



完整版请联系QQ: 859570073 (豆豆电子书)

本人因为职务便利, 不管是什么类型的图书
只要不是很新, 基本都能找到电子版

Python游戏 编程入门

MORE PYTHON PROGRAMMING
FOR THE ABSOLUTE BEGINNER

[美] Jonathan S. Harbour 著 李强 译



人民邮电出版社
POSTS & TELECOM PRESS

游戏开发权威专家力作 掌握Python游戏编程佳选

学习一种编程语言，还有比开发游戏更好的方法吗？本书为读者提供了充分的实践和练习，并且关注Python编程中的高级话题，这些全部通过游戏示例和项目来介绍，而这已经证明是一种高效而有趣的学习方法。本书介绍了数据结构、文件处理、异常、面向对象编程、GUI编程、多媒体编程、命名空间和程序规划。本书将使读者掌握Python语言的深层知识。

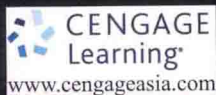
本书包括以下内容：

- 使用Pygame创建2D图形；
- 从数据文件读取数据，以及将数据写入到文件；
- 创建和操作精灵；
- 处理面向对象编程技术；
- 掌握冲突检测技术；
- 创建数组、列表和元组；
- 实现帧动画编程。

Jonathan Harbour 拥有近30年游戏开发经验。他编写了20多本（包括改编）书，涉及大多数主流的编程语言，如C++、C#、VB、Java和Python。他还熟悉Xbox、Xbox 360、Windows Phone、Android、Game Boy Advance 和Pocket PC等硬件。可以通过www.jharbour.com联系他。

本书具有以下特色：

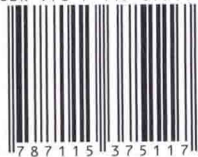
- 以开发游戏为学习方法；
- 利用已经学到的知识，逐渐增加编程项目的挑战性，逐步构建牢固的基础知识；
- 通过每章最后的挑战练习，鼓励读者测试自己的技能并应用自己的知识；
- Web站点（www.jharbour.com）包含了所有的源代码和其他资料。



分类建议：计算机 / 软件开发 / 游戏开发
人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-37511-7



9 787115 375117 >

ISBN 978-7-115-37511-7

定价:49.00 元



Python游戏

编程入门

[美] Jonathan S. Harbour 著 李强 译

人民邮电出版社
北京

图书在版编目 (C I P) 数据

Python游戏编程入门 / (美) 哈伯 (Harbour, J. S.)
著 ; 李强译. — 北京 : 人民邮电出版社, 2015. 1
ISBN 978-7-115-37511-7

I. ①P… II. ①哈… ②李… III. ①游戏程序—程序设计 IV. ①TP311.1

中国版本图书馆CIP数据核字 (2014) 第272294号

版权声明

More Python Programming for the Absolute Beginner

Jonathan S. Harbour

Copyright © 2012 Course Technology, a part of Cengage Learning.

Original edition published by Cengage Learning. All Rights reserved.

本书原版由圣智学习出版公司出版。版权所有, 盗印必究。

Posts & Telecom Press is authorized by Cengage Learning to publish and distribute exclusively this simplified Chinese edition. This edition is authorized for sale in the People's Republic of China only (excluding Hong Kong, Macao SAR and Taiwan). Unauthorized export of this edition is a violation of the Copyright Act. No part of this publication may be reproduced or distributed by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

本书中文简体字翻译版由圣智学习出版公司授权人民邮电出版社独家出版发行。此版本仅限在中华人民共和国境内 (不包括中国香港、澳门特别行政区及中国台湾) 销售。未经授权的本书出口将被视为违反版权法的行为。未经出版者预先书面许可, 不得以任何方式复制或发行本书的任何部分。

978-7-115-37511-7

Cengage Learning Asia Pte. Ltd.

151 Lorong Chuan, #02-08 New Tech Park, Singapore 556741

本书封面贴有 Cengage Learning 防伪标签, 无标签者不得销售。

-
- ◆ 著 [美] Jonathan S. Harbour
译 李 强
责任编辑 陈冀康
责任印制 张佳莹 彭志环
- ◆ 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>
三河市海波印务有限公司印刷
- ◆ 开本: 800×1000 1/16
印张: 19.25
字数: 360 千字 2015 年 1 月第 1 版
印数: 1—3 500 册 2015 年 1 月河北第 1 次印刷
著作权合同登记号 图字: 01-2014-6322 号
-

定价: 49.00 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 是一种解释型、面向对象、动态数据类型的高级程序设计语言，在游戏开发领域，Python 也得到越来越广泛的应用，并由此受到重视。

本书教授用 Python 开发精彩游戏所需的最为重要的概念。本书不只是介绍游戏编程概念的相关内容，还深入到复杂的主题。全书共 14 章，依次介绍类、Pygame、文件 I/O、用户输入、数学和图形编程、位图图形、精灵动画和冲突检测、数组、计时和声音、编程逻辑、三角函数、随机地形、角色扮演游戏等重要的知识和概念。每章通过一个示例游戏来展示这些知识和工具的实际应用。学完本书，读者将掌握使用这些概念来构建较为复杂的游戏，甚至进行较为复杂的 Python 编程。

本书内容浅显易懂，示例轻松活泼，适合 Python 初学者阅读，尤其适合想要掌握 Python 游戏编程的读者学习参考。

目录

第 1 章 使用类的 Python	1	3.2.3 处理异常	39
1.1 了解 Geometry 程序	1	3.2.4 Mad Lib 游戏	39
1.2 初识 Python	2	3.3 文件输入/输出	42
1.2.1 Python 工具	3	3.3.1 操作文本	42
1.2.2 Python 语言	7	3.3.2 操作二进制文件	44
1.3 Python 中的对象	7	3.4 Trivia 游戏	46
1.3.1 在面向对象之前是什么	8	3.4.1 用 Pygame 打印文本	47
1.3.2 接下来是什么	11	3.4.2 Trivia 类	47
1.3.3 OOP: Python 的方式	14	3.4.3 加载 Trivia 数据	48
1.3.4 单继承	16	3.4.4 显示问题和答案	49
1.3.5 多继承	17	3.4.5 响应用户输入	51
1.4 小结	19	3.4.6 继续下一个问题	52
第 2 章 初识 Pygame: Pie 游戏	21	3.4.7 主代码	52
2.1 了解 Pie 游戏	21	3.5 小结	54
2.2 使用 Pygame	22	第 4 章 用户输入: Bomb Catcher 游戏	55
2.2.1 打印文本	23	4.1 认识 Bomb Catcher 游戏	55
2.2.2 循环	24	4.2 Pygame 事件	56
2.2.3 绘制圆	25	4.2.1 实时事件循环	57
2.2.4 绘制矩形	26	4.2.2 键盘事件	58
2.2.5 绘制线条	28	4.2.3 鼠标事件	59
2.2.6 绘制弧形	29	4.3 设备轮询	59
2.3 Pie 游戏	30	4.3.1 轮询键盘	59
2.4 小结	33	4.3.2 轮询鼠标	62
第 3 章 I/O、数据和字体: Trivia 游戏	34	4.4 Bomb Catcher 游戏	64
3.1 了解 Trivia 游戏	34	4.5 小结	67
3.2 Python 数据类型	35	第 5 章 Math 和 Graphics: Analog Clock 示例程序	69
3.2.1 关于打印的更多知识	36	5.1 Analog Clock 示例程序简介	69
3.2.2 获取用户输入	38		

