第1章

Python语言及其编程环境

本章教学目标:

- 了解 Python 语言的特点。
- 学会 Python 编程环境的安装。
- 逐步熟悉使用一种第三方 Python 编辑器。

1.1 Python 语言概述

Python 是一种面向对象、解释型计算机程序设计语言，由 Guido van Rossum 于 1989 年发明，于 1991 年公开了第一个发行版本。

Python 语言简洁、清晰，具有丰富和强大的类库，能够把用其他语言（如 C/C++）制作的各种模块很轻松地联结在一起，又被昵称为胶水语言。Python 开发环境是纯粹的自由软件，源代码和解释器 CPython 均遵循 GPL（General Public License）协议。由于 Python 语言的简洁、优雅、开发效率高，既能快速生成程序的原型，又能方便地将 Python 程序封装成可调用的扩展类库，程序无须修改就能在 Windows，Linux，UNIX，Mac OS 等操作系统上跨平台使用，因此，它常被用于网站开发、网络编程、图形处理、黑客攻防等。根据 2019 年 8 月 TIOBE 编程语言排行榜（见图 1-1），Python 已迅速上升为第三大广泛应用的编程语言。

Aug 2019	Aug 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.028%	-0.85%
2	2		C	15.154%	+0.19%
3	4	^	Python	10.020%	+3.03%
4	3	v	C++	6.057%	-1.41%
5	6	^	C#	3.842%	+0.30%
6	5	v	Visual Basic .NET	3.695%	-1.07%
7	8	^	JavaScript	2.258%	-0.15%
8	7	v	PHP	2.075%	-0.85%
9	14	^^	Objective-C	1.690%	+0.33%
10	9	v	SQL	1.625%	-0.69%

图 1-1　2019 年 8 月 TIOBE 编程语言排行榜

Python 语言崇尚优雅、明确、简单。在其行命令编程环境中输入“import this”，就会呈现出 Tim Peters 编写的、被业界称为“Python 之禅”的编程格言，如图 1-2 所示。这些格言逐渐成为 Python 程序开发者追求“More Pythonic（更具有 Python 风格）”的指导思想。

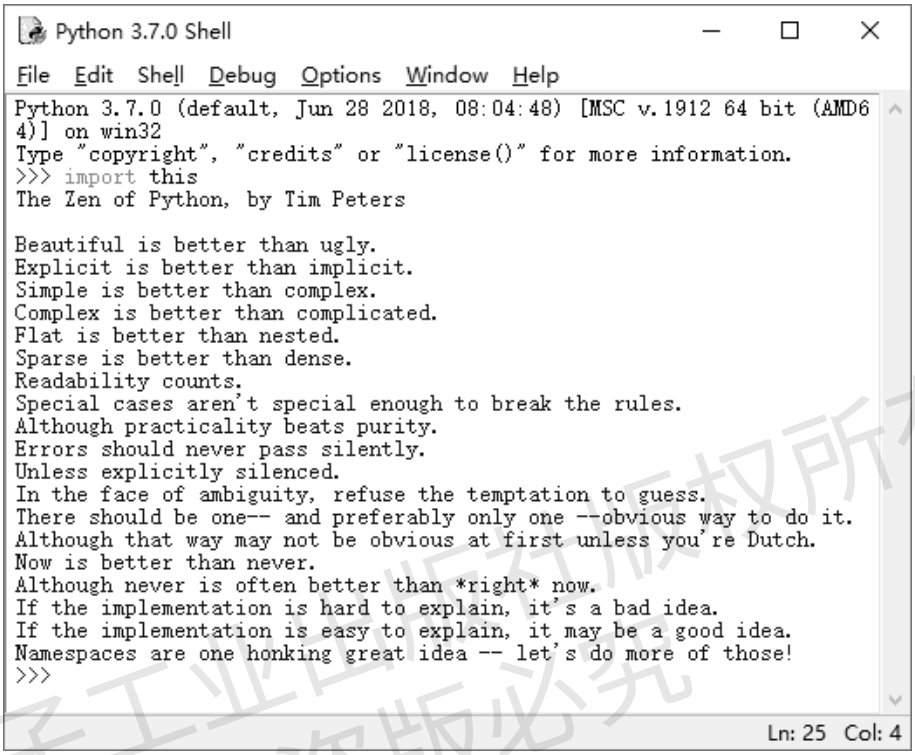


图 1-2　Python 之禅

Python 之禅的中文译意是：

- 优美胜于丑陋，显式胜于隐式。
- 简洁胜于复合，复合胜于复杂。
- 扁平胜于嵌套，稀疏胜于密集。
- 可读性很重要。
- 虽然语言的实用性胜于其纯粹性，但也不可因此而打破规则。
- 错误永远不会悄悄过去，除非明确地忽略错误。
- 面对歧义，拒绝猜测的诱惑。
- 应该有一种（最好只有一种）显而易见的解决方案。
- 解决方案最初可能并不是显而易见的，除非你就是那个荷兰人（指 Guido）。
- 马上做胜于永远不做，但盲目动手做有时还不如不做。
- 如果某个方案难以阐释，那么这通常是一个糟糕的方案。
- 如果某个方案容易解释，那么它可能是一个好主意。
- 命名空间是一个很棒的主意——我们可以多做些尝试。

1.2　Python 的安装

Python 开发环境是完全免费的自由软件，但下载安装前应考虑如下问题。

- **支持的操作系统**。Python 支持 Windows，Linux，UNIX，Mac OS 等不同操作系统，应选择对应的安装程序。
- **操作系统字长**。应根据操作系统的 32 位或 64 位字长选择对应的安装程序，以获得最佳运行环境。
- **Python 的版本**。选择 3.x 版还是 2.7 版？3.x 版与 2.x 版并不完全兼容，大批用 Python 2.x 版编写的库函数无法在 3.x 版下使用。虽然 2.x 版已经被广泛应用多年，且较为成熟，网上有大量类库资源，但 Python 2.x 版到 2.7 版后不再升级，3.x 版以后的版本升级将是 Python 语言的未来。至 2015 年年初，绝大多数 Python 语言编写的库函数都可以稳定高效地在 Python 3.x 版下运行。基于以上原因，本教材使用 3.x 版。

目前，Python 安装主要通过官网下载安装和 Anaconda 集成环境下载安装两种途径。

1.2.1　Python 的官网下载安装

最新版本的安装程序可从 Python 官网 www.python.org 免费下载，如图 1-3 所示。

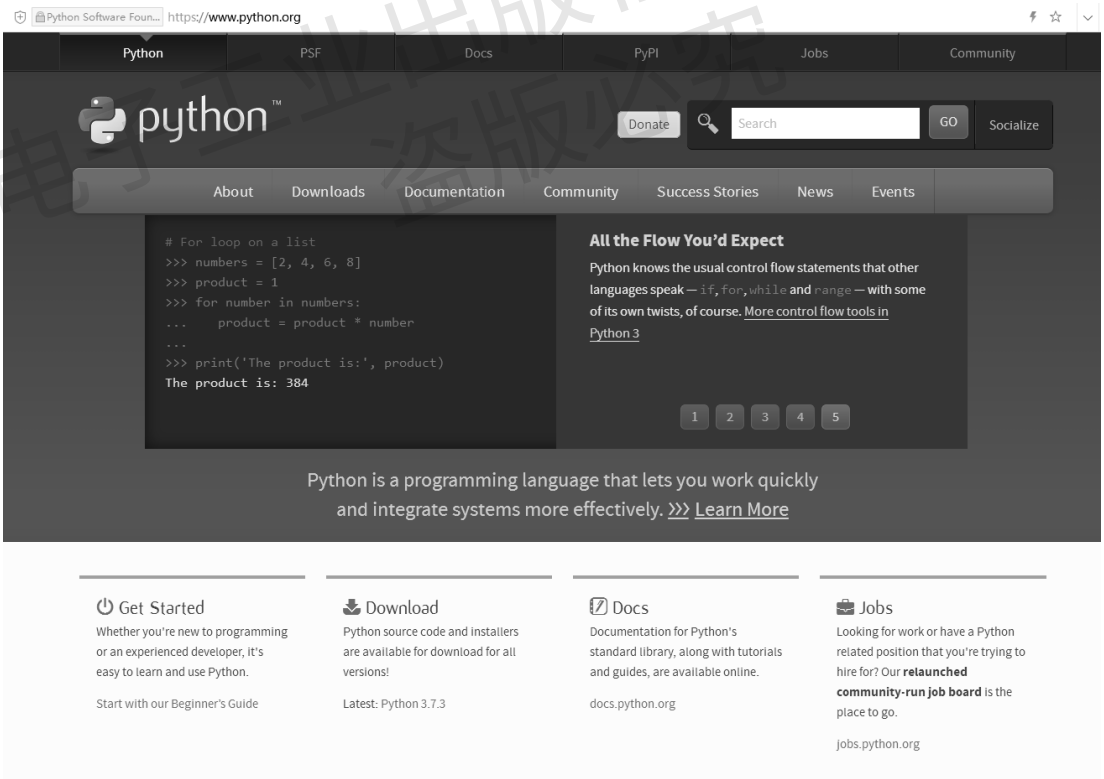


图 1-3　官网下载页面

在官网下载页面中可选择安装的操作系统和系统字长，以及安装包的形式。以 Windows 操作系统可执行程序安装为例，双击 python-3.x.x.exe 文件，即可按向导提示进行安装，如图 1-4 所示。

图 1-4　Python 安装向导

为使以后操作系统中任意路径上的 Python 程序都能正确找到安装路径，可在安装时选择“Add Python 3.7 to PATH”复选框（见图 1-4 的下部）。为方便今后对安装路径的操作，建议选择“Customize installation”将安装路径（Customize install location）设置为“C:\Python37”。

添加 Python 安装路径也可通过对操作系统环境变量的设置实现，步骤为：打开控制面板主页，单击“高级系统设置”项，在“系统属性”对话框中，单击“高级”选项卡中的“环境变量”按钮，在“环境变量”对话框的“系统变量”列表框中，选择“Path”项，单击“编辑”按钮，在打开的对话框中添加安装路径和脚本工具安装路径（如“C:\Python37”和“C:\Python37\Scripts”），如图 1-5 所示。

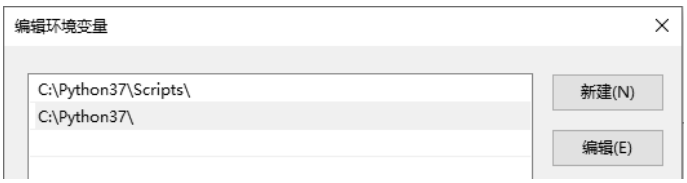


图 1-5　向操作系统环境变量中添加 Python 安装路径

1.2.2　Anaconda 集成开发环境

Anaconda 是一个 Python 的科学计算集成开发环境开源发行版本，可从其官网 www.anaconda.com（见图 1-6）或其国内镜像网站 mirrors.tuna.tsinghua.edu.cn/anaconda/archive 免费下载安装。

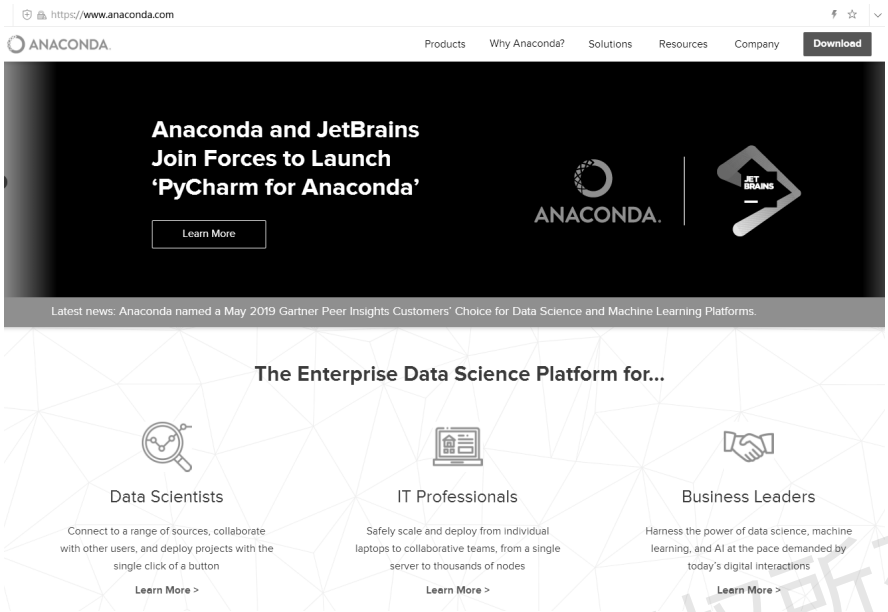


图 1-6　Anaconda 官网

Anaconda 在 Windows 操作系统中安装成功后，可见图 1-7 所示的程序组。其中常用的有第三方包管理工具 Anaconda Navigator、行命令窗口 Anaconda Prompt、交互笔记 Jupyter Notebook、编程环境 Spyder 等。

Anaconda 不仅集成了 Python 开发环境，还包含 numpy、pandas、scipy、matplotlib、PIL、NLTK 等 200 余个科学计算等常用第三方包。

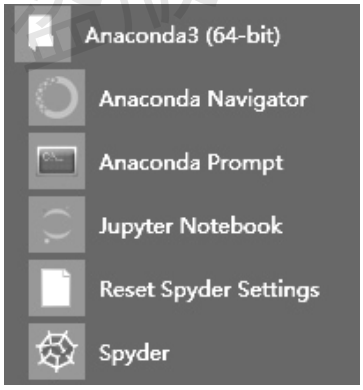
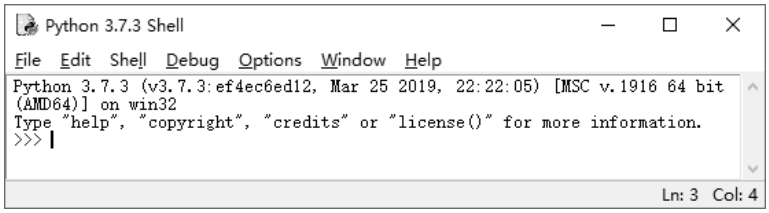


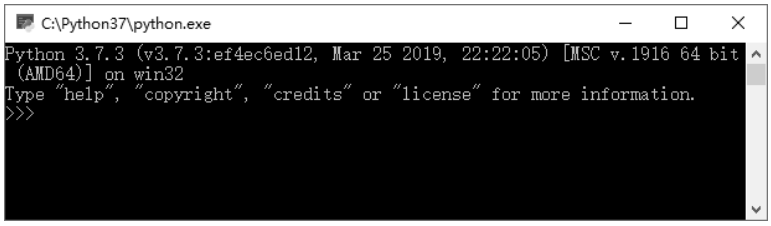
图 1-7　Anaconda 程序组

1.3　Python 程序设计步骤

Python 自带的 IDLE 运行环境（GUI Shell），如图 1-8（a）所示，其行命令运行环境如图 1-8（b）所示。



（a）Python 的 IDLE 运行环境



（b）Python 的行命令运行环境

图 1-8　Python 的运行环境

在 IDLE 运行环境中，使用“File”→“New File”菜单命令，可呼出程序编辑器，该编辑器除文本编辑功能外，还包含关键字颜色区分、简单的智能提示、自动缩进等辅助编辑功能（见图 1-9）。

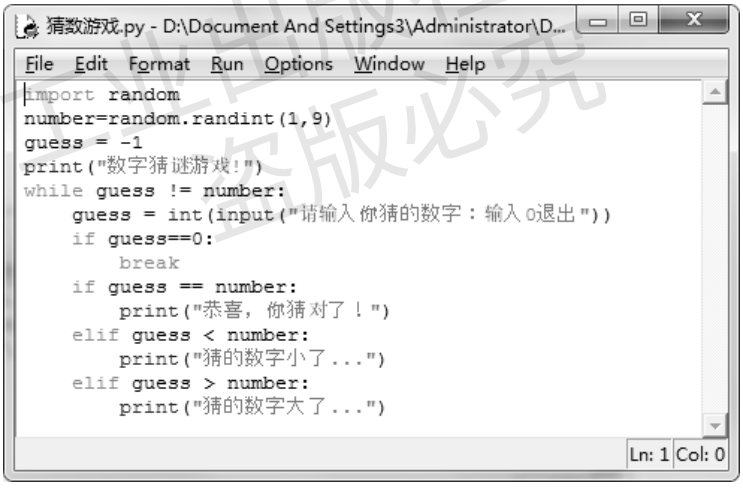


图 1-9　IDLE 的程序编辑器

Python 源程序以.py 为扩展名。当运行.py 源程序时，系统会自动生成一个对应的.pyc 字节编译文件，用于跨平台运行和提高运行速度。另外，还有一种扩展名为.pyo 的文件，是 Python 编译优化后的字节编译文件。

