Developer's Library

PEARSON

针对 Python 3.0/3.1 而写
内容详实全面，实例丰富，便于学习

Python 3 程序开发指南（第 2 版 · 修订版）

Programming in Python 3

Second Edition

A Complete Introduction to the Python Language

[美] Mark Summerfield 著
王弘博 孙传庆 译

人民邮电出版社
POSTS & TELECOM PRESS

Python 3
程序开发指南
（第2版·修订版）

Programming in
Python 3
Second Edition

[美] Mark Summerfield 著
王弘博 孙传庆 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Python 3程序开发指南 / (美) 萨默菲尔德(Summerfield,M.) 著 ; 王弘博, 孙传庆译. -- 2版（修订本）. -- 北京 : 人民邮电出版社, 2015.2
ISBN 978-7-115-38338-9

Ⅰ. ①P… Ⅱ. ①萨… ②王… ③孙… Ⅲ. ①软件工具－程序设计－指南 Ⅳ. ①TP311.56-62

中国版本图书馆CIP数据核字(2015)第007863号

◆ 著 [美] Mark Summerfield
译 王弘博 孙传庆
责任编辑 傅道坤
责任印制 张佳莹 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：33.25
字数：641 千字 2015 年 2 月第 2 版
印数：3 501-6 500 册 2015 年 2 月北京第 1 次印刷

著作权合同登记号 图号：01-2010-4743 号

定价：69.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315

内容提要

Python 是一种脚本语言，在各个领域得到了日益广泛的应用。本书全面深入地对 Python 语言进行了讲解。

本书首先讲述了构成 Python 语言的 8 个关键要素，之后分章节对其进行了详尽的阐述，包括数据类型、控制结构与函数、模块、文件处理、调试、进程与线程、网络、数据库、正则表达式、GUI 程序设计等各个方面，并介绍了其他一些相关主题。全书内容以实例讲解为主线，每章后面附有练习题，便于读者更好地理解和掌握所讲述的内容。

本书适合于作为 Python 语言教科书使用，对 Python 程序设计人员也有一定的参考价值。

前　言

在应用广泛的各种语言中，Python 或许是最容易学习和最好使用的。Python 代码很容易阅读和编写，并且非常清晰，而没有什么隐秘的。Python 是一种表达能力非常强的语言，这意味着，在设计同样的应用程序时，使用 Python 进行编码所需要的代码量要远少于使用其他语言（比如 C++或 Java）的代码量。

Python 是一种跨平台的语言：一般来说，同样的 Python 程序可以同时在 Windows 平台与 UNIX 类平台（比如 Linux、BSD 与 Mac OS X）上运行——只需要将构成 Python 程序的单个或多个文件复制到目标机器上，而不需要“构建”或编译（Python 是解释型语言）。当然，Python 程序使用特定平台功能也是可能的，但通常很少需要这样做，因为几乎所有 Python 标准库与大多数第三方库都是完全跨平台的，或至少对用户是透明的。

Python 的强大功能之一是带有一个非常完全的标准库，通过该标准库，我们可以方便地实现大量功能，比如，从 Internet 下载一个文件、对压缩的存档文件进行解压，或创建一个 Web 服务器，而这些貌似复杂的功能，只需要少数几行 Python 代码就可以实现。除标准库外，还有数以千计的第三方库，其中一些提供了比标准库更强大、更复杂的功能，比如，Twisted 网络库与 NumPy 数值型库。其他一些库提供了极专业化的功能，因而没有包含在标准库中，比如，SimPy 模拟包。大多数第三方库都可以通过 Python Package Index，网址为 http://pypi.python.org/pypi 进行访问。

虽然本质上是一种面向对象语言，但是实际上 Python 可以用于进行过程型程序设计、面向对象设计，以及某种程度上的函数型程序设计。本书主要展示如何使用 Python 进行过程型程序设计与面向对象程序设计，也介绍了 Python 的函数型程序设计功能。