5.2 基本三角函数	70	7.3.2 冲突	112
5.2.1 圆理论	70	7.3.3 源代码	113
5.2.2 遍历圆周	74	7.4 小结	116
5.2.3 圆示例	76	第 8 章 精灵冲突: Zombie Mob	
5.3 Analog Clock 示例程序	78	游戏	117
5.3.1 获取时间	78	8.1 Zombie Mob 游戏简介	117
5.3.2 绘制时钟	79	8.2 冲突检测技术	118
5.4 小结	85	8.2.1 两个精灵之间的矩形	
第 6 章 位图图形: Orbiting Spaceship		检测	118
示例程序	87	8.2.2 两个精灵之间的圆	
6.1 认识 Orbiting Spaceship 示例		检测	119
程序	87	8.2.3 两个精灵之间的像素精确	
6.2 使用位图	88	遮罩检测	120
6.2.1 加载位图	88	8.2.4 精灵和组之间的矩形	
6.2.2 绘制背景	89	冲突	120
6.2.3 绘制行星	91	8.2.5 两个组之间的矩形冲突	
6.2.4 绘制航空飞船	91	检测	121
6.3 环绕行星轨道	94	8.3 Zombie Mob 游戏	121
6.4 小结	100	8.3.1 创建自己的模块	122
第 7 章 用精灵实现动画: Escape the		8.3.2 高级定向动画	125
Dragon 游戏	101	8.3.3 与僵尸冲突	128
7.1 认识 Escape the Dragon 游戏	101	8.3.4 获得生命值	129
7.2 使用 Pygame 精灵	102	8.3.5 游戏源代码	131
7.2.1 定制动画	102	8.4 小结	136
7.2.2 加载精灵序列图	104	第 9 章 数组、列表和元组: Block	
7.2.3 更改帧	104	Breaker 游戏	137
7.2.4 绘制一帧	105	9.1 Block Breaker 游戏简介	137
7.2.5 精灵组	106	9.2 数组和列表	137
7.2.6 MySprite 类	107	9.2.1 有一个维度的列表	138
7.2.7 测试精灵动画	109	9.2.2 创建栈式列表	140
7.3 Escape the Dragon 游戏	110	9.2.3 创建队列式列表	141
7.3.1 跳跃	111		

9.2.4	更多维度的列表	141	11.2.1	画出蛇来—— SnakeSegment 类	172
9.3	元组	145	11.2.2	增长蛇——Snake 类	172
9.3.1	打包元组	145	11.2.3	蛇吃食物—— Food 类	173
9.3.2	解包元组	145	11.2.4	初始化游戏	174
9.3.3	搜索元素	146	11.2.5	主程序	176
9.3.4	计数元素	146	11.2.6	通过吃食物而长长	178
9.3.5	作为常量数组的 元组	147	11.2.7	咬到自己是不 明智的	179
9.4	Block Breaker 游戏	148	11.2.8	跌落世界之外	180
9.4.1	Block Breaker 关卡	148	11.3	教蛇学会自己移动	180
9.4.2	加载和修改关卡	151	11.3.1	自动移动	181
9.4.3	初始化游戏	152	11.3.2	获得当前方向	182
9.4.4	移动挡板	153	11.3.3	朝着食物移动	183
9.4.5	移动球	154	11.3.4	其他代码修改	183
9.4.6	撞击挡板	155	11.4	小结	184
9.4.7	撞击砖块	155	第 12 章	三角函数: Tank Battle 游戏	185
9.4.8	主代码	156	12.1	Tank Battle 游戏简介	185
9.4.9	更新 MySprite	157	12.2	角速率	186
9.5	小结	159	12.2.1	计算角速率	186
第 10 章	计时和声音: Oil Spill 游戏	160	12.2.2	Pygame 笨拙的 旋转	187
10.1	Oil Spill 游戏简介	160	12.2.3	以任意角度前后移动 坦克	188
10.2	声音	161	12.2.4	改进角度折返	190
10.2.1	加载音频文件	161	12.3	构建 Tank Battle 游戏	190
10.2.2	播放音频剪辑	162	12.3.1	坦克	190
10.3	构建 Oil Spill 游戏	162	12.3.2	子弹	194
10.3.1	游戏逻辑	162	12.3.3	主程序代码	195
10.3.2	源代码	165	12.4	小结	201
10.4	小结	169			
第 11 章	编程逻辑: Snake 游戏	170			
11.1	Snake 游戏简介	170			
11.2	开发 Snake 游戏	171			

第 13 章 随机地形: Artillery Gunner 游戏.....202

13.1 Artillery Gunner 游戏简介202

13.2 创建地形203

13.2.1 定义高度地图203

13.2.2 平滑地形208

13.2.3 定位栅格点210

13.3 大炮212

13.3.1 放置大炮212

13.3.2 绘制炮塔213

13.3.3 发射大炮213

13.3.4 让炮弹再飞一会儿214

13.3.5 计算机开火215

13.3.6 为击中计分215

13.4 完整的游戏217

13.5 小结224

第 14 章 更多内容: Dungeon 角色扮演游戏.....226

14.1 Dungeon 游戏简介.....226

14.2 回顾经典的 Dungeon RPG.....227

14.2.1 Rogue.....228

14.2.2 NetHack.....229

14.2.3 AngBand.....230

14.2.4 Sword of Fargoal.....232

14.2.5 Kingdom of Kroz.....232

14.2.6 ZZT.....232

14.3 创建一个地下城关卡.....234

14.3.1 理解 ASCII 字符.....234

14.3.2 模拟文本控制台 显示.....238

14.3.3 生成随机房间.....241

14.3.4 生成随机的通道.....246

14.4 填充地下城.....252

14.4.1 添加入口和出口.....252

14.4.2 添加金子.....254

14.4.3 添加武器、盔甲和 生命值.....255

14.4.4 添加怪兽.....257

14.4.5 完整的 Dungeon 类.....257

14.4.6 添加玩家的角色.....262

14.5 高级游戏逻辑.....266

14.5.1 捡拾物品.....266

14.5.2 与怪兽战斗.....270

14.5.3 移动怪兽.....273

14.5.4 可见性范围.....275

14.5.5 退出关卡.....277

14.5.6 结束游戏逻辑.....277

14.6 小结.....281

附录 A 安装 Python 和 Pygame.....283

A.1 安装 Python.....283

A.2 安装 Pygame.....286

附录 B Pygame 按键代码.....288

第 1 章

使用类的 Python

本章是 Python 的一个快速介绍，接触到基本的面向对象编程知识，并帮助读者感受 Python 语言看上去略有些奇怪的语法。Python 既是一种工具，也是一种语言。

根据 Python 标准，它包括了代码的语法和格式。工具是 Python 安装时所带的一个软件包，其中包括一个编辑器。这些内容对于第 1 章来说有点厚重。如果这是你第一次接触 Python 语言，不要被本章的学习步伐给落下，我们马上会介绍一些重要的细节，但是，本书不会随着后面的每一章而变得越来越难。在本章中，你将学到：

- ◎ 如何把 Python 代码输入到 IDLE 编辑器中；
- ◎ 使用 Python 自带的工具；
- ◎ 回顾 Python 语言的功能；
- ◎ 追溯编程语言的历史；
- ◎ 关注最新的编程方法学；
- ◎ 多态和继承；
- ◎ 使用多继承编写一个示例。

1.1 了解 Geometry 程序

本章带你快速地了解 Python 的面向对象编程功能，并且从头开始以“OOP 的方式”加快你使用 Python 编程的速度。如果你不能一次性地了解本章中所涵盖的所有内容，也不要担心，因为我们从现在开始将会在每一章中回顾所有这些概念，同时通过创建游戏来学习（不，是精通）Python 语言。第一个示例如图 1.1 所示。



图 1.1 Geometry 演示程序可以快速地了解 Python 的面向对象编程功能

1.2 初识 Python

Python 既是一个软件工具包，也是一种语言。Python 软件包包含了一个名为 IDLE 的编辑器。Idle 是一个人的名字，而不是集成开发（integrated development...）的缩写，尽管 IDLE 看上去有点像是缩写。这个人的名字是 Eric Idle，他是 Monty Python 的创始成员之一，而 Monty Python 则是 Python 语言的名称的由来，Python 是向 British TV 的一部电视剧致敬。Python 语言也很奇怪，因此，它这个名字是很合适的。当然，它是以一种可爱的方式来表现出奇怪。如果你真的是初次接触 Python，并且没有阅读过 Michael Dawson 的入门图书（Python Programming for the Absolute Beginner），那么，你可能会对 Python 不同于其他的编程语言感到惊喜。这使得学习 Python 有了一些挑战，但尽管如此也是值得的。



如果想要下载供你的操作系统使用的最新的 Python 包, 请访问 <http://www.python.org>.

