Python 3.6
零基础入门与实战

王启明 罗从良 著

清華大學出版社
北京

内 容 简 介

随着大数据技术的发展及 Python 在人工智能领域的火热应用，Python 得到越来越多的应用。本书就是在这个背景下编写的，是一本 Python 3.6 入门教材，特别适合想直接切入爬虫编程及大数据分析处理的读者学习使用。本书赠送示例源代码与教学视频。

本书分为 16 章，主要内容包括开发环境、数据结构、函数、面向对象、多线程、模块、包、GUI 模块、图形模块、正则模块、文件处理模块、网络编程模块和爬虫模块等，并且为每个模块提供了实战示例，最后用两章来介绍数据库编程实战和爬虫框架实战。

本书内容详尽、示例丰富，适合广大 Python 入门读者和 Python 开发人员阅读，同时也可作为高等院校和培训学校计算机相关专业的师生教学参考。

图书在版编目（CIP）数据

Python 3.6 零基础入门与实战 / 王启明，罗从良著. —北京：清华大学出版社，2018
ISBN 978-7-302-50930-1

Ⅰ.①P…　Ⅱ.①王… ②罗…　Ⅲ.①软件工具－程序设计　Ⅳ.①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 190116 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市少明印务有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**20.25　　**字　　数：**518 千字
版　　次：2018 年 10 月第 1 版　　**印　　次：**2018 年 10 月第 1 次印刷
定　　价：59.00 元

产品编号：078325-01

前　言

不管你从事的是什么行业，进行数据分析也好，开发网页也好，做数据库后台编程也好，做股票分析也好，Python 都是你必须学会的一门语言。市场上有很多图书，随 Python 版本的升级会显得比较旧，讲解方式也比较费解。本书选择比较新的 Python 3.6.4 版本用初学者容易上手的示例学习方法进行讲解，全书逻辑线索清晰，方便读者轻松入门。

本书特色

如何快速学习 Python 编程一直是很多初学者的疑问，网上的资料很多，但不系统，很多系统的教程又过于偏重讲解，示例较少，让初学者很难坚持。因此，对于很多入门读者，更好的方式是先学习基础的 Python 语法，然后学习各种常见模块，最后在实践中完善代码编写技巧。学习过程中贯穿大小示例，方便读者对知识点做实践，基于这种想法，笔者编写了本书。本书特色如下：

1. 上手门槛低，完全无基础也可入门

作为入门图书，不会涉及计算机原理、操作系统等枯燥内容，读者可以没有这方面的基础，本书提供详细的开发环境搭建步骤及编程技巧讲解，手把手指导读者入门 Python。

2. 多个操作系统版本介绍，Linux、Windows、MacOS 都可以轻松学习

当前流行的操作系统各异，有些读者喜欢 Linux，有些公司提供 MacOS，更多的是常见的 Windows，本书很多案例都会提供不同操作系统的介绍，让读者了解 Python 的跨平台特性，在任何操作系统下都可轻松学习。

3. 多个上手小示例，几乎每章最后都有应用实战，让读者综合练习，学完就会

读者学会了 Python 语法，只是了解了如何写 Python 代码，但是如何用 Python 解决问题却需要很多项目来练手。本书几乎每章最后都提供或小或大的实战案例，让读者既学会语法也学会编程。

代码、教学视频下载

本书配套的示例代码与教学视频下载地址可以通过扫描右边的二维码获取。如果下载有问题或阅读中存在疑问，请联系 booksaga@163.com，邮件主题为“Python3.6 零基础入门与实战”。

本书读者

本书适合以下读者阅读：

- 没有学过编程，但对 Python 编程感兴趣的读者
- 有计算机语言基础，想入门 Python 编程的读者
- Python 数据分析处理入门读者
- 机器学习入门读者
- 网络爬虫爱好者
- 初级网络管理员
- 企业网络运维人员

本书作者

本书由王启明、罗从良编著，其中第 1~8 章、第 15~16 章由平顶山学院信息工程学院的王启明编写，第 9~14 章由罗从良编写，其他参与创作的还有王晓华、刘鑫、陈素清、常新峰、林龙、王亚飞、薛燚、王刚、吴贵文、李雷霆、李一鸣、谢志强，排名不分先后。

编者

2018 年 8 月

目　录

第 1 章

搭建 Python 开发环境

Python 是一门解释型编程语言，编写完毕后可直接执行，无须编译，发现 Bug 后立即修改，节省了编译时间。Python 流行的主要原因是其代码重用性高，可以把包含某个功能的程序当成模块代入其他程序中使用，因此 Python 的模块库非常庞大，几乎无所不包，不管是在科学计算、机器学习还是 Web 开发等领域都有其“模块”的身影。

Python 是跨平台性的，几乎所有的 Python 程序可以不加修改地运行在不同的操作平台，并能得到同样的结果。

因为 Python 有简单、无所不能及跨平台的特性，越来越多的企业选择用 Python 开发产品，也就造就了越来越多的 Python 岗位。这个时代，如果想学一门语言，那么 Python 肯定是首选。本章先从最简单的环境搭建学起。

1.1 Python 的版本说明

目前，Python 有两个版本：Python 2 和 Python 3。Python 2 的最终版本是 Python 2.7.14，写本书时，Python 3.6 的最终正式版本已是 Python 3.6.5，Python 3.7.0 版本正在完善中。

Python 2 已经不添加新的特性了，仅修复原有的安全问题，据说官方会在 2020 年关闭对它的维护。由于有很多常用库（如 Django、Numpy）也宣布逐步放弃对 Python 2 的支持，因此当前的主流选择都是 Python 3，本书将以 Python 3.6.4 版本进行讲解。

1.2 Python 的安装

本节介绍在 Windows、Linux 操作环境下安装 Python 的步骤。

1.2.1 Windows 下安装 Python

这里以 Windows 10 操作系统为例，演示如何在 Windows 系统下安装 Python。

步骤01 进入 Python 的官方网址 https://www.python.org/，首先单击主菜单中的 Downloads 选项，然后将鼠标指向 Windows，会出现下载版本，单击 Python 3.6.4，将自动下载文件 Python-3.6.4.exe，如图 1.1 所示。

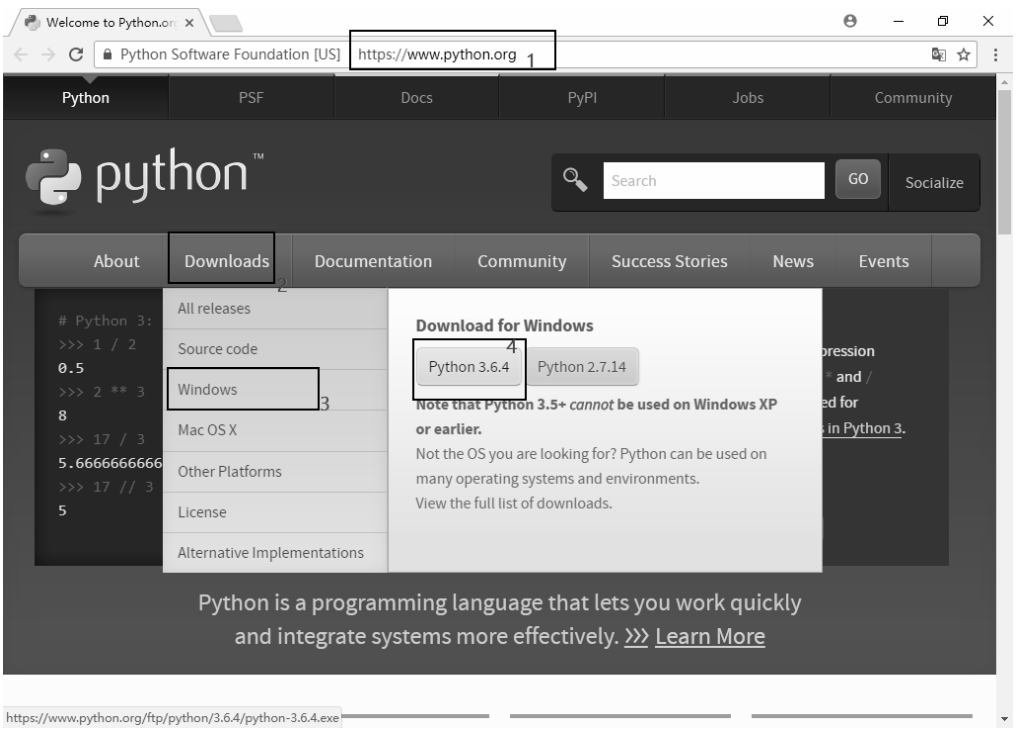

图 1.1 官方下载页面

注 意
Python 3.5+版本不能运行在 Windows XP 或更早的 Windows 版本上。

步骤02 默认下载的是 32 位版本，若操作系统是 64 位，就单击 Windows 选项，打开下载页面，选择适合自己的版本，如图 1.2 所示。

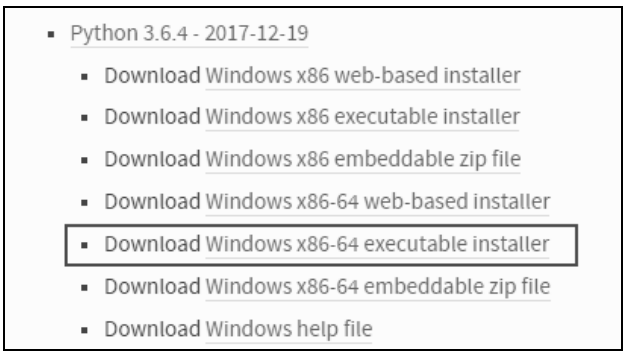

图 1.2　选择 64 位

x86 表示 32 位操作系统，x86-64 则是 64 位操作系统。每个系统下一般会有 3 个版本：

- web-based 版本：基于网络安装的，下载的文件会比较小。
- executable 版本：exe 可执行版本，推荐使用该版本。
- embeddable zip 版本：压缩版本。

步骤03 64 位下载的文件名是 python-3.6.4-amd64.exe，双击它，系统会弹出安全警告提示框，如图 1.3 所示。单击“运行”按钮即可。

图 1.3　安全警告提示框

步骤04 运行安装程序后，有两个选项：

- Install Now：立刻按默认设置安装。
- Customize installation：自定义安装，可选择安装路径、默认的一些安装组件。

因为是新手，所以直接选择第一项默认安装即可。

注　意
勾选 Add Python 3.6 to PATH 复选框（见图 1.4），安装程序会自动添加环境变量。如果忘记勾选，需要自己手动在环境变量中添加。

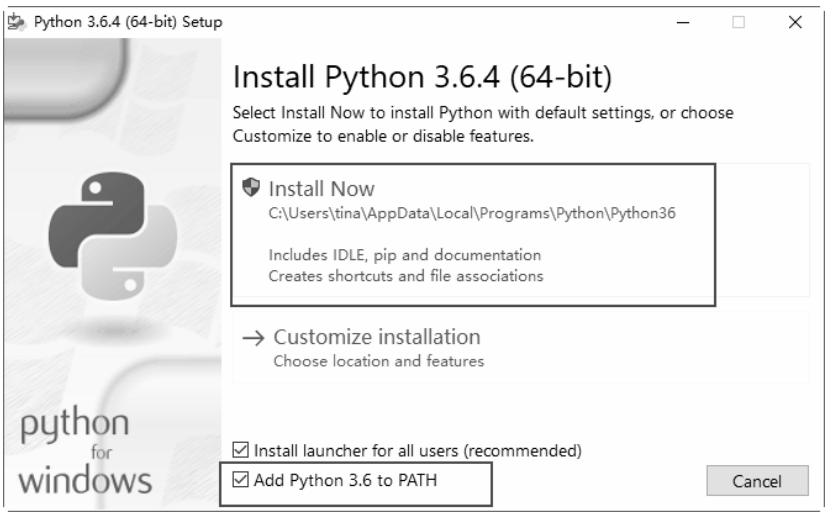

图 1.4　勾选 Add Python 3.6 to PATH 复选框

步骤05 安装过程如图 1.5 所示。

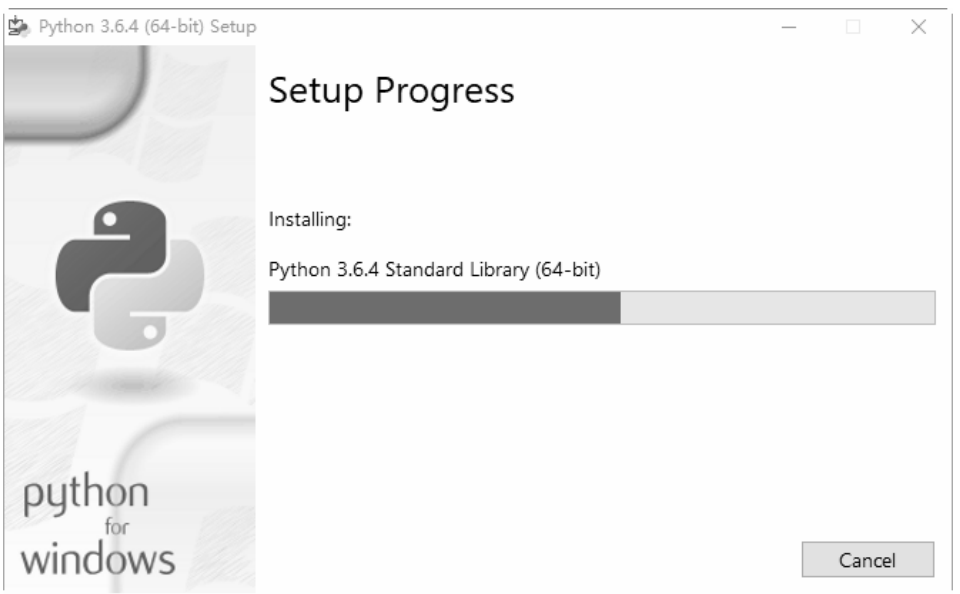

图 1.5　开始安装

步骤06 2 分钟后，出现完成窗口，如图 1.6 所示。这里有一个 Disable 开头的选项，用于解决系统的 path 长度限制，单击该选项，防止后面路径出现不可知的问题（操作系统低于 Windows 10 版本可能没有此选项）。单击 Close 按钮完成安装。

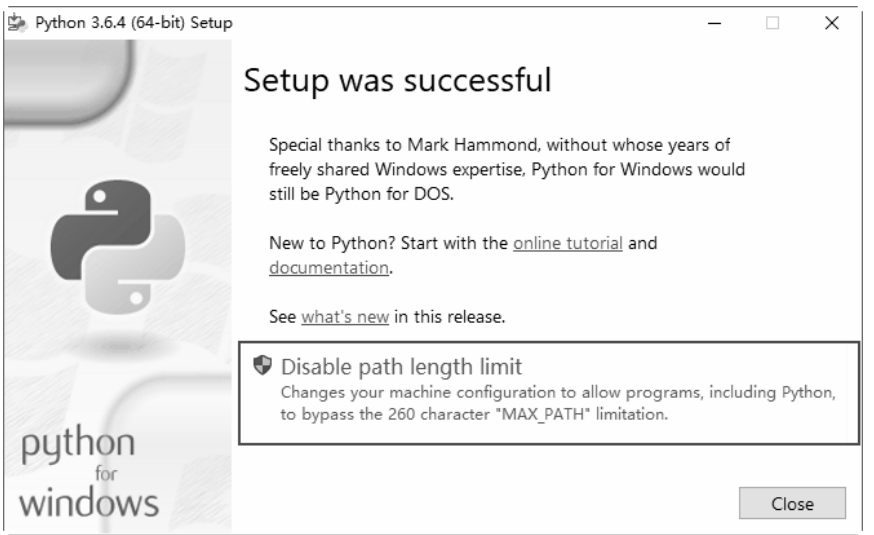

图 1.6　完成窗口

提　　示

安装完成后，打开操作系统的“高级系统设置|高级|环境变量|用户变量|path”，从窗口中可看到默认已经设置好了 Python 的路径，如图 1.7 所示。

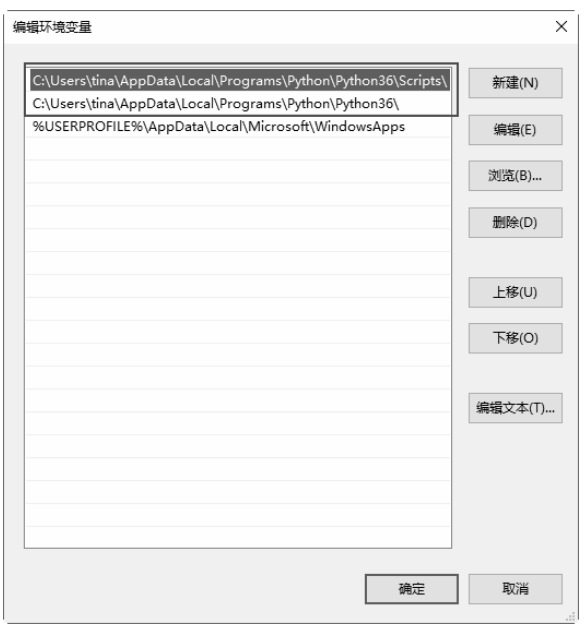

图 1.7　已配置好的环境变量

安装 Python 后，会在菜单栏中看到如图 1.8 所示的新增菜单项。

图 1.8　已安装的选项

这 4 项内容分别是：

- Python 3.6 Manuals（64-bit）：CHM 版本的 Python 3.6 官方使用文档。
- IDLE（Python 3.6 64-bit）：官方自带的 Python 集成开发环境。
- Python 3.6 Module Docs（64-bit）：模块速查文档，有网页版本。
- Python 3.6（64-bit）：我们常说的 Python 终端。

