第3章 数据类型与基本运算

知不足，然后能自反也；知困，然后能自强也。

——《礼记·学记》

3.1 问题求解中的数据抽象

3.1.1 数据和数据类型

前面提到，在程序设计语言中，现实世界中的信息需要用程序设计语言提供的符号化手段进行表示，这种符号化表示称为数据。

数据的表示是对现实是世界问题的抽象。数据表示的选择，必须依据数据所执行的操作来考虑，以便更方便、高效地处理数据。

为什么要将数据划分为各种数据类型？

客观世界是复杂的、多样的、多变的，因此数据也是复杂的、多样的、多变的，而且不同的数据在存储、表示、运算上都有所不同。为此，程序需要对数据进行分类，以便规范和简化数据的处理过程。

一般来说，高级程序设计语言均提供以下 5 种数据类型。

(1) 整数。

(2) 浮点数：取值可以带小数点的数字类型。

(3) 布尔类型：取值为 True 和 False。

(4) 字符串类型：编程语言中表示文本的数据类型。

(5) 组合类型：多种类型的组合，往往是在基本数据类型的基础上构造的较复杂的数据类型。

在 Python 中的数据类型主要有数字类型、Bool(布尔)型、字符串类型、组合类型。其中，组合类型包括序列类型(包括字符串、列表、元组)、映射类型(字典)和集合类型。

一般高级语言中都提供类型判断函数。Python 中使用 type(x)函数进行类型判断。格式：

```
type(对象)
```

作用：返回对象的相应数据类型。

在 Python 解释器内部，所有数据类型都采用面向对象方式实现，封装为若干个类。所以 type()函数返回的是各个类的名称。<class 'int'>代表整型，<class 'float'>代表浮点型，<class 'str'>代表字符串型，<class 'list'>代表列表类型，<class 'tuple'>代表元组类型，<class 'dict'>代表字典类型，<class 'set'>代表集合类型。例如：

```
>>> type(10)
<class 'int'>
>>> type(4.5)
<class 'float'>
>>> type("结果")
<class 'str'>
>>> type([1,2,3,4,5])
<class 'list'>
>>> type((1,2,3,4,5))
<class 'tuple'>
>>> type({">=90":2,">=80":15,">=70":10,">=60":5})
<class 'dict'>
>>> type({2,15,"ab"})
<class 'set'>
```

因为数据类型决定了合法的数据操作，不合法的操作将导致程序错误，因此数据类型的重要作用是通过类型检查来发现程序中的错误。例如，如果将一位学生的姓名乘以他的分数显然是没有意义的，可是计算机无法帮助我们发现这种无意义的操作错误，这种错误只能在程序运行时才能暴露出来。但如果有了类型的概念，编译器或解释器就能尽早发现程序中的这类错误，使得在程序运行之前就有机会发现和修改错误。

此外，在程序设计语言中，每一种数据类型由两部分组成：全体合法的值和对合法值执行的各种运算(即各种数据类型的运算操作)。

3.1.2　常量、对象、变量和动态类型化

一般程序设计语言使用常量、对象和变量三种基本的方式来引用数据。

1. 常量

常量是在程序执行期间值不发生改变的量。在 Python 中，常量主要有两种：直接常量和符号常量。

(1) 直接常量。直接常量就是各种数据类型的常数值，如 123、123.45、True、False、'abc'等。

(2) 符号常量。符号常量是具有名字的常数，用名字代替永远不变的数值。

在一些模块中有时用到符号常量，如常用的 math 模块中的 pi 和 e。

```
>>> from math import *
>>> pi
3.141592653589793
```

```
>>> e
2.718281828459045
```

2. 对象

在 Python 中，一切皆对象。数据、符号、函数等都是对象。Python 中每一个对象都有唯一的身份标识(id)、一种类型和一个值。

(1) 对象的 id 是一个整数，一旦创建就不再改变，可以把它当作对象在内存中的地址，使用 id()函数可以获得对象的 id 标识。例如：

```
>>> id(107)
1503203088
>>> id('abc')
243623031952
```

(2) 对象的类型决定了对象支持的操作，也定义了对象的取值范围。对象的类型也不能改变。前面介绍过，使用 type()函数可以返回对象的类型。

(3) 根据对象的值是否可以改变，分为可变对象和不可变对象。Python 中大部分对象是不可变对象，如数值对象、字符串、元组等。字典、列表等是可变对象。

(4) 可以使用 del 语句来删除单个或多个对象。del 语句的语法格式如下：

```
del var1[,var2[,var3[...,varN]]]
```

