



图灵程序设计丛书

Python 语言及其应用

Introducing Python

[美] Bill Lubanovic 著
丁嘉瑞 梁杰 禹常隆 译



Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo
O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北京

图书在版编目（C I P）数据

Python语言及其应用 / (美) 卢布诺维克
(Lubanovic, B.) 著 ; 丁嘉瑞, 梁杰, 禹常隆译. — 北
京 : 人民邮电出版社, 2016. 1
(图灵程序设计丛书)
ISBN 978-7-115-40709-2

I. ①P… II. ①卢… ②丁… ③梁… ④禹… III. ①
软件工具—程序设计 IV. ①TP311. 56

中国版本图书馆CIP数据核字(2015)第245113号

内 容 提 要

本书介绍 Python 语言的基础知识及其在各个领域的具体应用，基于最新版本 3.x。书中首先介绍了 Python 语言的一些必备基本知识，然后介绍了在商业、科研以及艺术领域使用 Python 开发各种应用的实例。文字简洁明了，案例丰富实用，是一本难得的 Python 入门手册。

本书适合所有编程初学者阅读。

-
- ◆ 著 [美] Bill Lubanovic
 - 译 丁嘉瑞 梁 杰 禹常隆
 - 责任编辑 岳新欣
 - 执行编辑 李艳玲 冯雪松
 - 责任印制 杨林杰
 - ◆ 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 印刷
 - ◆ 开本：800×1000 1/16
 - 印张：25.25
 - 字数：630千字 2016年1月第1版
 - 印数：1-5 000册 2016年1月北京第1次印刷
 - 著作权合同登记号 图字：01-2015-3677号
-

定价：79.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

O'Reilly Media, Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O'Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O'Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar 博客有口皆碑。”

——*Wired*

“O'Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O'Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

献给Mary、Karin、Tom和Roxie。

目录

前言	xiv
第 1 章 Python 初探	1
1.1 真实世界中的 Python	5
1.2 Python 与其他语言	5
1.3 为什么选择 Python	7
1.4 何时不应该使用 Python	8
1.5 Python 2 与 Python 3	8
1.6 安装 Python	9
1.7 运行 Python	9
1.7.1 使用交互式解释器	9
1.7.2 使用 Python 文件	10
1.7.3 下一步	11
1.8 禅定一刻	11
1.9 练习	11
第 2 章 Python 基本元素：数字、字符串和变量	13
2.1 变量、名字和对象	13
2.2 数字	16
2.2.1 整数	17
2.2.2 优先级	20
2.2.3 基数	21
2.2.4 类型转换	22
2.2.5 一个 int 型有多大	23
2.2.6 浮点数	24

2.2.7	数学函数	24
2.3	字符串	24
2.3.1	使用引号创建	25
2.3.2	使用 <code>str()</code> 进行类型转换	27
2.3.3	使用 \ 转义	27
2.3.4	使用 + 拼接	28
2.3.5	使用 * 复制	28
2.3.6	使用 [] 提取字符	28
2.3.7	使用 [start:end:step] 分片	29
2.3.8	使用 <code>len()</code> 获得长度	31
2.3.9	使用 <code>split()</code> 分割	32
2.3.10	使用 <code>join()</code> 合并	32
2.3.11	熟悉字符串	32
2.3.12	大小写与对齐方式	33
2.3.13	使用 <code>replace()</code> 替换	34
2.3.14	更多关于字符串的内容	35
2.4	练习	35
	第3章 Python 容器：列表、元组、字典与集合	36
3.1	列表和元组	36
3.2	列表	37
3.2.1	使用 [] 或 <code>list()</code> 创建列表	37
3.2.2	使用 <code>list()</code> 将其他数据类型转换成列表	37
3.2.3	使用 [offset] 获取元素	38
3.2.4	包含列表的列表	39
3.2.5	使用 [offset] 修改元素	39
3.2.6	指定范围并使用切片提取元素	40
3.2.7	使用 <code>append()</code> 添加元素至尾部	40
3.2.8	使用 <code>extend()</code> 或 += 合并列表	40
3.2.9	使用 <code>insert()</code> 在指定位置插入元素	41
3.2.10	使用 <code>del</code> 删除指定位置的元素	41
3.2.11	使用 <code>remove()</code> 删除具有指定值的元素	42
3.2.12	使用 <code>pop()</code> 获取并删除指定位置的元素	42
3.2.13	使用 <code>index()</code> 查询具有特定值的元素位置	42
3.2.14	使用 <code>in</code> 判断值是否存在	42
3.2.15	使用 <code>count()</code> 记录特定值出现的次数	43
3.2.16	使用 <code>join()</code> 转换为字符串	43
3.2.17	使用 <code>sort()</code> 重新排列元素	44
3.2.18	使用 <code>len()</code> 获取长度	44
3.2.19	使用 = 赋值，使用 <code>copy()</code> 复制	45

3.3	元组	46
3.3.1	使用 () 创建元组	46
3.3.2	元组与列表	47
3.4	字典	47
3.4.1	使用 {} 创建字典	48
3.4.2	使用 dict() 转换为字典	48
3.4.3	使用 [key] 添加或修改元素	49
3.4.4	使用 update() 合并字典	50
3.4.5	使用 del 删除具有指定键的元素	51
3.4.6	使用 clear() 删除所有元素	51
3.4.7	使用 in 判断是否存在	51
3.4.8	使用 [key] 获取元素	52
3.4.9	使用 keys() 获取所有键	52
3.4.10	使用 values() 获取所有值	53
3.4.11	使用 items() 获取所有键值对	53
3.4.12	使用 = 赋值, 使用 copy() 复制	53
3.5	集合	53
3.5.1	使用 set() 创建集合	54
3.5.2	使用 set() 将其他类型转换为集合	54
3.5.3	使用 in 测试值是否存在	55
3.5.4	合并及运算符	56
3.6	比较几种数据结构	58
3.7	建立大型数据结构	59
3.8	练习	60
第 4 章 Python 外壳: 代码结构		61
4.1	使用 # 注释	61
4.2	使用 \ 连接	62
4.3	使用 if、elif 和 else 进行比较	63
4.4	使用 while 进行循环	66
4.4.1	使用 break 跳出循环	66
4.4.2	使用 continue 跳到循环开始	67
4.4.3	循环外使用 else	67
4.5	使用 for 迭代	68
4.5.1	使用 break 跳出循环	69
4.5.2	使用 continue 跳到循环开始	69
4.5.3	循环外使用 else	69
4.5.4	使用 zip() 并行迭代	70
4.5.5	使用 range() 生成自然数序列	71
4.5.6	其他迭代方式	71

4.6	推导式	72
4.6.1	列表推导式	72
4.6.2	字典推导式	74
4.6.3	集合推导式	74
4.6.4	生成器推导式	74
4.7	函数	75
4.7.1	位置参数	79
4.7.2	关键字参数	79
4.7.3	指定默认参数值	79
4.7.4	使用 * 收集位置参数	80
4.7.5	使用 ** 收集关键字参数	81
4.7.6	文档字符串	82
4.7.7	一等公民：函数	82
4.7.8	内部函数	84
4.7.9	闭包	84
4.7.10	匿名函数：lambda() 函数	85
4.8	生成器	86
4.9	装饰器	87
4.10	命名空间和作用域	89
4.11	使用 try 和 except 处理错误	91
4.12	编写自己的异常	93
4.13	练习	94
第 5 章 Python 盒子：模块、包和程序		95
5.1	独立的程序	95
5.2	命令行参数	96
5.3	模块和 import 语句	96
5.3.1	导入模块	96
5.3.2	使用别名导入模块	98
5.3.3	导入模块的一部分	98
5.3.4	模块搜索路径	98
5.4	包	99
5.5	Python 标准库	99
5.5.1	使用 setdefault() 和 defaultdict() 处理缺失的键	100
5.5.2	使用 Counter() 计数	101
5.5.3	使用有序字典 OrderedDict() 按键排序	103
5.5.4	双端队列：栈 + 队列	103
5.5.5	使用 itertools 迭代器结构	104
5.5.6	使用 pprint() 友好输出	105

5.6 获取更多 Python 代码.....	105
5.7 练习.....	106
第 6 章 对象和类.....	107
6.1 什么是对象.....	107
6.2 使用 <code>class</code> 定义类.....	108
6.3 继承.....	109
6.4 覆盖方法.....	111
6.5 添加新方法.....	112
6.6 使用 <code>super</code> 从父类得到帮助.....	112
6.7 <code>self</code> 的自辩.....	113
6.8 使用属性对特性进行访问和设置.....	114
6.9 使用名称重整保护私有特性.....	117
6.10 方法的类型.....	118
6.11 鸭子类型.....	119
6.12 特殊方法.....	120
6.13 组合.....	123
6.14 何时使用类和对象而不是模块.....	124
6.15 练习.....	126
第 7 章 像高手一样玩转数据.....	127
7.1 文本字符串.....	127
7.1.1 Unicode.....	127
7.1.2 格式化.....	134
7.1.3 使用正则表达式匹配.....	137
7.2 二进制数据.....	144
7.2.1 字节和字节数组.....	144
7.2.2 使用 <code>struct</code> 转换二进制数据.....	145
7.2.3 其他二进制数据工具.....	148
7.2.4 使用 <code>binascii()</code> 转换字节 / 字符串.....	149
7.2.5 位运算符.....	149
7.3 练习.....	149
第 8 章 数据的归宿.....	152
8.1 文件输入 / 输出.....	152
8.1.1 使用 <code>write()</code> 写文本文件.....	153
8.1.2 使用 <code>read()</code> 、 <code>readline()</code> 或者 <code>readlines()</code> 读文本文件.....	154
8.1.3 使用 <code>write()</code> 写二进制文件.....	156
8.1.4 使用 <code>read()</code> 读二进制文件.....	157
8.1.5 使用 <code>with</code> 自动关闭文件.....	157

8.1.6 使用 <code>seek()</code> 改变位置	157
8.2 结构化的文本文件	159
8.2.1 CSV	159
8.2.2 XML	161
8.2.3 HTML	163
8.2.4 JSON	163
8.2.5 YAML	165
8.2.6 安全提示	166
8.2.7 配置文件	167
8.2.8 其他交换格式	168
8.2.9 使用 <code>pickle</code> 序列化	168
8.3 结构化二进制文件	169
8.3.1 电子数据表	169
8.3.2 层次数据格式	169
8.4 关系型数据库	170
8.4.1 SQL	170
8.4.2 DB-API	172
8.4.3 SQLite	172
8.4.4 MySQL	174
8.4.5 PostgreSQL	174
8.4.6 SQLAlchemy	174
8.5 NoSQL 数据存储	179
8.5.1 dbm family	180
8.5.2 memcached	180
8.5.3 Redis	181
8.5.4 其他的 NoSQL	189
8.6 全文数据库	189
8.7 练习	190
第 9 章 剖析 Web	191
9.1 Web 客户端	192
9.1.1 使用 <code>telnet</code> 进行测试	193
9.1.2 Python 的标准 Web 库	194
9.1.3 抛开标准库: <code>requests</code>	195
9.2 Web 服务端	196
9.2.1 最简单的 Python Web 服务器	196
9.2.2 Web 服务器网关接口	198
9.2.3 框架	198
9.2.4 Bottle	198
9.2.5 Flask	201

