**第3章 数值类型**

数值类型(Numeric Types )用于存储数值，它是Python中最基础的一种变量类型,也是 Python程序实现计算功能的基础o Python中可支持的常用数值类型有整型(int)、浮点型(float)、 复数类型(complex)和布尔类型(bool)。

XI舞型

3JJ

类似-2、-1、0、1、2这样的数被称为整型数据，简称整型。

早期Python中普通整型是有限制的，它的取值范围与C语言中长整型的取值范围相同， 即-23i^231-l(32位机中)，换言之，Python中的普通整型最长只能为32位，若值超出此范围 便会报错。

后来Python放宽了对整型的限制，在23版的Python中，若在终端输入一个较长的整数， 解释器不会报错，而是在整数后添加一个“L”，表示长整型，如下所示。

»> 10000000000000000000

10000000000000000000L

用户也可显式地定义长整型，只需在普通整数之后添加字符或T'即可(推荐使用*，* 可避免与数字1混淆)。示例如下：

»> a = 1000L

type()函数可查看对象的类型。使用type()函数查看以上定义的对象a的类型，可以看到 其长整型为"long",如下所示：

>>> type(a)

<type 1 long1>

3J.2 Pjthoo 3

再后来,Python有意地消除对整型数据的长度限制oPython 3版本中取消了对长整型的定义, 整数类型不再受到内部规定位数的影响，其取值范围只与计算机的内存有关，换言之，只要计

第3章数值类型*遍*43 算机的内存足够大，Python的整型数据就能足够长。因此用户几乎无须考虑溢出问题。

3JD3篷醴髓骚素方還

在Pythom中可以使用4种不同的进制表示整型，分别为：二进制(以“0B”或“Ob”开头)、

八进制(以“0。”或“00”开头)、十进制和十六进制(以“Ox”或“0X”开头)*。*示例如下:

|  |  |
| --- | --- |
| »> a = ObOllll  »> type (a) <class \*int\*>  >>> b = 0o71  >>> type(b) <class v int? >  >>> c = 0x3A  »> type (c) <class ? int? > | #二进制  #八进制  #十六进制 |

3.14 ®!W

Python内置了一系列进制转换函数,包括bin()、oct()、hex()和int(),下面将对这些函数 的功能逐一进行讲解。

1 bin()

bin()函数用于将十进制整数转换为一个符合二进制整型规范的字符串，其中参数为十进制 整数，其用法如下所示。

|  |  |
| --- | --- |
| >>> bin(23)  'OblOlll' | #将十进制转换为二进制  #获得二进制23的字符串 |

2. oct()

oct()函数用于将十进制整数转换为一个符合八进制整型规范的字符串，其中参数为十进制 整数，其用法如下所示。

|  |  |
| --- | --- |
| >>> oct(23)  1 0o27' | #将十进制转换为八进制  #获得八进制23的字符串 |

3. hex()

hex()函数用于将十进制整数转换为一个符合十六进制规范的字符串，其中参数为十进制 整数，其用法如下所示。

|  |  |
| --- | --- |
| >» hex (23)  '0x171 | #将十进制转换为十六进制  #获得十六进制23的字符串 |

4. int()

int()函数可接收一个符合整型规范的字符串，并将该字符串转换为整型。实际上，除符合 十进制整型的字符串外，into函数还可接收符合二进制、八进制和十六进制整型规范的字符串， 只是在转换非十进制规范的字符串时，into函数还需传入该字符串中包含的整型的进制，其格 式如下:

int(strz radix)

下面通过示例来展示将符合不同进制整型规范的字符串转换为十进制的方法。

44 毎捉 Pythoo SttSSfio Pyfihon

|  |  |  |
| --- | --- | --- |
| »> | int(?0b0111001\2) | #二进制 |
| 57 |  |  |
| »> | int(?0o75\8) | #八进制 |
| 61 |  |  |
| »> | int(!0xAF!,16) | #十六进制 |
| 175 |  |  |

结合以上4个进制转换函数，可在Python中实现各种类型之间的互转。二进制、八进制、 十进制和十六进制之间互相转换的方式如表34所示。

表3-1进制转换表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 二迸制 | 八进制 | 十进制 | 十六进制 |
| 二进制 | 一 | bin(int(x,8)) | bin(int(x510)) | bin(int(x, 16)) |
| 八进制 | oct(int(x,2)) | — | oct(int(x,10)) | oct(int(x,16)) |
| 十进制 | int(x,2) | int(x,8) | 一 | int(x,16) |
| 十六进制 | hex(int(x,2)) | hex(int(x,8)) | hex(int(x,10)) | —Will" |

