第3章 chapter 3

基本数据类型

2.3节已经简单介绍了Python数据类型，本章将详细介绍基本数据类型的使用和操作方法。

3.1 数值类型

表示数值大小、可以进行数值运算的数据称为数值类型数据。现实中的很多数据都是数值型的，如年龄18岁、成绩100分、利率0.035、月收入8000元等。Python提供了3种数值类型：整数、浮点数和复数，分别对应数学中的整数、实数、复数。

3.1.1 整数类型、浮点数类型和复数类型

1. 整数类型

Python中的整数类型为int型，可以用十进制、二进制、八进制和十六进制表示，默认用十进制表示，若用其他进制，则需要增加引导符号。二进制以0b或者0B引导，八进制以0o或者0O引导，十六进制以0x或者0X引导。

例如：

```
>>>110
110
>>>0b1101110
110
>>>0o156
110
>>>0x6E
110
>>>type(0x6E)
<class 'int'>
```

理论上，整数类型在Python中是没有取值范围限制的，它只受运行Python程序的计算机内存大小的限制。

2. 浮点数类型

Python 中的浮点数类型为 float 型，表示带有小数部分的数值，小数部分可以是 0。浮点数可以用十进制表示，也可以用科学记数法表示。

例如：

```
>>>0.0
0.0
>>>-17.0
-17.0
>>>9.6E5
960000.0
>>>3.14e-4
0.000314
>>>type(3.14e-4)
<class 'float'>
```

3. 复数类型

Python 中的复数类型为 complex 型，一般形式为 a＋bj 或者 a＋bJ，a 为实数部分，b 为虚数部分，a 和 b 都是浮点数类型，例如 2＋3j、0.5j、2＋0j、1.2e-5＋1.2e10j 都是 Python 中的复数类型。

复数的实部和虚部都是以复数的属性存在的，这两个属性分别为 real 和 imag，例如：

```
>>>z = -3 + 4.5j
>>>type(z)
<class 'complex'>
>>>z.real                                    #获得 z 的实部
-3.0
>>>z.imag                                    #获得 z 的虚部
4.5
```

3.1.2 数值运算符

数值类型的数据可以参与数值型运算，需要使用 Python 算术运算符，如表 3-1 所示。

表 3-1 算术运算符

运 算 符	功 能	示 例
x＋y	x 与 y 的和	32＋5 的结果为 37
x－y	x 与 y 的差	32－5 的结果为 27
x * y	x 与 y 的积	32 * 5 的结果为 160

续表

运　算　符	功　　能	示　　例
x/y	x 与 y 的浮点除法	30/5 的结果为 6.0
x//y	x 与 y 的整数商，不大于 x 与 y 的商的最大整数	32//5 的结果为 6
x**y	x 的 y 次幂	4**0.5 的结果为 2.0
x%y	x 与 y 的商取余数，模运算	32%5 的结果为 2

其中，运算符“/”执行的是浮点除法，其结果为浮点数。即使是两个整数进行浮点除法，也会产生一个浮点数结果，例如 30/5 的结果为 6.0。

运算符“//”执行的是整数除法，会对结果取整，小数部分会被直接舍掉，例如 32//5 的结果为 6。

运算符“**”执行的是幂运算，即 x ** y 相当于 x^y，例如 4 ** 2 的结果为 16。

运算符“%”执行的是取余或者取模运算，即进行除法取余数，例如 7%3 的结果为 1；32%5 的结果为 2。可以用取模运算判断整除，例如下面是判断一个数是否是偶数的代码。

```
>>>x=16
>>>if x %2 ==0:
        print("{}是偶数".format(x))

16是偶数
```

【例 3.1】 用户输入以分为单位的总金额，程序将其转换为元、角、分的表示形式并输出。

实现这个简单的转换器需要通过以下步骤。

① 提示用户输入一个整数，即以分为单位的总金额。

② 将分除以 100，其商即为转换的元，余数即剩余的分。

③ 将剩余的分除以 10，其商即为转换的角，余数即剩余的分。

```
n = eval(input("请输入一个整数,例如 3145,代表 3145 分钱:"))
remaining_fen = n                          #n 保存用户输入的分,remaining_fen 存储变化的分

yuan = remaining_fen // 100                #提取分中包含的最多的 yuan
remaining_fen = remaining_fen %100         #提取 yuan 后剩余的分

jiao = remaining_fen // 10                 #提取剩余分中包含的最多的 jiao
remaining_fen = remaining_fen %10          #提取 jiao 后剩余的分

fen = remaining_fen

print("{}分相当于{}元{}角{}分!".format(n, yuan, jiao, fen))
```