例如：

```
del var_a, var_b
```

3. 变量和动态类型化

变量是在程序运行过程中其值可以发生变化的量。变量具有名字、数据类型和值等属性。

绝大多数编程语言对变量的使用都有严格的类型限制。而 Python 语言采用的是另一种技术——动态类型化。Python 使用动态类型化来实现语言的简洁、灵活性和多态性。所谓 Python 的动态类型化，就是在程序运行的过程中自动决定对象的类型。

在 Python 中，变量并不是某个固定内存单元的标识，Python 的变量不需要事先声明，可以直接使用赋值运算符“=”对其赋值，根据所赋的值来决定其数据类型，也就是说，不需要预先定义变量的类型。

变量指向一个对象，从变量到对象的连接称为引用。例如：

```
x = 5
```

表示创建了一个整型对象 5、变量 x，并使变量 x 连接到对象 5，也称变量 x 引用了对象 5 或 x 是对象 5 的一个引用。这个引用是可以动态的。变量类型就是它所引用的数据的类型。对变量的每一次赋值，都可以能改变变量的类型。

因为整数在程序中的使用非常广泛，为了优化速度，对于[－5，256]范围内的整数，Python 采取重用对象内存的办法。也就是说，此时 Python 采用的是基于值的内存管理

方式，如果为不同变量赋值相同值，则在内存中只有一份该值，多个变量指向同一块内存地址。例如：

```
>>> x = 107
>>> y = 107
>>> id(x)
1503203088
>>> id(y)
1503203088
```

[－5，256]范围以外的整数则不采用此内存管理方式。例如：

```
>>> x = 2222
>>> y = 2222
>>> id(x)
632978868944
>>> id(y)
632982163696
```

3.2　常用数据类型：数字、布尔型和字符串

3.2.1　数字类型

数字类型用于存储数值。改变数字数据类型会分配一个新的对象。当指定一个值时，数字对象就会被创建。例如：

```
var1 = 1
var2 = 10
```

Python 支持三种不同的数字类型：int(整数)、float(浮点数)和 complex (复数)。

1. 整数

在 Python 3.x 中，不再区分整数和长整数。整数的取值范围受限于运行 Python 程序的计算机内存大小。

整数类型有 4 种进制表示：十进制、二进制、八进制和十六进制，默认使用十进制，其他进制需要增加前导符，如表 3-1 所示。

表 3-1　整数类型的 4 种进制表示

进制	前导符	描　述
十进制	无	默认情况，例如，123、－125
二进制	0b 或 0B	例如，0b11 表示十进制的 3
八进制	0o 或 0O	例如，0o11 表示十进制的 9
十六进制	0x 或 0X	例如，0x11 表示十进制的 17

2. 浮点数

Python 的浮点数就是数学中的小数，浮点数可以用一般的数学写法，如 1.23、3.14、−9.01 等。而对于很大或很小的浮点数，就必须用科学计数法表示，把 10 用 e 替代，把 1.23×10^9 写成 1.23e9 或 12.3e8，把 0.000012 写成 1.2e−5 等。

在运算中，整数与浮点数运算的结果是浮点数，整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的，而浮点数运算则可能会有四舍五入的误差。

注意：Python 要求所有浮点数必须带有小数部分，小数部分可以是 0，这种设计可以很好地区分浮点数和整数。

Python 浮点数的数值范围和小数精度受不同计算机系统的限制，使用 Python 变量 sys.float_info 可以查看所运行系统的浮点数的各项参数，依次为最大值、基数为 2 时最大值的幂、基数为 10 时最大值的幂、最小值、基数为 2 时最小值的幂、基数为 10 时最小值的幂、能准确计算的浮点数的最大个数、科学计数法中系数的最大精度、计算机所能分辨的两个相邻浮点数的最小差值等。例如：

```
>>> 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)
```

3. 复数

复数由实数部分和虚数部分组成，一般形式为 x+yj，其中的 x 是复数的实数部分，y 是复数的虚数部分，这里的 x 和 y 都是实数。注意，虚数部分的字母 j 大小写都可以，如 5.6+3.1j 与 5.6+3.1J 是等价的。

对于复数 z，可以用 z.real 和 z.imag 分别获得它的实数部分和虚数部分。例如：

```
>>> a = 1+2j
>>> a.real
1.0
>>> a.imag
2.0
```