Python 使用缩进来表示代码块，习惯上缩进 4 个半角空格，同一个代码块的语句必须包含相同的缩进空格数，不建议随意变化缩进空格数或使用制表位键。

Python 通常是一行写完一条语句，但如果语句很长，可以使用反斜杠“\”来实现语句转行。

Python 可以在同一行中放置多条语句，语句之间使用分号“;”分隔，但为易读起见，不建议在同一行中放置多条语句。

Python 中单行注释以“#”开头。在程序调试时，如果临时需要不执行某些行，建议在不执行的行前加“#”，可避免大量删改操作。

1.4　常用的 Python 第三方编辑器

1. 记事本

Python 的源程序与其他高级语言一样，是纯文本文件，可以用操作系统自带的记事本打开和编辑（见图 1-10）。

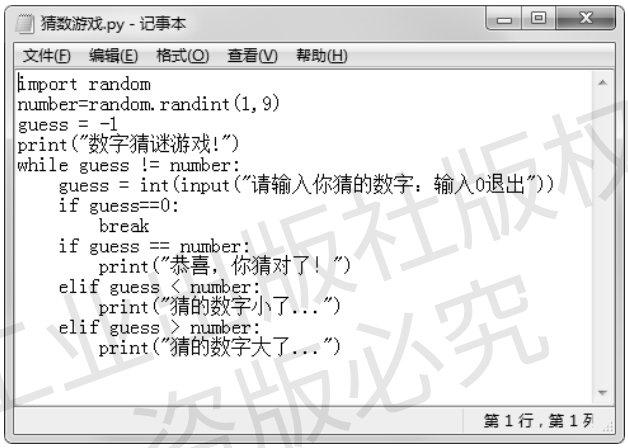


图 1-10　用记事本编写 Python 源程序

值得注意的是，记事本默认保存为 ANSI 编码的.txt 文件（关于编码，详见第 5 章），可使用“另存为”命令，在弹出的保存对话框中选择保存类型为“所有文件(*.*)”，并手工添加文件扩展名.py。在 3.x 版的 Python 源程序中，若包含中文等非英文字符，也可直接选择 UTF-8 编码方式保存（见图 1-11）。

图 1-11　Python 源程序的保存类型和编码

如果以 ANSI 编码方式保存的 Python 源程序中含有中文等非英文字符，在打开时可能

出现如图 1-12 所示的编码选择对话框，让用户确认以何种编码方式进行读取。其中，cp936 是操作系统默认的中文简体扩展字符集编码（即 GBK）。为避免在运行源程序前弹出该对话框，可在源程序最前面添加编码注释“#coding:GBK”。也可使用与 ISO-8859-1 完全兼容、几乎可以表示世界上所有字符的字符编码 UTF-8，在源程序最前面添加编码注释“#coding:UTF-8”。网上也有一些个性化的编码注释表达方式，如“_*_coding=utf-8_*_”等，Python 源程序都能兼容识别。

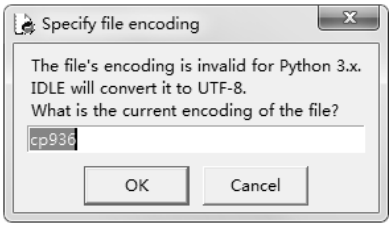


图 1-12　编码选择对话框

2. Notepad++

Python 程序员通常会选用第三方集成开发环境（Integrated Development Environment，IDE）进行程序设计。常用的集成开发环境有 Notepad++，PyScripter，PyCharm，Eclipse with PyDev，Komodo，Wing IDE 等，它们通常具有一些自动代码完成、参数提示、代码错误检查等功能。

如图 1-13 所示为 Windows 操作系统下的免费文本编辑器 Notepad++。它支持包括 Python 语言在内的许多种计算机程序设计语言，有语法高亮显示、语法折叠功能，并且支持扩充基本功能的外挂。

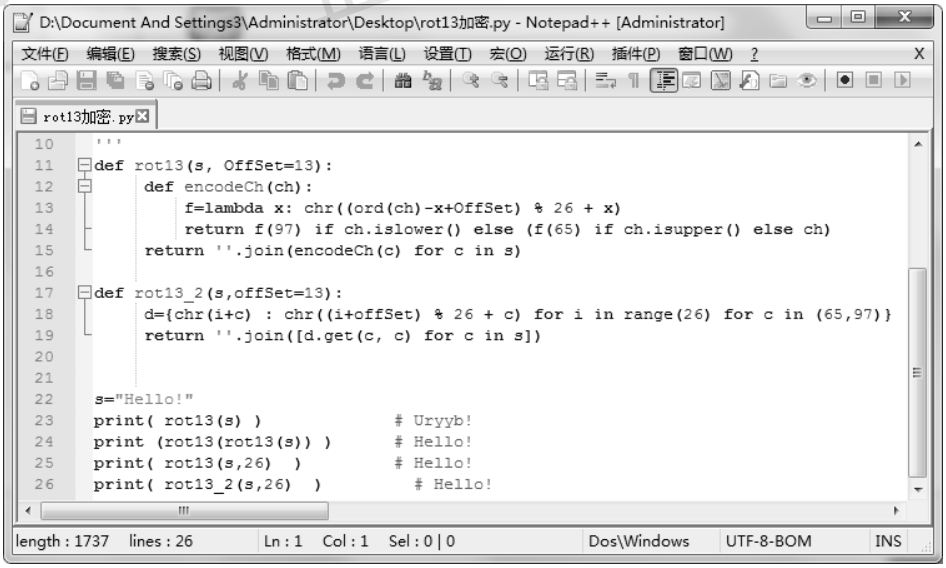


图 1-13　Notepad++编辑器

3. PyScripter

如图 1-14 所示为开源的 Python 语言集成开发环境 PyScripter，可从 https://github.com/pyscripter/pyscripter 免费下载。其具有语法高亮显示、语法自动补全、语法检查、断点调试等功能，还可以编辑 JavaScript，PHP，HTML，XML 等类型的文件。

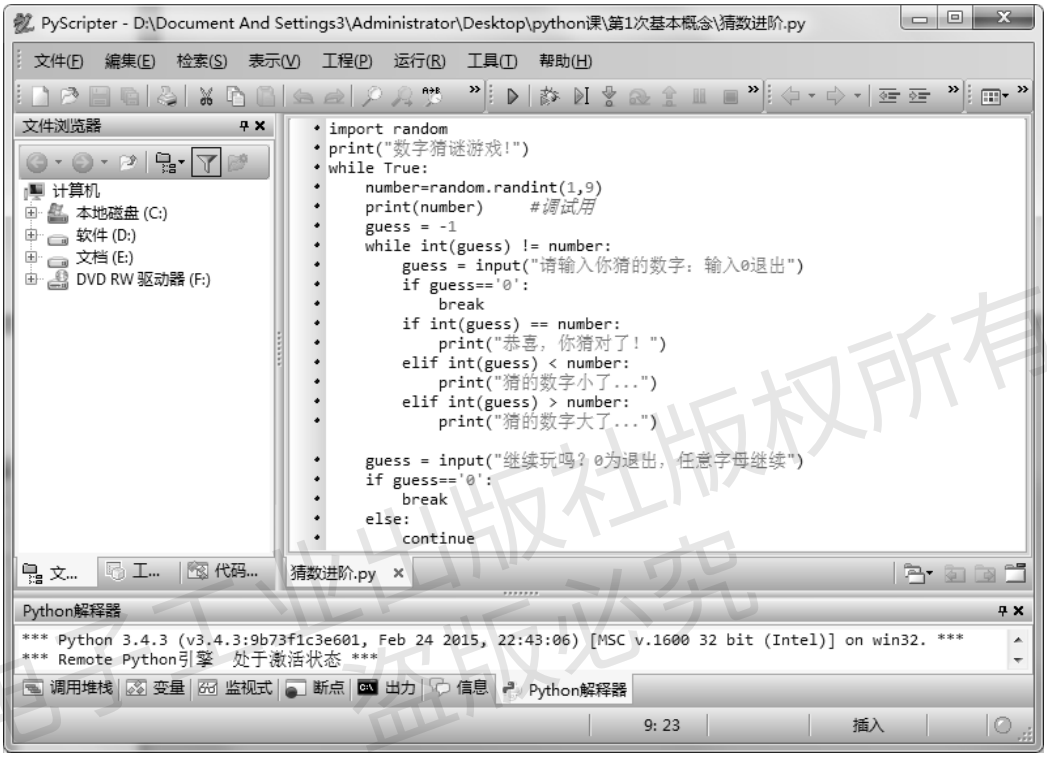


图 1-14　PyScripter 集成开发环境

4. PyCharm

如图 1-15 所示的 PyCharm 是 JetBrains 出品的专业集成开发环境，分为专业版（Professional，用于科学和 Web Python 开发，支持 HTML、JavaScript 和 SQL 语言）和社区版（Community，用于纯 Python 开发），可从 http://www.jetbrains.com/pycharm 下载。专业版试用期内免费，社区版完全免费并开源。

PyCharm 是一套 Python 程序开发的高效率工具，除具有调试、语法高亮显示、Project 管理、代码跳转、代码智能提示、代码自动完成、单元测试、版本控制等一般功能外，还提供了支持 Django 等框架下的专业 Web 开发等高级功能。尤其是快捷键映射设置，可兼容常见集成开发环境（如 Eclipse，Visual Studio，IntelliJ IDEA，Emacs，Mac OS 等）的使用习惯（见图 1-16），让使用其他语言的程序员尽快适应编程环境。

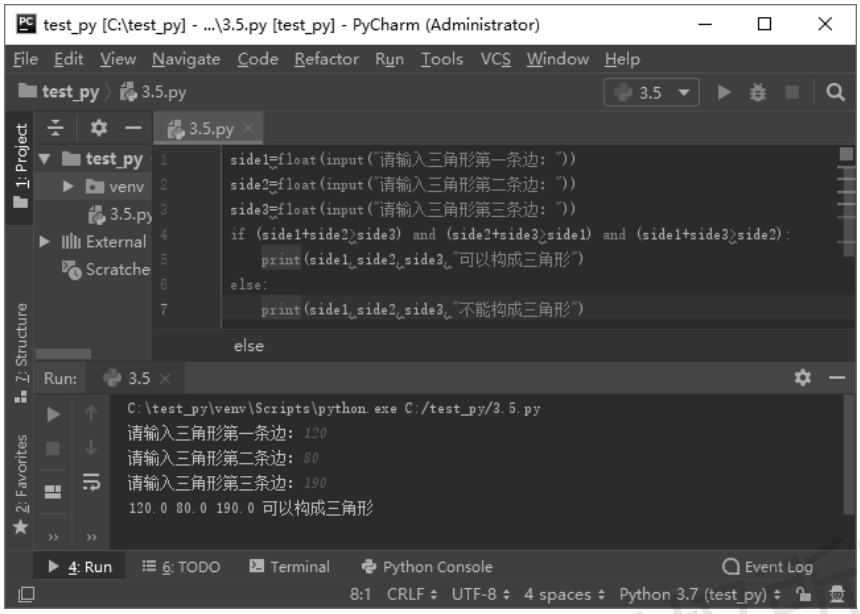


图 1-15　PyCharm 集成开发环境

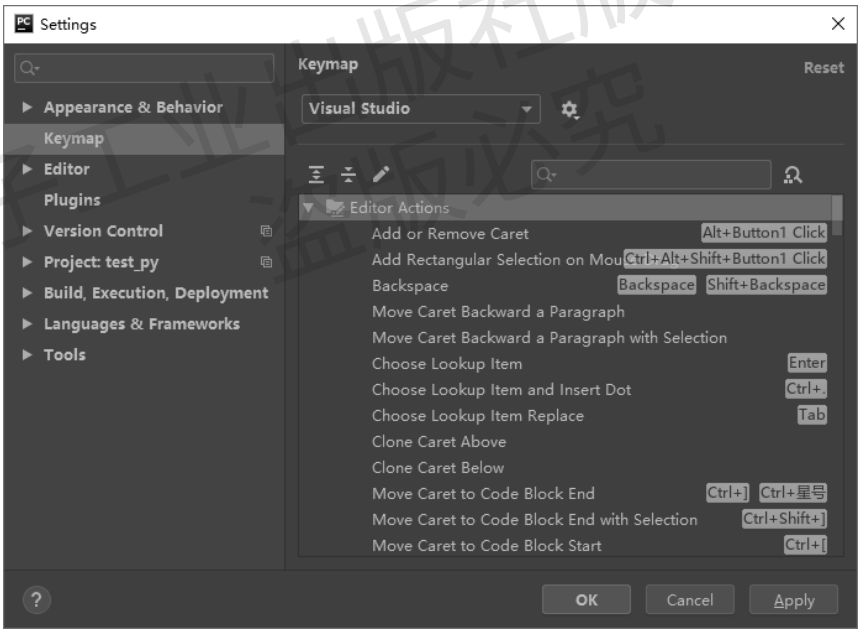


图 1-16　PyCharm 的快捷键映射设置

5. Visual Studio Code

Visual Studio Code（简称 VS Code）是微软出品的轻量级代码编辑器（见图 1-17），支持 Windows，OS X 和 Linux 操作系统。其内置了对 JavaScript 和 Node.js 的支持，并且

有丰富的插件生态系统；支持 C++，C#，Java，PHP，Python，Visual Basic，XML，R，Objective-C，JavaScript，JSON，HTML，CSS 等语言和语法结构；编辑器界面沿用了微软经典的 Visual Studio 风格，支持语法高亮显示、代码格式化、代码智能提示、括号匹配等编辑功能，并可直接访问 Git 托管平台。VS Code 可以编辑单个文件，但建议以文件夹方式打开和编辑项目。

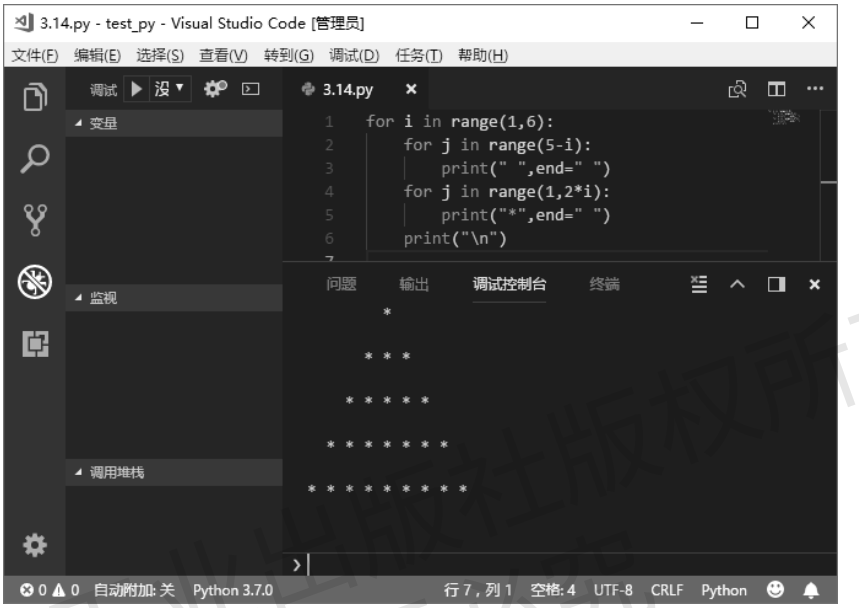


图 1-17　VS Code 集成开发环境

习题 1

1．下载并安装 Python 3.x，检查系统变量 Path 中的安装路径，体验并编写一个简单的 Python 程序。

2．下载并安装一种第三方 IDE，并逐渐熟悉使用它。

获取本章资源

第 2 章

Python的基本语法

本章教学目标:

