第 2 章　Python 的基础语法

要想精通一门编程语言，首先需要学会基本的语法和语义规范。Python 的语言特性简洁明了，当运行一个功能时，Python 通常只使用一种固定的方式。虽然不像其他计算机语言有丰富的语法格式，但是 Python 可以完成其他计算机语言所能完成的功能，而且更容易。本章主要讲述 Python 的一些基本语法。

- 熟悉 Python 的程序结构
- 掌握 Python 的输入和输出方法
- 熟悉定义和使用变量的方法
- 熟悉标识符和保留字
- 掌握简单数据类型的使用方法
- 熟悉 Python 中的结构数据类型
- 掌握运算符的使用方法

2.1　程序结构

学习 Python 开发之前，首先需要了解 Python 的程序结构。

2.1.1　缩进分层

与其他常见的语言不同，Python 的代码块不使用大括号（{}）来控制类、函数及其他逻辑判断。Python 语言的主要特色就是用缩进分层来写模块。

【例 2.1】 严格执行缩进（源代码 2.1.py）。

```
if True:
    print ("牧童骑黄牛")
    print ("歌声振林樾")    #严格执行缩进
else:
    print ("意欲捕鸣蝉")
    print ("忽然闭口立")
```

保存并运行程序，结果如图 2-1 所示。

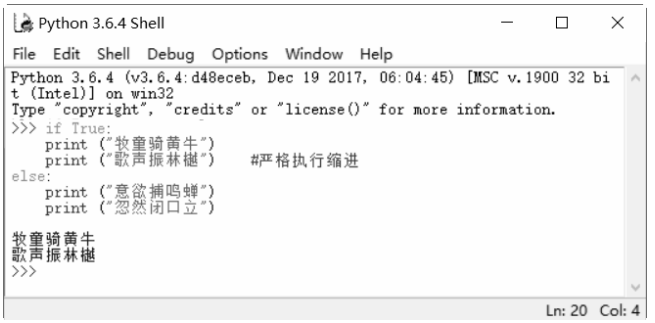

图 2-1 程序运行结果

这里位于>>>号之后的都是用户输入的 Python 语句，按 Enter 键后会立即执行；没有在>>>号之后的都是 Python 语句运行时的输出信息。

Python 程序中缩进的空白数量虽然是可变的，但是所有代码块语句必须包含相同的缩进空白数量，这个要严格执行。

【例 2.2】 没有严格执行缩进（源代码 2.2.py）。

```
if True:
    print ("牧童骑黄牛")
print ("歌声振林樾")    #没有严格执行缩进
else:
    print ("意欲捕鸣蝉")
    print ("忽然闭口立")
```

保存并运行程序，结果如图 2-2 所示。

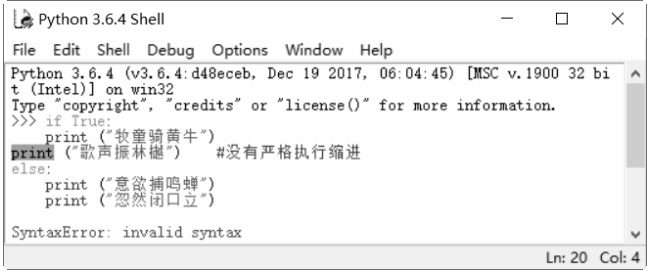

图 2-2 程序运行结果

除了保证相同的缩进空白数量，还要保证相同的缩进方式，因为有的使用 Tab 键缩进，有的使用两个或四个空格缩进，需要改为相同的方式。

Python 的编程规范指出：缩进最好采用空格的形式，每一层向右缩进 4 个空格。

2.1.2 换行问题

在 Python 语言中，常见的换行问题如下：

1. 换行符

如果是 Linux/UNTX 操作系统，换行字符为 ASCII LF（linefeed）；如果是 DOS/Windows 操作系统，换行字符为 ASCII CR LF（return + linefeed）；如果是 Mac OS 操作系统，换行字符为 ASCII CR（return）。

例如，在 Windows 操作系统中换行：

```
>>>print ("牧童骑黄牛\n 歌声振林樾")
```

运行结果如图 2-3 所示。

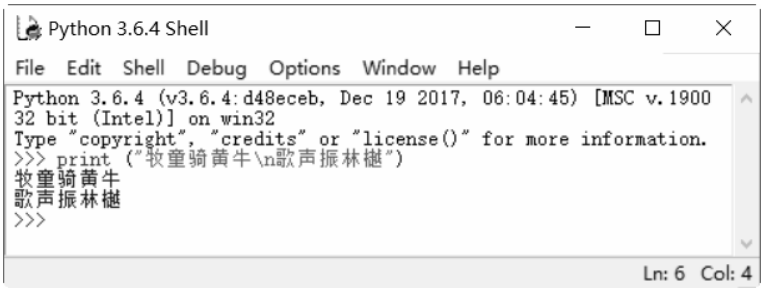

图 2-3 程序运行结果

2. 程序代码超过一行

如果程序代码超过一行，可以在每一行的结尾添加反斜杠（\），继续下一行，这与 C/C++ 的语法相同。例如：

```
if 1900 < year < 2100 and 1 <=month <=12\
   and 1 <= day <= 31 and 0 <= hour < 24 \
   and 0 <= minute < 60 and 0 <= second < 60:   #多个判断条件
```

注 意

每个行末的反斜杠(\)之后不加注释文字。

如果是以小括号()、中括号[]或大括号{}包含起来的语句，不必使用反斜杠(\)就可以直接分成数行。例如：

```
>>>month_names = ['Januari', 'Februari',  'Maart',
           'April',   'Mei',     'Juni',
           'Juli',    'Augustus', 'September',
           'Oktober', 'November', 'December']
```

3. 将数行表达式写成一行

如果要将数行表达式写成一行，只需在每一行的结尾添加上分号(;)即可。例如：

```
>>>x = 100; y = 200; z = 300
>>> x
100
>>> y
200
>>> z
300
```

2.1.3 代码注释

Python 中的注释有单行注释和多行注释。Python 中单行注释以#开头，例如：

```
# 这是一个注释
print("Hello, World!")
```

多行注释用 3 个单引号（'''）或 3 个双引号（"""）将注释括起来。

（1）3 个单引号。

```
'''
这是多行注释，用 3 个单引号
这是多行注释，用 3 个单引号
这是多行注释，用 3 个单引号
'''
print("这是 Python 语言的注释")
```

（2）3 个双引号。

```
"""
这是多行注释，用 3 个双引号
这是多行注释，用 3 个双引号
这是多行注释，用 3 个双引号
"""
print("这是 Python 语言的注释")
```

2.2 Python 的输入和输出

Python 的内置函数 input()和 print()用于输入和输出数据。下面将讲述这两个函数的使用方法。

2.2.1 接收键盘输入

Python 提供的 input() 函数从标准输入读入一行文本，默认的标准输入是键盘。input()函数的基本语法格式如下：

```
input([prompt])
```

其中，prompt 是可选参数，用来显示用户输入的提示信息字符串。用户输入程序所需要的数据时，就会以字符串的形式返回。

【例 2.3】 测试键盘的输入。

```
x= input("请输入最喜欢的水果：")
```

上述代码用于提示用户输入水果的名称，然后将名称以字符串的形式返回并保存在 x 变量中，以后可以随时调用这个变量。

测试结果如图 2-4 所示。当运行此句代码时，会立即显示提示信息“请输入最喜欢的水果：”，之后等待用户输入信息。当用户输入“葡萄”并按下 Enter 键时，程序就接收了用户的输入。最后调用 x 变量，就会显示变量所引用的对象——用户输入的水果名称。

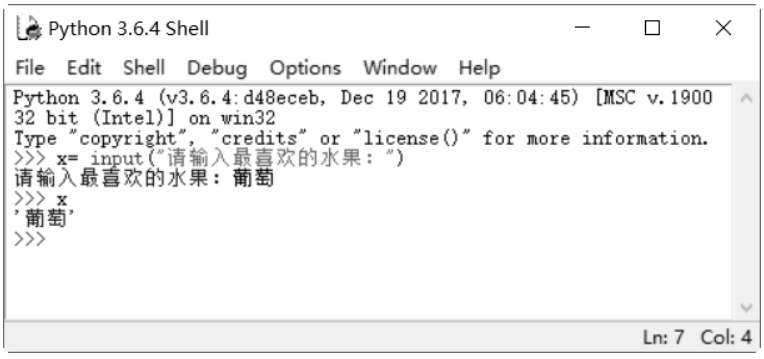

图 2-4 程序运行结果

从结果可以看出，添加提示用户输入信息是比较友好的，对于编程时所需要的友好界面非常有帮助。

注 意

用户输入的数据全部以字符串形式返回，如果需要输入数值，就必须进行类型转换。

2.2.2 输出处理结果

print ()函数可以输出格式化的数据，与 C/C++的 printf()函数功能和格式相似。print()函数的基本语法格式如下：

```
print(value,…, sep=' ' ,end='\n')    #此处只说明了部分参数
```

上述参数的含义如下：

（1）value 是用户要输出的信息，后面的省略号表示可以有多个要输出的信息。
（2）sep 用于设置多个要输出信息之间的分隔符，其默认的分隔符为一个空格。
（3）end 是一个 print()函数中所有要输出信息之后添加的符号，默认值为换行符。

【例 2.4】 测试处理结果的输出（源代码 2.3.py）。

```
print("牧童骑黄牛","歌声振林樾")              #输出测试的内容
print("牧童骑黄牛","歌声振林樾",sep='*')       #将默认分隔符修改为'*'
print("牧童骑黄牛","歌声振林樾",end='>')       #将默认的结束符修改为'>'
print("牧童骑黄牛","歌声振林樾")              #再次输出测试的内容
```

保存并运行程序，结果如图 2-5 所示。这里调用了 4 次 print()函数。其中，第 1 次为默认输出，第 2 次将默认分隔符修改为'*'，第 3 次将默认的结束符修改为'>'，第 4 次再次调用默认的输出。

图 2-5 程序运行结果

从运行结果可以看出，第一行为默认输出方式，数据之间用空格分开，结束后添加了一个换行符；第二行输出的数据项之间以'*'分开；第三行输出结束后添加了一个'>'，与第 4 条语句的输出放在了同一行中。

注 意

从 Python 3 开始，将不再支持 print 输出语句，例如 print "Hello Python"，解释器将会报错。

如果输出的内容既包括字符串，又包含变量值，就需要将变量值格式化处理。

例如：

```
>>>x = 100
>>>print ("x = %d" % x)
```

运行结果如图 2-6 所示。

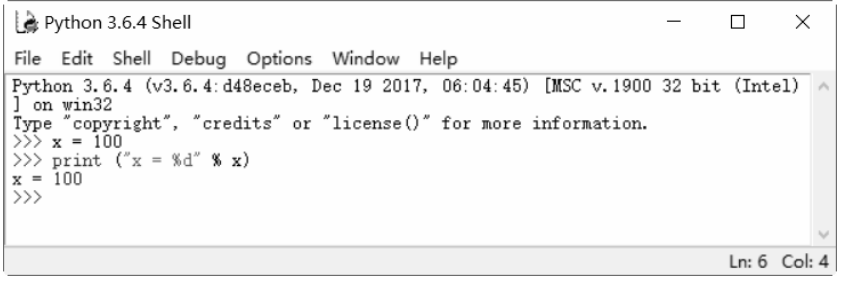

图 2-6 程序运行结果

这里要将字符串与变量之间以（%）符号隔开。

如果没有使用（%）符号将字符串与变量隔开，Python 就会输出字符串的完整内容，而不会输出格式化字符串。运行结果如图 2-7 所示。

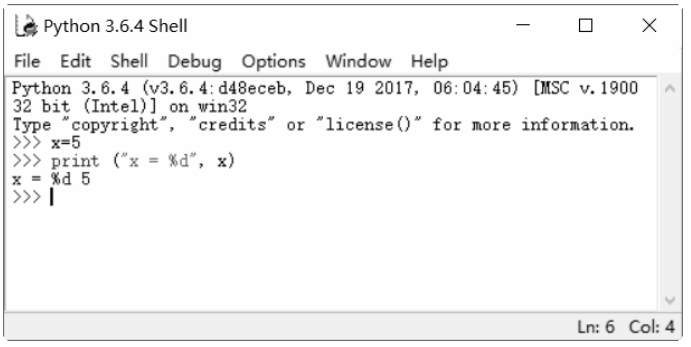

图 2-7　程序运行结果

【例 2.5】 实现不换行输出（源代码 2.4.py）。

```
a="春风又绿江南岸，"
b="明月何时照我还。"
#换行输出
print( a )
print( b )

print('---------')
# 不换行输出
print( a, end=" " )
print( b, end=" " )
print()
```

保存并运行程序，结果如图 2-8 所示。

图 2-8　程序运行结果

在本示例中，通过在变量末尾添加 end=""，可以实现不换行输出的效果。读者从结果可以看出换行和不换行的不同之处。

2.3 变量

在 Python 解释器内可以直接声明变量的名称，不必声明变量的类型，Python 会自动判别变量的类型。例如，声明一个变量 x，其值为 100：

```
>>>x =100
>>>x
100
```

例如，声明一个变量 y，其值为 15：

```
>>>y=15
>>>print(y)
15
```

读者可以在解释器内直接做数值计算，例如：

```
>>>100 + 200
300
```

当用户输入一个变量后，Python 会记住这个变量的值。例如：

```
>>> x =20
>>>y =x + 30
>>>y
50
```

Python 中的变量不需要声明。每个变量在使用前都必须赋值，变量赋值以后才会被创建。如果创建变量时没有赋值，会提示错误，例如：

```
>>> u
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    u
NameError: name 'u' is not defined
```

