# 第3章

# Python程序设计

本章介绍程序设计语言的概述和 Python 语言的使用。

## 3.1 计算机语言和程序开发环境

为编写计算机程序而使用的语言称为**计算机语言**或**程序设计语言**，本章要介绍的 Python 语言是众多的高级程序设计语言之一。

### 3.1.1 程序设计语言和语言处理程序

程序设计语言是人与计算机之间交流的工具，按照和硬件结合的紧密程度，程序设计语言经历了机器语言、汇编语言和高级语言的发展阶段。

**1. 程序设计语言的发展**

1）机器语言

**机器语言**是计算机系统能够识别的，不需要翻译就可以供机器使用的程序设计语言。在机器语言编写的程序中，每一条指令都是采用二进制的形式，即操作码和操作数都是用 0 和 1 序列表示的。由于计算机能够识别二进制的指令和数据，所以用机器语言编写的程序可以直接执行，因此执行效率较高。

但是，用机器语言编写的程序可读性差、不易记忆，无论是编写还是调试难度都很大，不易掌握和使用。同时，机器语言依赖于具体的机器，在某种类型计算机上编写的机器语言程序不能直接在另一类计算机上使用，因此，它的可移植性差。

2）汇编语言

**汇编语言**也是面向机器的语言，它采用特定的助记符号来表示机器语言中的指令和数据，即采用比较容易识别和记忆的符号，这些符号可以是单词，例如，使用 ADD 表示加法；也可以是单词的一部分，例如，使用 SUB 表示减法，如下面的指令：

```
ADD AX,BX
```

指令中的 AX 和 BX 是 CPU 中的两个寄存器，该指令表示将寄存器 AX 中的内容和 BX 中的内容相加并将结果保存到 AX 中，显然，它比用机器语言编写的程序易于理解和记忆。

但是,汇编语言只是将机器语言中的指令符号化,它仍然是依赖于机器的语言,同样也存在可移植性差的问题。

不论是机器语言还是汇编语言,在编写程序时都和计算机的硬件结构关系非常密切,从而导致在一个计算机系统下编写的程序在另一个计算机系统中不一定能够正确执行,这两类语言又被称为**低级语言**。

3) 高级语言

高级语言接近自然语言(通常是英语),使用单词、单词缩写、数学运算符、运算式表示命令、数据运算和运算式,不依赖计算机硬件,因此,通用性和移植性都较好。

高级语言的种类较多,常用的有面向过程的结构化程序设计语言,如 FORTRAN、BASIC、C、Pascal 等;还有面向对象的程序设计语言,如 Visual Basic、C ++ 、C ♯ 、Java、Delphi、Python 等;也有用于动态网页设计中的脚本设计语言,如 VBScript、JavaScript、PHP、Perl 等。

**2. 语言处理程序**

在程序设计语言中,使用除机器语言之外的其他语言编写的程序都必须经过翻译,转换为机器指令,才能在计算机上执行,翻译的工具称为**语言处理程序**。

翻译之前的程序称为**源程序**(Source Program)或**源代码**(Source Code),翻译之后产生的程序称为**目标程序**(Object Program)。

因此,计算机上能提供的各种语言,必须配备相应语言的语言处理程序,语言处理程序有汇编程序、解释程序和编译程序。

1) 汇编程序

用汇编语言编写的程序称为**汇编语言源程序**,对于计算机来说并不能直接识别和执行,因此,在执行之前必须先将汇编语言编写的程序翻译成机器语言程序才能被执行,这一翻译过程称为**汇编**,完成汇编操作的是事先保存在计算机中的**汇编程序**,翻译后的机器语言程序称为**目标程序**。

2) 解释程序

将高级语言编写的源程序翻译成机器语言指令时,有两种翻译方式,分别是“解释”方式和“编译”方式,分别由解释程序和编译程序完成。

**解释方式**是通过解释程序对源程序一边翻译一边执行,早期的 BASIC 语言采用的就是解释方式。解释是翻译一句,执行一句。再次执行时需再次解释执行。

3) 编译程序

**编译过程**是这样的,首先将源程序编译成目标程序,目标程序文件的扩展名是.obj,然后再通过连接程序将目标程序和库文件相连接形成可执行文件,可执行文件的扩展名是.exe。编译是一次性地将源程序翻译成机器语言程序。

大多数高级语言编写的程序采用编译的方式,不同的高级语言对应了不同的编译程序。由于编译后形成的可执行文件独立于源程序,因此可以反复地运行,运行时只要给出可执行程序的文件名即可,因此运行速度较快。

本书介绍的 Python 语言用的是解释方式。

无论是编译程序还是解释程序，都需要事先将源程序输入计算机中，然后才能进行编译或解释。为了方便地进行程序的开发，目前，许多编译软件都提供了集成开发环境(Integrated Development Environment，IDE)。**集成开发环境**是指将程序的编辑、编译、运行、调试等功能集成在一起的软件，使程序设计者既能高效地编写、运行程序，又能方便地调试程序。