- 熟悉 Python 语言的基本语法。
- 理解数值数据类型的特点及其操作方法。
- 初步掌握字符串数据类型。
- 逐步熟悉 Python 的基本运算、表达式和优先级。

相比其他大多数程序设计语言，Python 语言的语法更为简洁。本章我们来学习 Python 的基本语法。

2.1 Python 源程序的书写格式与基本规则

2.1.1 基本词法单位、标识符/常量/运算符等的构成规则与关键字

Python 的基本词法单位包括常量、变量、关键字、运算符、表达式、函数、语句、类等。

常量是指初始化（第一次赋予值）后就保持固定不变的值。例如：1，3.14，'Hello!'，False，这 4 个分别是不同类型的常量。关于基本数据类型会在 2.2 节中详述。

在 Python 中没有命名常量，通常可用一个固定值的变量代替。例如，PI=3.14 可表示圆周率π。在一些库中定义了专门表示固定意义的变量对象，例如，math.pi 是数学库中的圆周率，tkinter.END 是图形界面库的文本末尾。

标识符用于标识不同的词法单位，通俗地讲就是名称。标识符可以作为变量、函数、类的名字。合法的标识符必须遵守以下规则。

- 由一串字符组成，字符可以是任意字母、数字、下画线、汉字，但这串字符中的开头字符不能是数字。
- 不能与关键字同名。关键字也称为“保留字”，是被语言保留起来具有特殊含义的词，不能再用于定义名称。

【例 2-1】 查看 Python 语言的所有关键字。

```
>>> import keyword
>>> keyword.kwlist
```

```
['False', 'None', 'True', 'and', 'as', 'assert', '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']
```

在标识符中唯一能使用的标点符号只有下画线，不能含有其他标点符号（包括空格、括号、引号、逗号、斜线、反斜线、冒号、句号、问号等）。

例如，“x”“var1”“FirstName”“stu_score”“平均分 2”等，都是合法的标识符；但是，“stu-score”“First Name”“2 班平均分”都是不合法的标识符。

变量是指在运行过程中值可以被修改的量。变量的名称除必须符合标识符的构成规则外，要尽量遵循一些约定俗成的规范。

- 除循环控制变量可以使用 i 或者 x 这样的简单名称外，其他变量最好使用有意义的名称，以提高程序的可读性。例如，表示平均分的变量应使用 average_score 或者 avg_score，而不建议用 as 或者 pjf。直接用汉字命名也是可以的，但限于输入烦琐和编程环境对汉字兼容等因素，习惯上很少使用。
- 用英文名称时，多个单词之间为表示区隔，可以用下画线来进行连接，或者把每个单词的首字母大写。
- 用于表示固定不变值的对象名称一般用全大写英文字母，例如，PI，MAX_SIZE。变量名称一般使用大小写混合的方式。
- 因为以下画线开头的变量在 Python 中有特殊含义，所以，自定义名称时，一般不用下画线作为开头字符。

此外，还要注意 Python 标识符是严格区分大小字母的。也就是说，Score 和 score 会被认为是两个不同的名称。

运算符指示常量/变量之间进行何种运算。Python 有丰富的运算符，例如，赋值、算术、比较、逻辑等，将在 2.3 节中详述。

表达式由常量、变量加运算符构成。一个表达式可能包含多次多种运算，与数学表达式在形式上很接近。例如，1+2，2*(x+y)，0<=a<=10 等。

函数是相对独立的功能单位，可以执行一定的任务。其形式上类似于数学函数，例如，math.sin(math.pi/2)。可以使用 Python 内核提供的各种内置（built-in）函数，也可以使用标准模块（如数学库 math）中的函数，还可以自定义函数。

语句由表达式、函数调用组成。例如，x=1，c=math.sqrt(a*a+b*b)，print('Hello world!')等。另外，各种控制结构也属于语句，例如，if 语句、for 语句。

类是同一类事物的抽象。我们处理的数据都可以被看作数据对象。Python 是面向对象的程序设计语言，它把一个事物的静态特征（属性）和动态行为（方法）封装在一个结构里，称之为**对象**。例如，“张三”这个学生对象有学号、姓名、专业等属性，也有选课、借阅图书等方法。类是相似对象的抽象，或者说是类型。例如，“张三”“李四”都是 Student 类的对象，也可以说它们都是 Student 类型的。关于类和对象的更多知识，会在第 7 章中介绍。

2.1.2 源程序的书写格式与基本规则

Python 源程序的书写格式有严格的要求，不按照格式书写有可能导致程序不能正确运行，例如，代码缩进必须按照代码块层次要求。在《Google 开源项目风格指南》中，还列出了一些常见的书写格式基本规则建议。虽然并不影响程序执行结果，但良好的编程风格会显著提升程序的可读性。

1. 缩进

Python 语言使用缩进来区分代码块的级别。Python 语言中没有采用花括号或 begin-end 等来分隔代码块，而是使用冒号和代码缩进区分代码之间的层次。代码缩进是 Python 语言重要的语法规则，错误的缩进可能导致代码的含义完全不同。例如：

```
x = 0              # 把 0 赋值给变量 x
if x == 1:         # 判断 x 是否等于 1。如果是，则执行代码块内的两条语句
    x = x + 1      # x 在原来数值基础上再加 1
    print(x)       # 输出变量 x 的值
```

执行结果没有任何输出。因为 print(x)位于 if 代码块内，和 x=x+1 是同一个级别的，所以都没有被执行。

而下面这段的代码，最后一行的缩进不同：

```
x = 0
if x == 1:         # 判断 x 是否等于 1。如果是，则执行代码块内的一条语句
    x = x + 1
print(x)
```

执行结果会输出 0。因为 print(x)位于 if 代码块外，和 if 语句是一个级别的，会被执行。

建议在代码前输入 4 个空格来表示代码缩进，不推荐其他数量的空格或使用制表符的方式来完成缩进。

部分 Python 编辑器（如 IDLE，Notepad++，PyScripter，PyCharm 等）能根据所输入的代码层次关系自动缩进代码，提高编码效率。

2. 分号

Python 语言允许在行尾加分号，但不建议加分号，也不要用分号将两条语句放在同一行中。建议每条语句单独一行。

3. 长语句行

除非遇到长的导入模块语句或者注释里的 URL，建议每行不宜超过 80 个字符。

对于超长语句，允许但不提倡使用反斜杠连接行。建议在需要的地方使用圆括号来实现行连接。

例如：

```
#不推荐的写法（用反斜杠连接行）
if year % 4 == 0 and year % 100 != 0 or \
    year % 400 == 0:
    print('闰年!')
#建议的写法（用圆括号来实现行连接）
if (year % 4 == 0 and year % 100 != 0 or
    year % 400 == 0):
    print('闰年!')
```

如果一个文本字符串在一行中放不下，也可以使用圆括号来实现隐式行连接：

```
x = ('This will build a very long long '
    'long long long long long long string')
```

在注释中，即使超过 80 个字符，也要将长的 URL 或长的导入模块语句放在同一行中。

4. 圆括号

不建议使用不必要的圆括号。除非用于实现行连接，否则不要在返回语句或条件语句中使用圆括号。例如：

```
if (x):  # x 两侧的圆括号多余
    foo()

if not (x):  # x 两侧的圆括号多余
    foo()

return (x)  # x 两侧的圆括号多余
```

5. 空行

顶级定义之间空两行。在变量定义、类定义及函数定义之间，可以空两行。
类内部的方法定义之间，类定义与第一个方法之间，建议空一行。
在函数或方法中，如果有必要，可以空一行。

6. 空格

对于赋值运算符（=），比较运算符（= =，<，>，!= ，<>，<=，>=，in，not in，is，is not），布尔运算符（and，or，not）等，在运算符两边各加上一个空格，可以使代码更清晰。而对于算术运算符，可以按照自己的习惯决定，但建议在运算符两侧保持一致。例如：

```
# 建议的写法
x == 1
# 不推荐的写法
x==1
```

不建议在逗号、分号、冒号前面加空格，但建议在它们后面加空格（除了在行尾处）。

例如：

```
# 建议的写法
if x == 0:
    print(x, y)
x, y = y, x
# 不推荐的写法
if x == 0 :
    print(x , y)
x , y = y , x
```

参数列表、索引或切片的左括号前不要加空格。例如：

```
# 建议的写法
func（1）
x[1] = y[3:5]
# 不推荐的写法
func  （1）
x [1] = y [3:5]
```

当等号用于指示关键字参数或默认参数值时，不建议在其两侧使用空格。例如：

```
# 建议的写法
def average(sum, num=100): return sum/num
# 不推荐的写法
def average(sum, num = 100): return sum/num
```

不建议用空格来垂直对齐多行间的标记，因为这会成为维护的负担（适用于:，#，=等）。例如：

```
# 建议的写法
    x = 1  # 注释
    score_1 = 2  # 不整齐的注释
    dictionary = {
        "ID": 1,
        "grade": 2,
       }
# 不推荐的写法
    x = 1        # 注释
    score_1 = 2  # 整齐的注释
    dictionary = {
        "ID"    : 1,
        "grade" : 2,
       }
```

7. 注释

注释通常以“#”开始直到行尾结束。

行内注释是指和语句在同一行中的注释。行内注释应该以“#”和单个空格开始，并且应该至少用两个空格和前面的语句分开。

注释块通常后面跟着代码，且注释块应该与相关代码的缩进一致。注释块中每行以“#”和一个空格开始，注释块内段落之间以仅含单个“#”的行分隔。注释块上下方最好各空一行。

```
# 建议的写法
# 这个函数用于计算班级所有学生的平均分
#
# 例子：Avg(score2,100)

def Avg(Score,Num):
    pass
```

8. 文档字符串

文档字符串是 Python 语言独特的注释方式。文档字符串是包、模块、类或函数中的第一条语句。文档字符串可以通过对象的__doc__成员被自动提取。

在书写文档字符串时，应在其前、后使用三重双引号"""或三重单引号'''。

一个规范的文档字符串应该首先是一行概述，接着是一行空行，然后是文档字符串剩下的部分，并且应该与文档字符串的第一行的第一个引号对齐。

【例 2-2】 建议的文档字符串示例。

```
def Avg(Score, Num=100):
    """计算班级的平均分

    从 Score 中读取所有学生的成绩，逐一累加求总分，然后把总分除以人数 Num，结果就
    是平均分，返回该结果

    参数:
        Score: 记录所有学生成绩的列表
        Num: 班级总人数，默认值为 100

    返回值:
        float 类型的平均分
    """
    pass
```

文档字符串可以通过__doc__成员进行查看，也可以出现在 help()函数的结果里：

```
>>> print(Avg.__doc__)
    计算班级的平均分

    从 Score 中读取所有学生的成绩，逐一累加求总分，然后把总分除以人数 Num，结果就
    是平均分，返回该结果
```

```
    参数:
        Score: 记录所有学生成绩的列表
        Num: 班级总人数，默认值为 100

    返回值:
        float 类型的平均分

>>> help(Avg)
Help on function Avg in module__main__:

Avg(Score, Num=100)
    计算班级的平均分

    从 Score 中读取所有学生的成绩，逐一累加求总分，然后把总分除以人数 Num，结果就
    是平均分，返回该结果

    参数:
        Score: 记录所有学生成绩的列表
        Num: 班级总人数，默认值为 100

    返回值:
       float 类型的平均分
```

文档字符串通常用于提供在线帮助信息。

2.2 Python 的基本数据类型

Python 有着丰富的数据类型，其中字符串类型的详细讲解，以及列表、元组、字典等内置组合数据类型的介绍见第 4 章。本节主要介绍数值类型，并初步了解字符串类型。

2.2.1 数值类型

Python 有 4 种数值数据类型：整数（int）、浮点数（float）、布尔值（bool）、复数（complex）。使用内置函数 type(object)可以返回 object 的数据类型。内置函数 isinstance(obj, class)可以用来测试对象 obj 是否为指定类型 class 的实例。例如：

```
>>> type（1）
<class 'int'>
>>> type(1.0)
<class 'float'>
>>> type('1')
<class 'str'>
```

也可以使用函数 isinstance()来判断某个对象是否属于某个类型。例如：

```
>>> isinstance(1,int)
True
>>> isinstance('1',str)
True
```

1. 整数

整数是不带小数部分的数字，如 100，0，−273。和其他大多数编程语言不同，Python 中整数没有长度限制，甚至可以书写和计算有几百位数字的大整数。例如：

```
>>> 9**100  # 9 的 100 次方
26561398887587476933878132203577962682923345265339449597457496173909
24909013021829943846990440001
```

Python 中整数的书写支持 4 种数制：十进制、二进制、八进制和十六进制。十进制数直接用默认方式书写，而后三种则需要特殊的前缀，分别是 0b、0o、0x，其中的字母也可以用大写字母。在十六进制数中，使用 A～F 这 6 个字母来代表十进制数 10～15，换成小写字母 a～f 也是一样的。例如：

```
>>> x = 0b1010
>>> x
10
>>> y = 0o15
>>> y
13
>>> z = 0x2f
>>> z
47
```

2. 浮点数

浮点数是带小数的数字，如 4.，.5，−2.7315e2。其中 4.相当于 4.0，.5 相当于 0.5，−2.7315e2 是科学记数法，相当于-2.7315×10^2，即−273.15。

所谓“浮点”（floating-point）是相对于“定点”（fixed-point）而言的，即小数点不再固定于某个位置，而是可以浮动的。在数据存储长度有限的情况下，采用浮点表示方法，有利于在数值变动范围很大或者数值很接近 0 时，仍能保证一定长度的有效数字。

与整数不同，浮点数存在上限和下限。计算结果超出其上限和下限的范围会导致溢出错误。例如：

```
>>> 100.0**100  # 100.0 的 100 次方
1e+200
>>> 100.0**1000  # 100.0 的 1000 次方
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
```

```
    100.0**10000
OverflowError: (34, 'Result too large')
```

注意：浮点数只能以十进制数形式书写表达。

需要说明的是，计算机不一定能够精确地表示程序中书写或计算的实数。有两个原因：

- 因为存储空间有限，计算机不能精确显示无限小数，会产生误差。
- 计算机内部采用二进制数表示，但是，不是所有的十进制实数都可以用二进制数精确表示。由于机内表示浮点数的二进制位数所限，计算结果最后的二进制位被截断，产生的精度误差称为截断误差。

例如：

```
>>> 2/3
0.6666666666666666
>>> 1-2/3
0.33333333333333337
>>> 2.3+5.6
7.8999999999999995
```

3. 布尔值

布尔值就是逻辑值，只有两种：True 和 False，分别代表“真”和“假”。Python 3.x 中将 True 和 False 定义成了关键字，但实质上它们的值仍是 1 和 0，并且可以与数值类型的值进行算术运算。

下面两条语句比较左右两个值是否相等：

```
>>> 1 == 1.0
True
>>> 123 == '123'
False
```

4. 复数

复数是 Python 内置的数据类型，使用 1j 表示−1 的平方根。复数对象有两个属性，用于查看实部（real）和虚部（imag）。例如：

```
>>> (3+4j) * (3-4j)
(25+0j)
>>> (3-4j).real
3.0
>>> (3-4j).imag
-4.0

>>> a = -1
```

```
>>> b = a ** 0.5
>>> b
(6.123233995736766e-17+1j)
>>> b.real
6.123233995736766e-17
>>> b.imag
1.0
```

2.2.2　字符串类型

字符串（string）是由字符组成的序列，例如，'Python is wonderful!'，'16300240001'，'张三'，"等。其中，"表示空字符串。字符串和数字一样，都是**不可变对象**。所谓不可变，是指不能原地修改对象的内容。例如：

```
>>> a = b = 'abc'
>>> id(a)
2188185863144
>>> id(b)
2188185863144

>>> a = 'ABC'  # 修改 a
>>> id(a)  # a 引用了另一处空间
2188185865720
>>> b  # b 的内容不变，可见对字符串变量 a 的赋值，并不是原地修改
'abc'
```

1. 字符串界定符

字符串界定符用来区分字符串和其他词法单位，有以下三种形式。

① 单引号，如' '，'1+1=2'，'He said "how are you?"'。当字符串中含有双引号时，最好使用单引号作为界定符。

② 双引号，如""，"中国"，"It's my book."。当字符串中含有单引号时，最好使用双引号作为界定符。

③ 三引号，可以是连续三个单引号，也可以是连续三个双引号，如'''Hello'''，"""您好"""。其常用于多行字符串，例如，之前介绍的文档字符串。编程时三引号也可用于注释多行语句。