在 Python 中，变量就是变量，没有类型，这里所说的“类型”是变量所指的内存中对象的类型。等号（=）用来给变量赋值。等号运算符左边是一个变量名，等号运算符右边是存储在变量中的值。

Python 允许用户同时为多个变量赋值。例如：

```
>>>a =b =c =100
>>>print(a,b,c)
100 100 100
```

创建一个整型对象，值为 100，3 个变量被分配到相同的内存空间上。

也可以同时为多个对象指定不同的变量值，例如：

```
>>>a, b, c = 100, 200, "春花秋月何时了"
>>>print(a,b,c)
100 200 春花秋月何时了
```

两个整型对象 100 和 200 分配给变量 a 和 b，字符串对象"春花秋月何时了"分配给变量 c。两个变量可以相互赋值，例如：

```
>>> a,b = b,a
>>> a = 50
>>> b
50
```

2.4 标识符与保留字

标识符用来识别变量、函数、类、模块及对象的名称。Python 的标识符可以包含英文字母（A-Z、a-z）、数字（0-9）及下画线符号（_），但它有以下几个方面的限制：

（1）标识符的第 1 个字符必须是字母表中的字母或下画线（_），并且变量名称之间不能有空格。

（2）Python 的标识符有大小写之分，如 Data 与 data 是不同的标识符。

（3）在 Python 3 中，非 ASCII 标识符也被允许使用。

（4）保留字不可以当作标识符。

保留字也叫关键字，不能把它们用作任何标识符名称。读者可以使用以下命令查看 Python 的保留字：

```
>>>import keyword
>>>keyword.kwlist
```

运行结果如图 2-9 所示。

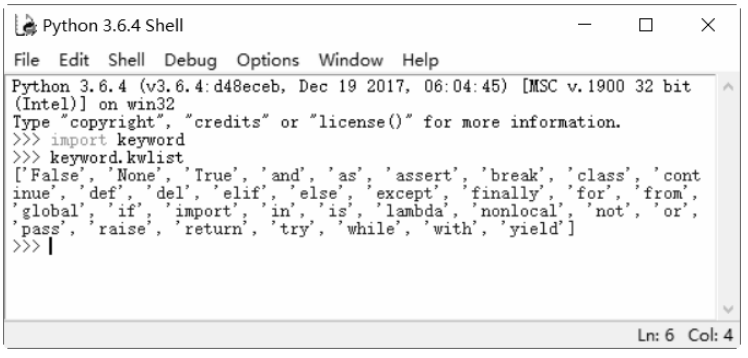

图 2-9　查看 Python 的保留字

2.5 简单数据类型

Python 3 中有两个简单的数据类型，即数字类型和字符串类型。

2.5.1 数字类型

Python 3 支持 int、float、bool、complex（复数）4 种数字类型。

在 Python 2 中是没有 bool（布尔型）的，用数字 0 表示 False，用 1 表示 True。在 Python 3 中，把 True 和 False 定义成了关键字，但它们的值还是 1 和 0，可以和数字相加。

1. int（整数）

下面是整数的示例：

```
>>> a = 123456
>>> a
123456
```

可以使用十六进制数值来表示整数，十六进制整数的表示法是在数字之前加上 0x，如 0x80120000、0x100010100L。

例如：

```
>>> a=0x4EEFFFFF
>>> a
1324351487
```

2. float（浮点数）

浮点数的表示法可以使用小数点，也可以使用指数的类型。指数符号可以使用字母 e 或 E 来表示，指数可以使用+/－符号，也可以在指数数值前加上数值 0，还可以在整数前加上数值 0。

例如：

```
3.14    10.     .001     1e100     3.14E-10     1e010    08.1
```

使用 float()内置函数可以将整数数据类型转换为浮点数数据类型，例如：

```
>>> float(150)
150.0
```

3. bool（布尔值）

Python 的布尔值包括 True 和 False，只与整数中的 1 和 0 有对应关系。例如：

```
>>> True==1
True
```

```
>>> True==2
False
>>> False==0
True
>>> False==-1
False
```

这里利用符号（==）判断左右两边是否绝对相等。

4. complex（复数）

复数的表示法是使用双精度浮点数来表示实数与虚数的部分，复数的符号可以使用字母 j 或 J。

例如：

```
1.5 + 0.5j      1J       2 + 1e100j        3.14e-10j
```

可以使用 real 与 imag 属性分别取出复数的实数和虚数部分，例如：

```
>>> a=2.6+0.8j
>>> a.real
2.6
>>> a.imag
0.8
>>> a
(2.6+0.8j)
```

可以使用 complex(real,imag)函数将 real 与 imag 两个数值转换为复数。real 参数是复数的实数部分，imag 参数是复数的虚数部分。例如：

```
>>> complex(2.6,0.8)
(2.6+0.8j)
```

数值之间可以通过运算符进行运算操作，例如：

```
>>> 50 + 40  # 加法
90
>>> 5.6 - 2 # 减法
3.6
>>> 30 * 15  # 乘法
450
>>> 1/2  # 除法，得到一个浮点数
0.5
>>> 1//2  # 除法，得到一个整数
0
>>> 15 % 2 # 取余
```

```
1
>>> 2 ** 10  # 乘方
1024
```

在数字运算时，需要注意以下问题：

（1）数值的除法（/）总是返回一个浮点数，要获取整数使用（//）操作符。
（2）在整数和浮点数混合计算时，Python 会把整型转换为浮点数。

用户可以将数值使用在函数内，例如：

```
>>> round(12.32, 1)
12.3
```

可以对数值进行比较，但不可以对复数进行比较，例如：

```
>>>x = 2
>>>0 < x < 5
Ture
>>> 0.5 + 1.5j < 2j
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    0.5 + 1.5j < 2j
TypeError: unorderable types: complex() < complex()
```

可以将数值做位移动（shifting）或屏蔽（masking），例如：

```
>>>16 << 2
64
>>>30 & 0x1B
26
>>>2 | 5
7
>>>3 ^ 5
6
>>>~2
-3
```

2.5.2 字符串类型

Python 将字符串视为一连串的字符组合。例如，字符串“Parrot”在 Python 内部被视为"P" "a" "r" "r" "o" "t"6 个字符的组合。因为第 1 个字符的索引值永远是 0，所以存取字符串“Parrot”的第 1 个字符“P”时使用"Parrot"[0]。例如：

```
>>> "Parrot"[0]
'P'
>>> "Parrot"[1]
```

```
'a'
```

要创建一个字符串时，可以将数个字符以英文单引号、双引号或三引号包含起来，例如：

```
>>>a = "Parrot"
>>>a
'Parrot'
>>>a = 'Parrot'
>>> a
'Parrot'
>>>a = '''Parrot'''
>>>a
'Parrot'
```


注 意

字符串开头与结尾的引号要一致。

下面的的示例将字符串开头使用双引号、结尾使用单引号。

```
>>> a = "Parrot'
Traceback (  File "<interactive input>", line 1
    a = "Parrot'
              ^
SyntaxError: invalid token
```

由此可见，当字符串开头与结尾的引号不一致时，Python 会显示一个 invalid token 的信息。


注 意

当字符串长度超过一行时，必须使用三引号将字符串包含起来，因为单引号与双引号不可以跨行。例如:

```
>>>a="""Content-type: text/html
 <h1>Hello Python</h1>
 <a href="http://www.python.org">Go to Python</a>"""
>>> a
'Content-type: text/html\n<h1>Hello Python</h1>\n<a href="http://www.
python.org">Go to Python</a>'
```

2.5.3 数据类型的相互转换

有时候，用户需要对数据内置的类型进行转换。数据类型的转换，只需要将数据类型作为函数名即可。以下几个内置的函数可以执行数据类型之间的转换，这些函数返回一个新的对象，表示转换的值。

1. 转换为整数类型

语法格式如下：

```
int(x)
```

将 x 转换为一个整数，例如：

```
>>>int(3.6)
3
```

2. 转换为小数类型

语法格式如下：

```
float(x)
```

将 x 转换为一个浮点数，例如：

```
>>>int(10)
10.0
```

3. 转换为字符串类型

语法格式如下：

```
str(x)
```

将 x 转换为一个字符串，例如：

```
>>>str(567)
'567'
```

2.6 Python 结构数据类型

Python 语言中结构数据类型有很多种，常见的就是集合类型、列表类型、元组类型和字典类型。本节先了解这 4 种结构数据类型的基本概念。

2.6.1 集合类型（Sets）

Sets（集合）是一个无序不重复元素的集。它的主要功能是自动清除重复的元素。创建集合时用大括号（{}）来包含其元素。

例如：

```
>>> goods = {'冰箱', '洗衣机','空调', '冷风扇'}
>>> print(goods)            # 输出集合的内容
```

输出结果如图 2-10 所示。

图 2-10　输出集合的内容

如果集合中有重复的元素，就会自动将其删除。

例如：

```
>>> goods = {'冰箱', '洗衣机', '冰箱', '冰箱', '冷风扇'}
>>> print(goods)                    # 删除重复的
```

删除结果如图 2-11 所示。

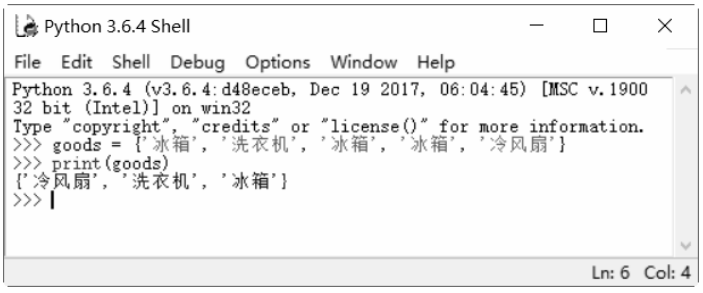

图 2-11　删除重复的元素

如果要创建一个空集合，就必须用 set()函数，例如：

```
>>>goods = set()
```

2.6.2　列表类型（List）

List（列表）是 Python 中使用比较频繁的数据类型。列表可以完成大多数集合类的数据结构实现。列表中元素的类型可以不相同，支持数字、字符串，甚至可以包含列表（所谓嵌套）。列表是写在中括号（[]）之间、用逗号分隔开的元素列表。

要创建一个列表对象，使用中括号（[]）来包含其元素。例如：

```
>>> s = [10,20,30,40]
```

列表对象 s 共有 4 个元素，可以使用 s[0]来返回第 1 个元素、s[1]来返回第 2 个元素，以此类推。

访问列表中元素的方法如图 2-12 所示。

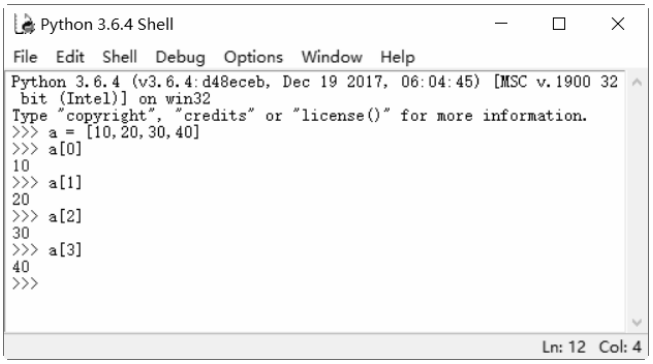

图 2-12　访问列表中的元素

如果索引值超出范围，Python 就会抛出一个 IndexError 异常，如图 2-13 所示。

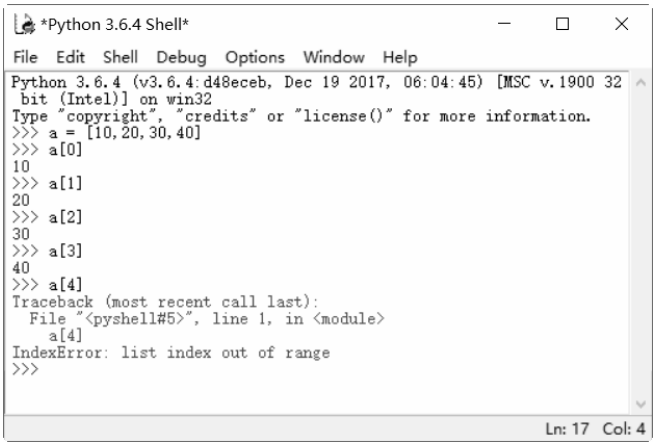

图 2-13　抛出一个 IndexError 异常

Python 为访问最后一个列表元素提供了一种特殊语法。通过将索引指定为-1，可以让 Python 返回一个列表元素。例如：

```
>>>b = [100,200,300,400]
>>>b[-1]
```

