



计 算 机 科 学 丛 书

PEARSON

原书第2版

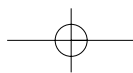
Python计算与编程实践 多媒体方法

(美) Mark Guzdial Barbara Ericson 著 王江平 译

Introduction to Computing and Programming in Python
A Multimedia Approach Second Edition



机械工业出版社
China Machine Press



计算机科学丛书

Python计算与编程实践

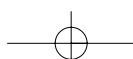
多媒体方法

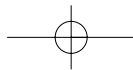
(原书第2版)

Introduction to Computing and Programming in Python:
A Multimedia Approach (Second Edition)

(美) Mark Guzdial 著
Barbara Ericson
佐治亚理工学院
王江平 译

华章图书





本书是一本别出心裁的程序设计入门教程，以Python数字多媒体编程为主线，依次讲解了图像、声音、文本和电影的处理，其中穿插介绍了大量的计算机程序设计基础知识。方法独到，示例通俗易懂，条理清晰，将趣味性和实用性融于讲解之中。

本书适合用做计算机专业导论课或非计算机专业编程课程的教材，也可用做软件开发人员学习计算机数字多媒体处理知识和Python语言的专业参考书。

Authorized translation from the English language edition, entitled INTRODUCTION TO COMPUTING AND PROGRAMMING IN PYTHON: A MULTIMEDIA APPROACH, 2E, 9780136060239 by Mark Guzdial, Barbara Ericson, published by Pearson Education, Inc, publishing as Prentice Hall, Copyright © 2010.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD., and CHINA MACHINE PRESS Copyright © 2012.

本书中文简体字版由Pearson Education（培生教育出版集团）授权机械工业出版社在中华人民共和国境内（不包括中国台湾地区和香港、澳门特别行政区）独家出版发行。未经出版者书面许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

本书封面贴有Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问 北京市展达律师事务所

本书版权登记号：图字：01-2011-2890

图书在版编目（CIP）数据

Python计算与编程实践：多媒体方法（原书第2版）/（美）古兹迪阿尔（Guzdial, M.），（美）埃里克森（Ericson, B.）著；王江平译. —北京：机械工业出版社，2012.6
（计算机科学丛书）

书名原文：Introduction to Computing and Programming in Python: A Multimedia Approach, Second Edition

ISBN 978-7-111-38738-1

I. P… II. ①古… ②埃… ③王… III. 软件工具-程序设计 IV. TP311.56

中国版本图书馆CIP数据核字（2012）第121641号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：盛思源

印刷

2012年7月第1版第1次印刷

185mm×260mm · 22印张（含1.5印张彩插）

标准书号：ISBN 978-7-111-38738-1

定价：69.00元

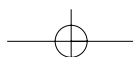
凡购本书，如有缺页、倒页、脱页，由本社发行部调换

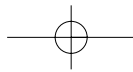
客服热线：(010) 88378991；88361066

购书热线：(010) 68326294；88379649；68995259

投稿热线：(010) 88379604

读者信箱：hzjsj@hzbook.com





文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson、McGraw-Hill、Elsevier、MIT、John Wiley & Sons、Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum、Bjarne Stroustrup、Brain W. Kernighan、Dennis Ritchie、Jim Gray、Afred V. Aho、John E. Hopcroft、Jeffrey D. Ullman、Abraham Silberschatz、William Stallings、Donald E. Knuth、John L. Hennessy、Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方式如下：

华章网站：www.hzbook.com

电子邮件：hzjsj@hzbook.com

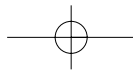
联系电话：(010) 88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037



华章科技图书出版中心



译者序

Introduction to Computing and Programming in Python: A Multimedia Approach, 2E

这是一本什么书

这是一本什么样的书呢？根据主要内容，我们不妨先给出3种可能的诠释：

- 1) 一本讲Python编程的书，以多媒体处理为例增强趣味性；
- 2) 一本讲多媒体编程的书，选用Python作为示例语言；
- 3) 一本别出心裁的“计算机导论”教程，以Python多媒体编程为线索讲解各种有趣的计算机科学知识，把读者带进充满乐趣的计算机科学殿堂。

每个答案都不算错，但最恰当的当数第3个。没错，本书以Python数字多媒体编程为主线，依次讲解了图像、声音、文本和电影的处理，其中穿插介绍了大量的计算机科学基础知识。方法独到，示例通俗，条理清晰，将趣味性和实用性融于讲解之中，所有这些都会给阅读本书带来美好的体验。

除了主要内容（数字多媒体处理）之外，本书的最后一部分还介绍了更通用的计算机科学与编程知识：计算机性能、函数式编程和多媒体编程。

本书读者对象

显然，本书适合用做计算机专业导论课或者非计算机专业编程课程的教材。

想了解或温习一下计算机数字多媒体处理知识的专业程序员也可以翻翻这本书。

想学习如何用程序处理多媒体的非专业人士也可以读一读。新东方的李笑来老师因为“能够编写一些批处理脚本”，才有了《TOFEL核心词汇21天突破》。想学习编程的朋友们，可别忘了Python是非常强大、非常流行，同时又易懂易学的脚本语言。

最后，所有对Python语言感兴趣，想通过一些简单实用的例子来学习这门编程语言的朋友，都可以看看这本书。

我与华章

自从跟聂雪军、王昕等朋友合作翻译《代码之美》起，这是我与机械工业出版社华章公司的第5次合作了，非常感谢他们五六年来对我一贯的信任，特别是陈冀康先生。

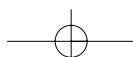
华章一直在大量引进国外的优秀计算机图书，我本人也是受益者，最近正在精读他们引进的《Computer Systems: A Programmer's Perspective》第2版。

致谢

首先感谢我的家人在这4个多月的时间里给予的支持。

本书的审校工作再次邀请了校对奇人张伸（新浪微博：@loveisbug）帮忙，此人眼尖、心细，再加上因博览群书而培养起来的文字感，每次都能帮上大忙。上次审校《Java语言精粹》一书，出版后发现的唯一一处错别字出现在未经他过目的“译者序”中。这次他又帮忙通读了全书译稿（包括译者序），十分感谢他的辛勤和一丝不苟。

遇到较难翻译的内容时，译者经常上译言网（www.yeeyan.org）论坛求助。感谢耐心解答过问题的以下网友：nc、桥头堡、dingdingdang、qmiao和sparklark。



在我的gtalk上，高远（新浪微博：@狼大人）是随叫随到的翻译顾问。谢谢他的不厌其烦和及时响应。

齐艳昀也为本书的翻译出了一份力，同样表示感谢。

最后不得不提及的是：初译本书时，有超过70%的内容是利用上下班时间在地铁上靠笔译完成的。感谢上海地铁2号线和7号线全体员工，有了你们，才有了平稳运行的地铁，让我可以平稳地做翻译。

联系译者

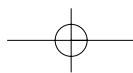
尽管译者尽心尽力试图译好每一句话、每一个词，但限于时间和水平，错误疏漏在所难免，恳请读者朋友多多批评，不吝赐教。

勘误信息可以提交到译者个人的豆瓣小站上：<http://site.douban.com/120940>，也可以直接发往译者的邮箱：steadhorse@163.net，还可以关注译者的新浪微博：@steadhorse，直接互动。

王江平

2011年8月 于上海浦东





第2版前言

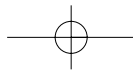
Introduction to Computing and Programming in Python: A Multimedia Approach, 2E