### 3.1.2　安装 Python 语言环境

Python 语言诞生于 1990 年，由 Guido van Rossum 设计并领导开发，经历了 30 多年持续不断的发展。由于 Python 语言具有简单、易学、高级、面向对象、可扩展、可嵌入、丰富的库等特点，现在已广泛应用于 Web 开发、网络编程、科学计算、数据可视化、图像处理、自然语言处理、机器学习等多个领域，成为目前非常流行的程序设计语言之一。

目前常用的是 Python 的 3.x 版本，本书以 Python 3.7(其他版本类似)为基础进行介绍。

在 Windows 操作系统下，安装 Python 环境的步骤如下。

(1) 登录 https://www.python.org/网站。

(2) 选择菜单栏中的 Downloads，然后单击 Download for Windows Python 3.7.4(其他 3.x 版本亦可)下载 Python 安装程序，如图 3-1 所示。

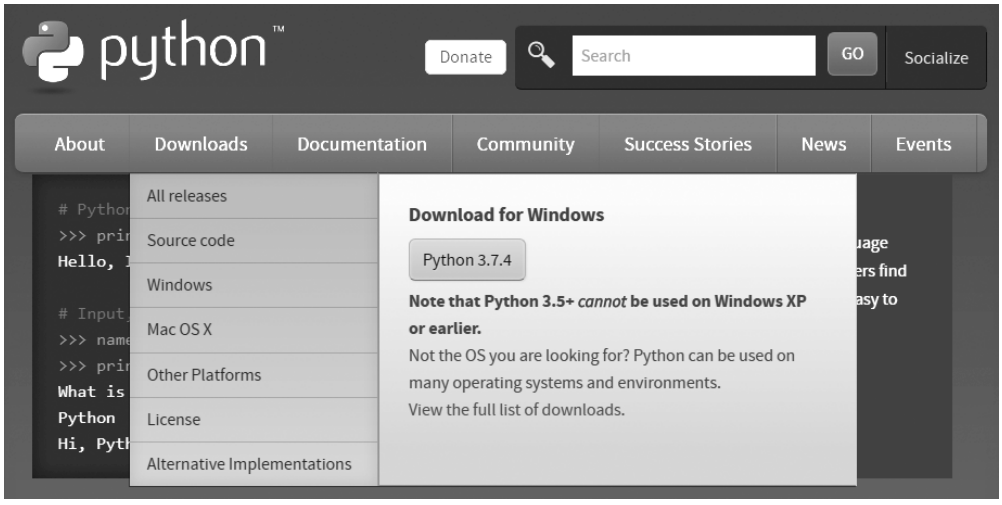



图 3-1　下载 Python 安装程序

(3) 双击下载的 python-3.7.4 安装程序，出现如图 3-2 所示的安装界面。

(4) 在图 3-2 中单击 Install Now 选项。如果需要改变安装路径，可以单击 Customize installation 选项。

(5) 等待安装 Python 程序进度条完成，单击 Close 按钮后，就完成了 Python 程序的安装。

此外，Python 也有许多 IDE，如 PyCharm、Anaconda 等。


图 3-2　安装界面

### 3.1.3　使用 Python 编程

运行 Python 程序有两种方式：交互式和文件式。使用交互式时在 Python 解释器中输入一条命令，解释器立即给出结果。文件式是将程序保存在一个或多个文件中，然后启动解释器执行程序中的所有命令，这是常用的编程和执行方式。

**1. 使用交互式的带提示符的解释器**

在 Windows“开始”菜单(图 3-3)中，单击“开始”→“程序”→Python 3.7→IDLE (Python 3.7 32-bit)，打开 Python 交互式解释器窗口，如图 3-4 所示。


图 3-3　菜单命令

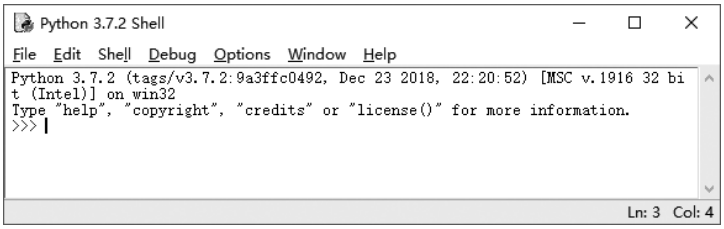



图 3-4　交互式解释器窗口

在交互式解释器窗口中的“>>>”提示符下可以输入 Python 的语句，然后回车即可执行该语句并在下一行显示运行的结果。

例如：

```
>>> print("Hello, World!")
Hello, World!
```

又如：

```
>>>print(1+1/2)
1.5
```

**2. 在集成环境中编写并执行程序**

操作过程如下。

(1) 在如图 3-4 所示的交互式解释器窗口中,选择 File→New File 命令,打开 Python 程序编辑窗口,在编辑窗口内输入 Python 程序,如图 3-5 所示。

