第3章 数据类型和表达式

3.1 标识符、变量及其赋值

3.1.1 标识符

标识符(identifier)是由程序员定义的符号,用来给程序中的数据、函数和其他用户自定义对象命名。不同的高级语言具有不同的标识符格式。Python 3.7 的标识符允许采用大写字母、小写字母、数字、下画线(_)和汉字等字符,但标识符的首字符不能是数字,中间不能出现空格。Python 语言在语法上没有对标识符的长度做限制,也就是说 Python 3.7 里的标识符理论上可以任意长。但由于受到计算机存储资源的限制,Python 3.7 的标识符长度一般还是有限的。根据上述规则,以下这些标识符都是合法的: Python_3_7、python_你好、_python_ABC。虽然 Python 3.7 允许使用汉字标识符,但我们不建议这么做(主要原因是在输入时切换中英文会很麻烦,而且程序不易移植)。Python 3.7 的标识符对大小写敏感,如 Python_3、python_3、PYTHON_3、PyThOn_3 和 pYtHoN_3 是 5 个不同的标识符。Python3.7 的标识符命名规则详情可以参考《Python 语言参考手册》(https://docs.python.org/3/reference/lexical_analysis.html)。

保留字(Keyword),也叫关键字,指被编程语言内部定义并保留使用的标识符。程序员编写程序时不能定义与保留字相同的标识符。Python 3.7 的保留字如表 3-1 所示。

表 3-1 Python 3.7 的保留字

false	none	true	and	as	assert	async
await	break	class	continue	def	del	elif
else	except	finally	for	from	global	if
import	in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with	yield

Python 3.7 还包括相当多的内置函数,比如 print、input 等,一般情况下我们在命名标识符时,都会避免使用这些内置函数的名称,以免产生混淆。

下画线对解释器有特殊的意义,是内建标识符所使用的符号,如 Python 3.7 用下画线作为变量前缀和后缀指定特殊变量,像__name__、__doc__等。因此不建议使用下画线作为

标识符的前缀或后缀，除非是有特定的用途。

3.1.2 变量及其赋值

Python是面向对象的编程语言，在Python中一切都是对象(有关对象的详细内容请参考第8章)。对象是Python语言中最基本的概念，是Python对数据的抽象。每个对象都有一个标识(identity)、一种类型(type)及一个值(value)。

一个对象一旦被创建，它的标识就确定了，在它的生命周期之内是不会被改变的，实际上标识就对应了这个对象在内存当中的唯一地址。函数id()返回一个整数，这个整数代表对象的标识。在Python中，函数id()的返回值就是存储该对象的内存地址。

对象的类型决定了该对象可以保存什么类型的值，可以进行什么样的操作，以及遵循什么样的规则。一般情况下，对象的标识和类型都是只读的(在有些情况下对象的类型是可以被改变的，但不建议这么做，尤其是初学者)。

值是对象所表示的数据内容。大部分对象的值是可以被改变的，这类对象称作可变对象。与之相对，值不可以被改变的对象称作不可变对象。对象的值是否可以被改变由对象的类型决定，具体内容会在后续章节陆续介绍。

变量是在编程过程中使用得最多的，用于保存输入、完成中间的计算以及保存最终的运算结果。Python中的变量不需要事先声明(这一点和C语言很不一样)，变量的赋值操作即是变量的声明和定义的过程。例如：

```
x = 5
```

Python中的符号“=”是主要的赋值操作符(增量赋值操作符是另外一种赋值操作符，后边会介绍)。赋值语句的执行过程是：首先把等号右侧表达式的值计算出来，然后在内存中寻找一个位置把值存放进去(可以理解为先创建对象)，最后创建变量并指向这个内存地址(或者说指向对象)。Python中的变量类似Java中的引用式变量，因此变量并不直接存储对象，而是存储了对象的内存地址或者引用。给变量赋值，与在对象上加一个新的标签类似，这也是Python中变量类型可以轻易改变的原因。例如：

```
>>> a = 123
>>> print("The type of a is", type(a))
The type of a is <class 'int'>        # 变量a的类型是整数类型
>>> a = "123"
>>> print("Now the type is", type(a))
Now the type is <class 'str'>         # 变量a的类型变成了字符串类型
```