& 2位运算

程序中的所有数据在计算机内存中都以二进制形式存储，位运算即以二进制位为单位进行 的操作。Pythg中的位运算主要有6种，具体如下：

® vv：按位左移。

® »：按位右移O

® &：按位与。

@ |：按位或。

® A：按位异或。

*陰*〜：按位取反。

3.2J

Python中的位运算是针对整型数据进行运算，在讲解位运算之前，先来学习一下整数在计 算机内存中的存储方式。

计算机只能识别高低电平•（高为1，低为0）,因此计算机内的数据都以二进制形式存储， 但为了方便计算，计算机在内存中存储的不是整型数据本身，而是整数的补码。

整数的二进制表示形式有3种，即原码、反码和补码,这三种表示方法都分为符号位和数 值两部分,正数的符号位统一为“0”，负数的符号位统一为T o

«原码；整数原码的数值部分由其他进制的数值直接转换而来，此外最高位添加符号位0 或lo如9的原码为01001, -9的原码为11001等。

•反码：正数的反码与其原码相同，负数的反码为将其原码的数值部分按位取反（即0变1,

1变0）*，*符号位保持不变。如9的反码为01001, -9的反码为10110等。 •

补码：正数的补码仍与原码相同，负数的补码为保留符号位，使数值部分在反码的基 础上加如9的补码为01001, -9的补码为10111等。

3.2.2 Ufa®®

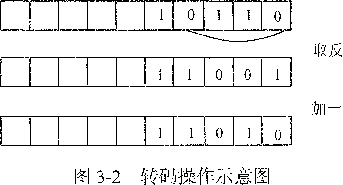
按位取反需要使用符号“〜”来操作，示例如下：

~ob j

以上语句表示将操作数。bj按位取反。按位取反遵循的规则为：位为1则取其反0,位为0 则取其反1 （包含符号位）。

以十进制9 （二进制值为01001 ）的按位取反操作为例，实现过程如图34所示。

由于计算机以补码为基础进行运算，因此需经过转码才能获取操作结果的原码，转码过程 如图3-2所示。



0 10 0 1

10 110

图3-4按位取反操作示意图

由图3・2可知，执行“〜9”操作后获得的二进制整数原码为11010,即十进制的-10。在 Python解释器中验证以上操作:

>>> bin（〜9）

{~0bl010?

»>〜9

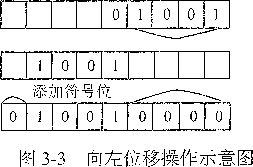
观察表达式执行结果，与图3『2中操作结果相同。

按位取反是掌握负数在计算机中的存储方式，即掌握负数原码转反码和补码操作的基础，读 者应先掌握按位取反运算符的用法，理解计算机中数字存储的原理，再继续其他运算符的学习。

3n2D3

按位左移需要使用符号“〈V”来操作，示例如下：

obj << n

以上语句表示将操作数。bj的二进制位向左平移11位，低位补零。在其他语言中，因整型 位数有限，所以高位溢出时会将其丢弃，但Python 3中整型数据的取值范围极大，因此Python 中的位移不存在溢出问题。

左移

补位

以十进制的9 （二进制值为01001）为例,将其左移4 *位,*则其实现过程如图3-3所示。

由图3；可知，执行“9«4”操作后得到的二进制整 数应*为*0100100009即十进制的144。下面在Python解释

46 w Pyfth©o 映饕掌 Pjthon

器中验证以上操作：

>>> bin(9<<4)

'OblOOlOOOO ?

»> 9«4

144

观察表达式执行结果，与图3；中操作结果相同。

值得一提的是，左移操作相当于“obj\*(2\*\*n)”，即操作数乘以2的n次方，用户可借此 原理自行实现按位左移功能。

按位与需要使用符号“>>”来操作，示例如下:

ob j » n

以上语句表示将操作数。bj的二进制位向右平移11位，移出的低位被舍弃，高位补一位符号位。 以十进制的9 (二进制值>01001)*为*例,将其右移2位,则实现过程如图3-4所示。

由图3.4可知，执行“9>>2”操作后得到的二进制整数应为010，即十进制的2。在 Python解释器中验证以上操作:

»> bin (9»2)