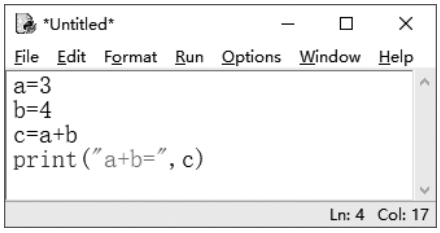



图 3-5　在编辑窗口中输入程序

(2) 在程序编辑窗口中,选择 File→Save As… 命令,打开“另存为”对话框,如图 3-6 所示。

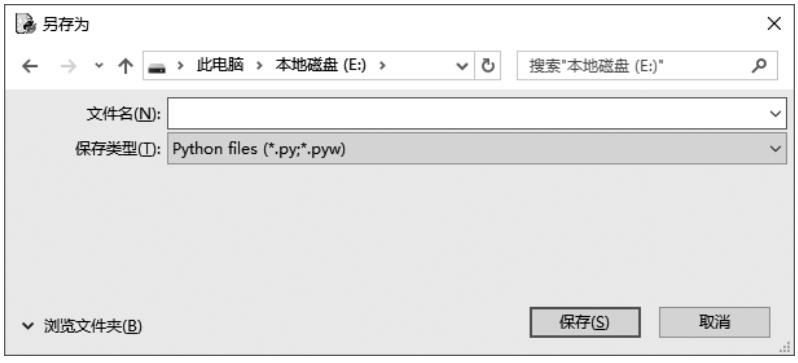



图 3-6　“另存为”对话框

(3) 在“另存为”对话框中,输入程序文件的名称,输入时文件名后面一定要加上后缀“.py”,并**注意保存的路径**,然后单击“保存”按钮。

(4) 在程序编辑窗口中,选择 Run→Run Module 命令(图 3-7,或直接按 F5 键)运行程序,运行后的结果显示在交互式解释器窗口中(图 3-8)。

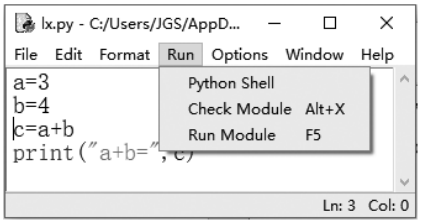



图 3-7　运行程序

**3. 在命令提示符下运行源文件**

假定在 Python 文件夹中已经存在 Python 源程序文件“x3.py”,则在命令提示符下运行该文件的方法如下。

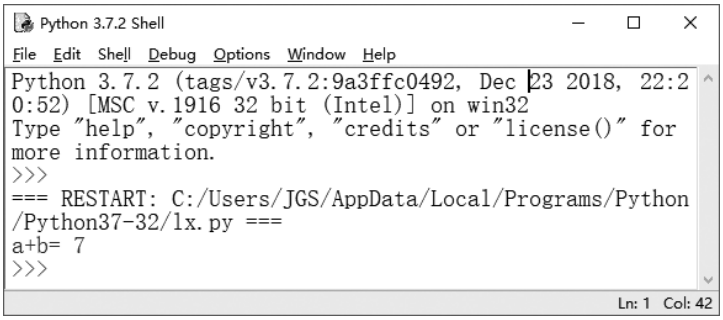



图 3-8　程序的运行结果

(1) 在 Windows 桌面单击“开始”→“Windows 系统”→“命令提示符”。

(2) 输入“cd\python37”(设 python 的安装路径为 c:\python37,如果是其他版本,可能是 python36 或 python39,“python37”是 Python 的安装路径)。

(3) 输入“python　x3.py”。

本章主要使用交互方式执行单独的命令,用 IDLE 集成环境运行完整的程序。

### 3.1.4　Python 程序的编码特点

#### 1. 编码规则

一般每行一条语句,也是推荐的编码方式。如:

```
print("Hello World")
```

如果一行有多条语句,语句之间用分号隔开。如:

```
print("Hello World"); print("Hello CHINA")
```

一行写不下时,使用“\”表示续行,如:

```
print("Hello World",\
"Hello Python")
```

使用续行时注意以下两点:

(1) 不能在标识符中间、常量中间等有意义的符号中间进行换行。

(2) 语句中包含[],{}或()括号时不需要使用多行连接符。

#### 2. 添加注释

为程序添加注释,是对代码添加解释性的文字,添加注释是一种好的编程习惯,注释仅为阅读理解,不会被执行。

通常有下面两种形式的注释。

(1) 单行注释:一行中用“#”号说明后面的文字为**单行注释**。

(2) 多行注释:用一对三引号"'括起的一段程序,表示一段注释。它表示的实际是一个字符串。

对于程序中暂时不想执行的语句也可以改成注释。

**3. 缩进方式**

Python 通过缩进对齐方式来区分不同的语句块。一般缩进为 4 个空格或一个制表位,在一个语句块中的各条语句需要保持一致的缩进量,不要混合使用空格和 Tab。

对于分支、循环、函数等的语句块要向右缩进并对齐。例如,下面的 if 语句中的 y=-x 和 y=x 是缩进对齐的。

```
if(x<0):
    y=-x
else:
    y=x
```

相同的语句如果使用了不同的缩进对齐方式,会产生不同的运行结果,例如,下面的两段程序的结果是完全不同的。

下面一段最后一条语句 print("s=",s)和上面一行的 s=s+i 上下对齐,都作为循环体中的语句。

```
s=0
for i in[1,2,3,4,5]:
    s=s+i
    print("s=",s)
```