第二次对变量a的赋值，是让a指向了新的内存地址，即字符串对象“123”的地址。当然，我们并不鼓励随意修改变量的类型，这样容易使得程序变得混乱。

Python中赋值语句并不返回值(与C语言不同)，因此下边这条语句是不合法的：

```
x = (y = y + 1)
```

但Python支持多重赋值，比如：

```
>>> x = y = z = 5
>>> x
```

```
5
>>> y
5
>>> z
5
```

Python 支持多个变量同时赋值，也称为元组赋值，因为等号两边都是元组（元组是 Python 的组合数据类型，第 5 章中有介绍）。比如：

```
>>> x , y , z = 5 , 10 , 15
>>> x
5
>>> y
10
>>> z
15
```

该语句执行过后，变量 x 的值为 5，y 的值为 10，z 的值为 15。这种形式非常有用，假如现在想交换变量 x 与 y 的值，可以直接写：

```
>>> x , y = y , x
>>> x
10
>>> y
5
```

3.1.3 常量

常量是指在程序执行过程中不能改变的数据。例如：数字 5，字符串"abc"等都是常量。Python 并没有提供定义常量的保留字，也就是说 Python 没有命名常量，所以在 Python 中不能像 C 语言那样给常量起一个名字。

3.2 基本数据类型

3.2.1 布尔型

Python 3 支持布尔类型，其表示及运算与布尔代数完全一致。一个布尔值只有 True、False 两种值。在 Python 中，可以直接用 True、False 表示布尔值，也可以通过布尔运算计算出来。比如：

```
>>> 3 < 5
True
```

3.2.2 数值类型

Python 3 中提供了三种不同的数值类型：整数（int）、浮点数（float）和复数（complex）。

在 Python 3 中我们可以使用多种进制的整数，并且整数可以用有无穷精度。也就是

说，在 Python 3 中整数的取值范围理论上是正负无穷大，所以 Python 在计算大数或高精度计算时很有优势。当然，数据实际的取值范围也会受到计算机内存大小的限制。

整数的表现形式以十进制数字字符串写法出现，与数学上一样。如：1、32、1024、－1805。此外 Python 3 中还可以使用二进制、八进制和十六进制等形式，如表 3-2 所示。

表 3-2　整数类型的 4 种进制表示

进制种类	引导符号	说　明
十进制	无	默认情况，如 1000、－739
二进制	0b 或 0B	由 0 和 1 组成，如 0b1011、0B1001
八进制	0o 或 0O	由 0 到 7 组成，如 0o4567、0O3210
十六进制	0x 或 0X	由 0 到 9、a 到 f、A 到 F 组成，如 0xFF、0X0D

Python 3 中的浮点型类似 C 语言中的 double 类型，是双精度浮点型。Python 3 要求所有浮点数必须带有小数部分，小数部分可以是 0。浮点数有两种表示方法：十进制表示和科学记数法表示。如：1.23、3.14、1.2e－5（表示 1.2 乘以 10 的－5 次方）等。

Python 3 中浮点数的数值范围有限制，小数精度也有限制。这种限制与我们用的计算机系统本身有关。我们可以通过运行函数 float_info 获取当前系统中浮点数的各项参数，如：

```
>>> import sys
>>> sys.float_info
sys.float_info(max = 1.7976931348623157e + 308, max_exp = 1024, max_10_exp = 308, min =
2.2250738585072014e - 308, min_exp = - 1021, min_10_exp = - 307, dig = 15, mant_dig = 53,
epsilon = 2.220446049250313e - 16, radix = 2, rounds = 1)
```

这里我们可以获得当前系统浮点数类型能表示的最大值（max）、最小值（min）、科学记数法表示下最大值的幂（max_10_exp）、最小值的幂（min_10_exp）、基数为 2 时最大值的幂（max_exp）、最小值的幂（min_exp）、科学记数法表示中系数的最大精度（mant_dig），计算机所能分辨的两个浮点数的最小差值（epsilon）、能准确计算的浮点数最大个数（dig）等信息。