3.2.2 数字类型的运算

运算符是用来连接运算对象、进行各种运算的操作符号。Python 解释器为数字类型提供数值运算符、数值运算函数和数字类型转换函数等。

1. 数值运算符

数值运算符如表 3-2 所示，其中“+”“−”运算符在单目运算（单个操作数）中做取正号和负号运算，在双目运算（两个操作数）中做算术加减运算，其余都是双目运算符。

表 3-2　数值运算符与示例

数值运算符	描述	优先级	示　例

**	幂运算	1	>>> 2**3
			8
			>>> 27**(1/3)
			3.0

～	按位取反(按操作数的二进制数运算,1 取反为 0,0 取反为 1)	2	>>> ～5 －6 因为 5 的 9 位二进制为 00000101,按位取反为 11111010,即－6
＋、－	一元加号、一元减号	3	＋3 的结果是 3,－3 的结果是－3

* 、/、//、%	乘法、除法(默认进行浮点数运算,输出也是浮点数)、整商、求余数(模运算)	4	>>> 2 * 3
			6
			>>> 10/2
			5.0
			>>> 10//3
			3
			>>> 10%3
			1

＋、－	加法、减法	5	>>> 10＋3
			13
			>>> 10－3
			7

<<、>>	向左移位、向右移位	6	>>> 3<<2
			12
			>>> 3>>2
			0

&	按位与(将两个操作数按相同位置的二进制位进行操作,两者均是 1 时结果为 1,否则为 0)	7	>>> 2&3 2
^	按位异或(将两个操作数按相同位置的二进制位进行操作,不相同时结果为 1,否则为 0)	8	>>> 2^3 1
|	按位或(将两个操作数按相同位置的二进制位进行操作,只要有一个为 1 结果即为 1,否则为 0)	9	>>> 2|3 3

数字类型之间相互运算所生成的结果是“更宽”的数字类型,即整数＜浮点数＜复数。基本规则如下。

(1) 整数之间运算,如果数学意义上的结果是整数,结果是整数。

(2) 整数之间运算,如果数学意义上的结果是小数,结果是浮点数。

(3) 整数和浮点数混合运算,输出结果是浮点数。

(4) 整数或浮点与复数运算,输出结果是复数。

例如:

```
>>> 123+4.0
127.0
>>> 5.0-1+2j
(4+2j)
```

表 3-3 列出了 Python 语言支持的赋值运算符。

表 3-3 赋值运算符与示例

赋值运算符	描　　述	示　　例
=	简单的赋值运算符,把赋值号“=”右边的结果赋值给左边的变量	c = a + b
+=	加法赋值操作符	c += a 类似于 c = c + a
-=	减法赋值操作符	c -= a 类似于 c = c - a
*=	乘法赋值操作符	c *= a 类似于 c = c * a
/=	除法赋值操作符	c /= a 类似于 c = c / a
%=	取模赋值操作符	c %= a 类似于 c = c % a
**=	幂赋值运算符	c **= a 类似于 c = c ** a
//=	整商赋值运算符	c //= a 类似于 c = c // a

2. 数值运算函数

在 Python 解释器提供了一些内置函数,其中常用的数值运算函数如表 3-4 所示。

表 3-4 常用的内置数值运算函数与示例

数值运算函数	描　　述	示　　例

abs(x)	求绝对值,参数可以是整型,也可以是复数;若参数是复数,则返回复数的模	>>> abs(-5)
		5
		>>> abs(3+4j)
		5.0

divmod	分别向下取整商和求余数	>>> divmod(10,3)
		(3, 1)
		>>> divmod(-10,3)
		(-4, 2)
		>>> divmod(-10.6,3)
		(-4.0, 1.4000000000000004)

pow(x,y[,z])	pow(x,y)返回 x**y 的值
	pow(x,y,z)返回 x**y%z 的值
	pow()函数将幂运算和模运算同时进行,速度快,在加密解密算法和科学计算中非常适用
		>>> pow(2,4)
		16
		>>> pow(2,4,3)
		1

续表

数值运算函数	描　　述	示　　例

round(x[,n])	返回 x 的四舍五入值，如给出 n 值，则表示舍入到小数点后的位数	>>> round(3.333)
		3
		>>> round(3.333,2)
		3.33

max(x1,x2,…,xn)	返回 x1,x2,…,xn 的最大值，参数可以为序列	>>> max(1,2,3,4)
		4
		>>> max((1,2,3),(2,3,4))
		(2, 3, 4)

min(x1,x2,…,xn)	返回 x1,x2,…,xn 的最小值，参数可以为序列类型	>>> min(1,2,3,4)
		1
		>>> min((1,2,3),(2,3,4))
		(1, 2, 3)