执行结果如图 2-14 所示。

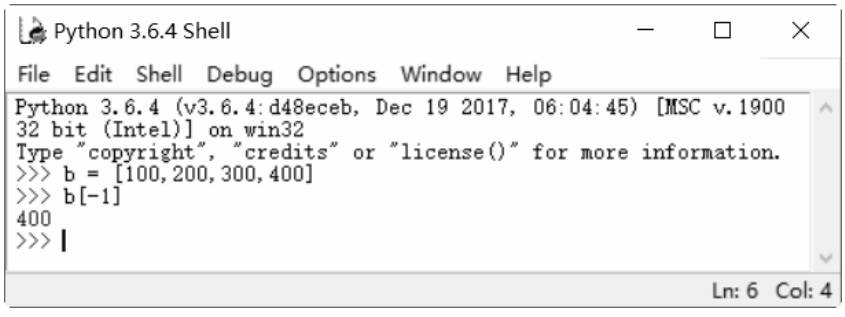

图 2-14　访问列表最后一个元素

在不知道列表长度的情况下，上述方法很实用。以此类推，索引-2 表示倒数第二个列表的元素。

2.6.3 元组类型（Tuple）

Tuple（元组）对象属于序数对象，是一群有序对象的集合，并且可以使用数字来做索引。元组对象与列表对象类似，差别在于元组对象不可以新增、修改与删除。要创建一个元组对象，可以使用小括号()来包含其元素。其语法如下：

```
variable = (element1, element2, ...)
```

下面创建一个元组对象，含有 4 个元素：100、200、300 和 400。

```
>>>c=(100,200,300,400)
>>> c          #查看元组的元素
```

结果如图 2-15 所示。

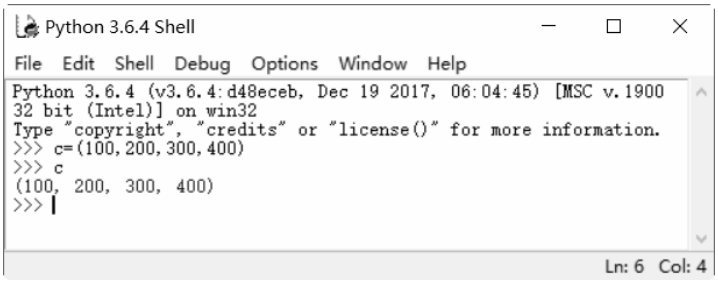

图 2-15　查看元组的元素

也可以省略小括号()，直接将元素列出。

```
>>>c = 100,200,300,400    #省略小括号
>>>c                      #查看元组的元素
```

结果如图 2-16 所示。

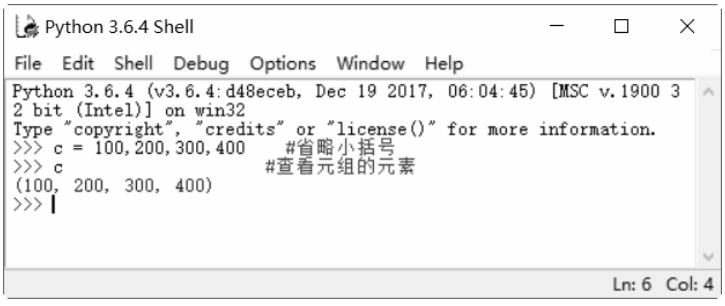

图 2-16　省略小括号

与列表的索引一样，元组索引从 0 开始，例如：

```
>>>t=(100,200,300)
```

```
>>>t[0]
100
```

2.6.4 字典类型（Dictionary）

Dictionary（字典）是 Python 内非常有用的数据类型。字典使用大括号{}将元素列出。元素由键值（key）与数值（value）组成，中间以冒号（:）隔开。键值必须是字符串、数字或元组，这些对象是不可变动的。数值则可以是任何数据类型。字典的元素排列没有一定的顺序，因为可以使用键值来取得该元素。

创建字典的语法格式如下：

```
字典变量={关键字 1:值 1,关键字 2:值 2，……}
```

在同一个字典之内，关键字必须互不相同。

例如，创建字典并访问字典中的元素。

```
bb={'一部': '销售部','二部': '财务部','三部': '市场部'}
bb ['一部']
bb ['二部']
bb ['三部']
```

结果如图 2-17 所示。

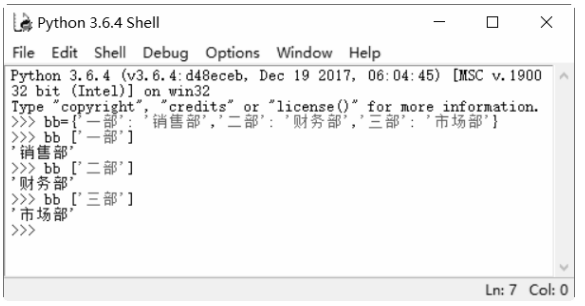

图 2-17 创建字典并访问字典中的元素

2.7 运算符和优先级

在 Python 语言中，支持的运算符包括算术运算符、比较运算符、赋值运算符、逻辑运算符、位运算符、成员运算符和身份运算符。

2.7.1 算术运算符

Python 语言中常见的算术运算符如表 2-1 所示。

2-1 算术运算符

运算符	含义	举例
+	加，两个对象相加	1+2=3
-	减，得到负数或一个数减去另一个数	3-2=1
*	乘，两个数相乘或返回一个被重复若干次的字符串	2*3=6
/	除，返回两个数相除的结果，得到浮点数	4/2=2.0
%	取模，返回除法的余数	21%10=1
**	幂，a**b 表示返回 a 的 b 次幂	$10**21=10^{21}$
//	取整除，返回相除后结果的整数部分	7/3=2

【例 2.6】 使用算术运算符（代码 2.5.py）。

```
x = 10
y = 12
z = 30
#加法运算
a = x + y
print ("a 的值为：", a)
#减法运算
a =x - y
print ("a 的值为：", a)
#乘法运算
a = x * y
print ("a 的值为：", a)
#除法运算
a = x / y
print ("a 的值为：",a)
#取模运算
a= x % y
print ("a 的值为：", a)
#修改变量 x 、y、z
x = 10
y = 12
z = x**y
print ("z 的值为：", z)
#整除运算
x=15
y = 3
z = x//y
print ("z 的值为：", z)
```

保存并运行程序，结果如图 2-18 所示。

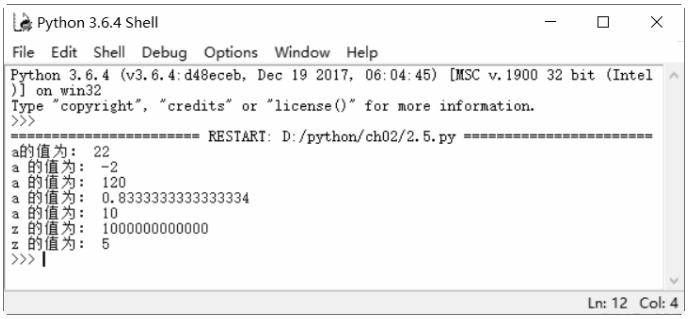

图 2-18 程序运行结果

2.7.2 比较运算符

Python 语言支持的比较运算符如表 2-2 所示。

表2-2 比较运算符

运算符	含义	举例
==	等于，比较对象是否相等	(1==2) 返回 False
!=	不等于，比较两个对象是否不相等	(1!=2) 返回 Ture
>	大于，x>y 返回 x 是否大于 y	2>3 返回 False
<	小于，x<y 返回 x 是否小于 y	2<3 返回 Ture
>=	大于等于，x>=y 返回 x 是否大于等于 y	3>=1 返回 Ture
<=	小于等于，x<=y 返回 x 是否小于等于 y	3<=1 返回 False

【例 2.7】 使用比较运算符（源代码 2.6.py）。

```
a = 16
b = 4
# 判断变量 a 和 b 是否相等
if ( a == b ):
   print ("a 等于 b")
else:
   print ("a 不等于 b")
# 判断变量 a 和 b 是否不相等
if ( a != b ):
   print ("a 不等于 b")
else:
   print ("a 等于 b")
# 判断变量 a 是否小于 b
if ( a < b ):
   print ("a 小于 b")
else:
   print ("a 大于等于 b")
# 判断变量 a 是否大于 b
if ( a > b ):
   print (" a 大于 b")
```

```
else:
   print (" a 小于等于 b")
# 修改变量 a 和 b 的值
a = 15;
b = 32;
# 判断变量 a 是否小于等于 b
if ( a <= b ):
   print (" a 小于等于 b")
else:
   print (" a 大于  b")
# 判断变量 b 是否大于等于 a
if ( b >= a):
   print (" b 大于等于 a")
else:
   print (" b 小于 a")
```

保存并运行程序，结果如图 2-19 所示。

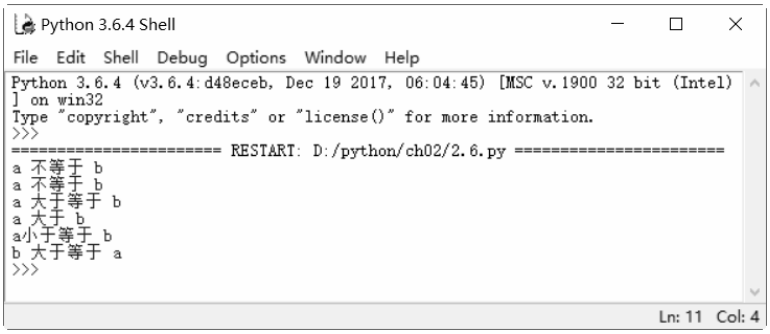

图 2-19　程序运行结果

2.7.3　赋值运算符

赋值运算符表示将右边变量的值赋给左边变量，常见的赋值运算符的含义如表 2-3 所示。

表2-3　赋值运算符

运算符	含义	举例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

【例 2.8】 使用赋值运算符（代码 2.7.py）。

```
a = 24
b = 8
c = 6
#简单的赋值运算
c = a + b
print ("c 的值为：", c)
#加法赋值运算
c += a
print ("c 的值为：", c)
#乘法赋值运算
c *= a
print ("c 的值为：", c)
#除法赋值运算
c /= a
print ("c 的值为：", c)
#取模赋值运算
c = 12
c %= a
print ("c 的值为：", c)
#幂赋值运算
a=3
c **= a
print ("c 的值为：", c)
#取整除赋值运算
c //= a
print ("c 的值为：", c)
```

保存并运行程序，结果如图 2-20 所示。

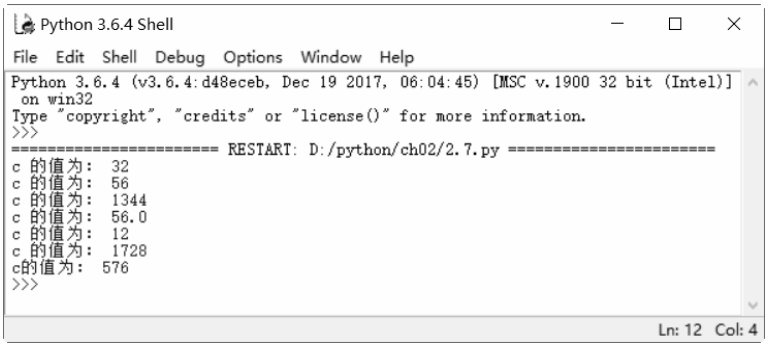

图 2-20 程序运行结果

2.7.4 逻辑运算符

Python 支持的逻辑运算符如表 2-4 所示。

表2-4 逻辑运算符

运算符	含义	举例
and	布尔"与"，x and y 表示如果 x 为 False，那么 x and y 返回 False，否则返回 y 的计算值	(10 and 15)返回 15
or	布尔"或"，x or y 表示如果 x 是 True，就返回 True，否则返回 y 的计算值	(10 or 15)返回 15
not	布尔"非"，not x 表示如果 x 为 True，就返回 False。如果 x 为 False，它返回 True	not (10 and 15) 返回 False

【例 2.9】 使用逻辑运算符（代码 2.8.py）。

```
a = 12
b = 24
#布尔"与"运算
if ( a and b ):
   print ("变量 a 和 b 都为 true")
else:
   print ("变量 a 和 b 有一个不为 true")
#布尔"或"运算
if ( a or b ):
   print ("变量 a 和 b 都为 true，或其中一个变量为 true")
else:
   print ("变量 a 和 b 都不为 true")
# 修改变量 a 的值
a = 0
if ( a and b ):
   print ("变量 a 和 b 都为 true")
else:
   print ("变量 a 和 b 有一个不为 true")
if ( a or b ):
   print ("变量 a 和 b 都为 true，或其中一个变量为 true")
else:
   print ("变量 a 和 b 都不为 true")
# 布尔"非"运算
if not( a and b ):
   print ("变量 a 和 b 都为 false，或其中一个变量为 false")
else:
   print ("变量 a 和 b 都为 true")
```

保存并运行程序，结果如图 2-21 所示。

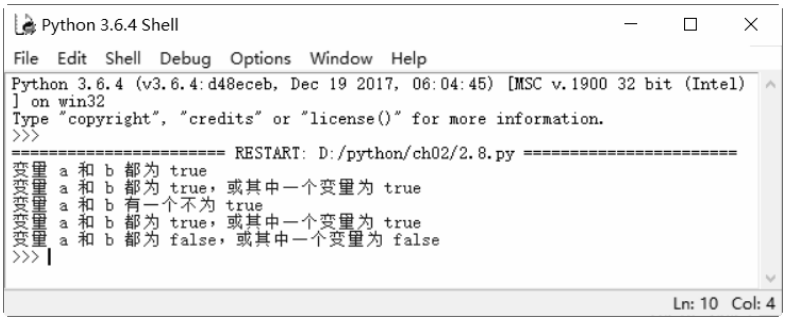

图 2-21 程序运行结果

2.7.5 位运算符

在 Python 中，位运算符把数字看作二进制来进行计算。Python 支持的位运算符如表 2-5 所示。

表2-5 位运算符

运算符	含义	举例
&	按位与，参与运算的两个值，如果两个相应位都为 1，则该位的结果为 1，否则为 0	(12&6)=4，二进制为：0000 0100
|	按位或，只要对应的两个二进位有一个为 1，结果位就为 1	(12|6)=14，二进制为：0000 1110
^	按位异或，当两个对应的二进位相异时，结果为 1，否则为 0	(12^6)=10，二进制为：0000 1010
~	按位取反，对数据的每个二进制位取反，即把 1 变为 0、把 0 变为 1	(~6)=-7，二进制为：1000 0111
<<	左移动，把“<<”左边的运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补 0	(12<<2)=48，二进制为：0011 0000
>>	右移动，把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数	(12>>2)=3，二进制为：0000 0011