该程序的运行结果为

```
s=1
s=3
s=6
s=10
s=15
```

下面一段的最后一条语句 print("s=",s)和 for 对齐,作为循环之外的语句。

```
s=0
for i in[1,2,3,4,5]:
    s=s+i
print("s=",s)
```

运行结果如下。

```
s=15
```

# 3.2 简单 Python 程序的编写

## 3.2.1 数据类型

**1. 不同的数据类型**

计算机中使用不同的方式存储不同的数据,一种存储形式及这种形式的数据运算构

成一种数据类型,例如,数字、字符串、列表、字典、元组、文件,其中,数字类型还有整数(int)、浮点数(float)、复数(complex)、逻辑型(bool)。

复数由实数部分和虚数部分构成,可以用 $a+b$j,或者 complex($a$,$b$)表示,复数的实部 $a$ 和虚部 $b$ 都是浮点型。

此外,其他类型还有字符串(string)、列表(list)、元组(tuple)和字典(dictionary)等。

**2. 使用 type()查看数据类型**

可以使用 type()函数来查看数据或变量的数据类型,格式如下。

```
type(表达式)
```

或

```
type(变量名)
```

例如:

```
>>> type(2+3)
<class 'int'>
```

表示 2+3 的结果是整型。

```
>>> type("string")
<class 'str'>
```

查看结果是字符串型。

```
>>> type(3+4j)
<class 'complex'>
```

查看结果是复数。

```
>>> type(1.2)
<class 'float'>
```

查看结果是浮点型。

```
>>> type(True)
<class 'bool'>
```

查看结果是布尔型。

**3. 常量**

常量是直接写出来的数据,例如,2、3.4、3+5j、"computer"。

1) 整型常量

整型数据就是整数,可以有正负号。

整型常量有十进制、二进制、八进制、十六进制之分。

十进制:16,0,-88。

二进制：0b 或 0B 开头(前面是数字 0)。例如，0b1111。

八进制；0o 或 0O 开头(前面是数字 0,后面是大写字母 O 或小写字母 o)。例如,0o1234。

十六进制：0x 或 0X 开头。例如,0xabc。

2) 实型常量

浮点型用于表示实数,常量书写有两种形式：十进制小数形式,如 10.0、3.14,带有小数点;指数形式即科学记数法,如 12.34e－5 表示 $12.34\times10^{-5}$ 即 0.0001234(其中的 e 可以大写)。

由于实数在计算机中的表示是不精确的,实数运算时可能会出现精度损失的问题。例如：

```
>>> a=2/3
>>> print("%.16f"%(a))
0.6666666666666666
```

其中,"%.16f"表示显示实数,保留小数点后 16 位,a 是要显示的数。print 一行中可改的是 16 和 a,其他均为格式要求。

3) 字符串

用单引号、双引号或三引号引起来的一串符号叫作**字符串常量**,用三引号括起来的字符串可以是多行的(三引号实际是三个单引号或三个双引号,如'''或""")。

```
>>> print('hello world')
hello world
>>> print("hello world")
hello world
>>> print("""hello world""")     #三个单引号或三个双引号
hello world
```

4) 复数类型常量

复数类型(complex)的一般形式是：$a+b$j,其中,$a$ 是实部,$b$ 是虚部。

```
>>> a=3+4j
>>> b=5-6j
>>> print(a+b)
(8-2j)
>>> type(a)
<class 'complex'>
>>> x=10+20j
>>> x.real
10.0
>>> x.imag
20.0
```

5) 布尔类型

**布尔值**也叫**逻辑值**,描述逻辑判断的结果,只有两个值：逻辑真(True,整数 1)和逻辑假(False,整数 0)。布尔表达式包括关系运算表达式和逻辑运算表达式,在程序中表示

条件,条件满足时结果为 True,条件不满足时结果为 False。

例如:

```
>>> a=4; b=5
>>> a>b
False
>>> a<b
True
>>> type(a<b)
<class 'bool'>
```

又如下面的例子,用于两个字符串之间的比较,结果也是布尔型的。

```
>>> c='ascend';  d='descend'
>>> type(c>d)
<class 'bool'>
>>> c>d
False
```

6) 转义字符

**转义字符**是以反斜杠"\"开头,后跟一个具有特定含义的字符,主要用来表示不方便表达的控制代码,常用的转义字符如下。

\0 表示空字符。

\t 表示水平制表符(Tab)。

\n 表示换行符。

\r 表示回车符。

\\表示反斜杠。

\' 表示单引号。

\" 表示双引号。

例如,语句 print("123\n# # #\t@@@\n456")的输出结果如下。

```
123
###      @@@
456
```

语句中的输出字符串中有两个"\n",在这两个字符之后的内容另起一行显示,所以结果输出了三行,在输出的第二行中,"\t"是制表符,其后的内容跳到下一个制表位,由于一个制表区占 8 位,因此,# # 和@@@之间有 5 个空格。