注意：函数中的参数之间使用的是英文逗号，否则会出现语法错误。

例如，下例中第 2 个逗号为中文逗号，因而出现了错误信息。

```
>>> pow(2,4,3)
SyntaxError: invalid character in identifier
```

3. 数字类型转换函数

前面提到，在 Python 中，数字类型之间相互运算所生成的结果是“更宽”的数据类型，即数值运算符可以隐式地把输出结果的数字类型进行转换。此外，也可以通过内置的数字类型转换函数可以显式地进行转换，如表 3-5 所示。

表 3-5　常用的内置数字类型转换函数与示例

数字类型转换函数	描　　述	示　　例

int(x[,base])	把一个数字或字符串 x 转换成整数（舍去小数部分），base 为可选参数，指定 x 的进制，默认为十进制	>>> int(3.9)
		3
		>>> int("11",2)
		3
		>>> int("11",8)
		9
		>>> int("11",16)
		17

float(x)	把一个数字或字符串 x 转换成浮点数	>>> float(12)
		12.0
		>>> float("12")
		12.0
		注意：复数不能直接转换成其他数字类型，可以通过.real 和.imag 将复数的实部或虚部分别进行转换。例如：
		>>> float((10+99j).imag)
		99.0

续表

数字类型转换函数	描　　述	示　　例

complex(real [,imaginary])	把字符串或数字转换为复数。如果第一个参数(实数部分)为字符串,则不需要指定第二个参数(虚数部分)	>>> complex("2+1j")
		(2+1j)
		>>> complex("2")
		(2+0j)
		>>> complex(2,1)
		(2+1j)

3.2.3 布尔类型

Python 中的布尔类型用于逻辑运算,包含两个值:True(真)或 False(假),因为 Python 中布尔类型是整型的子类,所以 True 和 False 分别对应 1 和 0。例如:

```
>>> True == 1
True
>>> False == 0
True
>>> True + False + 2
3
```

注意:Python 指定,0(包括 0.0、0j 等)、空值(None)和空对象 Null(空字符串、空列表、空元组等)等价于 False,任何非 0、非空值和非空对象则等价于 True。

bool()函数用于将给定参数转换为布尔类型。例如:

```
>>> False == 0.0
True
>>> False == 0j
True
>>> bool(0)
False
>>> bool(1.5)
True
>>> print(bool(None))
False
>>> print(bool([]))
False
>>> print(bool(''))
False
```

3.2.4 字符串类型

序列类型是 Python 中常用的数据结构。Python 的常用序列类型包括字符串、列表、

元组。这里请注意它们的使用特点：

(1) 序列类型具有双向索引的功能。

序列类型中的每个元素都分配一个数字——它的位置或索引，第一个索引是 0，第二个索引是 1，以此类推；如果使用负数作为索引，则最后一个元素下标为－1，倒数第二个元素下标为－2，以此类推。可以使用负数作为索引是 Python 序列类型的一大特色。

(2) 序列类型具有切片(截取序列中部分对象)的功能。

注意：格式为

```
s1 = s[起始位置 m:结束位置 n:[步长 k]]
```

作用：将 s 中指定区间的元素复制到 s1 中，这里注意索引的区间范围是“左闭右开”，即当步长为正数时，s[m:n:k]的对象范围是 s[m]至s[n－k]；步长为负数时，按逆序获得对象，即 s[m:n:－k]的对象范围是 s[n]至 s[n＋1]。例如，s[1:5:1]的对象范围是 s[1]至 s[4]。s[10:5:－2]的对象范围是 s[10]至 s[7]。

在后面字符串、列表、元组的切片操作中，将会举例说明。

1. 字符编码

最早的字符串编码是美国标准信息交换码 ASCII，仅对 10 个数字、26 个大写英文字母、26 个小写英文字母及一些其他符号进行了编码。ASCII 采用一个字节来对字符进行编码，共定义 128 个字符。

随着信息技术的发展和信息交换的需要，各国的文字都需要进行编码，不同的应用领域和场合对字符串编码的要求也略有不同，于是分别设计了不同的编码格式，常见的主要有 UTF-8、UTF-16、UTF-32、GB2312、GBK、CP936、base64、CP437 等。