感谢大家对本书第1版的兴趣，正是大家的热情支持我们完成了第2版。感谢那些不知道名字的朋友提供的阅读评论，这些评论指导我们完成了本次修订。

第1版和第2版的主要区别如下：

- 1) 增加了对Python语言的覆盖面，包括更多标准库方面的话题、全局域和更多的控制结构。
 - 2) 更加强调了抽象代码和可复用代码。
 - 3) 第13章增加了一些内容，指导大家制作标准的AVI或QuickTime电影与他人分享。
 - 4) 每章结尾处大大增加了练习题的数量。
 - 5) 所有的下标都改为从0开始，而不是从1开始。使用0，而不是1作为第一个下标，与标准Python更兼容。
 - 6) 去掉了使用Swing创建用户界面和介绍JavaScript的一章。
 - 7) 重写了关于设计和调试的一章，加入了设计和测试示例，强调了维护。
 - 8) 把创建和修改文本的一章拆成了两章。增加了有关隐写术（steganography）的例子。
 - 9) 把关于语言范式（编程风格）的一章拆成了两章，以提供更多关于函数式编程（比如非可变函数：non-mutable function）和面向对象编程（使用与Logo中类似的小海龟来介绍对象）的内容。
 - 10) 增加了更多教师们希望在导论课中提及的概念，比如负数的二进制表示。
 - 11) 整体上，语言描述更加清晰，删掉了一些不必要的细节。（希望如此。）
 - 12) 所有软件和素材的最新版本可从<http://mediacomputation.org>网站找到。
- 第2版增加了一位合著者为本书的写作出版带来了美好的回忆。

Mark Guzdial和Barbara Ericson
佐治亚理工学院



计算机教育方面的研究表明：人们不只是“学习编程”，而是学习编程去实现某种事物[5, 22]，而做事动机的不同会决定人们是否学习编程[8]。每一位教师都面临着一个挑战：挑选某种事物来激发学习编程的兴趣，且要有足够强大的激发效果。

人们需要交流，交流的欲望是首要兴趣之一。人们逐渐将计算机更多地用做交流工具，而不是计算工具。今天，所有公开发行的文本、图像、声音、音乐和电影基本上都是用计算机技术制作的。

本书教大家如何编写程序，从而实现用数字媒体进行的交流。本书关注的是如何像专业程序员那样处理图像、声音、文本和电影，但使用的却是连学生都能写的程序。我们知道，大多数人会使用专业级的应用软件来实现这种类型的数字媒体处理。然而，懂得如何自己编写程序意味着你能做更多事情，而不是只能做手边的应用程序所支持的那些事情。你的表达能力不会受应用程序的限制。

还有一点：了解数字媒体应用程序中算法的工作原理会让你更好地使用它们，或者更方便地从一种应用程序转向另一种应用程序。对应用程序来说，如果你关心的是哪个菜单项做哪些事情，那么每种应用程序都是不同的。相反，如果你关心的是按自己喜欢的方式来调整像素的位置和颜色，那么，透过菜单项关注自己想要表达的东西可能会更容易。

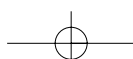
本书不只讲数字媒体编程。处理数字媒体的程序实现起来并不容易，即使实现了其行为也可能出乎你的意料。问题自然就来了，比如“同样的图像为什么在Photoshop中做滤镜操作速度更快？”或者“这太难调试了——有更容易调试的编程方法吗？”回答这样的问题是计算机科学家的事情。本书末尾有好几章讲到了计算，而不只是编程。最后的这几章内容从数字媒体处理延伸到了更一般的话题。

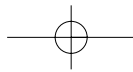
计算机是人类设计出来的、最令人惊叹的创造性设备。它完全由精神素材（mind-stuff）构成。“与其梦想它，不如成为它”的理念在计算机上完全可行。只要能想象一件事，就可以让它在计算机上成为“现实”。玩转编程可以是、也应该是极大的乐趣。

致教师

本书的内容符合ACM/IEEE计算机课程2001（Computing Curriculum 2001）标准文档[4]中描述的“从命令式开始”的要求。内容从赋值、顺序操作、迭代、条件式和函数定义这样的基础内容开始。当学生掌握这些内容之后，才强调抽象内容（如算法复杂度、程序效率、计算机组成、层次式分解、递归和面向对象编程）。

这种非常规的教学次序依据的是学习科学（learning science）中的研究发现：记忆是关联性的。我们记忆新的东西靠的是把它跟旧的东西关联起来。基于“将来某天可能有用”的假定，人们能学会概念和技巧，但这些概念和技巧只跟那个假定相关联。结果就像所谓的“脆弱知识”（brittle knowledge）[9]——这种知识能让你通过考试，但你很快会忘记它，因为除了在那课堂上听过它之外，没有其他事情与之关联了。





VIII

如果概念和技巧能跟许多不同的思想关联起来,或者跟日常生活中的想法关联起来,那么记忆就会更牢固。想让学生获得可转移的知识 (**transferable knowledge**, 即可在新情形中运用的知识), 我们必须帮助他们把新知识和更一般的问题关联起来, 这样, 记忆就可以通过关联这些问题来索引它们[26]。本书中, 我们通过学生可以考察、关联 (比如在消除照片红眼效果时使用的条件式), 之后再在上面叠加抽象 (比如分别使用递归和函数式滤镜加映射来达到同样的目标) 的具体体验来讲解计算与编程。

我们知道, 对学习计算机的学生来说, 从抽象起步效果并不好。Ann Fleury揭示了这样的事实: 如果讲封装和复用[13], 那么计算机导论课上的学生根本听不懂。学生们更喜欢便于跟踪的简单代码, 而且真正认为这样的代码更好。学生们需要时间和经验来理解设计良好的系统所包含的价值。没有经验, 学生们很难学习抽象。

本书采用的多媒体计算教学方法从一些常见任务开始, 如图像处理、考察数字音乐、查看和制作网页、制作视频等, 许多人都用计算机做这些事。然后, 我们基于这些活动解释编程和计算。我们想让学生在访问Amazon (打个比方) 网站时会想到: “这是个网上书店——我知道它是用数据库和把数据库实体格式化网页的一组程序来实现的。” 我们想让学生在使用Adobe Photoshop和GIMP时, 思考图像滤镜是如何实际处理一个像素的红、绿、蓝颜色分量的。从一个有意义的上下文开始, 知识和技能更容易传递。它也使书中的例子更有意思、更能激发学生兴趣, 这些都有助于学生上课时集中精力。

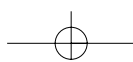
基于多媒体计算的教学方法将大约2/3的时间花在多媒体的体验上, 在一个能调动学习兴趣的环境中让学生获得各种不同媒体的经验。在这2/3以后, 他们自然会问一些与计算有关的问题, 典型的问题如“为什么Photoshop比我的程序快?” 或者“电影的代码很慢——为什么会这么慢?” 此时, 我们会引入抽象和来自计算机科学的真知灼见来回答他们的问题。那是本书最后一部分的内容。

另外一个计算机教育领域的研究团体考察了计算机导论课的退课率和失败率居高不下的原因。一种常见的论点是: 计算机课程看上去“不切实际”, 而且过分关注诸如效率之类“单调乏味的细节” [1, 31]。然而, 学生们认为 (在问卷调查和面谈中告诉我们的) 与交流有关的上下文是切合实际的。这个切合实际的上下文部分地解释了我们在佐治亚理工学院 (Georgia Tech) 导论课程上的长期成功的原因, 本书就是为这一课程写就的。

在本书采用的教学方法中, 知识次序的不同不只体现在最后介绍抽象这一点上。我们在第3章的第一个重要程序中就开始使用数组和矩阵。通常, “计算机导论” 课程都会把数组的知识推到后面, 因为它们明显比保存简单数值的变量复杂。一种既切合实际又具体可行的上下文是非常强大的[22]。我们发现学生们处理图片中的像素矩阵完全没有问题。

据统计, “计算机导论” 课程中, 学生们退课或得到D、F分数的比率 (通常称为WDF率) 处在30%~50%的范围或更高。根据最近一项针对计算机导论课程失败率的全球性调查, 美国54所大学的平均失败率为33%, 17所国际大学的平均失败率为17% [6]。在佐治亚理工学院, 从2000年到2002年, 所有专业要求的导论课程中, 平均WDF率为28%。我们在自己的“媒体计算导论” (Introduction to Media Computation) 课程中使用了本教程的第1版。首次试开这门课的时候, 有121名学生选修, 其中没有计算机或工程专业的学生, 而且2/3是女生。最终我们的WDF率只有11.5%。