程序运行结果如下所示。

```
请输入一个整数,例如 3145,代表 3145 分钱:12345
12345 分相当于 123 元 4 角 5 分!
```

3.1.3 增强赋值运算符

3.1.2 节的运算符可以和赋值运算符(=)结合构成增强赋值运算符,如表 3-2 所示。

表 3-2 增强赋值运算符

增强赋值运算符	示 例	等 价 运 算
+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
//=	x //= 2	x = x // 2
**=	x **= 2	x = x ** 2
%=	x %= 2	x = x % 2

注意:使用增强运算符时,运算符和赋值运算符(=)之间不可以有空格。

3.1.4 数值运算函数

函数是完成特定任务的一组语句的集合。Python 语言提供了一个内置函数库,我们已经用过了 input()、print()、eval()函数,使用这些内置函数不需要导入任何模块,直接调用即可。

对于数值类型的数据,内置函数库中也有处理这些数据的函数,如表 3-3 所示。

表 3-3 内置数值运算函数

函 数	功 能
abs(x)	返回 x 的绝对值
divmod(x, y)	返回元组(x//y, x%y)
max(x1, x2, …)	返回 x1, x2, …中的最大值
min(x1, x2,…)	返回 x1, x2, …中的最小值
pow(x, y)	返回 x 的 y 次方的值,等价于 x**y
round(x)	返回最接近 x 的整数,如果 x 与两个整数同等接近,则返回偶数
round(x, n)	返回 x 保留 n 位小数后的浮点数
sum(x1,x2,…)	返回 x1,x2,…的和

以下是使用这些内置函数的示例。

```
>>>abs(-3.14)
3.14
>>>divmod(10, 3)
(3, 1)
>>>max(3,3.14,2,-5)
3.14
>>>min(3,3.14,2,-5)
-5
>>>pow(2,-3)
0.125
>>>pow(0.2,3.3)
0.004936270901760079
>>>round(3.5)                    #若与两个整数接近程度相同,则返回偶数
4
>>>round(-3.1)                   #返回最接近的整数
-3
>>>round(3.14159,2)              #保留两位小数
3.14
>>>
```

对于数字类型的数据,Python 还提供了内置的类型转换函数,如表 3-4 所示。

表 3-4 内置类型转换函数

函　数	功　能
int(x)	将 x 转换为整数,x 可以是浮点数(不进行四舍五入)或者数字字符串
float(x)	将 x 转换为浮点数,x 可以是整数或者数字字符串
complex(re[,im])	返回一个复数,实部为 re,虚部为 im

例如:

```
>>>int(3.6)
3
>>>int('1234')
1234
>>>float(3)
3.0
>>>float("3")
3.0
>>>complex(3,-5)
(3-5j)
>>>int("345ab")                  #报错,参数字符串不是数字字符串,不能转换成整数
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    int('345ab')
ValueError: invalid literal for int() with base 10: '345ab'
```

3.2 字符串类型

字符串类型数据表示文本信息，例如昵称为“Tom”、学号为“201901234”、性别为“女”等。Python 中的字符串(str)类型数据是一串由字符组成的序列，可以包含字母字符、数字字符、汉字、特殊字符等。

3.2.1 字符串与字符串运算符

字符串是由一对单引号(')或者一对双引号(")括起来的字符序列。使用单引号时，双引号可以作为字符串的一部分；使用双引号时，单引号可以作为字符串的一部分。也可以用成对的 3 个单引号(''')或者成对的 3 个双引号(""")括起单行或者多行字符串。我们把这些成对的单引号、双引号以及成对的 3 个单引号、成对的 3 个双引号称为字符串的定界符。例如：

```
>>>type("Good")
<class 'str'>
>>>print('A')
A
>>>print('''I
love
China!''')                          #由成对的 3 个单引号括起来的多行字符组成的字符串
I
love
China!
>>>print("I'm a student. ")          #字符串中包括单引号,定界符就需要用双引号
I'm a student
>>>print("""I'm a student. """)      #由成对的 3 个双引号括起来的字符组成的字符串
I'm a student
```

程序设计中经常会用到一些特殊字符，例如 print()方法中输出的字符串中间需要换行，换行符就是一个特殊的不可打印字符，这些特殊字符用转义字符表示。Python 字符串中可以包含转义字符，这些转义字符是由一个反斜杠(\)引导的，如'\n'代表换行，'\t'表示制表符，"\'"代表单引号，'\"'代表双引号，'\r'代表光标移到本行行首位置等。因为反斜杠(\)是转义字符的引导符号，所以字符串中用“\\”代表反斜杠本身。

例如：

```
>>>print("Python\\"+'\n'+"语言"+'\t'+"设计基础")
Python\
语言    设计基础
```

Python 提供了字符串的基本运算符，如表 3-5 所示。

表 3-5 字符串运算符

运算符	功 能	示 例
x + y	x 与 y 首尾连接	"Pyth" + "on"的结果为 "Python"
x * n	复制 n 次字符串,和 n * x 等价	"Py" * 3 的结果为 "PyPyPy"
x in s	x 是 s 的子串,返回 True,否则返回 False	"th" in "Python"的结果为 True
x not in s	x 不是 s 的子串,返回 True,否则返回 False	"th" not in "Python"的结果为 False