Python 3.x 完全支持中文，使用 Unicode 编码格式。Unicode 也称为统一码、万国码、单一码，是计算机科学领域里的一项业界标准。Unicode 为世界上所有字符都分配了一个唯一的数字编号，这个编号范围从 0x000000 到 0x10FFFF(十六进制)，有 110 多万。每个字符的 Unicode 编号一般写成十六进制，在前面加上 U＋。例如，“马”的 Unicode 是 U＋9A6C。Unicode 编号怎么对应到二进制表示呢？有多种方案，主要有 UTF-8、UTF-16、UTF-32，也就是说，UTF-8、UTF-16、UTF-32 都是 Unicode 的一种实现。

GB2312 是我国制定的中文编码，使用一个字节表示英文字符，2 个字节表示中文；GBK 是 GB2312 的扩充，CP936 是微软公司在 GBK 基础上开发的编码方式。GB2312、GBK 和 CP936 都是使用 2 个字节表示中文。不同编码格式之间相差很大，采用不同的编码格式意味着不同的表示和存储形式，把同一字符存入文件时，写入的内容可能会不同，在理解其内容时必须了解编码规则并进行正确的解码。如果解码方法不正确就无法还原信息。

在 Python 3.x 中，无论是一个数字、英文字母，还是一个汉字，都按一个字符对待和处理。

2. Python 字符串的界定符

字符串是 Python 中最常用的数据类型。可以使用引号(单引号、双引号和三引号)

来创建字符串。例如：

```
>>> var1 = 'Hello World!'
>>> print("var1: ", var1)
var1:  Hello World!
```

（1）使用单引号作为界定符时，可以使用双引号作为字符串的一部分。例如：

```
>>> print('"x =:"')
"x =:"
```

（2）使用双引号作为界定符时，可以使用单引号作为字符串的一部分。例如：

```
>>> print("'x =:'")
'x =:'
```

（3）使用三引号作为界定符时，可以使用单引号或双引号作为字符串的一部分，还可以换行。例如：

```
>>> print('''"python"''')
"python"
>>> print('''hello
python''')
hello
python
```

3. Python 转义字符

当需要在字符中使用特殊字符时，Python 用反斜杠(\)转义字符，如表 3-6 所示。

表 3-6　Python 转义字符

转义字符	描　　述
\(以行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页

续表

转义字符	描　　述
\0yy	八进制数，yy 代表的字符，例如，\012 代表换行
\xyy	十六进制数，yy 代表的字符，例如，\x0a 代表换行
\other	其他的字符以普通格式输出

3.2.5　字符串类型的运算

1. 字符串运算符

字符串运算符如表 3-7 所示。

表 3-7　字符串运算符与示例

字符串运算符	描　　述	示　　例
+	字符串连接	>>> 'Hello'+'Python' 'HelloPython'
*	x * n 或 n * x 表示重复输出 n 次字符串 x	>>> 'Hello' * 2 'HelloHello'

in	成员运算符，如果字符串中包含给定的字符返回 True，否则返回 False	>>> 'h' in 'hello'
		True
		>>> 'H' in 'hello'
		False

not in	成员运算符，如果字符串中不包含给定的字符返回 True，否则返回 False	>>> 'h' not in 'hello'
		False
		>>> 'H' not in 'hello'
		True

[]	索引：使用下标索引来访问值	>>> a='Hello'
		>>> a[1]
		'e'

[:]	切片：截取字符串中的子串	>>> a='Hello'
		>>> a[1: 4]
		'ell'
		>>> a[4: 1: -2]
		'ol'

2. 字符串类型的格式化

Python 支持格式化字符串的输出。最基本的用法是将一个值插入到一个包含字符串格式符 %s 的字符串中。例如：

```
>>> print ("My name is %s and weight is %d kg!" % ('xiaoming', 61))
My name is xiaoming and weight is 61 kg!
```

另外，Python 已经不在后续版本中使用类似 C 语言中 printf 的格式化方法，而是主

要采用 format()方法进行字符串格式化，建议读者尽量采取此方法。

(1) format()方法的基本使用格式。

format()方法的基本使用格式如下：

```
<模板字符串>.format(<逗号分隔的参数>)
```

用法：将＜逗号分隔的参数＞按照序号关系替换到＜模板字符串＞的一对大括号“{}”所代表的槽中(从 0 开始编号)，若无序号则按照出现顺序替换。例如：

```
>>> "My name is {} and weight is {}!".format("xiaoming",60)
'My name is xiaoming and weight is 60!'
```

