name', None}
#输出结果为{1, 'age', 6.66, 'name', None}
print(st1)
st2 = frozenset({1, 'name', 1, 'age', 6.66, 'name', None})
#输出结果为 frozenset({1, 'age', 6.66, 'name', None})
print(st2)
```

不可变集合属于不可变的数据类型,其元素值不允许修改,所以不可变集合不具有添加、删除和修改其内部元素的方法。

5.10.4 集合的相关操作

1. 添加

通过 add()方法给可变集合添加元素,如果添加的元素在集合中已存在,则不执行任何操作。注意,该方法无返回值,但会修改原集合,其语法格式如下:

```
add(element)
```

其中,参数 element 表示元素,示例代码如下:

```
#资源包\Code\chapter5\5.10\0574.py
st = {1, 6.66, 'name', None}
st.add('Python')
print(st)
```

2. 删除

删除可变集合中的元素有 3 种方式。

1) remove()方法

该方法可以删除可变集合中的指定元素,如果删除一个不存在的元素,该方法则会发生

错误。注意，该方法无返回值，但会修改原集合，其语法格式如下：

```
remove(element)
```

其中，参数 element 表示元素，示例代码如下：

```
#资源包\Code\chapter5\5.10\0575.py
st = {1, 6.66, 'name', None}
st.remove('name')
print(st)
st.remove('100')
#报错，删除不存在的元素
print(st)
```

2）discard()方法

该方法可以删除可变集合中的指定元素，如果删除一个不存在的元素，该方法则不会发生错误，而是直接输出原集合。注意，该方法无返回值，但会修改原集合，其语法格式如下：

```
discard(element)
```

其中，参数 element 表示元素，示例代码如下：

```
#资源包\Code\chapter5\5.10\0576.py
st = {1, 6.66, 'name', None}
st.discard('name')
print(st)
st.discard('100')
#删除不存在的元素，不会报错，直接输出原集合
print(st)
```

3）pop()方法

该方法可以删除可变集合中的一个随机元素。注意，该方法无返回值，但会修改原集合，其语法格式如下：

```
pop()
```

示例代码如下：

```
#资源包\Code\chapter5\5.10\0577.py
st = {1, 6.66, 'name', None}
st.pop()
print(st)
```

3. 清空

通过 clear()方法清空可变集合。注意，该方法无返回值，但会修改原集合，其语法格式如下：

```
clear()
```

示例代码如下：

```
#资源包\Code\chapter5\5.10\0578.py
st = set({1, 6.66, 'name', None})
st.clear()
print(st)
```

4. 复制

通过 copy()方法可以返回一个可变集合或不可变集合的浅复制。注意，该方法返回复制之后的新集合，不会修改原集合，其语法格式如下：

```
copy()
```

示例代码如下：

```
#资源包\Code\chapter5\5.10\0579.py
st1 = {1, 6.66, 'name', None}
new_st1 = st1.copy()
print(new_st1)
st2 = frozenset({1, 6.66, 'name', None})
new_st2 = st2.copy()
print(new_st2)
```

5.10.5 集合推导式

集合推导式提供了一种简明扼要的方法来创建集合，其语法格式如下：

```
{ 表达式 for 语句 [, if 语句 [, for 语句 ...] ] }
```

集合推导式的结构是在一个大括号里包含一个表达式，之后是一个 for 语句，然后是 0 个或多个 if 语句或 for 语句，最后在这个以 if 语句和 for 语句为上下文的表达式运行完成之后，返回一个新的集合。其执行顺序为左边第 2 条语句是最外层，依次往右进一层，左边第 1 条语句是最后一层，示例代码如下：

```
#资源包\Code\chapter5\5.10\0580.py
#集合推导式的方式
st1 = {x * y for x in range(1, 5) if x > 2 for y in range(1, 4) if y < 3}
#输出结果为{8, 3, 4, 6}
print(st1)
#集合推导式的执行顺序
st2 = set()
for x in range(1, 5):
    if x > 2:
        for y in range(1, 4):
            if y < 3:
                st2.add(x * y)
#输出结果同样为{8, 3, 4, 6}
print(st2)
```