TURING 图灵程序设计丛书

Addison Wesley

Python Essential Reference Fourth Edition

Python参考手册

（第4版）

[美] David M. Beazley 著
谢俊 杨越 高伟 译

- 经典著作全面升级
- Python程序员案头必备
- 涵盖Python 2和Python 3共有特性

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Python Essential Reference Fourth Edition

Python参考手册

（第4版）

[美] David M. Beazley 著
谢俊 杨越 高伟 译

图书在版编目（CIP）数据

Python参考手册 : 第4版 / (美) 比兹利(Beazley,D.M.) 著 ; 谢俊, 杨越, 高伟译. -- 北京 : 人民邮电出版社, 2011.1
(图灵程序设计丛书)
书名原文: Python Essential Reference,Fourth Edition
ISBN 978-7-115-24259-4

Ⅰ. ①P… Ⅱ. ①比… ②谢… ③杨… ④高… Ⅲ. ①软件工具－程序设计－技术手册 Ⅳ. ①TP311.56-62

中国版本图书馆CIP数据核字(2010)第230245号

内 容 提 要

本书是Python编程语言的权威参考指南，书中详尽解释了Python核心语言和Python库中最重要的部分，涉及类型和对象、操作符和表达式、编程结构和控制流、输入和输出、测试、调试等，也包括一些Python官方文档或其他参考资料中未提及的高级主题。

本书面向Python程序员，或有其他编程语言经验的开发人员。

图灵程序设计丛书

Python参考手册（第4版）

◆ 著 [美] David M. Beazley
译 谢 俊 杨 越 高 伟
责任编辑 明永玲
执行编辑 丁晓昀

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：33.75
字数：927千字 2011年1月第1版
印数：1－3 000册 2011年1月北京第1次印刷

著作权合同登记号 图字：01-2010-4024号

ISBN 978-7-115-24259-4

定价：89.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

本书是Python编程语言的一份简明参考。尽管有经验的程序员也可以通过本书学习Python语言，但本书并非讲述如何进行编程的一份全面指南或专著。相反，本书的目标是准确而简练地介绍核心的Python语言以及Python库中最重要的部分。本书假定读者以前拥有Python或其他语言（如C或Java方面）的编程经验。另外，对系统编程（例如基本的操作系统概念和网络编程）有一定的了解可能会对理解库参考有所帮助。

在http://www.python.org上可以免费下载Python。几乎针对所有操作系统都有可用的版本，包括UNIX、Windows和Macintosh。另外，Python网站包含文档、指南和各种第三方软件的链接。

这一版是在Python发展的关键时刻面世的。Python 2.6和Python 3.0几乎是同时发布的，但Python 3与以前的Python版本不兼容。作为一位作者和程序员，我面临着一个两难问题：是直接跳到Python 3.0，还是使用大多数程序员更为熟悉的Python 2.x版本？

多年前，身为C程序员的我经常把某些书当作选择使用语言特性的权威。例如，如果使用K&R书中未曾提到过的某些特性，很可能导致无法移植，因此你就得格外谨慎才行。作为程序员，我运用这种方法十分得心应手，因此我也决定在本书的这个版本中沿袭这一做法。也就是说，这一版删除了Python 3中已经不再支持的Python 2特性。而且，我也没有重点讲述尚不能向后移植的Python 3特性（但附录中仍然会介绍这些特性）。最终，我希望本书能够成为Python程序员案头必备的参考书，无论你们使用的是哪个版本的Python。

本书的这一版还包含一些自第1版出版至今10余年来最激动人心的一些变化。在过去的数年间，大部分Python开发的重点都放在了新的编程语言特性上——特别是与功能和元编程相关的特性。因此，讲述函数和面向对象编程的章已经得到了极大的扩充，包括的主题有生成器、迭代器、协程、装饰器和元类。讲述库的章已经把重点转移到更加现代的模块上。整本书中的示例和代码片段都进行了更新。我认为大多数程序员将对扩充之后的内容感到满意。

最后应该注意，Python网站上已经包含了数千页有用的文档。本书的内容在很大程度上基于该文档，但又存在很多关键的区别。首先，这份参考讲述信息的方式更加紧凑，提供不同的例子，并对很多主题提供了额外的描述。其次，库参考中的大量主题都进行了扩展，包含很多外部的参考资料。对于底层系统和网络模块尤其如此，因为对模块的有效使用往往依赖于手册和外部参考中列出的种种选项。另外，为了使参考更加简明，这一版还删掉了很多已经废弃和相对较为晦涩的库模块。

我的目标是写出一本真正包含使用Python及其众多模块所需的一切内容的参考指南。本书绝不是一本全方位介绍Python语言的百科全书，但我希望本书能够成为你的实用参考。十分欢迎读者对本书提出意见和建议。