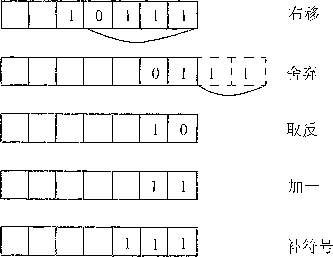
'OblO1

»> 9»2

2

'观察表达式执行结果，与图&4中操作结果相同。

在计算机内部，由于负数以补码形式存储，因此实际上计算机在对负数进行位移操作后， 需先经过转码才会将结果返回到终端。以十进制的-9(二进制原码为11001,补码为10111 )为例, 将其右移2位，则实现过程如图*¥5*所示。



|  |  | |o| | 111 | 0 | |o| | 11 | 右移 |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | | |  |
|  |  |  |  |  | 1 | 0 | 0 II J舍弃 |
|  | | |  |  |  |  |  |
|  |  |  |  | 0  V / | 1 | 0 | 补付巧' |

图3-4向右位移操作Z5意图

图3<负数右位移操作示意图

由图3峪可知，执行“-9>>2”操作后得到的二进制整数应为111,即十进制的-3。在 Python解释器中验证以上操作:

>>> bin (-9>>2)

?-0bll1

»> -9»2

-3

观察表达式执行结果，与图3.5中操作结果相同。

向左位移可通过获取操作数obj与2”的乘积实现，理论上向右位移可通过获取操作数obj 与2〃的商实现，但并非所有操作数都能被整除，因此在模拟实现右位移操作时，使用的表达式 应为 “obj〃(2\*\*n)”，而非 “obj/(2\*\*n)”。

3d2q5 '

按位与需要使用符号“&”来操作，示例如下：

obj 1 & obj 2

以上语句表示将操作数。bjl与操作数obj2按位与，并返回操作结果。按位与操作遵循的 规律为：若相与的两个二进制位都为L结果则为L否则为0。

以十进制9(二进制值为01001 )和3(二进制值为011 )的按位与操作为例，实现过程如图34 所示。

由图3-6可知，执行“9&3”操作后得到的二进制整数为00001，即十进制的1。在Python 解释器中验证以上操作：

>>> bin(9&3)

!0bl?

»> 9&3

1 ..

观察表达式执行结果,与图3-6中操作结果相同o

由于正数的符号位为0,负数的符号位为1,因此两个负数进行位与操作的结果必定为一 个负数；由于负数的补码与原码不同，因此计算机内部在进行位与操作后，需经过转码才会返 回结果的原码。以十进制数-9 (补码为10111)与-6 (10010)的位与操作为例，其实现过程 如图3・7所示。

& rnTTFWTF

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| 丨丨丨丨丨丨。| | 1丨。丨。丨1丨 | LLI 1 1 hlohliH |  |
|  |  |  | 取反 |
| &丨丨丨丨丨「。| | 。|。|小丨 | 1丨丨丨丨UI,HU'E |  |
| *—一—~——* |  |  | 加一 |
| 丨丨丨丨丨 | 0丨0丨0丨1丨 | 丨丨丨丨丨UEIFl |  |
| 图3»6位与操作示意图 | | 图3-7负数位与操作示意图 |  |

由图3-7可知，执行9&-6”操作后得到的二进制整数为HHO,即十进制的T4。在 Python解释器中验证以上操作:

>>> bin(-9&-6)

\*-0blll0?

»> -9&~6

-14

观察表达式执行结果，与图3-7中操作结果相同。

0 10 0 1

0 10 11

观察表达式执行结果，与图3・9中操作结果相同。

图3-9按位异或操作示意图

48 & 囊臆疆齡 關饕学Pjfthon

3.2.6浚癒襲

按位或需要使用符号T'来操作，示例如下:

obj 1 | obj 2

以上语句表示将操作数objl与操作数obj2按位或，并返回操作结果。按位或操作遵循的 规律为：若相或的两个二进制位中有L结果为L否则为0。

以十进制9（二进制值为01001 ）和3（二进制值为011 ）的按位或操作为例，实现过程如图3屹 所示。

由图3・8可知，执行“9|3”操作后得到的二进制整数为010H,即十进制的1"在Python 解释器中验证以上操作: .

p 1 o p i

»> bin （9 I 3） 1————1~1——L———I—L\_\_l——i

|  |  |
| --- | --- |
| 'OblOll! | 1 I I I I I | “ 1 c 1 n 1 1 1 1 1 |
| »> 9 | 3 | ! 1丨丨丨丨丨°丨。丨。丨1丨1 1 |
| 11 |  |
| 观察表达式执行结果，与图3-8中操作结果相同。 | 丨丨丨丨丨丨。"丨。1小丨 |
|  |
| 3.2.7漏密異戢 | 图3-8按位或操作示意图 |