本书的目标是展示如何使用良好的 Python 3 惯用风格编写 Python 程序，在阅读本书之后，你就可以发现，本书是一本非常有用的 Python 3 语言索引。虽然与 Python 2 相比，Python 3 所做的改进和改变是渐进的，而非革新，但是在 Python 3 中，Python 2 中的一些既有做法变得不再合适或不再必要，因此必须介绍和使用 Python 3 中的一些新做法，以便充分利用 Python 3 的功能。毋庸置疑，Python 3 优于 Python 2；它构建于 Python 2 多年的实践基础上，并添加了大量的新功能（还摒弃了 Python 2 的一些不良特性）。与 Python 2 相比，使用 Python 3 更富于乐趣，更便利、容易和具有一致性。

本书旨在讲解 Python 语言本身，虽然中间也涉及很多标准 Python 库，但是没有

全部介绍。不过这不是问题，因为在阅读本书之后，将具备充分的 Python 知识，读者可以自如地使用任意的标准库或任意第三方库，并可以创建自己的库模块。

本书适用于多种不同类型的读者，包括自学者、程序设计爱好者、学生、科学家、工程师，以及工作中需要进行程序设计的人，当然，也包括计算专业工作者和计算机科学家。要面对这些不同类型的读者，既让已具备丰富知识的读者不厌烦，又让经验不足的读者可以理解，因此，本书假定读者至少具备一定的程序设计经验（任何程序语言）。特别是，本书需要读者了解数据类型（比如数与字符串）、集合数据类型（比如集合与列表）、控制结构（比如 if 与 while 语句）以及函数。此外，有些实例与练习需要读者具备 HTML markup 的相关知识，后面某些更专业化的章节需要读者具备一定领域的知识，比如，数据库那一章需要读者具备基本的 SQL 知识。

在结构上，本书尽可能让读者阅读时最富有效率。在第 1 章结束时，读者应该就可以编写短小但有用的 Python 程序。后续的每一章都分别讲述一个新主题，在内容上通常都会比前一章更广、更深。这意味着，如果顺序阅读本书各章，在每一章结束后，都可以停止阅读，并利用该章讲解的知识编写完整的 Python 程序，当然，你也可以继续阅读以便学习更高级、更复杂的技术。出于这一考虑，有些主题在某一章中介绍，在后续的一章或几章中又进行了深入讲解。

讲解一门新的程序设计语言时，有两个关键的问题。第一个问题是：有时候，需要讲解某个特定概念时，会发现该概念依赖于另外一个概念，而这个概念反过来又直接或间接地依赖于这个“特定概念”。第二个问题是：在最开始的时候，由于读者对该语言毫无所知，或者只具备极为有限的知识，因此要给出有趣的、有用的实例或练习非常困难。在本书中，我们力图解决这两个问题。对第一个问题，首先要求读者具备一定的程序设计经验，了解基本的概念；对第二个问题，我们在第 1 章中就讲解了 Python 的“beautiful heart”——Python 的 8 个关键要素，足以用于编写良好的程序。这种做法也有一个不足的地方：在前几章中，有些实例在风格上会有一点刻意为之的痕迹，这是因为这些实例中只是使用了到该章为止所讲解的知识，不过这种副作用越到后面的章节越弱，到第 7 章结束时，所有实例都使用完全自然的 Python 3 惯用风格编写。

本书所讲述的方法是完全实践型的，我们建议读者尝试书中讲述的每个实例，做好每一个练习，以便获取实际的动手经验。在可能的地方，本书都提供了虽然短小但是完整的程序，这些程序实例展现了真实的应用场景。本书所带实例、练习及其解决方案都可以在 www.qtrac.eu/py3book.html 处获取，并且都已经在 Windows、Linux、Mac OS X 等操作平台上的 Python 3 环境下进行了测试。

本书的组织结构

第 1 章，提出了 Python 的 8 个关键要素，这些要素足以用于编写完整的 Python

程序。本章描述了一些可用的 Python 程序设计环境，给出了两个小实例，这两个实例都是使用前面讲述的 8 个关键要素构建的。

第 2 章～第 5 章介绍了 Python 的过程型程序设计功能，包括基本数据类型与集合数据类型、很多有用的内置函数与控制结构，以及比较简单的文本文件处理功能。第 5 章展示了如何创建自定义模块与包，并提供了 Python 标准库概览，以便读者对 Python 提供的功能有充分的了解，避免重复工作。