之后的两年里 (从2003年春到2005年秋), 我们的课程在佐治亚理工学院的平均WDF率 (多位讲师, 数千名学生) 为15% [21]。事实上, 之前28%的WDF率与现在15%的WDF率是没



有可比性的，因为所有专业都采用了之前的课程，只有人文、建筑和管理专业采用了新课程。个别专业取得了更显著的变化。比如，管理专业在1999~2003年采用较早的课程时，WDF率为51.5%，而采用新课程的两年中只有11.2%的失败率[21]。自本书第1版发行之后，其他几所学校也采用了这种方法并根据自身的情形做了调整而且评估了结果，他们在成功率上都取得了类似的显著进步[36, 37]。

怎样使用本书

本书以基本一致的次序呈现了我们在佐治亚理工学院的授课内容。个别老师可能会跳过某些章节（比如有关加法合成、MIDI和MP3的部分），不过，所有的内容我们都在自己的学生身上试讲过的。

然而，本书的使用并非仅限于此，还有人以其他方式使用过它：

- 只用第2章和第3章就可以对计算做个简单介绍，或者还需要第4章、第5章的一些内容。甚至有人用本书讲过只有一天的媒体计算专题研讨会。
- 第6~8章基本上重复了第3~5章中的计算机科学概念，但上下文是声音而不是图像。我们发现这种重复很有用——有些学生似乎在使用一种媒体时比使用另一种能更好地关联迭代和条件式的概念。此外，它让我们有机会指出同一种算法可在不同的媒体中拥有类似的效果（比如，缩放图片与调整音高用的是同一种算法）。但如果要节省时间，当然可以跳过这部分。
- 第12章在编程和计算方面没有介绍新的概念。电影的处理尽管能激发学生的兴趣，但为了节省时间也可以跳过。
- 关于最后一部分，我们建议有些章节至少要讲一讲，从而引导学生以更抽象的方式思考计算和编程，但显然没必要涵盖所有章节。

Python和Jython

本书使用的编程语言是Python。人们把Python描述为“可执行的伪代码”（executable pseudo-code）。我们发现计算机专业和非专业人士都可以学习Python。由于Python实际用于交互任务（比如网站开发），因此它很适合用做“计算机导论”课程的语言。例如，Python网站（<http://www.python.org>）上的招聘广告显示：像谷歌和Industrial Light & Magic这样的公司都在招Python程序员。

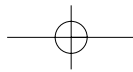
具体来讲，本书使用的Python“方言”是Jython（<http://www.jython.org>）。Jython就是Python。Python（通常用C实现）和Jython（用Java实现）的区别类似于任何两种语言实现之间的区别（比如Microsoft和GNU C++的实现）——基础语言是完全一致的，区别仅在于某些库和细节方面，大多数学生都注意不到。

格式说明

Python示例代码的字体如：`x = x + 1`。更长的示例版式如下：

```
def helloWorld():  
    print "Hello, world!"
```

当显示用户输入对Python响应的内容时，字体风格类似，但用户的输入将出现在Python提示符（`>>>`）之后：



X

```
>>> print 3 + 4  
7
```

在本书中，你还会发现几种特殊类型的图标。



计算机科学思想
关键的计算机科学概念以此种图标显示。



常见bug
可能导致程序失败的常见问题以此种图标显示。



调试技巧
事前预防bug潜入程序的好办法以此种图标突出显示。

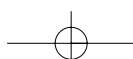


实践技巧
真正有帮助的最佳实践或技术以此种图标突出显示。

致谢

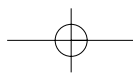
衷心感谢：

- Jason Ertle、Claire Bailey、David Raines和Joshua Sklare，他们在非常短的时间内以惊人的高质量做出了第1版的JES。多年来，Adam Wilson、Larry Olson、Yu Cheung (Toby) Ho、Eric Mickley、Keith McDermott、Ellie Harmon、Timmy Douglas、Alex Rudnick、Brian O'Neill和William Fredrick (Buck) Scharfnorth III使JES成为今天这个简单实用且仍然易于理解的工具。
- Adam Wilson构建了对研究声音和图像以及处理视频来说非常有用的MediaTools。
- Andrea Forte、Mark Rickman、Matt Wallace、Alisa Bandlow、Derek Chambless、Larry Olson和David Rennie帮忙整理了教程讲义。Derek、Mark和Matt编写了许多示例程序。
- 许多佐治亚理工学院人共同努力成就了这项工作。副教务长Bob McMath和计算机学院的教务副主任Jim Foley很早就投入了这项工作。Kurt Eiselt努力使这项工作变成现实，说服他人认真对待这项工作。“为初涉计算机科学的学生讲授媒体计算”，Janet Kolodner和Aaron Bobick对这种想法感到兴奋和鼓舞。Jeff Pierce审查了书中所用媒体函数的设计并给出了建议。关于如何精确地传达数字化素材的内容，Aaron Lanterman提出了很多建议。Joan Morton、Chrissy Hendricks、David White，以及GVU中心的所有工作人员都确保满足我们的需要并处理好全部细节，确保工作向着同一个目标前进。Amy Bruckman和Eugene Guzdial为Mark争取了时间，使最终版本得以完成。
- 非常感谢Colin Potts和Monica Sweat，他们在佐治亚理工学院讲这门课，为我们提供了许多与课程有关的见解。



- Charles Fowler是佐治亚理工学院之外冒险在自己的学校（盖恩斯维尔学院）尝试这门课程的第一人，对此我们深表谢意。
- 2003年春季，佐治亚理工学院开设的试讲课程对这门课程的改进非常重要。Andrea Forte、Rachel Fithian和Lauren Rich评估了这次试讲，这对我们理解哪些地方有效果、哪些地方没有效果具有不可低估的价值。第一批助教（Jim Gruen、Angela Liang、Larry Olson、Matt Wallace、Adam Wilson和Jose Zagal）为这一教学方法的设计做了大量工作。Blair MacIntyre、Colin Potts和Monica Sweat帮忙调整了教程讲义，从而方便其他人使用。Jocken Rick把CoWeb/Swiki变成了CS1315课上学生们最爱逛的好地方。
- 许多学生指出了书中的错误并提出了改进建议。感谢Catherine Billiris、Jennifer Blake、Karin Bowman、Maryam Doroudi、Suzannah、Gill、Baillie Homire、Jonathan Laing、Mireille Murad、Michael Shaw、Summar Shoaib，特别是Jonathan Longhitano，他在文字编辑方面很有天赋。
- 感谢之前媒体计算课上的学生Constantino Kombosch、Joseph Clark和Shannon Joiner同意我在示例中使用他们的课堂快照。
- 与本书相关的研究工作得到了国家科学基金会（National Science Foundation）的赞助——赞助来自本科教育部门的CCLI项目，以及CISE教育革新项目。感谢你们的支持。
- 感谢计算机课上的学生Anthony Thomas、Celines Rivera和Carolina Gomez允许我们使用他们的图片。
- 最后，但最重要的，感谢我们的孩子Matthew、Katherine和Jenifer Guzdial，他们同意为了爸爸妈妈的媒体项目而被拍照、录音，他们对这门课程非常支持并为之欢欣鼓舞。

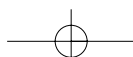
Mark Guzdial和Barbara Ericson
佐治亚理工学院

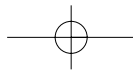


目 录

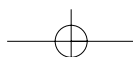
Introduction to Computing and Programming in Python: A Multimedia Approach, 2E