David Beazley
2009年6月
于伊利诺伊州芝加哥

致　　谢

本书能与读者见面，要感谢很多人的大力支持。首先要感谢Noah Gift参与这个项目，并提出了许多建设性意见。Kurt Grandis也对很多章节发表了中肯的见解。我还要感谢前几版的技术审稿人Timothy Boronczyk、Paul DuBois、Mats Wichmann、David Ascher和Tim Bell，他们的精彩意见和建议促成了过去几版的成功。Guido van Rossum、Jeremy Hylton、Fred Drake、Roger Masse和Barry Warsaw也对第1版提供了极大的帮助，同时还在1999年的炎热夏天招待了我好几个星期。还有很重要的是，没有读者们的热情反馈，就不可能有本书的面世。要感谢的人实在太多，这里无法一一列出他们的名字，但我已经尽力采纳你们的建议来让本书变得更好。我还要感谢Addison-Wesley和Pearson Education的工作人员们，他们对这个项目给予了一贯的支持与帮助。Mark Taber、Michael Thurston、Seth Kerney和Lisa Thibault都对本书的顺利出版倾注了很多心血。还要特别感谢Robin Drake，他在第3版出版的过程中做了大量的编辑工作。最后，我要感谢我伟大的妻子和好搭档Paula Kamen给我的鼓励、欢乐和爱。

目　录

第一部分　Python 语言

Part 1 第一部分

Python 语言

本部分内容

第1章 Python简介

本章快速介绍Python这门语言，目标是在阐明Python的大部分基本特性的同时，又不会太过纠缠于特殊的规则或细节。为此，本章简要讲述一些基本概念，如变量、表达式、控制流、函数、生成器、类和输入/输出。本章不追求大而全，但有经验的程序员应该能够从本章中的资料推而广之，创建出更加高级的程序。初学者应该多尝试一些实例，才能对这门语言有一定的了解。如果你对Python不熟悉也没有使用过Python 3，可以使用Python 2.6来学习本章内容。实际上，本章介绍的主要概念同时适用于这两个版本，但在Python 3中存在极少数可能与本章中给出的例子不符的关键语法变化，其中大多数与打印和I/O有关。请参考附录A，以了解详细信息。

1.1 运行 Python

Python程序是由解释器来执行的。通常，只要在命令shell中输入python即可启动解释器。然而，解释器和Python开发环境存在多种实现（例如Jython、IronPython、IDLE、ActivePython、Wing IDE、pydev等），因此需要参考相应文档中的启动说明。解释器启动后将出现一个命令提示，在此可以开始输入程序，进入简单的读入-求值循环。例如，在下面的输出中，解释器显示了版权消息和>>>提示符，用户可以在这里输入熟悉的打印“Hello World”命令：

```
Python 2.6rc2 (r26rc2:66504, Sep 19 2008, 08:50:24)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World"
Hello World
>>>
```

注意 如果在尝试前述例子时出现了语法错误，你使用的很可能就是Python 3。遇到这种情况并不妨碍你继续学习本章的内容，但要知道打印语句在Python 3中已经变为一个函数。在下面的例子中，只要在要打印的内容两边加上括号即可正常运行，例如：

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

如果要打印的内容只有一项，在要打印内容两边放置括号的方法在Python 2中同样有效。然而，这种语法在现有的Python代码中并不常见。在后面的章节中，这种语法有时会用在与打印无直接关系的展示特性的例子中，但这些例子应该同时适用于Python 2和3。

Python的交互模式是它最有用的功能之一。在交互式shell中，可以输入任意合法的语句或语句序列，然后立即查看结果。很多人甚至使用交互式Python作为桌面计算器，作者本人也如此。例如：

```
>>> 6000 + 4523.50 + 134.12
10657.620000000001
>>> _ + 8192.32
18849.940000000002
>>>
```

以交互方式使用Python时，特殊变量 _ 用于保存最后一次运算的结果。如果要在后续语句中保存或使用最后一次运算的结果，使用此变量十分方便。但要强调一点，此变量只有在以交互方式工作时才有定义。

如果要创建可以重复运行的程序，可将语句放到一个文件中：

```
# helloworld.py
print "Hello World"
```

Python源文件是普通的文本文件，后缀通常是.py。#字符表示整个一行都是注释。

要执行helloworld.py文件，可通过如下方式将文件名提供给解释器：

```
% python helloworld.py
Hello World
%
```

在Windows中，双击一个.py文件或者在Windows开始菜单的“运行”命令中输入程序名称，均可启动Python程序。这会启动解释器，并在控制台窗口中运行程序。但要知道，当程序执行完成后，控制台窗口将立即消失（通常来不及看清楚输出）。如要进行调试，最好是在像IDLE这样的Python开发工具中运行程序。

在UNIX中，可以在程序的首行中使用#!，如下所示：

```
#!/usr/bin/env python
print "Hello World"
```

解释器不断运行语句，直到到达输入文件的结尾。如果是以交互方式运行，有两种方法可以退出解释器，一种是输入EOF（End of File，文件结束），另一种是从Python IDE的下拉菜单中选择Exit。在UNIX中，EOF是Ctrl+D，而在Windows中则是Ctrl+Z。程序可以通过抛出SystemExit异常来请求退出。

```
>>> raise SystemExit
```

1.2 变量和算术表达式

程序清单1-1中的程序通过执行一次简单的复利计算，说明变量和表达式的用法。

程序清单1-1 简单的复利计算

```
principal = 1000            # 初始金额
rate = 0.05                 # 利率
numyears = 5                # 年数
year = 1
while year <= numyears:
    principal = principal * (1 + rate)
    print year, principal     # 注意：在Python 3中是print(year, principal)
    year += 1
```