如果需要输出大括号，使用两层嵌套即可。例如：

```
>>> "圆周率{{{1}{2}}}是{0}".format("无理数",3.1415926,"")
'圆周率{3.1415926}是无理数'
```

(2) format()方法的格式控制。

在 format()方法的模板字符串的一对大括号“{}”所代表的槽中，除了可以包括参数序号，还可以包括格式控制信息，格式如下，其中的参数均是可选的。

```
{[参数序号]:[填充][对齐][宽度][,][.精度][类型]}
```

- 填充：指宽度内除了参数外的字符采用什么方式表示，默认为空格。
- 对齐：指宽度内参数输出的对齐方式，使用＜、＞、^分别表示左对齐、右对齐和居中对齐，默认为左对齐。
- 宽度：指定参数输出的字符宽度，如果参数实际宽度大，则使用实际宽度，如果实际宽度小，则用填充符填充，默认用空格填充。
- 逗号：用于显示数字类型的千位分隔符。
- 精度：表示浮点数的小数部分输出的有效位数或者字符串输出的最大长度。
- 类型：通过格式化符号控制输出格式。Python 字符串格式化符号如表 3-8 所示。

表 3-8 字符串格式化符号

格式化符号	描　　述
c	输出对应的 ASCII 码字符
b	输出二进制整数
d	输出十进制整数
o	输出八进制整数
x、X	输出十六进制数(小写、大写)
e、E	输出浮点数的指数形式(基底写为 e、E)
f	输出浮点数的标准浮点形式，可指定小数点后的精度
g、G	f 和 e 的功能组合、f 和 E 的功能组合
%	输出浮点数的百分形式

例如：

```
>>> s = "python"
>>> "{0:30}".format(s)
'python                        '
>>> "{0:>30}".format(s)
'                        python'
>>> "{0:*^30}".format(s)
'************python************'
>>> "{0:3}".format(s)
'python'
>>> "{0:-^20,}".format(12345.6789)
'----12,345.6789-----'
>>> "{0:.2f}".format(12345.6789)
'12345.68'
>>> "{0:.4}".format("python")
'pyth'
>>> "{0:c},{0:b},{0:d},{0:o},{0:x},{0:X}".format(20)
'\x14,10100,20,24,14,14'
>>> "{0:e},{0:E},{0:f},{0:F},{0:g},{0:G},{0:%}".format(1230000)
'1.230000e+06,1.230000E+06,1230000.000000,1230000.000000,1.23e+06,1.23E+06,
123000000.000000%'
```

3. 字符串运算函数

在 Python 解释器提供了一些内置函数，其中字符串运算函数如表 3-9 所示。

表 3-9　常用的内置字符串运算函数与示例

字符串运算函数	描　　述	示　　例

len(x)	返回字符串 x 的长度(一个英文和中文字符都是一个长度单位)，或其他组合数据类型的元素个数	>>> len("Python，你好!")
		10
		>>> len([1,2,3])
		3

str(x)	返回任意类型 x 的字符串形式	>>> str(123.45) '123.45'

eval(x)	计算字符串 x 中有效的表达式值，从而将 x 转换成数字类型	>>> eval('2 + 2')
		4
		>>> eval('80')
		80

chr(x)	返回 Unicode 编码 x 对应的单字符	>>> chr(65) 'A'
ord(x)	返回单字符 x 对应的 Unicode 编码	>>> ord("A") 65
hex(x)	返回整数 x 对应十六进制数的小写形式字符串	>>> hex(12) '0xc'
oct(x)	返回整数 x 对应八进制数的小写形式字符串	>>> oct(9) '0o11'

4. 字符串处理方法

字符串类共包含 43 个内置方法，常用的如表 3-10 所示(其中 string 代表字符串)。

表 3-10 常用的内置字符串处理方法与示例

字符串处理方法	描　　述	示　　例
string.lower()	把 string 中的所有字符转换为小写	>>> 'Abc'.lower() 'abc'
string.upper()	把 string 中的所有字符转换为大写	>>> 'Abc'.upper() 'ABC'
string.islower()	如果 string 中的所有字符都是小写，返回 True，否则返回 False	>>> 'Abc'.islower() False
string.isprintable()	如果 string 中的所有字符都是可打印的，返回 True，否则返回 False	>>> 'Abc'.isprintable() True
string.isalpha()	如果 string 中的所有字符都是字母，返回 True，否则返回 False	>>> '123a'.isalpha() False
string.isnumeric()	如果 string 中的所有字符都是数字，返回 True，否则返回 False	>>> '123a'.isnumeric() False
string.isspace()	如果 string 中的所有字符都是空格，返回 True，否则返回 False	>>> '　'.isspace() True