按位异或需要使用符号％”来操作，示例如下:

objl A obj2

以上语句表示将操作数。bjl与操作数obj2按位异或，并返回操作结果。按位异或操作遵 循的规律为：若进行异或的两个二进制位不同，结果为1，否则为0。

以十进制9 （二进制值为01001 ）和3 （二进制值为011 ）的按位异或操作为例，实现过程 如图3-9所示。

由图3・9可知，执行“9人3"操作后得到的二进制整数为01010,即十进制的10。在Python 解释器中验证以上操作：

»> bin （9A3）

^blOlO1

»> 9A3

10

X3浮点型

3h3d1 .

浮点型用于表示实数，如3」4、528、6X）等。

在Pythoii中，浮点数一般以十进制或科学计数法表示，示例如下:

»> a = 314.15

»> b = 3.14e2

以上定义的两个浮点型数据大小相同，其中b使用科学计数法定义，Python中科学计数法 定义浮点数的格式如下：

*尾数*e阶码

尾数E阶码

以上格式中。和E含义相同，都表示阶码的基——10；尾数和阶码可正可负，分别使用“+” 或表示，若为正数，则“+”可省略（上例中定义的对象b便省略了 “+”）。

3.3.2薄盧數H麒僵范圖

Python中的浮点型是双精度的，每个浮点型数据占8个字节（即64位），且遵守IEEE标准， 其中52位用于存储尾数，11位用于存储阶码，剩余一位则存储符号。Pythoii中浮点型的取值 范围为-L8e308〜L8e308,若超出这个范围，Python则会将值视为无穷大或无穷小。示例如下：

>>>3 »14e5

314000.0

»> 1/3

0.3333333333333333

»> 100/3

33.333333333333336

3.3.3濤寫歡齣彥備

使用Python 25之前版本的读者，在定义浮点数时可能会遇到如下问题旨

»> a = 3.13

»> a

3.1299999999999999

以上代码中，定义了一个值为3」3的浮点型数字我，但解释器打印的结果却只是一个与3」3 接近的包含17位有效数字的浮点数。之所以出现这种情况，是因为Python中浮点数是用二进 制小数（进制的知识将在后续小节讲解）存储的，而用户在定义浮点数时使用十进制小数表示， Python中浮点数的位数有限，因此在十进制小数转换为二进制小数时会发生截断。

大多数十进制小数在转换为二进制小数时都会出现上述问题，如此一来，当多个不够精确 的浮点数进行运算后，所得结果的偏差可能更加明显。但是，实际情况下膏一般用户并不需要 担心这一现象，因为Pythoii浮点数的精度可以满足大多数计算需求，且Python 2/7之后的版本 中也对上述问题进行了一定的处理。

3.3.4葛癩廈縁慮戀 -

Pythg中浮点型的有效位只有17位，若有效位超过17位膏Python 2/7之后的版本同样会 对其进行截断,如在3.6版本中输入一个超长的浮点数:

»> 3.12345 97134717 35 61984 91

3.1234597134717355

我们发现实际存储的仍然是只有17位有效数字的浮点数，此时若想要使用更高精度的浮 点数，则需通过decimal模块进行定义。

decimal模块实现的十进制运算可满足对精度要求较高的场合，其用法如下：

>>> import decimal

»> a = decimal.Decimal(f2.12313141432354 435 66v )

>» print (a)

2.1231314143235443566

该方法在Python 24及更高版本中都可使用。

3O4复數类型

复数和虚数最早出现在数学领域。“虚数”由17世纪的著名数学家笛卡儿提出，用于解 决负数的平方根问题，通常记为i或j (Python中一般记为j),数学中认为i2=-lo复数在18 世纪末逐渐为大多数人所接受。

自L4版本起，Python中加入了复数类型(Complex Type )简称复数。形似3+2j、 3」+4°9j、\_23L9j这样的数，在Python中都被称为复数。复数由“实部”和“虚部”两部分组 成，其中实部是一个实数，虚部则是一个实数与j (例如2j、49、-L9j等)的组合。

3c4o2

Python中的复数类型有如下几个特点：

®复数由实部和虚部构成，其一般形式为：real+imagj;

•实部real和虚部imag都是浮点型；

•虚部必须有后缀j或J。

3^4.3

在Python中可以直接将一个复数赋给一个变量，例如:

»> a = 3+2j

>>> print(a)

(3+2j)