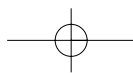
出版者的话	
译者序	
第2版前言	
第1版前言	
第一部分 导 论	
第1章 计算机科学与媒体计算导论	2
1.1 计算机科学是关于什么的	2
1.2 编程语言	4
1.3 计算机理解什么	5
1.4 媒体计算：为什么要把媒体数字化	7
1.5 大众的计算机科学	8
1.5.1 计算机科学与交流有关	8
1.5.2 计算机科学与过程有关	9
习题	9
第2章 编程导论	11
2.1 编程与命名有关	11
2.2 Python编程	13
2.3 JES编程	13
2.4 JES媒体计算	15
2.4.1 显示图片	18
2.4.2 播放声音	19
2.4.3 数值命名	20
2.5 构建程序	22
习题	27
第3章 使用循环修改图片	30
3.1 图片的编码	30
3.2 处理图片	35
3.3 改变颜色值	40
3.3.1 在图片上运用循环	40
3.3.2 增/减红(绿、蓝)	42
3.3.3 测试程序：它真的能运行吗	45
3.3.4 一次修改一种颜色	46
3.4 制作日落效果	47
3.5 亮化和暗化	51
3.6 制作底片	52
3.7 转换到灰度	53
习题	55
第4章 修改区域中的像素	58
4.1 复制像素	58
4.2 图片镜像	60
4.3 复制和转换图片	66
4.3.1 复制	66
4.3.2 制作拼贴图	72
4.3.3 通用复制	74
4.3.4 旋转	75
4.3.5 缩放	77
习题	81
第5章 高级图片技术	84
5.1 颜色替换：消除红眼、深褐色调和色调分离	84
5.1.1 消除红眼	86
5.1.2 深褐色调和色调分离：使用条件式选择颜色	88
5.2 合并像素：图片模糊化	92
5.3 比较像素：边缘检测	93
5.4 图片融合	94
5.5 背景消减	96
5.6 色键	98
5.7 在图像上绘图	101
5.7.1 使用绘图命令	102
5.7.2 向量和位图表示	104
5.8 指定绘图过程的程序	105
习题	107
第二部分 声 音	
第6章 使用循环修改声音	110
6.1 声音是如何编码的	110





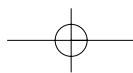
6.1.1 声音的物理学	110	第9章 构建更大的程序	164
6.1.2 探索声音的样子	113	9.1 自顶向下设计程序	164
6.1.3 声音编码	115	9.1.1 自顶向下设计示例	165
6.1.4 二进制数和二进制补码	116	9.1.2 设计顶层函数	166
6.1.5 存储数字化的声音	117	9.1.3 编写子函数	168
6.2 处理声音	118	9.2 自底向上设计程序	171
6.2.1 打开声音并处理样本数据	118	9.3 测试程序	172
6.2.2 使用JES媒体工具	121	9.4 调试技巧	174
6.2.3 循环	123	9.4.1 找出担心的语句	174
6.3 改变音量	123	9.4.2 查看变量	175
6.3.1 增大音量	123	9.4.3 调试冒险游戏	176
6.3.2 真的行吗	124	9.5 算法和设计	179
6.3.3 减小音量	127	9.6 在JES之外运行程序	180
6.3.4 理解声音函数	128	习题	181
6.4 声音规格化	128	第三部分 文本、文件、网络、 数据库和单媒体	
习题	131	第10章 创建和修改文本	186
第7章 修改一段样本区域	133	10.1 文本作为单媒体	186
7.1 用不同方法处理不同声音片段	133	10.2 字符串：构造和处理字符串	187
7.2 剪接声音	135	10.3 处理部分字符串	189
7.3 通用剪辑和复制	140	10.3.1 字符串方法：对象和点号语法 简介	190
7.4 声音倒置	142	10.3.2 列表：强大的结构化文本	191
7.5 镜像	143	10.3.3 字符串没有字体	194
习题	144	10.4 文件：存放字符串和其他数据的 地方	194
第8章 通过合并片段制作声音	146	10.4.1 打开文件和操作文件	195
8.1 用加法组合声音	146	10.4.2 制作套用信函	197
8.2 混合声音	147	10.4.3 编写程序	197
8.3 制造回声	148	10.5 Python标准库	201
8.3.1 制造多重回声	149	10.5.1 再谈导入和私有模块	202
8.3.2 制作和弦	149	10.5.2 另一个有趣模块：Random	202
8.4 采样键盘工作原理	150	10.5.3 Python标准库的例子	204
8.5 加法合成	153	习题	205
8.5.1 制作正弦波	153	第11章 高级文本技术：Web和信息	208
8.5.2 把正弦波叠加起来	155	11.1 网络：从Web获取文本	208
8.5.3 检查结果	156	11.2 通过文本转换不同媒体	211
8.5.4 方波	157	11.3 在图片中隐藏信息	216
8.5.5 三角波	158	习题	219
8.6 现代音乐合成	160		
8.6.1 MP3	161		
8.6.2 MIDI	161		
习题	162		





XIV

第12章 产生Web文本	221	14.3.1 时钟频率和实际的计算	268
12.1 HTML: Web的表示方法	221	14.3.2 存储: 什么使计算机速度慢	269
12.2 编写程序产生HTML	225	14.3.3 显示	270
12.3 数据库: 存放文本的地方	229	习题	270
12.3.1 关系型数据库	231	第15章 函数式编程	272
12.3.2 基于散列表的关系型数据库示例	231	15.1 使用函数简化编程	272
12.3.3 使用SQL	234	15.2 使用Map和Reduce进行函数式编程	275
12.3.4 使用数据库构建Web页面	236	15.3 针对媒体的函数式编程	277
习题	237	15.4 递归: 一种强大的思想	279
第四部分 电 影		15.4.1 递归式目录遍历	284
第13章 制作和修改电影	240	15.4.2 递归式媒体函数	286
13.1 产生动画	241	习题	287
13.2 使用视频源	247	第16章 面向对象编程	289
13.3 自底向上制作视频效果	250	16.1 对象的历史	289
习题	254	16.2 使用“小海龟”	290
第五部分 计算机科学议题		16.2.1 类和对象	290
第14章 速度	258	16.2.2 创建对象	290
14.1 关注计算机科学	258	16.2.3 向对象发送消息	291
14.2 什么使程序速度更快	258	16.2.4 对象控制自己的状态	292
14.2.1 什么是计算机真正理解的	258	16.2.5 小海龟的其他函数	293
14.2.2 编译器和解释器	259	16.3 教小海龟新的技艺	295
14.2.3 什么限制了计算机的速度	263	16.4 面向对象的幻灯片	297
14.2.4 让查找更快	265	16.4.1 Joe the Box	300
14.2.5 永不终止和无法编写出的算法	266	16.4.2 面向对象的媒体	302
14.2.6 为什么Photoshop比JES更快	268	16.4.3 为什么使用对象	306
14.3 什么使计算机速度更快	268	习题	307
		附录A Python快速参考	309
		参考文献	313

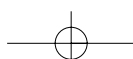


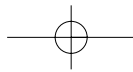
|第一部分|

Introduction to Computing and Programming in Python: A Multimedia Approach, 2E

导 论

- 第1章 计算机科学与媒体计算导论
- 第2章 编程导论
- 第3章 使用循环修改图片
- 第4章 修改区域中的像素
- 第5章 高级图片技术





第1章

Introduction to Computing and Programming in Python: A Multimedia Approach, 2E

计算机科学与媒体计算导论

本章学习目标

- 计算机科学是关于什么的和计算机科学家关心什么的。
- 我们为什么要把媒体数字化。
- 为什么学习计算（computing）是有价值的。
- 编码（encode）的概念。
- 计算机的基础组件。

1.1 计算机科学是关于什么的

计算机科学是关于过程（process）的学习：我们，或者计算机如何做事情，我们如何指定要做的事情，如何指定要处理的东西。这是个相当枯燥的定义。让我们试着给出一个比喻性的描述吧。