1.2.1 Python 工具

正如人们所预期的那样, Python 包内含 Python 解释器和运行时库, 但是, 它还包含了几个有用的工具, 我们现在来介绍一下这些工具。

Module Docs (Pydoc)

针对不同操作系统的 Python 包是不同的, 但大多数常用的包都包含 Python 的文档工具 Pydoc。这个工具是一个较小的搜索工具包, 它可以在 Python 文档中查找项目, 以列表形式给出搜索结果, 然后用默认的 Web 浏览器访问其中任何一项。在 Python 程序组中, 这款工具也叫作 Module Docs, 如图 1.2 所示。

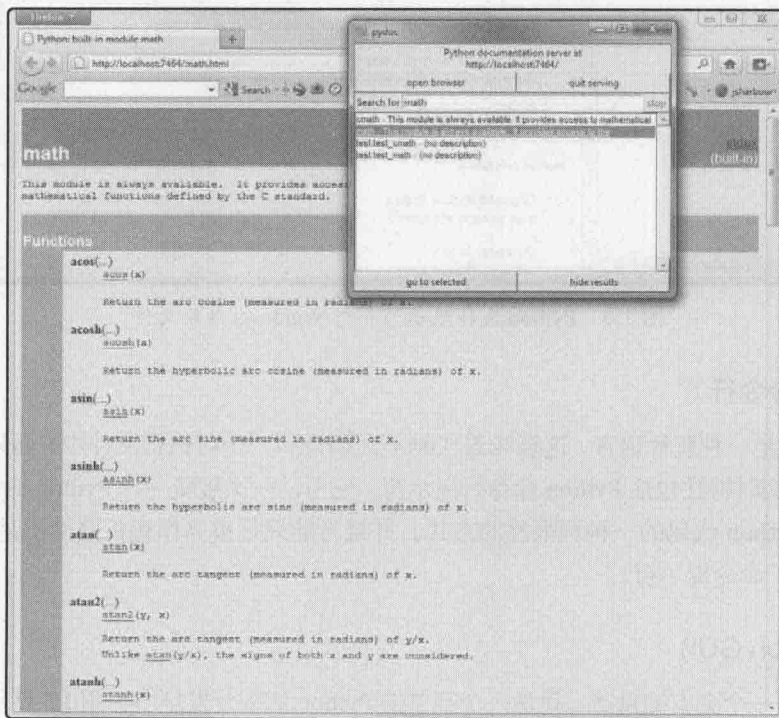


图 1.2 Pydoc 在默认的 Web 浏览器中显示帮助页面

Python Manuals(Pyhelp)

在程序菜单下，还有一个可选的项 Python Manuals，它可以以 Windows 帮助文件的形式来显示 Python 文档，如图 1.3 所示。这个版本的文档是可搜索的，但是，这可能不是找到想要的信息的一种快速的方式。

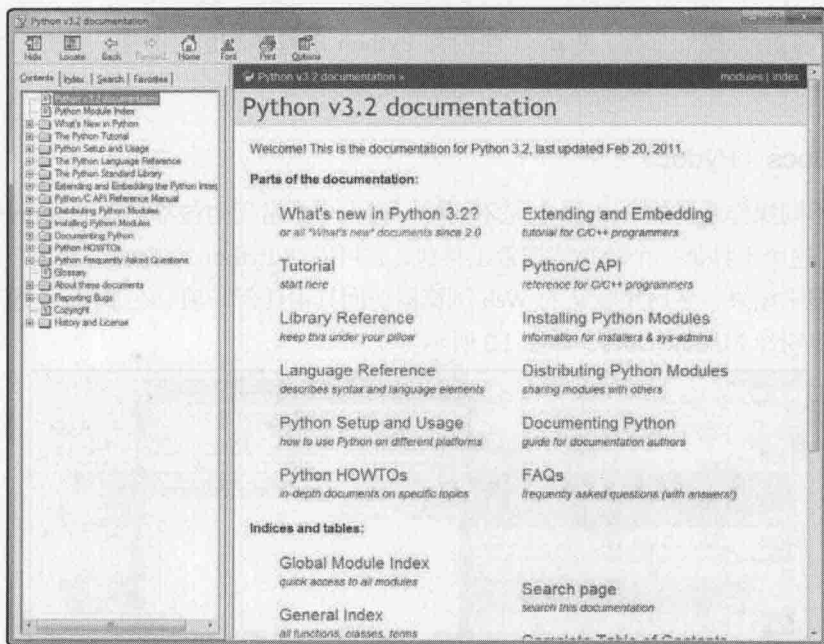


图 1.3 Python 文档显示为一个 Windows 帮助文件

Python（命令行）

Python 是一种解释语言，这意味着代码不会编译到一个可执行文件中，而只是实时地解释。这一实时特性包括 Python 命令行提示符，它可以一次接受一行 Python 命令。当然，这是编写 Python 代码的一种局限性的方式，并且可能只是被当作解析器而不是“代码”。图 1.4 展示了命令提示符。

IDLE(Python GUI)

IDLE 是一个文本编辑器，也是一个简单的 Python 编程开发环境。图 1.5 展示了 IDLE，其中显示了针对当前正在输入的代码的一个弹出式帮助菜单。在这个例子中，它显示了