9.2.6 非 Python 的 Web 服务器	204
9.2.7 其他框架	206
9.3 Web 服务和自动化	207
9.3.1 <code>webbrowser</code> 模块	207
9.3.2 Web API 和表达性状态传递	208
9.3.3 JSON	209
9.3.4 抓取数据	209
9.3.5 用 <code>BeautifulSoup</code> 来抓取 HTML	209
9.4 练习	210
第 10 章 系统	212
10.1 文件	212
10.1.1 用 <code>open()</code> 创建文件	212
10.1.2 用 <code>exists()</code> 检查文件是否存在	213
10.1.3 用 <code>isfile()</code> 检查是否为文件	213
10.1.4 用 <code>copy()</code> 复制文件	213
10.1.5 用 <code>rename()</code> 重命名文件	214
10.1.6 用 <code>link()</code> 或者 <code>symlink()</code> 创建链接	214
10.1.7 用 <code>chmod()</code> 修改权限	214
10.1.8 用 <code>chown()</code> 修改所有者	214
10.1.9 用 <code>abspath()</code> 获得路径名	215
10.1.10 用 <code>realpath()</code> 获得符号的路径名	215
10.1.11 用 <code>remove()</code> 删除文件	215
10.2 目录	215
10.2.1 使用 <code>mkdir()</code> 创建目录	215
10.2.2 使用 <code>rmdir()</code> 删除目录	215
10.2.3 使用 <code>listdir()</code> 列出目录内容	216
10.2.4 使用 <code>chdir()</code> 修改当前目录	216
10.2.5 使用 <code>glob()</code> 列出匹配文件	216
10.3 程序和进程	217
10.3.1 使用 <code>subprocess</code> 创建进程	218
10.3.2 使用 <code>multiprocessing</code> 创建进程	219
10.3.3 使用 <code>terminate()</code> 终止进程	219
10.4 日期和时间	220
10.4.1 <code>datetime</code> 模块	221
10.4.2 使用 <code>time</code> 模块	223
10.4.3 读写日期和时间	225
10.4.4 其他模块	227
10.5 练习	228

第 11 章 并发和网络.....	229
11.1 并发.....	230
11.1.1 队列.....	231
11.1.2 进程.....	231
11.1.3 线程.....	232
11.1.4 绿色线程和 <code>gevent</code>	234
11.1.5 <code>twisted</code>	236
11.1.6 <code>asyncio</code>	238
11.1.7 Redis.....	238
11.1.8 队列之上.....	241
11.2 网络.....	241
11.2.1 模式.....	242
11.2.2 发布 - 订阅模型.....	242
11.2.3 TCP/IP.....	245
11.2.4 套接字.....	246
11.2.5 ZeroMQ.....	250
11.2.6 <code>scapy</code>	253
11.2.7 网络服务.....	253
11.2.8 Web 服务和 API	255
11.2.9 远程处理.....	256
11.2.10 大数据和 MapReduce.....	260
11.2.11 在云上工作.....	261
11.3 练习.....	264
第 12 章 成为真正的 Python 开发者.....	265
12.1 关于编程.....	265
12.2 寻找 Python 代码.....	265
12.3 安装包.....	266
12.3.1 使用 <code>pip</code>	266
12.3.2 使用包管理工具.....	267
12.3.3 从源代码安装.....	267
12.4 集成开发环境.....	268
12.4.1 IDLE.....	268
12.4.2 PyCharm.....	268
12.4.3 IPython	269
12.5 命名和文档.....	269
12.6 测试代码.....	270
12.6.1 使用 <code>pylint</code> 、 <code>pyflakes</code> 和 <code>pep8</code> 检查代码	270
12.6.2 使用 <code>unittest</code> 进行测试.....	272

12.6.3 使用 <code>doctest</code> 进行测试.....	276
12.6.4 使用 <code>nose</code> 进行测试.....	277
12.6.5 其他测试框架.....	278
12.6.6 持续集成.....	278
12.7 调试 Python 代码.....	278
12.8 使用 <code>pdb</code> 进行调试.....	279
12.9 记录错误日志.....	284
12.10 优化代码.....	286
12.10.1 测量时间	286
12.10.2 算法和数据结构	288
12.10.3 Cython、NumPy 和 C 扩展	289
12.10.4 PyPy.....	289
12.11 源码控制.....	289
12.11.1 Mercurial	290
12.11.2 Git.....	290
12.12 复制本书代码.....	292
12.13 更多内容.....	293
12.13.1 书	293
12.13.2 网站	293
12.13.3 社区	293
12.13.4 大会	294
12.14 后续内容.....	294
附录 A Python 的艺术.....	295
附录 B 工作中的 Python	307
附录 C Python 的科学.....	320
附录 D 安装 Python 3	339
附录 E 习题解答	349
附录 F 速查表	380
作者介绍.....	383
封面介绍.....	383

前言

本书介绍 Python 编程语言，主要面向编程初学者。不过，如果你是一位有经验的程序员，想再学门 Python 编程语言，本书也很适合作为入门读物。

本书节奏适中，从基础开始逐步深入其他话题。我会结合食谱和教程的风格来解释新术语和新概念，但不会一次介绍很多。你会尽早并且常常接触到真实的 Python 代码。

虽然本书是入门读物，但我还是介绍了一些看起来比较高阶的话题，比如 NoSQL 数据库和消息传递库。之所以介绍它们，是因为在解决某类问题时它们比标准库更加合适。你需要下载并安装这些第三方 Python 包，从而更好地理解 Python “内置电池”适用于什么场景。此外，尝试新事物本身也充满乐趣。

我还会展示一些反面的例子，提醒你不要那么去做。如果你之前使用过其他语言并且想把风格照搬到 Python 的话，要格外注意。还有，我不认为 Python 是完美的，我会告诉你哪些东西应该避免。



书中有时会出现类似本条的提示内容，主要用于解释一些容易混淆的概念或者用更合适的 Python 风格的方法来解决同一个问题。

目标读者

本书的目标读者是那些对世界上最流行的计算语言感兴趣的人，无论你之前是否学过编程。

本书结构

本书前 7 章介绍 Python 基础知识，建议按顺序阅读。后面 5 章介绍如何在不同的应用场景中使用 Python，比如 Web、数据库、网络，等等，可以按任意顺序阅读。附录 A、B、C 介绍 Python 在艺术、商业和科学方面的应用，附录 D 是 Python 3 的安装教程，附录 E 和附录 F 是每章练习题的答案和速查表。

- 第 1 章

程序和织袜子或者烤土豆很像。通过一些真实的 Python 程序可以了解这门语言的概貌、能力以及在真实世界中的用途。Python 和其他语言相比有很多优势，不过也有一些不完美的地方。旧版本的 Python（Python 2）正在被新版本（Python 3）替代。如果你在使用 Python 2，请安装 Python 3。你可以使用交互式解释器自行尝试本书中的代码示例。

- 第 2 章

该章会介绍 Python 中最简单的数据类型：布尔值、整数、浮点数和文本字符串。你也会学习基础的数学和文本操作。

- 第 3 章

该章会学习 Python 的高级内置数据结构：列表、元组、字典和集合。你可以像玩乐高积木一样用它们来构建更复杂的结构，并学到如何使用迭代器和推导式来遍历它们。

- 第 4 章

该章会学习如何在之前学习的数据结构上用代码实现比较、选择和重复操作。你会学习如何用函数来组织代码，并用异常来处理错误。

- 第 5 章

该章会介绍如何使用模块、包和程序组织大型代码结构。你会学习如何划分代码和数据、数据的输入输出、处理选项、使用 Python 标准库并了解标准库的内部实现。

- 第 6 章

如果你已经在其他语言中学过面向对象编程，就可以轻松掌握 Python 的写法。该章会介绍对象和类的适用场景，有时候使用模块甚至列表和字典会更加合适。

- 第 7 章

该章会介绍如何像专家一样处理数据。你会学到如何处理文本和二进制数据以及 Unicode 字符和 I/O。

- 第 8 章

数据需要地方来存放。在该章中，你首先会学习使用普通文件、目录和文件系统，接着会学习如何处理常用文件格式，比如 CSV、JSON 和 XML。此外，你还会了解如何从关系型数据库甚至是最新的 NoSQL 数据库中存取数据。

- 第 9 章

该章单独介绍 Web，包括客户端、服务器、数据抓取、API 和框架。你会编写一个带请求参数处理和模板的真实网站。

- 第 10 章

该章会介绍系统相关内容，难度较高。你会学习如何管理程序、进程和线程，处理日期和时间，实现系统管理任务自动化。

- 第 11 章

该章会介绍网络相关内容：服务、协议和 API。该章示例覆盖了底层 TCP 套接字、消

息库以及队列系统、云端部署。

- 第 12 章

该章会介绍 Python 相关的小技巧，比如安装、使用 IDE、测试、调试、日志、版本控制和文档，还会介绍如何寻找并安装有用第三方包、打包自己的代码以供重用，以及如何寻找更多有用的信息。祝你好运。

- 附录 A

附录 A 会介绍 Python 在艺术领域的应用：图像、音乐、动画和游戏。

- 附录 B

Python 在商业领域也有应用：数据可视化（图表、图形和地图）、安全和管理。

- 附录 C

Python 在科学领域应用得尤其广泛：数学和统计学、物理科学、生物科学以及医学。

附录 C 会介绍 NumPy、SciPy 和 Pandas。

- 附录 D

如果你还没有安装 Python 3，附录 D 会介绍 Windows、Mac OS/X、Linux 和 Unix 下的安装方法。

- 附录 E

附录 E 包含每章结尾的练习答案。请在亲自尝试解答之后再查看答案。

- 附录 F

附录 F 包含一些有用的速查内容。

Python 版本

开发者会不断向计算机语言中加入新特性、修复问题，因此计算机语言一直在变化。本书中的代码示例在 Python 3.3 中编写和测试。在本书编辑期间 Python 3.4 发布了，我会介绍一些新版本的内容。如果你想了解相关信息和特性的发布时间，可以阅读 What's New in Python 页面 (<https://docs.python.org/3/whatsnew/>)。这个页面技术性比较强，对于 Python 初学者来说难度较大，不过如果你之后想研究 Python 的兼容性，可以阅读它。

排版约定

本书使用了下列排版约定。

- 楷体

表示新术语。

- 等宽字体 (`constant width`)

表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键字等。

- 加粗等宽字体 (**constant width bold**)
表示应该由用户输入的命令或其他文本。



该图标表示一般注记。



该图标表示警告或警示。

使用代码示例

补充材料（代码示例、练习等）可以从 <https://github.com/madscheme/introducing-python> 下载。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如：“*Introducing Python* by Bill Lubanovic(O'Reilly). Copyright 2015 Bill Lubanovic, 978-1-449-35936-2.”