使用type。函数查看变量a的类型，将会获取其类型“complex”，如下所示。

>>> type(a)

<class 9 complex 1>

也可以使用内建函数complex(real?imag),通过传入实部和虚部的方式定义复数,例如:

>>> a = complex(3 7 2)

>>> print(a)

(3+2j)

若imag缺省\*则imag使用默认值0o示例如下:

>>> complex(3)

(3+0j)

3.4.4

复数的实部和虚部可以通过符号“了获取，如获取以上定义的复数初的实部和虚部，可以 使用如下方式：

>>> a.real

3.0

>>> a.imag

2.0

3O5

Python中的布尔类型（Bool Type）只有两个取值：True （真）和False （假）。实际上布 尔类型也是整型，其值True对应整数，Fake对应整数0。布尔类型一般用于布尔测试，一般 情况下布尔测试的结果只有True和False （但也有例外）。

306数宇运算

和其他编程语言相比，Python中的运算符更为丰富，且功能更为强大，因此Pythoti中的 数据可以以相对简单的方式，实现丰富的运算功能。Pythcm中的运算符可分为如下几类：算术 运算符、赋值运算符、比较运算符、逻辑运算符等。

Pythcm中的算术运算符有：+、-、\*、/、〃、％和\*七这些运算符都是双目运算符，在终端 输入由两个操作数和一个运算符组成的表达式，Pythoii解释器就会解析表达式，并打印计算结果。

以操作数a = 3, b = 5为例,Python中各个运算符的功能及示例如表3『2所示。

羨3或算术运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 说 明 | 示 例 |
| + | 加:使两个操作数相加，获取操作数的和 | a + b,结果为8 |
| II—lllll | 减：使两个操作数相减，获取操作数的差 | a-b 9结果为-2 |
| \* | 乘：使两个操作数相乘，获取操作数的积 | a \* b,结果为15 |
|  | 除：使两个操作数相除，获取操作数的商 | a/b,结果为0.6 |
| // | 整除：使两个操作数相除，获取商的整数部分 | a//b,结果为0 |
| % | 取余：使两个操作数相除，获取余数 | a%b,结果为2 |
| \*\* | 幕：使两个操作数进行冨运算，获取objl的obj2次驀 | a \*\* b,结果*为*243 |

在终端中直接输入示例中的表达式，结果将会直接在终端输出，但在此过程中，运算符两 侧的操作数a和b并没有被修改，这是因为运算的结果并未被保存到变量a或b之中。以加法为例, 示例如下:

>>> a = 3

»> b = 5

»> a + b

8

>>> print(a)

3

>>> print(b)

5

Pythoii中的算术运算符支持对相同或不同类型的数字进行各种运算(注意：对复数进行除 +、-、\*之外的运算时可能会产生警告)，且无须进行转换，例如进行如下的混合运算：

»> 3 + (3+2j)

#整型+*复数*

(6+2j)

»> 3 \* 4«,5

#整型\*浮点型

13.5

»> 5.5 - (2 + 3j)

#浮点型-复数

(3.5-3j)

#布尔类型+复数

>>> True + (1+2j)

(2+2j)

解释器都会给出正确结果。这是由于Python在对不同类型的对象进行运算时，会强制将 对象的值进行临时类型转换，这些转换遵循如下规律：

•布尔类型在进行算术运算时，将其分别视为数值0和1。

•整型与浮点型运算时，将整型转化为浮点型。

•其他类型与复数运算时，将其他类型转换为复数类型。

简单来说，混合运算中类型相对简单的操作数会被转换为与复杂类型操作数相同的类型。

［泠多学一招：+和大的特殊用法

*'%—*

1. +的特殊用法

在其他语言中，“+”运算符操作的对象只能是数值，但Pythoii中，“+”还能对字符串 进行拼接操作，如对字符串"hello"和字符串"world"进行“+”操作，示例如下：

>>> , hello 1 + ? world?

s hello worlds

由表达式的执行结果可知，以上两个字符串被连接成了一个长字符串hello worldo

1. \*的特殊用法

若要在C语言中打印一个由30个七”组成的分隔线，需要在p『intf()函数中完整地传入 30个七”，示例如下：

printf — — \n")

但在Pythoii中，使用“旳便能简单地实现以上功能，具体示例如下： >>> print(°-"\*30)

由以上示例可知，在Pythoii中，字符串与数值由乘号连接时，Pythoii解释器会将字符串 重复打印多次(次数由数值决定)。

乳&2

赋值运算符的功能是：将一个表达式或对象赋给一个左值，其中左值必须是一个可修改的 值,不能为一个常量。“=”是基本的赋值运算符，此外“=”可与算术运算符组合成复合赋值 运算符。Pythoii中的复合赋值运算符有：+=、-=、\*=、/=、〃=、％=、\*\*=,它们的功能相似， 例如％+=b”等价于%-=b”等价于 京=建-b” *，*诸如此类。

以a = 3, b = 5为例，Python中各个赋值运算符的功能及示例如表3-3所示。.