1.2.2　Linux 下安装 Python

连接到虚拟机 pyDebian 上，连接工具选择 Putty。下面先用 Putty 连接 Linux 机器。

步骤01 双击 Putty 图标，打开 Putty.exe，输入 IP 地址和端口信息，如图 1.9 所示。

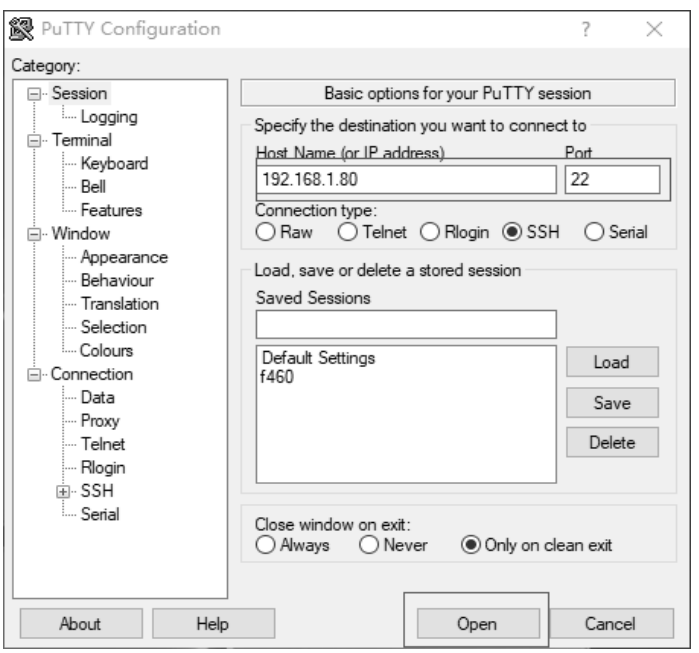

图 1.9　Putty 连接设置

步骤02 单击 Open 按钮，第一次使用 Putty 登录 Linux 系统会弹出一个安全警告提示框，如图 1.10 所示。

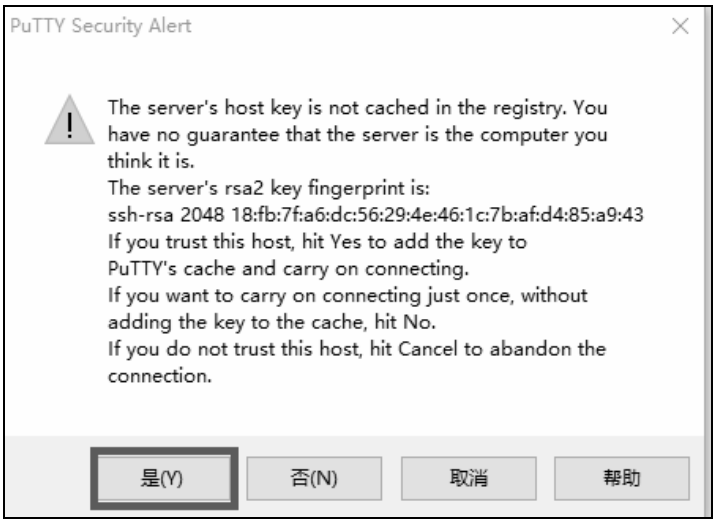

图 1.10　Putty 安全警告提示框

步骤03 单击“是(Y)”按钮，进入 Linux 的登录界面（用户名和密码使用默认的 king:qwe123 ），如图 1.11 所示。

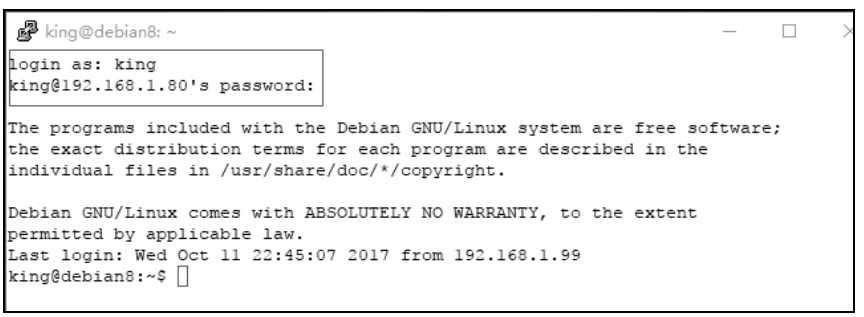

图 1.11　登录 Linux

步骤04 输入用户名和用户密码后（用户密码不回显），登录到 Linux。

Debian Linux 默认安装了 Python 2 和 Python 3（几乎所有的 Linux 发行版本都默认安装 Python）。Python 命令默认指向 Python 2.7，验证一下 Python 的路径，执行命令：

```
where is python
ls -l /usr/bin/python
ls -l /usr/bin/python3
```

执行的结果如图 1.12 所示。

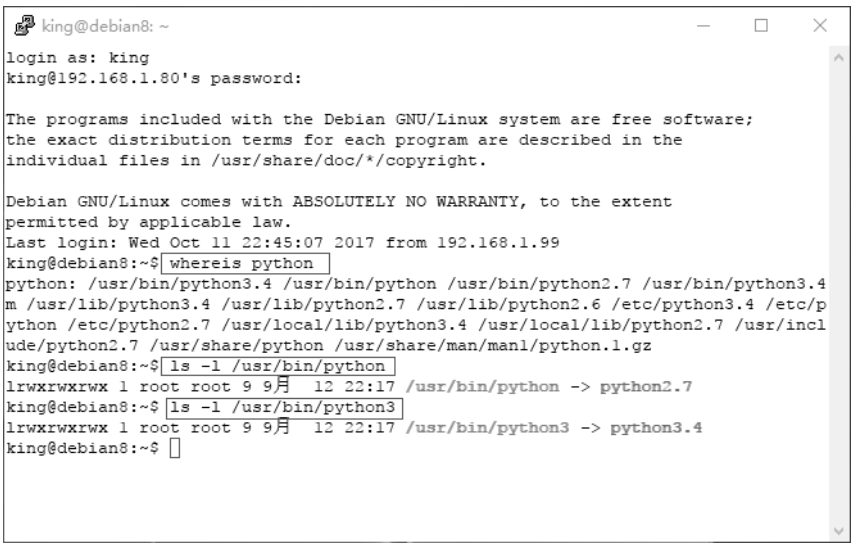

图 1.12　查看 Python 路径

再来看看 Python 的版本信息，分别执行命令：

```
python2 -V
python3 -V
```

执行的结果如图 1.13 所示。

图 1.13　Python 版本信息

从图 1.13 中可以看出，Linux 上安装的 Python 版本与官方网站上的最新版本（Python 3.6.4）是不同的。这是正常现象，一般来说 Debian Linux 会使用软件的最稳定版本，而 Ubuntu Linux 会使用软件的最新版本。

1.3　打开 Python 的方式

打开 Python 的方式有多种，这里简单介绍一下。

（1）因为环境变量中已经添加了 Python，所以直接调用 Windows 的命令行（cmd），输入 python 命令，即可打开 Python，如图 1.14 所示。

图 1.14　输入 python 命令

（2）通过安装后菜单中的 Python 终端（图 1.15 左）和 IDEL（图 1.15 右）也可以打开 Python。

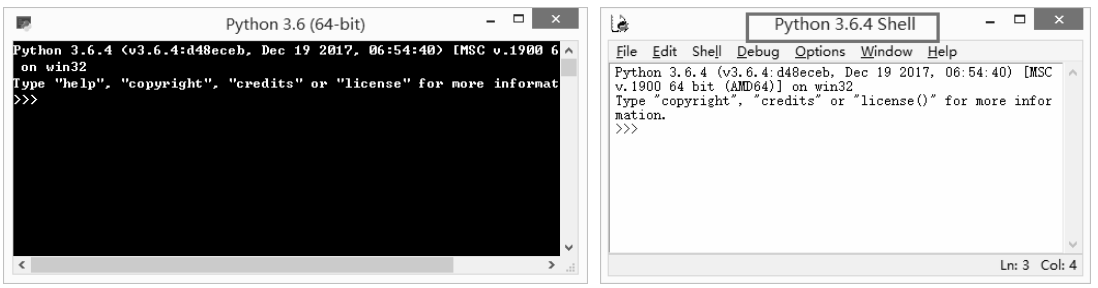

图 1.15　Python 终端和 IDEL

1.4　交互模式解释器

打开 Python 后，会看到“>>>”符号。简单来说，这就是 Python 交互模式解释器。

这里会讲到两个概念：交互和解释器。学过计算机基本原理的读者应该明白解释器的意思，即将高级语言解释给机器听，也就是将代码转换成计算机能懂的机器码。交互就是你问我、我回复你。

有了交互模式解释器，我们就不用创建、编辑、保存后再运行源文件了。Python 中的交互模式解释器以“>>>”开头，当我们用它执行代码时，不但会检查代码的正确性，而且还能在把这段新代码加入源文件之前进行各种操作，如查看数据结构。

例如，我们可以直接用 print 函数输出一段中文、一个计算表达式的结果，如图 1.16 所示。这个时候并不需要保存脚本文件，就可以执行代码。

注　意
Python 语句的最后可以加分号，也可以不加分号。建议统一不加分号，除非几个语句写在同一行中，才用分号间隔。

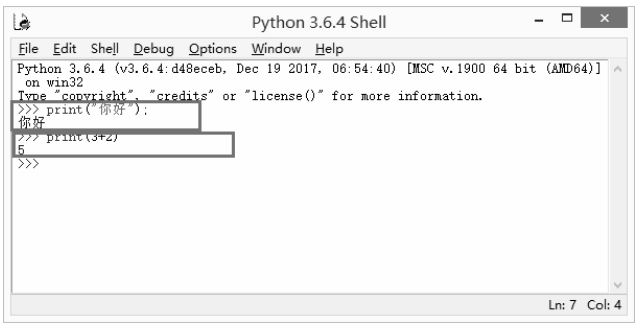

图 1.16　交互模式解释器

1.5　第一个 Python 程序 Hello World

鉴于 Python 运行代码的多样性，本节介绍两种实现第一个程序的方法：交互式和脚本式。

1.5.1　交互式

前面已经介绍过交互模式解释器，可以直接输入代码。下面编写第一个程序：

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

直接在 Python IDEL 中输入以上代码，如图 1.17 所示。

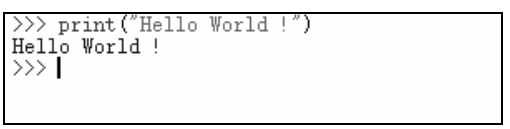

图 1.17　Python IDEL

1.5.2　脚本式

脚本式就是将代码保存为脚本文件，然后使用 python 命令执行这个文件。

打开 Python IDEL，单击 File|New File，打开 IDEL 的编辑器，输入如下代码，如图 1.18 所示。

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

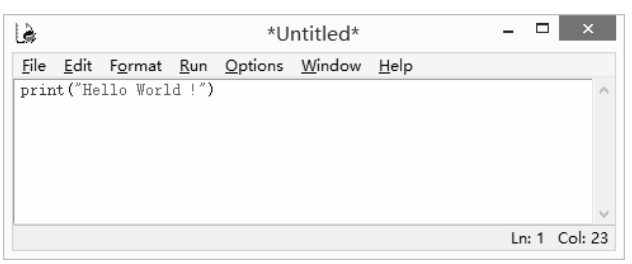

图 1.18　IDEL 的编辑器

保存上述文件到合适的位置（如 D 盘），并命名为 hello.py，如图 1.19 所示。

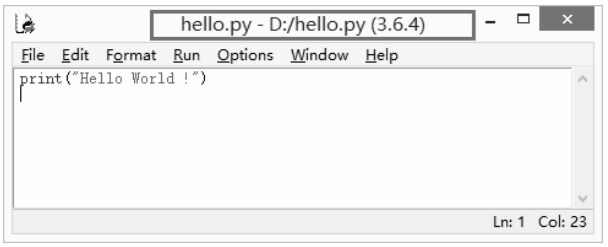

图 1.19　保存 py 文件

此时，可以直接使用 Run|Run Module 菜单命令执行这个脚本文件，如图 1.20 所示。

```
========== RESTART: D:\hello.py ==========
Hello World !
>>>
```

图 1.20　执行脚本

打开 Windows 命令行客户端（cmd），输入以下命令，也能执行这个脚本文件，如图 1.21 所示。

```
python d:\hello.py
```

图 1.21　在命令行执行脚本

1.6　Python 开发工具

集成开发环境（Integrated Development Environment，IDE）是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具。它集成了代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件服务套件。Python 常用的 IDE 有两种：自带的 IDEL 和 PyCharm。

1.6.1　Python 自带集成开发环境 IDEL

本节通过一段代码来演示 IDEL 的使用。虽然很多高手会建议初学者使用更好的编辑器（有些是收费的），但鉴于这是 Python 自带的开发环境，还是讲解一下它的使用方法，让读者在比较小的学习成本基础上方便自己的开发。

初学者要重点学习 IDEL 的三部分内容：编辑器、解释器和调试器。

1. 编辑器

打开 Python IDEL，单击 File|New File 菜单，打开编辑器，输入以下代码，如图 1.22 所示。【示例 1-1】

```
num1=input('请输入第 1 个数值：')
num1=int(num1)
num2=input('请输入第 2 个数值：')
num2=int(num2)
if num1>num2:
    print ('第 1 个数值大于第 2 个数值。')
else:
    print ('第 1 个数值并不比第 2 个数值大。')
```

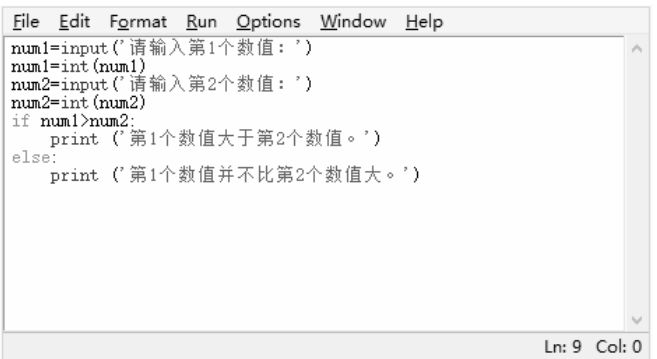

图 1.22　编辑器

在编辑器窗口中有菜单栏、文本输入区域。这里说一下编辑器的特色：

（1）高亮显示 Python 语法。读者会看到橘黄色的 if 和 else、绿色的字符串、紫色的函数等（实际效果请读者在电脑上打开这个文件观察）。

（2）自动缩进。Python 有严格的缩进要求，当输入 if 条件后面的冒号再回车后，编辑器会自动缩进。缩进的长度可以通过菜单 Format|New Indent Width 修改，默认是 4。

（3）自动完成。这是初学者比较喜欢的功能，对于一个函数名称，我们只需要输入前几个字母，就可以使用 Alt+\（或菜单 Edit|Expand Word）自动完成。

（4）查询复杂函数。如果记不住某个函数的名字，只知道前三个字母，可通过 Ctrl+Space（或菜单 Edit|Show Completetions）罗列出符合前几个字母的所有函数，如图 1.23 所示。

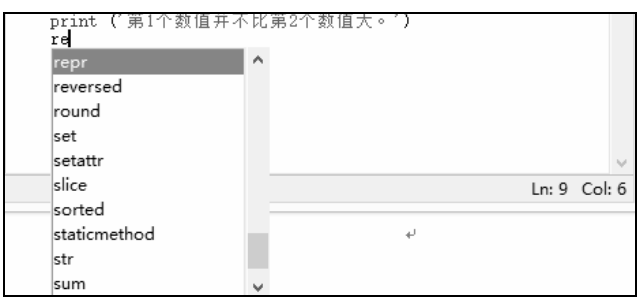

图 1.23　罗列函数

（5）自动增加或去掉注释。大部分编辑器都具备将选中的行变为注释段或取消注释的功能，

IDEL 也是。选中一段代码，然后按 Alt+3 组合键（或菜单 Format|Comment Out Region），就会在行前面增加##符号，如图 1.24 所示。按 Alt+4 组合键（或菜单 Format|Uncomment Region）会取消注释。