2. 转义符和原始字符串

转义符是一些特殊的字符。Python 用反斜杠（\）来转义字符，以便表示那些特殊字符，见表 2-1。

表 2-1　常见的转义符

转　义　符	描　　述
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格（Backspace）
\n	换行
\t	横向制表符
\v	纵向制表符
\r	回车
\f	换页
\ooo	八进制数 ooo 代表的字符，例如，\012 代表换行，因为八进制数 012 就是十进制数 10，而 10 是换行符的编码
\xhh	十六进制数 hh 代表的字符，例如，\x0a 也代表换行
\other	其他的字符以普通格式输出

以下是使用转义符的几个例子：

```
>>> print('a\tb\nc\\')
a	b
c\
>>> "\"Great!\""
'"Great!"'
>>> print("\"Great!\"")
"Great!"
```

若需要显示包含转义符的原始字符串，不让转义符生效。这就要用 r 或 R 来定义原始字符串。例如：

```
>>> print(r'\t\n')
\t\n
```

在上例中，如果不使用原始字符串，就得多次使用转义符'\\'：

```
>>> print('\\t\\r')
\t\r
```

3. 对字符串中字符的操作

字符串中的字符元素是不能用赋值操作进行改变的。Python 内置了一些对字符串中字符操作的方法，见表 2-2。

表 2-2　Python 内置的对字符串中字符操作的方法

方　法	意　义
s.lower()和 s.upper()	将 s 中所有字符元素转为小写或大写，并返回
s.split(sep=' ')	返回 s 中被 sep 分隔的字符串列表，sep 默认值为空格符
s.replace(s1,s2)	返回 s 中所有 s1 被 s2 替代的字符串
s.strip(x)	返回从 s 中去掉两端的 x 字符后的子串，但在 s 中间的 x 字符不能被去掉
a.join(s)	返回在 s 的每个元素之间插入字符串 a 的新字符串

值得注意的是，执行表 2-2 中的操作后，原字符串 s 并不发生变化。

例如：

```
>>> 'aBcD'.lower()
'abcd'
>>> 'aBcD'.upper()
'ABCD'

>>> 'ab 12 cdE fGH'.split()
['ab', '12', 'cdE', 'fGH']

>>> '第二军医大学'.replace('第二','海军')
'海军军医大学'

>>> '  甲乙 丙丁 '.strip(' ')  #通常用于去掉字符串两端的空格
'甲乙 丙丁'

>>> '-'.join('甲乙丙丁')
'甲-乙-丙-丁'
```

4. 字符串长度

使用 len()函数，可以确定字符串中包含多少个字符，即字符串的长度。例如：

```
>>> len('Abc 123!')
8
>>> len('中国')
2
>>> len('')
0
>>> len('a\nb\\c')
5
```

5. 字符串连接

利用加法运算符“+”可以让两个字符串首尾相连。例如：

```
>>> 'Python ' + 'Programming'
'Python Programming'
>>> 'He said ' + '"It\'s me!"'
```

```
'He said "It\'s me!"'
>>> print('He said ' + '"It\'s me!"')
He said "It's me!"
```

从第 2 条语句的输出可以看出，IDLE 运行环境的输出显示自动加了一个'\'，以免引起歧义。第 3 条语句的输出说明，'\'实际并不包含在连接结果里。

6. 字符串重复

利用乘法运算符“*”可以让一个字符串自身多次重复并拼接在一起。例如：

```
>>> 'bla ' * 4
'bla bla bla bla '
>>> 10 * '=*='
'=*==*==*==*==*==*==*==*==*==*='
```

2.3 Python 的基本运算和表达式

2.3.1 变量的操作

1. 变量的赋值和存储

（1）变量定义

变量定义是通过对变量第一次进行赋值来实现的，在 Python 中不需要变量定义语句。

【例 2-3】 变量定义示例。

```
==================== RESTART: Shell ====================
>>> x       # x 为未定义的变量，不能访问
Traceback (most recent call last):
  File "<pyshell#176>", line 1, in <module>
    x
NameError: name 'x' is not defined
>>> x = 1   # 对 x 的第一次赋值，也就是对 x 的定义，此后，变量 x 就存在了
>>> x
1
>>> x = 1.5 # 对 x 再次赋值，可以修改变量的值
>>> x
1.5
>>> del x   # 使用 del 语句删除变量 x，之后变量 x 就不能被访问了
>>> x
Traceback (most recent call last):
  File "<pyshell#182>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

变量必须定义之后才能访问。Python 中的变量比较灵活，同一个变量名称可以先后被赋予不同类型的值，定义为不同的变量对象参与计算。在上面的例子中，x 一开始是整型变量，之后又变成了浮点型变量。

（2）变量删除

使用 del 命令可以删除一个对象（包括变量、函数等），删除之后就不能再访问这个对象了，因为它已经不存在了。当然，也可以通过再次赋值重新定义该变量。

变量是否存在，取决于变量是否占据一定的内存空间。当定义变量时，操作系统将内存空间分配给变量，该变量就存在了。当使用 del 命令删除变量后，操作系统释放了变量的内存空间，该变量也就不存在了。

Python 具有垃圾回收机制，当一个对象的内存空间不再使用（引用计数为 0）后，这个内存空间就会被自动释放。所以 Python 不会像 C 语言那样发生内存泄漏而导致内存空间不足甚至系统死机的现象。Python 的垃圾空间回收是系统自动完成的，而 del 命令相当于程序主动地进行空间释放，将其归还给操作系统。

（3）变量引用

Python 中的变量实质是引用，其逻辑如图 2-1 所示。

图 2-1　变量引用的逻辑示意图

（4）变量修改赋值

Python 中的变量可以通过赋值来修改变量的“值”，但并不是原地址修改。例如，变量 x 先被赋值为 1，然后又被赋值为 1.5 之后的逻辑如图 2-2 所示。

由图 2-2 可见，并不是变量 x 的值由 1 变成了 1.5，而是另外开辟了一个地址空间存储对象，让 x 指向它。变量的值并不是直接存储在变量里，而是以“值”对象的形式存储在内存某地址中。我们可以说变量指向那个“值”对象。因此，Python 变量里存放的实际是“值”对象的位置信息（内存地址）。这种通过地址间接访问对象数据的方式，称为**引用。**

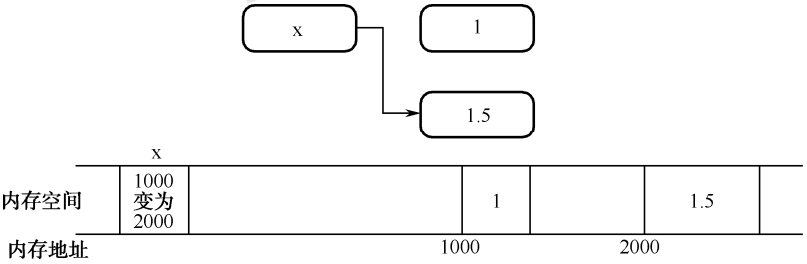


图 2-2　变量修改赋值的逻辑示意图

使用 id()函数可以确切地知道变量引用的内存地址，使用运算符 is 可以判断两个变量是否引用同一个对象。例如：

```
>>> x = 1
>>> id(x)        # 使用 id()查看 x 引用哪里
1559482096
>>> x = 2
>>> id(x)        # 再次查看，发现 x 引用的地址变了
```

```
1559482128
>>> y = 2
>>> id(y)        # 发现 y 和 x 引用同一个对象
1559482128
>>> x = 'Hello'
>>> y = 'Hello'
>>> x is y       # 利用运算符 is 可以判断两个变量是否引用同一个对象
True
```

显然，x 和 y 都被赋值为相同的小整数或者短字符串时，两个变量所引用的是同一个对象。这也被称为“驻留机制”。这是 Python 为提高效率所做的优化，节省了频繁创建和销毁对象的时间，也节省了存储空间。但是，当两个变量赋值为相同的大整数或者长字符串时，默认引用的是两个不同的对象。例如：

```
>>> x = 10**1000
>>> y = 10**1000
>>> x is y
False
>>> x = 'Good morning.'
>>> y = 'Good morning.'
>>> x is y
False
```

我们可以利用变量之间的赋值，来让两个变量引用相同的对象。例如：

```
>>> y = x
>>> x is y
True
```


2. 变量类型的转换

Python 是强类型语言。当一个变量被赋值为一个对象后，这个对象的类型就固定了，不能隐式转换成另一种类型。当运算需要时，必须使用显式的变量类型转换。例如，input()函数所获得的输入值总是字符串，有时需要将其转换为数值类型，方能进行算术运算。对于较为复杂的表达式，如果难以确定变量类型，可以用 type()函数进行测试。例如：

```
>>> x = input('请输入一个整数：')
请输入一个整数：1
>>> x
'1'
>>> type(x)
<class 'str'>
>>> int(x)
1
>>> type(int(x))
<class 'int'>
```

变量的类型转换并不是对变量原地进行修改，而是产生一个新的预期类型的对象。Python 以转换目标类型名称提供类型转换内置函数。

（1）float()函数。将其他类型数据转换为浮点数。例如：

```
>>> float（1）
1.0
>>> float('1.23')
1.23
>>> float('1.2e-3')
0.0012
>>> float('1.2e-5')
1.2e-05
```

（2）str()函数。将其他类型数据转换为字符串。例如：

```
>>> str（1）
'1'
>>> str(-1.0)  # 转换之后不会省略小数部分，因为-1 和-1.0 的类型不同
'-1.0'
>>> str(1.2e-3)
'0.0012'
>>> str(1.2e-5)
'1.2e-05'
>>> str(1.0e-5)  # 转换之后省略了'.0'，因为 1e-5 和 1.0e-5 都是浮点数
'1e-05'
```

从上述最后两个例子我们可以看出，Python 会尽可能转换成字符串长度较短的形式，以节省空间。

（3）int()函数。将其他类型数据转换为整型。例如：

```
>>> int(3.14)
3
>>> int(3.5)  # int()函数并不是四舍五入取整，而是扔掉所有小数部分
3
>>> int(True)  # 布尔值 True 相当于整数 1
1
>>> int(False)  # 布尔值 False 相当于整数 0
0
>>> int('3')
3
>>> int('3.5')  # 有的字符串不能直接转换为整型
Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    int('3.5')
ValueError: invalid literal for int() with base 10: '3.5'
>>> int(float('3.5'))  # 应该分两步转换
3
```

（4）round()函数。将浮点型数值圆整为整型。例如：

```
>>> round(1.4)
1
>>> round(1.5)  # 向上圆整
2
>>> round(2.5)  # 向下圆整为 2
2
```

圆整计算总是“四舍”，但并不一定总是“五入”。因为总是逢五向上圆整会带来计算概率的偏差，所以，Python 采用的是“银行家圆整”：将小数部分为.5 的数字圆整为最接近的偶数，即“四舍六入五留双”。

（5）bool()函数。将其他类型数据转换为布尔类型。例如：

```
>>> bool(0)     # 0 转换为 False
False
>>> bool(-1)    # 所有非 0 值转换为 True
True
>>> bool('a')   # 非空字符串转换为 True
True
>>> bool('')    # 空字符串转换为 False
False
```

数值 0 和空字符串转换为布尔类型为 False，非 0 值和非空串转换为布尔类型为 True。

（6）chr() 和 ord()函数。进行整数和字符之间的相互转换：chr()将一个整数按 ASCII 码转换为对应的字符；ord() 是 chr()的逆运算，把字符转换成对应的 ASCII 码或 Unicode 值。例如：

```
>>> chr(65)
'A'
>>> ord('a')
97
>>> ord('字')
23383
>>> chr(23383)
'字'
```

（7）eval()函数。将以一个字符串类型的算术表达式转换为其执行结果，返回表达式的值。例如：

```
>>> eval('1+2*3')
7

>>> import math
>>> eval('2.035**2+math.sin(2.3)')
4.88693021217672
```

2.3.2　运算符

Python 支持算术运算符、赋值运算符、关系运算符、逻辑运算符等基本运算符。

按照运算所需要的操作数数目，可以分为单目、双目、三目运算符。

- 单目运算符只需要一个操作数。例如，单目减（-）、逻辑非（not）。
- 双目运算符需要两个操作数。Python 中的大多数运算符是双目运算符。
- 三目运算符需要三个操作数。条件运算是三目运算符，例如，b if a else c。

运算符具有不同的优先级。我们熟知的“先乘除后加减”就是优先级的体现。只不过，Python 运算符种类很多，优先级也分成了高低不同的很多层次。当一个表达式中有多个运算符时，按优先级从高到低依次运算。

运算符还具有不同的结合性：左结合或右结合。当一个表达式中有多个运算符，且优先级都相同时，就根据结合性来判断运算的先后顺序。

- 左结合就是自左至右依次计算。Python 运算符大多是左结合的。
- 右结合就是自右至左依次计算。所有的单目运算符和圆括号()都是右结合的。实际上，圆括号是自右向左依次运算的，即内层的圆括号更优先，从内向外运算。

以上所说的通过优先级、结合性来决定运算次序，只在没有圆括号的情况下成立。使用圆括号可以改变运算符的运算次序。

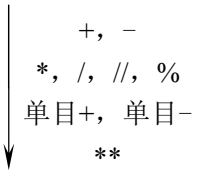
2.3.3　算术运算

Python 的算术运算符见表 2-3。

表 2-3　Python 的算术运算符

运　算　符	描　　述	实　　例
+	加法	5+2 返回 7，5.5+2.0 返回 7.5
-	减法	5-2 返回 3，5.5-2.0 返回 3.5
*	乘法	5*2 返回 10，5.5*2.0 返回 11.0
/	浮点除法	5/2 返回 2.5，5.5/2.0 返回 2.75
//	整除运算，返回商	5//2 返回 2，5.5//2.0 返回 2.0
%	整除运算，返回余数，也叫取模	5%2 返回 1，5.5%2.0 返回 1.5
**	幂运算	5**2 返回 25，5.5**2.0 返回 30.25

算术运算符的优先级，按照从低到高的顺序（同一行优先级相同）排列如下：



再看几个例子：

```
>>> x = 1
>>> -x       # -也可以作为单目运算符
-1
>>> 5 % 3
2
>>> -5 % 3  # 余数的正负号和除数一致
1
>>> 5 % -3
-1
>>> -5 % -3
-2
```

以上的例子都是在相同类型之间的数据运算。如果是不同类型之间的数据运算，会发生隐式类型转换。转换规则是：低类型向高类型转换。可以进行算术运算的各种数据类型，从低到高排列为：bool < int < float < complex。例如：

```
>>> True + 1
2
>>> True + 1.5
2.5
>>> True + 1j
(1+1j)
>>> 1 + 1.5
2.5
```

常用的 Python 数学运算类的内置函数见表 2-4。

表 2-4　常用的 Python 数学运算类的内置函数

函　数　名	描　　述	实　　例
abs	绝对值	abs(−5)返回 5，abs(−5.0)返回 5.0
divmod	取模，返回商和余数	divmod(5,2)返回(2,1)
pow	乘方	pow(5,2)返回 25，pow(5.0,2.0)返回 25.0
round	四舍五入取整	round(1.5)返回 2，round(2.5)返回 2
sum	可迭代对象求和	sum([1,2,3,4])返回 10
max	求最大值	max(3,1,5,2,4)返回 5
min	求最小值	min(3,1,5,2,4)返回 1

math 模块中的函数见表 2-5。

表 2-5　math 模块中的函数

函　数　名	描　　述	实　　例
fabs	绝对值，返回 float	fabs(−5)返回 5.0
ceil	大于等于参数的最小整数	ceil(2.2)返回 3，ceil(−5.5)返回−5
floor	小于等于参数的最大整数	floor(2.2)返回 2，floor(−5.5)返回−6
trunc	截取为最接近 0 的整数	trunc(2.2)返回 2，trunc(−5.5)返回−5
factorial	整数的阶乘	factorial(5)返回 120
sqrt	平方根	sqrt(5)返回 2.23606797749979
exp	以 e 为底的指数运算	exp(2)返回 7.38905609893065
log	对数	log(math.e)返回 1.0，log(8,2)返回 3.0

math 模块中还包含了两个数学运算中的常量。

- math.pi：数学常量 π，math.pi = 3.141592653589793。
- math.e：数学常量 e，math.e = 2.718281828459045。

使用 math 模块前要先导入，使用函数时要在函数名前面加上“math.”。例如：