对浮点数应尽量避免直接比较两个数是否相等，如：

```
>>> 2.3 - 2.1 == 0.2        #两个等号(==)的含义是相等,相当于数学上的等号
False
```

如果我们想判断两个浮点数是否相等，通常是看这两个浮点数之差的绝对值是否小于一个非常小的数，比如 10 的负 7 次方。例如：

```
>> abs(2.000000001 - 1.9999999995)< = 1e - 7
True  #表示两个浮点数相等,虽然它们在数学上是不等的,但由于两个数差的绝对值很小,在精度
#要求不太高时,可以认为是相等的。如果精度要求高,可以进一步加大
```

Python 3 支持复数。复数由实数部分和虚数部分构成，复数的虚数部分通过后缀“j”或“J”来表示，如：12.3＋45.6j、7.8－2.9J。

复数的实部和虚部的数值都是浮点型。对于复数 z，可以通过 z.real 和 z.imag 分别获得它的实数部分和虚数部分，如：

```
>>> (8.6 - 6.2j).real
8.6
>>> (8.6 - 6.2j).imag
```

-6.2

Python 3.7 支持在数值中间位置使用单个下画线作为分隔来提高数值的可读性，类似于数学上使用逗号作为千位分隔符。在 Python 数字中单个下画线可以出现在中间任意位置，但不能出现开头和结尾位置，也不能使用多个连续的下画线。如：1_000_000_000 表示整数 1000000000，0x12_34 表示十六进制整数 1234（十进制为 4660），6_8+3_4j 表示复数（68+34j），1_23.4_5 表示浮点数 123.45 等。

3.2.3　字符串

在 Python 中字符串是以单引号或双引号或三个引号括起来的任意文本。其中单引号和双引号都可以表示单行字符串，两者作用相同。三引号可以表示单行或多行字符串。比如'你好',"世界",'''Hello world''',"""Beautiful is better than ugly."""等。

Python 3 对文本和二进制数据进行了更为清晰的区分。文本内的元素一定是 Unicode 字符，由 str 类型表示，就是所谓的字符串（Python 2 提供了一种专用的表示语法，结果为 Unicode 字符串。这种语法在 Python 3 中依然管用，但是已经多余。更多相关内容可参考 https://docs.python.org/3/howto/unicode.html）。二进制数据则由 bytes 类型表示，就是所谓的字节串。表示 bytes 对象可以使用前缀 b 或 B，比如 b'Hello world'、B'Hello world'等。

在 Python 中没有字符类型，这可能是单引号和双引号在表示单行字符串时没有区别的原因之一。单引号与双引号都可以表示字符串有一个好处，当字符串本身包含单引号时可以用双引号括起来，而当字符串本身包含双引号是可以用单引号括起来。比如"Let's go!"包含的字符是 L、e、t、'、s、空格、g、o、! 这 9 个字符。当然这个字符串也可以写成这样'Let\'s go!'。这里的“\”为转义字符，代表字符单引号。Python 与其他语言类似，也用“\”作为转义字符。字符串和字节串中都可识别的转义字符如表 3-3 所示。只在字符串中可以识别的转义字符如表 3-4 所示。

表 3-3　字符串和字节串中均可识别的转义字符

转义字符	描　述
\新行	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格
\f	进纸符或者换页符
\n	换行
\r	回车
\t	横向制表符
\v	纵向制表符
\ooo	八进制数 ooo 表示的字符，数字可以是 1 到 3 个
\xhh	十六进制数 hh 表示的字符，数字必须是 2 个

表 3-4　只在字符串中可识别的转义字符

转义字符	描　述
\N{name}	Unicode 数据库中用 name 命名的字符
\uxxxx	16 比特十六进制数 xxxx 表示的字符，数字必须是 4 个
\Uxxxxxxxx	32 比特十六进制数 xxxxxxxx 表示的字符，数字必须是 8 个

比如在 Spyder(一种 Python 集成开发环境)中我们输入"This is a cat:\N{Cat}"会有如下结果：

```
>>>"This is a cat:\N{Cat}"
'This is a cat:🐈'
```