True 和 False 是 Python 的布尔类型常量,True 代表真,即成立;False 代表假,即不成立。本书将在 3.4 节详细介绍这类数据类型。

3.2.2 字符串索引与切片

字符串是字符的有序序列,是 Python 组合数据类型中的一种序列类型(组合数据类型将在第 6 章详细介绍)。每个字符在字符串中都有自己的位置序号,称为索引。字符串有两种索引方式:正向从 0 递增索引和反向从−1 递减索引,如图 3-1 所示。

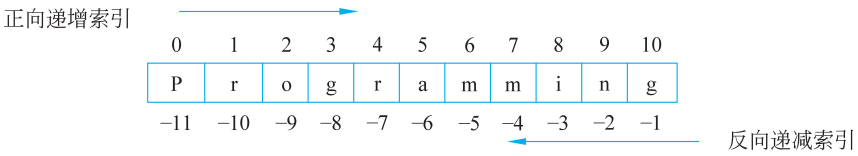

图 3-1 字符串索引

可以用 s[index]引用字符串中 index 索引位置的字符,例如:

```
>>>s = "Programming"
>>>s[0]
'P'
>>>s[-1]
'g'
>>>print(s[4],s[-7])
r r
>>>s[11]                          #索引超出范围,报错
Traceback (most recent call last):
  File "<ipython-input-2-665bb6993e1f>", line 1, in <module>
    s[11]
IndexError: string index out of range
```

注意:字符串是不可变的,不能像 s[2]= 'a'这样对某一索引位置进行赋值以改变字符串的内容,但是可以建立一个新的字符串对已存在的变量赋值,如对以上的 s 进行重新赋值,如下所示。

```
>>>s[2] = 'A'                     #出错
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    s[2]='A'
```

```
5   TypeError: 'str' object does not support item assignment
6   >>>s = s + '!'                    #用 s + '!'的结果对 s 重新赋值,这时 s 对象已经变成新的对象
7   >>>print(s)
8   Programming!
```

切片操作是 Python 序列的一种重要操作,字符串可以利用切片操作截取子字符串,语法如下。

```
s[M:N]
```

可以截取 s 字符串索引位置 M 到 N－1 的一个子串。例如:

```
1   >>>s = "Programming"
2   >>>s[1:4]
3   'rog'
```

M 或 N 是可以省略的,如果 M 被省略,则起始索引为 0;如果 N 被省略,则结束索引为最后的位置;如果两个都省略,则返回字符串本身。例如:

```
1   >>>s = "Programming"
2   >>>s[:4]
3   'Prog'
4   >>>s[8:]
5   'ing'
6   >>>s[:]
7   'Programming'
```

M 和 N 既可以同时用正索引或者负索引,也可以两者混用。例如:

```
1   >>>s = "Programming"
2   >>>s[-3:]
3   'ing'
4   >>>s[-3:-1]
5   'in'
6   >>>s[8:-1]
7   'in'
```

切片操作还有以下的用法。

```
s[M:N:step]
```

这种切片从 s 字符串索引 M 到 N－1,每 step 个字符提取一个以构成子串。如果 step 是正值,则从左向右提取;如果 step 是负值,则从右向左提取。

例如:

```
1   >>>s = "Programming"
2   >>>s[::2]                         #从第 0 个开始,每两个字符取一个,也就是取偶数索引位的字符
3   'Pormig'
4   >>>s[1::2]                        #从第 1 个开始,每两个字符取一个,也就是取奇数索引位的字符
```

```
5   'rgamn'
6   >>>s[::-1]                          #step 为-1,将字符串从右到左每个字符取一个
7                                       #所以实现的是字符串从头到尾的翻转
8   'gnimmargorP'
```

在进行切片操作时,M 或者 N 的取值超出了字符串的索引范围并不会报错,而是按照情况自动取到字符串的最左端或者最右端;或者当 M～N－1 的切片方向与 step 指定的方向不同时,返回空字符串。例如:

```
1   >>>s = "Programming"
2   >>>s[:100]
3   'Programming'
4   >>>s[-100:1]
5   'P'
6   >>>s[-1:1]                          #切片方向和 step 指定的方向不同,所以取不到,返回空字符串
7   ''
8   >>>s[-1:1:-1]
9   'gnimmargo'
```