如果不想发生转义,可以在字符串前添加一个 r,表示**原始字符串**。

例如,表达转义时,语句 print("hello,\nworld")中的"\n"换行有效,所以输出结果如下。

```
hello,
world
```

不想发生转义时,语句 print(r"hello,\nworld")的输出结果是 hello,\nworld,此时的"\n"不作为换行,而是照原样输出的两个字符。

函数 len()用于返回字符串所包含字符的个数，例如，len("hello")的结果为 5，len("hello,\nworld")的结果为 12，而函数 len(r"hello,\nworld")的结果则为 13。

### 4. 变量和赋值运算

**变量**是保存和表示数据值的一种符号，在计算机中它对应一个存储单元，变量的值可以通过赋值改变。对变量赋值使用赋值运算符“=”，该运算符左边是一个变量名，右边是表达式。其格式如下。

变量名=表达式

上面格式的含义是先计算表达式，然后将表达式的值赋给左边的变量。在 Python 中，对变量赋值不需要类型声明。

每个变量在使用前都必须赋值，赋值后该变量才会被创建。

Python 允许同时为多个变量赋相同的值或分别赋不同的值。

例如，a=b=c=1 的结果是 a、b、c 三个变量的值都是 1。

为多个变量赋予不同的值时赋值语句的格式如下。

变量 1,变量 2,…,变量 *n*=表达式 1,表达式 2,…,表达式 *n*

例如，a,b,c=1,2,"Hello"，该赋值语句的结果是 a、b、c 三个变量的值分别是 1、2 和 "Hello"。

又例如：

```
>>>a, b=10, 20
>>>a, b=b, a
>>>a
20
>>>b
10
```

上面的操作实际上完成了两个变量值的交换。

Python 语言中的变量具有如下的特点。

- 使用前不需要先定义。
- 变量类型由表达式值的类型确定。
- 变量的类型可以改变。

### 5. 标识符

**标识符**是程序中表示变量、函数、类名等的符号，标识符的命名规则如下。

- 包含字母、数字、下画线。
- 以字母或下画线开头。
- 使用的字母区分大小写。
- 不能使用系统保留的关键字，例如 if、else、int 等。

### 6. Python 的关键字

**关键字**也称为**保留字**，是被语言内部定义并保留使用的标识符，在程序中给变量、函

数等命名时不能使用和保留字相同的标识符。

Python 3.x 版本有如下 33 个保留字,保留字也是区分大小写的。

| | | | | |
|---|---|---|---|---|
| and | del | from | None | True |
| as | elif | global | nonlocal | try |
| assert | else | if | not | while |
| break | except | import | or | with |
| class | False | in | pass | yield |
| continue | finally | is | raise | |
| def | for | lambda | return | |

## 3.2.2 运算符和表达式

表示运算的符号叫**运算符**,参与运算的数据称为**操作数**。常量、变量以及由运算符将变量、常量、表达式连接起来的式子称为**表达式**,例如,b * b－4 * a * c 是一个表达式。

Python 的基本运算主要包括算术运算、关系运算、逻辑运算和位运算等,由相应的运算符完成。对应地,组成的表达式称为算术表达式、关系表达式和逻辑表达式,位运算表达式也是算术表达式。

### 1. 算术运算符

算术运算符见表 3-1。

**表 3-1 算术运算符**

| 算术运算符 | 含 义 | 示例(设 $a=10, b=20$) |
|---|---|---|
| ＋ | 加 | $a+b$,结果为 30 |
| － | 减 | $a-b$,结果为－10 |
| * | 乘 | $a*b$,结果为 200 |
| / | 除 | $a/b$,结果为 0.5 |
| % | 求余 | a%$b$,结果为 10 |
| ** | 乘方 | $a$**3,结果为 1000 |
| // | 整除 | 123//10,结果为 12 |

算术运算符的含义比较容易理解,下面主要介绍取模和整除的用法,看下面的示例。

```
>>> a=216
>>> a//100,a//10,a%10
(2, 21, 6)
```

可以看出,对于一个三位的整数,如果整除 100,则得到该三位数中的百位数;如果计算除以 10 的余数,则可以得到该数的个位数。

进一步,对于一个四位的整数,如果整除 1000,则得到该四位数中的千位数,可以以此类推得到更多位。

而不论是几位的整数，计算除以 10 的余数都可以得到该数的个位数，如果这两个运算符配合使用，可以取出任意位整数中的每一位。

如果计算除以 2 的余数，根据结果是否为零，可以对该数进行奇偶性的判断。

### 2. 比较运算符

**比较运算符**也称为**关系运算符**，共有 6 个，见表 3-2。

**表 3-2　比较运算符**

| 比较运算符 | 含　义 | 示例（设 $a=10, b=20$） |
|---|---|---|
| == | 相等 | $a==b$ 的结果是 False |
| != | 不相等 | $a!=b$ 的结果是 True |
| > | 大于 | $a>b$ 的结果是 False |
| >= | 大于或等于 | $a>=b$ 的结果是 False |
| < | 小于 | $a<b$ 的结果是 True |
| <= | 小于或等于 | $a<=10$ 的结果是 True |