<3-3赋值运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 说 明 | 示 例 |
| *=* | 等：将右值赋给左值 | a = b, a 为 5 |
| += | 加等：使右值与左值相加，将和赋给左值 | a += b, a 为 8 |
| -= | 减等：使右值与左值相减，将差赋给左值 | a-=b , a 为-2 |
| \*= | 乘等：使右值与左值相乘，将积赋给左值 | a \*= b, a 为］5 |
| /= | 除等：使右值与左值相除，将商赋给左值 | a /= b, a 为 0.6 |
| //= | 整除等：使右值与左值相除，将商的整数部分赋给左值 | a //= b ? a 为 0 |
| %= | 取余等：使右值与左值相除，将余数赋给左值 | a %= b, a 为 2 |
| 脣= | 審等：获取左值的右值次方，将结果赋给左值 | a \*\*= b, a 为 243 |

经以上操作后，左值a发生了改变，但右侧操作数b并没有被修改。以“+=”为例，示例 如下：

>>> a = 3

»> b = 5

»> a += b

>>> print(a)

8

>>> print(b)

5

需要说明的是，与C语言不同，Pythoil中在进行赋值运算时，即便两侧操作数的类型不同 也不会报错，且左值可正确地获取右操作数的值(不会发生截断等现象)，这与Python中变量 定义与赋值的方式有关，这点将在后续小节(3.8节)中着重讲解。

比较运算符同样能够与两个操作数构成一个表达式，这种表达式通常用于布尔测试， 测试的结果只能是True或Falseo比较运算符的操作数可以是表达式或对象,Python中的比 较运算符有：==、！=、>、V、*>=、<=,*以a=3, b=5为例，其功能与相关示例分别如表3・4 所示。

表3-4比较运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 说 明 | 示 例 |
| == | 比较左值和右值，若相同则为真，否则为假 | a = 3, b = 5  a==b不成立,结果为False |
| != | 比较左值和右值，若不相同则为真，否则为假 | a = 3, b = 5  a != b成立，结果为True |
| > | 比较左值和右值，若左值大于右值则为真，否  则为假 | a = 3, b = 5  a>b不成立，结果为False |
| < | 比较左值和右值，若左值小于右值则为真，否 则为假 | a = 3, b = 5  a< b成立，结果为True |
| >= | 比较左值和右值，若左值大于或等于右则为 真，否则为假 | a = 3, b = 5, a>=b不成立,结果*为*False a= 3, b = 3, a>=b 成立,结果为 True |
| v= | 比较左值和右值，若左值小于或等于右值则为 真，否则为假 | a = 3, b = 5, a<=b 成立，结果为 True  a. = 3, b = 3, a<=b 成立，结果为 True |

比较运算符只对操作数进行比较，不会对操作数自身造成影响须即经过比较运算符运算后 的操作数不会被修改。

Python中也支持逻辑运算，但Python中逻辑运算符的功能与其他语言有所不同。Python 中分别使用“or” "and” “not”这三个单词作为逻辑运算“或” “与“非”的运算符，其 中皿与and为双目运算符，not为单目运算符。

逻辑运算符的操作数可以为表达式或对象，下面将对它们的功能分别进行说明。

(1 ) or

若(M运算符左操作数的布尔值为True,则返回左操作数，否则返回右操作数或其计算结 果(若为表达式)，示例如下:

»> 0 or 3 + 5 #左操作数布尔值为False

8

〉〉〉3 or [ ] #左操作数布尔值为True

3 「

1. and

若左操作数的布尔值为False,则返回左操作数或其计算结果(若为表达式)，否则返回 右操作数的执行结果，示例如下：

>>> 3-3 and 5

0

>>> 3-4 and 5

5

1. not

若操作数的布尔值为False则返回True,否则返回False,示例如下：

»> not (3-5)

False

>>> not (False)