【例 2.10】 使用位运算符（代码 2.9.py）。

```
a = 12           # 12 =0000 1100
b = 6           # 6= 0000 0110
c = 0
#按位与运算
c = a & b;        # 4 = 0000 0100
print ("c 的值为：", c)
#按位或运算
c = a | b;        # 14 = 0000 1110
```

```
print ("c 的值为：", c)
#按位异或运算
c = a ^ b;        # 10 = 0000 1010
print ("c 的值为：", c)
#按位取反运算
c = ~a;          # -13 = 1000 1101
print ("c 的值为：", c)
#左移动运算
c = a << 2;       # 48 = 0011 0000
print ("c 的值为：", c)
#右移动运算
c = a >> 2;       # 3 = 0000 0011
print ("c 的值为：", c)
```

保存并运行程序，结果如图 2-22 所示。

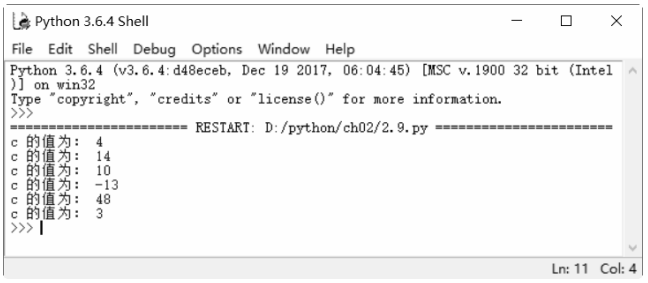

图 2-22　程序运行结果

2.7.6　成员运算符

Python 还支持成员运算符，测试实例中包含了一系列的成员，如字符串、列表、元组。成员运算符包括 in 和 not in，x in y 表示若 x 在 y 序列中则返回 True；x not in y 表示若 x 不在 y 序列中则返回 True。

【例 2.11】 使用成员运算符（代码 2.10.py）。

```
a ='洗衣机'
b = '风扇'
goods = ['电视机', '空调', '洗衣机', '冰箱', '电脑' ];
# 使用 in 成员运算符
if ( a in goods ):
   print ("变量 a 在给定的列表 goods 中")
else:
   print ("变量 a 不在给定的列表 goods 中")
# 使用 not in 成员运算符
if ( b not in goods ):
```

```
    print ("变量 b 不在给定的列表 goods 中")
else:
    print ("变量 b 在给定的列表 goods 中")
# 修改变量 a 的值
a = '冷风扇'
if ( a in goods ):
    print ("变量 a 在给定的列表 goods 中")
else:
    print ("变量 a 不在给定的列表 goods 中")
```

保存并运行程序，结果如图 2-23 所示。

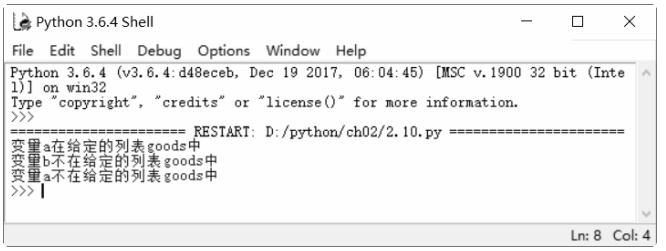

图 2-23　程序运行结果

2.7.7　身份运算符

Python 支持身份运算符为 is 和 not is。其中，is 判断两个标识符是不是引用自一个对象；is not 判断两个标识符是不是引用自不同对象。

【例 2.12】 使用身份运算符（代码 2.11.py）。

```
a ='风扇'
b = '冷风扇'
#使用 is 身份运算符
if ( a is b):
    print ("a 和 b 有相同的标识")
else:
    print ("a 和 b 没有相同的标识")
#使用 not is 身份运算符
if ( a not in b ):
    print ("a 和 b 没有相同的标识")
else:
    print ("a 和 b 有相同的标识")
# 修改变量 a 的值
a = '冷风扇'
if ( a is b):
    print ("修改后的 a 和 b 有相同的标识")
```

```
else:
  print ("修改后的 a 和 b 仍然没有相同的标识")
```

保存并运行程序，结果如图 2-24 所示。

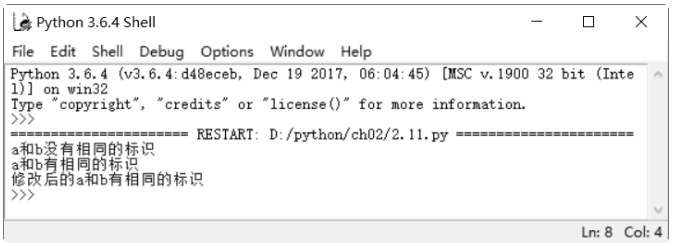

图 2-24　程序运行结果

2.7.8　运算符的优先级

下面是 Python 的运算符，以处理顺序的先后排列。

（1）()、[]、{}。

（2）object。

（3）object[i]、object[1:r]、object.attribute、function()。

“.”符号用来存取对象的属性与方法。下面的示例调用对象 t 的 append()方法，在对象 t 的结尾添加一个字符"t"：

```
>>> t = ["P","a","r","r","o"]
>>> t.append("t")
>>> t
['P', 'a', 'r', 'r', 'o', 't']
```

（4）+x、-x、~x。

（5）x**y：x 的 y 次方。

（6）x * y、x / y、x % y：x 乘以 y、x 除以 y、x 除以 y 的余数。

（7）x + y、x – y：x 加 y、x 减 y。

（8）x << y、x >> y：x 左移 y 位、x 右移 y 位。例如：

```
>>> x = 4
>>> x << 2
16
```

（9）x & y：位 AND 运算符。

（10）x ^ y：位 XOR 运算符。

（11）x | y：位 OR 运算符。

（12）<、<=、>、>=、==、!=、<>、is、is not、in、not in。

in 与 not in 运算符应用在列表（list）上。is 运算符检查两个变量是否属于相同的对象。is

not 运算符则是检查两个变量是否不属于相同的对象。

!=与<>运算符是相同功能的运算符，都用来测试两个变量是否不相等。Python 建议使用!=运算符，而不要使用<>运算符。

（13）not。

（14）and。

（15）or、lambda args:expr。

使用运算符时注意下面的事项：

① 除法应用在整数时，其结果会是一个浮点数。例如，8/4 会等于 2.0，而不是 2。余数运算会将 x / y 所得的余数返回来，如 7%4 =3。

② 如果将两个浮点数相除取余数的话，那么返回值也会是一个浮点数，计算方式是 x – int(x / y) * y。例如：

```
>>>7.0 % 4.0
3.0
```

③ 比较运算符可以连在一起处理，如 a < b < c < d，Python 会将这个式子解释成 a < b and b < c and c < d。像 x < y > z 也是有效的表达式。

④ 如果运算符（operator）两端的运算数（operand），其数据类型不相同，Python 就会将其中一个运算数的数据类型转换为与另一个运算数一样的数据类型。转换顺序为：若有一个运算数是复数，则另一个运算数也会被转换为复数；若有一个运算数是浮点数，则另一个运算数也会被转换为浮点数。

⑤ Python 有一个特殊的运算符——lambda。利用 lambda 运算符能够以表达式的方式创建一个匿名函数。lambda 运算符的语法如下：

```
lambda args : expression
```

args 是以逗号（,）隔开的参数列表 list，而 expression 则是对这些参数进行运算的表达式。例如：

```
>>>a=lambda x,y:x + y
>>>print (a(3,4))
7
```

x 与 y 是 a()函数的参数，a()函数的表达式是 x+y。lambda 运算符后只允许有一个表达式。要达到相同的功能也可以使用函数来定义 a，如下所示：

```
>>> def a(x,y):     #定义一个函数
 return x + y     #返回参数的和
>>> print (a(3,4))
7
```

【例 2.13】 运算符的优先级（源代码 2.12.py）。

```
a = 10
b = 6
c = 4
d = 2
e = 0
e = (a + b) * c / d       #(16 *4 ) / 2
print ("(a + b) * c / d 运算结果为：", e)
e = ((a + b) * c) / d     # (16 *4 ) /2
print ("((a + b) * c) / d 运算结果为：", e)
e = (a + b) * (c / d);    # (16) * (4/2)
print ("(a + b) * (c / d) 运算结果为：", e)
e = a + (b * c) / d;      # 10 + (24/2)
print ("a + (b * c) / d 运算结果为：", e)
```

保存并运行程序，结果如图 2-25 所示。

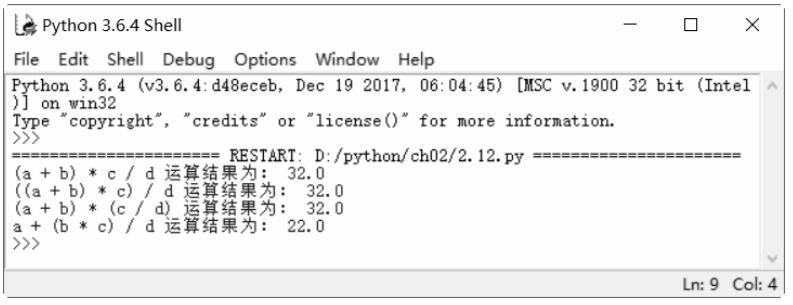

图 2-25　程序运行结果

2.8　疑难解惑

疑问 1：该编写什么样的注释？

编写注释的目的是表述代码的主要功能。对于分开开发系统的合作团队，清晰明了的注释显得尤为重要，这也是一个专业程序员必须掌握的一项技能。作为新手，最好的习惯就是在代码中编写清晰、简洁的注释。

疑问 2：如何查看变量的类型？

内置的 type()函数可以用来查询变量所指的对象类型。

例如：

```
>>>a, b, c, d = 20, 5.5, True, 4+3j
>>> print(type(a), type(b), type(c), type(d))
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

第 3 章　列表、元组和字典的基本操作

数据结构是通过某种方式组织在一起的数据元素的集合，这些元素可以是数字或字符。Python 有许多特殊的数据结构，常用的就是列表、元组和字典。其中，列表与元组属于序数（sequence）类型，它们是数个有序对象的组合；字典则属于映像（mapping）类型，是由一个对象集合来作为另一个对象集合的键值索引。本章将讲述列表、元组和字典的基本操作。

- 熟悉列表对象的特性
- 掌握操作列表的常见方法
- 熟悉列表的内置函数和方法
- 熟悉元组对象的特性
- 掌握元组内置函数的使用方法
- 熟悉字典对象的特性
- 掌握字典的内置函数和方法

3.1　列表的基本操作

列表对象属于序数对象，是一群有序对象的集合，并且可以使用数字来做索引。列表对象可以做新增、修改和删除的操作。

3.1.1　列表对象的特性

列表由一系列按特定顺序排列的元素组成。在 Python 中，用方括号[]来表示列表，用逗号来分割其中的元素。例如：

```
>>>aa=["泠泠七弦上，","静听松风寒。","古调虽自爱，","今人多不弹。"]
>>>print(aa)
```

运行结果如图 3-1 所示。

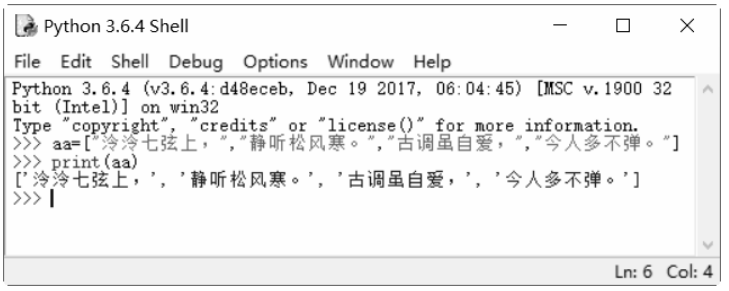

图 3-1　运行结果

从结果可以看出，Python 不仅输出列表的内容，还包括方括号。

列表的常见特性如下：

（1）列表对象中的元素可以是不同的类型，例如：

```
>>>aa=[12,"何当共剪西窗烛",1.66,5+3j]
```

（2）列表对象中的元素可以是另一个列表，例如：

```
>>>aa = [12,"何当共剪西窗烛",1.66,["一夕轻雷落万丝","霁光浮瓦碧参差",3.66]]
```

（3）访问列表中对象的方法比较简单，列表中的序号是从 0 开始的。例如，访问下面列表中的第 4 个元素。

```
>>>aa =[12,"何当共剪西窗烛",1.66,["一夕轻雷落万丝","霁光浮瓦碧参差",3.66]]
>>>aa[3]
```

运行结果如图 3-2 所示。

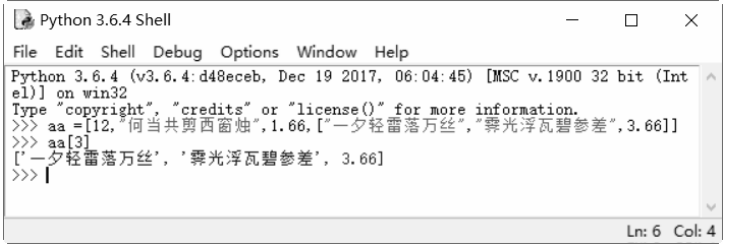

图 3-2　运行结果

（4）列表是可以嵌套的，如果要读取列表对象中嵌套的另一个列表，可使用另一个中括号[]来做索引。例如：