这里\N{Cat}代表🐈。为了避免对字符串中的转义字符进行转义，可以使用原始字符串。在字符串前面加上字母 r 或 R 表示原始字符串，其中的所有字符都表示原始的含义而不会进行任何转义。比如：

```
>>> path = 'C:\notepad\notepad++.exe'
>>> print(path)
C:
otepad
otepad++.exe
```

字符串里面的“\n”被转义为换行符，而：

```
>>> path = r'C:\notepad\notepad++.exe'
>>> print(path)
C:\notepad\notepad++.exe
```

这里因为使用了 r 前缀表示是原始字符串，任何字符都不转义，所以得到了我们想要的结果。需要注意的是原始字符串不能以单个反斜杠结尾，如：

```
>>> print(r'C:\notepad\')
SyntaxError: EOL while scanning string literal
```

字符串本质上是一个字符序列，一个字符串的最左端位置标记为 0，依次增加。字符串中这种给字符位置的编号叫作“索引”。Python 中字符串索引从 0 开始，一个长度为 L 的字符串最后一个字符的索引是 L－1，Python 同时允许使用负数从字符串右边末尾向左边进行反向索引，最右侧索引值是－1，这样从末尾开始引用字符就很方便了，如表 3-5 所示。

表 3-5　字符串索引

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
－11	－10	－9	－8	－7	－6	－5	－4	－3	－2	－1

我们可以通过索引辅助访问字符串中的特定位置，如：

```
s = "Hello world"
```

则 s[4]为"o"，s[－3]为"r"。

在 Python 中还支持对字符串的“切片”操作，也就是说可以通过两个索引值确定一个位置范围，返回这个范围的子串。切片的基本格式为：

<string>[<start>:<end>]

start 和 end 都是整数型数值，这个子序列从索引 start 开始直到索引 end 结束，但不包括 end 位置。以字符串 s="Hello World"为例，我们来看一下如何进行切片：

```
>>> s[0:5]
'Hello'
>>> s[6:11]
'world'
>>> s[3:9]
'lo wor'
>>> s[:5]
'Hello'
>>> s[6:]
'world'
>>> s[:]
'Hello World'
```

最后的三个例子表明，如果我们省略了某个值，字符串的开始和结束都是假定的默认值。要特别注意的是，在 Python 中，字符串属于不可变序列类型，也就是说不允许通过索引改变字符串的值，如：s[3]="H" 是不合法的。

在 Python 中字符串之间可以通过＋或 * 进行连接，加法操作(＋)是将两个字符串连接成为一个新的字符串，乘法操作(*)是生成一个由其本身字符串重复连接而成的字符串，如：

```
>>> "Hello" + "World"
'HelloWorld'
>>> "Hello" * 3
'HelloHelloHello'
>>> 5 * "Hello"
'HelloHelloHelloHelloHello'
```

3.2.4　字符串的格式化

Python 支持多种字符串格式化的方法。以前主要使用字符串格式运算符(%)，这种方法与 C 语言中的 printf 函数使用的方法类似，但 Python 已经不会在后续的版本中改进这种方法了。Python 可以采用字符串的 format 方法(这里的方法可以理解为函数)对字符串进行格式化，format 方法的基本使用方式如下：

```
>>> "{}是一种程序设计语言，它目前的版本是{}。".format("Python",3.7)
'Python 是一种程序设计语言，它目前的版本是 3.7。'
```

双引号括起来的是需要显示的字符串，其中的花括号{}括起来的部分为替换字段，每个替换字段都要被 format 方法的参数依次替换(本例的"Python"和 3.7)。如果字符串内需要输出花括号本身，则需要两个花括号，如：

```
>>> 'Format strings contain {} surrounded by curly braces {{}}.'.format("replacement fields")
'Format strings contain replacement fields surrounded by curly braces {}.'
```

作为替换字段的花括号里如果什么也没有，则参数按顺序依次替换。如果花括号中有

序号，则按参数序号依次替换。序号从 0 开始编号，如：

```
>>> "{1}是一种程序设计语言,它目前的版本是{0}。".format(3.7,"Python")
'Python 是一种程序设计语言,它目前的版本是 3.7。'
```