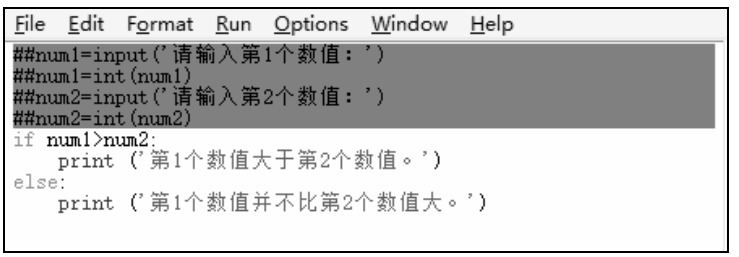

图 1.24　自动增加注释

#是 Python 的单行注释符号，'''是多行注释，如下代码所示：

```
'''
这是注释
这是注释
这是注释
'''
```

2. 解释器

交互模式解释器前面已经介绍过，在编辑器窗口中单击菜单 Run|Run Module 命令就会自动转换到解释器窗口，并给出执行效果。

3. 调试器

如果代码有问题，可以使用调试器。在 IDEL 窗口中单击菜单 Debug|Debugger 命令打开调试器，此时解释器也发生了改变，如图 1.25 所示。关闭 Debug 后 ON 会变为 OFF。

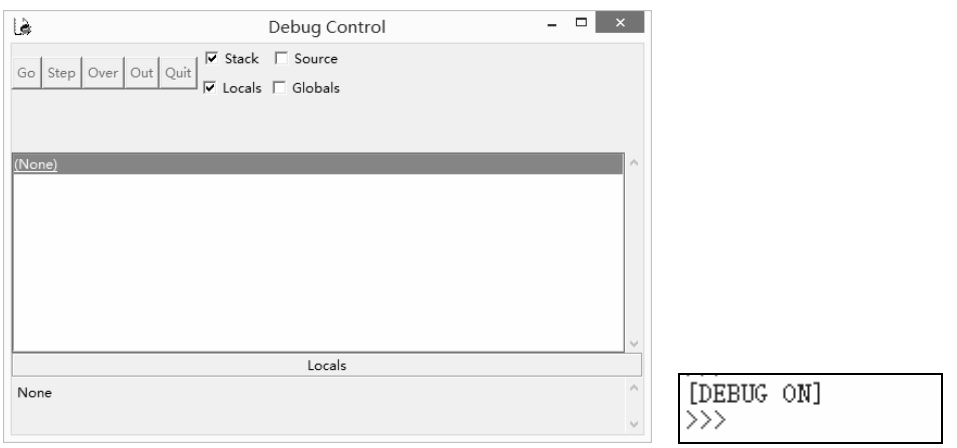

图 1.25　调试器

在解释器中输入 print(1+3)，将看到调试器的变化，如图 1.26 所示。

图 1.26　调试器的变化

1.6.2　安装 PyCharm 集成开发环境

PyCharm 是一种 Python IDE，带有一整套可以帮助用户在使用 Python 语言开发时提高效率的工具，如调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试、版本控制。目前使用比较多的 Python IDE 就是 PyCharm，其可以跨平台，在 Mac OS 和 Windows 系统下都可以用。缺点是专业版只有 30 天免费，如果要使用专业版就需要花钱购买。

PyCharm 的官方网址是 http://www.jetbrains.com/pycharm/。从网址可以看出，其属于 JetBrains 公司，位于布拉格，为人所熟知的产品是 Java 集成开发环境——IntelliJ IDEA。

步骤01 打开官网，如图 1.27 所示，然后单击 DOWNLOAD NOW 按钮，出现操作系统选择，如图 1.28 所示，有社区版和专业版，社区版是免费开源的。这里使用专业版来讲解，读者也可以选用社区版学习本书内容。

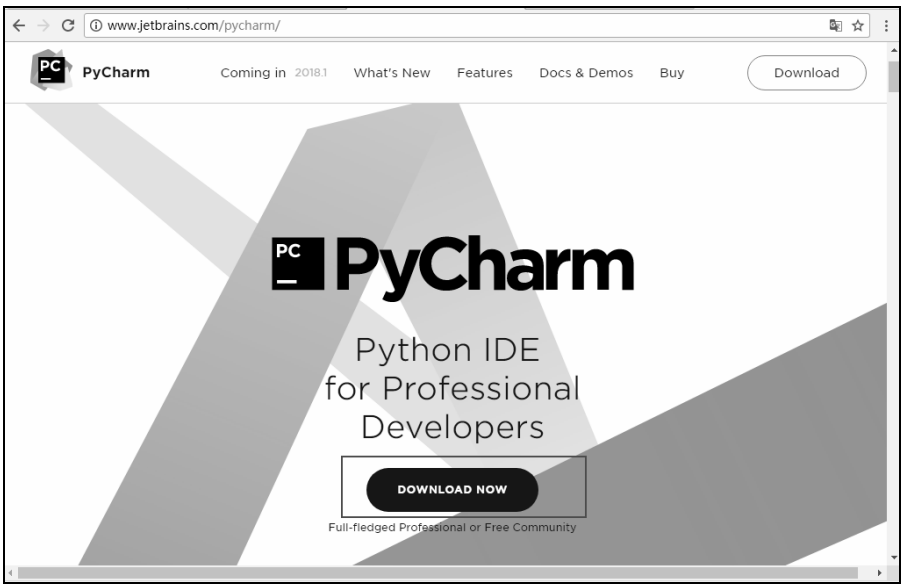

图 1.27　PyCharm 官网

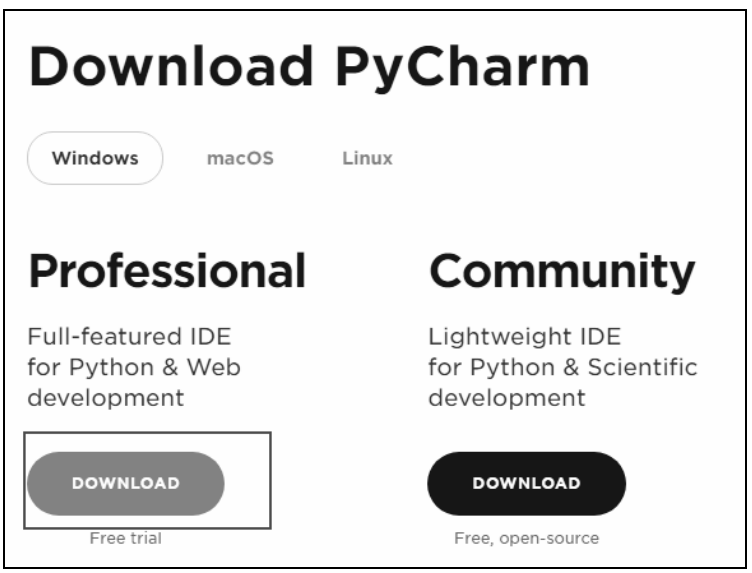

图 1.28　选择操作系统

步骤02 选择 Windows 下的 Professional（专业）版，单击 DOWNLOAD 按钮会自动下载，下载后的文件名为 pycharm-professional-2017.3.3.exe，大小为 250MB。

步骤03 双击下载的文件进行安装，如图 1.29 所示。

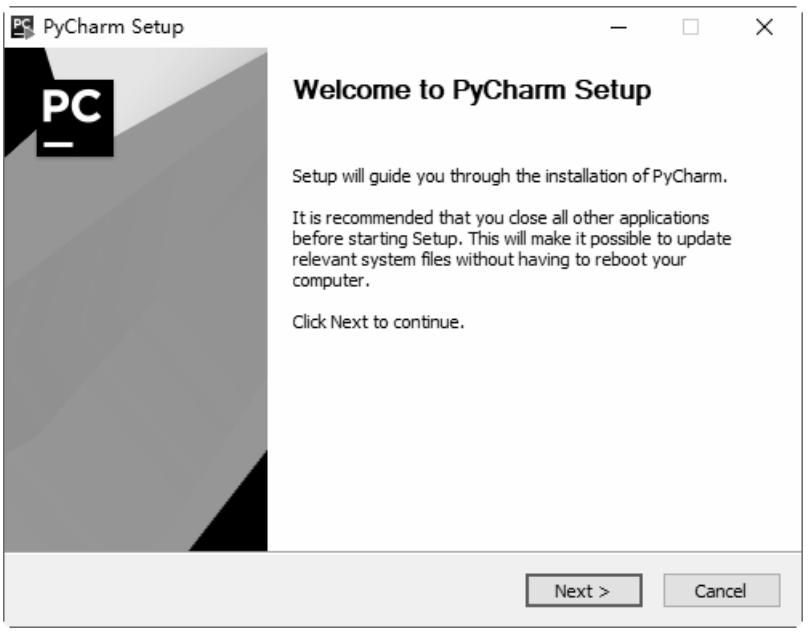

图 1.29　开始安装 PyCharm

步骤04 单击 Next 按钮，然后选择安装位置，这里没有特殊要求，如图 1.30 所示。

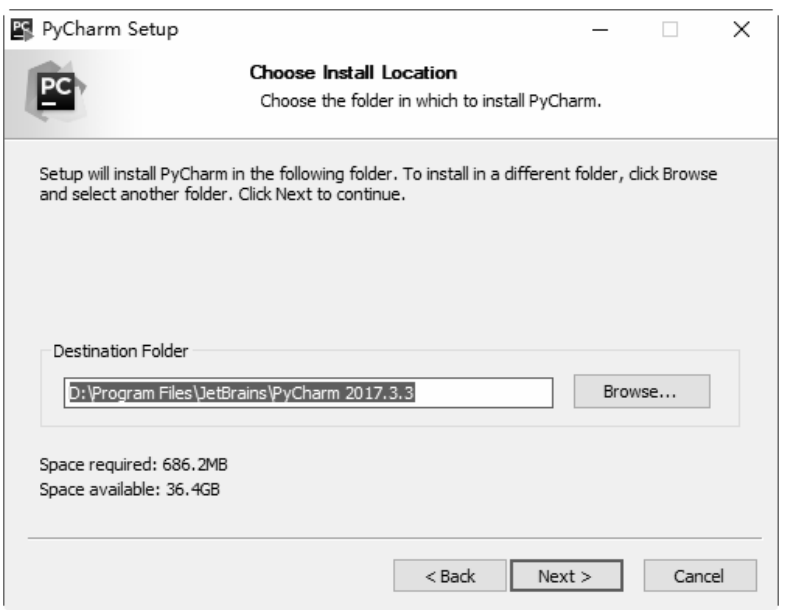

图 1.30　选择安装位置

步骤05 单击 Next 按钮，出现如图 1.31 所示的配置界面，根据系统选择是 32 位还是 64 位，然后勾选关联.py 扩展名的复选框。

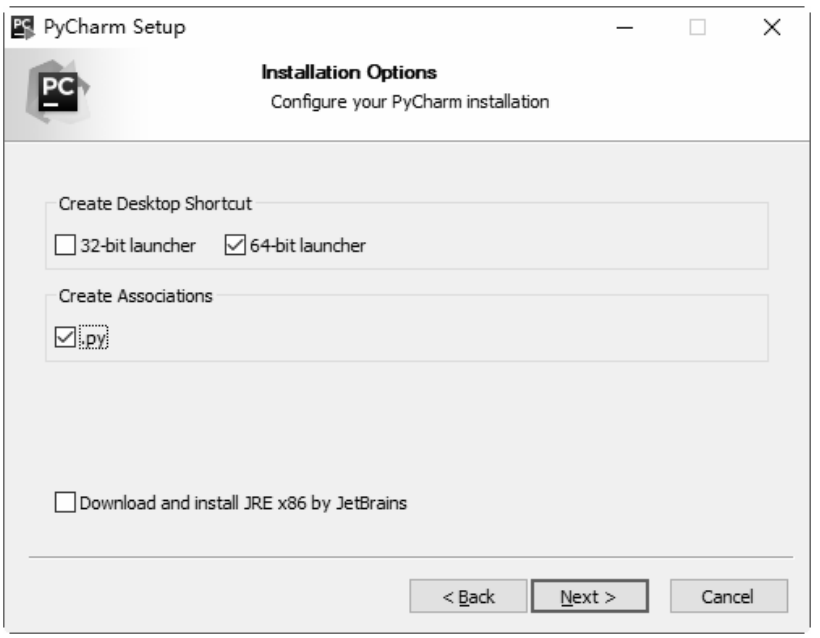

图 1.31　选择 64 位

步骤06 单击 Next 按钮，在主菜单中创建程序的快捷方式，默认命名即可，如图 1.32 所示。

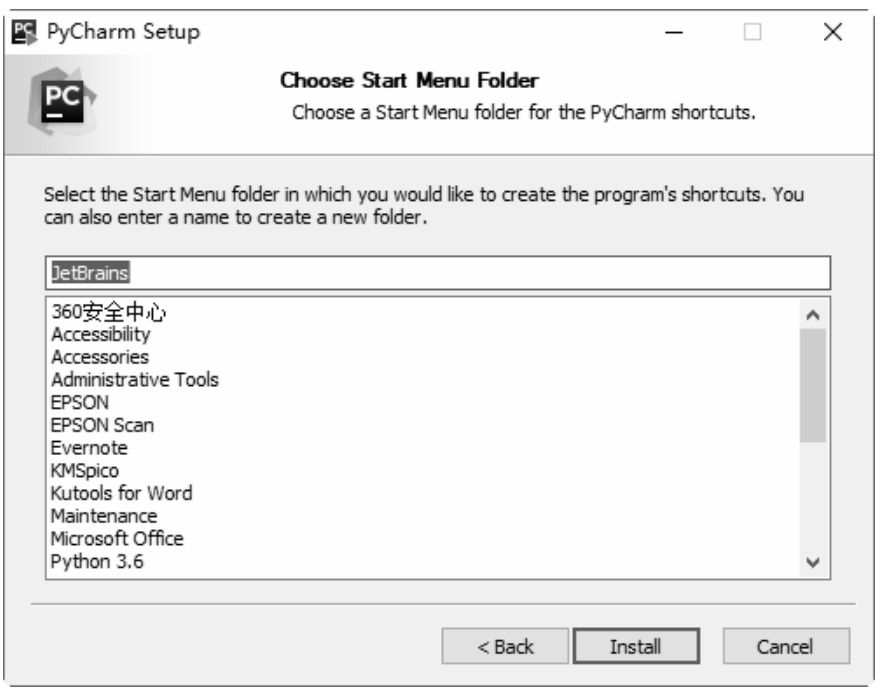

图 1.32　添加快捷项到主菜单

步骤07 单击 Install 按钮开始解压文件，1 分钟安装完毕，如图 1.33 所示。可以勾选立刻运行的 Run PyCharm 复选框。

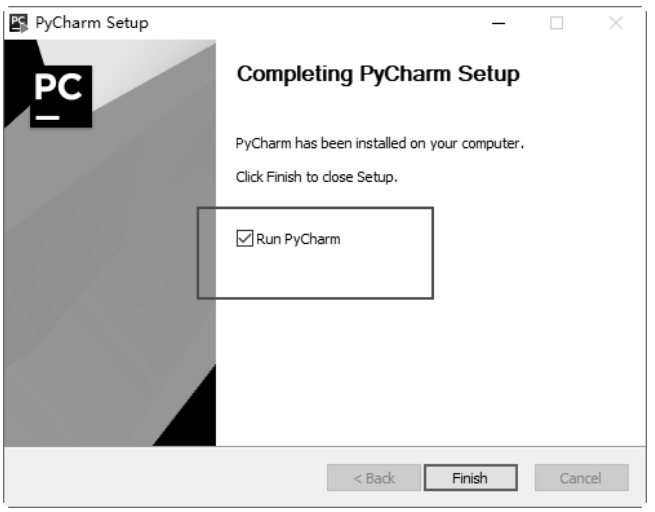

图 1.33　安装初步完成

步骤08 单击 Finish 按钮会打开 PyCharm，第一次打开会有两个导入包的设置项，这里选择默认的第 2 个，如图 1.34 所示。

图 1.34　是否导入包

步骤09 单击 OK 按钮后出现许可协议，再单击 Accept 按钮，如图 1.35 所示。

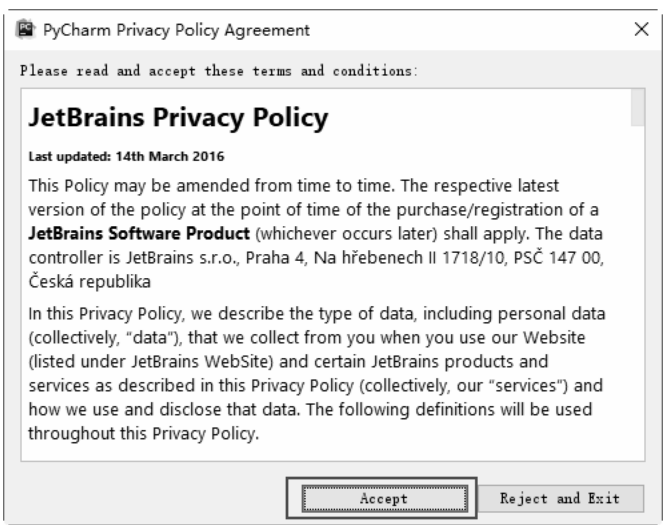

图 1.35　接受协议

步骤10 此时会出现注册账号的窗口，我们选择免费试用 Evaluate for free，单击 Evaluate 按钮（见图 1.36），然后出现一个试用协议，直接单击 Accept 按钮即可。