3.2.3 字符串的遍历操作

在 Python 中,可以用 for 循环对字符串中的所有字符进行顺序遍历操作。例如,以下代码可以显示输出字符串 s 中的所有字符,这个 for 循环在没有使用索引的情况下也方便地访问了所有字符,并在每个字符的后面加了一个空格之后同行输出。

```
1   >>>s=" Python"
2   >>>for ch in s :
3       print(ch,end=" ")               #print()方法的 end 参数可以实现不换行输出
4           #end=" "为输出的 ch 值后面加上" ",并且输出结束后不换行
5   P y t h o n
```

如果想用更灵活的方式遍历字符串中的部分字符,则需要借助索引进行访问。例如,下列代码就实现了在字符串中每隔一个字符取一个字符并输出显示的功能。

```
1   >>>s = "Python"
2   >>>for i in range(0, len(s), 2):
3       print(s[i],end = " ")
4
5   P t o
```

代码第 2 行中的 len()方法可以返回 s 的长度,range(0, len(s), 2)函数在 0～6(达不到 6)中每两个数取一个,所以返回 0、2、4 这三个数字,i 即在这三个数字中遍历,依次取到 s[0]、s[2]、s[4]并显示输出。遍历循环和 range()函数将在第 4 章详细介绍。

3.2.4 字符串处理函数

Python 提供了处理字符串的内置函数,如表 3-6 所示。

表 3-6 内置字符串函数

函数	功 能	示 例
ord(s)	返回单个字符 s 对应的 Unicode 编码值对应的十进制值	ord('a') 的结果为 97；ord('中') 结果为 20013
chr(n)	返回 Unicode 编码值 n(十进制值)对应的单个字符	chr(97)的结果为'a',chr(20013)的结果为'中'
str(x)	返回数值 x 对应的字符串形式	str(1.23)的结果为'1.23'
len(s)	返回字符串 s 的长度,即包含字符的个数	len('I love 中国！')的结果为 10

字符在计算机内部是用二进制存储的,这种字符和二进制之间的映射关系称为字符编码。编码方式有很多,英文字符最通用的编码方式是 ASCII(美国信息交换标准代码)编码,它用 7 位二进制数对 128 个英文字符进行编码,每个字符和对应的 ASCII 码值形成一个 ASCII 编码表。有关 ASCII 编码表,读者可以查阅相关资料。

Python 3 支持字符的 Unicode 编码。Unicode 编码把所有语言文字都统一到了一套编码表中,也称为统一码。一个 Unicode 编码值可以以“\u”开始,用 4 位十六进制数字表示,范围从“\u0000”到“\uFFFF”。ASCII 编码表只能表示英文字符,它是 Unicode 编码表的一个子集。例如,a 的 ASCII 编码和 Unicode 编码对应的十进制都是 97。例如:

```
>>>print("\u4E2D\u56FD")          #十六进制编码 4E2D 对应的字符为"中"
中国
>>>print("\u0061")                #十六进制编码 0061 相当于十进制的 97,对应的字符为"a"
a
```

3.2.5 字符串对象

在 Python 中,所有数据实际上都是对象。一个数字、一个字符串都是一个对象,不同的数字是不同的数字类型对象,不同的字符串是不同的字符串类型对象。

在程序运行时,系统会自动给每个对象分配一个独特的整数,这个整数类似对象的唯一代号(id),在程序运行过程中不会改变。但是每次程序运行时,系统都会重新分配 id。可以用 id()函数返回这个整数,也可以用 type()函数返回对象的类型。

Python 中的变量实际上是一个对象的引用,如 n=3,严格来说,运行这条语句时系统会创建一个 int 类型的对象 3,n 实际上是一个引用这个 int 类型对象 3 的变量。通常可以简化描述为:n 是一个值为 3 的整型变量。

例如:

```
>>>n = 3                         #为 3 创建一个 int 型对象,n 为引用这个对象的变量
>>>id(3)                         #对象 3 的 id 值
1405142768
>>>id(n)                         #变量 n 是对于对象 3 的引用,所以 n 的 id 值就是 3 的 id 值
1405142768
>>>type(n)                       #对象的类型是 int 型
<class 'int'>
```

```
>>>s = "Python"                #为"Python"创建一个 str 类型的对象,s 为引用这个对象的变量
>>>id(s)                       #对象"Python"的 id 值
2135079873144
>>>type(s)                     #对象的类型是 str 型
<class 'str'>
```

在 Python 中,对象的类型是由对象的类(class)决定的,class 是 Python 面向对象编程中的术语,将在第 8 章进行介绍。

特定类型的对象上有特定的操作,这些操作称为对象的方法,是由函数定义的。可以调用这些方法以操作对象,调用格式为

```
对象.方法名(参数)
```

例如,字符串类型对象有 upper()方法和 lower()方法,下列代码便调用了这些方法。

```
>>>s="Python"
>>>s.upper()                   #把 s 中的所有字母转换成大写,返回这个新的大写字符串
'PYTHON'
>>>s.lower()                   #把 s 中的所有字母转换成小写,返回这个新的小写字符串
'python'
>>>s                           #对象 s 本身不变
'Python'
```