print()函数的语法。但是，这不是 IDLE 编辑器，这只是 IDLE 命令提示符。

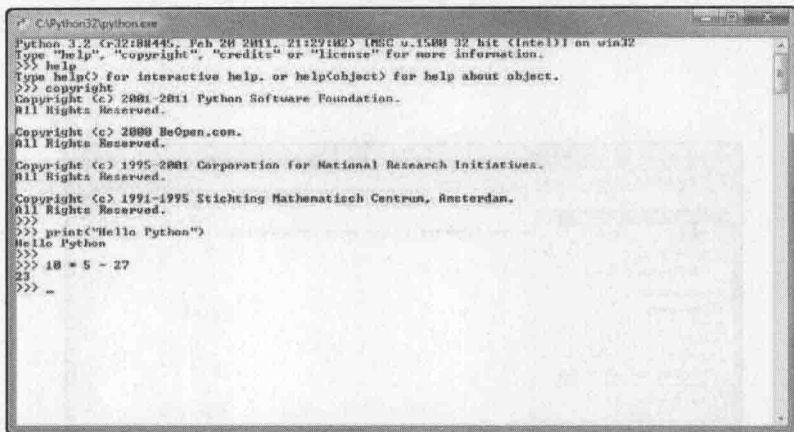


图 1.4 Python 命令提示符将解释命令

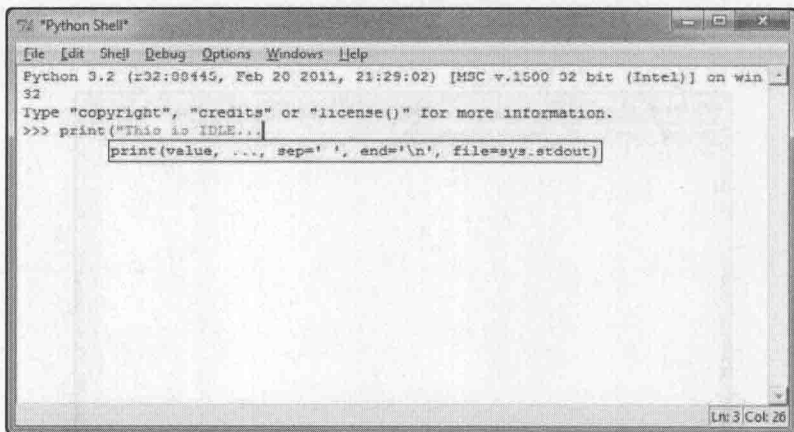


图 1.5 IDLE 是 Python 所包含的一个文本编辑器

是的，我们可以运行如图 1.4 所示的一个独立的提示符，或者使用 IDLE 内建的一个提示符。要开始真正地编辑代码，使用 File 菜单并且选择 New Window，如图 1.6 所示。这会创建一个新的源代码编辑器窗口，如图 1.7 所示。

在做任何其他事情之前，首先要将新的源代码保存为文件。做了这件事情之后，才能让 Python 运行（或解释）你的代码。使用 File 菜单来保存文件，然后打开 Run 菜单，并且选择 Run Module。也可以按下 F5 键来运行代码。现在，当你运行程序的时候，发生了

一件有趣的事情。输出在最初弹出的主 IDLE 窗口中出现了, 如图 1.8 所示。当编辑文件的时候, 应该让提示符窗口 (也叫作 Python Shell) 保持打开状态, 因为它是运行程序的主输出窗口, 即便在使用 Pygame (下一章将详细介绍) 这样的图形化窗口的时候, 也是如此。

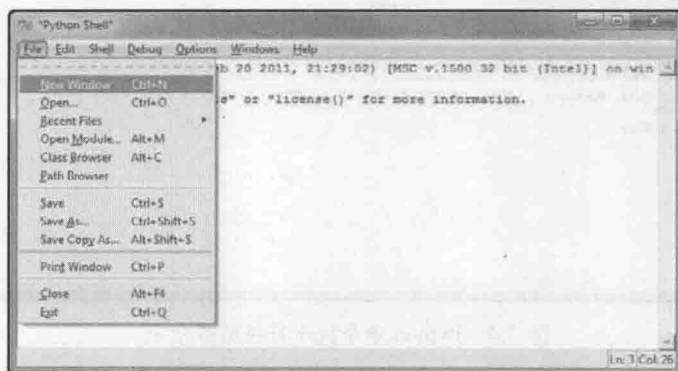


图 1.6 使用 IDLE 创建一个新的源代码编辑器窗口

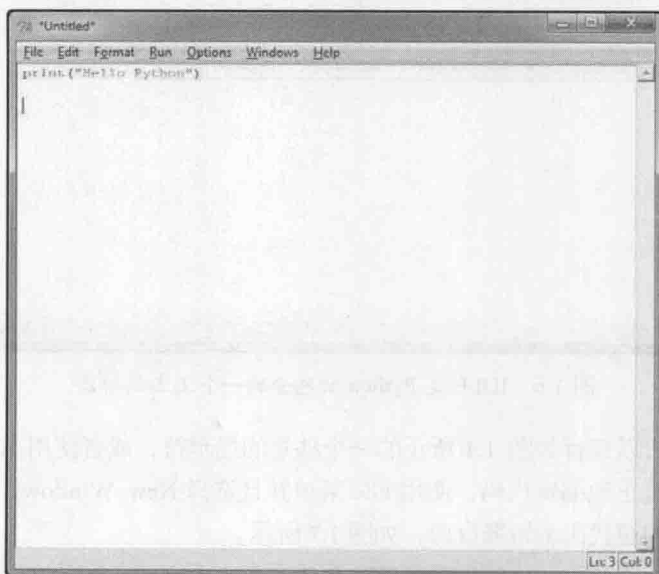


图 1.7 在新的源代码窗口中输入代码



图 1.8 在新的源代码窗口中输入代码

1.2.2 Python 语言

Python 语言是一种看上去很奇怪的语言，似乎是由一个喜欢晦涩的 Isles 式幽默的旅行戏剧团设计的语言，而美国人认为那种幽默令人讨厌且无法理解。当然，这只是一种充满情绪化的、散布在大学课堂中的观点，因此，建议你不要全盘接受这种观点。Python 功能强大，而且用途广泛，一旦你熟悉了它，会对它的功能感到吃惊。

要将 Python 和诸如 C++ 这样的语言进行比较，真的是非常困难的，因为 Python 中没有开始括号和结束括号，也没有可以识别的函数名。Python 类的构造函数不是很好看，哦，我不想立刻吓着你，搞得你要回过头去使用 BASIC。倒不是说 BASIC 有什么错。我恰好特别喜欢一款叫做 QB64 (www.qb64.net) 的工具，另一本名为 Video Game Programming for Kids 的书中将用到它。IDLE 是 Python 包所包含的一款非常有用的文本编辑器，并且，我们将在本书中使用它。



在这里：<http://docs.python.org/reference>，有一个针对 Python 的在线参考手册。

1.3 Python 中的对象

Python 是面向对象编程语言，这意味着，它至少支持一些面向对象编程概念。现在，

我们将花一些时间来介绍这些概念，因为这是一种编写代码的高效方式。面向对象编程（OOP）是一种方法学，也就是做事情的方式。在计算机科学中，有几种较大的、“伞状的”方法学，也就是说，定义了编程语言的功能的方法学。要让我们的技能成为可以传播的，方法学对于这个产业来说很重要。如果每个公司使用他们自己的方法学，那么，为该公司工作的过程中所获取的技能，对于另一个不同的组织来说将会是无用的。软件工程也是一个充满挑战的领域，并且培训的成本很高，因此，对于这个领域相关的每个人（经验丰富的开发者、老板以及教授概念的讲师）来说，方法学都是有益的。

1.3.1 在面向对象之前是什么

天生的好奇心，这是有天分的程序员的共同特点，如果你也有的话，那么，你肯定会问，在面向对象泛型之前，人们使用的是哪一种编程类型呢。让我们来了解一下这个主题，在我们还没有真正开始使用 Python 之前，先说明一下为什么这个问题如此重要。在编程方法学方面，我们先搞清楚起源在哪里，才能够理解今天位于何处。

结构化编程

在 OOP 之前，人们所采用的方法学叫作过程化编程（procedural programming）或结构化编程（structured programming），这意味着，在这种情况下使用的是过程和结构。过程通常叫作函数，并且，我们如今仍然在使用函数。是的，甚至在 OOP 程序中，仍然有独立的函数，如 `main()`。包含在一个对象中的函数，叫作方法，并且当作为对象的一部分讨论的时候，使用方法这个术语而不是函数。但是，在对象之外，函数仍然存在，并且这是从之前的“时代”（方法学）沿用而来的。