此程序的输出如下所示：

```
1 1050.0
2 1102.5
3 1157.625
4 1215.50625
5 1276.2815625
```

Python是一种动态类型的语言，在程序执行过程中，可将变量名称绑定到不同的值，而且这些值可以属于不同的类型。赋值运算符的作用仅仅是在名称和值之间创建一种关联。尽管每个值都有一个相关类型，如integer或string，但变量名称是无类型的，在执行过程中可以引用任意类型的数据。这与C语言不同，例如在C语言中，名称代表了用于保存值的固定类型、大小和内存位置。Python的动态行为可以从程序清单1-1的principal变量看出来。最初给它分配的是一个integer值，但程序稍后给它重新赋了值，如下所示：

```
principal = principal * (1 + rate)
```

这条语句对表达式求值，并把名称principal重新与结果关联。principal的原始值是整数类型的1000，但现在的新值是浮点数（rate被定义为浮点数，因此上述表达式的值也是浮点数）。因此在程序中，principal的显式类型从integer动态变为了float。然而准确地说，不是principal的类型变了，而是principal名称引用的值的类型变了。

换行代表一条语句的结束。然而，也可以在同一行上使用分号来隔开多条语句，如下所示：

```
principal = 1000; rate = 0.05; numyears = 5;
```

while语句对随后的条件表达式进行测试。如果被测试的语句为True，while语句的主体就会执行。然后再次测试条件，再执行主体，直到条件变为False。因为循环主体是由缩进表示的，每次循环时都会执行程序清单1-1中while之后的3条语句。Python不会指定所需缩进的量，只要在一个代码块中保持一致即可。然而，每个缩进层次使用4个空格是最常见的情况，而且通常也建议这么做。

程序清单1-1中的程序有一个问题，即输出不是很美观。为了改进这一点，可以让各列右对齐，并将principal的精度限制为两位。实现这种格式有几种方法。最常用的方法是使用字符串格式化运算符%，如下所示：

```
print "%3d %0.2f" % (year, principal)
print("%3d %0.2f" % (year, principal))    # Python 3
```

现在程序的输出如下：

```
1 1050.00
2 1102.50
3 1157.63
4 1215.51
5 1276.28
```

格式化字符串包含普通文本和特殊的格式化字符序列，如"%d"、"%s"和"%f"。这些序列分别用于指定特定类型数据的格式，如整数、字符串或浮点数。特殊的字符序列还可以包含用于指定宽度和精度的修饰符。例如，"%3d"将一个整数格式化为在一个宽度为3的列中右对齐，而"%0.2f"将一个浮点数格式化为在小数点后只出现两位数字。格式化字符串的行为与C语言中的printf()函数几乎完全相同，第4章将对此进行详细说明。

更新的字符串格式化的方法是使用format()函数单独格式化每个部分。例如：

```
print format(year,"3d"),format(principal,"0.2f")
print(format(year,"3d"),format(principal,"0.2f")) # Python 3
```

format()函数使用的格式说明符类似于传统字符串格式化运算符（%）使用的格式说明符。例如，"3d"将一个整数格式化为在一个宽度为3的列中右对齐，而"%0.2f"将一个浮点数格式化为两位精度。字符串还有一个format()方法，可用于一次性格式化很多值。例如：

```
print "{0:3d} {1:0.2f}".format(year,principal)
print("{0:3d} {1:0.2f}".format(year,principal)) # Python 3
```

在这个例子中，"{0:3d}"和"{1:0.2f}"中冒号前的数字代表传递给format()方法的相关参数，而冒号后的部分则是格式说明符。

1.3 条件语句

if与else语句可执行简单的测试，如下所示：

```
if a < b:
    print "Computer says Yes"
else:
    print "Computer says No"
```

if和else子句的主体是用缩进表示的。else子句是可选的。

要创建一条空子句，可以使用pass语句，如下所示：

```
if a < b:
    pass       # Do nothing
else:
    print "Computer says No"
```

使用or、and和not关键字可以建立布尔类型的表达式：

```
if product == "game" and type == "pirate memory" \
                    and not (age < 4 or age > 8):
    print "I'll take it!"
```

注意 编写复杂的测试用例通常需要编写很长的代码行，看起来令人生厌。为了提高代码的可读性，可以像上面一样在一行的结尾使用反斜杠（\），然后就可以在下一行继续书写上一条语句的内容。如果这样做，正常的缩进规则不适用于下一行，因此可以随意设置后续行的格式。

Python没有专门的switch或case语句用于测试多个值。要处理多个测试用例，可以使用elif语句，如下所示：

```
if suffix == ".htm":
    content = "text/html"
elif suffix == ".jpg":
    content = "image/jpeg"
elif suffix == ".png":
    content = "image/png"
else:
    raise RuntimeError("Unknown content type")
```

要表示真值，可以使用布尔值True和False，例如：

```
if 'spam' in s:
    has_spam = True
else:
    has_spam = False
```