3.2.6 字符串处理方法

Python 字符串类型除了以上两个方法,还有以下常用方法。

1. 判断字符串中的字符

如表 3-7 所示,这些字符串类型的方法可以判断字符串的大小写、是否是数字字符串、是否是空格字符串、是否是字母字符串等。

表 3-7 字符串对象的方法表(1)

方法	功 能	示 例
islower()	若字符串中所有字符都是小写,则返回 True,否则返回 False	"Python".islower()的结果为 False "python".islower()的结果为 True
isupper()	若字符串中所有字符都是大写,则返回 True,否则返回 False	"Python".isupper()的结果为 False "PYTHON".isupper()的结果为 True
isdigit()	若字符串中所有字符都是数字,则返回 True,否则返回 False	"123".isdigit()的结果为 True "P123".isdigit()的结果为 False
isspace()	若字符串中所有字符都是空格,则返回 True,否则返回 False	" ".isspace()的结果为 True " 1 2 3" .isspace()的结果为 False
isalpha()	若字符串中所有字符都是字母,则返回 True,否则返回 False	"aba".isalpha()的结果为 True "ab123".isalpha()的结果为 False

2. 搜索和处理字符串子串

搜索和处理字符串子串的方法如表 3-8 所示。

表 3-8 字符串对象的方法表(2)

方 法	功 能
endswith(s0[,start[,end]])	判断字符串[start: end]切片部分是否以 s0 结尾,如果没有指定 start 和 end,则在整个字符串中判断;如果只指定 start 而没有指定 end,则从字符串 start 位置开始到字符串结束的部分进行判断
startswith(s0[,start[,end]])	判断字符串[start: end]切片部分是否以 s0 开始,如果没有指定 start 和 end,则在整个字符串中判断;如果只指定 start 而没有指定 end,则从字符串 start 位置开始到字符串结束的部分进行判断
find(s0)	返回 s0 在字符串中第一次出现的索引位置,如果不存在 s0,则返回 -1
rfind(s0)	返回 s0 在字符串中最后一次出现的索引位置,如果不存在 s0,则返回 -1
count(s0)	返回 s0 在字符串中出现的次数,如果不存在 s0,则返回 0
replace(old,new[,count])	返回字符串中所有 old 子串全部替换为 new 子串之后形成的新字符串,如果找不到 old 子串,则返回原字符串。如果指定 count,则前 count 个 old 子串被替换

以下是调用这些方法操作字符串的一些示例。

```
>>>s = "Python Programming"
>>>s.endswith('on')
False
>>>s.endswith('on',0,6)     #在 s[0:6]中判断是否以'on'结尾
True
>>>s.find('o')              #'o'在 s 中出现的第一个位置的正向索引值 4
4
>>>s.rfind('o')             #'o'在 s 中出现的最后一个位置的正向索引值 9
9
>>>s.find('p')              #在 s 中找不到'p'
-1
>>>s1 = s.replace('o','O')  #将 s 中的'o'全部替换成'O',返回一个新字符串对象并赋给 s1
>>>s
'Python Programming'
>>>s1
'PythOn PrOgramming'
```

3. 删除字符串两端的空白字符

删除字符串两端空白字符的方法如表 3-9 所示。

表 3-9 字符串对象的方法表(3)

方 法	功 能
lstrip()	返回删除字符串左端空白字符的新字符串
rstrip()	返回删除字符串右端空白字符的新字符串
strip()	返回删除字符串两端空白字符的新字符串

' '、'\t'、'\n'都是这里所说的空白字符,而且这几个方法只能删除左右两端的空白字符,对字符串内部的空白字符没有作用。以下是调用这些方法操作字符串的一些示例。

```
>>>s = '  Python Programming\n   '
>>>s1 = s.lstrip()          #删除 s 左端的空白字符,返回一个新的字符串并赋给 s1
>>>s1
'Python Programming\n   '
>>>s2 = s1.rstrip()         #删除 s1 右端的空白字符,返回一个新的字符串并赋给 s2
>>>s2
'Python Programming'
>>>s3 = s.strip()           #删除 s 两端的空白字符,返回一个新的字符串并赋给 s3
>>>s3
'Python Programming'
```

4. 格式化字符串

格式化字符串的方法如表 3-10 所示。

表 3-10 字符串对象的方法表(4)

方 法	功 能
center(width[,char])	返回长度为 width 的字符串,调用该方法的字符串位于新字符串的中心位置,两端新增位置用 char 填充,不指定 char 则用空格填充
ljust(width[,char])	返回长度为 width 的字符串,调用该方法的字符串位于新字符串的最左端,后面新增位置用 char 填充,不指定 char 则用空格填充
rjust(width[,char])	返回长度为 width 的字符串,调用该方法的字符串位于新字符串的最右端,前面新增位置用 char 填充,不指定 char 则用空格填充
format()	返回字符串的特定格式化结果,详见 3.2.7 节的讲解