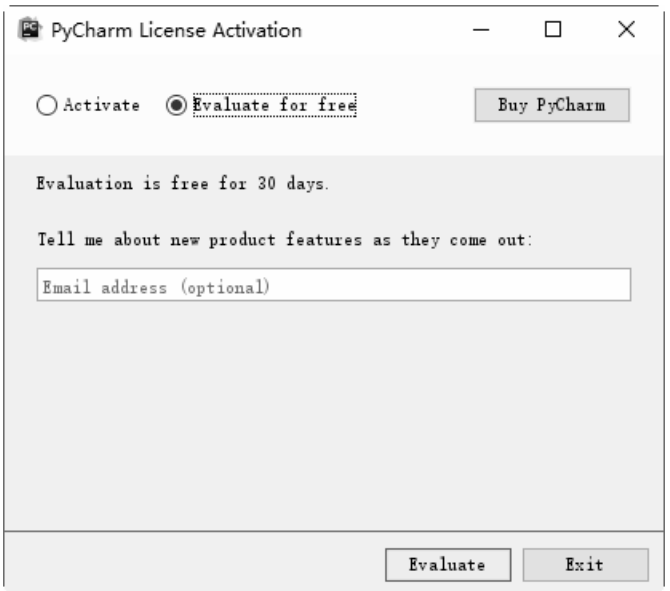

图 1.36　免费试用

步骤11 第一次打开也需要设置 UI 主题，如图 1.37 所示，根据自己的爱好进行选择。选择完成后，单击左侧的 Skip Remaining and Set Defaults，以后就会默认这个 UI 主题。

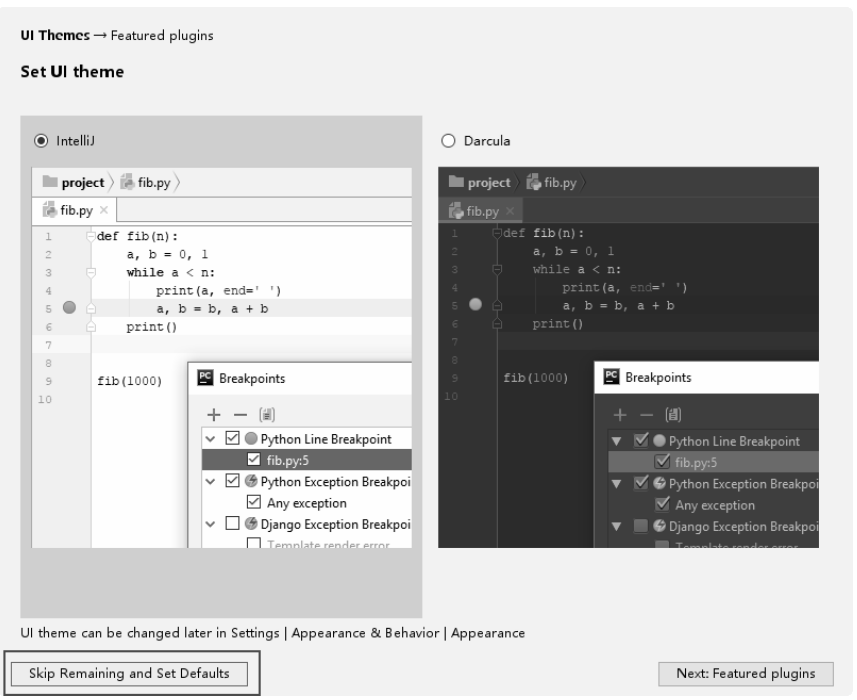

图 1.37　选择主题

步骤12 截止到现在，才真正打开 PyCharm，如图 1.38 所示。可以打开已经存在的项目，也可以新建项目。

图 1.38　PyCharm 初始界面

步骤13 单击 Create New Project 选项，出现项目类型的选择界面，如图 1.39 所示。

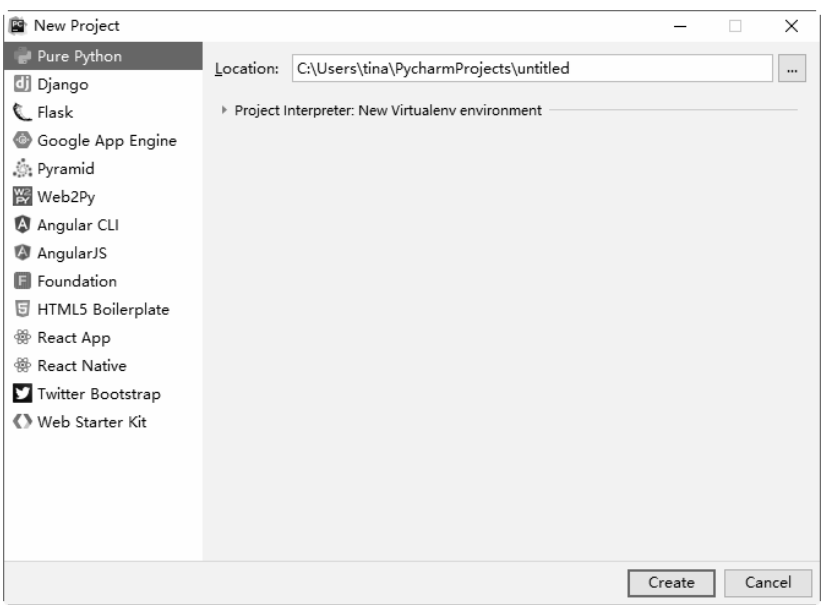

图 1.39　选择项目类型

步骤14 如果希望自己的项目保存在特定位置，可以修改此处，然后单击 Create 按钮。此时需要等待 1 分钟的时间配置环境。最终创建好的项目界面如图 1.40 所示。

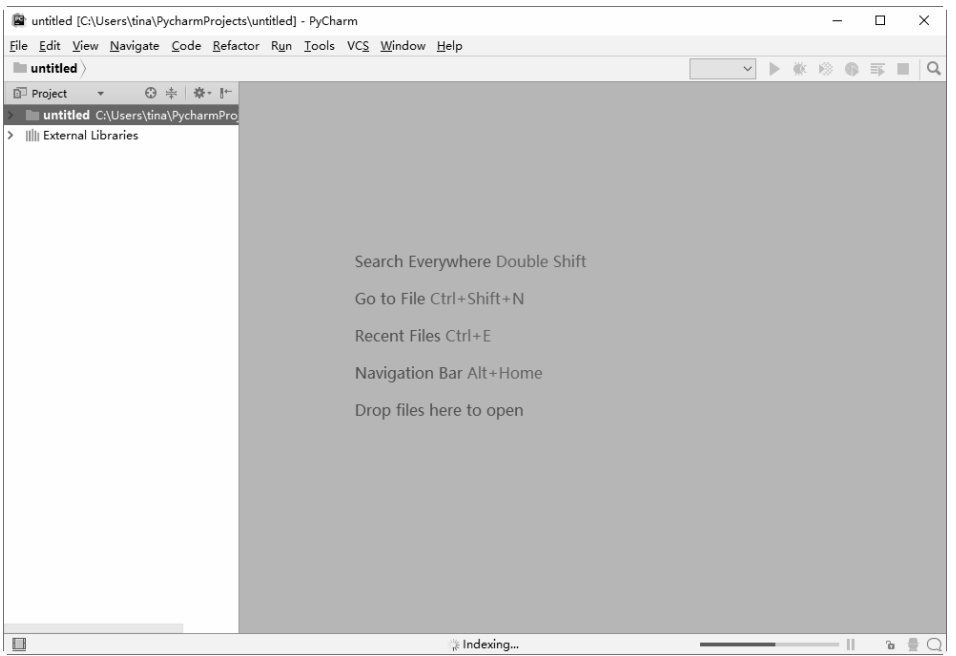

图 1.40　创建好的项目界面

1.6.3　使用 PyCharm 集成开发环境

PyCharm 的功能有很多，使用起来比 IDEL 复杂，本书的大部分例子都是使用 IDEL 进行测试。下面简单介绍一下 PyCharm 的使用。

1. 创建 Python 文件

步骤01 右击新建的项目，选择 New|Python File，输入文件名，如 py1.py，如图 1.41 所示。

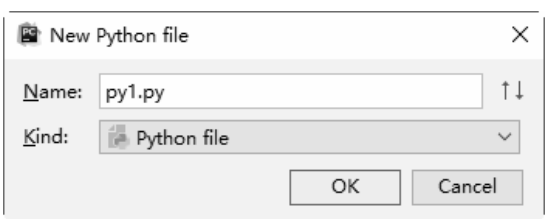

图 1.41　创建 Python 文件

步骤02 单击 OK 按钮，鼠标会停留在右侧的编辑界面，输入以下代码：

```
print('Hello Python')
```

按 Ctrl+S 组合键保存，这样第一个 Python 文件就创建好了。

2. 运行 Python 文件

PyCharm 的运行都在菜单 Run 中。选择 Run|Run ‘py1’命令（或按 Shift+F10 组合键），就会出现一个控制台，输出上述代码的运行结果，如图 1.42 所示。

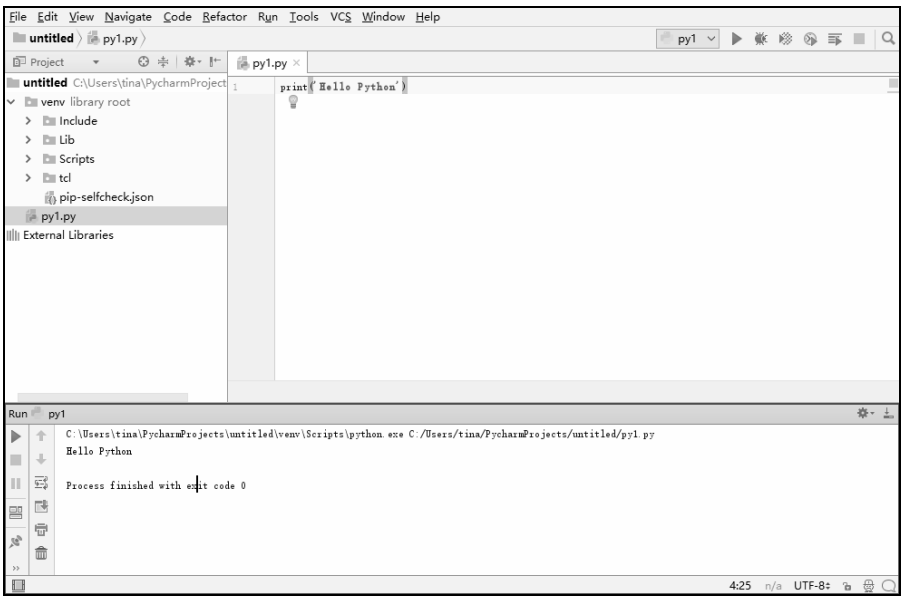

图 1.42　运行 Python 文件

1.7　注意 Python 的缩进

学习过 C、Java、JavaScript、C#语言的读者应该知道，这些语言都使用{}表示代码段，如一段 if 代码：

【示例 1-2】

```
//判断用户的输入
 if (choice=="1")
 {
    Console.Write ("你目前的开发工作是:1 ");
 }
 else if (choice == "2")
 {
    Console.Write ("你目前的开发工作是:2");
 }
 else
 {
    Console.Write ("对不起你选择错误");
    Console.Write ("请重新选择");
 }
```

在 Python 中，相同的缩进代码才表示它们属于一个代码段。下面使用前面学习过的一段代码【示例 1-1】，我们给 num1 的值加 10，缩进大小和 else 语句中的内容保持相同。读者可以思考：是不管 num1 和 num2 谁大谁小都会输出 num1 的值，还是只有当 num1<=num2 时才输出 num1 的值？

【示例 1-3】

```
num1=input('请输入第 1 个数值：')
num1=int(num1)
num2=input('请输入第 2 个数值：')
num2=int(num2)
if num1>num2:
    print ('第 1 个数值大于第 2 个数值。')
else:
    print ('第 1 个数值并不比第 2 个数值大。')
    num1=num1+10
    print(num1)
```

代码执行的结果对比如图 1.43 所示。只有当 num1<=num2 时才输出 num1 的值。

```
>>>
========== RESTART: D:/test1.py ==========
请输入第1个数值：1
请输入第2个数值：5
第1个数值并不比第2个数值大。
11
>>>
>>>

>>>
========== RESTART: D:/test1.py ==========
请输入第1个数值：5
请输入第2个数值：1
第1个数值大于第2个数值。
>>>
```

图 1.43　缩进对比 1

继续更改上一段代码的缩进，如下：

【示例 1-4】

```
num1=input('请输入第 1 个数值：')
num1=int(num1)
num2=input('请输入第 2 个数值：')
num2=int(num2)
if num1>num2:
    print ('第 1 个数值大于第 2 个数值。')
else:
    print ('第 1 个数值并不比第 2 个数值大。')
num1=num1+10
print(num1)
```

不管 num1 和 num2 谁大谁小都会输出 num1 的值，如图 1.44 所示。

```
>>>
========== RESTART: D:/test1.py ==========
请输入第1个数值：1
请输入第2个数值：5
第1个数值并不比第2个数值大。
11
>>>
>>>
>>>
========== RESTART: D:/test1.py ==========
请输入第1个数值：5
请输入第2个数值：1
第1个数值大于第2个数值。
15
>>>
```

图 1.44　缩进对比 2

第 2 章

Python 中的数据与结构

学习一门语言，读者需要了解该语言中数据的存在形式。存在形式多种多样，有数字、字符……为了方便学习，语言会将它们进行归类，这就是常说的数据类型。每种语言的数据类型都差不多，如 Python 中有数字类型、字符串类型，Java 中也有，C#语言中也有。学习这些类型是每种语言基础的语法。本章介绍的是 Python 中的数据类型。

2.1 Python 中的标准数据类型

Python 中的标准数据类型有 6 种：

- Number（数字）：用来表示数据的一些数字。
- String（字符串）：用来表示文本的一些字符。
- List（列表）：用来表示一组有序的元素，后期还可以更改。
- Tuple（元组）：用来表示一组有序的元素，后期不可以更改。
- Sets（集合）：用来表示一组无序不重复的元素。
- Dictionary（字典）：用键值对的形式保存一组元素。

要想更有效地记忆这些类型，可对其进行分类：

- 用存储方式来分类，可分为原子类型（数字、字符串）和容器类型（列表、集合、元组、字典）。
- 按访问方式来分类，可分为直接访问（数字）、顺序访问（字符串、列表、集合、元组）、映射访问（字典）。

注　　意

在 Python 中，一切皆为对象。对象就是保存在内存中的一个数据块，我们有时候也会说创建一个数字对象、字符串对象。

2.2　变　　量

变量对应着内存中的一块存储位置。简单地说，变量是计算机存储于内存中其值可改变的量。这里的“可改变”是指程序运行期间的可改变。

在 Python 程序中，不需要声明变量，但必须为变量赋值后才可以使用。比如其他语言是先声明再赋值：

```
int x;
x=200;
```

而 Python 则是：

```
x=200
```

等号“=”用来为变量赋值，变量本身没有类型，为其赋值 200，我们会说这是一个整型变量，这里的“类型”是变量所指的内存中对象的类型。

可以同时为多个变量赋值。例如，下面这 3 个整型变量的值都是 200。

```
x=y=z=200
```

也可以同时赋值为不同类型。例如，在下面定义的变量中，x 和 y 为整型，z 为字符串。

```
x, y, z = 1, 2, "hello world"
```