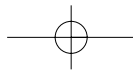
计算机科学思想：学习计算机科学就像研究菜谱（recipe）

它们是一类特殊的菜谱——一种可被计算设备执行的菜谱，但这一点只对计算机科学家有意义。总体而言，重要的一点是：计算机科学的菜谱精确地定义了必须完成的工作。

如果你是一位生物学家，想要描述生物迁徙或DNA复制的原理，那么以一种可被全面定义和理解的方式编写一套菜谱，精确地说明所发生的事情，将很有帮助。如果你是一位化学家，想要解释化学反应中如何达到平衡，情况也一样。使用计算机程序，工厂经理可以定义机器和皮带的布局，甚至测试它如何工作——而不必真正把那些重家伙搬到实际位置上。可以在计算机上运行的菜谱称为程序。计算机之所以能对科学的研究和理解带来如此深刻的变化，精确定义任务和模拟事件的能力是其首要原因。

用菜谱来比喻程序听起来有点儿滑稽，但这个比喻非常有用。计算机科学家研究的很多东西都可以用菜谱来形容：

- 有些计算机科学家研究如何编写菜谱：完成一项任务有更好或更差的方法吗？如果你曾经试图把蛋清跟蛋黄分开，那么就会明白是否懂得正确方法结果大不相同。计算机科学的理论家关注最快和最短的菜谱，以及占用空间最少的菜谱（可以把它想象成空间对抗——这是个不错的比喻）。菜谱如何工作与如何编写菜谱完全不同，这方面的工作称为算法研究。软件工程师关注大的团队如何把菜谱合并起来且仍然正常工作。（某些程序，就像跟踪Visa/MasterCard记录的那种，其菜谱包含上百万的步骤！）术语软件指的是完成一项任务的计算机程序的集合。
- 有些计算机科学家研究菜谱中使用的单位。菜谱使用公制度量单位还是英制度量单位重要吗？使用任何一种都可以，但如果不知道一磅或一杯是多少，那么对你来说菜谱就不

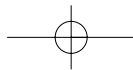


那么好理解了。也有些单位对某种任务有意义而对其他任务没有意义，但如果能将任务与使用的单位匹配好，那么你就能更方便地解释自己的想法，更快速地完成任任务——并且避免错误。有没有想过大海中的轮船为什么使用海里/小时“knot”来度量速度？为什么不用类似“米/秒”这样的单位？有时在某种特殊情况下（比如在大海中的轮船上）常见的术语并不适合或者根本不好用。计算机科学中对单位的研究称为数据结构。有些科学家研究如何跟踪基于大量不同单位的大量数据，他们研究的是数据库。

- 任何事情都可以写成菜谱吗？是否有些菜谱是写不出来的？计算机科学家知道有些菜谱是写不出来的。比如，你无法写出一份菜谱来精确判断另外一些菜谱是否有实际效果。智能又是怎样的情况呢？我们能写出一份菜谱，让遵循它的计算机做到真正思考吗（你又如何判断它是否正确呢）？计算机原理、智能系统、人工智能和计算机系统方面的科学家关注的就是这类东西。
- 甚至有些计算机科学家关注人们是否喜欢菜谱给出的结果，就像报社的餐饮评论家。其中有一些是人机界面（产生人们使用的某种界面的“菜谱”，这类界面包括窗口、按钮、滚动条，以及我们能想到的运行中的程序所使用的其他元素）专家，关心人们是否喜欢菜谱工作的方式。
- 正如某些厨师专长于某一类菜谱，如烤饼或烤肉，有些计算机科学家也会专攻特定种类的菜谱。致力于图形领域的科学家主要关注产生图片、动画甚至电影的菜谱。致力于计算机音乐领域的科学家主要关注产生声音的菜谱（常常是有旋律的声音，但也不一定）。
- 还有一些计算机科学家研究菜谱的突现特性（**emergent property**）。考虑一下万维网。万维网实际上是由数百万相互关联的菜谱（程序）组成的集合。为什么网络的一部分在某一点上会变慢？这是数百万程序中出现的现象，当然不是人们预期的。这些是网络计算机科学家研究的内容。真正令人惊讶的是：这些突现特性（仅当你有许多同时交互的菜谱时才会发生的事情）也可以解释非计算机领域的事情。举例来说，蚂蚁寻食、白蚁筑堆，这些都可以描述成一种特殊的事情：它们仅发生在有大量的小程序在做简单、交互式的任务时。

换一个角度来考虑，菜谱的比喻仍然恰当。大家都知道菜谱中有些东西可以改变但不会使结果产生太大变化。你可以将所有单位增大一个因子（比如加倍）来产出更多的东西。在意大利面酱中加入更多的大蒜或牛至粉（**oregano**）。但菜谱中也有一些东西是不能改变的。如果菜谱中需要发酵粉，那么就不能以小苏打代之。如果打算把饺子煮熟再煎一下，顺序反过来恐怕也不会有好结果（如图1.1所示）。

同样的道理也适用于软件菜谱。通常，有些东西很容易改变：事物的实际名字（尽管你应该按一致的方式改变名字）、某些常量（以普通数字形式，而非变量形式出现的数值），或者还有待处理数据的某些范围（数据片段）。然而，发给计算机的命令，通常必须严格保持给定的次序。随着我们的讲解，你将学会哪些东西可以安全地改变，哪些不可以。



4 · 第一部分 导 论

砂锅鸡	
剔骨鸡胸肉3整块	切碎的番茄1听（28盎司）
中等大小的洋葱1个，切碎	番茄汁1听（15盎司）
大蒜碎末1汤匙	蘑菇1听（6.5盎司）
橄榄油2汤匙，之后1/4杯	番茄酱1听（6盎司）
面粉1.5杯	意大利面酱*1/2罐（26盎司）
Lawry调味盐1/4杯	意大利调味料3汤匙
青椒1个，切碎（可选），颜色随意	大蒜粉1茶匙（可选）

鸡肉切成1英寸见方的小块。翻炒洋葱和大蒜直至洋葱呈透明状。加入面粉和Lawry调味盐。按1：4～1：5的比例混合调味盐和面粉，多到足以涂满鸡肉表面。把切好的鸡肉和调味的面粉放进一个袋子，抖动一下使面粉涂在鸡肉表面上。把涂好的鸡肉放进洋葱和大蒜中。快速翻炒直至鸡肉呈褐色。放一点油，防止它们黏在一起或者炒焦；我有时会加1/4杯橄榄油。放入番茄、面酱、蘑菇和番茄酱（以及可选的青椒），炒熟。放入意大利调味料。我喜欢大蒜，因此通常也会放点大蒜粉，搅匀。因为加了面粉，所以各种酱会变得很黏。我通常用意大利面酱把它们打开，最多半瓶。慢炖20~30分钟。

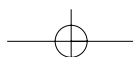
图1.1 一道菜谱——你可以放入双倍调味料，但多放一杯面粉鸡肉就分不开了，而且，不要尝试在放入番茄汁后再爆炒鸡肉

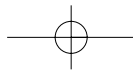
1.2 编程语言

计算机科学家使用编程语言编写菜谱（如图1.2所示）。针对不同目的，可以使用不同的编程语言。有些语言广泛流行，如Java和C++。有些语言冷门一些，如Squeak和T。另外一些是为了让计算机科学思想更易于理解，如Scheme或Python，但易于学习的事实并不总能使它们非常流行或者成为专家们构建更大更复杂菜谱时的首选。讲授计算机科学时，选择一门既易于学习又足够流行且有用，而学生有动力去学习的语言，权衡起来颇有难度。

Python/Jython
<pre>def hello(): print "Hello World"</pre>
Java
<pre>class HelloWorld { static public void main(String args[]) { System.out.println("Hello World!"); } }</pre>
C++
<pre>#include <iostream.h> main() { cout << "Hello World!" << endl; return 0; }</pre>
Scheme
<pre>(define helloworld (lambda () (display "Hello World") (newline)))</pre>