结构是复杂的用户定义类型（user-defined types, UDT），它可以将很多的变量包含在一起。最流行的结构化语言是 C。然而，结构化编程是一种历史悠久而且颇为成功的方法学，一直延续至今。结构化运动的时间是从 20 世纪 80 年代到 20 世纪 90 年代，当然，这个时间和其他的方法学的发展有一些重叠。在电子产业中，很多软件开发工具包（SDK）仍然按照结构化的方式来开发，提供了函数库来控制一个电子设备（例如，显卡或嵌入式系统）。可以说，C 语言的开发（大概是在 20 世纪 70 年代）是以结构化编程为主要方式而进行的。C 语言用来创建 UNIX 操作系统。

如下是 Python 的结构化程序的一个快速示例。

```
# Structured program
```

```
# Function definition
def PrintName(name):
    print("The name is " + name + ".")
```

```
# Start of program
PrintName("Jane Doe")
```

这段程序产生如下的输出。

```
The name is Jane Doe.
```



Python 中的注释行都是以#字符开头的。

函数的定义以 `def` 开头，后面跟着函数名、参数和一个冒号。Python 中没有代码块符号，如 C++ 中的开始花括号 (`{`) 和结束花括号 (`}`)。在 Python 中，函数的结尾是未定义的，假设函数在下一个未缩进的行之前结束。让我们做一点试验，来测试 Python 的行为。如下还是我们的示例，不带任何的注释行。你认为它会输出什么？

```
def PrintName(name):
    print("The name is " + name + ".")
print("END")
PrintName("Jane Doe")
```

输出是：

```
END
The name is Jane Doe.
```

大多数的 Python 初学者会对此感到惊奇。这里所发生的事情是，`print("END")` 行向左缩进，因此，它变成了程序的第一行，后面跟着第二行，即 `PrintName("Jane Doe")`。函数定义不被看作是主程序的部分，并且，只有当调用该函数的时候才会运行。如果我们像下面这样，把函数定义放在主程序的下方，会发生什么情况？

```
PrintName("Jane Doe")

def PrintName(name):
    print("The name is " + name + ".")
```

这段代码实际上会产生语法错误，因为无法找到 `PrintName` 函数。这就告诉我们，在调用函数之前 Python 必须先解析它。换句话说，函数定义必须位于函数调用的“上方”。

```
Traceback (most recent call last):
```



```
File "FunctionDemo.py", line 4, in <module>
    PrintName("Jane Doe")
NameError: name 'PrintName' is not defined
```



当使用 IDLE 保存源代码的时候，确保要包含扩展名.PY，因为 IDLE 不会自动添加扩展名。

顺序式编程

结构化编程是从早期的顺序式编程方法学发展而来的。这不是正式的教科书的说法，但却是更富有描述性的一种说法。顺序式程序要求在每行代码之前都要有行号。尽管跳转到程序的其他行也是可能的（使用 `goto` 或 `gosub` 命令），并且这是结构化编程的一个早期的发展方向，但是，顺序式程序倾向于陷入某种程度的复杂性，使得代码变得难以识别或无法修改。这个时候所导致的问题，称为“意大利面条式代码”，这是由于程序似乎要去向每个方向的“流”而导致的。两种最常用的顺序式语言是 BASIC 和 FORTRAN，并且这些语言的全盛期是 20 世纪 70 年代到 20 世纪 80 年代。随着开发者对于维护“意大利面条式代码”感到厌烦，人们迫切地需要进行范型迁移。随着诸如 Pascal 和 C 这样的新的结构化语言的引入，结构化编程应运而生。

```
10 print "I am freaking out!"
20 goto 10
```



你是否真正认为这段顺序式代码很有趣呢？我是这么认为的。它把我带回到了几年之前。有一款叫作 QB64 (www.qb64.net) 的不错的编译器（并且是免费的），它支持 BASIC、QBASIC 以及 QuickBasic（它是结构式的，但不是顺序式的）的所有老式的风格。此外，QB64 支持 OpenGL，因此，它潜在性地支持高级图形和游戏设置，并且支持 BASIC 的老式变体。

助记式编程

在顺序式编程之前，开发者编写的代码更接近于计算机硬件的层级，而他们使用的是汇编语言。有一个“汇编器”程序，就像是编译器一样，但是，它将会把助记式的指令直接转换为对象或二进制文件中的机器代码，准备好让处理器一次一个字节地运行它们。一条汇编式的助记式指令，直接关联到处理器所能够理解的一条机器指令。这就像是在说机器自身的语言，并且很有挑战性。在 MS-DOS 的时代，这些汇编性的指令能够把显示模式转换成分辨率为 320×200 并且具有 256（8 位）色的图形模式，这

对于 20 世纪 90 年代的 IBM PC 游戏来说已经很好了，因为这会很快。记住，在那个时代，我们没有今天这样的显卡，只有构建到 ROM BIOS 中的“视频输出”以及操作系统所支持的各种模式。这就是那个时代的所有游戏开发者都喜欢的声名狼藉的“VGA mode 13h”。

```
mov ax, 13h
int 10h
```



有一个有趣的历史性站点，专门介绍了 VGA mode 13h 编程：

<http://www.delorie.com/djgpp/doc/ug/graphics/vga.html>.

“AX”是一个 16 位的处理器寄存器，处理器上的实际的物理电路可以当作一种通用目的的“变量”对待，这里使用了你所熟悉的术语而没有使用电子工程的语言。还有其他 3 种通用目的寄存器：BX、CX 和 DX。它们自身都是从 8 位的 Intel 处理器升级而来的，而后者拥有叫作 A、B、C 和 D 的寄存器。当发展到 16 位的时候，这些寄存器扩展为 AL/AH、BL/BH、CL/CH 和 DL/DH，它们分别表示每个 16 位寄存器的两个 8 位的部分。乍一听起来，这并不复杂。将一个值放到一个或多个这些变量寄存器之中，然后通过调用一个中断来“加载”一个过程。在 VGA 模式更改的例子中，中断是 10h。

现实世界

如果你喜欢电子工程和汇编语言这个主题，那么有一个和老式的工作对应的现代工种，这就是设备驱动编程。如今，这已经成为一种魔法，专门为那些真正理解硬件的工程师而保留。因此，你可以看到，如果你对这个工作感兴趣，学习汇编语言对此是非常有益处的。

1.3.2 接下来是什么

我们已经简单地回顾了从过去到现在的编程方法学，以理解和掌握当今所拥有的工具和语言的方法，下面，我们来介绍一下当前的情况以及有些什么发展。如今，面向对象编程仍然是专业程序员所采用的最主要的方法学。它是 Microsoft 的 Visual Studio 和 .NET Framework 等流行的工具的基础。如今的商业和科学领域中，最主要的编译型 OOP 语言是 C++、C#、BASIC（其现代变体是 Visual Basic）以及 Java。当然还有其他语言，但是，这些是最主要的。

Python 和 LUA 都是脚本编程语言。和 C++ 这样的编译型语言相比, Python 和 LUA 的处理方式有很大不同, 它们是解释型的, 而不是编译型的。当你运行一个 Python 程序的时候 (扩展名为 .PY 的一个文件), 它不会进行编译, 而会运行。你可能会在一个 Python 函数中带入语法错误, 但是, 在调用该函数之前, Python 不会提示错误。

```
# Funny syntax error example
```

```
# Bad function!
```

```
def ErrorProne():
```

```
    printgobblegobble("Hello there!")
```

```
print("See, nothing bad happened. You worry too much!")
```

Python 或者这段程序中没有一个名为 `printgobblegobble()` 的函数, 因此, 这里应该产生一个错误。输出如下。

```
See, nothing bad happened. You worry too much!
```