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

Safari® Books Online



Safari Books Online (<http://www.safaribooksonline.com>) 是应运而生的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM

Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O'Reilly 的每一本书都有专属网页，你可以在那儿找到本书的相关信息，包括勘误表、示例代码以及其他信息。本书的网站地址是：

<http://shop.oreilly.com/product/0636920028659.do>

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

<http://www.oreilly.com>

我们在 Facebook 的地址如下：

<http://facebook.com/oreilly>

请关注我们的 Twitter 动态：

<http://twitter.com/oreillymedia>

我们的 YouTube 视频地址如下：

<http://www.youtube.com/oreillymedia>

致谢

非常感谢那些阅读本书初稿并给予反馈的人。我尤其要感谢仔细审阅本书的 Eli Bessert、Henry Canival、Jeremy Elliott、Monte Milanuk、Loïc Pefferkorn 和 Steven Wayne。

第1章

Python初探

我们从一个小谜题以及它的答案开始。你认为下面这两行的含义是什么？

(Row 1): (RS) K18,ssk,k1,turn work.
(Row 2): (WS) Sl 1 pwise,p5,p2tog,p1,turn.

它们看起来像是某种计算机程序。实际上，这是一个针织图案。更准确地说，这两行描述的是如何编织袜子的足跟部分。对我来说，看懂它们就像让猫看懂《纽约时报》上的填字游戏一样难，但是对我妻子来说却轻而易举。如果你也懂编织，一样可以轻松看懂。

来看另一个例子。虽然你不知道最终会做出什么，但是马上就能明白下面的内容是什么。

1/2杯黄油或者人造黄油
1/2杯奶油
2.5杯面粉
1茶匙盐
1汤匙糖
4杯糊状土豆(冷藏)

确保在加入面粉之前冷藏所有材料。

混合所有材料。

用力揉。

揉成20个球并冷藏。

对于每一个球：

在布上洒上面粉。

用擀面杖把球擀成圆饼。

入锅，炸至棕色。

翻面继续炸。

即使你不会做饭，应该也能看懂这是一个菜谱：一系列食物原料以及准备工作。这道菜是什么呢？是 lefse，一道和玉米饼很像的挪威美食。做好之后抹上黄油、果酱或者其他你喜欢吃的东西，最后卷起来吃。

编织图案和菜谱有一些共同的特征。

- 专有名词、缩写以及符号。有些很常见，有些很难懂。
- 规定专有名词、缩写以及符号的使用方法，也就是它们的语法。
- 一个操作序列，按照顺序进行。
- 有时需要重复一些操作（循环），比如炸 lefse 的每一面。
- 有时需要引用其他操作序列（用计算机术语来说就是一个函数）。在菜谱中，你可能需要引用另一个将土豆捣成糊状的菜谱。
- 假定已经有相关知识。菜谱假定你知道水是什么以及如何烧水。编织图案假定你学过编织并且不会经常扎到手。
- 一个期望的结果。在我们的例子中分别是袜子和食物，千万不要把它们混在一起哦。

以上这些概念都会出现在计算机程序中。这两个例子的目的是让你知道编程并不像看起来那么高深莫测，其实只是学习一些正确的单词和规则而已。

下面来看看真正的程序。你知道它在做什么吗？

```
for countdown in 5, 4, 3, 2, 1, "hey!":  
    print(countdown)
```

这其实是一段 Python 程序，会打印出下面的内容：

```
5  
4  
3  
2  
1  
hey!
```

看到了吗？学习 Python 就像看懂菜谱或者编织图案一样简单。此外，你可以在桌子上舒服并且安全地练习编写 Python 程序，完全不用担心被热水烫到或者被针扎到。

Python 程序有一些特殊的单词和符号——`for`、`in`、`print`、逗号、冒号、括号以及其他符号。这些单词和符号是语法的重要组成部分。好消息是，Python 的语法非常优秀，相比其他大多数编程语言，学习 Python 需要记住的语法内容很少。Python 语法非常自然，就像一份菜谱一样。

下面的 Python 程序会从一个 Python 列表（list）中选出一条电视新闻的常用语并打印出来：

```
cliches = [  
    "At the end of the day",  
    "Having said that",  
    "The fact of the matter is",  
    "Be that as it may",  
    "The bottom line is",  
    "If you will",  
]  
print(cliches[3])
```

程序会打印出第四条常用语：

```
Be that as it may
```

一个 Python 列表，比如 `cliches`，就是一个值序列，可以通过它们相对于列表起始位置的偏移量来访问。第一个值的偏移量是 0，第四个值的偏移量是 3。



人们通常从 1 开始数数，所以从 0 开始数似乎很奇怪。用偏移量来代替位置会更好理解一些。

列表在 Python 中很常用，第 3 章会讲解列表的用法。

下面这段程序同样会打印出一条引用内容，但是这次是用说话者的人名而不是列表中的位置来进行访问：

```
quotes = {  
    "Moe": "A wise guy, huh?",  
    "Larry": "Ow!",  
    "Curly": "Nyuk nyuk!",  
}  
stooge = "Curly"  
print(stooge, "says:", quotes[stooge])
```

运行这个小程序会打印出：

```
Curly says: Nyuk nyuk!
```

`quotes` 是一个 Python 字典。字典是一个集合，包含唯一键（本例中是跟屁虫“Stooge”的名字）及其关联的值（本例中是跟屁虫说的话）。使用字典可以通过名字来存储和查找东西，和列表一样非常有用。第 3 章会详细讲解字典。

常用语的例子中使用方括号 ([和]) 来创建 Python 列表，跟屁虫的例子中使用大括号 ({ 和 })，大括号的英文是 curly bracket，但是大括号和 Curly 没有任何关系¹ 来创建 Python 字典。这些都是 Python 的语法，在之后的内容中你会看到更多语法。

现在我们来看另一个完全不同的例子：示例 1-1 中的 Python 程序会执行一系列复杂的任务。你可能还看不懂这段程序，没关系，学完本书之后就可以看懂了。这个例子的目的是让你了解典型的 Python 程序长什么样。如果你了解其他计算机语言，可以对比一下。

示例 1-1 会连接 YouTube 网站并获取当前评价最高的视频的信息。如果 YouTube 返回的是常见的 HTML 文本，那就很难从中挖掘出我们想要的信息（9.3.4 节会介绍网页抓取）。幸运的是，它返回的是 JSON 格式的数据，这种格式可以直接用计算机处理。JSON (JavaScript Object Notation, JavaScript 对象符号) 是一种人类可以阅读的文本格式，它描述了类型、值以及值的顺序。JSON 就像一个小型编程语言，使用 JSON 在不同计算机语言和系统之间交换数据已经成为了一种非常流行的方式。更多关于 JSON 的内容请参考 8.2.4 节。

注 1：Curly 是美国乌鸦童子军 (Crow Scouts) 的一员，乌鸦童子军是美国和印第安人打仗时由印第安人战俘组成的军队。Curly 是小巨角河战役中为数不多的幸存者之一。小巨角河战役是美军和北美势力最庞大的苏族印地安人之间的战争，在这场战争中印第安人歼灭了美国历史上最有名的第七骑兵团，Curly 当时没有参战，他是第一个报告第七骑兵团战败的人，也因此出名。——译者注

Python 程序可以把 JSON 文本翻译成 Python 的数据结构——你会在之后的两章中学习它们——和你自己创建出来的一样。这个 YouTube 响应包含很多数据，作为演示我只打印出了前 6 个视频的标题。再说一次，这是一个完整的 Python 程序，你自己也可以运行一下。

示例 1-1：intro/youtube.py

```
import json
from urllib.request import urlopen
url = "https://gdata.youtube.com/feeds/api.standardfeeds/top_rated?alt=json"
response = urlopen(url)
contents = response.read()
text = contents.decode('utf8')
data = json.loads(text)
for video in data['feed']['entry'][0:6]:
    print(video['title']['$t'])
```

最后一次运行这个程序得到的输出是：

```
Evolution of Dance - By Judson Laipply
Linkin Park - Numb
Potter Puppet Pals: The Mysterious Ticking Noise
"Chocolate Rain" Original Song by Tay Zonday
Charlie bit my finger - again !
The Mean Kitty Song
```

这个 Python 小程序仅仅用了 9 行代码就很好地完成了任务，并且具备很高的可读性。如果你看不懂下面的术语，没关系，接下来的几章会让你明白它们的意思。

- 第 1 行：从 Python 标准库中导入名为 `json` 的所有代码。
- 第 2 行：从 Python 标准 `urllib` 库中导入 `urlopen` 函数。
- 第 3 行：给变量 `url` 赋值一个 YouTube 地址。
- 第 4 行：连接指定地址处的 Web 服务器并请求指定的 Web 服务。
- 第 5 行：获取响应数据并赋值给变量 `contents`。
- 第 6 行：把 `contents` 解码成一个 JSON 格式的文本字符串并赋值给变量 `text`。
- 第 7 行：把 `text` 转换为 `data`——一个存储视频信息的 Python 数据结构。
- 第 8 行：每次获取一个视频的信息并赋值给变量 `video`。
- 第 8 行：使用两层 Python 字典 (`data['feed']['entry']`) 和切片操作 (`[0:6]`)。
- 第 9 行：使用 `print` 函数打印出视频标题。

视频信息中包含多种你之前见过的 Python 数据结构，第 3 章会详细介绍。

在这个例子中，我们使用了一些 Python 标准库模块（它们是安装 Python 时就已经包含的程序），但是它们并不是最好的。下面的代码使用第三方 Python 软件包 `requests` 重写了这个例子：

```
import requests
url = "https://gdata.youtube.com/feeds/api.standardfeeds/top_rated?alt=json"
response = requests.get(url)
data = response.json()
for video in data['feed']['entry'][0:6]:
    print(video['title']['$t'])
```

新版代码只有 6 行，并且我认为可读性更高。第 5 章会详细介绍 `requests` 以及其他第三方 Python 软件。

1.1 真实世界中的Python

那么，是否真的值得付出时间和努力来学习 Python 呢？它真的有用吗？实际上，Python 诞生于 1991 年（比 Java 还早），并且一直是最流行的十门计算机语言之一。公司需要雇用程序员来写 Python 程序，包括你每天都会用到的 Google、YouTube、Dropbox、Netflix 和 Hulu 等。我用 Python 开发过许多产品级应用，从邮件搜索应用到商业网站都有。对于发展迅速的组织来说，Python 能极大地提高生产力。