string.startswith(obj, [start[,end]])	检查字符串是否在 start 至 end 指定的范围内且以 obj 开头，是则返回 True，否则返回 False	>>> 'abcde'.startswith('a')
		True
		>>> 'abcde'.startswith('a',1,3)
		False

string.endswith(obj, [start[,end]])	检查字符串是否在 start 至 end 指定的范围内且以 obj 结束，是则返回 True，否则返回 False	>>> 'abcde'.endswith('e')
		True
		>>> 'abcde'.endswith('e',0,3)
		False

string.split(str="" [, num = string.count(str)])	以 str 为分隔符(默认分隔符为空格)切片 string，并放入一个列表中，如果 num 有指定值，则仅分隔为 num +1 个子字符串	>>> '192.168.3.2'.split('.')
		['192', '168', '3', '2']
		>>> '192.168.3.2'.split('.', 2)
		['192', '168', '3.2']

string.count(str, [start[,end]])	返回在 start 至 end 指定的范围内 str 在 string 中出现的次数	>>> 'abcade'.count('a')
		2
		>>> 'abcade'.count('a', 2,5)
		1

string.replace(str1, str2, num = string.count(str1))	把 string 中的 str1 替换成 str2，如果指定 num，则替换前 num 次	>>> 'abcade'.replace('a', '2')
		'2bc2de'
		>>> 'abcade'.replace('a', '2',1)
		'2bcade'

续表

字符串处理方法	描　　述	示　　例

string.center(width)	将 string 居中对齐，并使用空格将 string 填充至长度为 width	>>> 'abcde'.center(7)
		' abcde '
		>>> 'abcde'.center(8)
		' abcde '
		>>> 'abcde'.center(1)
		'abcde'

string.lstrip()	删除 string 左边的空格	>>> ' abcde '.lstrip() 'abcde '
string.rstrip()	删除 string 字符串末尾的空格	>>> ' abcde '.rstrip() ' abcde'
string.strip([obj])	在 string 上执行 lstrip()和 rstrip()	>>> ' abcde '.strip() 'abcde'

string.zfill(width)	返回长度为 width 的字符串，原字符串 string 右对齐，前面填充 0	>>> 'abc'.zfill(5)
		'00abc'
		>>> '-123'.zfill(5)
		'-0123'

string.join(seq)	以 string 作为分隔符，将组合数据类型 seq 变量中的所有元素（以字符串表示）合并为一个新的字符串	>>> color='red','blue','green'
		>>> '&'.join(color)
		'red&blue&green'

例 3-1　使用字符串函数和方法进行微信注册账号的判断和处理。要求：使用手机号注册微信账号，即长度为 11 位，必须是数字，而且以数字 1 开头。

【程序 3-1.py】

```
account = input("请输入微信账号:")
if len(account) == 11 and account.isnumeric() and account[0] == "1":
    print("账号格式正确!")
else:
    print("账号格式不正确!")
```

3.3　列　　表

3.3.1　列表定义与特点

Python 的列表功能非常强大，有人戏称它是“打了激素”的数组。

列表是一组有序存储的数据。比如，菜单就是一种列表。列表的主要特点如下。

（1）列表是一个有序序列。

（2）同一个列表中，可以包含任意类型的对象。

(3) 列表是可变的,可以添加、删除、直接修改列表成员。

(4) 列表存储的是对象的引用,而不是对象本身。

3.3.2 列表基本操作

1. 创建列表

可以使用方括号把由逗号分隔的不同数据项括起来,或使用 list()方法创建一个列表。

list()方法的语法格式如下:

```
list(seq)
```

作用:将元组或字符串 seq 转换为列表。

比如,在下例中,我们看到在同一个列表中,可以包含任意类型的对象。

```
>>> ['physics', 'chemistry', 19.97, 2000]
['physics', 'chemistry', 19.97, 2000]
>>> list('abcd')
['a', 'b', 'c', 'd']
```

2. 索引

可以使用下标索引来访问列表中的值,前面介绍过,序列类型具有双向索引的功能。列表使用方括号作为索引操作符,索引序号不能超过列表的元素范围,否则会出现 IndexError 错误。例如:

```
>>> x = [1, 2, 3, 4, 5, 6, 7 ]
>>> x[1]
2
>>> x[-1]
7
>>> x[-2]
6
>>> x[8]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    x[8]
IndexError: list index out of range
```