以下是调用这些方法操作字符串的一些示例。

```
>>>s = "Python"
>>>s1 = s.center(20)
>>>s1
'       Python       '
>>>s2 = s.center(20,'*')
>>>s2
```

```
'*******Python*******'
>>>s3 = s.ljust(20,'*')
>>>s3
'Python**************'
>>>s4 = s.rjust(20)
>>>s4
'              Python'
```

5. 其他常用的字符串类型方法

字符串类型的 split([sep])方法可以用字符串中的 sep 作为分隔符，将字符串分隔为多个子字符串，每个子字符串作为一个元素形成一个列表对象。如果没有指定 sep，则以空格作为分隔符。列表是一种组合数据类型，将在后续章节中详细讲解。例如：

```
>>>s = input("请输入 3 个字符串,每个字符串以空格作为分隔:")
请输入 3 个字符串,每个字符串以空格作为分隔:Tom Amy Billy
>>>s
'Tom Amy Billy'
>>>s.split()                    #字符串以空格作为分隔被分成了 3 个字符串,组成了一个列表
['Tom', 'Amy', 'Billy']
```

字符串类型的 join(seq)方法可以将组合数据类型对象 seq 中的每个元素用字符串连接起来，形成一个新的字符串对象。这个组合数据对象也可以是字符串，例如：

```
>>>s='124'
>>>'-'.join(s)                   #用'-'将字符串中的每个字符连接起来,返回一个新的字符串对象
'1-2-4'
```

【例 3.2】 接收用户输入的一行字符串，统计其中字母、数字以及空格的个数。

```
c,n,b = 0,0,0                   #c 统计字母个数,n 统计数字个数,b 统计空格个数
strs = input("请随意输入一行字符:")

for s in strs:
    if s.isdigit() :
        n += 1
    if s.isalpha():
        c += 1
    if s.isspace():
        b += 1
print("这行字符串里包含{0}个字母,{1}个数字,{2}个空格".format(c, n, b))
```

程序运行结果如下。

```
请随意输入一行字符:lkasjf837dkdf8 b  sd8f83d
这行字符串里包含 15 个字母,7 个数字,3 个空格
```

注意：因为字符串是不可变对象，所以以上各表中的字符串对象方法均不改变原字符串对象的值，而是返回一个新的字符串。

3.2.7 字符串格式化

字符串对象的 format()方法功能强大，它可以输出格式化的字符串，使用起来非常灵活，既可以将数值型数据处理成格式化的字符串，也可以对字符串进行格式化输出。format()方法的基本格式为

```
<模式字符串>.format(参数列表)
```

例如以下代码。

```
>>>interest =333.33333333333337
>>>print("本月应付利息为:{:.2f}".format(interest))
本月应付利息为:333.33
```

代码中的“"本月应付利息为：{：.2f}"”就是模式字符串，模式字符串除了要输出字符串内容之外，最重要的就是其中的花括号部分，这部分是为 format()方法的参数占位使用的。运行 format()方法时，参数列表中参数的值按参数位置关系替换模式字符串中的相应“{}”部分，同时按照“{}”中的格式规定对参数的值进行格式化。“{}”的内部样式为

```
{<参数位置序号>：<格式规定标记>}
```

语句“print("本月应付利息为：{：.2f}".format(interest))”中，“"本月应付利息为：{：.2f}"”为模式字符串，其中{：.2f}部分＜参数位置序号＞被省略了，因为之后的 format()方法的参数只有一个 interest，所以＜参数位置序号＞省略即对应着 interest，＜格式规定标记＞是.2f，代表浮点型取 2 位小数，按这个格式规定将 interest 值格式化为 333.33，并用这个格式化结果替换模式字符串中的相应“{}”部分，所以最后 print()的输出为

```
本月应付利息为:333.33
```

将以上 print 语句修改成以下形式，进一步观察 format()方法的使用。

```
>>>n = 5
>>>interest = 333.33333333333337
>>>print("{0}月应付利息为:{1:.2f}".format(n, interest))
5月应付利息为:333.33
>>>print("{}月应付利息为:{:.2f}".format(n, interest))
5月应付利息为:333.33
>>>print("{1}月应付利息为:{0:.2f}".format(interest,n))
5月应付利息为:333.33
```

第 3 行语句中的 format()增加了一个参数 n，n 是 format()的第 0 个位置的参数，对

应模式字符串中的“{0}”；interest 是 format()的第一个位置的参数，对应“{1:.2f}”。其中，“{}”中的 0、1 就是＜参数位置序号＞。n 对应的“{0}”中没有规定任何格式，所以省略了冒号，并原样输出了 n。