变量的命名要注意：

- 变量的首字符必须是字母或下画线“_”。
- 其他部分由字母、数字和下画线组成。
- 变量区分大小写。

变量的命名不能取 Python 中的保留字，如 if、else、print 这些常见的都是保留字。在解释器中可以输入以下命令来查看保留字（见图 2.1）。

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

其中，import 用来引入 Python 标准库中的 keyword 模块。

```
>>> 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', 'retu
rn', 'try', 'while', 'with', 'yield']
>>>
```

图 2.1　查看保留字

2.3 数　　字

Python 支持的数字有 int（整型）、float（浮点型）、bool（布尔型）、complex（复数型）4 种。本节分别介绍这些数字类型。

2.3.1 使用整型

整型用来表示一些数字，如 1、-10、060、0x29。整型不区分长短和符号，Python 3 中的 int 可以存储比 64 位更大的整数。

说　明
每次安装软件的时候，我们常在 32 位和 64 位之间做选择，这个位数就是处理数据的能力。64 位是可以处理在 2^{64} 范围里面的数据，32 位是可以处理在 2^{32} 范围里面的数据，但 Python 中的整数可以无限扩展，它取决于可用内存的大小，并不受 32/64 位的限制。

使用整型有两种方式：一种是直接赋值为数字；另一种是使用 int 函数将其他类型转换为整型。举例代码如下：

```
num1=100            #直接赋值
var1='200'
num2=int(var1)          #转换类型
```

要赋值多个整型，可以这样写：

```
num1=num2=num3=100
```

读者可以思考一个问题：这 3 个变量的数据存储在内存中是占 3 个数据的位置还是 1 个数据的位置？

我们使用 id 函数打印变量在内存中的位置，结果如图 2.2 所示。可以看出，其实 3 个变量是指向内存中的同一位置。

```
#inttest.py
num1=num2=num3=100
print(id(num1))
print(id(num2))
print(id(num3))
```

```
========== RESTART: D:/inttest.py ==========
1593801280
1593801280
1593801280
>>>
```

图 2.2　内存位置

注　　意

继续思考这个问题，如果将 num1、num2、num3 分别赋值，但赋值相同，那是不是所占的内存地址也相同？

学习语言中，读者肯定经常看到“表达式”这个概念。表达式（Expression）是将相同类型的数据（如数字、字符串等），用运算符号按一定的规则连接起来的、有意义的代码。这里有两点需要注意：

- 相同类型的数据，比如 1+2 可以，但 1+'2' 就不行。
- 用运算符连接，每种类型都有不同的运算符，后面会详细介绍。

整型一般用来进行一些表达式的运算，常见的整型运算符是+、-、*、/，可以使用如图 2.3 所示的代码进行测试。

```
>>> 1+2
3
>>> 5-3
2
>>> 80*5
400
>>> 80/5
16.0
>>>
```

图 2.3　整型的运算

2.3.2　使用浮点型

浮点型就是我们常说的带小数的类型，如 13.30、-80.16、30.2e100。使用浮点型有 3 种方式：赋值、强制转换、两个整型相除。

（1）直接赋值：

```
num1=15.0
```

（2）使用 float 函数强制转换：

```
num1=float(15)
```

（3）两个整型相除：

```
num1=15/3
```

在第 3 种方式中，虽然两个整型可以整除，但是在 Python 中依然得出的是浮点型。下面用代码测试一下：

```
num1=15/3
print(num1)
```

输出结果是 5.0。如果计算结果要输出整型，则需要使用“//”而不是“/”。以下代码的输出结果就是 5。

```
num1=15//3
print(num1)
```

2.3.3 使用布尔型

布尔型就是 True 或 False。在 Python 中，True 的值是 1，False 的值是 0，可以和数字相加，因此把它放在数字这个分类中。

因为好多表达式运算的结果也是布尔型，所以使用布尔型的方法有很多，这里我们简单介绍几种。

（1）直接用一个关键字 True 或 False 赋值。

```
T = True
F = False
print(T,F)          #输出为 True  False
```

也可以将其用于数值运算，例如：

```
T = True
print(T+10)         #输出为 11
```

（2）使用 bool 函数。

```
F=bool(0)
print(F)            #输出为 False
```

（3）表达式的运算结果也为布尔型。

```
print(1>2)          #输出为 False
```

布尔型的运算符有 and、or、not（必须小写），读者还需要了解两个布尔表达式的运算规则，参见表 2.1。

表 2.1 布尔型的运算符

操作符	结果
x or y	x,y 只要有一个值为 True，结果就为 True
x and y	x,y 只要有一个值为 False，结果就为 False
not x	取 x 的相反值

下面演示布尔运算的操作：

【示例 2-1】

```
01  T = True
02  F = False
03  #F=bool(0)
04  print(T or F)           # True
05  print(T and F)          # False
06  print(not F)            # True
```

注 意

Python 3 中 if(True)的效率比不上 if(1)的效率。

我们常用的表达式一般是加、减、乘、除，优先级在小学就学过（先加减后乘除）。布尔运算的优先级低于表达式，读者可以测试一下下面这段代码：

```
print( 1>2 and 2<1 )        #False
```

2.3.4　使用复数型

在 Python 的数字中，复数 complex 是一个比较复杂的类型。从概念上来说，复数是一个实数和一个虚数的组合。实数就是我们常说的 1、100、350.60 等，虚数是一个虚拟的数，数学家称之为 j，广泛用于科学计算中。在 Python 中也用 j 或 J 表示这个虚数，比如 15.0j、5.16J、3.2e-6j 都是复数。

关于复数有以下几个注意事项：

- 虚数不能单独存在，必须和实数部分一起构成一个复数。
- 实数部分和虚数部分都是浮点数。
- 虚数部分必须有后缀 j 或 J。

复数的定义一般有两种形式：

（1）直接赋值。

【示例 2-2】

```
com1=15.0j
print(type(com1))           # <class 'complex'>
```

这里的 type 函数用来输出 com1 变量的类型。

（2）使用 complex 函数，它有两个参数，当然也可以两个参数都不输入。

【示例 2-3】

```
com1=15.j
print(complex(1))           # (1+0j)
print(complex('3+5j'))      # (3+5j)
print(complex(3,2))         # (3+2j)
print(complex())            # 0j
print(com1)                 # 15j
```

2.4　字 符 串

字符串变量的定义特别简单，直接赋值即可，例如：

```
str1 = 'Hello World!'
```

如果我们要操作字符串，比如拆分、连接、获取字符串的一部分，就需要学习字符串的运算符、内置操作函数等内容。本节将逐一介绍这些知识点。

2.4.1　字符串的单引号、双引号、三引号

字符串赋值时可以使用单引号、双引号、三引号形式。单引号和双引号并没有太大的区别，

比如：

```
str1 = 'Hello World!'
str2 = "I'm fine,and you?"
```

注　意

如果字符串中包含单引号就要使用双引号进行定义。

如果是特别长的字符串换行时，单引号和双引号需要加“\”符号，例如：

```
str1 = 'Hello \
World!'
str2 = "I'm fine,\
and you?"
```

“\”符号又会涉及一些字符串的转义，为了更直观，特别长的字符串可以使用三引号，例如：

```
str3=''' this
is
world '''
```

这 3 种方法并没有太大的区别，读者可根据实际生产环境使用。

2.4.2　字符串的截取

说得好没有练得好，还是先来演练一下：

【示例 2-4】

```
01  str1 = 'Hello World!'
02  str2 = "I'm fine,and you?"
03
04  print ("str1 原文: ", str1)
05  print ("str2 原文: ", str2)
06
07  print ("1.str1[1]: ", str1[1])
08  print ("2.str1[2:5]: ", str1[2:5])
09  print ("3.str1[-2:1]: ", str1[-2:1])
10  print ("4.str1[5:5]: ", str1[5:5])
11
12  print ("1.str2[0]: ", str2[0])
13  print ("2.str2[:5]: ", str2[:5])
14  print ("3.str2[6:]: ", str2[6:])
15  print ("4.str2[-2:]: ", str2[-2:])
16  print ("5.str2[:-2]: ", str2[:-2])
```

以上执行结果如图 2.4 所示。

```
========== RESTART: D:\strsub.py ==========
str1原文:  Hello World!
str2原文:  I'm fine,and you?
1.str1[1]:  e
2.str1[2:5]:  llo
3.str1[-2:1]:
4.str1[5:5]:
1.str2[0]:  I
2.str2[:5]:  I'm f
3.str2[6:]:  ne,and you?
4.str2[-2:]:  u?
5.str2[:-2]:  I'm fine,and yo
>>>
```

图 2.4　字符串的截取

代码定义了两个字符串 str1 和 str2，并分别对它们进行了不同形式的字符串截取操作。str1 有 4 个操作：

- str1[1]，表示截取第几个字符，这个编号从 0 开始，0 表示第 1 个字符，1 表示第 2 个字符。
- str1[2:5]，表示截取从第 1 个参数开始到第 2 个参数前一位的字符。注意，这是从第几位到第几位，并不是从第 2 位截取长度为 5 的字符。
- str1[-2:1]，从输出结果看这个并没有输出任何字符。-2 表示从倒数第 2 位开始选择，这里要注意倒数开始的时候编号不是从 0 开始的。
- str1[5:5]，没有输出任何结果。第 1 个参数的值必须小于第 2 个参数。

str2 有 5 个操作：

- str2[0]，表示截取第 1 个字符。
- str2[:5]，第 1 个参数为空，表示从头开始截取。
- str2[6:]，第 2 个参数为空，表示截取到字符串的最后。
- str2[-2:]，表示从倒数第 2 个开始截取，一直到字符串的最后。
- str2[:-2]，表示从头开始截取，一直到倒数第 2 位。

说　明
这种截取部分数据的功能有一个专门的概念，叫切片（Slice）。从倒数开始截取数据的功能也叫倒数切片。大部分语言的字符串操作都支持切片功能，Python 中很多数据类型都支持切片。

2.4.3　字符串的拼接

字符串的拼接有 3 种方法：+符号、join 函数和格式化拼接。

（1）使用+符号拼接比较简单，代码如下：

【示例 2-5】

```
01  str1 = 'Hello'
02  str2 = "Python"
03  str3 = "!"
04  strjoin1=str1+" "+str2+" "+str3
```

```
05  print ("1.+拼接: ", strjoin1)              # 1.+拼接:  Hello Python !
```

因为使用+符号拼接后的字符串需要划定新的内存空间来存储，所以普遍认为这种拼接方式效率低，尤其是越多的字符串拼接效率就越低。

（2）使用 join 函数拼接则稍显复杂，函数语法如下：

```
str.join(sequence)
```

sequence 是要拼接的字符串序列，str 是拼接需要使用的字符，如空格、逗号等。因此，在使用该函数时要定义两个变量，举例如下：

【示例 2-6】

```
01  strq=('Hello','Python','!')
02  strflag=" "
03  strjoin2=strflag.join(strq)
04  print ("2.join拼接: ", strjoin2)           # 2.join拼接:  Hello Python !
```

（3）格式化拼接需要用到格式化符号，我们在这里先举例，具体格式化符号的使用，后面会详细介绍。%s 表示需要字符串参数，中间的%表示这是一段格式化输出，后面的参数用括号封闭，用逗号间隔。

【示例 2-7】

```
01  str1 = 'Hello'
02  str2 = "Python"
03  str3 = "!"
04  strjoin3='%s %s %s' % (str1,str2,str3)
05  print ("3.格式化拼接: ", strjoin3)          # 3.格式化拼接:  Hello Python !
```

2.4.4 字符串的各种常用运算符

前面学习过数字运算符有+、-、*、/等，字符串也是有运算符的，不过不是加减乘除，而是一些针对字符串操作的符号，如前面介绍过的+、[]、[:]都是字符串的运算符。常用的字符串运算符参见表 2.2。

表 2.2 字符串常用运算符

符号	说明
+	字符串拼接
*	重复输出字符串，加入 str='hello'，如果是 str*3，就表示输出 3 次 hello
[]	通过索引获取字符串中字符
[:]	截取字符串中的一部分
in	如果字符串中包含给定的字符，就返回 True，如'H' in str1 返回 True
not in	如果字符串中不包含给定的字符，就返回 True，如'H' not in str1 返回 False
r/R（大小写都可以）	所有的字符串都是直接按照字面的意思使用，没有转义特殊或不能打印的字符。使用方法是在字符串的第一个引号前加上字母"r"
%s	格式字符串

这里要特别讲解一下 r 的使用，字符串有很多转义符号，如\n 表示换行、\r 表示回车，如果我

们要在字符串中显示\n，而不是当作换行使用，此时就需要在字符串前添加 r，比如：

```
print (r"这里介绍\n 的使用")
print ("这里介绍\n 的使用")
```

这两行的输出结果如图 2.5 所示，其中第 2 个输出中有换行操作。

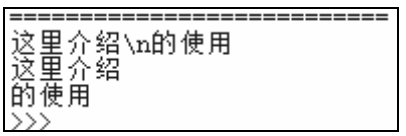

图 2.5　有转义和没有转义的对比

2.4.5　字符串的转义

字符串中的转义就是说字符本身并不是它原来的字面意思。字符串的转义符号都是以\开头的，常用的转义符号参见表 2.3。

表 2.3　转义符号

字符	说明
\(用在字符串的行尾)	表示下一行和当前行是同一行
\\	\符号本身
\'	单引号
\"	双引号
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\b	退格（Backspace 键）
\0nn	八进制数 nn 代表的字符，如\012 代表换行
\xnn	十六进制数 nn 代表的字符，如\x0A 代表换行

注　意
八进制是\0（零），不是字母 o。

这里要特别说明一下\0 和\x，它们后面跟的是代表某个字符的数据，而这个数据来自 ASCII 码表，如图 2.6 所示是部分表 2.3 的内容，圈出的位置就是表 2.3 中的换行符，比如\010 代表退格键，\x0D 代表回车键。

Bin(二进制)	Oct(八进制)	Dec(十进制)	Hex(十六进制)	缩写/字符	解释
0000 0000	0	0	00	NUL(null)	空字符
0000 0001	1	1	01	SOH(start of headline)	标题开始
0000 0010	2	2	02	STX (start of text)	正文开始
0000 0011	3	3	03	ETX (end of text)	正文结束
0000 0100	4	4	04	EOT (end of transmission)	传输结束
0000 0101	5	5	05	ENQ (enquiry)	请求
0000 0110	6	6	06	ACK (acknowledge)	收到通知
0000 0111	7	7	07	BEL (bell)	响铃
0000 1000	10	8	08	BS (backspace)	退格
0000 1001	11	9	09	HT (horizontal tab)	水平制表符
0000 1010	12	10	0A	LF (NL line feed, new line)	换行键
0000 1011	13	11	0B	VT (vertical tab)	垂直制表符
0000 1100	14	12	0C	FF (NP form feed, new page)	换页键
0000 1101	15	13	0D	CR (carriage return)	回车键

图 2.6　ASCII 码表

我们来看一个例子。

【示例 2-8】

```
01   print ( "这里仅输出斜杠\\")
02   print ( "这里是 Tab 缩进"+"\t"+"的距离")
03   print ( "这里介绍换行\n 的使用")
04   print ( "这里介绍换行\012 的使用")
05   print ( "这里介绍换行\x0A 的使用")
06   print ( r"这里仅输出\t\r")
```

上述代码的执行结果如图 2.7 所示。

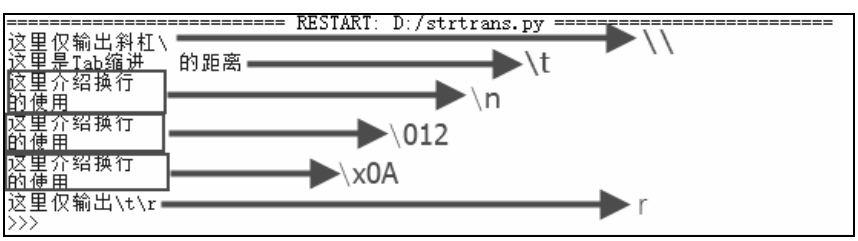

图 2.7　各种转义符号的使用

2.4.6　字符串的格式化符号

在日常开发中，使用 print 函数经常会用到字符串的格式化符号，如果要输出一个字符串，就用%s；如果要输出一个整数，就用%d。常见的格式化符号参见表 2.4。

表 2.4　格式化符号

格式化符号	说明
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%p	用十六进制数格式化变量的地址
%%	字符%