我们可以看到，3.7 替换的是 0 号，"Python"替换的是 1 号。

如果花括号里是名字，则按名字替换，如：

```
>>> "{name}是一种程序设计语言,它目前的版本是{version}。".format(version = 3.7, name =
"Python")
'Python 是一种程序设计语言,它目前的版本是 3.7。'
```

按顺序替换的(无论是否有序号)称为位置参数，按名字替换的称为关键字参数。在一条语句里可以同时使用位置参数和关键字参数，但关键字参数在圆括号内要出现在所有位置参数之后。

在花括号内还可以加格式控制信息，具体方式就是用冒号(:)作为引导符号，后边跟格式控制标记。格式控制标记的基本格式如下：

[[填充]对齐][符号][#][0][宽度][分隔符][.精度][类型]

表 3-6 是格式控制标记。

表 3-6 格式控制标记

标记类型	说　明
填充	用于填充的单个字符
对齐	<左对齐；> 右对齐；^居中对齐；= 仅对数字有效，将填充字符放到符号与数字间，例如+0001234
符号	仅对数字有效，+正数加正号，负数加负号；－正数不变，负数加负号；空格，正数加空格，负数加负号
宽度	数字，为输出宽度
分隔符	,或－，数字的分隔符
精度	浮点数小数的精度或字符串的最大输出长度
类型	b、c、d、e、E、f、F、g、G、n、o、s、x、X、%

井号(#)：对于二进制、八进制、十六进制，如果前边加上井号，会显示 0b/0o/0x。

类型：类型用于确定参数的类型，默认为 s，即字符串。

对于整数：

- b 表示以二进制格式输出；
- c 表示将整数转换成对应的 Unicode 字符；
- d 表示以十进制输出(默认选项)；
- o 表示以八进制输出；
- x/X 表示以十六进制小/大写输出；
- n 与 d 相同，但使用当前环境的分隔符来分隔每 3 位数字。

对于浮点数：

- e/E 为指数标记，使用科学记数法输出，用 e/E 来表示指数部分，默认精度为 6；
- f/F 表示以定点形式输出数值，默认精度为 6；
- g/G 为通用格式，可自动在 e/E 和 f/F 间切换；

- n 与 g 相同，但使用当前环境的分隔符来分隔每 3 位数字；
- %为百分比标记，使用百分比的形式输出数值，同时设定 f 标记。

我们再来看一些例子：

```
>>> '{:<30}'.format('Hello World')                    #左对齐
'Hello World                   '
>>> '{:*^30}'.format('Hello World')                   #居中对齐,用"*"填充
'**********Hello World**********'
>>> '{:+f}; {:+f}'.format(1.23, -4.56)                #总是显示符号
'+1.230000; -4.560000'
>>> '{:_}'.format(1234567890)                         #用_作为千位分隔符
'1_234_567_890'
>>> '{:.2f}'.format(12.345)                           #保留 2 位小数
'12.35'
>>> '{:#b}'.format(121)                               #二进制
'0b1111001'
>>> '{0:%}'.format(1.23)                              #百分数
'123.000000%
```

从 Python 3.6 开始支持一种新的字符串格式化方式，官方称为 Formatted String Literals，或简称 f-string。就是在字符串前加字母 f(大小写均可)，字符串里花括号内部可以进行值替换，也同样支持格式控制标记(与字符串的 format 方法一致)。比如：

```
>>> f'{7 * 23}'                                       #任意表达式
'161'
>>> name = 'Zhang'
>>> age = 19
>>> f'My name is {name}, and I am {age} years old.'   #普通变量替换
'My name is Zhang, and I am 19 years old.'
>>> width = 15
>>> precision = 4
>>> value = 81/7
>>> f'val = :{value:{width}.{precision}}'             #宽度 15,精度 4
'val = :         11.57'
```

f-string 字符串简洁实用、可读性高，而且不易出错，是我们推荐使用的方法。更多关于 f-string 的内容请参考 https://www.python.org/dev/peps/pep-0498/。