```
>>> import math
>>> math.pi * math.sqrt（5）
7.024814731040727
```

如果要频繁使用某单一模块中的函数，为避免每次写模块名的麻烦，也可以按下面方式导入：

```
>>> from math import *
>>> pi * sqrt（5）
7.024814731040727
```

这样，就可以像内置函数那样来使用模块函数了。但是多个模块中可能有同名函数，如果都按这种方式导入，则会产生名字冲突的问题。

2.3.4　输入与输出

1. 输入

输入语句用于在程序运行时从输入设备获得数据。标准输入设备就是键盘。通过 input()函数可以获取键盘输入数据。一般格式为：

```
x = input(<提示字符串>)
```

input()函数首先输出提示字符串，然后等待用户键盘输入，直到用户按回车键结束，函数最后返回用户输入的字符串（不包括最后的回车符），系统继续执行 input()函数后面的语句。例如：

```
>>> name=input("请输入您的专业：")
```

系统会弹出字符串“请输入您的专业：”，等待用户输入，用户输入内容并按回车键，相应的输入内容将被保存到 name 变量中。

如果需要将输入的字符串转换为其他类型（如整型、浮点型等），调用对应的转换函数即可。

2. 输出

输出语句用于将程序的运行结果显示在输出设备上，供用户查看。标准输出设备就是显示器屏幕。一般格式为：

```
print(<输出值 1>[,<输出值项 2>, … , <输出值 n>, sep=' ', end='\n'])
```

通过 print()函数可以将多个输出值转换为字符串并且输出，这些值之间以 sep 分隔，最后以 end 结束。sep 默认值为空格符，end 默认值为换行符。

【例 2-4】 输出语句示例。

```
print('abc',123)
x = 1.5
x
print(x)
```

运行程序，输出结果为：

```
abc 123
1.5
```

上述两行输出是两个 print()函数执行的结果。本例代码第 3 条语句中的 x 并没有任何输出。这说明，只有在命令提示符>>>后面检查某个变量或表达式的值时，才能看到输出显示。而如果在.py 程序运行的模式下，必须使用 print()函数才会有输出显示。

第 1 行屏幕输出结果'abc 123'，是由本例代码第 1 条语句 print('abc',123)输出的。我们可以看出，两个输出项之间自动添加了空格，这是因为 print()函数的参数 sep 默认值为空格符。如果希望输出项之间是逗号，则可以把本例代码第 1 条语句改为：

```
print('abc',123,sep=',')
```

本例代码第 4 条语句 print(x)的屏幕输出结果是另起一行输出 1.5。这是因为 print()函数的参数 end 默认值为换行符（'\n'），所以在第 1 行输出之后自动添加了一个换行符。如果不需要换行，可以将下一个 print()函数的输出字符串直接连在其后，也可使用 end=''。如果希望不换行而是加一个逗号，则可以把第 1 条语句改为：

```
print('abc',123,sep=',',end=',')
```

修改后的程序输出结果为：

```
abc,123,1.5
```

2.3.5　赋值、关系和逻辑运算

1. 赋值运算符

赋值运算符用“=”表示，一般形式为：

```
变量 = 表达式
```

其左边只能是变量，而不能是常量或表达式。例如，5=x 或 5=2+3 都是错误的。

注意，Python 中的赋值运算是没有返回值的。也就是说，赋值操作不显示运算结果，只有效果——变量的值被改变了。

例如：

```
>>> x = 1
>>> y = x
>>> y
1
>>> x = 1.5
>>> x
1.5
>>> y
1
```

应注意的是，程序中的 y=x 不是数学中的方程等式，不代表 y 恒等于 x，不符合数学的交换律。赋值只是一个瞬间动作。

除基本赋值外，赋值运算还包括序列赋值、多目标赋值和复合赋值等（详见 2.3.7 节）。

2. 关系运算符

关系运算符也称为比较运算符，可以对两个数值类型或字符串类型的数据进行大小比较，返回一个 True 或 False 的布尔值，见表 2-6。

所有关系运算符的优先级相同。

表 2-6　关系运算符

运　算　符	描　　述	实　　例
>	大于	5>2 返回 True，'5'>'12'返回 True
>=	大于或等于	'a'>='A'返回 True，'ab'>='a'返回 True
<	小于	5<2 返回 False，'5'<'12'返回 False
<=	小于或等于	'a'<='A'返回 False，'ab'<='a'返回 False
= =	等于	5= =2 返回 False，'5'= =5 返回 False
!=	不等于	5!=2 返回 True，'5'!=5 返回 True
is	等于	5 is 2 返回 False，'5' is 5 返回 False
is not	不等于	5 is not 2 返回 True，'5' is not 5 返回 True

特别要注意的是，比较运算符是双等号“==”，而不是“=”，这是初学者常犯的错误。

在比较过程中，遵循以下规则：

① 若两个操作数是数值类型的，则按大小进行比较。

② 若两个操作数是字符串类型的，则按“字典顺序”进行比较，即：首先取两个字符串的第 1 个字符进行比较，较大的字符所在的字符串更大；如果相同，则再取两个字符串的第 2 个字符进行比较，其余类推。结果有三种情况：第一种，某次比较分出胜负，较大的字符所在字符串更大；第二种，始终不分胜负，并且两个字符串同时取完所有字符，那么这两个字符串相等；第三种，在分出胜负前，一个字符串已经取完所有字符，那么这个较短的字符串较小。第三种情况也可以认为是空字符和其他字符的比较，空字符总是最小。

常用字符的大小关系为：空字符 < 空格符 < '0'～'9' < 'A'～'Z' < 'a'～'z' < 汉字。

比较浮点数是否相等时要注意：因为有精度误差，可能产生本应相等但比较结果却不相等的情况。例如：

```
>>> a= 0.1 + 0.1 + 0.1
>>> a == 0.3
False
>>> a
0.30000000000000004
```

我们可以用两个浮点数的差距小于一个极小值来判定是否“应该相等”，这个“极小值”可以根据需要自行指定。例如：

```
>>> epsilon = 1e-6
>>> abs(a-0.3) < epsilon
True
```

复数不能比较大小，只能比较是否相等。

Python 允许 x<y<z 这样的链式比较，它相当于 x<y and y<z。甚至可以用 x<y>z，它相当于 x<y and y>z。

3. 逻辑运算符

逻辑运算符见表 2-7。

表 2-7　逻辑运算符

运　算　符	描　　述	例　　子
and	逻辑与运算符。只有两个操作数都为真，结果才为真	True and True 返回 True
or	逻辑或运算符。只要有一个操作数为真，结果就为真	False or False 返回 False
not	逻辑非运算符。单目运算符，反转操作数的逻辑状态	not True 返回 False

逻辑运算符的优先级，按照从低到高的顺序排列为：or < and < not。

or 是一个短路运算符，如果左操作数为 True，则跳过右操作数的计算，直接得出结果为 True。只有在左操作数为 False 时才会计算右操作数的值。

and 也是一个短路运算符，如果左操作数为 False，则跳过右操作数的计算，直接得出

结果为 False。只有在左操作数为 True 时，才会计算右操作数的值。

短路运算可以节省不必要的计算时间，而且 Python 会按照“最贪婪”的方式进行短路，以至于看上去跨越了优先级次序。例如：

```
>>> a,b,c = 1,2,3
>>> a == 1 or b==2 and c==3
True
```

在这个例子中，b = = 2 and c = = 3 全部被短路了，并不会因为优先级高而先计算 and。证明方法是，把上面的例子改写成下面的形式：

```
>>> def equal(x,v):  # 定义一个比较相等的函数
    print(x)  # 如果函数被执行，则先输出
    return x == v
>>> a==1 or equal(b,2) and equal(c,3)
True
```

equal()函数并没有被执行，说明 equal(b,2) and equal(c,3)全都被短路了。

2.3.6　表达式

表达式由运算符和参与运算的数（操作数）组成。操作数可以是常量、变量，可以是函数的返回值。

按照运算符的种类，表达式可以分成：算术表达式、关系表达式、逻辑表达式、测试表达式等。

多种运算符混合运算形成复合表达式，按照运算符的优先级和结合性依次进行运算。当存在圆括号时，运算次序会发生变化。

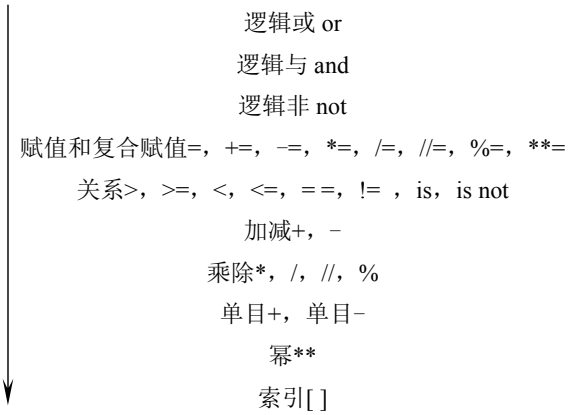
很多运算对操作数的类型有要求，例如，加法（+）要求两个操作数类型一致，当操作数类型不一致时，可能发生隐式类型转换。例如：

```
>>> x,y = 1,1.5
>>> a = x + y   # 整型和浮点型混合运算，整型隐式转换为浮点型
>>> a           # 结果为浮点型
2.5
```

差别较大的数据类型之间可能不会发生隐式类型转换，需要进行显式类型转换。例如：

```
>>> '3' + 1
Traceback (most recent call last):
  File "<pyshell#304>", line 1, in <module>
    '3' + 1
TypeError: Can't convert 'int' object to str implicitly
>>> int('3') + 1
4
>>> '3' + str（1）
'31'
```

常见运算符的优先级，按照从低到高的顺序排列（同一行优先级相同）如下：



表达式结果类型由操作数和运算符共同决定。

- 关系、逻辑和测试运算的结果一定是逻辑值。
- 字符串进行连接（+）和重复（*）的结果还是字符串。
- 两个整型操作数进行算术运算的结果大多还是整型的。浮点除法（/）的结果是浮点型的。幂运算的结果可能是整型的也可能是浮点型的，例如，5 ** -2 返回 0.04。
- 浮点型操作数进行算术运算的结果还是浮点型的。

2.3.7 赋值语句

1. 单变量赋值

使用赋值号（=）将右边的值（表达式）赋给左边变量的语句称为赋值语句。例如：

```
>>> name="张三"
>>> age=18
>>> score=82.5
>>> value=3+2j
```

上述 4 条赋值语句分别实现：为变量 name 赋值一个字符串，为变量 age 赋值一个整数，为变量 score 赋值一个浮点数，为变量 value 赋值一个复数。

2. 复合赋值

复合赋值语句是用复合运算符（包括算术复合运算符和位复合运算符）的赋值语句，包括序列赋值、多目标赋值和复合赋值等。

（1）序列赋值

例如：

```
>>> x,y = 1,1.5
>>> print(x,y)
1 1.5
```

序列赋值可以为多个变量分别赋予不同的值，变量之间用英文逗号隔开。实际上，这是利用元组和序列解包（sequence unpacking）实现的。

例如：

```
>>> first, second, third, fourth, fifth="hello"
```

上述语句的功能是分别将 5 个字符依次赋值给 5 个变量，first 的值为"h"，second 的值为"e"，其余类推。

又如：

```
>>> name, age, score, value="李四", 20, 95.5, 8+5j
```

上述语句的功能是分别将右侧的 4 个值赋值给左边的 4 个变量，name 的值为"李四"，age 的值为 20，其余类推。

Python 可以通过序列赋值语句实现两个变量值的交换。例如：

```
>>> math,english=80,75
>>> math,english=english,math
```

执行以上两条语句之后，math 与 english 的值发生了互换，math 的值为 75，english 的值为 80。

（2）多目标赋值

多目标赋值就是将同一个值赋值给多个变量。例如：

```
>>> first=second=third="welcome"
```

first, second, third 三个变量的值均为"welcome"。

多目标赋值通常只用于赋予数值或字符串这种不可变类型，如果欲赋予可变类型（如列表类型，见第 4 章），则可能会出现问题。

（3）复合赋值

复合赋值是运算操作与赋值操作的组合。所有复合赋值运算符的优先级和赋值运算符的一样。其中，+=（加等于），-=（减等于），*=（乘等于），/=（除等于），%=（取余等于），**=（幂等于），//=（整除等于）为算术复合赋值运算符（见表 2-8）。例如：

```
>>> age = 18
>>> age += 3  # 等价于 age=age+3，将 age+3 的值再赋给变量 age，age 的值为 21
>>> age /= 3  # 等价于 age=age/3，将 age/3 的值再赋给变量 age，age 的值为 7.0
```

而<<=（左移等于），>>=（右移等于），&=（与等于），|=（或等于），^=（异或等于）等运算符为位复合赋值运算符。例如：