我们可以把格式化字符串想象成带有几个填空项的模板。比如：

```
____的数据成绩是____
```

第一个填空项是人名（字符串），第二个填空项是分数（整数），在代码中就是：

```
%s 的数据成绩是%d
```

输出后的内容与模板一模一样，只将填空的内容填上即可。下面在代码中输入填空项：

```
print( '%s 的数据成绩是%d'%('王丽华',98))                    # 王丽华的数据成绩是 98
```

在模板和具体内容之间有一个%，表示这是一个格式化操作。具体填写的内容用括号封闭起来，多个参数之间用逗号间隔。如果只有一个参数，就可以省略括号。

我们还可以在格式化符号前添加几类符号：

- -，左对齐标志，默认为右对齐。
- +，表示应该包含数字的正负号。
- 0，表示用 0 填充。

下面再看一段代码：

【示例 2-9】

```
01   print( '%5s 的数据成绩是%03d'%('王丽华',98))
02   print( '%-5s 的数据成绩是%03d'%('王丽华',98))
03   print( '今天的温度是%+3d'%+30)
04   print( '今天的温度是%3d'%+30)
```

输出结果如图 2.8 所示。在第 1 行的输出中，%5s 表示字符串长度为 5，但因为给出的参数只有 3 位，所以前面补充了两个空格进行输出（默认用空格填充）；%03d 表示长度为 3 的整数，长度不够时前面用 0 填充，输出的是 098。默认情况下，数值的+号是不输出的，但如果使用+这个符号，则正负号都会输出。

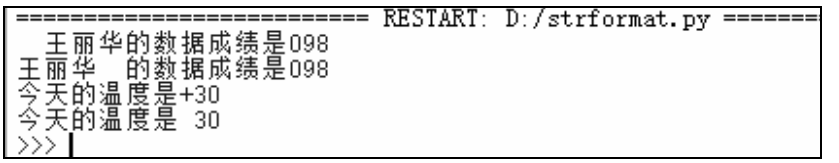

图 2.8 格式化输出

除了各种符号外，Python 也提供了一个格式化函数 format。它通过“{}”和“:”代替传统%操作。format 函数的特色就是可以接受无限个参数，而且位置可以不按模板的顺序，这和我们前面所说的填空的例子略有不同。

【示例 2-10】

```
01  print( '{}的数据成绩是{}'.format('王丽华',98))
02  print( '{0}、{1}的数据成绩都是{2}'.format('王丽华','刘晓娜',98))
03  print( '{0}的数据成绩是{2}，{1}的数据成绩是{2}'.format('王丽华','刘晓娜',98))
```

这里用“{}”表示模板中每个填空的位置，如果不按固定的顺序，就可以按{0}指定顺序，如第 3 行代码的{2}使用了两次，也就是第 3 个参数被调用两次。

“{}”和“:”的组合通常用于数值的格式化，表 2.5 是一些通用的格式化写法。

表 2.5 格式化写法

格式	说明
{:.2f}	保留小数点后两位，如 print("{:.2f}".format(3.1415926))最后输出的是 3.14
{:+.2f}	保留小数点后两位，保留正负号
{:.0f}	保留整数，不带小数位
{:0>3d}	指定长度为 2，不够时左边填充 0
{:0<3d}	指定长度为 2，不够时右边填充 0

下面是一段代码：

【示例 2-11】

```
print("{:.2f}".format(3.1415926))
01  print("{:+.2f}".format(3.1415926))
02  print("{:.0f}".format(3.1415926))
03  print("{:0>3d}".format(30))
04  print("{:0<3d}".format(30))
```

输出结果如图 2.9 所示。

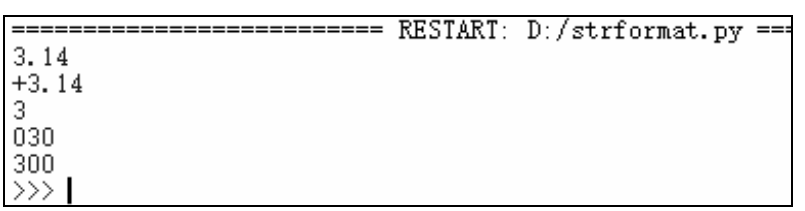

图 2.9 格式化数字

2.4.7　字符串的内置函数

函数的使用想必读者已经不再陌生，前面也介绍过 join 拼接字符串、format 格式化字符串等函数。常用的字符串内置函数还包括表 2.6 所示的这些。

表 2.6　内置函数

函数	说明
capitalize()	将字符串的首字母转换为大写
center(width, fillchar)	返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格
count(str, start= 0,end=len(string))	返回 str 在 string 里面出现的次数，如果 start 或 end 指定，就返回指定范围内出现的次数
encode(encoding='UTF-8',errors='strict')	以 encoding 指定的编码格式编码字符串，如果出错默认报一个 ValueError 的异常，那么 errors 还可以指定 ignore（忽略）或 replace（替换指定内容）
endswith(obj, start=0, end=len(string))	字符串是否以 obj 结束，如果 start 或 end 指定，就检查指定的范围内是否以 obj 结束，如果是，返回 True，否则返回 False
expandtabs(tabsize=8)	把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8
find(str, start=0 end=len(string))	检测 str 是否包含在字符串中，如果指定范围 start 和 end，就检查是否包含在指定范围内，如果包含，就返回开始的索引值，否则返回-1
index(str, start=0, end=len(string))	跟 find()方法一样，只不过 str 不在字符串中时会报错
isalnum()	如果字符串至少有一个字符并且所有字符都是字母或数字，就返回 True，否则返回 False
isalpha()	如果字符串至少有一个字符并且所有字符都是字母，就返回 True，否则返回 False
isdigit()	如果字符串只包含数字，就返回 True，否则返回 False
islower()	如果字符串中包含至少一个区分大小写的字符，并且所有这些（区分大小写的）字符都是小写，就返回 True，否则返回 False
isnumeric()	如果字符串中只包含数字字符，就返回 True，否则返回 False
isspace()	如果字符串中只包含空白，就返回 True，否则返回 False
istitle()	如果字符串是“标题化”的，就返回 True，否则返回 False
isupper()	如果字符串中包含至少一个区分大小写的字符，并且所有这些（区分大小写的）字符都是大写，就返回 True，否则返回 False
isdecimal()	检查字符串是否只包含十进制字符，如果是，就返回 True，否则返回 False
len(string)	返回字符串的长度
ljust(width[, fillchar])	返回一个原字符串左对齐，并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格
lower()	转换字符串中所有大写字符为小写
lstrip([chars])	删除字符串左侧的空格或指定的字符
max(str)	返回字符串 str 中最大的字母
min(str)	返回字符串 str 中最小的字母

（续表）

函数	说明
replace(old, new [, max])	把字符串中的 old 替换成 new，如果指定 max，就替换不超过 max 次
rfind(str, start=0,end=len(string))	类似于 find()函数，从右边开始查找
rindex(str, start=0, end=len(string))	类似于 index()函数，从右边开始查找
rjust(width,[, fillchar])	返回一个原字符串右对齐，并使用 fillchar（默认空格）填充至长度 width 的新字符串
rstrip([chars])	删除字符串右侧的空格或指定的字符
split(str="", num=string.count(str))	以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num 个子字符串
splitlines([keepends])	按照行（'\r'，'\r\n'，\n'）分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符
startswith(obj, start=0,end=len(string))	检查字符串是否以 obj 开头，是就返回 True，否则返回 False。如果 start 和 end 指定值，就在指定范围内检查
strip([chars])	在字符串上执行 lstrip()和 rstrip()函数
swapcase()	将字符串中的大写字母转换为小写、小写字母转换为大写
title()	返回"标题化"的字符串，就是说所有单词都是以大写开始，其余字母均为小写
translate(table, deletechars="")	根据 table 给出的翻译表（包含 256 个字符）转换 string 的字符，要过滤掉的字符放到 deletechars 参数中
upper()	转换字符串中的所有小写字母为大写
zfill (width)	返回长度为 width 的字符串，原字符串右对齐，前面填充 0

这里要特别说明的是 translate 函数，它需要一个 table 参数，这个 table 一般称为翻译表或转换表。比如根据最新广告法规定，不允许使用“最好的”“最厉害的”这类用语，还有网站也经常需要过滤一些不文明用语，这时候就可以制作一个翻译表，比如将“最”替换为“*”。翻译表使用 maketrans 函数制作，下面给一个详细的例子：

【示例 2-12】

```
01  intab = "最"
02  outtab = "*"
03  trantab = str.maketrans(intab, outtab)
04  str1="这是最好的一次体验。"
05  print (str1.translate(trantab))
```

上述代码输出结果如图 2.10 所示。首先使用 maketrans 函数制作一个翻译表(“最”翻译为“*”)，然后使用 translate 函数输出翻译后的字符串。

```
========== RESTART: D:/strfun.py =====
这是*好的一次体验。
>>>
```

图 2.10　翻译表

2.5　列　　表

列表表示一组有序的元素，比如每个学生有学号、姓名、性别这 3 个属性，我们就可以把这 3 个属性放在一个列表中。Python 中的列表类似其他语言的数组，细节上有所不同，但存储概念上类似。本节将介绍列表的使用。

2.5.1　使用列表

1. 普通定义

列表使用[ ]闭合定义，比如一个学生列表：

```
stu1=['10001','张晓光','男']
```

也可以定义一个成绩列表，全部是数字：

```
score1=[45,68,98]
```

列表中的元素可以是任意数据类型，也可以多种数据类型混合使用，给学生列表添加年龄属性：

```
stu1=['10001','张晓光','男',20]
```

列表也可以初期定义为空，然后使用列表的 append 函数向其追加元素，该函数只能包括一个参数。

```
stu1=[]
stu1.append('张晓光')
```

相信很多读者听说过二维数组，就是数组的元素也是数组。同理，列表的元素也可以是列表，我们也可以称之为二维列表。它使用[[],[],[]]这种方式，下面创建 3 条学生信息：

```
stu1=[['10001','张晓光','男',20],['10002','李淑霞','女',21],['10003','王心智','男',19]]
```

也可以看起来更直接一些：

```
stu1=[
    ['10001','张晓光','男',20],
    ['10002','李淑霞','女',21],
    ['10003','王心智','男',19]
]
```

要访问第 2 个学生的年龄，可以使用 stu1[1][3]，第 1 个[]表示访问第几行，第 2 个[]表示访问第几列，也就是使用 stu1[row][col]来访问二维列表中的元素。

2. 快速定义

列表还可以使用快速定义的方式：

```
stu1=[0]*width
```

0 是默认的内容，width 是个数，比如要定义 4 个元素的列表：

```
stu1=[0]*4
```

创建学生列表，然后逐个赋值：

【示例 2-13】

```
01  stu1=[0]*4
02  print('学生：',stu1)
03
04  stu1[0]='10001'
05  stu1[1]='张晓光'
06  stu1[2]='男'
07  stu1[3]=20
08  print('学生：',stu1)
```

输出结果为：

```
学生： [0, 0, 0, 0]
学生： ['10001', '张晓光', '男', 20]
```

那么，二维列表是否也可以这样快速定义呢？

先来快速定义一个 3 行 4 列的列表，然后给第 1 行第 1 列赋值'10001'：

【示例 2-14】

```
01  stu1=[[0]*3]*4
02  stu1[0][0]='10001'
03  print('学生：',stu1)
```

我们会以为输出结果只是第 1 行第 1 列的值改变了，但输出结果却是：

```
学生： [['10001', 0, 0], ['10001', 0, 0], ['10001', 0, 0], ['10001', 0, 0]]
```

第 1 列的结果全部改变了。因为[[]]是一个含有一个空列表元素的列表，[[]]*4 表示 4 个指向这个空列表元素的引用，所以修改任何一个元素都会改变整个列表。

那如何快速定义二维列表呢？

【示例 2-15】

```
01  stu1=[([0] * 3) for i in range(4)]
02  stu1[0][0]='10001'
03  print('学生：',stu1)
```

此时输出结果如下，正是我们需要的结果。

```
学生： [['10001', 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

这是通过一种循环的方式创建二维列表，相信读者学完 for...in 语句后会有更深的感悟。

2.5.2 访问列表

使用列表后，经常需要访问其中的某一个元素，Python 中直接用索引访问，索引从 0 开始，负数表示从倒数的位置开始。比如访问第 1 个元素，用[0]，访问倒数第 1 个元素，用[-1]：

```
stu1=['10001','张晓光','男',20]
```

```
print('学生学号：%s，学生年龄：%d'%(stu1[0],stu1[-1]))

# 学生学号：10001，学生年龄：20
```

如果要访问部分元素，可以使用[start:end]切片形式，从 start 位置开始访问，一直到 end 位置之前，比如[1:3]就是访问第 2 个元素到第 4 个元素之前，就是 2、3 两个元素。

```
stu1=['10001','张晓光','男',20]
print('学生：',stu1[1:3])

# 学生：  ['张晓光', '男']
```

2.5.3　列表常用的内置函数

Python 为列表提供了一系列函数，如追加、插入、排序、移除等，常见的函数参见表 2.7。

表 2.7　列表常用的内置函数

名称	说明
list.append()	追加
list.count(x)	计算列表中参数 x 出现的次数
list.extend(L)	向列表中追加另一个列表 L
list.index(x)	获得参数 x 在列表中的位置
list.insert(x,y)	向列表中的 x 位置插入数据 y
list.pop([index])	删除列表中 index 位置的元素（通过下标删除）
list.remove(x)	删除列表中的指定元素 x（直接删除）
list.reverse()	将列表中元素的顺序颠倒
list.sort()	将列表中的元素进行排序

还是使用前面创建的列表 stu1，下面演示几个函数的使用：

【示例 2-16】

```
01  stu1=['10001','张晓光','男',20];
02  print('学生：',stu1)
03
04  #1.追加城市
05  stu1.append('上海')
06  print('1.',stu1)
07
08  #2.追加另一个列表
09  stu2=['10002','李淑霞','女',21,'上海']
10  stu1.extend(stu2)
11  print('2.',stu1)
12
13  #3.上海在列表中出现的次数
14  print('3.',stu1.count('上海'))
15
16  #4.获取位置
17  print('4.',stu1.index('李淑霞'))
18
19  #5.插入开头
```

```
stu1.insert(0,'学生信息：')
print('5.',stu1)

#6.pop 移除
stu1.pop(0)
#stu1.pop()
#stu1.pop(-1)
print('6.',stu1)

#7.remove 移除
stu1.remove('上海')
print('7.',stu1)

#8.reverse 反转
stu1.reverse()
print('8.',stu1)

#9.sort 排序
stu1.sort()
print('9.',stu1)
```

上述代码的输出结果如图 2.11 所示。

```
========== RESTART: D:/list1.py ==========
学生： ['10001', '张晓光', '男', 20]
1. ['10001', '张晓光', '男', 20, '上海']
2. ['10001', '张晓光', '男', 20, '上海', '10002', '李淑霞', '女', 21, '上海']
3. 2
4. 6
5. ['学生信息：', '10001', '张晓光', '男', 20, '上海', '10002', '李淑霞', '女', 21, '上海']
6. ['10001', '张晓光', '男', 20, '上海', '10002', '李淑霞', '女', 21, '上海']
7. ['10001', '张晓光', '男', 20, '10002', '李淑霞', '女', 21, '上海']
8. ['上海', 21, '女', '李淑霞', '10002', 20, '男', '张晓光', '10001']
Traceback (most recent call last):
  File "D:/list1.py", line 69, in <module>
    stu1.sort()
TypeError: '<' not supported between instances of 'int' and 'str'
>>>
```

图 2.11　列表函数应用

这里有几点要特别说明：

（1）使用 pop 移除元素时，指定的索引可以为空，也可以为正负数。为空时，是移除列表最后的元素；为负数时，是从倒数位置移除指定的元素。

（2）追加另一个列表时，追加的内容并不会与原列表组成二维列表，而是在原列表中追加一些元素，因此获取位置时出现的是 6，而不是 stu1[][]这种形式。

（3）使用 sort 函数时出现了如下错误，因为列表中元素的类型不限定，本例中包含了字符串和数字，所以这里出现运行错误。

```
stu1.sort()
TypeError: '<' not supported between instances of 'int' and 'str'
```

要了解这个错误，笔者把 sort 函数单独列出来介绍，继续下一节。

2.5.4　列表排序

在 Python 中，排序使用 sort，语法如下，这两个参数都是可选的。

```
sort (key=None, reverse=False)
```

1. 普通排序（升序或降序）

如果列表中的元素是统一的单一数据类型，就直接使用 sort()，默认升序。例如：

【示例 2-17】

```
stu1=['10001','男','张晓光'];
stu1.sort()
print('学生：',stu1)          # 学生： ['10001', '张晓光', '男']

score1=[80,100,98,59]
score1.sort();
print('成绩：',score1)        # 成绩： [59, 80, 98, 100]
```

默认为升序，要是使用降序呢？sort 函数中有一个 reverse 参数，将其设置为 True，就可以实现降序：

【示例 2-18】