所有比较运算符的返回值为逻辑值 True 或 False，参考表 3-2 中的示例。

### 3. 赋值运算符

赋值运算符包括简单赋值运算符和复合赋值运算符，见表 3-3。

**表 3-3　赋值运算符**

| 赋值运算符 | 含　义 | 示　例 |
|---|---|---|
| = | 简单赋值运算符 | $c=a+b$ 将 $a+b$ 的结果赋值给 $c$ |
| += | 加法赋值运算符 | $c+=a$ 等价于 $c=c+a$ |
| -= | 减法赋值运算符 | $c-=a$ 等价于 $c=c-a$ |
| *= | 乘法赋值运算符 | $c*=a$ 等价于 $c=c*a$ |
| /= | 除法赋值运算符 | $c/=a$ 等价于 $c=c/a$ |
| %= | 取模赋值运算符 | $c\%=a$ 等价于 $c=c\%a$ |
| **= | 乘方赋值运算符 | $c**=a$ 等价于 $c=c**a$ |
| //= | 整除赋值运算符 | $c//=a$ 等价于 $c=c//a$ |

### 4. 逻辑运算符

逻辑运算符共有 3 个，分别是逻辑“与”、逻辑“或”和逻辑“非”，见表 3-4。

**表 3-4　逻辑运算符**

| 逻辑运算符 | 含　义 | 示例（设 $a=10, b=20$） |
|---|---|---|
| and | 逻辑与 | ($a$ and $b$)的结果是 20，表示 True |
| or | 逻辑或 | ($a$ or $b$)的结果是 10，表示 True |
| not | 逻辑非 | not($a$ and $b$)的结果是 False |

逻辑运算符一般连接关系表达式,例如,$a>0$ and $a<1$,结果为 True 或 False;也可以连接算术表达式,例如 $a$ and $b$。连接算术表达式时,结果是一个数值。结果非零时作为逻辑真,为零时作为逻辑假。

对于逻辑"与"操作,例如,$a$ and $b$,如果 $a$ 为 False,其结果为 False,否则返回 $b$ 的计算值。例如,如果 $a=10$,$b=20$,因为 $a$ 的值非零(为真),则 $a$ and $b$ 的结果为 $b$ 的值,即 20。

对于逻辑"或"操作,例如,$a$ or $b$,如果 $a$ 为非零,其结果为 $a$ 的值,否则返回 $b$ 的计算值。例如,如果 $a=10$,$b=20$,则 $a$ or $b$ 的结果为 10。

对于逻辑"非"的操作,如果 $a$ 非 0 或为 True,则返回 False;如果 $a$ 为 0 或为 False,则返回 True。

再看下面的例子:

```
>>> 0 and 7>6
0
>>> 'a' and 'b'
'b'
>>> not 0
True
```

如果判断 $x$ 是否在区间[0,10]中,可以写成 $x>=0$ and $x<=10$,也可以写成 $0<=x<=10$。

判断某个字符 $x$ 是否为大写字母,可以表示为 $x>=$'A' and $x<=$'Z',也可以写成 'A'$<=x<=$'Z'。

**5. 位运算符**

位运算是针对整数的二进制形式的每个二进制数位所做的运算。位运算符共有 6 个,分别是 &(按位"与")、|(按位"或")、^(按位"异或")、~(按位"取反")、<<(左移)和>>(右移)(见表 3-5)。其中,按位取反"~"是单目运算符,有一个操作对象;另外 5 个是双目运算符,需要两个操作对象。

**表 3-5 位运算符**

| 位 运 算 符 | 含 义 | 示例($a=60$,$b=13$) |
|---|---|---|
| & | 按位与 | $a$&$b$ 的结果是 12 |
| \| | 按位或 | $a$\|$b$ 的结果是 61 |
| ^ | 按位异或 | $a$^$b$ 的结果是 49 |
| ~ | 按位取反 | ~$a$ 的结果是-61 |
| << | 左移 | $a$<<2 的结果是 240 |
| >> | 右移 | $a$>>2 的结果是 15 |

理解位运算操作时,要先将整数写成二进制的形式,对于双目运算符,各位对齐后,分

别计算每一位，计算后的结果再转换成十进制。

例如，$a=60$，$b=13$，这两个值对应的二进制数为 $a=00111100B$，$b=00001101B$（末尾的“B”表示这是一个二进制数）。

$a\&b$ 的结果是 0000 1100B，转换成十进制是 12。

$a|b$ 的结果是 0011 1101B，转换成十进制是 61。

其他运算的结果可以参考表 3-5。

**6. 运算符的优先级别**

Python 中各类运算符的优先级别见表 3-6。

**表 3-6　运算符的优先级别**

| 运　算　符 | 优先级别（从高到低） |
|---|---|
| ** | 乘方（最高） |
| ～、+、- | 按位取反、正号、负号 |
| *、/、%、// | 乘、除、取余、整除 |
| +、- | 加法、减法 |
| >>、<< | 右移、左移 |
| & | 按位与 |
| ^、\| | 按位异或、按位或 |
| <=、<、>、>= | 比较运算符 |
| ==、!= | 相等、不相等 |
| =、%=、/=、//=、-=、+=、*=、**= | 赋值运算符、复合赋值运算符 |
| is、is not | 身份运算符 |
| in、not in | 成员运算符 |
| not、and、or | 逻辑运算符 |