3.2.5 空值

Python 有一个特殊的类型，被称为 Null 对象或 NoneType，它只有一个值就是 None。它不支持任何运算，没有任何内置方法，也没什么有用的属性，它的布尔值总是 False。None 不能理解为 0 或者空字符串等，因为这些值是有意义的，而 None 是一个特殊的空值。

3.2.6 常用数据类型的转换

和其他语言类似，当不同的数据类型进行混合运算时(合法的)，Python 会隐式地进行数据类型的转换。转换的原则是将所有的数据类型统一到最“大”的类型。如整数和浮点数进行混合运算，运算结果为浮点数；浮点数和复数进行混合运算，运算结果为复数。此外

Python 还提供了一些内置函数可以对数据类型进行显式的转换，这里介绍几种常用的数据类型转换函数。

内置函数 bin()、oct()、hex()用来将整数转换为二进制、八进制和十六进制形式，这三个函数都要求参数必须为整数。如：

```
>>> bin(100)
'0b1100100'
>>> oct(100)
'0o144'
>>> hex(100)
'0x64'
```

内置函数 int()用来将其他形式的数转换为整数，参数可以为整数、实数、分数或合法的数字字符串。当参数为数字字符串时，还允许指定第二个参数 base 用来说明数字字符串的进制，base 的取值应为 0 或 2～36 的整数，其中 0 表示按数字字符串隐含的进制进行转换。如：

```
>>> int(3.8)                    #把实数转换为整数，不是四舍五入，而是直接舍弃小数部分
3
>>> int('100')                  #把十进制字符串'100'转换为整数 100
100
>>> int('0b1100100',2)          #非十进制字符串必须指定第二个参数
100
```

内置函数 float()用来将其他类型数据转换为实数，如：

```
>>> float(5)                    #把整数转换为浮点数
5.0
>>> float('5.6')                #把字符串转换为浮点数
5.6
```

内置函数 str()可以将其任意类型参数转换为字符串。

```
>>> str(100)                    #把整数转换为字符串
'100'
```

3.3 运算符及表达式

表达式是程序设计语言中最基本的计算描述手段，是描述计算值过程的一个抽象层次。表达式通常是由运算符、操作数和括号组成的，是用于计算求值的基本单位。运算符是实现各种或各类运算的符号，用于表明要完成的计算类型。数学中的加、减、乘、除就是很典型的运算符。操作数是要接受运算的数，可以是变量、常量、函数等。每一个运算符都有一个、两个或多个操作数。如：3、3＋2、x－y 等都是合法的表达式。

运算符是构建表达式的基本工具，Python 语言提供了非常丰富的运算符，这里介绍一些基本的运算符，特殊的运算符我们用到时再做介绍。

3.3.1 算术运算符

Python 3 的算数运算符及其含义如表 3-7 所示。

表 3-7 算术运算符

运 算 符	描 述
+	加法运算符
-	减法运算符
*	乘法运算符
**	幂运算符
/	除法运算符
//	整除(或者称地板除)运算符,即求得两个整数相除的整数商
%	取余运算符

这些运算符与数学用法习惯一致,运算结果也符合数学意义,但有几个地方要解释一下:

(1) 加法运算及乘法运算还可以用于字符串类型(前面已经介绍过)。

(2) 幂运算支持复数、实数及整数。如:

```
>>> 2 ** 4
16
>>> 0 ** 0
1
```

(3) 除法运算是数学意义上的除法,特别要注意的是如果两个整数相除,无论商是不是整数,在 Python 3 中运算结果都是浮点数(和 C 语言有区别,C 语言的整数除法求得的是整数商)。如:

```
>>> 6 / 3
2.0
```

(4) 整除运算可以用于实数和整数,其结果不四舍五入,而是直接向下取整。如果两个操作数都是整数,则结果为整数,如果操作数中有实数,则结果为实数形式的整数值。如:

```
>>> 15 //4
3
>>> -15//4
-4
>>> 15.0//4
3.0
>>> 17.4 // 2.5
6.0
```