True

*3.1*运算綺備先级

Python中支持使用多个不同的运算符连接简单表达式，实现相对复杂的功能，为了避免含 有多个运算符的表达式出现歧义• Python为每种运算符都设定了优先级。Pythg中各种运算符 的优先级由低到高依次如表3-5所示。

|  |  |
| --- | --- |
| 表3-5运算符优先级 | |
| 运算符 | 说 明 |
| or | 布尔“或“ |
| and | 布尔“与” |
| not | 布尔“非” |
| in, not in | 成员测试(字符串、列表、元组、字典中常用) |
| is, is not | 身份测试 |
| <，<=,>5 >=，！=，== | 比较 |
| 1 | 按位或 |
| A | 按位异或 |
| & | 按位与 |
| «9 » | 按位左移、按位右移 |
|  | 加法，减法 |
|  | 乘法、除法，取余 |
| +x, -X | 正负号 |
| *〜* | 按位取反 |
| \*\* | 指数 |

默认情况下，运算符的优先级决定了复杂表达式中的哪个单一表达式先执行，但用户可使 用圆括号“()”改变表达式的执行顺序。通常圆括号中的表达式优先执行，如对于表达式“3+4\*5”， 若想让加法先执行9可写为"(3+4)\*5" °此外9若有多层圆括号，则最内层圆括号中的表达式 先执行。

运算符一般按照自左向右的顺序结合，如在表达式“3+5-4”中，+、-优先级相同，解释 器会先执行“3+5”，再将3+5的执行结果8与操作数4 一起，执行4”，即执行顺序等同 于“(3+5)-4” ；但赋值运算符的结合性为自右向左，如表达式％ = b = c”，Python解释器会 先将c的值赋给b，再将b的值赋给a9即执行顺序等同于“a = (b = c)” o

类型機撅

在程序中，最常对数字进行的操作便是运算。例如，在终端输入如下语句：

»>3 + 5

8

实际就是进行了一次算术运算。

当然，上例的意义并不大，因为这样的程序只能单一地执行两个固定数据的运算。程序中 更常设定的功能，是提示用户进行输入，交互地获取对象的值。示例如下：

num\_01 = input (叩请输入变量 num\_0M “)

num\_02 = input (叫请输入变量 num\_02 ： °)

以上的两条语句可利用input()函数获取两个用户输入，并将获取到的数据分别赋给 和iium\_02。假设用户根据第一条提示输入的数据为“1” *，*根据第二条提示输入的数 据为“5”，那么在终端执行求和语句：

>>> sum = num\_01 + num\_02

>>> print(sum)

将会获取如下结果：

15

这个结果由字符串“1”和字符串％”拼接而来屏若用户本意为计算两个数字类型的和， 那么此结果显然与期望不符。这是因为，在Python 3中，使用input()函数获取到的数据是一个 字符串，即便用户输入的是形如“3” “34' "3+2j” “False”等的数值类型的数据，解释器 也会将其视为字符串。

3B8J襲壟鷹攧圖數

综上所述,若用户想要将从终端获取的数据作为数值类型进行计算，就必须先进行类型转 换。Python中内置了一系列可实现强制类型转换的方法，保证用户在有需求时，可使用iiit()、 float()、complex。和str()函数,将目标内容转换为指定类型。这4个函数的功能分别如表3『6所示。

表3・6类型转换函数

|  |  |
| --- | --- |
| 函数 | . 说 明 |
| int() | 将浮点型、布尔类型和符合数值类型规范的字符串转换为整型 |
| float() | 将整型和符合数值类型规范的字符串转换为浮点型 |
| complex() | 将其他数值类型或符合数值类型规范的字符串转换为复数类型 |
| str() | 将数值类型转换为字符串 |

3.8.2餐醴鵜臉淺意曩囑

在使用以上函数时有两点需要注意：

(1 ) int()函数、Aoat()函数和complex。函数只能转换符合数值类型格式规范的字符串

(2 )使用int()函数将浮点数转换为整数时，若有必要会发生截断(取整)而非四舍五入。 用户在使用以上函数时膏必须考虑到以上两点，否则可能会因字符串不符合要求而导致在 转换时产生错误，或因截断而产生预期之外的计算结果。

3.8^3襲颦鶉麒彖例

»>

3

»>

3。0

»>

int()、float()A complexQ和str()函数的简单用法分别如下:

#浮点型转整型 #小数部分被截断 #整型转浮点型

int (3.6)

float (3)

complex(\* 3+2j')

#字符串转复数

(3+2j)

#字符串转整型

»> demo\_str = 1 345 ?