## 3.2.3　输入和输出

Python 中的输入和输出可以分别通过函数 input()和 print()完成。

**1. input()函数**

input()函数的使用格式如下。

```
input([提示字符串])
```

该函数在屏幕显示提示字符串并等待用户输入，从键盘读入一行字符串后按回车键即可。

该函数返回值的类型为字符串。

例如，下面的程序：

```
x=input()
print(type(x))
```

程序运行时,如果从键盘输入 123,则显示结果为<class 'str'>,表明函数的结果是字符串类型。

如果需要输入其他类型的数据,可以使用类型转换函数进行类型转换。

例如,下面的程序输入两个整数,然后计算这两个整数的和。

```
x=int(input("请输入第一个整数"))   #input 输入; int 将输入转换为整型数
y=int(input("请输入第二个整数"))
z=x+y
print("两数之和为:",z)
```

程序的运行结果如下。

```
请输入第一个整数 3
请输入第二个整数 4
两数之和为: 7
```

函数 int()的作用是将括号内的数据的类型转换为整型。另外,float()函数可以将括号内的数据的类型转换为浮点型,例如, x=float(input("请输入一个实数"))。

**2. eval()函数**

eval()函数的参数是字符串,该函数的作用是将括号内的字符串作为一个表达式进行计算,并将计算结果作为函数的值。

例如,eval('3+4')的结果为 7。如果有一个字符串变量 $x$="3.1416 * 10 * 10",则 eval($x$)的结果是 314.16。

eval()和 input()函数配合起来可以实现从键盘输入一个算式并计算算式的结果,例如,语句 $y$=eval(input())执行后,如果从键盘输入 3+4,则变量 $y$ 的结果是 7。

下面是各种类型的转换函数和 input()函数配合使用的不同情况。

```
>>> a=input("输入一个字符串")
输入一个字符串 Hello
>>> b=int(input("输入一个整数"))
输入一个整数 4
>>> c=float(input("输入一个实数"))
输入一个实数 3.5
>>> d=eval(input("输入一个表达式"))
输入一个表达式 3 * 4
>>> e=eval(input("输入一个表达式"))
输入一个表达式 b+c
>>> print(a,b,c,d,e)
Hello 4 3.5 12 7.5
>>>
```

使用 input()函数也可以输入复数,例如,下面的语句完成了从键盘输入一个复数。

```
>>> a=eval(input())
3+4j
>>> type(a)
<class 'complex'>
```

**3. print()函数**

print()函数用来在屏幕上显示输出结果,该函数有多种使用格式。

1) 指定数据之间的分隔符和行结束符

它的一般格式如下。

```
print([输出项 1,输出项 2,…,输出项 n][,sep=分隔符][,end=结束符])
```

在上面的格式中,各个参数的含义如下。

- 前面的“输出项 1,输出项 2,…,输出项 *n*”是要输出的每一项,即输出项表列。
- sep 用来指出各个输出项之间的分隔符,如果没有写 sep,则默认为用空格分隔。
- end 表示结束符,end=''表示不换行。如果没有写 end,则默认为回车换行结束,即该行后面的 print()另起一行输出。
- []表示可选,即可以没有。

如果不使用任何参数,print()输出一个空行。

下面的语句给出了分隔符'#'和结束符'%':

```
print(10, "abcd",20, sep='#', end='%')
```

该语句的输出结果是:

```
10#abcd#20%
```

下面两条语句中第一个 print()函数指定的行分隔符是空字符串:

```
print("Hello", end='')
print("123456")
```

则两条语句在同一行输出:

```
Hello123456
```

下面两条语句中第一个 print()函数没有指定行分隔符,则默认为回车换行符:

```
print("Hello")
print("123456")
```

则两条语句分别在两行输出:

```
Hello
123456
```

2) 格式化输出

print()函数进行格式化输出时,有以下三种方法。

- 使用字符串格式化运算符%。

- 使用字符串的 format()方法。
- 使用 format()函数。

使用字符串格式化运算符%,将输出格式和输出项用%隔开,这时函数的参数格式如下。

```
格式字符串 %(输出项 1,输出项 2,…,输出项 n)
```

格式字符串要用引号括起来,各个输出项要用括号括起来。

在格式字符串中,格式字符串包含普通字符和格式说明符,可以使用不同的格式说明符来指定各个输出项的输出格式,这些格式符以%开始,例如,%c 指定输出项按字符输出,%s 指定按字符串输出,%d 指定按十进制整数输出,%o 指定按八进制整数输出,%x 指定按十六进制整数输出,%f 指定按浮点数输出。

例如,下面的语句:

```
x=10
y=20.345
print("x=%d,y=%.2f"%(x, y))
```

输出结果为

```
x=10, y= 20.34
```

上例 print()语句中的输出项为 $x$ 和 $y$,格式串中的%d 和%.2f 表示分别按整数和实数输出 $x$ 和 $y$,%.2f 中的.2 表示输出实数时小数部分输出两位。