Python 可以应用在许多计算环境下，如下所示：

- 命令行窗口
- 图形用户界面，包括 Web
- 客户端和服务端 Web
- 大型网站后端
- 云（第三方负责管理的服务器）
- 移动设备
- 嵌入式设备

Python 程序从一次性脚本——就像你在本章中看到的一样——到几十万行的系统都有。我们会介绍 Python 在网站、系统管理和数据处理方面的应用，还会介绍 Python 在艺术、科学和商业方面的应用。

1.2 Python与其他语言

Python 和其他语言相比如何呢？什么时候该选择什么语言呢？本节会展示一些其他语言的代码片段，这样更直观一些。如果有些语言你从未使用过，也不必担心，你并不需要看懂所有代码（当你看到最后的 Python 示例时，会发现没学过其他语言也不是什么坏事）。如果你只对 Python 感兴趣，完全可以跳过这一节。

下面的每段程序都会打印出一个数字和一条描述语言的信息。

如果你使用的是命令行或者终端窗口，那你使用的就是 shell 程序，它会读入你的命令、运行并显示结果。Windows 的 shell 叫作 cmd，它会运行后缀为 .bat 的 batch 文件。Linux 和其他类 Unix 系统（包括 Mac OS X）有许多 shell 程序，最流行的称为 bash 或者 sh。shell 有许多简单的功能，比如执行简单的逻辑操作以及把类似 * 的通配符扩展成文件名。你可以把命令保存到名为“shell 脚本”的文件中稍后运行。shell 可能是程序员接触到的第一个程序。它的问题在于程序超过百行之后扩展性很差，并且比其他语言的运行速度慢很多。下面就是一段 shell 程序：

```
#!/bin/sh
language=0
echo "Language $language: I am the shell. So there."
```

如果你把这段代码保存为 `meh.sh` 并通过 `sh meh.sh` 命令来运行它，就会看到下面的输出：

```
Language 0: I am the shell. So there.
```

老牌语言 C 和 C++ 是底层语言，只有极其重视性能时才会使用。它们很难学习，并且有许多细节需要你自己处理，处理不当就可能导致程序崩溃和其他很难解决的问题。下面是一段 C 程序：

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int language = 1;
    printf("Language %d: I am C! Behold me and tremble!\n", language);
    return 0;
}
```

C++ 和 C 看起来很相似，但是特性完全不同：

```
#include <iostream>
using namespace std;
int main(){
    int language = 2;
    cout << "Language " << language << \
        ": I am C++! Pay no attention to that C behind the curtain!" << \
        endl;
    return(0);
}
```

Java 和 C# 是 C 和 C++ 的接班人，解决了后者的许多缺点，但是相比之下代码更加冗长，写起来也有许多限制。下面是 Java 代码：

```
public class Overlord {
    public static void main (String[] args) {
        int language = 3;
        System.out.format("Language %d: I am Java! Scarier than C!\n", language);
    }
}
```

如果你没写过这些语言的程序，可能会觉得很奇怪：这都是什么东西？有些语言有很大的语法包袱。它们有时被称为静态语言，因为你必须告诉计算机许多底层细节，下面我来解释一下。

语言有变量——你想在程序中使用的值的名字。静态语言要求你必须声明每个变量的类型：它会使用多少内存以及允许的使用方法。计算机利用这些信息把程序编译成非常底层的机器语言（专门给计算机硬件使用的语言，硬件很容易理解，但是人类很难理解）。计算机语言的设计者通常必须进行权衡，到底是让语言更容易被人使用还是更容易被计算机使用。声明变量类型可以帮助计算机发现更多潜在的错误并提高运行速度，但是却需要使用者进行更多的思考和编程。C、C++ 和 Java 代码中经常需要声明类型。举例来说，在上面的例子中必须使用 `int` 将 `language` 变量声明为一个整数。（其他类型的存储方式和整数不同，比如浮点数 `3.14159`、字符以及文本数据。）

那么为什么它们被称为静态语言呢？因为这些语言中的变量不能改变类型。它们是静态的。整数就是整数，永远无法改变。

相比之下，动态语言（也被称为脚本语言）并不需要在使用变量前进行声明。假设你输入 `x = 5`，动态语言知道 5 是一个整数，因此变量 `x` 也是整数。这些语言允许你用更少的代码做更多的事情。动态语言的代码不会被编译，而是由解释器程序来解释执行。动态语言通常比编译后的静态语言更慢，但是随着解释器的不断优化，动态语言的速度也在不断提升。长期以来，动态语言的主要应用场景都是很短的程序（脚本），比如给静态语言编写的程序进行数据预处理。这样的程序通常称为胶水代码。虽然动态语言很擅长做这些事，但是如今它们也已经具备了处理大型任务的能力。

许多年来，Perl (<http://www.perl.org/>) 一直是一门万能的动态语言。Perl 非常强大并且有许多扩展库。然而，它的语法非常难用，并且似乎无法阻挡 Python 和 Ruby 的崛起。下面是一段 Perl 代码：

```
my $language = 4;
print "Language $language: I am Perl, the camel of languages.\n";
```

Ruby (<http://www.ruby-lang.org/>) 是一门新语言。它借鉴了一些 Perl 的特点，并且因为 Web 开发框架 Ruby on Rails 红遍大江南北。Ruby 和 Python 的许多应用场景相同，选择哪一个通常看个人喜好或者是否有你需要的库。下面是一段 Ruby 代码：

```
language = 5
puts "Language #{language}: I am Ruby, ready and aglow."
```

PHP (<http://www.php.net/>) 在 Web 开发领域非常流行，因为它可以轻松结合 HTML 和代码，就像例子中展示的那样。然而，PHP 语言本身有许多缺陷，并且很少被应用在 Web 以外的领域。

```
<?PHP
$language = 6;
echo "Language $language: I am PHP. The web is <i>mine</i>, I say.\n";
?>
```

最后是我们的主角，Python：

```
language = 7
print("Language %s: I am Python. What's for supper?" % language)
```

1.3 为什么选择Python

Python 是一门非常通用的高级语言。它的设计极大地增强了代码可读性，可读性远比听上去重要得多。每个计算机程序只被编写一次，但是会被许多人阅读和修改许多次。提高可读性也可以让学习和记忆更加容易，因此也更容易修改。和其他流行的语言相比，Python 的学习曲线更加平缓，可以让你很快具备生产力，当然，想成为专家还需要深入学习才行。

Python 简洁的语法可以让你写出比静态语言更短的程序。研究证明，程序员每天可以编写的代码行数是有限的——无论什么语言，因此，如果完成同样的功能只需要编写一半长度的代码，生产力就可以提高一倍。对于重视这一点的公司来说，Python 是一个不算秘密的秘密武器。

在顶尖的美国大学中 (<http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext>)，Python 是计算机入门课程中最流行的语言。此外，它也被两千多名雇主 (<http://blog.codeeval.com/codeevalblog/2014#.U73vaPlUpw=>) 用来评估编程技能。

当然，它是免费的，就像啤酒和演讲一样。你可以免费用 Python 来编写任何东西并用在任何地方。没人可以一边阅读你的 Python 程序一边说：“这是一个非常棒的小程序，希望不会发生什么意外。”

Python 几乎可以运行在任何地方并且其标准库中有很多有用的软件。

不过，选择 Python 最关键的理由可能出乎你的意料：大家都喜欢它。实际上，大家不只是把 Python 当作一个完成工作的工具，而是非常享受用它编程。在工作中不得不用其他语言时，人们通常会非常想念 Python 的某些特性。这就是 Python 能够胜出的原因。

1.4 何时不应该使用Python

Python 并非在所有场合都是最好用的语言。

它并不是默认安装在所有环境中。如果你的电脑上没有 Python，附录 D 会告诉你如何安装。

对于大多数应用来说，Python 已经足够快了，但是有些场合下，它的性能仍然是个问题。如果你的程序会花费大量时间用于计算（专业术语是中央处理器受限），那么可以使用 C、C++ 或者 Java 来编写程序从而提高性能。但是这并不是唯一的选择！

- 有时候用 Python 实现一个更好的算法（一系列解决问题的步骤）可以打败 C 中的低效算法。Python 对于开发效率的提升可以让你有更多的时间来尝试各种选择。
- 在许多应用中，程序会因为等待其他服务器的响应而浪费时间。这段时间里 CPU（中央处理单元，计算机中负责所有计算的芯片）几乎什么都不做，因此，静态和动态程序的端到端时间几乎是一样的。
- Python 的标准解释器用 C 实现，所以可以通过 C 代码进行扩展。12.10 节会介绍一些。
- Python 解释器变得越来越快。Java 最初也很慢，经过大量的研究和资金投入之后，它变得非常快。Python 并不属于某个公司，因此它的发展会更缓慢一些。12.10.4 节会介绍 PyPy 项目及其意义。
- 可能你的项目要求非常严格，无论如何努力 Python 都无法达到要求。那么，借用伊恩·荷姆在电影《异形》中说过的一句话，我很同情你。通常来说可以选择 C、C++ 和 Java，不过新语言 Go (<http://golang.org>，写起来像 Python，性能像 C) 也是一个不错的选择。

1.5 Python 2与Python 3

你即将面临的最大问题是，Python 有两个版本。Python 2 已经存在了很长时间并且预装在 Linux 和 Apple 电脑中。Python 是一门很出色的语言，但是世界上不存在完美的东西。和

其他领域一样，在计算机语言中许多问题很容易解决，但是也有一些问题很难解决。后者的难点在于不兼容：使用修复后的新版本编写的程序无法运行在旧的 Python 系统中，旧的程序也无法运行在新的系统中。

Python 的发明者（吉多·范·罗苏姆，<https://www.python.org/~guido>）和其他开发者决定把这些困难问题放在一起解决，并把解决后的版本称作 Python 3。Python 2 已经成为过去，Python 3 才是未来。Python 2 的最后一个版本是 2.7，它会被支持很长一段时间，但也就仅此而已，再也没有 Python 2.8 了。新的开发全部会在 Python 3 上进行。

本书使用的是 Python 3。如果你使用的是 Python 2 也不用担心，两者差别不大。最明显的区别在于调用 `print` 的方式，最重要的区别则是处理 Unicode 字符的方式，详情参见第 2 章和第 7 章。流行的 Python 软件需要逐步升级，和常见的“先有鸡还是先有蛋”问题一样。不过，看起来我们现在终于到达了发生转变的临界点。

1.6 安装Python

为了让这章更加简洁，安装 Python 3 的细节参见附录 D。如果你还没安装 Python 3 或者不确定是否安装过 Python，请阅读附录 D。

1.7 运行Python

安装好 Python 3 之后，可以用它来运行本书中的 Python 程序和你自己的 Python 代码。那么如何运行 Python 程序呢？通常来说有两种方法。

- Python 自带的交互式解释器可以很方便地执行小程序。你可以一行一行输入命令然后立刻查看运行结果。这种方式可以很好地结合输入和查看结果，从而快速进行一些实验。我会用交互式解释器来说明一些语言特性，你可以在自己的 Python 环境中输入同样的命令。
- 除此之外，可以把 Python 程序存储到文本文件中，通常要加上 .py 扩展名，然后输入 `python` 加文件名来执行。

我们来分别尝试一下这两种方式。

1.7.1 使用交互式解释器

本书中大多数代码示例都用到了交互式解释器。当你输入示例中的命令并且看到同样的输出时，就可以确定你没有跑偏。

可以在电脑上输入 Python 主程序的名称来启动解释器：应该是 `python`、`python3` 或者类似的东西。在本书接下来的内容中，我们会假设它叫 `python`；如果你的 Python 主程序名字不同，请把代码示例中的 `python` 全部替换成你电脑上的名称。

交互式解释器的工作原理基本上和 Python 对文件的处理方式一样，除了一点：当你输入一些包含值的东西时，交互式解释器会自动打印出这个值。举例来说，如果你启动 Python 并输入数字 61，它会立刻出现在终端中。



在下面的例子中，\$ 表示系统提示符，用来输入终端中的命令，比如 `python`。本书的代码示例都会使用它，尽管在你的电脑中提示符可能不是 \$。