```
>>> aa =[12,"何当共剪西窗烛",1.66,["一夕轻雷落万丝","霁光浮瓦碧参差",3.66]]
>>> aa[3][1]
```

运行结果如图 3-3 所示。

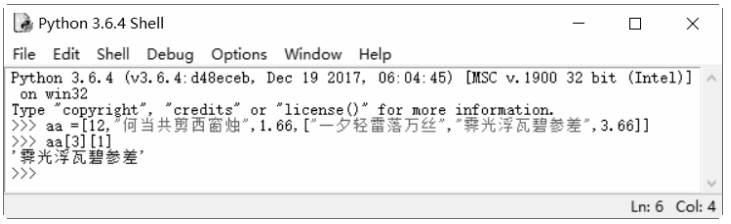

图 3-3 运行结果

in 运算符用于判断一个元素是否存在于列表中，例如：

```
>>>1 in [1, 2, 3]
Ture
```

3.1.2 列表的常见操作

列表创建完成后，还可以对其进行相关的操作。

1. 获取某个元素的返回值

使用列表对象的 index(c)方法（c 是元素的内容）来返回该元素的索引值。例如：

```
>>> aa =[12,"何当共剪西窗烛",1.66,["一夕轻雷落万丝","霁光浮瓦碧参差",3.66]]
>>> aa.index("何当共剪西窗烛")
>>> aa.index(1.66)
```

运行结果如图 3-4 所示。

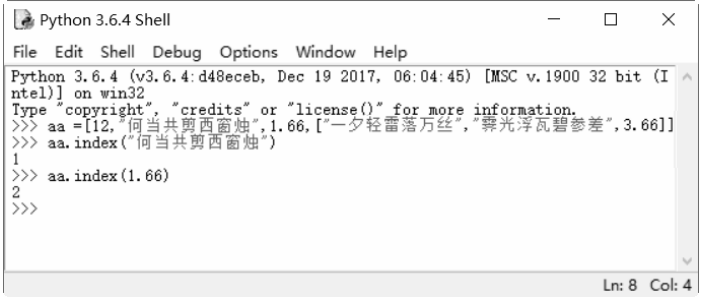

图 3-4 运行结果

2. 改变列表对象的元素值

列表中的元素值是可以改变的。例如，修改列表中的第 2 个元素：

```
>>> aa =[12,"何当共剪西窗烛",1.66,["一夕轻雷落万丝","霁光浮瓦碧参差",3.66]]
>>> aa[1] = "却话巴山夜雨时"
>>> aa
```

运行结果如图 3-5 所示。

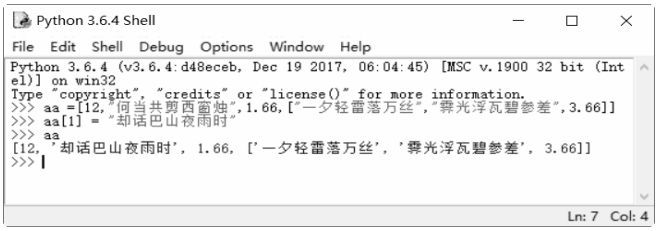

图 3-5　运行结果

3. 在列表中插入新元素

例如，在列表的第 4 个位置插入两个新元素：

```
>>> aa =[12,"何当共剪西窗烛",1.66]
>>> aa[3:] = ["却话巴山夜雨时",1.12]   #3：表示从左侧数第 4 个位置开始添加新元素
>>> aa
```

运行结果如图 3-6 所示。

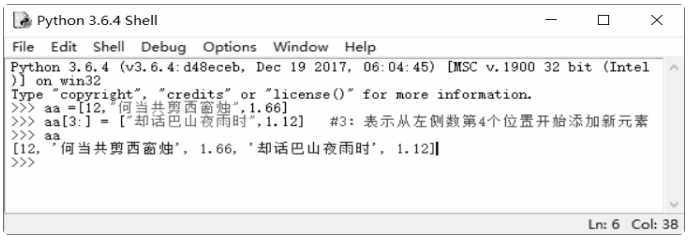

图 3-6　运行结果

4. 删除列表中的元素

使用 del 语句可以删除列表对象中的元素。
例如，删除列表中的第 3 个元素：

```
>>> aa =[12,"何当共剪西窗烛",1.66]
>>> del aa[2]
>>> aa
```

运行结果如图 3-7 所示。

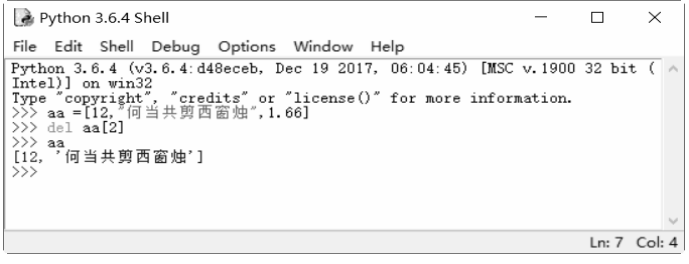

图 3-7　运行结果

如果想从列表中删除最后一个元素，可以使用序号-1。例如：

```
>>>aa= [100, ['A', 'B', 'C']]
>>> del aa[-1]  #-1 表示从右侧数第一个元素
>>> aa
[100]
```

如果想一次清除所有的元素，可以使用 del 语句操作，命令如下：

```
del a[:]
```

3.1.3 列表的操作符+和*

列表的常用操作符包括+和*。其中，列表对+和*的操作与字符串相似。+号用于组合列表，*号用于重复列表。

+号操作符经常用于字符串和列表元素的组合。例如：

```
>>>aa=[1,2,3]+ [4,5,6] + [7,8,9]
>>>aa
```

运行结果如图 3-8 所示。

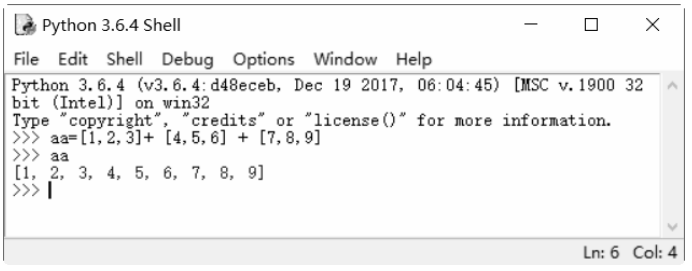

图 3-8 运行结果

```
>>>aa=["苹果","葡萄","柚子","桃子","橙子"]
>>>aa="我最喜欢的水果是"+aa[1]
>>> print(aa)
```

运行结果如图 3-9 所示。

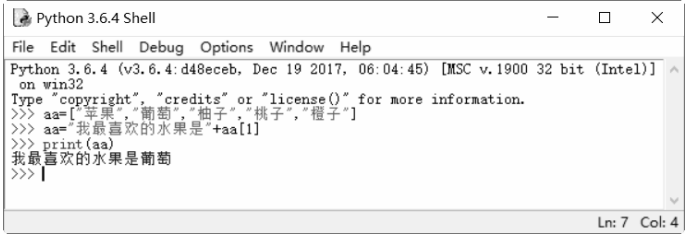

图 3-9 运行结果

*号运算符经常用于重复列表中的元素。例如，将列表中的元素重复 4 次：

```
>>>aa=[100,200,300,400]*4
>>>aa
```

运行结果如图 3-10 所示。

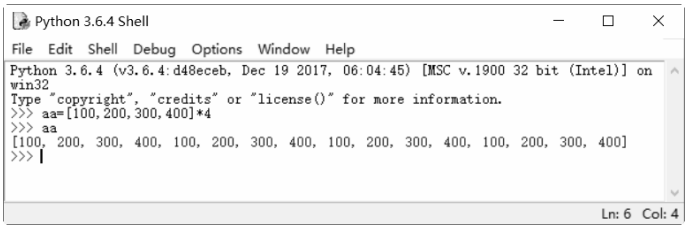

图 3-10　运行结果

如何才能创建一个占有 10 个元素空间而又不包括任何内容的列表呢？

空列表可以简单地通过中括号（[]）来表示，如果想创建 10 个元素空间而又不包括内容的列表，可以使用*号来实现，如[]*10，这样就生成了一个包含 10 个空元素的列表。然而，有时候可能需要一个值来代表空值，表示没有放置任何元素，可以使用 None。None 是 Python 的内建值，例如：

```
>>>aa=[None]*10
>>>aa
[None, None, None, None, None, None, None, None, None, None]
```

3.1.4　内置的函数和方法

列表对象有许多的内置函数和方法，下面学习这些函数和方法的使用技巧。

1. 列表的函数

列表内置的函数包括 len()、max()、min()和 list()。

（1）len()函数返回列表的长度。例如：

```
>>>aa=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>len(aa)
```

运行结果如图 3-11 所示。

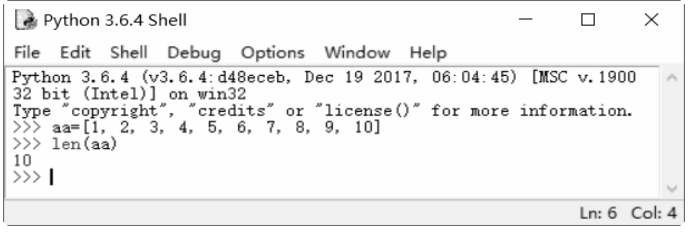

图 3-11　运行结果

（2）max()函数返回列表元素中的最大值。例如，求取列表中的最大值：

```
>>>aa=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>max(aa)
>>>bb=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'o', 'p']
>>>max(bb)
```

运行结果如图 3-12 所示。

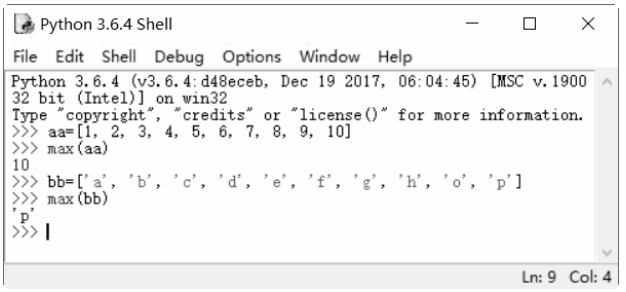

图 3-12　运行结果

注意　列表中的元素数据类型必须一致才能使用 max()函数，否则会出错。例如：

```
>>>aa=[100, 200, 300, 400, 'qq']
>>>max(aa)
```

报错信息如图 3-13 所示。

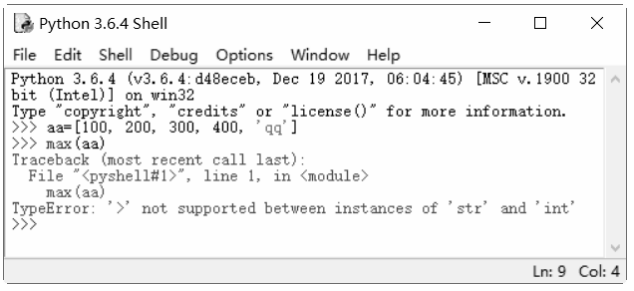

图 3-13　报错信息

（3）min()函数返回列表元素中的最小值。例如：

```
>>>aa=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>min(aa)
>>>bb=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'o', 'p']
>>>min(bb)
```

运行结果如图 3-14 所示。

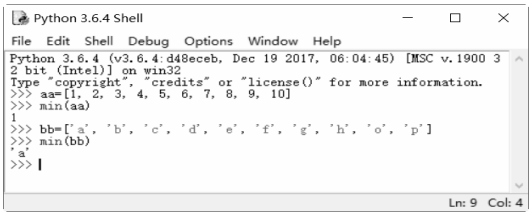

图 3-14　运行结果

2. 列表的方法

在 Python 解释器内输入 dir([])，就可以显示这些内置的列表方法。

```
>>>dir([])
```

运行结果如图 3-15 所示。

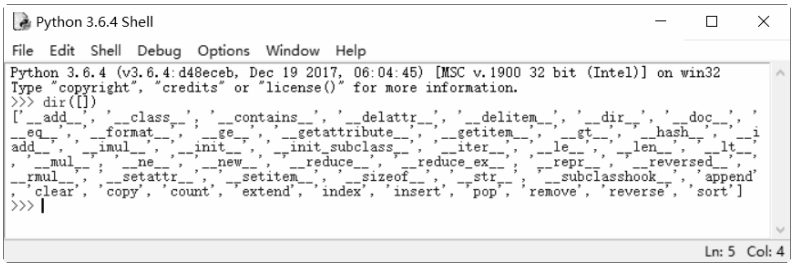

图 3-15　运行结果

下面将挑选常用的方法进行介绍。

（1）append(object)

append()方法在列表对象的结尾，加上新对象 object。例如：

```
>>>aa= [100,200,300,400]
>>>aa.append(500)
>>>aa
>>>aa.append([600,700])
>>>aa
```

运行结果如图 3-16 所示。

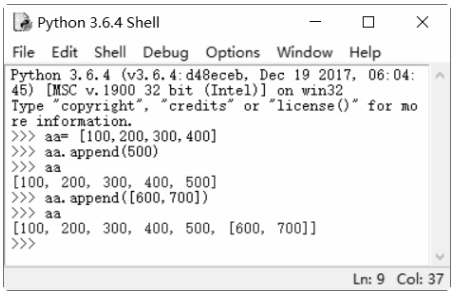

图 3-16　运行结果

（2）clear()

clear()函数用于清空列表，类似于 del a[:]。例如：