第 6 章对使用 Python 进行面向对象程序设计进行了全面深入的讲解。由于面向对象程序设计是建立在过程型程序设计基础之上的，因此，此前几章讲述的过程型程序设计相关的知识仍然可以用于面向对象程序设计，比如，利用同样的数据类型、集合数据类型以及控制结构。

第 7 章主要讲述文件的读、写。对于二进制文件，包括压缩、随机存取；对于文本文件，包括人工分析以及正则表达式的使用。本章也包括了如何读、写 XML 文件，包括使用元素树、DOM（文档对象模型）以及 SAX（用于 XML 的简单 API）。

第 8 章回顾了前面一些章节中讲述的内容，探讨了数据类型、集合数据类型、控制结构、函数、面向对象程序设计等领域一些更高级的内容。本章还介绍了很多新功能、类以及高级技术，包括函数型程序设计——其中的内容有挑战性，但也很有用。

第 9 章与其他章节的不同之处在于，它不是介绍新的 Python 特性，而是讨论了用于调试、测试和 profiling 程序的技术和库。

余下的几章讲述了其他一些高级主题。第 10 章展示了如何将程序的工作负载分布在多个进程与线程上；第 11 章展示了如何使用 Python 的标准网络支持功能编写客户端/服务器应用程序；第 12 章讲解了数据库程序设计（包括键-值对 DBM 文件与 SQL 数据库）；第 13 章讲述了 Python 的正则表达式 mini-language，介绍了正则表达式模块；第 14 章讲解使用正则表达式，以及使用两种第三方模块（PyParsing 和 PLY）的解析技术；第 15 章介绍了 GUI（图形用户界面）程序设计。

本书的大部分章都较长，这样是为了将所有相关资料放在一起，以便于查询引用，不过，各章都进一步划分为节、小节，因此，本书仍然是可以按照适合自己的节奏阅读的，比如，每次阅读一节或一个小节。

获取并安装 Python 3

如果使用的是较新版本的 Mac 或 UNIX 类系统并及时更新，就应该已经安装了 Python 3。要检查是否已经安装，可以在控制台（在 Mac OS X 上是 Terminal.app）中输入命令 python V（注意是大写的 V），如果版本为 3.X，就说明系统中已经安装了 Python 3，而不需要自己再安装，如果不是，请继续阅读。

对 Windows 与 Mac OS X 系统，存在易于使用的图形界面安装包，只需要按照提示就可以一步一步地完成安装过程。安装工具包可以从 www.python.org/download 处获取，该网站为 Windows 系统提供了 3 个独立的安装程序，一般需要下载的是普通的“Windows ×86 MSI Installer”，除非确认自己的机器使用的是 AMD64 或 Itanium 处理器，这种情况需要下载处理器特定的安装程序。下载安装程序后，只需要运行并按提示进行操作，就可以安装好 Python 3。

对 Linux、BSD 以及其他 UNIX 类系统，安装 Python 的最简单方法是使用该操作系统的软件包管理系统。大多数情况下，Python 安装程序是以几个单独的软件包形式提供的。比如，在 Fedora 中，用于 Python 的安装包为 python，用于 IDLE（一个简单的开发环境）的安装包为 python-tools。需要注意的是，只有在 Fedora 为更新的版本时（版本 10 或后续版本），这些安装包才是基于 Python 3 的。同样，对基于 Debian 的系统，比如 Ubuntu，对应的安装包为 python3 与 idle3。

如果没有适合自己操作系统的安装包，就需要从 www.python.org/download 处下载源程序，并从头编译 Python。你可以下载 source tarballs 中的任意一个，并根据其文件格式选择不同的工具进行解压：如果下载的是 gzipped tarball，则需要使用 tar xvfz Python-3.0.tgz；如果下载的是 bzip2 tarball，则需要使用 tar xvfj Python-3.0.tar.bz2。配置与构建过程是标准的，首先切换到新创建的 Python-3.0 目录，运行./configure（如果需要本地安装，可以使用--prefix 选项），之后运行 make。

安装 Python 3 时，可能出现的一种情况是，在安装结束时弹出提示消息，声称不是所有的模块都已经安装，这通常意味着机器上缺少某些必要的库或头文件。这种情况可以通过单独安装相应程序包处理，比如，如果 readline 模块无法构建，可以使用包管理系统安装相应的开发库，如在基于 Fedora 的系统上安装 readline-devel，在基于 Debian 的系统上安装 readline-dev（遗憾的是，相关包的名字并不总是那么显而易见的）。安装了缺少的包之后，再次运行./configure 与 make。