(5) 取余运算可用于实数和整数,如:

```
>>> 10 % 3
1
>>> 24.6 % 2.9
1.4000000000000021
```

3.3.2 逻辑运算符

Python 3 中逻辑运算符为 and、or、not(分别代表与运算、或运算、非运算)。其运算规

则与布尔代数中的与运算、或运算、非运算一致，但有两点要特别注意：

第一点是和C语言等其他程序设计语言一样，and和or具有惰性求值或逻辑短路的特点，当连接多个表达式时只计算必须要计算的值。如：

```
>>> 5 < 8 or 3 / 0
True
```

这里由于5<8为True，那么整个表达式的运算结果一定为True，所以or后边的表达式被短路掉，并没有被计算。我们把前边的内容改一下，会有不同的结果，可以验证我们的说法。如：

```
>>> 5 > 8 or 3 / 0
Traceback (most recent call last):
File "< pyshell # 20 >", line 1, in < module >
5 > 8 or 3 / 0
ZeroDivisionError: division by zero
```

这里由于5>8为False，所以要计算or后边的表达式才可能知道整个表达式的结果，就出现了0不能作除数的错误。

第二点是运算符and和or并不一定会返回True或False，而是将最后一个被计算的表达式的值作为结果，但是运算符not一定会返回True或False。如：

```
>>> 5 and 8          # 最后一个计算的表达式的值作为整个表达式的值
8
>>> 5 or 8           # 此处运用了逻辑短路
5
>>> not 8
False
>>> not 0
True
```

3.3.3 比较运算符

比较运算符也叫关系运算符，Python 3中的比较运算符包括==、!=、<、>、<=、>=，分别表示等于、不等于、小于、大于、小于或等于和大于或等于。这些运算符的含义与数学上的含义一致，运算结果为布尔值True或False。使用关系运算符的一个最重要的前提是，操作数之间必须可比较。如果把一个字符串和一个数字进行比较是毫无意义的，所以Python也不支持这样的运算。如：

```
>>> 3 == 5
False
>>> 3 < 5
True
```

Python关系运算符最大的特点是可以连用，其含义与我们日常的理解完全一致。如：

```
>>> 3 < 5 > 1            # 等价于 3 < 5 and 5 > 1
True
>>> 1 < 2 < 3
```

```
True
>>> 1 < 2 > 3
False
>>> a,b,c = 1,2,3
>>> a < b < c          # 在 C 语言中,判断 b 是否在 a 和 c 之间,应写成 a < b && b < c
True
```

3.3.4　赋值运算符

前边已经介绍过在 Python 中“=”是主要的赋值操作符,除此之外,表 3-7 中所有运算符(+、-、*、**、/、//、%)都有与之对应的增强赋值运算符(+=、-+、*=、**=、/=、//=、%=)。这些运算符与 C、Java 等语言的使用方式一样,如果用 op 表示表 3-7 中运算符,则下面两个赋值操作是等价的:

```
x op= y          # 注意: op= 是一个符号,不可以分开写
x = x op y
```

增强赋值运算符将计算的结果存入变量 x,简化了代码。如:

```
>>> x , y = 2 , 3
>>> x ** = y          # 等价于 x = x ** y
>>> x
8
>>> x += y            # 等价于 x = x + y
>>> x
11
```

要注意的是与 C、Java 等语言不同的是 Python 没有自增(++)和自减(--)运算符。

注:准确地讲在 Python 中,我们介绍的所有赋值符号都不能叫运算符,应该叫分隔符。因此在 Python 中可以将表达式分为算术表达式、关系表达式、逻辑表达式等,但没有赋值表达式。

3.3.5　运算符的优先级

对表达式计算过程的控制手段通常包括优先级(比如先算乘除后算加减)、结合顺序(比如左结合)、括号(提高优先级)。在 Python 3 中运算符的优先级通常是算术运算符高于关系运算符,关系运算符高于逻辑运算符。我们介绍过的运算符优先级如表 3-8 所示(表中运算符按优先级由低到高排列,同一行内的运算符优先级相同)。

表 3-8　运算符的优先级(由低到高)