```
>>>aa =[100,200,300,400,500,600]
>>>aa.clear()   #清空列表
>>>aa
```

运行结果如图 3-17 所示。

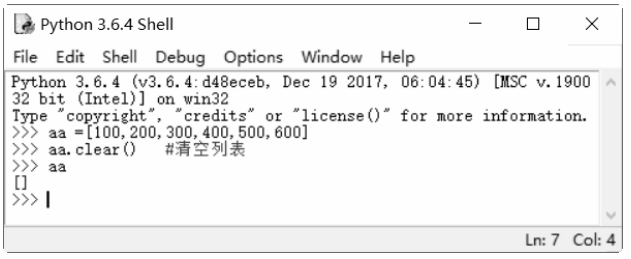

图 3-17　运行结果

（3）copy()

copy()函数用于复制列表。例如：

```
>>>aa = ['苹果', '香蕉', '柚子', 1.88, 2.66, 3.86]
>>>bb = aa.copy()
>>>bb
```

运行结果如图 3-18 所示。

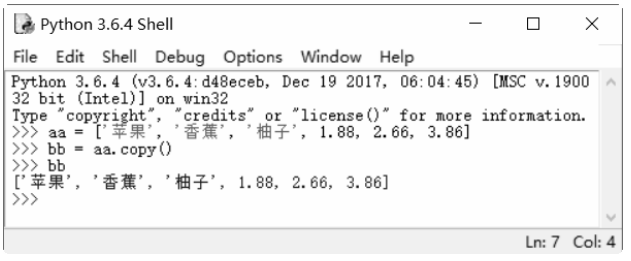

图 3-18　运行结果

（4）count(value)

count(value)方法针对列表对象中的相同元素值 value 计算其数目。例如，计算出列表值为 166 的元素个数：

```
>>>aa= [100,133,166,188,166,266]
>>>aa.count(166)
```

运行结果如图 3-19 所示。

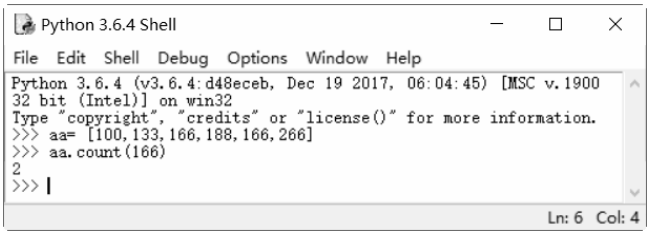

图 3-19　运行结果

（5）extend(list)

extend(list)方法将参数 list 列表对象中的元素加到此列表中，成为此列表的新元素。例如：

```
>>>aa= [100, 200, 300, 400]
>>>aa.extend([500,600,700,800])
>>>aa
```

运行结果如图 3-20 所示。

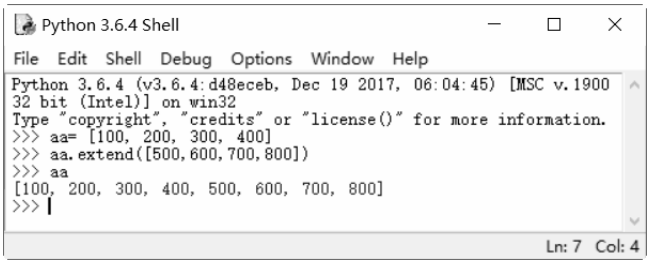

图 3-20　运行结果

（6）index(value)

index(value)方法将列表对象中元素值为 value 的索引值返回。例如：

```
>>>aa= [100, 200, 300, 400, 500, 600, 700, 800]
>>>aa.index(800)
```

运行结果如图 3-21 所示。

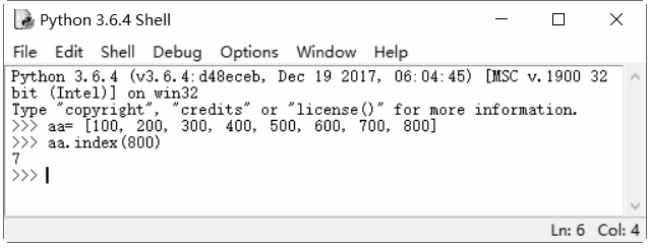

图 3-21　运行结果

（7）insert(index, object)

insert(index, object)方法将在列表对象中索引值为index的元素之前插入新元素object。例如：

```
>>>aa=[100, 200, 300, 400, 500, 600, 700, 800]
>>>aa.insert(1,"新元素")
>>>aa
```

运行结果如图 3-22 所示。

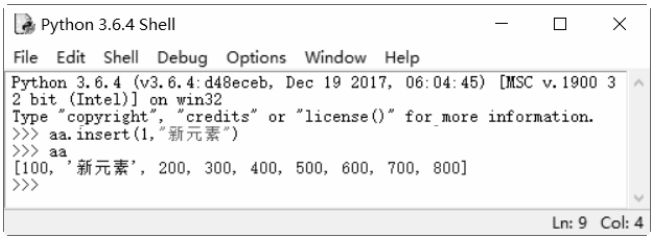

图 3-22　运行结果

（8）pop([index])

pop([index])方法将列表对象中索引值为 index 的元素删除。如果没有指定 index 的值，就将最后一个元素删除。例如，删除第 2 个元素和删除最后一个元素：

```
>>>aa= [100, 200, 300, [400, 500, 600]]
>>>aa.pop(1)
>>>aa
>>>aa.pop()
>>>aa
```

运行结果如图 3-23 所示。

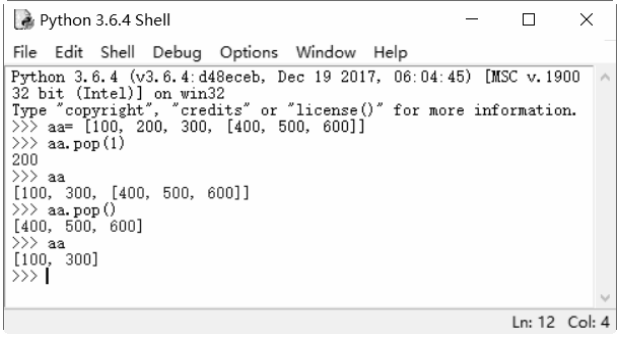

图 3-23　运行结果

（9）remove(value)

remove(value)方法将列表对象中元素值为 value 的删除。例如，删除值为 300 的元素：

```
>>>aa= [100, 200, 300, 400]
>>>aa.remove(300)
>>>aa
```

运行结果如图 3-24 所示。

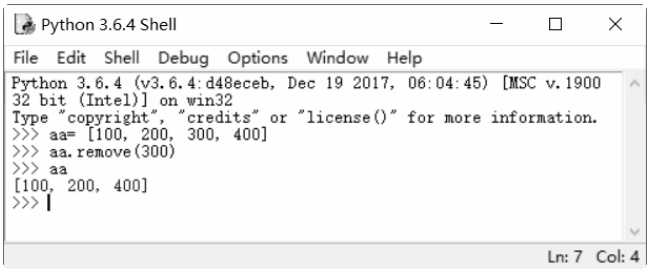

图 3-24　运行结果

（10）reverse()

reverse()方法将列表对象中的元素颠倒排列。例如：

```
>>>aa=[100, 200, 300, 400]
>>>aa.reverse()
>>>aa
```

运行结果如图 3-25 所示。

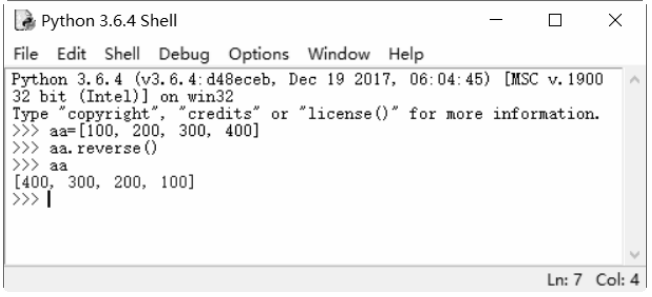

图 3-25　运行结果

（11）sort()

sort()方法将列表对象中的元素依照大小顺序排列。例如：

```
>>>aa = [100, 600, 800,400,500,200,300,700]
>>>aa.sort()
>>>aa
```

运行结果如图 3-26 所示。

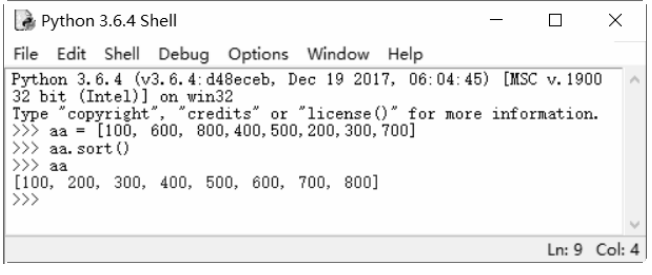

图 3-26　运行结果

3.1.5 递推式构造列表

从 Python 2.0 开始，可以使用递推式构造列表（list comprehension）的功能。所谓递推式构造列表，是使用列表内的元素创建新的列表。

递推式构造列表的语法如下所示：

```
[ expression for expression1 in sequence1
        [for expression2 in sequence2]
        [... for expressionN in sequenceN]
        [if condition] ]
```

sequence 代表序数对象，如字符串、元组、列表等。在列表包容的结果中，新列表的元素数目是所有序数对象的元素数目相乘的结果。

下面的示例将字符串对象 aa 与列表对象 laa 做列表包容，创建一个新的列表对象。

```
>>>aa= "ab"
>>>laa= [100,200,300,400]
>>>bb=[ (x,y) for x in aa for y in laa]
>>>bb
```

运行结果如图 3-27 所示。

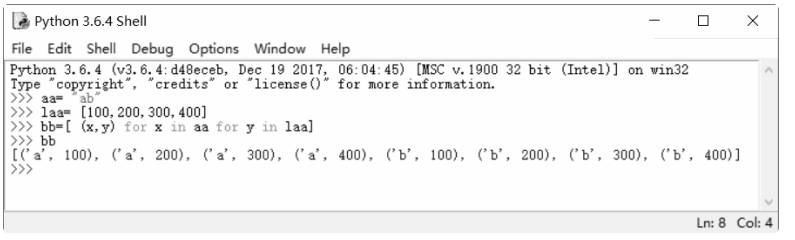

图 3-27 运行结果

aa 字符串对象有两个元素，laa 列表对象有 4 个元素，列表包容产生的新列表有 8 个元素。

毕达哥拉斯三元数组是数形结合的一个典型例子。毕达哥拉斯学派研究出了一个公式：若 m 是奇数，则 m、(m^2-1)/2 及(m^2+1)/2 便是三元数组，分别表示一个直角三角形的两条直角边和斜边。

下面的递推式构造列表创建了毕达哥拉斯三元组：

```
>>>aa=[(x,y,z) for x in range(1,30) for y in range(x,30) for z in range(y,30)
if x**2 + y**2 == z**2]
>>>aa
```

运行结果如图 3-28 所示。

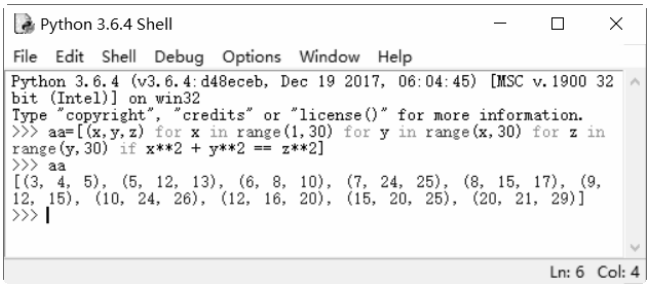

图 3-28　运行结果

3.2　元组的基本操作

与列表相比，元组对象不能修改，同时元组使用小括号、列表使用方括号。元组创建很简单，只需要在括号中添加元素并使用逗号隔开即可。

3.2.1　元组对象的常用操作

上一章已经讲过创建元组的方法，这里继续学习元组的常用操作方法。

1. 创建只有一个元素的元组

如果创建的元组对象只有一个元素，就必须在元素之后加上逗号（,），否则 Python 会认为此元素是要设置给变量的值。

```
>>>a = (100,)
>>>a
>>>a = (100)
>>>a
```

运行结果如图 3-29 所示。

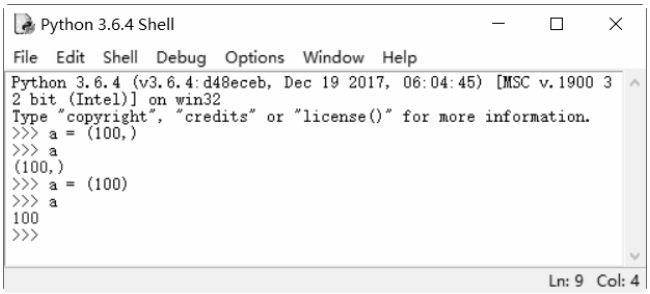

图 3-29　运行结果

2. 元组的对象值不能修改

在元组中，不可以修改元组对象内的元素值，否则会提示错误。