```
>>> age = 5
>>> age<<=2 # 等价于 age=age<<2，将 age<<2 的值再赋给变量 age，age 的值为 20
>>> age&= 3 # 等价于 age=age&3，将 age&3 的值再赋给变量 age，age 的值为 0
```

表 2-8　算术复合赋值运算符

运　算　符	描　　述	实　　例
+=	加法赋值运算符	a+=b 等价于 a=a+b
-=	减法赋值运算符	a-=b 等价于 a=a-b
*=	乘法赋值运算符	a*=b 等价于 a=a*b
/=	浮点除法赋值运算符	a/=b 等价于 a=a/b
//=	整除赋值运算符	a//=b 等价于 a=a//b
%=	取模赋值运算符	a%=b 等价于 a=a%b
**=	幂赋值运算符	a**=b 等价于 a=a**b

习题 2

一、选择题

1．以下选项中不符合 Python 语言变量命名规则的是_____。

A）A　　B）break

C）_AI　　D）TempStr

2．以下选项中不符合 Python 语言变量命名规则的是_____。

A）X　　B）3_1

C）_ss　　D）InputStr

3．可以使用_____接收用户的键盘输入。

A）input 命令　　B）input()函数

C）int()函数　　D）format()函数

4．下面_____不是合法的整数数字。（多选）

A）0x1e　　B）1e2　　C）0b1001　　D）0o29

5．下面_____是非法变量名。（多选）

A）my-name　　B）complex　　C）_address　　D）'ID'

6．下面_____不是“+”的用法。

A）字符串连接　　B）算术加法

C）逻辑与　　D）单目运算

7．下面运算符优先级最高的是_____，最低的是_____。

A）and　　B）+　　C）*=　　D）==

8．下面运算结果不是浮点型的有_____。

A）2 * 0.5　　B）2 ** -1　　C）5 // 2　　D）18 / 3

9．表达式 3 * (2 + 12 % 3) ** 3 / 5 的结果是_____。

A）129.6　　B）4　　C）43.2　　D）4.8

10．数学关系表达式−1<*x*<1，表示成 Python 表达式应该是_____。（多选）

A）−1 < x < 1　　B）−1 < x and < 1

C）−1 < x and x < 1　　D）−1 < x or x < 1

11．数学表达式 *xy*/(0.5*z*)，表示成 Python 表达式应该是_____。

A）xy / 0.5 / z　　B）x * y / 0.5 z

C）x * y / 0.5 * z　　D）x * y / (0.5*z)

二、填空题

1．_____是 Python 语言中的注释符。

2．Python 语言使用_____作为转义符的开始符号。

3．Python 中的数字有 4 种数据类型：_____、_____、_____和_____。

4．判断 *n* 是偶数的 Python 表达式应为_____。

获取本章资源

第 3 章

Python程序的基本流程控制

本章教学目标:

- 了解计算思维和程序设计基本方法。
- 理解并掌握程序设计的三种基本结构。
- 熟练运用三种结构解决各种顺序、选择及重复执行的问题。
- 初步掌握程序的调试方法。

第 2 章介绍了 Python 语言的基本语法，本章将学习程序设计的三种基本结构：顺序结构、选择结构及循环结构，并尝试采用这三种基本结构来设计程序以解决问题。

3.1 计算思维和程序设计基本方法

3.1.1 计算思维

计算思维（Computational Thinking，CT）的概念由美国科学家周以真（Jeannette M. Wing）教授于 2006 年在计算机权威期刊 *Communications of the ACM* 中提出：计算思维是运用计算机科学的基础概念进行问题求解、系统设计，以及人类行为理解等涵盖计算机科学之广度的一系列思维活动。计算思维主要通过抽象、转化、仿真、迭代等方法，把一个复杂困难的现实问题转化为一个人类知道的，可以利用计算机自动解决的问题。与具备传统的“阅读、写作、算术”能力一样，计算思维能力是信息时代每个人应该拥有的基本思维能力。

人类的科学思维主要包括：实证思维、逻辑思维和计算思维。其中，实证思维以观察和归纳为特征，以物理学科为代表；逻辑思维以推演和演绎为特征，以数学学科为代表；计算思维以有限性、确定性和机械性为特征，以计算学科为代表。值得注意的是，计算思维是人的思想和方法，而不是计算机的思维方式。计算思维与逻辑思维的区别在于：计算思维的语义必须是确定性的，不能出现二义性；计算思维的结论是必须是有限的，在计算机中不允许出现数学中趋于无穷性的解；计算思维的执行方式必须是机械的，可以通过具体步骤来实现。

计算思维的本质是抽象和自动化，即在充分理解计算过程能力和限制的基础上，将生活和工作中的复杂问题选择合适的方式进行分解和化简（抽象），转化为计算机所能处理的简单

问题，并通过编写或调用程序自动解决该问题（自动化）。逻辑思维注重演绎，往往可以从原理上推演结果；而计算思维则更注重自动化实现，往往基于机械累加等简单重复步骤却能实现复杂的计算（可从例 3-15 中体会到）。

计算思维的影响已经渗透到物理、化学、生物、医学等各类不同学科，其中 Python 语言以其语法简洁、类库丰富等优点，成为计算思维在各学科中应用的一种有效工具。

3.1.2　程序设计基本方法

众所周知，现代计算机的基本结构为冯・诺依曼结构，它包括五大部分：输入设备、运算器、控制器、存储器及输出设备。程序设计遵循的基本模式为 IPO（Input、Process、Output），即程序通过输入设备输入计算机中，经过运算器、控制器及存储器的相互合作完成各类算法处理工作，最后通过输出设备输出运算结果。其中，输入方式包括交互界面输入、文件输入、网络输入等，输出方式包括屏幕输出、文件输出、网络输出等。

程序设计方法主要包括面向过程的结构化设计方法、面向对象的程序设计方法等。其中，结构化设计方法是程序设计的基本方法，它包含三种基本结构：顺序结构、分支结构及循环结构。顺序结构指程序按照语句顺序逐条执行，分支结构指程序根据不同的条件执行不同的分支语句，循环结构指程序根据特定的条件重复执行部分语句。这三种结构都具备只有一个入口和一个出口的共同特点，从而使得程序结构层次清晰、简单易懂。

在设计一个程序解决较为复杂的问题时，通常采取自上而下的设计方法，先做顶层设计，然后将复杂问题进行分解，转化为若干个可独立解决的简单的子问题，“分而治之”。每个子问题均可使用顺序结构、选择结构、循环结构或它们的组合进行描述，即基于三种基本结构、借助于某种编程语言实现简单问题的代码编写和自动执行，从而得到简单子问题的解。采用自上而下的程序设计过程，可以暂不关心过程实现的细节，可以看作对功能算法的抽象。

而在程序编写完成后，执行程序所关心的是过程自动化实现的细节。对程序的测试通常采用自下而上的执行方法，从测试运行每个包含基本结构的细节实现模块开始，逐步上升到执行整个程序。

3.2　顺序结构

程序工作的一般流程为：数据输入、运算处理、结果输出。顺序结构是指为了解决某些实际问题，自上而下地依次执行各条语句，其流程图如图 3-1 所示。

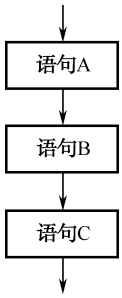


图 3-1　顺序结构流程图

例如：

```
dad = int(input("请输入爸爸的年龄："))
son = int(input("请输入儿子的年龄："))
difference = dad - son
print("爸爸与儿子的年龄差为：", difference)
```

下面通过几个例子学习使用顺序结构解决各种常见问题。

【例 3-1】 编写程序，从键盘输入语文、数学、英语三门功课的成绩，计算并输出平均成绩，要求平均成绩保留 1 位小数。

程序的执行流程为：输入三门功课成绩，计算平均成绩，输出平均成绩。输入时，使用转换函数将字符串转换为浮点数；输出时，采用格式输出方式控制小数点的位数。代码如下：

```
chinese = float(input("请输入您的语文成绩："))
math = float(input("请输入您的数学成绩："))
english = float(input("请输入您的英语成绩："))
average = ( chinese + math + english)/3
print("您的平均成绩为：%.1f" % average)
```

【例 3-2】 编写程序，从键盘输入圆的半径，计算并输出圆的周长和面积。

在计算圆的周长和面积时需要使用 π 的值，Python 的 math 模块中包含常量 pi，通过导入 math 模块可以直接使用该值，然后使用周长和面积公式进行计算即可。代码如下：

```
import math
radius = float(input("请输入圆的半径："))
circumference = 2*math.pi*radius
area = math.pi*radius*radius
print("圆的周长为：%.2f" % circumference)
print("圆的面积为：%.2f" % area)
```

【例 3-3】 编写程序，从键盘输入年份，输出当年的年历。

Python 的内置日历模块 calendar 有下列常用函数：

calendar. weekday(year, month, day)	获取指定日期为星期代码整数，0 为星期一，1 为星期二，其余类推
calendar. monthcalendar(year, month)	返回 year 年 month 月中以日期为一维元素列表，以每周日期列表为元素的二维列表
calendar. month(year, month)	以多行文本字符串格式返回 year 年 month 月的日历
calendar. calendar(year)	以多行字符串形式返回 year 年的日历

导入 calendar 模块，然后调用该模块中的 calendar 函数即可得到该年的日历。代码如下：

```
import calendar
year = int(input("请输入年份："))
table = calendar.calendar(year)
print(table)
```

3.3　分支结构

分支结构可以分为单分支结构和多分支结构，用于解决生活中形形色色的选择问题。在 Python 语言中，这两种结构分别用 if 语句和 if-elif-else 语句描述。

3.3.1　if 语句

if 语句仅处理条件成立的情况，其流程图如图 3-2 所示。从图中可以看出，当表达式的值为真时，执行相应的语句块（一条或多条语句）；当表达式的值为假时，直接跳出 if 语句，执行其后面的语句。

书写格式：关键字 if 与表达式之间用空格隔开，表达式后接英文冒号；语句块中的全部语句均缩进 4 个空格，如图 3-3 所示。

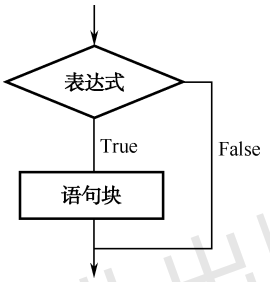


图 3-2　单分支结构流程图

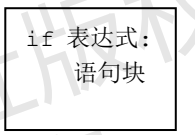


图 3-3　单分支结构书写格式

例如：

```
name = input("请输入您的姓名：")
age = int(input("请输入您的年龄："))
if age>=18:
    print(name,"已经成年")
    print("符合驾照考试规定")
```

3.3.2　if-elif-else 语句

if-elif-else 语句主要用于处理多种条件的情况，从而解决现实生活中复杂的多重选择问题，其流程图如图 3-4 所示。如果表达式 1 的值为真，则执行相应的语句块 A；如果表达式 1 的值为假，则继续判断表达式 2 的值，如果表达式 2 的值为真，则执行语句块 B；如果表达式 2 的值也为假，则继续判断表达式 3 的值；其余类推，直到所有的表达式都不满足（条件表达式的个数为 1 个或多个）为止，然后执行 else 后面的语句块。

书写格式：关键字 if 与表达式 1 之间用空格隔开，表达式 1 后接英文冒号；所有关键字 elif 均与关键字 if 左对齐，elif 与后面的各个表达式之间用空格隔开，表达式后接英文冒号；关键字 else 与关键字 if 左对齐，后接英文冒号；所有语句块左对齐，即所有语句块中的全部语句均缩进 4 个空格，如图 3-5 所示。

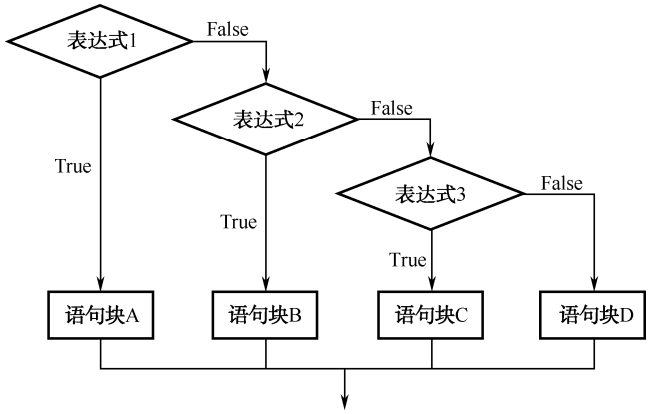


图 3-4　多分支结构流程图

```
if 表达式 1:
    语句块 A
elif 表达式 2:
    语句块 B
elif 表达式 3:
    语句块 C
else:
    语句块 D
```

图 3-5　多分支结构书写格式

例如：

```
name=input("请输入您的姓名：")
chinese=float(input("请输入语文成绩："))
math=float(input("请输入数学成绩："))
english=float(input("请输入英语成绩："))
average=(chinese+math+english)/3
if average>=85:
    print(name," 获一等奖")
elif average>=75:
    print(name," 获二等奖")
elif average>=60:
    print(name," 获三等奖")
else:
    print(name,"没有获奖")
```

如果只考虑一种表达式成立或不成立的结果（即没有 elif 分支），则多分支的 if 结构转化为双分支的 if 结构。

例如：

```
name=input("请输入您的姓名：")
score=float(input("请输入您的成绩："))
if score>=60:
    print(name,"通过考试")
    print("可以获得证书")
else:
    print(name,"未通过考试")
    print("不能获得证书")
```

在使用分支结构时，需要注意以下事项：

① 表达式可以是任意类型，如 5>3，x==y，x and y>z，3，0 等。其中，3 表示恒真（即 True），而 0 表示恒假（即 False）。

② 可以仅有 if 子句，构成单分支结构，但是 else 子句必须与 if 子句配对，不能出现仅有 else 子句没有 if 子句的情况。

③ 各语句块可以是一条或多条语句。如果是多条语句，则所有语句必须左对齐。

【例 3-4】 编写程序，从键盘输入一个整型数字，判断该数字是否为偶数。

判断一个整数是否为偶数的方法是用该数字对 2 取余数。如果结果等于 0，则该数字为偶数。代码如下：

```
number=int(input("请输入一个整型数据："))
if number%2==0:
    print(number,"是一个偶数")
```

【例 3-5】 编写程序，从键盘输入三条边，判断是否能够构成一个三角形。如果能，则提示可以构成三角形；如果不能，则提示不能构成三角形。

组成三角形的条件是任意两边之和大于第三边。如果条件成立，则能构成三角形；当条件表达式中的多个条件必须全部成立时，条件之间使用 and 运算符连接起来。

代码如下：

```
side1=float(input("请输入三角形第一条边："))
side2=float(input("请输入三角形第二条边："))
side3=float(input("请输入三角形第三条边："))
if (side1+side2>side3) and (side2+side3>side1) and (side1+side3>side2):
    print(side1,side2,side3,"可以构成三角形")
else:
    print(side1,side2,side3,"不能构成三角形")
```

【例 3-6】 编写程序，调用随机函数生成一个 1～100 之间的随机整数，从键盘输入数字进行猜谜，给出猜测结果（太大、太小、成功）的提示。

Python 的内置随机数模块 random 有下列常用函数：

`random.random()`	生成一个半开区间`[0.0,1.0)`内的浮点数
`random.ranint(start,stop)`	生成一个闭区间`[start,stop]`内的整数
`random.randrange(start,stop[,step])`	随机返回一个 `range(start,stop[,step])`中的整数
`random.choice(seq)`	随机从序列 `seq`（字符串、元组、列表）中挑选并返回一个元素
`random.shuffle(lst)`	将列表 `lst` 的序列随机重排（不能作用于字符串和元组）

此例中引入 random 模块，调用其中的 randint(a,b)函数产生介于 a 和 b 之间的随机整数（即产生的随机数大于等于 a 且小于等于 b），然后从键盘输入一个数字与该随机数进行比较，并输出判断结果。代码如下：

```
import random
randnumber=random.randint(1,100)
guess=int(input("请输入您的猜测："))
```

```
if guess>randnumber:
    print("您的猜测太大")
elif guess<randnumber:
    print("您的猜测太小")
else:
    print("恭喜您猜对了")
```

3.3.3 分支语句嵌套

当有多个条件需要满足并且条件之间有递进关系时，可以使用嵌套的分支语句。其中，if 语句、elif 语句以及 else 语句中都可以嵌套 if 语句或者 if-elif-else 语句。

书写格式：嵌套的 if 语句要求以锯齿形缩进格式书写，从而分清层次关系。

例如，婚姻法规定，男性 22 岁为合法结婚年龄，女性 20 岁为合法结婚年龄。因此如果要判断一个人是否到了合法结婚年龄，首先需要使用双分支结构判断性别，再用递进的双分支结构判断年龄，并输出判断结果。代码如下：

```
sex=input("请输入您的性别（M 或者 F）：")
age=int(input("请输入您的年龄（1-120）："))
if sex=='M':
    if age>=22:
        print("到达合法结婚年龄")
    else:
        print("未到合法结婚年龄")
else:
    if age>=20:
        print("到达合法结婚年龄")
    else:
        print("未到合法结婚年龄")
```

【例 3-7】 编写程序，从键盘输入用户名和密码，要求先判断用户名再判断密码。如果用户名不正确，则直接提示用户名输入有误；如果用户名正确，则进一步判断密码，并给出判断结果的提示。

因为要求先判断用户名再判断密码，所以本程序的一种做法是使用嵌套的 if 语句，外层 if 语句用于判断用户名，用户名正确时进入内层 if 语句，判断密码并给出判断结果，如果用户名不正确，则直接给出错误提示。代码如下：

```
username=input("请输入您的用户名：")
password=input("请输入您的密码：")
if username=="admin":
    if password=="123456":
        print("输入正确，恭喜进入！")
    else:
        print("密码有误，请重试！")
else:
```

```
    print("用户名有误，请重试！")
```

【例 3-8】 编写程序，开发一个小型计算器，从键盘输入两个数字和一个运算符，根据运算符（+、-、*、/）进行相应的数学运算。如果不是这 4 种运算符，则给出错误提示。

因为需要根据 4 种运算符的类别执行相应的运算，所以使用多分支 if-elif-else 语句；对于除法运算而言，由于除数不能为 0，因此需要使用嵌套的 if 语句来判断除数是否为 0，并执行相应的运算。代码如下：

```
first=float(input("请输入第一个数字："))
second=float(input("请输入第二个数字："))
sign=input("请输入运算符号：")
if sign=='+':
    print("两数之和为：",first+second)
elif sign=='-':
    print("两数之差为：",first-second)
elif sign=='*':
    print("两数之积为：",first*second)
elif sign=='/':
    if second!=0:
        print("两数之商为：",first/second)
    else:
        print("除数为 0 错误!")
else:
    print("符号输入有误！")
```


3.4　循环结构

循环问题在日常生活中随处可见，例如，登录邮箱时，如果输入的用户名或密码不正确，系统将提示重新输入，直到输入正确或超过次数限制为止。while 语句用于解决此类问题。

3.4.1　while 语句

while 语句用于在满足循环条件时重复执行某件事情，其流程图如图 3-6 所示。从图中可以看出，当表达式的值为真时，执行相应的语句块（循环体），然后再判断表达式的值，如果为真，则继续执行语句块……当表达式的值为假时，检查其后面是否有 else 子句（因为可选，所以流程图中未画出），如果有，则执行 else 子句；如果没有，则直接跳出 while 语句，执行其下面的语句。

书写格式：关键字 while 与表达式之间用空格隔开，表达式后接英文冒号，else 子句与 while 子句左对齐，后接英文冒号；所有语句块左对齐，即语句块中的全部语句均缩进 4 个空格，如图 3-7 所示。

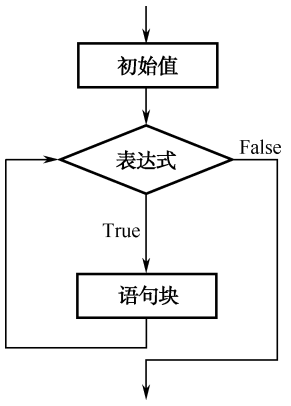


图 3-6　while 循环结构流程图