假如 $a=65$,语句 print("%d %o %x" % (a,a,a))的输出结果是 65 101 41,语句中的同一个变量输出了 3 次,分别按十进制、八进制和十六进制输出。

使用字符串的 format()方法时,其格式为

```
格式字符串.format(输出项 1,输出项 2,…,输出项 n)
```

格式字符串包含普通字符和格式说明符,这里的格式说明符用花括号括起来,即写成下面的形式:

```
{[序号]: 格式说明符 }
```

其中,序号对应输出项的位置,位置从 0 开始,默认按自然顺序输出。0 表示第一个输出项,1 表示第二个输出项,等等。

例如,下面的语句输出变量 $x$ 和 $y$ 的值。

```
x=10
y='abc'
print('x={0},y={1}'.format(x,y))
```

输出结果为

```
x=10, y=abc
```

使用 format()函数时,要对每一个输出项分别设置输出格式,每一项的设置方法如下。

```
format(输出项[,格式字符串])
```

例如，语句 print(format(65,'c'),format(3.1298,'.2f')) 的输出结果如下。

```
A 3.13
```

语句中有两个输出项，第一个 format(65,'c')表示将 65 按字符输出，结果是编码为 65 的字符'A'；第二项 format(3.1298,'.2f')表示将 3.1298 按实数输出并且小数部分只输出两位。

关于格式说明符的详细写法，请查阅 Python 文档。

## 3.3 控制结构

程序的基本结构有顺序结构、分支结构、循环结构三种，这些结构的共同特点是都有一个入口和一个出口。任何程序都可由这三种基本结构组合而成。

### 3.3.1 顺序结构

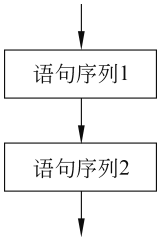


图 3-9 顺序结构

**顺序结构**是指程序按照线性顺序依次执行每条语句的一种运行方式，如图 3-9 所示，其中，语句序列可以是任意基本程序结构的语句组合。

Python 中同一级别(缩进量相同的)的语句从上到下顺序执行。

**【例 3-1】** 输入圆的半径，计算圆的面积和周长。

**【解】** 程序如下。

```
R=float(input("请输入圆半径:"))
S=3.14*R*R
L=2*3.14*R
print("圆的面积=%.2f, 周长=%.2f"%(S,L))
```

程序的运行结果如下。

```
请输入圆半径: 10.0
圆的面积=314.00, 周长=62.80
```

其中的 10.0 是运行后输入的半径的值。

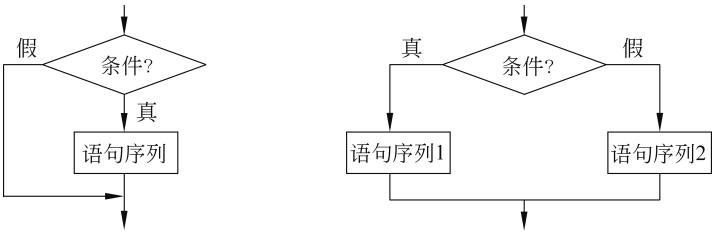
### 3.3.2 分支结构

**分支结构**是程序根据判断条件，选择不同的执行路径。分支结构包括单分支结构、二分支结构和多分支结构，单分支和二分支结构的流程图如图 3-10 所示。

if 语句用来实现分支结构，根据给出的条件是否成立进行选择，有三种使用形式。

**1. 没有 else 的 if 语句(单分支)**

这种形式的 if 语句格式如下。




(a) 单分支　　(b) 二分支

图 3-10　分支结构

```
if  <条件>:
      <语句块>        #语句块缩进对齐
<其他语句>
```

格式中的<条件>通常是关系表达式或逻辑表达式,也可以放在圆括号中,当<条件>为真时,执行<语句块>中的各条语句。

**【例 3-2】**　输入一个实数,如果该实数大于或等于 0,则计算并输出该数的平方根。

**【解】**　程序如下。

```
x=float(input("请输入一个实数"))
if x>=0:
    y=x**0.5
    print(x,"的平方根是:",y)
```

程序中的运算符"**"表示计算乘方,x**0.5 表示 $x$ 的 0.5 次方,也就是计算平方根。程序的运行结果如下。

```
请输入一个实数 2.0
2.0 的平方根是: 1.4142135623730951
```

**【例 3-3】**　从键盘输入两个整数,然后按从小到大的顺序输出。

**【分析】**　本题中,无论输入的顺序如何,都按从小到大的顺序输出,输入的数据保存在两个变量中,如果输入的两个数是先小后大,直接输出;否则,要先对这两个变量进行交换,然后再输出。

**【解】**　程序如下。

```
a=int(input("请输入第 1 个整数"))
b=int(input("请输入第 2 个整数"))
if(a>b):
    a,b=b,a      #交换两个变量的值
print("这两个数从小到大的输出为",a,b)
```

该程序的一次运行结果为

```
请输入第 1 个整数 4
请输入第 2 个整数 3
这两个数从小到大的输出为 3 4
```