第 5 行语句将“{}”中的 0、1 均省略，这时按照参数顺序依次进行替换，n 对应“{}”，interest 对应“{:.2f}”，所以输出结果相同。

第 7 行语句将 format()方法的两个参数交换了位置，输出结果依然相同，这是因为＜参数位置序号＞也进行了相应调整，这时“{1}”对应的是参数 n，“{0:.2f}”对应的是参数 interest。

format()方法的＜格式规定标记＞有一些选项，如表 3-11 所示。

表 3-11 format()方法的格式规定标记

标 记	功 能
＜填充＞	用于填充的单个字符
＜对齐＞	＜代表左对齐
	＞代表右对齐
	^代表居中对齐
＜宽度＞	整数值，代表输出宽度
＜精度＞	字符串的最大输出长度或者浮点数小数部分的精度
＜数据类型＞	b,d,o,x 分别代表整数的二、十、八、十六进制形式
	c 代表整数的 Unicode 字符
	e,E 代表浮点数的科学记数法形式
	f 是标准浮点数
	%代表浮点数的百分数形式

以上所有选项都是可选的，不同组的标记也可以自由组合。表 3-12 给出了一些使用实例。注意：输出是字符串形式。

表 3-12 format()方法的使用实例

变 量	格 式 化	输 出	描 述
x = 3.1415926	'{:.0f}'.format(x)	'3'	输出不带小数位
x = 3.1415926	'{:.2f}'.format(x)	'3.14'	保留小数点后两位
x = 3.1415926	'{:+.2f}'.format(x)	'+3.14'	带符号保留小数点后两位
x = 3	'{:0>2d}'.format(x)	'03'	数字左填充 0，长度为 2
x = 3	'{:0<4d}'.format(x)	'3000'	数字右填充 0，长度为 4
x = 1000000	'{:,}'.format(x)	'1,000,000'	用逗号分隔的数字格式
x = 0.25	'{:.2%}'.format(x)	'25.00'	百分比格式

续表

变　量	格　式　化	输　出	描　述
x = 3	'{:>5d}'.format(x)	' 3'	右对齐,长度为 5,用空格补充
x = 3	'{:<5d}'.format(x)	'3 '	左对齐,长度为 5,用空格补充
x = 3	'{:^5d}'.format(x)	' 3 '	居中对齐,长度为 5,用空格补充
x = 11	'{:b}'.format(x) '{:d}'.format(x) '{:o}'.format(x) '{:x}'.format(x) '{:#x}'.format(x) '{:#X}'.format(x)	'1011' '11' '13' 'b' '0xb' '0Xb'	依格式返回不同进制的值。b 为二进制,d 为十进制,o 为八进制,x 为十六进制

3.3　布尔型数据

前面已经提到过 Python 的布尔型数据也称逻辑型数据,即 bool 型。布尔型常量只有两个取值:True 和 False,分别表示真和假。在计算机内部,Python 使用 1 表示 True,使用 0 表示 False。

3.3.1　布尔型常量

布尔型常量与整型常量之间是可以相互转换的,int(True)的值为 1,int(False)的值为 0。也可以将数值转换为布尔型,这时遵循非 0 即真的原则,bool(0)的值为 False,其余的数值转换为布尔值后都是 True。

3.3.2　比较运算符

比较数值大小是数值类型数据的重要操作,Python 提供了比较运算符(也称关系运算符),比较运算的结果是布尔值,如表 3-13 所示。

表 3-13　比较运算符

运算符	功　能	示　例
<	小于	3 < 5 的结果为 True
<=	小于或等于	3 <= 5 的结果为 True
>	大于	3 > 5 的结果为 False
>=	大于或等于	3 >= 5 的结果为 False
==	等于	3 == 5 的结果为 False
!=	不等于	3 != 5 的结果为 True

比较相等在 Python 中是两个等号(==),单个等号(=)是赋值号。

注意:因为对于浮点数无法进行高精度运算,所以应当避免对浮点数直接进行相等比较,而是以两个浮点数之差的绝对值足够小作为判断两个浮点数是否相等的依据。例如:

```
>>>0.4+0.3
0.7
>>>0.4-0.3                    #两个浮点数相减
0.10000000000000003
>>>0.4-0.3==0.1               #直接比较两个浮点数是否相等
False
>>>abs((0.4-0.3)-0.1)<1e-8    #如果两个浮点数之差的绝对值足够小,则认为它们相等
True
```