```
stu1=['10001','男','张晓光'];
stu1.sort(reverse=True)
print('学生：',stu1)     # 学生： ['男', '张晓光', '10001']

score1=[80,100,98,59]
score1.sort(reverse=True);
print('成绩：',score1)        # 成绩： [100, 98, 80, 59]
```

2. 副本排序

如果要排序，但并不修改原来的内存位置，这个时候就要用到副本排序。先来看一段代码：

【示例 2-19】

```
score1=[80,100,98,59]
score2=score1
score2.sort()
print('成绩：',score2)
print('位置：',id(score1),id(score2))
```

输出结果：

```
成绩 1： [59, 80, 98, 100]
成绩 2： [59, 80, 98, 100]
位置： 2134350122056 2134350122056
```

如果直接使用赋值的方式创建一个副本，发现两者的存储位置一致，都进行了排序，这个时候并不能实现真正的副本排序。我们需要用切片方式[:]进行赋值：

【示例 2-20】

```
score1=[80,100,98,59]
score2=score1[:]
score2.sort()
```

```
print('成绩1：',score1)
print('成绩2：',score2)
print('位置：',id(score1),id(score2))
```

输出结果如下：

```
成绩1： [80, 100, 98, 59]
成绩2： [59, 80, 98, 100]
位置： 2948369107848 2948359917512
```

可以看到 score1 并没有被排序，而且两者的位置也不相同。

3. 复杂排序

sort 函数中有一个 key 参数，用于定义排序过程中调用的函数。key 是带一个参数的函数，用于为每个元素提取比较值，默认为 None，即直接比较每个元素。比如要通过列表中元素的长度进行排序：

```
stu1=['10001','男','张晓光'];
stu1.sort(key=len)
print('学生：',stu1)     # 学生： ['男', '张晓光', '10001']
```

key 指定的函数只能有一个输入参数，也只能有一个返回值。可以使用 Python 自带的函数，比如上述代码的 len，也可以自定义函数。

下面自定义一个函数 comp，有一个输入参数 x。函数中先用 type 判断 x 的数据类型，如果不是 str（字符串）类型，就返回'0'，如果是 str 类型，就直接返回 x，即保持原来的值不变。

【示例 2-21】

```
stu1=['10001','男','张晓光',20];
def comp(x):
    if type(x) is not str:
        return '0'
    else:
        return x

stu1.sort(key=comp)
print('学生：',stu1)        # 学生： [20, '10001', '张晓光', '男']
```

代码中使用 key=comp 让 stu1 列表的排序按我们自定义的方式。因为列表中既有字符串又有数字，而自定义函数的意思是将数字换为'0'，也就是让其保持最小值，以方便排序时保证它在前面的位置，所以最终实现了数字在前、字符串在后的排序。

2.5.5 删除列表

前面介绍列表函数时，使用 remove、pop 这两个函数删除列表中的元素。本小节再介绍两个函数，即 del 和 clear。

clear 也是列表的函数，不是删除某个元素，而是清空列表：

```
stu1=['10001','男','张晓光'];
stu1.clear()
print('学生：',stu1)         # 学生： []
```

del 用法和 pop 类似，也是删除指定索引的位置，并可以使用切片方法删除。

```
stu1=['10001','男','张晓光'];
del stu1[0]
del stu1[-1]
#del stu1[0:3]
print('学生：',stu1)        # 学生：  ['男']
```

提　示

如果 stu1 不指定删除的位置，直接使用 del stu1 会删除整个列表，此时如果再访问 stu1，会提示该变量未定义的错误。读者请自行测试一下。

2.5.6　获取列表中的最大值和最小值

Python 提供了几个函数用于操作列表，如获取列表的个数、最大值、最小值等，这些使用方法比较简单，这里简单讲解一下。

- len(list)：列表元素个数。
- max(list)：返回列表中元素的最大值。
- min(list)：返回列表中元素的最小值。

直接写一段代码：

【示例 2-22】

```
01   score1=[80,100,98,59]
02   print('%d 个成绩'%len(score1))
03   print('成绩最高：',max(score1))
04   print('成绩最低：',min(score1))
```

这里定义一个成绩列表，然后选出所有成绩的最大值和最小值，结果如下：

```
4 个成绩
成绩最高：  100
成绩最低：  59
```

2.5.7　列表常用运算符

数字有+、-、*、/等运算符，字符串有+、-、[]、[:]等运算符，列表也有一些运算符，如表 2.8 所示。

表 2.8　列表常用运算符

名称	说明
+	运算符两侧的列表组合在一起
*	根据右侧数字重复运算符左侧的列表
in	判断运算符左侧的元素是否属于右侧的列表

演示一段代码：

【示例 2-23】

```
stu1=['101','张晓光']
stu2=['男',20]

print('stu1+stu1:',stu1+stu2)
print('stu1*2:',stu1*2)
print('101 in stu1:','101' in stu1)
```

输出结果如下：

```
stu1+stu1: ['101', '张晓光', '男', 20]
stu1*2: ['101', '张晓光', '101', '张晓光']
101 in stu1: True
```

2.6 元　　组

元组是一组有序的元素，各种使用方法与列表类似，只是元组中的元素一旦定义，就不能更改。本节将学习元组的使用，其中学习过程与上一节类似，读者可以通过对比加深印象。

2.6.1 使用元组

元组的定义使用()，列表使用[]。我们知道列表的定义方式有以下两种：

```
stu1=['10001','张晓光','男']
stu1=[0]*width
```

元组是否也可以这样定义呢？

当然不可以，因为元组的元素是不能改变的，我们无法先使用[0]来定义元素，后期再做更改。因此使用元组的方式是：

```
stu1=('10001','张晓光','男')
```

注　　意

也可以使用 stu1=()的形式定义空元组，但因为元组内的元素不可以改变，所以这样基本没有意义。

元组还有一种更简单的定义方式。在 Python 中，默认用逗号间隔的一组元素自动会定义为元组。比如下面这段代码：

```
stu1='10001','张晓光','男'
```

我们直接在 Python 的交互模式解释器中输出 stu1 的类型，会显示它是一个 tuple（元组），如图 2.12 所示。

```
>>> stu1='10001','张晓光','男',20
>>>
>>>
>>> type(stu1)
<class 'tuple'>
>>>
```

图 2.12　输出元组类型

2.6.2　访问元组

访问元素也是使用[]索引的方式，0、正负数都可以，也支持[:]这种切片方式。下面举例说明：

```
stu1=('10001','张晓光','男',20)
print('学生学号：%s，学生年龄：%d'%(stu1[0],stu1[-1]))
print('学生：',stu1[1:3])
```

输出结果如下：

```
学生学号：10001，学生年龄：20
学生：  ('张晓光', '男')
```

这里要注意的是，如果 stu1 是一个列表，用[:]切片形式输出时就显示['张晓光','男']；如果 stu1 是一个元组，输出时就显示('张晓光','男')。读者要注意[]和()的区别。

2.6.3　元组常用的内置函数

因为元组不可修改，所以列表中有的追加、插入、移除等函数，元组都没有。常用的元组函数只有两个：

- count()：查找指定元素在元组中出现的次数。
- index()：查找指定元素第一次在元组出现的索引值。

举例如下：

```
stu1='10001','张晓光','男',20,20
print('20 出现的次数',stu1.count(20))  # 2
print('20 出现的位置',stu1.index(20))  # 3，返回第一次出现的位置
```

index()除了可以在整个元组中查找，还可以在指定开始位置和结束位置之间的元素块中查找，比如在 stu1 的最后 3 个元素中查找。

```
stu1.index(20,3,5)   # 3
```

注　　意

不管是在最后 3 个元素中查找，还是在所有元素中查找，index 函数返回的位置都是该元素在整个元组中的位置。

2.6.4 删除元组

元组没有 clear 清空函数，也不能使用 del 指定要删除的索引位置。删除元组只需要一句 del，如图 2.13 所示。

```
del stu1
```

```
>>> stu1='10001','张晓光','男',20
>>> del stu1
>>> print(stu1)
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    print(stu1)
NameError: name 'stu1' is not defined
>>>
```

图 2.13 删除元组

删除 stu1 元组后，如果再次访问该元组，系统会给出 is not defined 的未定义错误。

2.6.5 获取元组中的最大值和最小值

也可以使用 min、max 获取元组中的最小值和最大值。直接看一段代码：

【示例 2-24】

```
01  score1=(80,100,98,59)
02  print('%d 个成绩'%len(score1))
03  print('成绩最高：',max(score1))
04  print('成绩最低：',min(score1))
```

输出结果如下：

```
4 个成绩
成绩最高： 100
成绩最低： 59
```

2.6.6 元组常用运算符

元组的运算符和列表的运算符基本一致，常用的也是+、*、in。

- +：生成一个新的元组。
- *：重复几次。
- in：判断是否包含指定的元素。

还是直接举例：

【示例 2-25】

```
01  stu1=('101','张晓光')
02  stu2=('男',20)
03
04  print('stu1+stu1:',stu1+stu2)
05  print('stu1*2:',stu1*2)
```

```
06   print('101 in stu1:','101' in stu1)
```

输出结果如下：

```
stu1+stu1: ('101', '张晓光', '男', 20)
stu1*2: ('101', '张晓光', '101', '张晓光')
101 in stu1: True
```

2.6.7　元组与列表的转换

在 Python 中，元组类型和列表类型可以互换。

当将一个列表转换为元组时，使用 tuple(seq)函数，此时列表还是列表，会创建一个新元组。当将一个元组转换为列表时，使用 list(seq)函数，此时元组还是元组，会创建一个新列表。虽然意思有点拗口，但读者一定要注意，并不是原有的列表或元组发生了根本性的改变，只是创建了一个新对象。

1. 列表转元组

列表转元组使用 tuple 函数：

```
stu1=['10001','张晓光','男',20]
tup1=tuple(stu1)        #转元组
print(type(stu1))       # <class 'list'>
print(type(tup1))       # <class 'tuple'>
```

2. 元组转列表

元组转列表使用 list 函数：

```
stu1=('10001','张晓光','男',20)
list1=list(stu1)        #转列表
print(type(stu1))       # <class 'tuple'>
print(type(list1))      # <class 'list'>
```

2.7　字　　典

字典的概念来自英文 Dictionary 的翻译，意思是一对 key-value 的组合值，通常称为键-值对。本节将介绍 Python 中的字典。

2.7.1　使用字典

在 Python 中，使用{}定义字典，字典中的键-值对用冒号间隔，比如定义一个字典：

```
stu1={'学号':'10001','姓名':'张晓光','性别':'男','年龄':20}
```

在字典中，键是不可变的（数字、字符串、元组），但值是可以改变的，比如要改变上述字典中的年龄为 30，可以这样写：

```
stu1['年龄']=30
```

注　意

字典中的值可以是任意类型，因为键是不可改变的，所以不能是列表等可变类型。

在字典中，键是唯一的，虽然定义字典时允许输入两个相同的键，但实际上后一个键的值会覆盖上一个键的值，比如以下代码定义了重复的“姓名”键：

```
stu1={'学号':'10001','姓名':'张晓光','姓名':'李三','年龄':20}
```

stu1['姓名']输出的结果会是‘李三’。

上面我们演示的键都是字符串，其实键还可以是数字或元组，比如下面定义一组数字键：

```
day={1:'星期一',2:'星期二',3:'星期三'}

print(day[1])
```

也可以使用混合类型的键，比如既有数字键又有字符串的键：

```
day={1:'星期一',2:'星期二',3:'星期三','四':'星期四'}

print(day['四'])
```

2.7.2　访问字典

访问序列中的元素基本都用[]，字典也不例外。因为元素是键-值对，所以[]中还需要指定要访问的键。比如 stu1['姓名']就是访问“姓名”键所对应的值。

注　意

访问列表或元组时，可以使用 stu1[索引]的方式，但字典中并不可以，比如使用索引[0]并不会访问第 1 个元素，而是访问键为 0 的元素。

下面举例：

```
day={1:'星期一',2:'星期二',3:30,'四':'星期四'}

print(day['四'])        #星期四
print(day[2])           #星期二
```

2.7.3　字典常用的内置函数

字典中包括一些返回值、返回键的方法，如表 2.9 所示。

表 2.9　字典常用的内置函数

名称	说明
dict.copy()	返回一个字典的深拷贝
dict.fromkeys(seq[, value]))	创建一个字典，以序列 seq 中的元素做字典的键，value 可省略，为字典所有键对应的初始值，如果省略，值就为 None
dict.get(key, default=None)	返回指定键的值，如果值不在字典中，就返回 default 默认值

（续表）

名称	说明
dict.items()	以列表形式返回可遍历的(键,值)
dict.keys()	以列表形式返回一个字典所有的键
dict.setdefault(key, default=None)	和 get()类似，但如果键不存在于字典中，就会添加键并将值设为 default
dict.update(dict2)	把指定字典的键/值对更新到当前字典中
dict.values()	以列表形式返回字典中的所有值

说　明

一旦定义一个变量，其在内存中占据的位置一般不可变，都是通过一个引用指向该变量。如果复制某个变量，也只是增加一个引用，具体位置还是不变，这个时候就是浅拷贝；如果增加引用的同时具体位置也发生了变化，这种称为深拷贝。

下面举例：

【示例 2-26】

```
dict1={'姓名':'张晓光','年龄':20}

print('1.所有键：',dict1.keys())
print('2.所有值：',dict1.values())
print('3.所有键-值：',dict1.items())

dict2=dict1
dict3=dict1.copy()
print('4.浅拷贝和深拷贝：',id(dict1),id(dict2),id(dict3))

score1=(1,2,3,4)
dict4=dict1.fromkeys(score1)
print('5.通过元组创建字典：',dict4)
print('6.get 年龄：',dict1.get('年龄'))

dict1.setdefault('年龄',30)
print('7.setdefault 年龄：',dict1)

dict5={'成绩':'优良'}
dict1.update(dict5)
print('8.update 成绩：',dict1)
```

输出结果如图 2-14 所示。

```
========== RESTART: D:/dict1.py ==========
1.所有键： dict_keys(['姓名', '年龄'])
2.所有值： dict_values(['张晓光', 20])
3.所有键-值： dict_items([('姓名', '张晓光'), ('年龄', 20)])
4.浅拷贝和深拷贝： 1996866938632 1996866938632 1996867426848
5.通过元组创建字典： {1: None, 2: None, 3: None, 4: None}
6.get年龄： 20
7.setdefault年纪： {'姓名': '张晓光', '年龄': 20, '年纪': 30}
8.update成绩： {'姓名': '张晓光', '年龄': 20, '年龄': 30, '成绩': '优良'}
>>>
```

图 2.14　字典内置函数应用

这里要特别说明几点：

（1）如果是通过=赋值的方式创建一个新的字典，使用的就是浅拷贝，也就是说，并不开辟一块内存保存新字典。如果是通过 copy 函数创建一个新字典，就是深拷贝方式，通过 id(dict)输出的内存地址信息可以判断。

（2）get 和 setdefault 都可以获取指定键的值，但如果指定的键并不存在，get 就返回默认值 None，而 setdefault 就会将指定的键添加到字典中。

（3）update 是将指定的字典更新到当前字典中，但如果指定的字典中并不包含值，也就是只有键的情况，如{'张三','上海'}这种形式，代码并不报错，而是自动分割键，给当前字典增加键值对，比如增加后是{'张':'三','上':'海'}这种形式。因此，在使用 update 时要检查仔细。

2.7.4 删除字典

删除字典的方法有很多种，如 clear、pop、popitem、del 等，这些有的只能删除字典元素，有的可以删除字典，具体说明如下：

- clear(): 清空字典中所有键-值对。
- pop(key): 删除指定键的键-值对，有返回值，返回值为被删除的值。
- popitem(): 删除最后一项键-值对，有返回值，返回值为被删除的键-值对。
- del: 删除字典元素或字典，如果删除的是字典，再访问字典时会报错。

（1）首先介绍 clear，清空字典后，如果再访问字典，就不会报错，返回{}。

```
dict1={'姓名':'张晓光','年龄':20}
dict1.clear()
print(dict1)            # {}
```

（2）pop 函数在使用时有两个步骤：

- 一是在字典中删除键-值对。
- 二是返回被删除的值。

因为有返回值，所以可以定义一个变量接收该值，例如：

```
dict1={'姓名':'张晓光','年龄':20}
str1=dict1.pop('姓名')
print(str1)             # 张晓光
print(dict1)            # {'年龄': 20}
```

（3）popitem 没有参数，默认是删除最后一项键-值对（返回元组），例如：