```
$ python
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 01:25:11)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 61
61
>>>
```

这种自动打印值的省时特性只有交互式解释器中有，在 Python 语言中没有。

顺便说一句，也可以使用 `print()` 在解释器中打印内容：

```
>>> print(61)
61
>>>
```

如果你亲自动手在交互式解释器中执行了上面这些例子并看到了相同的结果，恭喜你，你成功运行了真正的 Python 代码（虽然有点短）。接下来的几章中会接触到更长的 Python 程序。

1.7.2 使用Python文件

如果你把 61 放在文件中并使用 Python 来执行它，确实可以，但是程序什么都不会输出。在非交互式的 Python 程序中，你必须调用 `print` 函数来打印内容，如下所示：

```
print(61)
```

我们来生成一个 Python 程序文件并运行它。

- (1) 打开你的文本编辑器。
- (2) 输入代码 `print(61)`，如上所示。
- (3) 保存文件，命名为 `61.py`。一定要确保存储为纯文本而不是“富文本”格式，比如 RTF 或者 Word。Python 程序文件并不是一定要以 `.py` 结尾，但是加上它可以使你清楚这个文件的作用。
- (4) 如果你使用的是图形用户界面——基本上每个人都会用——请打开一个终端窗口²。
- (5) 输入下面的命令来运行程序：

```
$ python 61.py
```

应该可以看到一行输出：

```
61
```

注 2：如果你不明白什么是终端窗口，请阅读附录 D。

成功了吗？如果成功了，恭喜你运行了你的第一个 Python 程序！

1.7.3 下一步

你可以向真正的 Python 系统中输入命令，不过它们必须符合 Python 的语法。我们不会一次性向你展示所有的语法规则，而是在接下来的几章中逐一进行讲解。

开发 Python 程序最基础的方法是使用一个纯文本编辑器和一个终端窗口。在本书中会直接展示纯文本，有时候在交互式终端中，有时候在 Python 文件中。不过除此之外，还有很多优秀的 Python 集成开发环境（IDE）。它们的图形用户界面可能会有许多高端文本编辑功能和辅助开发功能。在第 12 章中你会看到一些相关介绍。

1.8 禅定一刻

每种计算机语言都有自己的风格。在前言中我提到过，你可以用 Python 的方式来表达自己。Python 中内置了一些自由体诗歌，它们简单明了地说明了 Python 的哲学（就我所知，Python 是唯一一个包含这种复活节彩蛋的语言）。只要在交互式解释器中输入 `import this`，然后按下回车就能看到它们：

```
>>> import this
《Python之禅》 Tim Peters

优美胜于丑陋
明了胜于隐晦
简洁胜于复杂
复杂胜于混乱
扁平胜于嵌套
宽松胜于紧凑
可读性很重要
即便是特例，也不可违背这些规则
虽然现实往往不那么完美
但是不应该放过任何异常
除非你确定需要如此
如果存在多种可能，不要猜测
肯定有一种——通常也是唯一一种——最佳的解决方案
虽然这并不容易，因为你不是Python之父
动手比不动手要好
但不假思索就动手还不如不做
如果你的方案很难懂，那肯定不是一个好方案
如果你的方案很好懂，那肯定是一个好方案
命名空间非常有用，我们应当多加利用
```

这些哲学观点会贯穿全书。

1.9 练习

本章介绍了 Python 语言——它是干什么的、它是什么样的以及它在计算机世界中的作用。在每章的结尾我都会列出一些小练习来帮助你巩固刚学到的知识并为学习新知识做好准备。

- (1) 如果你还没有安装 Python 3，现在就立刻动手。具体方法请阅读附录 D。
- (2) 启动 Python 3 交互式解释器。再说一次，具体方法请阅读附录 D。它会打印出几行信息和一行 >>>，这是你输入 Python 命令的提示符。
- (3) 随便玩玩解释器。可以用它来计算 `8 * 9`，按下回车来查看结果，Python 应该会打印出 72。
- (4) 输入数字 47 并按下回车，解释器有没有在下一行打印出 47？
- (5) 现在，输入 `print(47)` 并按下回车，解释器有没有在下一行打印出 47？

Python基本元素：数字、字符串和变量

本章会从 Python 最基本的内置数据类型开始学习，这些类型包括：

- 布尔型（表示真假的类型，仅包含 `True` 和 `False` 两种取值）
- 整型（整数，例如 `42`、`100000000`）
- 浮点型（小数，例如 `3.14159`，或用科学计数法表示的数字，例如 `1.0e8`，它表示 `1` 乘以 `10` 的 `8` 次方，也可写作 `100000000.0`）
- 字符串型（字符组成的序列）

这些基本类型就像组成 Python 的原子一样。本章将学习如何单独使用这些基本“原子”，而第 3 章则会介绍如何将这些“原子”组合成更大的“分子”。

每一种类型都有自己的使用规则，计算机对它们的处理方式也不尽相同。此外我们还会学到变量的概念（与实际数据相关联的名字，后面有更详细的介绍）。

本章中的代码虽然都只是小的片段，但它们都是有效的 Python 程序。为了方便快速测试，我们将使用 Python 的交互式解释器，这样在输入代码的同时就可以获得执行结果。尝试在你自己搭建的 Python 环境中运行书中的代码片段，它们会由提示符 `>>>` 标出。第 4 章将开始编写能独立执行的 Python 程序。

2.1 变量、名字和对象

Python 里所有数据——布尔值、整数、浮点数、字符串，甚至大型数据结构、函数以及程序——都是以对象（object）的形式存在的。这使得 Python 语言具有很强的统一性（还有许多其他有用的特性），而这恰恰是许多其他语言所缺少的。

对象就像一个塑料盒子，里面装的是数据（图 2-1）。对象有不同类型，例如布尔型和整

型，类型决定了可以对它进行的操作。现实生活中的“陶器”会暗含一些信息（例如它可能很重，注意不要掉到地上，等等）。类似地，Python 中一个类型为 `int` 的对象会告诉你：可以把它与另一个 `int` 对象相加。



图 2-1：对象就像一个盒子

对象的类型还决定了它装着的数据是允许被修改的变量（可变的）还是不可被修改的常量（不可变的）。你可以把不可变对象想象成一个透明但封闭的盒子：你可以看到里面装的数据，但是无法改变它。类似地，可变对象就像一个开着口的盒子，你不仅可以看到里面的数据，还可以拿出来修改它，但你无法改变这个盒子本身，即你无法改变对象的类型。

Python 是强类型的（strongly typed），你永远无法修改一个已有对象的类型，即使它包含的值是可变的（图 2-2）。

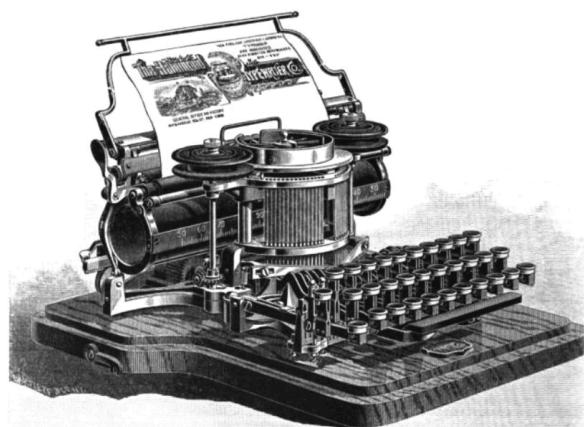


图 2-2：Strong Typing 不是指用力敲打键盘

编程语言允许你定义变量（variable）。所谓变量就是在程序中为了方便地引用内存中的值而为它取的名称。在 Python 中，我们用 `=` 来给一个变量赋值。



我们都在数学课上学过 `=` 代表“等于”，那么为什么计算机语言（包括 Python）要用 `=` 代表赋值操作呢？一种解释是标准键盘没有像左箭头一样的逻辑上能代表赋值操作的键，与其他键相比，`=` 显得相对不那么令人困惑；而在程序中，赋值出现的频率又要远远超过等于，因此把 `=` 分给了赋值操作来使用。

下面这段仅两行的 Python 程序首先将整数 7 赋值给了变量 a，之后又将 a 的值打印了出来：

```
>>> a = 7  
>>> print(a)  
7
```

注意，Python 中的变量有一个非常重要的性质：它仅仅是一个名字。赋值操作并不会实际复制值，它只是为数据对象取个相关的名字。名字是对对象的引用而不是对象本身。你可以把名字想象成贴在盒子上的标签（见图 2-3）。

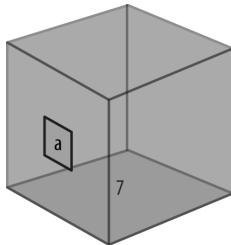


图 2-3：贴在对象上的名字

试着在交互式解释器中执行下面的操作：

- (1) 和之前一样，将 7 赋值给名称 a，这样就成功创建了一个包含整数 7 的对象；
- (2) 打印 a 的值；
- (3) 将 a 赋值给 b，这相当于给刚刚创建的对象又贴上了标签 b；
- (4) 打印 b 的值。