图1.2 编程语言比较：一种常见的简单编程任务就是在屏幕上打印“Hello World!”这句话





计算机科学家为什么不使用像英语或者西班牙语这样的人类自然语言呢？问题在于自然语言是沿着增进非常聪明的物种（人类）之间的沟通这条路发展起来的。正如我们将在下一节详细解释的那样，计算机是十分愚蠢的。它们对明确程度的要求，自然语言并不擅长。而且，我们在自然交流中彼此说的话与你在一套计算菜谱中说的话并不完全相同。你什么时候与人讲述过像毁灭战士（Doom）、雷神之槌（Quake）或超级玛丽兄弟（Super Mario Brothers）这种视频游戏的详细情节，详细到对方能复制出这款游戏（比如在纸上）的程度？英语并不擅长这类任务。

因为需要编写的菜谱种类很多，所以编程语言的种类也很多。通常，用C语言写的程序快速高效，但也常常难读难写，而且需要一些与计算机关系更为密切的单元，而不是与鸟类迁徙、DNA或其他任何你想要编写的菜谱有关的。Lisp语言（以及像Scheme、T和Common Lisp这样的相关语言）非常灵活，很适合作用来探索如何编写以前从没写过的菜谱，但Lisp与C这样的语言相比样子太奇怪了，以至于很多人都不用它，结果知道它的人自然就更少了。假如你想雇一百个人来做项目，与不那么流行的语言相比，找到一百个了解一种流行语言的人要容易得多——但这并不意味着对你的任务而言那门流行语言是最好的。

在本书中使用的编程语言是Python（更多信息参阅<http://www.python.org>）。Python是一门相当流行的编程语言，常用于Web或媒体编程。Web搜索引擎谷歌就使用了Python。媒体公司Industrial Light & Magic也使用Python。你可以从<http://wiki.python.org/moin/Organizations-UsingPython>获得一份使用Python的公司列表。Python易学易读，非常灵活，但效率一般。同样的算法分别用C和Python编写，C代码很可能更快。

本书使用的是称为Jython（<http://www.jython.org>）的Python版本。Python通常用C语言实现。Jython是用Java实现的Python——这意味着Jython实际是Java编写的程序。Jython支持跨多种计算机平台运行的多媒体程序。Jython是真正的编程语言，可用于重要的工作任务。你可以从Jython网站为自己的计算机下载某个Jython版本，然后就可以用它做各种事情。在本书里，我们将通过一种称为JES（Jython Environment for Students）的特殊编程环境来使用Jython，这种编程环境能使Jython编程更加方便。但JES中能做的任何事情，在一般的Jython中也能做——而且，用Jython编写的大部分程序在Python中也能正常运行。

以下解释了在本书中使用的重要术语：

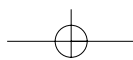
- 程序是使用编程语言描述的过程，这种过程能达到对某人有用的某种结果。程序可以很小（就像计算器的程序），也可以很大（就像银行用来跟踪所有账户的程序）。
- 算法（与程序不同）是独立于编程语言的过程描述。同样的算法可以在多种不同的程序中以多种不同的方式使用多种不同的语言实现，但同一种算法具有相同的过程。

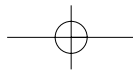
本书中使用的术语菜谱描述完成某种任务的程序或程序的一部分。将使用术语菜谱来强调完成某种有用的与媒体有关的任务的程序片段。

1.3 计算机理解什么

编写计算菜谱是为了在计算机上运行。计算机如何知道要执行什么呢？使用菜谱，我们又能让计算机做什么呢？答案是：“能做的少之又少。”计算机格外愚蠢。实际上，它们只知道数字。

其实，说计算机知道数字都不完全准确。计算机使用数字的编码（encoding）。计算机是





6 · 第一部分 导 论

响应线路电压的电子设备。每条线路称为一个位 (bit)。如果一条线路上有电压,那么我们就说它编码了一个“1”;如果没有电压,我们就说它编码了“0”。我们把这些线路(位)分组。一组8位称为一个字节(Byte)。于是,通过一组8条线路(一个字节),我们就有了一种8个0或1的模式,比如,01001010。使用二进制(binary)数字系统,我们可以把这个字节解释为一个数字(如图1.3所示)。我们说计算机知道数字,就是这么来的。

计算机有一段塞满字节的内存(memory)。在任一时刻,计算机处理的东西都存储在它的内存中。这就是说计算机处理的任何东西都编码在字节之中——JPEG图片、Excel表格、Word文档、讨厌的Web弹出广告,还有上一封垃圾邮件。

计算机可以用数字做许多事情。它可以把数字加、减、乘、除、排序、收集、复制、过滤(比如,“把这些数字复制一份,但只要偶数”。),或者比较它们并基于比较的结果做事。举例来说,一份菜谱可以告诉计算机:“比较这两个数。如果第一个小于第二个,就跳转到本菜谱的第5步;否则,继续执行下一步。”

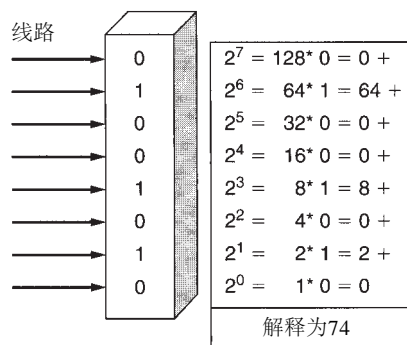


图1.3 具有电压模式的8条线路是一个字节,它被解释为8个0或1的一种模式,进而被解释为一个十进制数

说到现在,似乎计算机就是一种奇特的计算器,这当然是它被发明出来的原因。计算机最初的用途之一就是在第二次世界大战中计算抛射体弹道(如果风来自东南方,风速每小时15英里,想击中北偏东30度0.5英里处的目标,那么弹筒应该倾斜到……)。现代计算机每秒钟能做几十亿次运算。然而,使计算机可用于通用菜谱的原因不在其他,恰在于编码的概念。

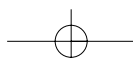


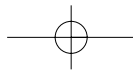
计算机科学思想:计算机可以把编码分层

计算机可以把编码分层,分到几乎任意复杂的程度。数字可以解释为字符,字符组合起来又可以解释成网页,而网页被解释之后,可以显现为多种字体和风格。但在最底层,计算机只“知道”被我们解释为数字的电压。

如果把某个字节解释为数字65,那么它可能就是数字65。或者,使用一种数字到字母的编码标准:美国标准信息交换编码(American Standard Code for Information Interchange, ASCII),它可以是字母A。如果65出现在被我们解释为文本的其他一组数字中,而且保存在以“.html”结尾的文件里,那么它可能是类似“<a href=...”结构的一部分,在Web浏览器中被解释为链接的定义。在计算机下层,那个A只是一种电压模式。菜谱一层层堆上去,在Web浏览器的层上,它就定义了你点击之以获取更多信息的某种东西。

如果计算机只理解数字(这已经是一种延伸),它如何处理这些编码呢?当然,它知道如





何比较数字，但它又怎样扩展这种能力从而按字母顺序排列一个类的列表呢？通常，每一层编码都实现为软件的一部分或一层。我们有懂得如何操控字符的软件。字符软件知道如何进行名字比较，因为它已经编码了诸如*a*在*b*之前这样的信息，而比较字母编码中的数字顺序，也就得到了字母顺序。字符软件又被其他软件使用，用来处理文件中的文本。像Microsoft Word、Notepad或TextEdit这样的软件就会用到这一层。另一种软件片段知道如何解释HTML（Web的语言），而同一软件的另外一层知道如何将HTML显示成正确的文本、字体、风格和颜色。