但是, 如果添加了对 `ErrorProne()` 函数的调用, 输出将会如下。

```
Traceback (most recent call last):
```

```
File "ErrorProne.py", line 9, in <module>
```

```
    ErrorProne()
```

```
File "ErrorProne.py", line 5, in ErrorProne
```

```
    printgobblegobble("Hello there!")
```

```
NameError: global name 'printgobblegobble' is not defined
```

现在, 对于 Python 中这一貌似忽略的部分有一些限制。如果你明显错误地定义了一个变量, 那么, 在运行之前, 它才会初次产生错误。在 Python 中, 还会因为做了另一件奇怪的事情而把事情搞砸, 那就是, 使用保留字作为变量:

```
Behold:
```

```
print = 10
```

```
print(print)
```

第一行没问题, 但是第二行导致了如下的错误。

```
Traceback (most recent call last):
```

```
File "ErrorProne.py", line 8, in <module>
```

```
    print(print)
```

```
TypeError: 'int' object is not callable
```

这条错误的意思是, `print` 变成了一个变量, 确切地说, 是一个整数, 其值设置为 10。然后, 我们试图调用旧的 `print()` 函数, 并且 Python 无法得到它。因为旧的 `print()` 函数

已经被忽略了。现在，这种奇怪的行为不再适用于 Python 语言中的保留字了，如 `while`、`for`、`if` 等保留字，而只是适用于函数。当你发现 Python 作为一种脚本语言有着巨大的灵活性的时候，我觉得你会感到惊讶的。

像 GCC 或 Visual C++ 这样的传统的编译器，甚至在考虑运行这样的代码的时候，你就会抓狂。毕竟，它们是编译器。在将程序转换成目标代码之前，它们完整地解析了程序的流程。这么做的缺点就是：编译器无法处理未知的东西，它们只能处理已知的东西，而脚本语言可以很好地处理未知的情况。

顺序式编程演变为结构化编程，结构化编程演变为 OOP，编程范型从 OOP 开始的下一次演进也将继续保持同样的方式，在范型发生变化之前，当前的编程方法学中将会出现一些明显的改变的迹象。今天，发生在 OOP 上的这些变化，可能会称为自适应编程 (adaptive programming)。在当今快节奏的世界中，没有人会像我们以前编程的时候那样，坐在计算机前阅读 WordPerfect 或 Lotus 1-2-3 的 200 页的手册。还是有人会认为“阅读手册”是解决技术问题的有效方法，但是如今，即便是带有类似手册的产品也很少见了。如今，系统必须具有交互性和自适应性。超越 OOP 的下一次演进，可能是面向实体编程 (EOP, entity oriented programming)。

想象一下，我们使用实体（使用简单规则来解决复杂问题的自包含对象）来编写代码，而不是使用包含了属性（变量）和方法（函数）的对象来编写代码。这似乎是 A.I. 的研究方向，而且应该能够与如今已有的 OOP 很好地适应。实际上，已经有了一些早期的迹象出现了。听说过 Web Service 吗？Web Service 是寄存在网上的自包含对象，程序可以使用它来执行独特的服务，而程序自身不知道如何进行这些服务。

这些 Web Service 可能会只是要求一个库存数据库的参数，并且返回与查询匹配的项目的列表。这种形式的程序交互，一定能够超越编写 SQL (structured query language, 结构化查询语句，这是关系数据库的语言)！那么，将其带入到下一个层级如何？使用某种库或搜索引擎在线查询一个服务，而不是接入一个已知的服务，这会怎么样？

作为另一个可能的示例，假设有一个在线的、可以用于游戏中的游戏实体的库（很可能是由独立开发者或开源团队创建的），其中的实体将会带有其自己的美工素材（2D 精灵、3D 网状物、材质、音频剪辑等）以及自身的行为（例如一段 Python 脚本）。需要某种格式的素材的一个已有的游戏引擎，可能会使用这种 EOP 的概念来扩展游戏设置。假设你要玩一个游戏，诸如 Minecraft (www.minecraft.net) 这样的某种世界构造游戏，并且，假设你是游戏中的某个新角色。因此，你向游戏提出查询：“我需要一把短的木头椅子”。在查询发出去后的片刻，一把短的木头椅子出现在你的游戏中。假设有一个用于 Minecraft 这样的引擎的在线游戏装备库，我们当然可以想象会

发生这种情况。

1.3.3 OOP: Python 的方式

我们已经进行了足够的历史分析和思考，从而可以触发一些有想象力的思路。现在，让我们来介绍一些具体而实际的内容，即当前的 OOP 方法学及其在 Python 中的实现。或者换句话说，我们用 Python 来创建对象。Python 确实支持 OOP 特性，但是，它不像高度特定性的语言 C++ 那样，在各个程度上支持 OOP。在开始之前，让我们先来了解一些术语。类是一个对象的蓝图。类不能做任何事情，因为它是一个蓝图。只有在运行时创建对象的时候，对象才会存在。因此，当我们编写类代码的时候，它只是一个类的定义，而不是一个对象。只有在运行时，通过类的蓝图来创建对象的时候，它才是真正的对象。类的函数也叫作方法。类的变量通常作为属性来访问（有一种方法用来获取或设置一个变量的值）。当创建一个对象的时候，类实例化为该对象。

让我们来了解 Python 的 OOP 特性的一些具体内容。示例如下。

```
class Bug(object):
    legs = 0
    distance = 0

    def __init__(self, name, legs):
        self.name = name
        self.legs = legs

    def Walk(self):
        self.distance += 1

    def ToString(self):
        return self.name + " has " + str(self.legs) + " legs" + \
            " and taken " + str(self.distance) + " steps."
```

每个定义的行末，都必须有一个冒号。关键字 `self` 描述当前的类，这和它在 C++ 中的作用是相同的。所有的类变量前面必须有一个“`self`”，以便可以认出这是类的成员；否则，它们将会被当作局部变量。`def __init__(self)` 这一行开始了类的构造函数，这是在

类实例化的时候运行的第一个方法。在构造函数之外，可以声明类变量并且在声明的时候进行初始化。

多态

术语多态表示有“多种形式”或“多种形状”，因此，多态是指具备多种形态的能力。在类的环境中，这意味着我们可以使用具有多种形态的方法，也就是说，参数的多种不同的集合。在 Python 中，我们可以使用可选的参数来让方法具备多种功能。新的 Bug 类的构造函数，可以使用可选的参数来进行变换，如下所示：

```
def __init__(self, name="Bug", legs=6):
    self.name = name
    self.legs = legs
```

同样，Walk()方法可以升级以支持一个可选的参数：

```
def Walk(self, distance=1):
    self.distance += distance
```

数据隐藏（封装）

Python 不允许变量和方法声明为私有的或受保护的，因为 Python 中的所有内容都是公有的。但是，如果你想要让代码像是数据隐藏一样地工作，这也是可以办到的。例如，如下这段代码可以用来访问或修改 distance 变量（我们假设它是私有的，即便它不是）。

```
def GetDistance(self):
    return p_distance

def SetDistance(self, value):
    p_distance = value
```

从数据隐藏的角度来看，你可以将 distance 重命名为 p_distance（使其看上去像是私有变量），然后，使用这两个方法来访问它。也就是说，如果数据隐藏对于你的程序来说很重要的话，可以这么做。

继承

Python 支持基类的继承。当定义一个类的时候，基类包含在圆括号中：

```
class Car(Vehicle):
```

此外，Python 支持多继承，也就是说，一个子类可以继承自多个父类或基类。例如：

```
class Car(Body, Engine, Suspension, Interior):
```