```
>>> a = 7  
>>> print(a)  
7  
>>> b = a  
>>> print(b)  
7
```

在 Python 中，如果想知道一个对象（例如一个变量或者一个字面值）的类型，可以使用语句：type(thing)。试试对不同的字面值（58、99.9、abc）以及不同的变量（a、b）执行 type 操作：

```
>>> type(a)  
<class 'int'>  
>>> type(b)  
<class 'int'>  
>>> type(58)  
<class 'int'>  
>>> type(99.9)  
<class 'float'>  
>>> type('abc')  
<class 'str'>
```

类（class）是对象的定义，第 6 章会详细介绍。在 Python 中，“类”和“类型”一般不加

区分。

变量名只能包含以下字符：

- 小写字母 (a~z)
- 大写字母 (A~Z)
- 数字 (0~9)
- 下划线 (_)

名字不允许以数字开头。此外，Python 中以下划线开头的名字有特殊的含义（第 4 章会解释）。下面是一些合法的名字：

- a
- a1
- a_b_c___95
- _abc
- _1a

下面这些名字则是非法的：

- 1
- 1a
- 1_

最后要注意的是，不要使用下面这些词作为变量名，它们是 Python 保留的关键字：

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

这些关键字以及其他的一些标点符号是用于描述 Python 语法的。在本书中，你会慢慢学到它们各自的作用。

2.2 数字

Python 本身支持整数（比如 5 和 1000000000）以及浮点数（比如 3.1416、14.99 和 1.87e4）。你可以对这些数字进行下表中的计算。

运算符	描述	示例	运算结果
+	加法	5 + 8	13
-	减法	90 - 10	80
*	乘法	4 * 7	28
/	浮点数除法	7 / 2	3.5
//	整数除法	7 // 2	3

(续)

运算符	描述	示例	运算结果
%	模（求余）	7 % 3	1
**	幂	3 ** 4	81

接下来会给你展示一些示例，这些示例体现了 Python 作为一个计算机器的非凡特性。

2.2.1 整数

任何仅含数字的序列在 Python 中都被认为是整数：

```
>>> 5  
5
```

你可以单独使用数字零 (0)：

```
>>> 0  
0
```

但不能把它作为前缀放在其他数字前面：

```
>>> 05  
File "<stdin>", line 1  
    05  
      ^  
SyntaxError: invalid token
```



这是你第一次看见 Python 异常——程序错误。在上面的例子中，解释器抛出了一个警告，提示 05 是一个“非法标识”(invalid token)。2.2.3 节会解释这个警告的意义。你会在本书中见到许多种异常，这是 Python 主要的错误处理机制。

一个数字序列定义了一个正整数。你也可以显式地在前面加上正号 +，这不会使数字发生任何改变：

```
>>> 123  
123  
>>> +123  
123
```

在数字前添加负号 - 可以定义一个负数：

```
>>> -123  
-123
```

你可以像使用计算器一样使用 Python 来进行常规运算。Python 支持的运算参见之前的表格。试试进行加法和减法运算，运算结果和你预期的一样：

```
>>> 5 + 9  
14  
>>> 100 - 7  
93
```

```
>>> 4 - 10  
-6
```

可以连续运算任意个数：

```
>>> 5 + 9 + 3  
17  
>>> 4 + 3 - 2 - 1 + 6  
10
```

格式提示：数字和运算符之间的空格不是强制的，你也可以写成下面这种格式：

```
>>> 5+9 + 3  
17
```

只不过添加空格会使代码看起来更规整更便于阅读。

乘法运算的实现也很直接：

```
>>> 6 * 7  
42  
>>> 7 * 6  
42  
>>> 6 * 7 * 2 * 3  
252
```

除法运算比较有意思，可能与你预期的有些出入，因为 Python 里有两种除法：

- `/` 用来执行浮点除法（十进制小数）
- `//` 用来执行整数除法（整除）

与其他语言不同，在 Python 中即使运算对象是两个整数，使用 `/` 仍会得到浮点型的结果：

```
>>> 9 / 5  
1.8
```

使用整除运算得到的是一个整数，余数会被截去：

```
>>> 9 // 5  
1
```

如果除数为 0，任何一种除法运算都会产生 Python 异常：

```
>>> 5 / 0  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: division by zero  
>>> 7 // 0  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: integer division or modulo by z
```

之前的例子中我们都在使用立即数进行运算，你也可以在运算中将立即数和已赋值过的变量混合使用：

```
>>> a = 95  
>>> a
```

```
95
>>> a - 3
92
```

上面代码中出现了 `a - 3`，但我们并没有将结果赋值给 `a`，因此 `a` 的值并未发生改变：

```
>>> a
95
```

如果你想要改变 `a` 的值，可以这样写：

```
>>> a = a - 3
>>> a
92
```

对于初学者来说，上面这行式子可能很费解，因为小学的数学知识让我们根深蒂固地认为 `=` 代表等于，因此上面的式子显然是不成立的。但在 Python 里并非如此，Python 解释器会首先计算 `=` 右侧的表达式，然后将其结果赋值给左侧的变量。

试着这样理解看看是否有帮助。

- 计算 `a-3`
- 将运算结果保存在一个临时变量中
- 将这个临时变量的值赋值给 `a`:

```
>>> a = 95
>>> temp = a - 3
>>> a = temp
```

因此，当输入：

```
>>> a = a - 3
```

Python 实际上先计算了右侧的减法，暂时记住运算结果，然后将这个结果赋值给了 `=` 左侧的 `a`。这种写法比使用临时变量要更加迅速、简洁。

你还可以进一步将运算过程与赋值过程进行合并，只需将运算符放到 `=` 前面。例如，`a -= 3` 等价于 `a = a - 3`：

```
>>> a = 95
>>> a -= 3
>>> a
92
```

下面的代码等价于执行 `a = a + 8`：

```
>>> a += 8
>>> a
100
```

以此类推，下面的代码等价于 `a = a * 2`：

```
>>> a *= 2
>>> a
200
```

再试试浮点型除法，例如 `a = a / 3`:

```
>>> a /= 3  
>>> a  
66.6666666666667
```

接着将 13 赋值给 `a`，然后试试执行 `a = a // 4`（整数除法）的简化版：

```
>>> a = 13  
>>> a //= 4  
>>> a  
3
```

百分号 % 在 Python 里有多种用途，当它位于两个数字之间时代表求模运算，得到的结果是第一个数除以第二个数的余数：

```
>>> 9 % 5  
4
```

使用下面的方法可以同时得到余数和商：

```
>>> divmod(9,5)  
(1, 4)
```

或者你也可以分别计算：

```
>>> 9 // 5  
1  
>>> 9 % 5  
4
```

上面的代码出现了一些你没见过的新东西：一个叫作 `divmod` 的函数。这个函数接受了两个整数：9 和 5，并返回了一个包含两个元素的结果，我们称这种结构为元组（tuple）。第 3 章会学习元组的使用，而关于函数的内容将在第 4 章进行讲解。

2.2.2 优先级

想一想下面的表达式会产生什么结果？

```
>>> 2 + 3 * 4
```

如果你先进行加法运算 $2 + 3 = 5$ ，然后计算 $5 * 4$ ，最终得到 20。但如果你先进行乘法运算， $3 * 4 = 12$ ，接着 $2 + 12$ ，结果等于 14。与其他编程语言一样，在 Python 里，乘法的优先级要高于加法，因此第二种运算结果是正确的：

```
>>> 2 + 3 * 4  
14
```

如何了解优先级规则？我在附录 F 中为你准备了一张优先级表，但在实际编程中我几乎没有查看过它，因为我们总可以使用括号来保证运算顺序与我们期望的一致：

```
>>> 2 + (3 * 4)  
14
```

这样书写的代码也可让阅读者无需猜测代码的意图，免去了检查优先级表的麻烦。

2.2.3 基数

在 Python 中，整数默认使用十进制数（以 10 为底），除非你在数字前添加前缀，显式地指定使用其他基数（base）。也许你永远都不会在自己的代码中用到其他基数，但你很有可能在其他人编写的 Python 代码里见到它们。

我们大多数人都有 10 根手指 10 根脚趾（我家里倒是有只猫多了几根指头，但我从来没见过它用自己的指头数数）。因此，我们习惯这样计数：0, 1, 2, 3, 4, 5, 6, 7, 8, 9。到了 9 之后，我们用光了所有的数字，于是将数字 1 放到“十位”，并把 0 放到“个位”。因此，10 代表“1 个十加 0 个一”。我们无法用一个字符代表数字“十”。接着是 11, 12, 一直到 19，然后仿照之前的做法，我们将新多出来的 1 加到十位来组成 20（2 个十加 0 个一），以此类推。

基数指的是在必须进位前可以使用的数字的最大数量。以 2 为底（二进制）时，可以使用的数字只有 0 和 1。这里的 0 和十进制的 0 代表的意义相同，1 和十进制的 1 所代表的意义也相同。然而以 2 为底时，1 与 1 相加得到的将是 10（1 个二加 0 个一）。

在 Python 中，除十进制外你还可以使用其他三种进制的数字：

- `0b` 或 `0B` 代表二进制（以 2 为底）
- `0o` 或 `0O` 代表八进制（以 8 为底）
- `0x` 或 `0X` 代表十六进制（以 16 为底）

Python 解释器会打印出它们对应的十进制整数。我们来试试这些不同进制的数。首先是单纯的十进制数字 `10`，代表“1 个十加 0 个一”。

```
>>> 10
10
```

接着，试试二进制（以 2 为底），代表“1（十进制）个二加上 0 个一”：

```
>>> 0b10
2
```

八进制（以 8 为底），代表“1（十进制）个八加上 0 个一”：

```
>>> 0o10
8
```

十六进制（以 16 为底），代表“1（十进制）个 16 加上 0 个一”：

```
>>> 0x10
16
```

可能你会好奇，十六进制用的是哪 16 个“数字”，它们是：0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e 以及 f。因此，`0xa` 代表十进制的 10，`0xf` 代表十进制的 15，`0xf` 加 1 等于 `0x10`（十进制 16）。

为什么要使用 10 以外的基数？因为它们在进行位运算时非常有用。有关位运算以及不同进制之间的转换将在第 7 章进行介绍。

2.2.4 类型转换

我们可以方便地使用 `int()` 函数将其他的 Python 数据类型转换为整型。它会保留传入数据的整数部分并舍去小数部分。

Python 里最简单的数据类型是布尔型，它只有两个可选值：`True` 和 `False`。当转换为整数时，它们分别代表 1 和 0：

```
>>> int(True)
1
>>> int(False)
0
```

当将浮点数转换为整数时，所有小数点后面的部分会被舍去：

```
>>> int(98.6)
98
>>> int(1.0e4)
10000
```

也可以将仅包含数字和正负号的字符串（如果你不知道字符串是什么，不用着急，先往后读，很快就会了解）转换为整数，下面有几个例子：