```
初始值
while 表达式:  # 表达式的值为真
      语句块 A  # 符合条件时执行的语句
else:          # 可选项
      语句块 B
```

图 3-7　while 循环结构书写格式

例如：

```
time=8
while time<12:
    print ("有效次数内")
    time=time+1
else:
    print("计次已满")
```

本例中计次的初始值 time 为 8，循环条件为计次小于 12，循环体为输出“有效次数内”并使计次加 1。每次循环均判断计次，直到计次等于 12 时，输出“计次已满”并结束循环。

在使用 while 语句时，需要注意以下事项：

① 与 if 语句类似，while 语句的表达式可以是任意类型，如 x!=y，x>3 or x<5，−5 等。

② 循环体中的语句块有可能一次也不执行。上例中，若初始值 time=13，则语句块不会执行。

③ 语句块可以是一条或多条语句。上例中，while 子句中的语句块为两条语句，else 子句中的语句块为一条语句。

④ while 循环中的 else 子句可以省略。上例中，若没有 else 子句，则当计次等于 12 时， while 语句结束，程序继续执行 while 循环后面的语句。

⑤ 程序中需要包含使循环结束的语句。上例中，若缺少语句 time=time+1，则程序无法终止。

死循环：在 while 循环中，如果表达式的值恒为真，循环将一直执行下去，无法靠自身终止，从而产生死循环。

例如：

```
while 1:
    print("Python 是一门编程语言")
```

书写程序时，有时要尽量避免死循环，但是在某些特定场合中，死循环却具有十分重要的作用，如嵌入式编程、网络编程中等。

【例 3-9】 编写程序，统计并输出 1～1000 以内所有能够同时被 3 和 7 整除的数字个数。

循环变量的初始值为 1，如果循环变量的值小于等于 1000（满足循环进行的条件），则进入循环体使用 if 语句进行判断，然后循环变量自增 1 并进入下一轮循环，循环结束后输出统计结果。代码如下：

```
number=1
count=0
while number<=1000:
    if number%3==0 and number%7==0:
        count=count+1
    number=number+1
print("同时能够被数字 3 和 7 整除的数字个数为：",count)
```

【例 3-10】 编写程序，用下列公式计算 π 的近似值，直到最后一项的绝对值小于 10^{-6} 为止。

$$\frac{\pi}{4}\approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-...$$

观察 π 的计算公式可知，循环变量的初始值为 1，循环条件为循环变量的绝对值大于等于 10^{-6}。循环变量的变化规律如上式所示，每项的分母比上一项增 2，符号与上一项的相反。代码如下：

```
import math
n=1  # 变量自增值
t=1   # 每项值
total=0   # 1/4π 的值
flag=1  # 标记位
while math.fabs(t)>=1e-6:  # 当每项值的绝对值大于 1e-6 时进行计算
    total=total+t
    flag=-flag
    n=n+2
    t=flag*1.0/n
print("π=%f" % (total*4))
```

3.4.2　for 语句和 range()内置函数

除 while 语句外，Python 还提供了另外一种功能强大的循环结构——for 语句。从可迭代对象（字符串、列表、元组、字典、迭代器等）的头部开始，依次选择每个元素并对其进行一些操作直到结束，这种处理模式被称为遍历（traversal）。for 语句用于遍历可迭代对象中的所有元素，遍历结束后可执行 else 子句（与 while 语句中的 else 子句类似，for 语句中的 else 子句是可选的）。

书写格式：关键字 for+空格+循环变量+空格+关键字 in+空格+对象（“+”的意思是后接），后接英文冒号，else 子句与 for 子句左对齐，后接英文冒号；所有语句块左对齐，即语句块中的全部语句均缩进 4 个空格，如图 3-8 所示。

```
for 循环变量 in 对象:
    语句块 A
else:  #可选
    语句块 B
```

图 3-8　for 循环结构书写格式

例如：

```
word="Hello"
for iNum in word:
    print(iNum,end=" ")  # end 值的双引号中为一个空格
```

程序运行结果为“H e l l o ”，即以空格隔开依次输出字符串"Hello"中的每个字母。

例如：

```
merge=[25,"hello",12.8,"A"]
for iNum in merge:
    print(iNum,end=" ")
```

程序运行结果为“25 hello 12.8 A ”，即以空格隔开依次输出列表中的每个元素。

例如：

```
word = '山羊上山山碰山羊角'
sum = 0
for letter in word:
    if letter == '山':
        sum += 1
print(sum)
```

程序运行结果为“4”，即以遍历方式计算出“山”在字符串中出现的次数。

for 语句经常与 range()内置函数配合使用。range()函数用于生成整数序列，通常的写法是：range(start, end, step)。其中，start 决定序列的起始值（起始值可以省略，省略时该值为 0），end 代表序列的终值（索引范围是半开区间，不包括 end 的值），step 代表序列的步长（可以省略，默认值是 1）。

例如：

```
for iNum in range(4,10,2):
    print(iNum,end=" ")
```

程序运行结果为“4 6 8”，因为索引范围是半开区间，所以不包括数字 10。

例如：

```
for iNum in range(10,2,-2):
    print(iNum,end=" ")
```

程序运行结果为“10 8 6 4”，因为步长为-2，所以输出结果依次递减。

例如：

```
for iNum in range(5):
    print(iNum,end=" ")
```

程序运行结果为“0 1 2 3 4”，起始值省略，从 0 开始；步长省略，步长为 1。

【例 3-11】 编写程序，使用 for 语句计算 1～10000 的自然数之和。

首先初始化总和的值为 0，然后使用 for 语句将 range()函数中的元素依次添加到总和中。因为 range()函数的索引范围为开区间，所以终值设为 10001。代码如下：

```
total=0
for iNum in range(1,10001,1):
    total=total+iNum
print("1~10000 的总和为：",total)
```

【例 3-12】 编写程序，解决以下问题。

4 个人中有一个人做了好事，已知有三个人说了真话，根据下面的对话判断是谁做的好事。

A 说：不是我。

B 说：是 C。

C 说：是 D。

D 说：C 胡说。

做好事的人是 4 个人其中之一，因此可以将 4 个人的编号存入列表中，然后使用 for 语句依次进行判断；有三个人说了真话，将编号依次代入，使用 if 语句判断是否满足“有三个人说了真话”（三个逻辑表达式的值为真）的条件，如果满足，则输出结果。代码如下：

```
for iNum in ['A','B','C','D']:
    if (iNum!='A')+ (iNum=='C') + (iNum=='D')+ (iNum!='D')==3:
        print(iNum,"做了好事！")
```

3.4.3　循环语句嵌套

为了解决复杂的问题，可以使用嵌套的循环语句，嵌套层数不限，但是循环的内外层之间不能交叉。其中，双层循环是一种常用的循环嵌套，循环的总次数等于内外层次数之积。

例如：

```
for i in range(1,3):
    for j in range(1,4):
        print  (i*j,end=" ")
```

当外层循环变量 i 的值为 1 时，内层循环 j 的值从 1 开始，输出 i*j 的值并依次递增，因此输出“1 2 3”，内层循环执行结束；然后回到外层循环，i 的值递增为 2，内层循环变

量 j 的值重新从 1 开始，输出 i*j 的值，并依次递增，输出“2 4 6”。因此，程序的运行结果为“1 2 3 2 4 6”。

【例 3-13】 编写程序，使用双重循环输出九九乘法表。

由于需要输出 9 行 9 列的二维数据，因此需要使用双重循环，外层循环用于控制行数，内层循环用于控制列数。为了规范输出格式，可以使用 print 语句的格式控制输出方式。其中，"\t"的作用是跳到下一个制表位。代码如下：

```
for i in range(1,10):
    for j in range(1,10):
        print("%s*%s=%2s" % (i,j,i*j), end="\t")
    print("\n")
```

【例 3-14】 编写程序，使用双重循环输出如图 3-9 所示三角形图案。

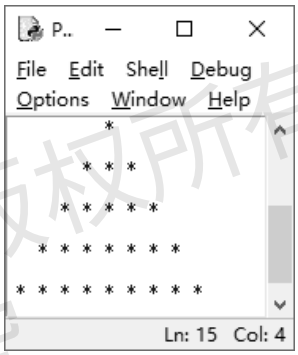


图 3-9　三角形图案

观察可知图形包含 5 行，因此外层循环执行 5 次。每行内容的由三部分组成：第一部分为输出空格，第二部分为输出星号，第三部分为输出回车，分别通过两个 for 循环和一条 print 语句实现。代码如下：

```
for i in range(1,6):
    for j in range(5-i):
        print(" ",end=" ")
    for j in range(1,2*i):
        print("*",end=" ")
    print("\n")
```

【例 3-15】 以一道“奥数”题的解题，体会计算思维与逻辑思维的差别。

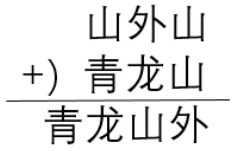


图 3-10　算术表达式

有一个以文字代替数字的算术表达式如图 3-10 所示，已知 4 个替代数字的文字中没有重复，编写程序求出文字所替代的数字。

按逻辑思维：如果 3 位数和 3 位数相加等于 4 位数，则“青”只能是 1；“山”+“青”大于等于 10，因此“山”只能是 9，得出“龙”是 0；个位的两个“山”相加，推得“外”等于 8。

而按计算思维，则注重于程序的实现，用穷举法设计嵌套的 4 层循环，把所有的数字都试一遍，找出 4 个数字不相互重复的组合满足加法等式条件。代码如下：

```
for qing in range(10):
    for long in range(10):
        for shan in range(10):
            for wai  in range(10):
                if (qing==long or qing==shan or qing==wai \
                 or long==shan or long==wai or shan==wai):
```

```
            continue
        elif (qing*1000+long*100+shan*10+wai==
              shan*100+ wai*10+shan+qing*100+long*10+shan):
            print('qing=%d,long=%d,shan=%d,wai=%d' \
                  %(qing,long,shan,wai))
            break
```

执行结果为：

```
qing=1,long=0,shan=9,wai=8
```

此例的 4 层嵌套循环的结构体现出了“自上而下”的总体设计，而具体运行却是从最内层循环“自下而上”地执行，逐步上升到执行整个程序。

3.4.4　转移和中断语句

1. break 语句

break 语句用于中断当前循环的执行，跳出循环结构。对于包含 else 子句的 while 循环和 for 循环而言，在 while 或 for 循环中一旦执行 break 语句，else 子句将没有机会执行。

【例 3-16】 编写程序，随机产生骰子的一面（数字 1～6），给用户三次猜测机会，程序给出猜测提示（偏大或偏小）。如果某次猜测正确，则提示正确并中断循环；如果三次均猜错，则提示机会用完。

使用随机函数产生随机整数，设置循环初值为 1，循环次数为 3，在循环体中输入猜测并进行判断，如果正确则使用 break 语句中断当前循环。 代码如下：

```
import random
point=random.randint(1,6)
count=1
while count<=3:
    guess=int(input("请输入您的猜测："))
    if guess>point:
        print("您的猜测偏大")
    elif guess<point:
        print("您的猜测偏小")
    else:
        print("恭喜您猜对了")
        break
    count=count+1
else:
    print("很遗憾，三次全猜错了！")
```

半路循环：前面介绍过死循环的概念，在死循环程序中，通过添加 break 语句终止程

序的执行，称为半路循环。

例如：

```
number=1
while 1:
    print("Python 是一门编程语言")
    if number>=5:
        break
    number=number+1
```

2. continue 语句

与 break 语句不同，continue 语句用于中断本次循环的执行，进入下一轮循环条件的判断。

【例 3-17】 编写程序，从键盘输入一段文字，如果其中包括“密”字（可能出现 0 次、1 次或者多次），则输出时过滤掉该字，其他内容原样输出。

从键盘输入的一段文字为字符串，可以使用 for 循环依次取出其中的每个字符，然后通过 if 语句进行判断，如果有“密”字，则使用 continue 语句跳出本次循环（不输出该字），进入下一轮循环条件的判断。 代码如下：

```
sentence=input("请输入一段文字：")
for word in sentence:
    if word=="密":
        continue
    print(word,end="")
```

【例 3-18】 编写程序，从键盘输入密码，如果密码长度小于 6，则要求重新输入。如果长度等于 6，则判断密码是否正确，如果正确则中断循环，否则提示错误并要求继续输入。

因为程序没有执行次数规定，所以循环条件设置为恒真，首先判断输入长度，如果输入长度过短，则直接使用 continue 语句中断本轮循环并进入下一轮输入；如果输入长度正确，则进行密码判断，如果正确，则使用 break 语句中断循环，否则提示错误并进入下一轮输入。 代码如下：

```
while 1:
    password=input("请输入密码：")
    if len(password)<6:
        print("长度为 6 位，请重试！")
        continue
    if password=="123456":
        print("恭喜您，密码正确！")
        break
    else:
        print("密码有误，请重试！")
```

3.5　程序调试

程序代码中出现错误是不可避免的，程序调试是将程序代码通过人工方法或使用 Python 解释器进行测试，并修改语法错误和逻辑错误的过程。

3.5.1　语法错误与逻辑错误

Python 程序错误包括语法错误和逻辑错误。

语法错误指程序代码不符合解释器语法规则，导致程序无法正常运行，如关键字拼写错误、缩进位置不正确、漏写规定符号等。语法错误通常可以在编写代码或编译时被发现，编译器就会以醒目的形式报错。含有语法错误的语句是不能通过编译的。

逻辑错误指程序代码符合语法规定，但是由于算法、运行环境等问题不能得到预期的计算结果，如遗漏重要代码段、算法使用错误、变量作用域错误、漏掉循环语句的结束条件等。逻辑错误往往能够通过编译因而难以被直接发现，需要通过人工或调试工具跟踪执行过程来排查。

3.5.2　常见语法错误

Python 语言的内置异常包含算术错误（ArithmeticError）、断言错误（AssertionError）、属性错误（AttributeError）、缓冲错误（BufferError）、结束条件错误（EOFError）、模块引入错误（ImportError）、查询错误（LookupError）、内存溢出错误（MemoryError）、对象名称错误（NameError）、操作系统错误（OSError）、引用对象错误（ReferenceError）、运行时错误（RuntimeError）、语法规则错误（SyntaxError）、系统内部错误（SystemError）、类型错误（TypeError）、赋值参数错误（ValueError）等常规异常（Exception）类型，其中许多异常类型还包含了子类型。

对于初学者，常见的语法错误包括对象名称错误、语法规则错误、查询错误、类型错误、模块引入错误、算术错误、操作系统错误、属性错误等。下面通过实例来分析常见的语法错误。

1. 对象名称错误

（1）对象名称拼写错误

变量名、函数名等对象名称的拼写错误是初学者常见的错误。并且，Python 语言对大小写敏感，即大写字母与小写字母代表不同的含义，对象引用时如果未加注意可能会导致变量未定义错误。

例如：

```
name="张三"
print(Name)
```

print 语句中变量名为“Name”而不是上面定义的“name”，程序无法通过编译，错误提示为：NameError: name 'Name' is not defined。

（2）对象名称未定义或赋值而直接使用

初学者往往可能忘记对变量赋初值就直接使用，例如：

```
a = 0
a += 1
b += 1
```

变量“b”没有初值，程序无法通过编译，错误提示为：NameError: name 'b' is not defined。

2. 语法规则错误

（1）语法符号的错漏

按 Python 语言的语法规则，在 if、elif、while、for 等子句后面需要加“:”实现条件对语句块的引导。如果用户未遵循规定，则会提示如图 3-11 所示的无效语法错误。

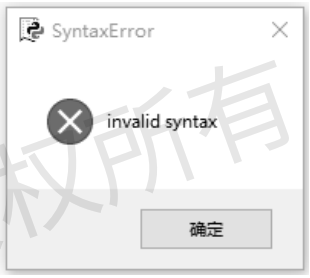


图 3-11　无效语法错误

例如：