成功构建之后，可以运行 make test，以便确认是否一切正常——尽管这并非必需，并且可能需要花费一些时间。

如果使用了--prefix 进行本地安装，那么只需要运行 make install。你可能需要为 python 可执行程序添加软链接（如果使用的是--prefix=$HOME/local/python3，并且 PATH 中包含$HOME/bin 目录，则需要 ln -s ~/local/python3/bin/python3.0 ~/bin/python3），为 IDLE 添加软链接也会带来不少方便（假定前提与上面的一样，则需要 ln -s ~/local/python3/bin/idle ~/bin/idle3）。

如果不使用--prefix 并具备 root 权限，应该以 root 用户登录，并执行 make install。在基于 sudo 的系统（比如 Ubuntu）上，则执行 sudo make install。如果系统上已经存在 Python 2，/usr/bin/python 并不会改变，同时 Python 3 将以 python3 的形式存在，同样地，Python 3 的 IDLE 以 idle3 的形式存在。

致谢

首先感谢读者对本书第一版的反馈，他们在反馈中给出了修改意见和建议。

其次要感谢的是本书的技术评审 Jasmin Blanchette，他是一位计算机科学家、程序员，我们曾共同编写过两本 C++/Qt 书籍。Jasmin 对章节布局的规划、对所有实例的建议与批评以及对本书的详细审阅，这一切都极大地提高了本书的质量。

Georg Brandl 是一位一流的 Python 开发人员，也是一位负责创建 Python 的新文档工具链的文档编辑。Georg 挑出了很多微妙的错误，并非常耐心、非常坚持地对其进行解释，直至可以被准确理解和纠正。他还对很多实例进行了改进。

Phil Thompson 是一位 Python 专家，也是 PyQt（可能是可用的 Python GUI 库中最好的）的创建者。Phil 的敏锐洞察力，有时候甚至是带有挑战性的反馈，都促使我对本书的很多内容进行了澄清和纠正。

Trenton Schulz 是 Nokia 的 Qt Software（以前的 Trolltech）部门的一位高级软件工程师，也是我以前撰写的所有书籍的有见地的评审，在本书的评审编辑中又一次给予了我宝贵的帮助。Trenton 对本书的细致阅读与提出的大量宝贵建议，帮助我澄清了很多问题，在很大程度上提高了本书质量。

除上面提及的各位评审人员之外（他们都读完了整本书），David Boddie，Nokia 的 Qt Software 的一位高级技术作者，也是一位经验丰富的 Python 老手和开源软件开发者，阅读了本书的部分章节并给出了有价值的回馈。

同时也要感谢 Guido van Rossum，Python 的创建者，感谢大量的 Python 社区，是他们的努力，使得 Python（尤其是库文件）变得如此有用而好用。

还要感谢 Jeff Kingston，Lout typesetting 语言（我使用这种语言的时间超过 10 年）的创建者。

特别感谢本书的编辑 Debra Williams Cauley，感谢她给予的支持，并再一次使得本书的整个编辑、出版过程尽可能顺畅；感谢 Anna Popick，他将本书的生产过程管理得非常好；感谢校对人员 Audrey Doyle 再一次做了良好的工作。

最后也是最重要的是，感谢我的妻子 Andrea，感谢她对我在凌晨 4 点起床，记录下编写本书的灵感，以及对代码进行纠正和测试时，所表现出来的忍耐，以及她的爱、忠诚和一如既往的支持。

目　录

第 1 章 过程型程序设计快速入门

本章提供了足以开始编写 Python 程序的信息。如果此时尚未安装 Python，强烈建议读者先行安装 Python，以便随时进行编程实践，获取实际经验，巩固所学的内容。

本章第 1 节展示了如何创建并执行 Python 程序。你可以使用自己最喜欢的普通文本编辑器来编写 Python 代码，但本节中讨论的 IDLE 程序设计环境提供的不仅是一个代码编辑器，还提供了很多附加的功能，包括一些有助于测试 Python 代码、调试 Python 程序的工具。