>>> type(demo\_str)

<class \* str \*>

>>> type(int(demo\_str))

<class \* int\*>

#复数转字符串

>>> str(2+3j) \2 + 3j)'

掌握以上函数后，想对从终端获取的数据，或两个符合数值类型格式的字符串进行算术运 算便十分简单了。例如，要求对两个符合数值类型格式的字符串进行求和运算，示例如下：

»> str\_01 = "2 + 4j”

»> str\_02 = "5+2 j °

»> sum = complex (str\_01) +complex (str\_02) >>> print(sum)

(7+6j)

由执行结果"(7+6j)”可知，字符串strl和str2中存储的字符串成功被转换为复数类型， 并进行了求和计算。

值得一提的是，在经过以上操作后，毗\_01和s、02仍为字符串，这是因为使用complexO 转换的结果只是一个临时对象，并未被存储。如若通过type<)测试str\_Ol str\_02和sum的类型, 获得的结果如下：

>>> type(str\_01)

<type ? str 5 >

>>> type(str\_02) <type \* str'>

>>> type(sum)

<type 1 complex'>

乳凱1对蒙

在Pythoii中定义的数据一般被称为对象(Object) o计算机中的数据是分块存储的，我们 可以把计算机的内存空间视为一个被等分为多个格子的储物柜*，*储物柜中的每个格子顺序编号，

当有对象被定义时，Pythg将对象的数值放入储物柜的某个格子中，并在格子上贴上标签，如 此便完成了对象的定义。

上述过程中，标签可视为“对象名(name)”，存储到格子中的内容视为对象的“值(value)", 而对象所在格子的编号则可视为对象在内存中的地址，Python中将对象的内存地址称为“身份 (id) ”。值和身份是Python中对象的重要特性,此外，对象还有一个重要的特性——类型(type ), 类型决定了对象在“储物柜”中占据“格子”的数量。

Pythoii对象的身份可唯一标识一个变量，任何对象的身份都可使用内建函数id()获取，如 在Python中定义一个整型对象a,并获取其身份，则可使用如下方法：

>>> a = 3

>>> id(a)

8939712

Pythoii对象的身份是只读的，用户不能直接更改对象的身份。Pythoii中部分对象的值也是 不可改变的，值不能被改变的对象称为不可变对象，Python的数值类型就是不可变对象。

读者或许会有疑问，在对一个已经定义好的数值类型对象重新赋值时明明可以操作成功， 如在进行下列操作时并不会报错:

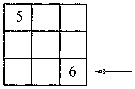
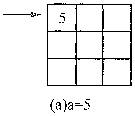
»> a = 5

»> a = 6

>>> print(a)

6

并且，对象芻的值也确实从5更改为6,为什么却说数值类型是不可改变的呢？这与 Python中变量的赋值方式有关。实际上，当进行以上操作时，改变的并非对象a的值，而是对 象a的引用。

Pythoji中的赋值通过引用实现。当用户在定义对象时，解释器将对象的值放入内存 地址，并将该地址的引用赋给对象，经此过程， 对象名便等同于内存地址的别名，用户可通过变 量名获取对象的值。而在对对象进行修改时，解 释器实际上会将新数值放入新内存地址，再将新 内存地址的引用赋给待修改对象，如图320所示。

a

a

(b)a=6

图3」0对象定义与再赋值

由图3-10可知，该对象原地址中存储的数值并没有被修改。下面在Pythoii解释器中进行 验证：

>>> a = 5

#定义对象a

#查看对象身份

#重新赋值

#查看对象身份

»> id (a) 8939776

»> a = 6 »> id (a)

8939808

观察以上各表达式的执行结果，可知重新赋值后对象的身份发生了改变。

Python中的身份运算符为：is和is not,用于判断两个对象是否相同。Pythoii中对象的唯一 标识即身份,因此身份运算符的运算过程即对象身份的比较过程。身份运算符的功能如表3刀所示o

\*3-7身份运算符

|  |  |
| --- | --- |
| 函数 | 说 明 |
| ' is | 测试两个对象是否相同，相同返回True,否则返回False |
| is not | 测试两个对象是否不同，不同返回True,否则返回False |

下面通过示例来展示身份运算符的用法。

|  |  |
| --- | --- |
| »> a = 5  »> b = a  >>> a is b  True  >>> a is not b False  >>> c = 6  »> b = c  >>> a is b | #定义对象a  #使用a定义对象b, b获取a的引用  #判断a是否与b相同  #判断a是否与b不同  #定义对象c  #使用对象c为b赋值 |
| False | #对象b的身份被修改，与对象a不再相同 |

>>> b is c

True

#对象b的身份与对象c的身份相同