```
>>>aa = (100, 200,300,400)
#以下修改元组元素操作是非法的。
>>>aa[1] = 500
```

运行结果如图 3-30 所示。

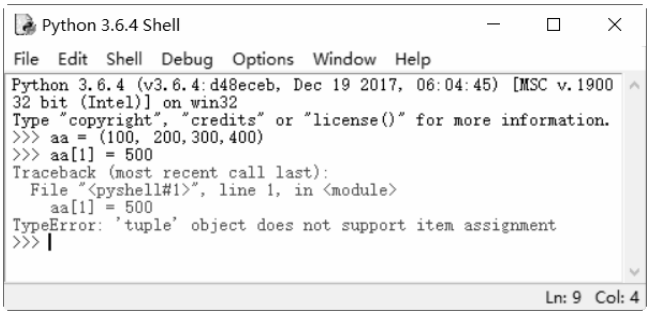

图 3-30　运行结果

3. 删除元组内的对象

虽然元组内的元素值不能修改，但是可以删除，从而达到更新元组对象的效果。例如，在下面的示例中删除元组中的 a[1]：

```
>>>a = (100,200,300,400)
>>>a = a[0],a[2],a[3]
>>>a
```

运行结果如图 3-31 所示。

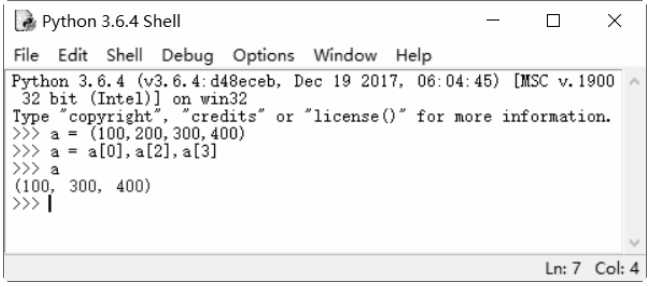

图 3-31　运行结果

4. 获取元组对象的元素值

元组对象支持使用索引值的方式来返回元素值。

```
>>>a = (100,200,300,400)
>>>a[0]
>>>a[1]
>>>a[2]
>>>a[3]
```

运行结果如图 3-32 所示。

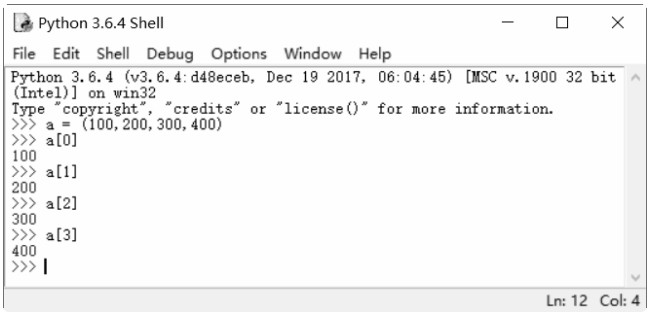

图 3-32　运行结果

5. 组合元组

虽然元组的元素值不能修改，但是可以组合。例如，组合元组 aa 和元组 bb 为新元组 cc：

```
>>>aa = (100,200)
>>>bb = ('河汉清且浅', '相去复几许')
# 组合成一个新的元组 cc
>>>cc = aa + bb
>>>cc
```

运行结果如图 3-33 所示。

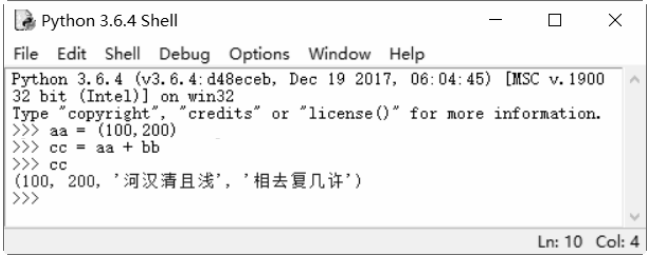

图 3-33　运行结果

6. 删除整个元组

使用 del 语句可以删除整个元组。例如：

```
>>>aa = (100,200, 300,400)  #定义新元组 aa
>>>aa                       #输出元组 aa
>>>del aa                    #删除元组 aa
>>>aa                       #再次输出元组 aa 时将报错
```

运行结果如图 3-34 所示。

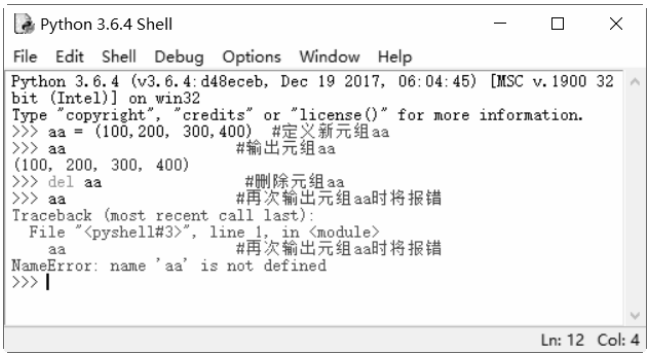

图 3-34 运行结果

从报错信息可以看出，元组已经被删除，再次访问该元组时会提示错误信息。

3.2.2 元组的内置函数

元组的内置函数包括 len()、max()、min()和 tuple()。下面将分别讲述这几个内置函数的使用方法。

1. len()函数

len()函数返回元组的长度。例如：

```
>>>la = (100,200,300,400,500,600,700,800,900)
>>>len(la)
```

运行结果如图 3-35 所示。

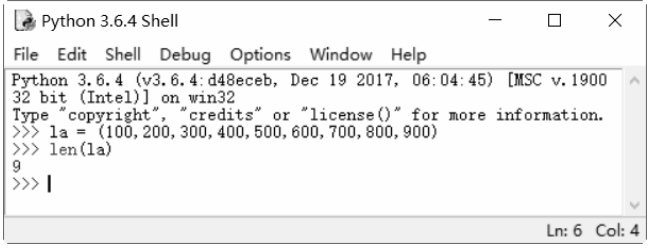

图 3-35 运行结果

2. max()函数

max()函数返回元组或列表元素中的最大值。例如：

```
>>>aa=(100,200,300,400,500,600,700,800,900)
>>>max(aa)
>>>bb=['a', 'c', 'd', 'e', 'f', 'g', 'h', 'o', 'p']
>>>max(bb)
```

运行结果如图 3-36 所示。

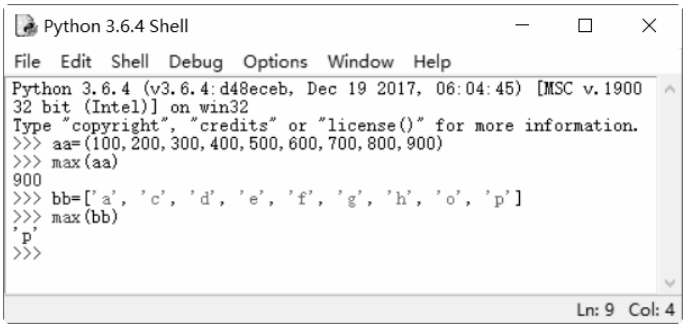

图 3-36　运行结果

注　意　元组中的元素数据类型必须一致才能使用 max()函数，否则会出错。

3. min()函数

min()函数返回元组或列表元素中的最小值。例如：

```
>>>aa=(100,200,300,400,500,600,700,800,900)
>>>min(aa)
>>>bb=['a', 'c', 'd', 'e', 'f', 'g', 'h', 'o', 'p']
>>>min(bb)
```

运行结果如图 3-37 所示。

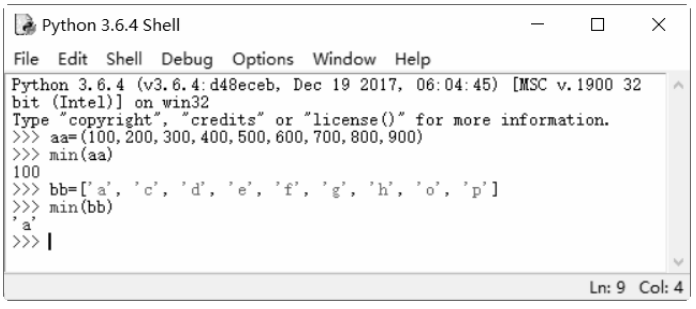

图 3-37　运行结果

注　意　元组中的元素数据类型必须一致才能使用 min()函数，否则会出错。

4. sum()函数

sum()函数返回元组中所有元素的和。

```
>>>aa=(100,200,300,400,500,600,700,800,900)
>>>sum(aa)
```

运行结果如图 3-38 所示。

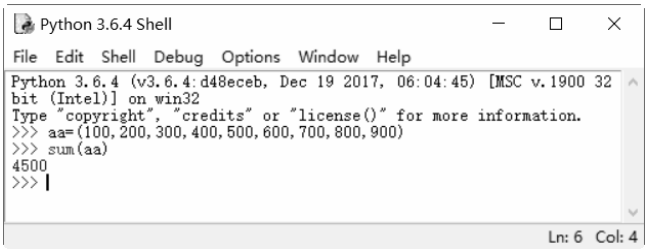

图 3-38　运行结果

3.3　字典的基本操作

与列表和元组有所不同，字典是另一种可变容器模型，且可存储任意类型的对象。本节将学习字典的基本操作。

3.3.1　字典对象的常用操作

字典的对象使用大括号{}将元素列出。字典的元素排列并没有一定的顺序，因为可以使用键值来取得该元素。

下面的示例将创建一个字典对象：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd
```

运行结果如图 3-39 所示。

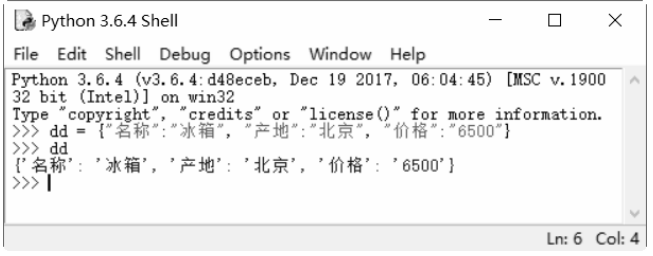

图 3-39　运行结果

1. 获取字典中的元素值

通过使用键值作为索引，可以返回字典中的元素。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd["名称"]
>>>dd["产地"]
>>>dd["价格"]
```

运行结果如图 3-40 所示。

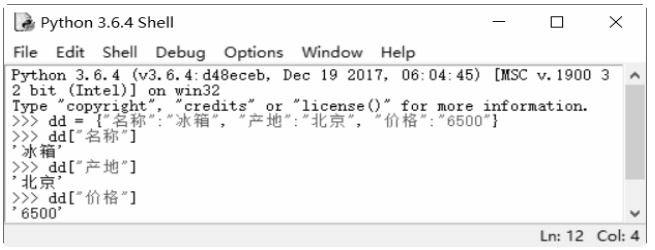

图 3-40　运行结果

在获取字典中的元素值时，必须保证输入的键值在字典中是存在的，否则 Python 会产生一个 KeyError 错误。

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd["姓名"]
```

运行结果如图 3-41 所示。

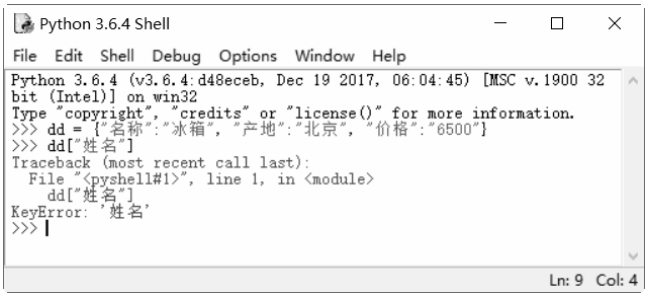

图 3-41　运行结果

从报错信息可以看出，这里不存在"姓名"的键值。

2. 修改字典中的元素值

字典中的元素值是可以修改的。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd["名称"] = "洗衣机"
>>>dd
```

运行结果如图 3-42 所示。

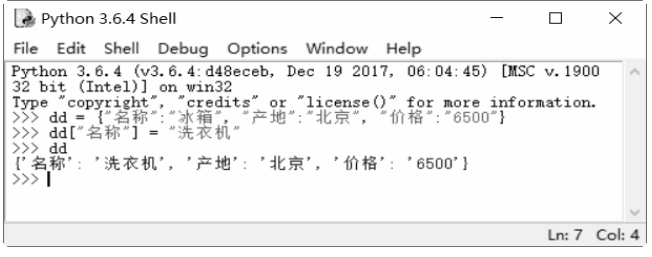

图 3-42　运行结果

3. 删除字典中的元素

使用 del 语句可以删除字典中的元素。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>del dd["名称"]
>>>dd
```