只要每个父类中的变量和方法与其他的变量和方法不冲突，新的子类可以访问它们而毫无问题。但是，如果有任何的冲突，来自父类的冲突变量和方法在继承顺序中具有优先性。

当一个 Python 类继承自一个基类，父类所有的变量和方法都是可用的。变量可以使用，方法可以覆盖。当调用一个基类的构造函数或任何方法的时候，我们可以使用 `super()` 来引用基类：

```
return super().ToString()
```

但是，当涉及多继承的时候，当共享相同的变量名或方法名的时候，必须使用父类的名称，以避免混淆。

1.3.4 单继承

我们先来看看单继承的示例。如下是一个 `Point` 类，以及继承自它的一个 `Circle` 类。

```
class Point():
    x = 0.0
    y = 0.0

    def __init__(self, x, y):
        self.x = x
        self.y = y
        print("Point constructor")

    def ToString(self):
        return "{X:" + str(self.x) + ",Y:" + str(self.y) + "}"

class Circle(Point):
    radius = 0.0

    def __init__(self, x, y, radius):
        super().__init__(x,y)
        self.radius = radius
        print("Circle constructor")

    def ToString(self):
```

```

    return super().ToString() + \
           ",{RADIUS=" + str(self.radius) + "}"

```

我们可以直接测试这些类：

```

p = Point(10,20)
print(p.ToString())

c = Circle(100,100,50)
print(c.ToString())

```

这会得到如下输出。

```

Point constructor
{X:10,Y:20}

Point constructor
Circle constructor
{X:100,Y:100},{RADIUS=50}

```

我们看到 `Point` 的功能很简单，但是，`Circle` 先调用 `Point` 的构造函数，然后才调用自己的构造函数，然后复杂地调用 `Point` 的 `ToString()` 并添加自己的新的 `radius` 属性。这真的有助于我们了解，为什么所有的类都有一个 `ToString()` 方法。



多继承是一片沼泽。我建议尽可能避免使用它，并且尽可能保持类的简单和直接，大多数情况下，可能只有一个层级的继承。尽可能地给你的类众多的功能，而不是将它们划分到多个类中。

现在，当创建 `Circle` 类的时候，调用构造函数并传递给它 3 个参数(100,100,50)。注意，调用了父类（`Point`）的构造函数来处理 `x` 和 `y` 参数，而 `radius` 参数在 `Circle` 中处理：

```

def __init__(self, x, y, radius):
    super().__init__(x,y)
    self.radius = radius

```

`super()`调用了 `Point` 类的构造函数，`Point` 类是 `Circle` 类的父类或基类。当使用单继承的时候，这种做法的效果令人惊奇。

1.3.5 多继承

尽管多继承是一片沼泽，但至少还是要展示一下它是如何工作的。使用多继承的时候，

我们基本上不会使用 `super()` 来调用父类中的任何内容，除非每个父类中的变量和方法都是独特的。这里有另一对类，它们构建在前面已经给出的两个类的基础之上。还记得吧，我警告过你，Python 是一种看上去很奇怪的语言。我们现在来看看。别忘了，Python 是一种脚本语言，而不是编译型语言。Python 代码是在运行时解释的。

```
class Size():
    width = 0.0
    height = 0.0

    def __init__(self,width,height):
        self.width = width
        self.height = height
        print("Size constructor")

    def ToString(self):
        return "{WIDTH=" + str(self.width) + \
            ",HEIGHT=" + str(self.height) + "}"

class Rectangle(Point,Size):
    def __init__(self, x, y, width, height):
        Point.__init__(self,x,y)
        Size.__init__(self,width,height)
        print("Rectangle constructor")

    def ToString(self):
        return Point.ToString(self) + "," + Size.ToString(self)
```

Size 类是一个新的辅助类，而 Rectangle 是我们这个示例中真正的焦点。这里，Rectangle 将继承自 Point 和 Size：

```
class Rectangle(Point,Size):
```

Point 是早就定义了的，而 Size 刚刚定义。现在，我们应该可以开始使用 Point.x、Point.y、Size.width 和 Size.height，以及每个类中的 ToString() 方法了。Python 应该不会抱怨。但是，思路是通过调用父类的构造函数来自动初始化父类。否则，我们会丧失 OOP 的所有优点，并且只是在编写结构化的代码。因此，Rectangle 构造函数必须按照名称来调用每个父类的构造函数：

```
def __init__(self, x, y, width, height):
    Point.__init__(self,x,y)
    Size.__init__(self,width,height)
```

注意，`x` 和 `y` 传递给了 `Point.__init__()`，而 `width` 和 `height` 传递给了 `Size.__init__()`。这些变量在它们各自的类中正确地初始化。当然，我们可以只是在 `Rectangle` 中定义 `x`、`y`、`width` 和 `height`，但是，这只是一个演示。通常，为了保持代码简单，我们不建议那么做。在真正的编程中，绝不要以这种方式使用继承。这里只是为了说明多继承。测试一下新的 `Size` 和 `Rectangle` 类：

```
s = Size(80,70)
print(s.ToString())

r = Rectangle(200,250,40,50)
print(r.ToString())
```

产生如下输出。

```
Size constructor
{WIDTH=80,HEIGHT=70}

Point constructor
Size constructor
Rectangle constructor
{X:200,Y:250},{WIDTH=40,HEIGHT=50}
```

现在，这真的有点意思了。`Size` 足够简单，很容易理解，但是看一下 `Rectangle` 的输出。我们调用了 `Point` 的构造函数和 `Size` 的构造函数，这完全是按照计划进行的。此外，`ToString()` 方法有效地组合了 `Point.ToString()` 和 `Size.ToString()` 各自的输出。

1.4 小结

本章是关于 Python 编程的快速介绍的第 1 章。进展这么快，是不是有点令你抓狂？不要担心，我们会以实用的方式来介绍代码编写，通过真正绘制点、圆、矩形以及其他内容来做到这点。在学习 Python 的工具的时候，我们还将创建一个精灵类，以用来在屏幕上绘制带有动画的游戏角色。好消息是，本章可能是最难的一章，因为这不但是你第一次接触奇怪的 Python 语法，也很可能是你初次接触面向对象编程。在后续的章节中，你将会发现，学习编程语言的最直接的方法，通常也是最好的方法。我希望你已经准备好了，因为从下一章开始，我们要学习 Pygame 了。

挑战

1. 打开 GeometryDemo.py 程序，并且创建一个继承自 Point 的新类，名为 Ellipse。它有一个水平半径和垂直半径，而不是像 Circle 那样只有一个半径。
2. 给 Rectangle 类添加一个名为 CalcArea()的方法，它返回 Rectangle 的面积。计算公式是：Area = Width * Height。测试该方法以确保它能工作。
3. 给 Circle 类添加一个名为 CalcCircum()的新方法，它返回圆的周长。计算公式是 Circumference = $2 * \text{Pi} * \text{Radius}$ (Pi = 3.14159)。测试该方法以确保它能工作。

第2章

初识 Pygame: Pie 游戏

本章介绍一个名为 Pygame 的游戏库，开发它是为了使得如下这些事情成为可能：绘制图形、获取用户输入、执行动画以及使用定时器让游戏按照稳定的帧速率运行。在本章中，我们只是初次认识 Pygame，学习绘制图形和文本的基础知识，并且编写一些代码。你将会看到，Pygame 不仅提供了针对图形和位图的绘制函数，还提供了用于获取用户输入、处理音频播放和监控鼠标和键盘的服务。我们将在适当的时候介绍这些额外的主题。