字符型数据也可以进行比较运算。当两个字符相比较时,从最左端开始逐对字符进行比较。如果此对字符相等,则右移一对继续比较,直到比较出第一对不同的字符,它们的大小决定了两个字符串的大小,后面的字符则不再比较。单个字符之间的比较是比较它们的 Unicode 编码值的大小。例如:

```
>>>"abc" <"abdef"          #第 3 对字符'c'和'd'的大小决定了两个字符串的大小
True
>>>"abc" <"abcde"
                           #如果一个字符串与另一个字符串的左端字符相同,则长字符串比较大
True
>>>"中" <"船"              #汉字比较也采用相同的规则
True
>>>ord("中")               #"中"的 Unicode 码值
20013
>>>ord("船")               #"船"的 Unicode 码值
33337
```

3.3.3 逻辑运算符

布尔值可以进行逻辑运算,Python 的逻辑运算符有 and、or、not,分别代表逻辑与、逻辑或、逻辑非,它们通常用来连接比较表达式或者逻辑表达式,以构成更复杂的逻辑表达式。表 3-14、表 3-15、表 3-16 分别是逻辑与、逻辑或和逻辑非的运算真值表。

表 3-14 and 运算的真值表

x	y	x and y
True	True	True
True	False	False
False	True	False
False	False	False

表 3-15 or 运算的真值表

x	y	x or y
True	True	True
True	False	True
False	True	True
False	False	False

表 3-16 not 运算的真值表

x	not x
True	False
False	True

例如：

```
>>>n = 2001
>>>print(n %4 ==0 or n %3 !=0)          #如果 n 能被 4 整除或者不能被 3 整除,则结果为 True
False
>>>print(n %4 ==0 and n %3 !=0)         #如果 n 能被 4 整除同时不能被 3 整除,则结果为 True
False
>>>print(not 0)                         #0 代表 False,not 0 取反得 True
True
>>>print(not True)                      #not True,对 True 取反得 False
False
```

比较运算符和逻辑运算符是构成条件表达式的重要组成部分。在程序的选择结构和循环结构中,条件表达式又是必不可少的,这两种结构将在第 4 章详细讲解。下面介绍两个运用比较运算符和逻辑运算符组成条件表达式的简单例题。

【例 3.3】 用以下方法实现例 3.2 的要求,其中用到了字符之间的比较运算以及逻辑运算。

```
c,n,b = 0,0,0                          #c 存储字符个数,n 存储数字个数,b 存储空格个数
strs = input("请随意输入一行字符:")

for s in strs:
    if 'a' <=s <='z' or 'A' <=s <='Z':  #如果 s 是大写字母或者小写字母
        c += 1
    if '0' <=s <='9':                   #如果 s 是数字字符
        n += 1
    if ' ' ==s:                         #如果 s 是空格
        b += 1
print("这行字符串里包含{0}个字母,{1}个数字,{2}个空格".format(c, n, b))
```

【例 3.4】 判定闰年：如果一个年份能被 4 整除但不能被 100 整除,或者这个年份能被 400 整除,则这个年份就是闰年。

分析：

用 year 变量存储年份,判断能被 4 整除就可以写成表达式 year % 4 == 0。

判断不能被 100 整除可以写成表达式 year % 100 != 0。

判断能被 400 整除可以写成表达式 year % 400 == 0。

组合成判断条件,即(year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)。

代码如下：

```
year = eval(input("请输入年份: "))
#定义一个闰年标志变量
isLeapYear = (year %4 ==0 and year %100 !=0) or (year %400 ==0)

if isLeapYear:                          #如果 isLeapYear 为 True,则为闰年,否则不是闰年
    print(year, "年是闰年")
else:
    print(year, "年不是闰年")
```

如果用户输入 2021，则程序运行结果为

```
请输入年份：2021
2021 年不是闰年
```

如果用户输入 2020，则程序运行结果为

```
请输入年份：2020
2020 年是闰年
```

3.4 运算符的优先级

Python 表达式中的各种运算符是有运算顺序的，这种运算的先后顺序称为运算的优先级。Python 中的各种运算符的优先级如表 3-17 所示。表 3-17 中的运算符从上到下的优先级逐级降低，同行中的运算符优先级相同。

表 3-17 运算符优先级

优 先 级	运 算 符
↓	**
	+、-（正负号）
	not
	*、/、//、%
	+、-（加、减运算）
	<、<=、>、>=
	==、!=
	and
	or

计算一个表达式时，总是先运算优先级较高的运算符，如果运算符的优先级相同，则按照从左至右的顺序依次进行运算。表达式的运算顺序可以用括号改变。

例如，表达式 x > 0 or x < 10 and y < 0 按照优先级的顺序，它和表达式 x > 0 or (x < 10 and y < 0)的运算结果是一样的。在组织表达式时，一定要熟悉各种运算符的优先级。

3.5 math 库和 random 库的使用

3.5.1 math 库的使用

math 库是 Python 数学计算的标准函数库，它提供了 44 个针对整数和浮点数的数