```
for cNum in "Python"
    print(cNum)
```

for 子句最后因为缺少“:”，程序无法通过编译，将光标定位在 for 子句的后面。

在编写代码时，由于忙乱，有时会漏掉字符串的半边引号或函数（方法）的半边括号，这都属于语法规则错误，分别为“EOL while scaning string literal”和“unexpected EOF while parsing”。

（2）误将关键字作为对象名称

与其他编程语言一样，Python 不能使用关键字作为变量名，否则也会提示如图 3-11 所示的无效语法错误。

例如：

```
class='三班'
print(class)
```

由于 class 是 Python 的关键字，编译时程序报错如图 3-11 所示，并将光标定位在赋值运算符上。Python 的关键字参见第 2 章的例 2-1。

（3）赋值运算符与比较运算符的误用

Python 语言中“==”为比较运算符，用于判定左右两边是否相等，而“=”为赋值运算符，用于将右边的值赋给左边的变量。如果用户不小心用错，则会提示如图 3-11 所示的无效语法错误。

例如：

```
import random
string = '床前明月光'
number = random.randint(0,4)
guess = input('请输入您的猜测：')
if guess = string[number]:
    print('猜对了')
else:
    print('猜错了')
```

由于“==”错用为“=”，程序无法通过编译，将光标定位在“=”上。

（4）缩进错误（IndentationError）

缩进错误是一种语法规则错误。Python 语言通过缩进表明层次逻辑关系，如果未按照逻辑关系进行缩进，或缩进空格不一致、Tab 和空格混用等，编译时都会提示如图 3-12 所示的缩进语法规则错误。

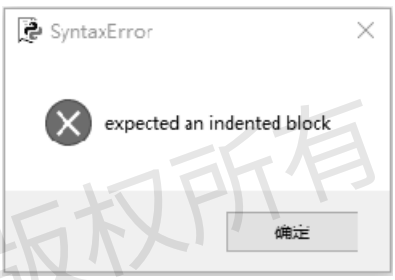


图 3-12　缩进语法错误

例如：

```
import math
if math.pi>3:
print('yes')
```

由于未按语法规则进行缩进，程序无法通过编译，将光标定位在“print”上。

3. 查询错误

（1）索引错误（IndexError）

Python 语言中，字符串、元组、列表等类型的每个元素都有相应的索引值，如果索引值超过范围，则会导致查询错误类别中的索引错误。

例如：

```
name="张三"
print(name[2])
```

由于 name 字符串索引值从下标 0 开始，name[2]元素不存在，程序无法正确执行，错误提示为：IndexError: string index out of range。

（2）映射错误（KeyError）

Python 语言的字典类型是由键和值组成的映射型组合数据类型（详见第 4 章），当程序映射中使用了字典中不存在的键时，会导致查询错误类别中的映射错误。

4. 类型错误

（1）字符串和元组为不可变数据类型，不能直接修改字符串中元素的值，否则会产生类型错误。

例如：

```
test="hello everyone."
test[0]="H"
print(test)
```

因为试图对 test[0]赋值，程序无法正确执行，错误提示为：TypeError: 'str' object does not support item assignment。

（2）字符串与数字不能直接连接，需要先对数字使用 str()函数转化为字符串，否则会产生类型错误。

例如：

```
age=18
name="张三"
merge=name+"今年"+age+"岁"
print(merge)
```

由于未将 age 转化为字符串，程序无法正确执行，错误提示为：TypeError: must be str, not int。

（3）使用 range()函数输出字符串、元组、列表中指定的元素时，需要先调用 len()函数计算字符串中元素的个数，否则会产生类型错误。（元组、列表见第 4 章。）

例如：

```
string ="123456789"
for i in range(1,string,2):
    print(string[i])
```

由于 range()函数中未计算字符串长度，程序无法正确执行，错误提示为：TypeError: 'str' object cannot be interpreted as an integer。

5. 模块引入错误

在导入模块时，如果模块名写错或者模块路径设置有问题，会导致模块引入错误。

例如：

```
import Calendar
```

因为 calendar 模块的第一个字母误写为大写字母，程序无法正确执行，错误提示为：ModuleNotFoundError: No module named 'Calendar'。

6. 算术错误

这类错误是指各种算术错误引发的内置异常，包括浮点计算错误（FloatingPointError）、溢出错误（OverflowError）和除零错误（ZeroDivisionError）。以除零错误为例，如果在计算时出现除数为 0 的情况，则会导致除零错误。

例如：

```
num1=15
num2=0
print(num1/num2)
```

因为 num2 的值为 0，程序无法正确执行，错误提示为：ZeroDivisionError: division by zero。

7. 操作系统错误

Python 语言中操作系统错误主要指文件打开错误、读写错误、操作权限不够、请求超时等。例如，文件名写错、文件路径不对、文件打开模式不对等都属于操作系统错误大类，详见第 5 章。

例如：

```
fobj=open("test.txt","r")
for line in fobj:
    print(line)
fobj.close()
```

该程序以只读方式打开文件 test.txt 并在显示器上打印（输出）其全部内容。如果文件不存在，则程序无法正确执行，错误提示为：FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'。

8. 属性错误

属性引用或赋值失败会导致属性错误。编写代码时，若方法名拼写错误也将提示为属性错误。

例如：

```
s='ABCDE'
s=s.lowerr()
print(s)
```

将大写字符串转变为小写的方法 lower()被误拼写为 lowerr()，程序编译时会提示：AttributeError: 'str' object has no attribute 'lowerr'。

3.5.3　排查程序错误的方法

语法错误大多无法通过编译，可通过报错信息直接找到。而逻辑错误或程序运行中继发的语法错误则往往需要跟踪执行过程中某些变量的值，才能发现。排查程序错误的方法有很多种，最简单的方法是在程序中插入 print()函数，输出中间值进行调试。其缺点是，当代码量很大时，该方法工作效率较低。

例如：

```
import math
a=float(input("请输入第一个系数："))
b=float(input("请输入第二个系数："))
c=float(input("请输入第一个系数："))
d=b*b-4*a*c
print(d)
x1=(-b+math.sqrt(d))/(2*a)
x2=(-b-math.sqrt(d))/(2*a)
```

在 IDLE 中，用 print()函数查错的结果如图 3-13 所示，通过输出 d 的值得知，由于其值小于 0 引发了根号里为负数的值错误。

```
请输入第一个系数：1
请输入第二个系数：2
请输入第一个系数：3
-8.0
Traceback (most recent call last):
  File "D:\test.py", line 7, in <module>
    x1=(-b+math.sqrt(d))/(2*a)
ValueError: math domain error
```

图 3-13　用 print()函数查错的结果

许多 Python 编译调试工具（如 PyCharm 等）具备断点设置、单步模式、变量查看、表达式计算等一系列调试功能，可作为高效的排查程序错误方法。

例如，用编译调试工具 PyCharm 排查程序错误（如图 3-14 所示）：

```
number=int(input("请输入一个整数："))
for i in range(2, number//2):
    if number%i==0:
        break;
if i>number//2:
    print("%d 是一个质数" % number)
else:
    print("%d 不是一个质数" % number)
```

该程序的作用是判断输入的一个整数是否为质数。当程序运行时，输入 17，结果显示了错误结果“17 不是一个质数”。为了查找程序中的逻辑错误，在程序第 3 行中设置了断点，通过单步执行依次观察 i 的值。跟踪结果发现，当 i 的值增大到 7 时，循环执行结束，由于 i 的值小于 8，输出错误的判断结果，由此可知，该程序错误的原因在于 range()函数的索引范围是一个半开区间，应该为(2, number//2+1)。

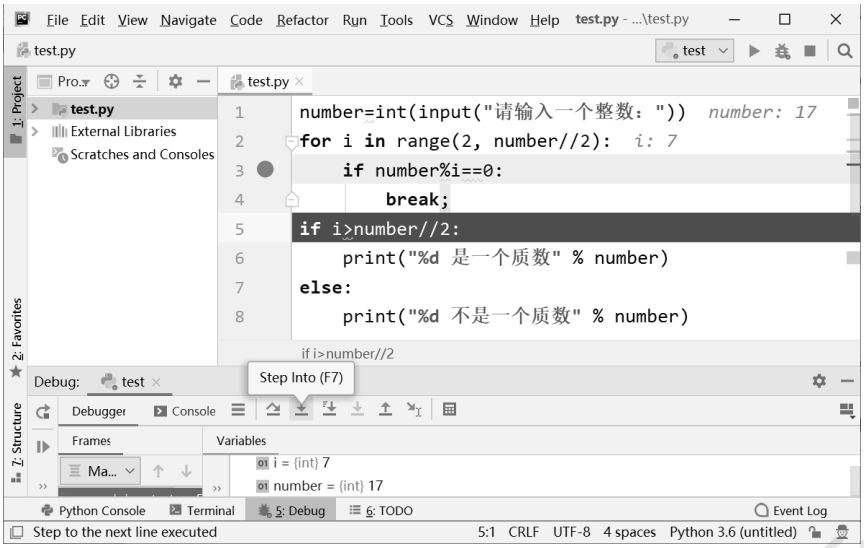


图 3-14　用编译调试工具 PyCharm 排查程序错误

3.5.4　程序运行中 try-except 异常处理

除在编写代码时应尽量保证代码正确外，一旦发现 Python 的解释器提示捕获到异常，就应该对异常进行处理，防止系统崩溃，提高系统的可靠性。异常处理语句分为 try-except 结构、try-except-else 结构、try-except-finally 结构等。

1. try-except 结构

try-except 结构是最基本的异常处理语句结构，try 子句后为正常语句块，如果其中有错误，则通过 except 子句将错误捕获并执行异常处理语句块。注意：except 子句可以有多个，但是最多只能有一个被执行。

书写格式：关键字 try 后接英文冒号；关键字 except 与 try 左对齐，except 与后面的错误名字用空格隔开，错误名字后接英文冒号；所有语句块左对齐，即语句块中的全部语句均缩进 4 个空格，如图 3-15 所示。

```
try:
    <正常语句块 A>
except <错误名字 1>:
        <异常处理语句块 B >
…
```

图 3-15　try-except-书写格式

例如：

```
try:
    fobj= open("test.txt","r")
except IOError:
    print("文件打开失败")
```

程序运行时，如果未找到该文件，Python 解释器将捕获“IOError”并输出“文件打开失败”。

2. try-except-else 结构

```
try:
    <正常语句块 A>
except <名字>:
    <异常处理语句块 B>
else:
    <正常语句块 C>
```

图 3-16　try-except-else-书写格式

try-except-else 结构在基本的异常处理语句后添加了 else 子句。try 子句后为正常语句块，如果其中有错误，则通过 except 子句将错误捕获并执行异常处理语句块；如果其中没有错误，则执行 else 子句后的正常语句块。

书写格式：关键字 else 与关键字 try 左对齐，后接英文冒号；所有语句块左对齐，即语句块中的全部语句均缩进 4 个空格，如图 3-16 所示。

例如：

```
try:
    num1=int(input("请输入分子："))
    num2=int(input("请输入分母："))
    num3=num1/num2
except ZeroDivisionError:
    print("除数为 0")
else:
    print(num3)
```

程序运行时，从键盘输入两个字符串并分别转换为整数赋值给 num1 和 num2。如果 num2 的值不为 0，计算并输出 num3 的值。如果 num2 的值为 0，Python 解释器将捕获“ZeroDivisionError”并输出“除数为 0”。

3. try-except-else-finally 结构

try-except-else-finally 结构在 try-except-else 语句后添加了 finally 子句，try 子句后为正常语句块，如果其中有错误，则通过 except 子句将错误捕获并执行异常处理语句块；如果其中没有错误，则执行 else 子句中的正常语句块；无论 try 子句中是否有错误，finally 子句终将执行。

书写格式：关键字 finally 后接英文冒号；所有语句块左对齐，即语句块中全部语句均缩进 4 个空格，如图 3-17 所示。

例如：

```
try:
    fobj= open("test.txt","r")
except IOError:
    print("文件打开失败")
```

```
try:
    <正常语句块 A>
except <名字>:
    <异常处理语句块 B>
else:
    <正常语句块 C>
finally:
    <正常语句块 D>
```

图 3-17　try-except-else-finally-书写格式

```
else:
    print("文件打开成功")
    fobj.close()
finally:
    print("文件测试结束")
```

程序运行时，如果未找到该文件，Python 解释器将捕获“IOError”并输出“文件打开失败”；否则输出“文件打开成功”并关闭文件；无论文件成功打开与否，均输出“文件测试结束”。

习题 3

1．编写程序，从键盘输入两点的坐标(x1,y1)和(x2,y2)，计算并输出两点之间的距离。

2．编写程序，从键盘输入年份值和月份值，输出该年当月的日历（调用 calendar 模块中的 month()函数）。

3．编写程序，产生两个 10 以内的随机整数，以第一个随机整数为半径，第二个随机整数为高，计算并输出圆锥体的体积。

4．编写程序，从键盘输入一个年份值，判断该年是否是闰年并输出判断结果。（提示：若该年份值能被 4 整除且不能被 100 整除或者该年份值能被 400 整除，则该年是闰年，否则不是。）

5．编写程序，从键盘输入三个数，计算并输出三个数中最大的数。

6．编写程序，从键盘输入三个数，实现三个数从小到大排序并输出结果。

7．编写程序，从键盘输入 a、b、c 的值，计算一元二次方程 $ax^2+bx+c=0$ 的根，并根据 b^2-4ac 的值大于 0、等于 0 及小于 0 分别进行讨论。

8．编写程序，从键盘输入一个字符，如果是大写字母则将其转换为小写字母，如果是小写字母则将其转换为大写字母，其他字符原样输出。

9．编写程序，从键盘输入数字 n，通过循环计算 1～n 的乘积。

10．编写程序，通过循环结构计算全部水仙花数。水仙花数是一个三位数，该数正好等于组成该三位数的各位数字的立方和。例如：$1^3+5^3+3^3=153$。

11．编写程序，通过循环结构计算并输出满足条件的正方体的体积：正方体棱长从 1 开始到 10 依次计算，当体积大于 100 时停止输出。

12．编写程序，从键盘输入一个整数并判断该数的类别：其因数之和等于数字本身的数称为完全数，比数字本身大的数称为丰沛数，比数字本身小的数称为不足数。

13．编写程序，使用双重循环结构输出如图 3-18 所示的运行结果。

```
        *
      * * *
    * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

图 3-18　第 13 题运行结果

14．编写程序，生成一个 0～100 之间的随机数，然后让用户

尝试猜测这个数字。程序给出猜测方向（更大或更小）的提示，用户继续进行猜测，直到用户猜测成功或输入一个 0～100 以外的数字后退出游戏。

15．编写程序，计算 Fibonacci 数列的前 20 项（Fibonacci 数列的特点是，第一、二项的值都为 1，从第三项开始，每项都是前两项之和）。

16．编写程序，从键盘输入两个正整数，计算两个数的最大公约数和最小公倍数。

17．编写程序，判断一个整数是否为素数（判断整数 x 是否为素数，最简单的方法就是用 2～x–1 之间的所有整数逐一去除 x，若 x 能被其中任意一个数整除，则 x 就不是素数，否则为素数）。

18．编写程序，实现一个循环 5 次计算的小游戏，每次随机产生两个 100 以内的数字，让用户计算两个数字之和并输入结果，如果计算结果正确则加一分，如果计算结果错误则不加分。如果正确率大于等于 80%，则闯关成功。

19．编写程序，从键盘输入一个姓名（可能为两个字、三个字或四个字），将该姓名的第二个汉字修改为*号。如果索引出错，则进行异常处理并提示索引错误。

20．编写程序，从键盘输入用户名和密码，判断该用户名和密码是否均在文件 information.txt 中。若在，则提示用户名和密码正确，否则提示用户名和密码错误。如果文件打开失败，则进行异常处理并提示文件打开失败，否则关闭文件。不论文件打开成功与否，最后均打印出输入的用户名和密码。

获取本章资源