```
dict1={'姓名':'张晓光','年龄':20}
tup1=dict1.popitem()
print(tup1)             # ('年龄', 20)
print(dict1)            # {'姓名': '张晓光'}
```

（4）如果 del 指定键，则删除的效果和 popitem 没有区别，但并不返回值；如果 del 不指定键，则会删除整个字典。

```
dict1={'姓名':'张晓光','年龄':20}
del dict1['姓名']              # 删除“姓名”键值对
print(dict1)

del dict1                 # 删除字典
print(dict1)
```

因为第 2 次用 del 删除了字典，所以访问时会报错，如图 2.15 所示。

```
============================== RESTART: D:/dict1.py ==============
{'年龄': 20}
Traceback (most recent call last):
  File "D:/dict1.py", line 51, in <module>
    print(dict1)
NameError: name 'dict1' is not defined
>>>
```

图 2.15　删除字典报错

2.7.5　字典常用运算符

字典没有+、*运算符，只有 in 运算符，用于判断指定的键是否在字典中，或者使用 not in 判断指定的键是否不在字典中。

下面举例：

【示例 2-27】

```
01  dict1={'姓名':'张晓光','年龄':20}
02  if '姓名' in dict1:
03      print(dict1['姓名'])
04
05  if '性别' not in dict1:
06      dict1.setdefault('性别','男')
07      print(dict1)
```

上述代码使用 if 做了两次判断，第 1 次判断字典中是否有“姓名”键，如果有，就输出该键对应的值。第 2 次判断是否没有“性别”键，如果没有，就使用 setdefault 添加该键，并输出当前字典。代码输出结果是：

```
张晓光
{'姓名': '张晓光', '年龄': 20, '性别': '男'}
```

2.8　集　　合

集合是一组无序的不能重复的元素，这个和列表、元组不同，虽然也是一组元素，但因为是无序的，所以无法使用[]索引的方式访问。集合不能重复，其作用就是去重（去掉重复数据）。本节将介绍集合的使用。

2.8.1 使用集合

要使用集合，就需要用 set 函数，这类要注意，其他序列（列表、元组）等都是通过()、[]直接定义，而集合不是，相当于是通过 set 函数将数据转换为集合。因此很多教程也不把集合当作 Python 的标准数据类型。

使用集合的方式如下：

```
set1=set([1,200,39,50])
set2=set(['王晓光','男'])
print(set1)
print(set2)
```

上述集合的输出结果如下：

```
{200, 1, 50, 39}
{'男', '王晓光'}
```

从结果可以看出，集合并没有按定义的顺序输出。在 Python 中，集合是无序的，打印结果取决于其内部存储结构和输出方式。

说　明
也可以先创建一个列表，然后使用 set 函数将列表转化为集合，如 list1=[45,68,98]、set1=set(list1)。

如果不使用[]的方式，直接定义一个字符串集合：

```
set1=set('hello')
print(set1)
```

从输出结果可以看出，重复的字母会被删除：

```
{'o', 'l', 'h', 'e'}
```

集合中的元素不能重复，假如定义以下带有重复元素的集合：

```
set1=set([1,200,200,39,50])
```

在使用时该集合并不会报错，但会默认将重复的值去掉，上述代码输出结果为：

```
{200, 1, 50, 39}
```

2.8.2 集合常用的内置函数

集合虽然是无序的，但在创建后还可以添加、更新等。Python 为集合提供了一些内置函数，如表 2.10 所示。

表 2.10　集合常用的内置函数

名称	说明
add	添加元素
clear	清空元素

（续表）

名称	说明
copy	复制集合
discard	删除指定元素，如果没有也不会报错
pop	随机删除一个元素
remove	删除指定元素，如果没有会报错
update	更新元素

下面举例：

【示例 2-28】

```
set1=set([1,200,39,50])
set1.add(30)                #添加
print('1.add 30: ',set1)

set2=set1.copy()            #复制
print('2.copy: ',set2)

set1.discard(200)           #删除
print('3.discard: ',set1)

set1.pop()                       #随机删除
print('4.pop: ',set1)

set1.remove(39)             #删除，如果没有会报错，终止程序
print('5.remove: ',set1)

set1.update([300,500])      #更新
print('6.update: ',set1)

set1.clear()                #清空
print('7.clear: ',set1)
```

输出结果如图 2.16 所示。

```
============================== RESTART: D:/set1.py ===
1.add 30:  {1, 39, 200, 50, 30}
2.copy:  {1, 50, 39, 200, 30}
3.discard:  {1, 39, 50, 30}
4.pop:  {39, 50, 30}
5.remove:  {50, 30}
6.update:  {300, 50, 500, 30}
7.clear:  set()
>>>
```

图 2.16　集合内置函数应用

这里有几点需要注意：

（1）discard 和 remove 虽然都是删除元素，但如果指定的元素不存在，则 discard 依然会继续执行，remove 会报错终止程序执行。

（2）pop 删除元素时是随机的。

（3）复制集合时，从结果可以看出，复制后的集合顺序和原集合顺序并不相同。

（4）clear 只是清空集合的内容，如果要删除集合，还是使用 del set1 这种方式。

2.8.3 集合常用运算符（交集、并集、差集、对称差集）

在数学中，由一个或多个确定的元素所构成的整体叫作集合。数学中的集合有三大特性：

- 确定性（集合中的元素必须是确定的）
- 互异性（集合中的元素互不相同）
- 无序性（集合中的元素没有先后之分）

通过前面对 Python 中集合的学习，我们会发现，Python 中的集合和数学中的集合一样。数学中的集合有一些运算，如交集、并集等，Python 中也一样，可用来进行一些科学计算。

本小节介绍的集合常用运算符主要有 4 个：

- set1 | set2 ： set1 和 set2 的并集。
- set1 & set2 ： set1 和 set2 的交集。
- set1 - set2 ： 差集（元素在 set1 中，但不在 set2 中）。
- set1 ^ set2 ： 对称差集（元素在 set1 或 set2 中，但不会同时出现在两者中）。

相对于集合的这些数学操作，Python 也提供了一些内置函数用于科学计算，参见表 2.11。

表 2.11 用于科学计算的内置函数

名称	相当于运算符	说明
set1.issubset(set2)	set1 <= set2	检查 set1 中的每一个元素是否都在 set2 中，子集
set1.issuperset(set2)	set1 >= set2	检查 set2 中的每一个元素是否都在 set1 中，父集
set1.union(set2)	set1 | set2	返回一个新的 set，包含 set1 和 set2 中的每一个元素，并集
set1.intersection(set2)	set1 & set2	返回一个新的 set，包含 set1 和 set2 中的公共元素，交集
set1.difference(set2)	set1 - set2	返回一个新的 set，包含 set1 中有但 set2 中没有的元素，差集
set1.symmetric_difference(set2)	set1 ^ set2	返回一个新的 set，包含 set1 和 set2 中不重复的元素，对称差集

下面举例，首先创建两个数据集合：

【示例 2-29】

```
set1=set([1,200,39,50])
set2=set([19,200,3,50,39,1])
print(set1)
print(set2)

print('1.issubset: ',set1.issubset(set2))       #set2 是否是 set1 的子集，True
print('1.set1 <= set2: ',set1 <= set2)          #set2 是否是 set1 的子集，True

print('2.issuperset: ',set1.issuperset(set2))   #set2 是否是 set1 的父集，False
print('3.union: ',set1.union(set2))             #并集
print('4.intersection: ',set1.intersection(set2))    #交集，set1 和 set2 都有的元素
print('5.difference: ',set2.difference(set1))        #差集，set2 中有，set1 中没有
```

```
13  print('6.symmetric_difference: ',set1.symmetric_difference(set2))# 对称差集，不重复的元素
```

输出结果如图 2.17 所示。

```
============================== RESTART: D:/set1.py =====
{200, 1, 50, 39}
{1, 3, 39, 200, 50, 19}
1.issubset:  True
1.set1 <= set2:  True
2.issuperset:  False
3.union:  {1, 3, 39, 200, 50, 19}
4.intersection:  {200, 1, 50, 39}
5.difference:  {19, 3}
6.symmetric_difference:  {3, 19}
>>>
```

图 2.17　科学计算函数的应用

使用这些集合运算时要注意以下几点：

（1）代码中使用差集时，因为本例 set1 集合中的内容都在 set2 中，所以举例时用的 set2.difference(set1))并不是 set1.difference(set2))，其他运算都是 set1.xxx 形式。

（2）因为 issubset 和 issuperset 只是判断，所以返回的是 True 或 False，并不返回新的集合，而其他运算都会返回新的集合，从输出结果中也可以看到。

（3）使用内置函数或使用 -、&、|、<= 等运算符的结果是一样的。

（4）集合主要的作用就是去重，通过结果可以看到，使用交集时，两个集合中相同的元素都被去掉了。

2.9　推 导 式

推导式（comprehensions）又称解析式，是 Python 的一种独有特性。推导式是可以从一个数据序列构建另一个新的数据序列的结构体。Python 共有 3 种推导式：

- 列表（list）推导式;
- 字典（dict）推导式;
- 集合（set）推导式。

2.9.1　初识推导

先来看一段代码：

```
T=[(x,y) for x in range(5) if x%2==0 for y in range(5) if y %2==1]
```

好长好奇怪，其实这就是常见的推导式，其基本语法如下：

```
variable = [expr  for  value  in  seq  if  condition]
```

首先需要一个变量接收推导式，右侧的[ ]表示这是列表推导式，字典或集合推导式都是使用

{ }。expr 是一个表达式或变量，可想象成每个符合条件的值。推导式中的 if 是根据条件过滤哪些值，不是必选。for 循环可理解为某个区间或某个范围。

下面以列表方式举例：

```
T = [i for i in range(40) if i % 4 is 0]
print(T)
```

返回结果：

```
[0, 4, 8, 12, 16, 20, 24, 28, 32, 36]
```

从结果可以看出，返回的是列表形式，通过 if 条件过滤下来的是可以被 4 整除的数。通过 for 循环依次输出 40 以下的满足 if 条件的值。

因为在语法中[ ]里面的第 1 个变量也可以是表达式，所以我们再改为表达式，这里创建一个函数。

```
def seq(x):
    return x*x
T = [seq(i) for i in range(40) if i % 4 is 0]
print(T)
```

以上输出变量的平方值：

```
[0, 16, 64, 144, 256, 400, 576, 784, 1024, 1296]
```

字典推导式、集合推导式与列表推导式的使用语法一致，只是需要将[ ]改为{ }。下面创建字典推导式，输出指定键的内容。

【示例 2-30】

```
01  dic1 = {'a': 20, 'c': 46, 'A': 9, 'B': 30,'d':50}
02  dic1_T = {
03      t.lower(): dic1.get(t)
04      for t in dic1.keys()  if t.lower() in ['a','b']
05  }
06  print(dic1_T)
```

首先创建一个字典 dic1，有 5 个键-值对。再创建一个字典推导式 dic1_T，过滤条件是字典的键 a、b 或 A、B。返回的内容是 t.lower(): dic1.get(t)。lower()返回小写，get()获取指定键的值。上述代码结果是：

```
{'a': 9, 'b': 30}
```

2.9.2 嵌套推导

列表都是可以嵌套的，列表的推导式也可以嵌套。下面用一个广泛流行的例子来说明。

有一个嵌套的名字列表，第 1 排是男孩名字，第 2 排是女孩名字。

```
names = [
    ['Tom','Billy','Jefferson','Andrew','Wesley','Steven','Joe'],
    ['Alice','Jill','Ana','Wendy','Jennifer','Sherry','Eva']
]
```

如果使用 for 循环输出“姓名中带有两个以上字母 e 的”姓名，那么代码如下：

```
tmp=[]
for lst in names:
    for name in lst:
        if name.count('e') >= 2:
            tmp.append(name)
print(tmp)
```

tmp 是一个临时列表，用于保存符合条件的姓名，代码中用了两个 for 遍历嵌套列表，代码结果是：

```
['Jefferson', 'Wesley', 'Steven', 'Jennifer']
```

嵌套推导式会让代码更加简洁，相同的嵌套列表，如果要用推导式的形式，代码如下：

```
T=[name for lst in names for name in lst if name.count('e') >= 2]
print(T)
```

读者可以仔细分析嵌套列表推导式的每个关键词，也是两个 for，只是都写在了一行里，其实关键词都差不多，上述代码的输出结果也与前面相同。

2.10　数据结构实战：文本统计分析

本节将使用本章前面介绍的各种数据结构实现一个简单的例子：对两个英文文档进行统计，得到两个英文文档使用了多少单词、每个单词的使用频率，并且对两个文档进行比较，返回有差异的行号和内容。

本节要实现的程序取名为 PyMerge，它需要实现两个功能模块：统计和比较。

- 统计功能：主要是统计总词汇数和每个词汇的数目。

可以将每个词汇作为 key 保存到字典中，对文本从开始到结束循环处理每个词汇，并将它的 value 设置为 1，如果已经存在该词汇的 key，说明该词汇已经使用过，就将它的 value 累加 1。

- 比较功能：主要是比较两个文本的差异，需要忽略空行和空格的影响，也就是因为多个空行或空格产生的文本差异不应该列为文本差异。

在实现这个功能的时候，首先要将文本分成一行一行的，对每一行进行处理，忽略空格的个数，将字符串里有效字符转换成列表，然后进行比较。

统计功能是将词汇放到字典类型中，用字典的 key 存放单词，用 value 存放个数。

2.10.1　文本统计功能

统计功能是将一个文本的每个字符作为 key 值放入字典中，以下代码实现了文本词汇数的统计。

```
01  >>> readtxt="""
```

```
02       this is a test txt!
03       can you see this ?
04       """
05   >>>
06   >>> readlist=readtxt.split()
07   >>> dict={}
08   >>> for every_word in readlist:
09       if  every_word in dict:
10          dict[every_word]+=1
11       else:
12          dict[every_word]=1
13
14   >>> print(dict)
15   {'this': 2, 'is': 1, 'a': 1, 'test': 1, 'txt!': 1, 'can': 1, 'you': 1, 'see': 1, '?': 1}
```

第 06 行代码，对文本字符串 readtxt 做 split 操作，就可以获得该文本字符串的所有词汇，每个词汇都作为列表的元素返回。第 08~12 行代码循环处理词汇列表，将词汇作为 dict 的 key，如果 key 已经存在，则它的 value 值累加 1，否则将 value 设置为 1。

上面实现的文本统计的代码需要封装起来给整个程序使用，可以使用函数封装，使用语法定义 def functionname(): 函数就可以了，具体的函数在后面的章节会介绍，这里直接使用。封装代码如下：

```
def wordcount(readtxt):
    dict={}
    readlist=readtxt.split()
    for every_word in readlist:
        if every_word in dict:
            dict[every_word]+=1
        else:
            dict[every_word]=1
    return dict
```

2.10.2 文本比较功能

文本比较首先需要将文本字符串分成一行一行的，使用字符串 splitlines 方法将一个字符串按行分成一个列表，对分成的列表删除空元素和空白字符元素，最后将两个文本进行循环比较。以下代码实现了文本比较功能：

```
01   def testcmp(test1,test2):
02       return_li=[]
03       word_list1=test1.splitlines()
04       word_list2=test2.splitlines()
05       li_word=[column for column in word_list1 if column and column.isspace()]
06       li_word2=[column for column in word_list2 if column and column.isspace()]
07       li_len=len(li_word)
08       li_len2=len(li_word2)
09       for step in range(max(li_len,li_len2)):
10           if step<li_len and  step<li_len2:
11               li_col1=li_word[step].split()
12               li_col2=li_word2[step].split()
13               if li_col1!=li_col2:
```

```
14                return_li.append((word_list1.index(li_word[step]),
15                    word_list2.index(li_word2[step]),li_word[step],li_word2[step]))
16          else:
17              if li_len>li_len2:
18                  return_li.append((word_list1.index(li_word[step]),-1,li_word[step],''))
19              else:
20                  return_li.append((-1,word_list2.index(li_word2[step]),'',
                                      li_word2[step]))
21      return return_li
```

代码的实现逻辑主要包括：

- 第 03~04 行，对两个文本按行划分。
- 第 05~06 行，一个列表推导式操作，作用是去掉空行和只有空白字符的行。
- 第 11~12 行，将文本每一行转换成列表，转换过程中忽略空白字符。
- 第 13~14 行，比较的结果如果不相等，就写信息到 return_li，以供函数返回信息。
- 第 16~20 行，两个文本行数不一致时，将多出来的行的文本信息写到 return_li，以供函数返回。

本节的例子并不完善，还没有实现文件读取功能等更复杂的技术。这里只是演示一下前面讲解的一些数据结构的使用，等读者全部学完本书内容后，可以继续完善这个例子。