运行结果如图 3-43 所示。

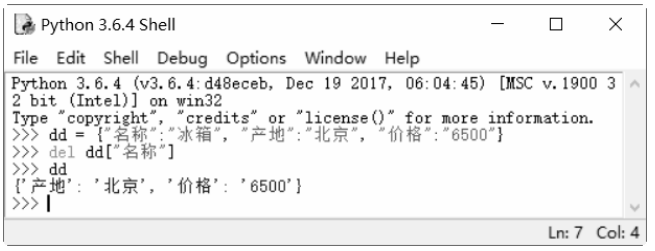

图 3-43 运行结果

4. 定义字典键值时需要注意的问题

字典键值是不能随便定义的，需要注意以下两点：

（1）不允许同一个键值多次出现。创建时如果同一个键值被赋值多次，那么只有最后一个值有效，前面重复的值将会被自动删除。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500","产地":"上海", "价格":"8500"}
>>>dd
```

运行结果如图 3-44 所示。

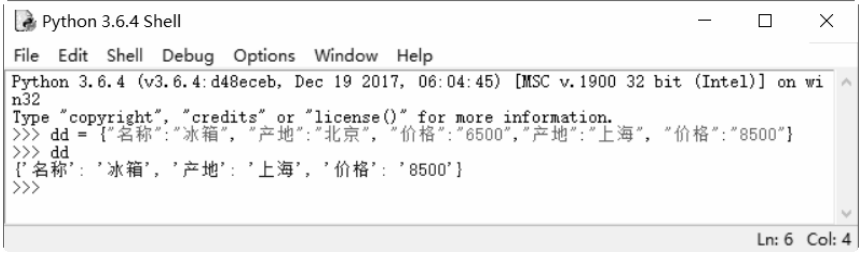

图 3-44 运行结果

（2）因为字典键值必须不可变，所以可以用数字、字符串或元组充当，列表则不行。如果用列表做键值，将会报错。例如：

```
>>>dd = {["名称"]:"冰箱", "产地":"北京", "价格":"6500"}
```

运行结果如图 3-45 所示。

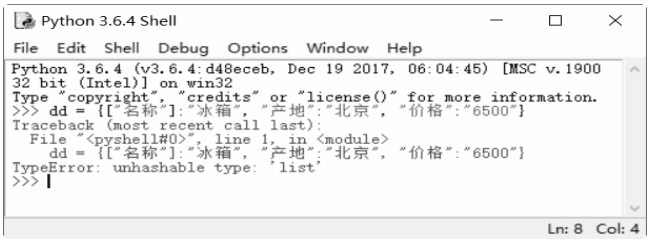

图 3-45　运行结果

3.3.2　字典的内置函数和方法

本节主要讲述字典的内置函数和方法。

1. 字典的内置函数

字典的内置函数包括 len()、str()和 type()。

（1）len(dict)：计算字典元素个数，即键值的总数。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>len(dd)
```

运行结果如图 3-46 所示。

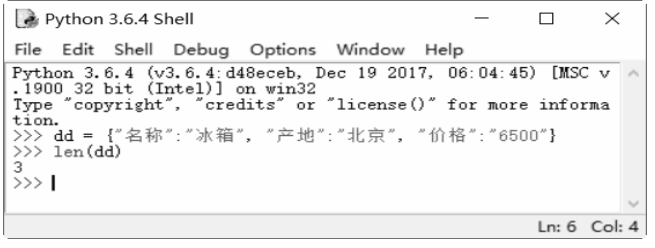

图 3-46　运行结果

（2）str(dict)：将字典的元素转化为可打印的字符串形式。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>str(dd)
```

运行结果如图 3-47 所示。

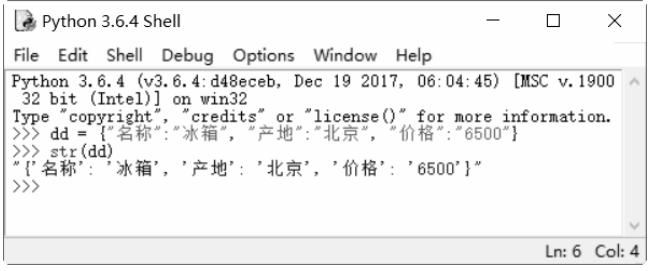

图 3-47　运行结果

（3）type(variable)：返回输入的变量类型，如果变量是字典，就返回字典类型。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>type(dd)
```

运行结果如图 3-48 所示。

图 3-48　运行结果

2. 字典的内置方法

字典对象有许多内置方法，在 Python 解释器内输入 dir({})，就可以显示这些内置方法的名称，结果如图 3-49 所示。

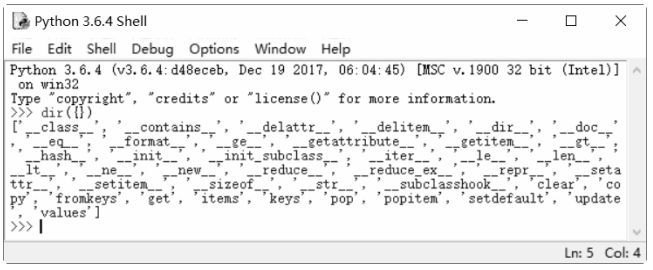

图 3-49　字典的内置方法

下面挑选常用的方法进行讲解。

（1）clear()：清除字典中的所有元素。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.clear()
>>>dd
```

运行结果如图 3-50 所示。

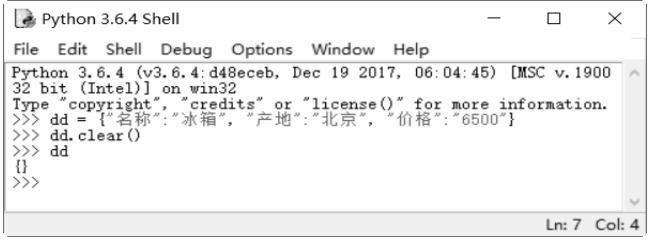

图 3-50　运行结果

（2）copy()：复制字典。例如：

```
>>>cc = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd = cc.copy()
>>>dd
```

运行结果如图 3-51 所示。

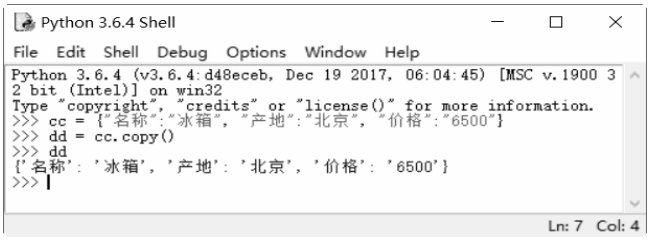

图 3-51　运行结果

（3）get(k [, d])：k 是字典的索引值，d 是索引值的默认值。如果 k 存在，就返回其值，否则返回 d。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.get("名称")
>>>dd.get("品牌","不存在")
```

运行结果如图 3-52 所示。

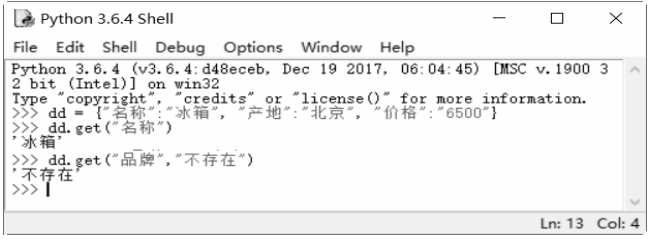

图 3-52　运行结果

（4）items()：使用字典中的元素创建一个由元组对象组成的列表。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.items()
```

运行结果如图 3-53 所示。

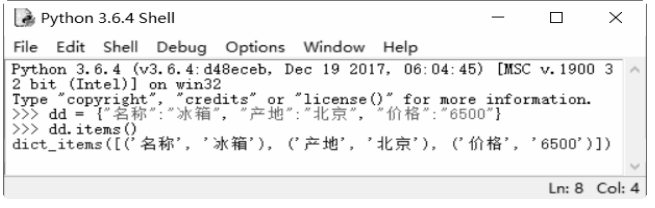

图 3-53　运行结果

（5）keys()：使用字典中的键值创建一个列表对象。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.keys()
```

运行结果如图 3-54 所示。

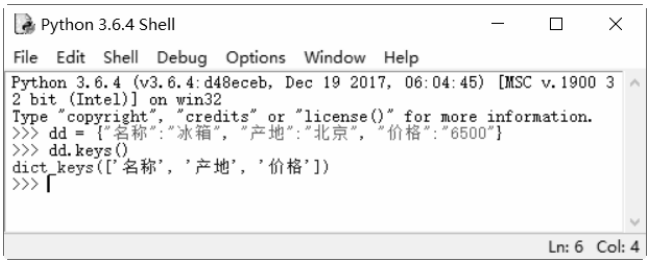

图 3-54　运行结果

（6）popitem()：删除字典中的最后一个元素。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.popitem()
>>>dd
>>>dd.popitem()
>>>dd
```

运行结果如图 3-55 所示。

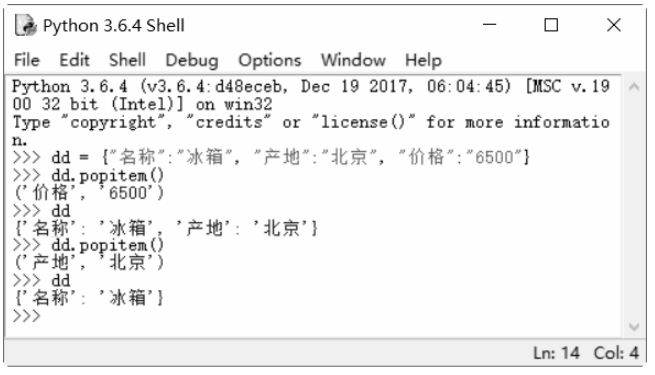

图 3-55　运行结果

（7）setdefault(k [, d])：k 是字典的键值，d 是键值的默认值。如果 k 存在，就返回其值；否则返回 d，并将新的元素添加到字典中。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.setdefault("名称")
>>>dd
>>>dd.setdefault("品牌","海尔")
>>>dd
```

运行结果如图 3-56 所示。

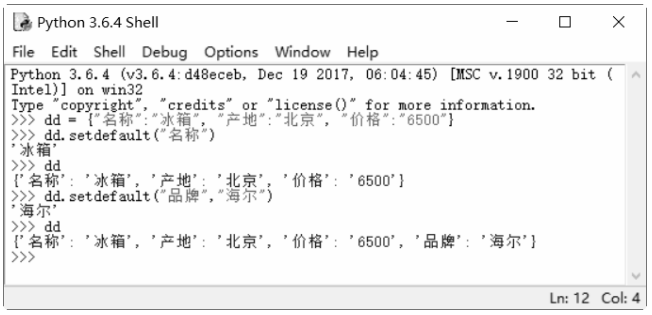

图 3-56　运行结果

（8）update(E)：E 是字典对象，由字典对象 E 来更新此字典。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.update({"品牌":"海尔"})
>>>dd
```

运行结果如图 3-57 所示。

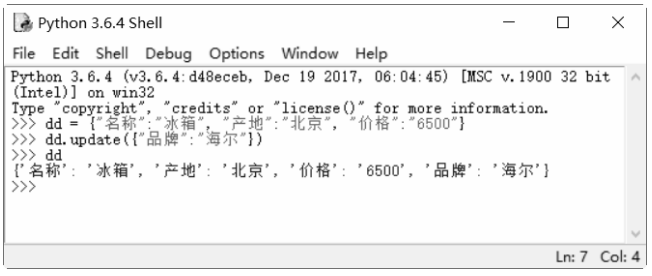

图 3-57　运行结果

（9）values()：使用字典中键值的数值创建一个列表对象。例如：

```
>>>dd = {"名称":"冰箱", "产地":"北京", "价格":"6500"}
>>>dd.values()
```

运行结果如图 3-58 所示。

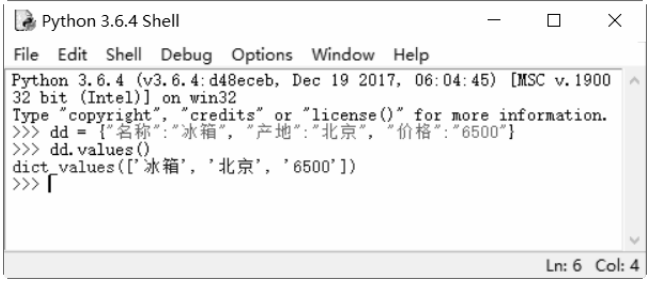

图 3-58　运行结果

3.4 疑难解惑

疑问 1：如何将元组转换为列表？

list()函数用于将元组转换为列表。元组与列表是非常类似的，区别在于元组的元素值不能修改，元组是放在括号中的，列表是放在方括号中的。例如：

```
>>>goods = (1200, '冰箱', '北京', 'S1002')
>>>lgoods = list(goods)
>>>print ("商品的信息为: ",lgoods)
```

运行结果如图 3-59 所示。

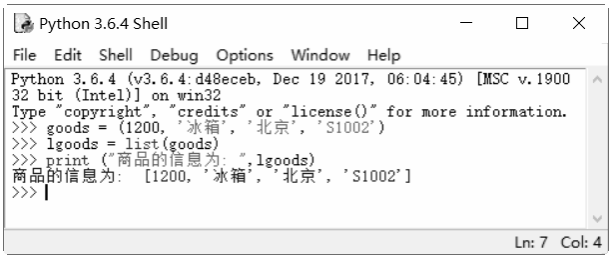

图 3-59 运行结果

疑问 2：如何将列表转换为元组？

tuple()函数用于将列表转换为元组。例如：

```
>>>goods =[1200, '冰箱', '北京', 'S1002']
>>>t1 = tuple(goods)
>>>print ("商品的信息为: ", t1)
```

运行结果如图 3-60 所示。

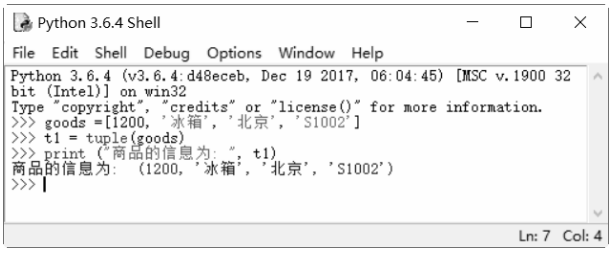

图 3-60 运行结果