第 2 节介绍了 Python 的 8 个关键要素，通过这 8 个要素本身，就足以编写有用的程序。这 8 个要素在本书的后续章节中将全面涉及与讲解，随着本书内容的推进，这些要素将被 Python 的其他组成部分逐渐补充、完善。到本书结束时，读者将对 Python 语言有完整的了解，并充分利用该语言提供的所有功能编写自己的 Python 程序。

本章最后一节介绍了两个短小的程序，这两个小程序利用了第 2 节中介绍的 Python 特性的一部分，以便读者可以及时尝试 Python 程序设计。

1.1 创建并运行 Python 程序

要编写 Python 代码，可以使用任意能加载与保存文本（使用 ASCII 或 UTF-8 Unicode 字符编码）的普通文本编辑器。默认情况下，Python 文件使用 UTF-8 字符编码，UTF-8 是 ASCII 的超集，可以完全表达每种语言中的所有字符。通常，Python 文件的扩展名为.py，不过在一些 UNIX 类系统上（比如 Linux 与 Mac OS X），有些 Python 应用程序没有扩展名，Python GUI（图形用户界面）程序的扩展名则为.pyw（特别是在 Windows 与 Mac OS X 上）。在本书中，我们总是使用.py 作为 Python 控制台程序与 Python 模块的扩展名，使用.pyw 作为 GUI 程序的扩展名。本书中提供的所有实例可以不需修改地在安装 Python 3 的所有平台上运行。

为确认系统已经正确安装 Python，也为了展示经典的第 1 个程序，在普通文本编

辑器（Windows 记事本即可，后面我们会使用更好的编辑器）中创建一个名为 hello.py 的程序，其中包含如下一些内容：

```
#!/usr/bin/env python3
print("Hello", "World!")
```

第 1 行为注释。在 Python 中，注释以#开始，作用范围为该行（后面我们将解释更隐秘的一些注释信息）第 2 行为空行，Python 会忽视空行，但空行通常有助于将大块代码分割，以便于阅读。第 3 行为 Python 代码，其中调用了 print()函数，该函数带 2 个参数，每个参数的类型都是 str（字符串，即一个字符序列）。

.py 文件中的每个语句都是顺序执行的，从第 1 条语句开始，逐行执行。这与其他一些语言是不同的，比如，C++与 Java 一般是从某个特定函数或方法（带有函数或方法名）开始执行。当然，下一节讨论 Python 控制结构时我们将看到，Python 程序的控制流也是可以改变的。

这里，我们假定 Windows 用户将其 Python 代码保存在 C:\py3eg 目录下，UNIX（包括 UNIX、Linux 与 Mac OS X）用户将其 Python 代码保存在$HOME/py3eg 目录下。输入上面的代码后，将其保存在 py3eg 目录，退出文本编辑器。

保存了程序之后，就可以运行该程序了。Python 程序是由 Python 解释器执行的，通常在控制台窗口内进行。在 Windows 系统上，控制台窗口称为“控制台”、“DOS 提示符”或“MS-DOS 提示符”，或其他类似的称谓，通常可以通过“开始”、“所有程序”、“附件”这一顺序打开。在 Mac OS X 上，控制台是由 Terminal.app 程序（默认情况下在应用程序/工具这一目录下）提供的，通过 Finder 可以进行访问。在其他 UNIX 系统上，可以使用 xterm 或窗口环境提供的控制台，比如 konsole 或 gnome- terminal。

启动一个控制台，在 Windows 系统上，输入如下命令（前提是假定 Python 安装在默认位置）——控制台的输出以粗体展示，输入的命令以细体展示。

```
C:\>cd c:\py3eg
C:\py3eg\>C:\Python30\python.exe hello.py
```

由于 cd（切换目录）命令是采用绝对路径的，因此从哪个目录启动并不会影响程序执行。

UNIX 用户需要输入如下命令（假定 Python 3 在 PATH 下）：*

```
$ cd $HOME/py3eg
$ python3 hello.py
```

上面两种情况下，输出应该是相同的：

```
Hello World!
```

需要注意的是，除非特别声明，Python 在 Mac OS X 上的行为与在其他 UNIX 系

* UNIX 提示符可能并不是$的形式，这并无影响。