本章包括如下内容。

- ◎ 使用 Pygame 库；
- ◎ 以一定字体打印文本；
- ◎ 使用循环来重复动作；
- ◎ 绘制圆、矩形、线条和弧形；
- ◎ 创建 Pie 游戏。

2.1 了解 Pie 游戏

本章的示例是一款叫作 Pie 游戏。Pie 游戏使用 Pygame 来绘制填充的饼块。要在 Pie 游戏中使用 Pygame 绘制一个饼块，用户按下与该饼块对应的数字键。然后，使用 Pygame 的绘制函数来绘制饼块。当按下针对所有饼块的按键而没有犯错的时候，玩家就获胜了。如图 2.1 所示。



在使用 Pygame 之前，必须先安装它，因为 Pygame 并不是和 Python 打包到一起的。从 <http://www.pygame.org/download.shtml> 下载 Pygame。获取与你所使用的 Python 版本匹配的 Pygame 版本，这一点是很重要的。本书使用 Python 3.2 和 Pygame 1.9。如果你需要在安装方面得到帮助，请参阅本书附录 A 了解更多细节。

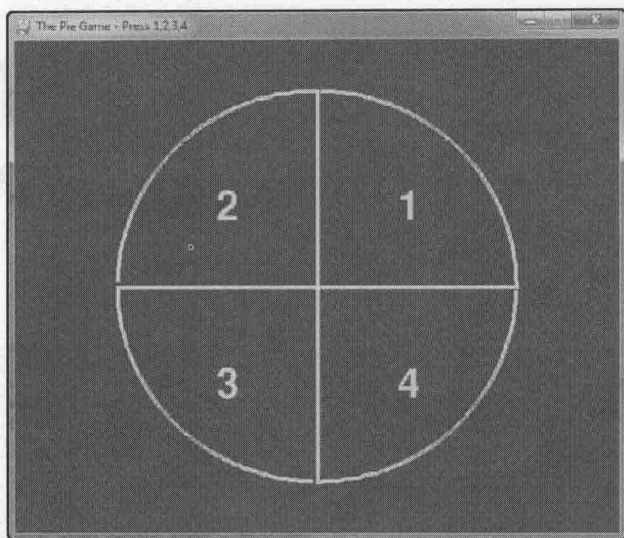


图 2.1 Pie 游戏

2.2 使用 Pygame

使用 Pygame 的第一步, 是将 Pygame 库导入到 Python 程序中, 以便可以使用它。

```
import pygame
```

下一个步骤是, 导入 Pygame 中的所有常量, 以准备好可以在我们的代码中访问它们。这是可选的, 但是, 这往往会让代码更整齐。由于担心效率, 一些 Python 程序员不喜欢导入一个库中的所有内容, 但是, 这样做会让我们的代码整个变得更容易阅读。

```
from pygame.locals import *
```

现在, 我们可以初始化 Pygame 了:

```
pygame.init()
```

初始化了 Pygame, 我们就可以访问这个库的所有的资源了。下一步是获取对显示系统的访问, 并且创建一个窗口。分辨率由你决定, 但是, 注意宽度和高度参数要放在圆括号中。(600,500) 对变成了带有 x 和 y 属性的一个点。在 Python 中, 源代码语法是由解释器松散地确保的, 因此, 我们编写代码的方式, 在某种较为强类型的语言 (如 C++) 中是不允许的。

```
screen = pygame.display.set_mode((600,500))
```



在 <http://www.pygame.org/docs/index.html>, 可以找到 Pygame 的一个不错的参考手册。

2.2.1 打印文本

Pygame 支持使用 `pygame.font` 将文本输出到图形窗口。要绘制文本, 我们必须先创建一个字体对象:

```
myfont = pygame.font.Font(None, 60)
```

可以向 `pygame.font.Font()` 构造函数提供一个 TrueType 字体, 诸如 “Arial”, 但是, 使用 `None` (不带引号) 将会导致使用默认的 Pygame 字体。字体大小 60 已经很大了, 但是, 这只是一个简单的示例。现在, 使用 Pygame 绘制文本并不是一个轻量型的进程, 而是一个重量型的进程。这意味着, 文本并不是快速地绘制到屏幕上, 而是渲染到一个平面, 然后再将其绘制到屏幕上。由于这是一个耗费时间的过程, 建议首先在内存中创建文本平面 (或图像), 然后再将文本当作一个图像来绘制。当我们必须实时地绘制文本的时候, 直接绘制是没问题的; 但是, 如果文本是无法修改的, 最好先把文本提前渲染到一个图像之上。

```
white = 255,255,255
blue = 0,0,255
textImage = myfont.render("Hello Pygame", True, white)
```

`textImage` 对象是可以使用 `screen.blit()` 绘制的平面, 我们的高难度的绘制函数, 将会在所有的游戏和示例中广泛地使用。第一个参数显然是文本消息, 第二个参数是抗锯齿字体 (为了提高质量) 的一个标志, 第三个参数是颜色 (一个 RGB 值)。

要绘制文本, 通常的过程是清除屏幕, 进行绘制, 然后刷新显示。让我们看看所有这三行代码:

```
screen.fill(blue)
screen.blit(textImage, (100,100))
pygame.display.update()
```

如果现在运行程序, 会发生什么情况? 继续前进并尝试一下。你是否看到了在程序运行后所出现的窗口? 由于我们的代码中没有任何延迟, 窗口应该会快速出现并关闭。需要有一个延迟。但是, 我们将继续进行, 而不是进行延迟。

2.2.2 循环

我们所看到的简化的示例有两个问题。首先，它只是运行一次然后就停止了。其次，没有办法获取任何用户输入（即便它不会立即退出）。因此，让我们看看如何修正这两个问题。首先，我们需要一个循环。在 Python 中，这通过关键字 `while` 来实现。While 语句将执行冒号后面的代码，直到条件为假。只要 `while` 条件为真，它将持续运行：

```
while True:
```

接下来，我们创建一个事件处理程序。在早期的阶段，我们期望窗口所发生的事情是，它能够等待用户关闭它。关闭事件可能是点击窗口右上角的“×”，或者只是按下任何的键。注意，`while` 循环中的代码是缩进的。在此之后缩进的任何代码，都将包含在这个 `while` 循环中。

```
while True:
```

```
    for event in pygame.event.get():
```

```
        if event.type in (QUIT, KEYDOWN):
```

```
            sys.exit()
```

最后，我们以缩进代码的形式在 `while` 循环中添加了绘制代码和屏幕刷新，并且程序由此结束。为了方便学习，这里给出了完整的没有任何空行和注释的程序。程序的输出如图 2.2 所示。

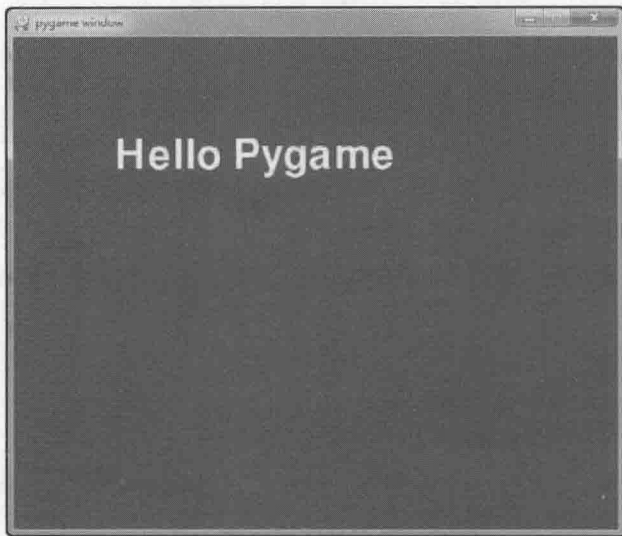


图 2.2 Hello Pygame 程序