按照类似的方法，我们可以在计算机中为自己的任务创建编码层次。我们可以告诉计算机细胞包含线粒体和DNA，DNA有4种核苷酸，工厂里有这些种类的印刷机和那些种类的压印器等。创建编码和解释的层次，从而让计算机针对特定问题使用正确的单元，这是数据表示的任务，或者说定义正确的数据结构。

听上去似乎有很多软件，的确如此。软件按这种方式分层后，会使计算机的速度有所下降，但计算机的强大之处就在于它的速度奇快——而且一直在变得更快。



计算机科学思想：摩尔定律（Moore's Law）

Intel（Intel制造运行Windows操作系统的计算机上所使用的处理器）的创立者之一戈登·摩尔（Gordon Moore）提出：晶体管（计算机的关键组件）的数目每18个月会增加一倍，而价格保持不变；这意味着同样的钱每隔18个月就能买到双倍的计算能力。也就是说，计算机一直在变得更小、更快、更便宜。几十年了，这一定律一直成立。

如今的计算机每秒钟可以执行数十亿个菜谱步骤。它们能把百科全书的数据保存在内存中！它们不知疲倦，永不厌烦。在100万用户中查找某个持卡人？没有问题！找出使方程取最优值的数值集合？小菜一碟！

处理数百万图片元素、声音片段或电影画面？这就是媒体计算了。通过这本书，你将编写出处理图像、声音和文本的菜谱，甚至其他菜谱。这是可能的，因为在计算机中任何东西都是以数字化的形式表示的，菜谱也是。看完这本书，你将编写出实现数字视频特效的菜谱，用来创建Web页面，就像Amazon和eBay那样；以及像Photoshop那样对图像进行滤镜处理的菜谱。

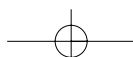
1.4 媒体计算：为什么要把媒体数字化

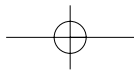
让我们考虑一种适合图片的编码。把图片想象成由一个个的小点组成。这不难想象：使靠近显示器或电视屏幕，你就能看到眼前的图片本来就是由小点组成的。每个小点都有自己的颜色。物理学告诉我们：颜色可以描述为红、绿、蓝的总和。红色和绿色相加得到的是黄色。这三种颜色加在一起得到的是白色。三种颜色都关掉，得到的就是一个黑点。

如果把图片中的每个点编码为三个字节的集合，每个字节表示该点显示在屏幕上时红色、绿色和蓝色的数量，情况会怎样呢？收集许多这样的三字节集合来决定给定图片上的所有点，又会怎样呢？这是一种相当合理的表示图片的方式，基本上也是第3章中我们将使用的方式。

操作这些点（每个点称为一个像素或图片元素）可能需要大量的处理。在一幅图片中，你可能需要从计算机或网页上处理上千甚至数百万个点。但计算机并不厌烦，而且速度极快。

我们将使用的声音编码每秒钟的声音包含44 100个双字节集合（这两个字节称做一个样本）。一首三分钟的歌曲需要158 760 000个字节（立体声要再加一倍）。在这些样本上做任何操作都需要大量的处理。但以每秒钟10亿次操作的速度，你可以在很短的时间里对这些字节做很多操作。





建立这种编码需要把媒体改变一下。看一看现实世界：它并不是由你能看清的大量小点组成的。听一听声音：你能听出每秒钟有几个声音小段吗？你听不出每秒钟内的声音小段，正是这一事实使编码的创建成为可能。我们的眼睛和耳朵是有局限的：只能感知这么多，只能感知这么小的东西。如果把一幅图像拆分成足够小的点，那么你的眼睛无法分辨出它其实不是连续的颜色流。如果把一段声音拆分成足够小的片段，你的耳朵也无法分辨出它其实不是连续的声音能量流。

将媒体编码成小片段的过程称为数字化（digitization），有时也称为“going digital”。（根据《American Heritage Dictionary》）Digital的意思是：“数字的、与数字有关的或类似数字的；特别是手指。”（Of, relating to, or resembling a digit, especially a finger.）将事物数字化就是把一种连续的、不可数的东西转化成可数的东西，好比用手指头。

对能力有限的人类感官来说，数字媒体如果做得好，感觉上与原来的媒体是一样的。唱片录音机（见过没？）连续用模拟信号将声音捕捉下来，照片也把光线作为持续流捕捉下来。有些人说他们能听出唱片跟CD的不同，但对我们的耳朵以及大多数测量工具来说，CD（数字化的声音）听起来是一样的，甚至更清晰。数码相机能以足够高的清晰度生成相片质量的图片。

为什么要把媒体数字化？因为那样一来媒体就更容易处理、精确复制、压缩和传输。举例来说，相片中的图像很难处理，但同样的图像经数字化之后处理起来就容易多了。本书就是关于如何利用并处理不断数字化的媒体世界——并在此过程中学习电脑计算的。

摩尔定律使得媒体计算可以作为引导性的入门主题。媒体计算依赖于计算机在大量字节上执行大量操作。现代计算机很容易做到这一点。即使用缓慢（但容易理解）的语言、低效（但易写易读）的菜谱，我们仍可通过媒体处理来学习计算。

处理媒体时应尊重作者的数字版权。在不违反公平使用法的前提下，为教学目的修改图像和声音是允许的。但是，分享或发行处理过的图像或声音就有可能侵犯所有者的版权了。

1.5 大众的计算机科学

为什么要通过编写媒体处理程序来学习计算机科学呢？为什么不想成为计算机科学家的人也应该学习计算机科学呢？对于通过媒体处理来学习计算的过程，你又如何会感兴趣呢？

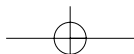
如今，多数专业都会用到媒体处理：报纸、视频、磁带录音、摄影、绘画。慢慢地，这种处理过程都改用计算机来完成了。如今，媒体大都以数字化的形式存在。

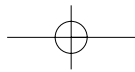
我们使用软件来处理这些媒体。我们使用Adobe Photoshop处理图像，使用Audacity处理声音，还可能用Microsoft PowerPoint把我们的媒体组装成幻灯片。我们使用Microsoft Word处理文本，使用Netscape Navigator或者Microsoft Internet Explorer来浏览因特网上的媒体。

那么，为什么不想成为计算机科学家的人也应该学习计算机科学呢？为什么应该学习编程呢？学会使用所有这些优秀的软件还不够吗？后面的几节给出了这些问题的答案。

1.5.1 计算机科学与交流有关

数字媒体用软件来处理。如果只能使用别人制作的软件来处理媒体，那么你的交流能力就会受到限制。如果你想表达一件事情，而Adobe、Microsoft、Apple或其他公司的软件都不能帮你表达，那该怎么办呢？或者，想以一种它们所不支持的方式来表达，又该怎么办呢？如果懂得如何编写程序，即使亲自动手将花费更多时间，那么你也拥有了按自己喜爱的方式处理媒体的自由。





从一开始就学习这些工具，又会有怎样的效果呢？在与计算机打交道的全部岁月里，我们见证了太多种类的软件，来了又去了，诸如绘图软件、绘画软件、字处理软件、视频编辑软件等。你不能只学习一种工具，然后指望在整个职业生涯中使用它。如果了解这些工具的原理，那么你便拥有了核心的理解力，就可以从一种工具转向另一种工具。你能够基于算法，而不是工具来思考自己的媒体作品。

最后，如果制作媒体是为了Web、市场、印刷、广播或其他类似目的，那就有必要了解一些对媒体能做什么不能做什么的常识。对媒体消费者来说，知道媒体可以怎样处理，知道什么是真实的，什么只是一种把戏，就更重要了。如果了解媒体计算的基本知识，那么你对媒体便有了超越任何工具所能提供的更深层次的理解。

1.5.2 计算机科学与过程有关

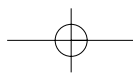
1961年，Alan Perlis在麻省理工学院做的一次演讲中提出：计算机科学，更明确地说是编程，应该是通识教育（liberal education）的一部分[17]。Perlis是计算机科学领域的重要人物。计算机科学的最高奖项是ACM图灵奖，Perlis是该奖的首位获得者。他是软件工程领域的著名人物，美国大学的第一批计算机科学系当中有好几个是他创立的。