运　算　符	描　　述
or	或运算符
and	与运算符
not	非运算符
<,<=,>,>=,!=,==	比较运算符
+,-	加法和减法运算符
*,/,%	乘法、除法和取余运算符
**	幂运算符

虽然 Python 运算符有一套严格的优先级规则，但我们仍然强烈建议在编写复杂表达式时，使用圆括号来明确说明其中的逻辑，以提高代码的可读性。

3.4 基本输入/输出

一个真正有用的程序通常要有输入和输出语句，以实现程序和外界环境的联系（交换信息），一个没有输出的程序是没有用的（有极少数程序可以没有输入）。在 Python 3 中分别使用 input()函数和 print()函数来实现标准控制台的输入和输出。

3.4.1 input()

在 Python3 中将 Python 2 中的 raw_input()和 input()进行了整合，去除了 raw_input()，仅保留了 input()函数，其接收任意输入，将所有输入默认为字符串处理，并返回字符串类型。

在获得用户输入之前 input()函数可以包含一些提示性文字。如：

```
>>> x = input("请输入:")
请输入:17
>>> x
'17'
```

可以看出"请输入："为提示性文字，x 的数据类型为字符串。如果我们需要将数据定义为其他的数据类型就要用 int()、float()等函数进行转换。如：

```
>>> x = int(input("请输入一个整数:"))
请输入一个整数:17
>>> x          #这时 x 的数据类型则为整型
17
```

如果需要在一行中输入多个数据，可以使用 split()函数进行分隔。如果输入多个数据之间用空格隔开，可以直接使用 split()，不需要在单引号中加上一个空格。

```
>>> n,m = input().split()
12 34
>>> print(n,m,n + m)
12 34 1234        #多个数据之间默认用空格隔开
```

这里的 n 和 m 都是字符串的类型，所以 n＋m 是字符串的连接，而不是整数之和。

如果要求在输入的多个数据之间用特殊的间隔符号隔开，在 split()中的参数就是这个间隔符号，并用单引号括起来。如果需要将输入的数据作为整数直接赋值给 n 和 m，则需要在前面加上 map()函数，将输入的数据转换为 int 类型。例如：

```
>>> n,m = map(int, input().split(','))      #将两个输入的数作为整数分别赋值给 n 和 m
12,34
>>> print(n,m,n + m)
12 34 46                          #这里 n 和 m 都是整数，所以加法的结果是 46，而不是字符串连接
```

3.4.2 print()

Python 3 的内置函数 print()用于输出信息到标准控制台或指定文件，语法格式为：

```
print( * objects, sep = ' ', end = '\n', file = sys.stdout, flush = False)
```

- 参数 objects 为需要输出的内容(可以有多个)。
- 参数 sep 用于指定数据之间的分隔符，默认为空格。
- 参数 end 用于指定输出的结尾，默认是换行符(\n)。如果不换行输出，则 end 中的参数为""。
- 参数 file 用于指定输出位置，默认为标准控制台，也可以重定向输出到文件。
- 参数 flush 确定输出是否使用缓存(默认 False)，通常是否使用缓存由 file 决定，但当 flush 值为 True 时，则强制清除缓存，立刻将所有内容输出到参数 file 指向的对象中。

我们看几个例子：

```
>>> print("Hello world")
Hello world
>>> print(3, 5, 7)                          #一次输出多个值
3 5 7
>>> print(3, 5, 7, sep = ',')               #指定分隔符
3,5,7
>>> print('100 + 200 = ', 100 + 200)
100 + 200 = 300
>>> a,b,c,d = map(int,input().split())      #输入 4 个整数
12 34 56 78
>>> print(a,b,end = "")    #输出 b 之后并不换行，后面的输出紧跟在 b 的内容之后输出
>>> print(c,d,end = "")    #输出 d 之后并不换行，后面的输出紧跟在 d 的内容之后输出
>>> print("end")
12 3456 78end              #a 和 b 之间有空格隔开，但 b 和 c 之间没有间隔。c 和 d 之间有空格隔
                           #开，但 d 和字符串"end"之间没有间隔
```

print()函数结合字符串的 format 方法或 f-string 可以得到各种格式化的输出，具体可参考 3.2.4 节。