```
>>> int('99')
99
>>> int('-23')
-23
>>> int('+12')
12
```

将一个整数转换为整数没有太多意义，这既不会产生任何改变也不会造成任何损失：

```
>>> int(12345)
12345
```

如果你试图将一个与数字无关的类型转化为整数，会得到一个异常：

```
>>> int('99 bottles of beer on the wall')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '99 bottles of beer on the wall'
>>> int('')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: ''
```

尽管上面例子中的字符串的确是以有效数字（99）开头的，但它没有就此截止，后面的内容不是纯数字，无法被 `int()` 函数识别，因此抛出异常。



第4章会详细介绍异常。现在，你只需知道异常是Python处理程序错误的方式（不像有些语言不做处理直接造成程序崩溃）。在本书里，我会经常展示各种出现异常的情况，而不是假定程序总是正确运行的，这样你可以更加了解Python是如何处理程序错误的。

`int()` 可以接受浮点数或由数字组成的字符串，但无法接受包含小数点或指数的字符串：

```
>>> int('98.6')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '98.6'
>>> int('1.0e4')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1.0e4'
```

如果混合使用多种不同的数字类型进行计算，Python 会自动地进行类型转换：

```
>>> 4 + 7.0  
11.0
```

与整数或浮点数混合使用时，布尔型的 `False` 会被当作 `0` 或 `0.0`, `True` 会被当作 `1` 或 `1.0`:

```
>>> True + 2  
3  
>>> False + 5.0  
5.0
```

2.2.5 一个int型有多大

在 Python 2 里，一个 `int` 型包含 32 位，可以存储从 -2 147 483 648 到 2 147 483 647 的整数。

一个 long 型会占用更多的空间：64 位，可以存储从 -9 223 372 036 854 775 808 到 9 223 372 036 854 775 807 的整数。

到了 Python 3, `long` 类型已不复存在, 而 `int` 类型变为可以存储任意大小的整数, 甚至超过 64 位。因此, 你可以进行像下面一样计算 (10^{**100} 被赋值给名为 `googol` 的变量, 这是 Google 最初的名字, 但由于其拼写困难而被现在的名字所取代) :

在许多其他编程语言中，进行类似上面的计算会造成整数溢出，这是因为计算中的数字或结果需要的存储空间超过了计算机所提供的（例如 32 位或 64 位）。在程序编写中，溢出会产生许多负面影响。而 Python 在处理超大数计算方面不会产生任何错误，这也是它的一

个加分点。

2.2.6 浮点数

整数全部由数字组成，而浮点数（在 Python 里称为 float）包含非数字的小数点。浮点数与整数很像：你可以使用运算符（+、-、*、/、//、** 和 %）以及 `divmod()` 函数进行计算。

使用 `float()` 函数可以将其他数字类型转换为浮点型。与之前一样，布尔型在计算中等价于 1.0 和 0.0：

```
>>> float(True)
1.0
>>> float(False)
0.0
```

将整数转换为浮点数仅仅需要添加一个小数点：

```
>>> float(98)
98.0
>>> float('99')
99.0
```

此外，也可以将包含有效浮点数（数字、正负号、小数点、指数及指数的前缀 e）的字符串转换为真正的浮点型数字：

```
>>> float('98.6')
98.6
>>> float('-1.5')
-1.5
>>> float('1.0e4')
10000.0
```

2.2.7 数学函数

Python 包含许多常用的数学函数，例如平方根、余弦函数，等等。这些内容被放到了附录 C，在那里我还会讨论 Python 的科学应用。

2.3 字符串

不是程序员的人经常会认为程序员一定都非常擅长数学，因为他们整天和数字打交道。但事实上，大多数程序员在处理字符串上花费的时间要远远超过处理数字的时间。逻辑思维（以及创造力）的重要性要远远超过数学能力。

对 Unicode 的支持使得 Python 3 可以包含世界上任何书面语言以及许多特殊符号。对于 Unicode 的支持是 Python 3 从 Python 2 分离出来的重要原因之一，也正是这一重要特性促使人们转向使用 Python 3。处理 Unicode 编码有时会非常困难，因此我将相关内容分散在了本书的不同地方。而本节只会使用 ASCII 编码的例子。

字符串型是我们学习的第一个 Python 序列类型，它的本质是字符序列。

与其他语言不同的是，Python 字符串是不可变的。你无法对原字符串进行修改，但可以将字符串的一部分复制到新字符串，来达到相同的修改效果。很快你就会学到如何实现。

2.3.1 使用引号创建

将一系列字符包裹在一对单引号或一对双引号中即可创建字符串，就像下面这样：

```
>>> 'Snap'  
'Snap'  
>>> "Crackle"  
'Crackle'
```

交互式解释器输出的字符串永远是用单引号包裹的，但无论使用哪种引号，Python 对字符串的处理方式都是一样的，没有任何区别。

既然如此，为什么要使用两种引号？这么做的好处是可以创建本身就包含引号的字符串，而不用使用转义符。可以在双引号包裹的字符串中使用单引号，或者在单引号包裹的字符串中使用双引号：

```
>>> "'Nay,' said the naysayer."  
"'Nay,' said the naysayer."  
>>> 'The rare double quote in captivity: '.'  
'The rare double quote in captivity: '.'  
>>> 'A "two by four" is actually 1 1/2" × 3 1/2".'  
'A "two by four is" actually 1 1/2" × 3 1/2".'  
>>> "'There's the man that shot my paw!' cried the limping hound."  
"'There's the man that shot my paw!" cried the limping hound."
```

你还可以使用连续三个单引号 '''，或者三个双引号 """ 创建字符串：

```
>>> '''Boom!'''  
'Boom'  
>>> """Eek!"""  
'Eek!'
```

三元引号在创建短字符串时没有什么特殊用处。它多用于创建多行字符串。下面的例子中，创建的字符串引用了 Edward Lear 的经典诗歌：

```
>>> poem = '''There was a Young Lady of Norway,  
... Who casually sat in a doorway;  
... When the door squeezed her flat,  
... She exclaimed, "What of that?"  
... This courageous Young Lady of Norway.'''  
>>>
```

（上面这段代码是在交互式解释器里输入的，第一行的提示符为 >>>，后面行的提示符为 ...，直到再次输入三元引号暗示赋值语句的完结，此时光标跳转到下一行并再次以 >>> 提示输入。）

如果你尝试通过单独的单双引号创建多行字符串，在你完成第一行并按下回车时，Python 会弹出错误提示：

```
>>> poem = 'There was a young lady of Norway,  
    File "<stdin>", line 1  
        poem = 'There was a young lady of Norway,  
                ^  
SyntaxError: EOL while scanning string literal  
>>>
```

在三元引号包裹的字符串中，每行的换行符以及行首或行末的空格都会被保留：

```
>>> poem2 = '''I do not like thee, Doctor Fell.  
...     The reason why, I cannot tell.  
...     But this I know, and know full well:  
...     I do not like thee, Doctor Fell.  
... '''  
>>> print(poem2)  
I do not like thee, Doctor Fell.  
    The reason why, I cannot tell.  
    But this I know, and know full well:  
    I do not like thee, Doctor Fell.
```

```
>>>
```

值得注意的是，`print()` 函数的输出与交互式解释器的自动响应输出存在一些差异：

```
>>> poem2  
'I do not like thee, Doctor Fell.\n    The reason why, I cannot tell.\n    But  
this I know, and know full well:\n    I do not like thee, Doctor Fell.\n'
```

`print()` 会把包裹字符串的引号截去，仅输出其实际内容，易于阅读。它还会自动地在各个输出部分之间添加空格，并在所有输出的最后添加换行符：

```
>>> print(99, 'bottles', 'would be enough.')  
99 bottles would be enough.
```

如果你不希望 `print()` 自动添加空格或换行，随后将学会如何避免它们。

解释器可以打印字符串以及像 `\n` 的转义符，关于转义符的内容你会在 2.3.3 节看到。

最后要指出的是 Python 允许空串的存在，它不包含任何字符且完全合法。你可以使用前面提到的任意一种方法创建一个空串：

```
>>> ''  
''  
>>> ""  
''  
>>> '''''  
''  
>>> ''''''  
''  
>>>
```

为什么用到空字符串？有些时候你想要创建的字符串可能源自另一字符串的内容，这时需要先创建一个空白的模板，也就是一个空字符串。

```
>>> bottles = 99  
>>> base = ''
```

```
>>> base += 'current inventory: '
>>> base += str(bottles)
>>> base
'current inventory: 99'
```

2.3.2 使用 str() 进行类型转换

使用 str() 可以将其他 Python 数据类型转换为字符串：

```
>>> str(98.6)
'98.6'
>>> str(1.0e4)
'10000.0'
>>> str(True)
'True'
```

当你调用 print() 函数或者进行字符串差值（string interpolation）时，Python 内部会自动使用 str() 将非字符串对象转换为字符串。第 7 章会了解到相关内容。

2.3.3 使用 \ 转义

Python 允许你对某些字符进行转义操作，以此来实现一些难以单纯用字符描述的效果。在字符的前面添加反斜线符号 \ 会使该字符的意义发生改变。最常见的转义符是 \n，它代表换行符，便于你在一行内创建多行字符串。

```
>>> palindrome = 'A man,\nA plan,\nA canal:\nPanama.'
>>> print(palindrome)
A man,
A plan,
A canal:
Panama.
```

转义符 \t (tab 制表符) 常用于对齐文本，之后会经常见到：

```
>>> print('\tabc')
    abc
>>> print('a\tbc')
a      bc
>>> print('ab\tc')
ab        c
>>> print('abc\t')
abc
```

(上面例子中，最后一个字符串的末尾包含了一个制表符，当然你无法在打印的结果中看到它。)

有时你可能还会用到 \' 和 \" 来表示单、双引号，尤其当该字符串由相同类型的引号包裹时：

```
>>> testimony = "\"I did nothing!\" he said. \"Not that either! Or the other
thing.\""
>>> print(testimony)
"I did nothing!" he said. "Not that either! Or the other thing."
>>> fact = "The world's largest rubber duck was 54'2\" by 65'7\" by 105'"
```

```
>>> print(fact)
The world's largest rubber duck was 54'2" by 65'7" by 105'
```

如果你需要输出一个反斜线字符，连续输入两个反斜线即可：

```
>>> speech = 'Today we honor our friend, the backslash: \\.'
>>> print(speech)
Today we honor our friend, the backslash: \.
```

2.3.4 使用+拼接

在 Python 中，你可以使用 + 将多个字符串或字符串变量拼接起来，就像下面这样：