3. 修改或添加元素

(1) 直接修改列表元素。例如:

```
>>> x = [1, 2, 3, 4, 5, 6, 7]
>>> x[2] = 'a'
>>> print ("x[1:5]: ", x[1:5])
x[1:5]:  [2, 'a', 4, 5]
```

(2) 添加单个对象。使用 append()方法可以在列表末尾添加一个对象,append()方

法的语法格式如下：

```
list.append(obj)
```

作用：将对象 obj 添加到列表 list。

例如：

```
>>> x = [1, 2, 3, 4, 5, 6, 7 ]
>>> x.append('a')
>>> x
[1, 2, 3, 4, 5, 6, 7, 'a']
```

(3) 添加多个对象。使用 extend()方法可以在列表末尾添加多个对象。extend()方法的语法格式如下：

```
list.extend(seq)
```

作用：将 seq 添加到列表 list。

例如：

```
>>> x = [1,2]
>>> x.extend(['a','b'])
>>> x
[1, 2, 'a', 'b']
```

(4) 插入对象。使用 insert()方法可以在指定位置插入对象，insert()方法的语法格式如下：

```
list.insert(index, obj)
```

作用：在索引位置 index 处插入对象 obj。

例如：

```
>>> x = [1,2]
>>> x.insert(0,'a')
>>> x
[ 'a',1, 2]
```

4. 删除元素

(1) 按值删除对象。使用 remove()方法可以删除指定对象。remove()方法的语法格式如下：

```
list.remove(obj)
```

作用：删除对象 obj。

例如：

```
>>> x = [1,2,4,3]
>>> x.remove(2)
>>> x
[1, 4, 3]
```

(2) 按位置删除对象。使用 pop()方法可以删除指定位置的对象。pop()方法的语法格式如下：

```
list.pop([index])
```

作用：删除对象索引值为 index 的对象，并且返回该元素的值。默认 index=－1，即删除最后一个列表值。

例如：

```
>>> x = [1,2,3,4]
>>> x.pop()
4
>>> x.pop(1)
2
```

(3) 使用 del 语句删除对象。使用 del 语句可以删除指定对象。语法格式如下：

```
del list[index]
```

作用：index 可以是单个元素索引值，也可以是连续几个元素的索引值。

例如：

```
>>> x = [1,2,3,4,5]
>>> del x[0]
>>> x
[2, 3, 4, 5]
>>> del x[2:4]
>>> x
[2, 3]
```

(4) 通过 clear()方法删除所有对象。例如：

```
>>> x = [1,2,3]
>>> x.clear()
>>> x
[]
```

5. 求长度

可用 len()函数求列表长度，即列表元素的个数。例如：

```
>>> len([1, 2, 3])
3
>>> len([1,2,('a'),[3,4]])
4
```

6. 合并

加法运算符可用于合并。例如：

```
>>> [1, 2, 3] + ['a',5, 6]
[1, 2, 3, 'a', 5, 6]
```

7. 重复

乘法运算符可用于创建具有重复值的列表。例如：

```
>>> ['@'] * 4
['@', '@', '@', '@']
>>> [1,2] * 3
[1, 2, 1, 2, 1, 2]
```

8. 判断元素是否存在

使用 in 运算符判断元素是否存在于列表中。例如：

```
>>> 2 in [1,2,3]
True
>>> 5 in[1,2,3]
False
```

9. 切片

列表与字符串类似，可以通过切片来获得列表中的部分对象。例如：

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[1:5]
[2, 3, 4, 5]
>>> x[5:10]
[6, 7, 8, 9]
>>> x[2:7:2]
[3, 5, 7]
>>> x[7:2:-1]
[8, 7, 6, 5, 4]
>>> x[7:2:-2]
[8, 6, 4]
```

10. 嵌套

可以通过嵌套列表来表示矩阵。例如：

```
>>> x = [[1,2,3],[4,5,6],[7,8,9]]
>>> x[0]
[1, 2, 3]
>>> x[0][0]
1
```

11. 复制列表

使用 copy()方法可以复制列表对象。例如：

```
>>> x = [1,2,3]
>>> y = x.copy()
>>> y
[1, 2, 3]
```

12. 列表排序

使用 sort()方法可以将列表对象排序，若列表中包含多种类型则会出错。例如：