```
>>> 'Release the kraken! ' + 'At once!'
'Release the kraken! At once!'
```

也可以直接将一个字面字符串（非字符串变量）放到另一个的后面直接实现拼接：

```
>>> "My word! " "A gentleman caller!"
'My word! A gentleman caller!'
```

进行字符串拼接时，Python 并不会自动为你添加空格，需要显示定义。但当我们调用 `print()` 进行打印时，Python 会在各个参数之间自动添加空格并在结尾添加换行符：

```
>>> a = 'Duck.'
>>> b = a
>>> c = 'Grey Duck!'
>>> a + b + c
'Duck.Duck.Grey Duck!'
>>> print(a, b, c)
Duck. Duck. Grey Duck!
```

2.3.5 使用*复制

使用 * 可以进行字符串复制。试着把下面这几行输入到交互式解释器里，看看结果是什么：

```
>>> start = 'Na ' * 4 + '\n'
>>> middle = 'Hey ' * 3 + '\n'
>>> end = 'Goodbye.'
>>> print(start + start + middle + end)
```

2.3.6 使用[]提取字符

在字符串名后面添加 []，并在括号里指定偏移量可以提取该位置的单个字符。第一个字符（最左侧）的偏移量为 0，下一个是 1，以此类推。最后一个字符（最右侧）的偏移量也可以用 -1 表示，这样就不必从头数到尾。偏移量从右到左紧接着为 -2、-3，以此类推。

```
>>> letters = 'abcdefghijklmnopqrstuvwxyz'
>>> letters[0]
'a'
>>> letters[1]
'b'
```

```
>>> letters[-1]
'z'
>>> letters[-2]
'y'
>>> letters[25]
'z'
>>> letters[5]
'f'
```

如果指定的偏移量超过了字符串的长度（记住，偏移量从 0 开始增加到字符串长度 -1），会得到一个异常提醒：

```
>>> letters[100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

位置索引在其他序列类型（列表和元组）中的使用也是如此，你将在第 3 章见到。

由于字符串是不可变的，因此你无法直接插入字符或改变指定位置的字符。看看当我们试图将 'Henny' 改变为 'Penny' 时会发生什么：

```
>>> name = 'Henny'
>>> name[0] = 'P'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

为了改变字符串，我们需要组合使用一些字符串函数，例如 `replace()`，以及分片操作（很快就会学到）：

```
>>> name = 'Henny'
>>> name.replace('H', 'P')
'Penny'
>>> 'P' + name[1:]
'Penny'
```

2.3.7 使用 [start:end:step] 分片

分片操作（slice）可以从一个字符串中抽取子字符串（字符串的一部分）。我们使用一对方括号、起始偏移量 `start`、终止偏移量 `end` 以及可选的步长 `step` 来定义一个分片。其中一些可以省略。分片得到的子串包含从 `start` 开始到 `end` 之前的全部字符。

- `[:]` 提取从开头到结尾的整个字符串
- `[start:]` 从 `start` 提取到结尾
- `[:end]` 从开头提取到 `end - 1`
- `[start:end]` 从 `start` 提取到 `end - 1`
- `[start:end:step]` 从 `start` 提取到 `end - 1`，每 `step` 个字符提取一个

与之前一样，偏移量从左至右从 0、1 开始，依次增加；从右至左从 -1、-2 开始，依次减小。如果省略 `start`，分片会默认使用偏移量 0（开头）；如果省略 `end`，分片会默认使用偏移量 -1（结尾）。

我们来创建一个由小写字母组成的字符串：

```
>>> letters = 'abcdefghijklmnopqrstuvwxyz'
```

仅仅使用：分片等价于使用 `0 : -1`（也就是提取整个字符串）：

```
>>> letters[:]
'abcdefghijklmnopqrstuvwxyz'
```

下面是一个从偏移量 20 提取到字符串结尾的例子：

```
>>> letters[20:]
'uvwxyz'
```

现在，从偏移量 10 提取到结尾：

```
>>> letters[10:]
'klmnopqrstuvwxyz'
```

下一个例子提取了偏移量从 12 到 14 的字符（Python 的提取操作不包含最后一个偏移量对应的字符）：

```
>>> letters[12:15]
'mno'
```

提取最后三个字符：

```
>>> letters[-3:]
'xyz'
```

下面一个例子提取了从偏移量为 18 的字符到倒数第 4 个字符。注意与上一个例子的区别：当偏移量 `-3` 作为开始位置时，将获得字符 `x`；而当它作为终止位置时，分片实际上会在偏移量 `-4` 处停止，也就是提取到字符 `w`：

```
>>> letters[18:-3]
'stuvw'
```

接下来，试着提取从倒数第 6 个字符到倒数第 3 个字符：

```
>>> letters[-6:-2]
'uvwxyz'
```

如果你需要的步长不是默认的 1，可以在第二个冒号后面进行指定，就像下面几个例子所示。

从开头提取到结尾，步长设为 7：

```
>>> letters[::-7]
'ahov'
```

从偏移量 4 提取到偏移量 19，步长设为 3：

```
>>> letters[4:20:3]
'ehknqt'
```

从偏移量 19 提取到结尾，步长设为 4：

```
>>> letters[19::4]
'tx'
```

从开头提取到偏移量 20，步长设为 5：

```
>>> letters[:21:5]
'afkpu'
```

(记住，分片中 end 的偏移量需要比实际提取的最后一个字符的偏移量多 1。)

是不是非常方便？但这还没有完。如果指定的步长为负数，机智的 Python 还会从右到左反向进行提取操作。下面这个例子便从右到左以步长为 1 进行提取：

```
>>> letters[-1::-1]
'zyxwvutsrqponmlkjihgfedcba'
```

事实上，你可以将上面的例子简化为下面这种形式，结果完全一致：

```
>>> letters[::-1]
'zyxwvutsrqponmlkjihgfedcba'
```

分片操作对于无效偏移量的容忍程度要远大于单字符提取操作。在分片中，小于起始位置的偏移量会被当作 0，大于终止位置的偏移量会被当作 -1，就像接下来几个例子展示的一样。

提取倒数 50 个字符：

```
>>> letters[-50:]
'abcdefghijklmnopqrstuvwxyz'
```

提取从倒数第 51 到倒数第 50 个字符：

```
>>> letters[-51:-50]
''
```

从开头提取到偏移量为 69 的字符：

```
>>> letters[:70]
'abcdefghijklmnopqrstuvwxyz'
```

从偏移量为 70 的字符提取到偏移量为 71 的字符：

```
>>> letters[70:71]
''
```

2.3.8 使用 len() 获得长度

到目前为止，我们已经学会了使用许多特殊的标点符号（例如 +）对字符串进行相应操作。但标点符号只有有限的几种。从现在开始，我们将学习使用 Python 的内置函数。所谓函数指的是可以执行某些特定操作的有名字的代码。

`len()` 函数可用于计算字符串包含的字符数：

```
>>> len(letters)
26
>>> empty = ""
```

```
>>> len(empty)
0
```

也可以对其他的序列类型使用 `len()`，这些内容都被放在第 3 章里讲解。

2.3.9 使用 `split()` 分割

与广义函数 `len()` 不同，有些函数只适用于字符串类型。为了调用字符串函数，你需要输入字符串的名称、一个点号，接着是需要调用的函数名，以及需要传入的参数：`string.function(arguments)`。4.7 节会学习更多关于函数的内容。

使用内置的字符串函数 `split()` 可以基于分隔符将字符串分割成由若干子串组成的列表。所谓列表（list）是由一系列值组成的序列，值与值之间由逗号隔开，整个列表被方括号所包裹。

```
>>> todos = 'get gloves,get mask,give cat vitamins,call ambulance'
>>> todos.split(',')
['get gloves', 'get mask', 'give cat vitamins', 'call ambulance']
```

上面例子中，字符串名为 `todos`，函数名为 `split()`，传入的参数为单一的分隔符 `,`。如果不指定分隔符，那么 `split()` 将默认使用空白字符——换行符、空格、制表符。

```
>>> todos.split()
['get', 'gloves,get', 'mask,give', 'cat', 'vitamins,call', 'ambulance']
```

即使不传入参数，调用 `split()` 函数时仍需要带着括号，这样 Python 才能知道你想要进行函数调用。

2.3.10 使用 `join()` 合并

可能你已经猜到了，`join()` 函数与 `split()` 函数正好相反：它将包含若干子串的列表分解，并将这些子串合成一个完整的大的字符串。`join()` 的调用顺序看起来有点别扭，与 `split()` 相反，你需要首先指定粘合用的字符串，然后再指定需要合并的列表：`string.join(list)`。因此，为了将列表 `lines` 中的多个子串合并成完整的字符串，我们应该使用语句：`'\n'.join(lines)`。下面的例子将列表中的名字通过逗号及空格粘合在一起：

```
>>> crypto_list = ['Yeti', 'Bigfoot', 'Loch Ness Monster']
>>> crypto_string = ', '.join(crypto_list)
>>> print('Found and signing book deals:', crypto_string)
Found and signing book deals: Yeti, Bigfoot, Loch Ness Monster
```

2.3.11 熟悉字符串

Python 拥有非常多的字符串函数。这一节将探索其中最常用的一些。我们的测试对象是下面的字符串，它源自纽卡斯尔伯爵 Margaret Cavendish 的不朽名篇 *What Is Liquid?*：

```
>>> poem = '''All that doth flow we cannot liquid name
Or else would fire and water be the same;
But that is liquid which is moist and wet
Fire that property can never get.
```

```
Then 'tis not cold that doth the fire put out  
But 'tis the wet that makes it die, no doubt. . .
```

先做个小热身，试着提取开头的 13 个字符（偏移量为 0 到 12）：

```
>>> poem[:13]  
'All that doth'
```

这首诗有多少个字符呢？（计入空格和换行符。）

```
>>> len(poem)  
250
```

这首诗是不是以 All 开头呢？

```
>>> poem.startswith('All')  
True
```

它是否以 That's all, folks!？结尾？

```
>>> poem.endswith('That\'s all, folks!')  
False
```

接下来，查一查诗中第一次出现单词 the 的位置（偏移量）：

```
>>> word = 'the'  
>>> poem.find(word)  
73
```

以及最后一次出现 the 的偏移量：

```
>>> poem.rfind(word)  
214
```

the 在这首诗中出现了多少次？

```
>>> poem.count(word)  
3
```

诗中出现的所有字符都是字母或数字吗？

```
>>> poem.isalnum()  
False
```

并非如此，诗中还包括标点符号。

2.3.12 大小写与对齐方式

在这一节，我们将介绍一些不那么常用的字符串函数。我们的测试字符串如下所示：

```
>>> setup = 'a duck goes into a bar...'
```

将字符串收尾的 . 都删除掉：

```
>>> setup.strip('.')  
'a duck goes into a bar'
```