Perlis的观点可以用微积分来类比一下。一般认为，微积分是通识教育的一部分：并非所有人都学微积分，但如果你想接受良好的教育，通常至少应修满一学期的微积分。微积分研究的是比率（rate），在许多领域都很重要。正如本章之前所讲的，计算机科学研究的是过程。过程对几乎所有的领域都重要，从商业到科学，从医药到法律。从规范的角度理解过程对每个人来说都很重要。计算机实现的过程自动化改变了每一个行业。

最近，Jeannette Wing提出每个人都应该学习计算思维（computational thinking）[34]。她认为计算学科中讲授的技巧类型对所有学生来说都是关键的技巧。这正是Alan Perlis预言的：自动化计算将改变我们了解世界的方式。

习题

- 1.1 如今每一种职业都使用计算机。试着用Web浏览器和搜索引擎（如谷歌）找到与你的研究领域相关的计算机科学或计算站点。比如搜索“生物计算机科学”或“管理计算”。
- 1.2 上网查找一份ASCII码表：一张罗列了各个字符及其相应数字表示的表格。写出组成你名字的ASCII字符所对应的数字序列。
- 1.3 上网查找一份Unicode表。看看ASCII和Unicode之间的区别是什么？
- 1.4 考虑1.4节描述的图片表示：图片中每个点（像素）用三个字节分别表示该点颜色的红、绿、蓝分量。基于这种方法，表示一张640×480像素的图片（网上常见的图片尺寸）需要多少字节？表示一张1 024×768像素的图片（常见的屏幕尺寸）又需要多少字节？（现在，你觉得所谓“300万像素”的数码相机是个什么概念？）
- 1.5 1位可以表示0或1。2位有4种可能的组合：00、01、10、11。4位或8位（1字节）有多少种不同的组合？2字节（16位）能表示多少数字？4字节又能表示多少数字？
- *1.6 如何基于字节表示一个浮点数？上网搜索一下“浮点”（floating point），看看能找到什么信息。
- 1.7 上网查一下Alan Key和《Dynabook》。Alan与媒体计算有什么关系？
- 1.8 上网查一下Grace Hopper。她对编程语言有怎样的贡献？



10 · 第一部分 导 论

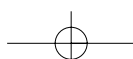
- 1.9 上网查一下Philip Emeagwali。他得过哪种计算机科学方面的奖项？
- 1.10 上网查一下Alan Turing。了解一下计算机能做什么和编码如何工作这些观念与他有什么联系。
- 1.11 上网查一下Harvard Computers。他们对天文学做过何种贡献？
- 1.12 上网查一下Adele Goldberg。她对编程语言有何种贡献？
- 1.13 上网查一下Kurt Gödel。关于编码，他做了哪些令人惊叹的事情？
- 1.14 上网查一下Ada Lovelace。在第一台机械式计算机建造出来之前，她做了哪些令人惊叹的事情？
- 1.15 上网查一下Claude Shannon。他为自己的硕士论文做了哪些工作？
- 1.16 上网查一下Richard Tapia。他为增进计算的多样性做过什么？
- 1.17 上网查一下Frances Allen。她得过哪种计算机科学方面的奖项？
- 1.18 上网查一下Mary Lou Jepsen。她在研究什么新技术？
- 1.19 上网查一下Ashley Qualls。她创建了什么价值百万美元的东西？
- 1.20 上网查一下Marissa Mayer。她是做什么的？

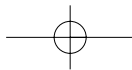
深入学习

James Gleick的《Chaos》（混沌）一书对突现特性做了更多描述——阐释了微小变化如何导致了显著结果，以及难以预见的交互为何能给设计带来意料之外的影响。

Mitchel Resnick的《Turtles, Termites and Traffic Jams: Exploration in Massively Parallel Microworlds》（海龟、白蚁和交通阻塞：大规模并行微观世界探索）一书[33]讲述了通过同时运行成百上千的微小过程（程序），并让它们彼此交互，就可以相当精确地描述蚂蚁、白蚁、甚至交通阻塞和黏液菌的行为。

《Exploring the Digital Domain》（探索数字领域）[3]是一本极好的计算科学入门书，其中有许多关于数字媒体的有用信息。





编程导论

本章学习目标

本章媒体学习目标:

- 制作并显示图片。
- 制作并播放声音。

本章计算机科学学习目标:

- 使用**JES**输入并执行程序。
- 创建并使用变量来保存值和对象, 比如图片和声音。
- 创建函数。
- 识别诸如整数、浮点数和媒体对象等不同类型的(编码)的数据。
- 顺序执行函数中的操作。

2.1 编程与命名有关



计算机科学思想: 大部分编程工作都与命名有关

计算机可以将名字, 或者符号, 关联到几乎任何东西: 特定的字节; 一组字节组成的数值变量或一串字符; 像文件、声音或图片这样的媒体元素; 甚至更抽象的概念, 比如命名的菜谱(程序)或命名的编码方式(类型)。就像哲学家和数学家一样, 计算机科学家也认为某些名字的选取具有更高的品质: 命名方案(名字及其命名的事物)应当优雅、节省且好用。命名是一种抽象形式。我们使用名字来指代被命名的东西。

显然, 计算机本身并不关心名字。名字是为人服务的。如果计算机只是个计算器, 那么记住词语及其关联值之间的关系不过是浪费内存。但对人来说, 命名却是非常强大的机制。它允许你以自然的方式使用计算机, 这种方式甚至全面扩展了我们理解菜谱(过程)的方式。

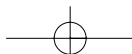
编程语言实际上就是一组名字的集合, 计算机拥有这些名字的编码, 于是它们能让计算机完成我们期望的工作, 或者按我们期望的方式解释数据。在有些语言中, 基于它所使用名字我们可以定义新的名字, 从而构造自己的编码层次。给变量赋值就是为计算机定义名字的一种方法。定义函数则是为菜谱命名。

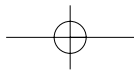
程序由一组名字和它们的值构成, 其中有些名字拥有计算机指令类型的值(“代码”)。我们的指令将使用**Python**语言来指定, 结合以上两个定义可以得出: **Python**编程语言为我们提供了一组有用的名字, 这些名字对计算机是有意义的。而我们的程序就是从这些有用的名字中选出一些, 然后加上我们自己定义的名字, 把它们联合起来就可以告诉计算机我们想让它做什么。



计算机科学思想: 程序以人为本, 而不是以机器为本

记住: 名字只对人有意义, 而非对计算机有意义。计算机只接受指令。好的程序是对人有意义(可理解且好用)的程序。





名字有好也有坏。一组良好的编码和名字能让我们以自然的方式定义菜谱而无须解释太多。各种语言可以分别理解为一套名字和编码的集合。对某种任务来说，有些语言更适合。描述同样的菜谱，有的语言需要你编写更多代码——但有时这种“更多”会带来（对人来说）可读性更好的菜谱，有助于别人理解你要表达的东西。

哲学家和数学家所追寻的品质感非常相似，他们尝试用几个词来描述世界，追寻既能涵盖更多情形又能保证同道中人可以理解的优雅词句组合。这恰恰是计算机科学家也在尝试的事情。

对于菜谱中的单位和值（数据），其解释方式常常也需要命名。还记得我们在1.3节说过的吗？任何东西都被编码成字节，但字节可以解释为数字。在有些编程语言中，你可以显式指定某个值是`byte`，之后又让语言把它作为数字看待，即一个`integer`（有时也叫`int`）。类似地，你可以告诉计算机这些字节是一组数字的集合（整数数组）、一组字符的集合（字符串），甚至是更为复杂的浮点数（`float`）（任何具有小数点的数字）的编码。

在Python中，我们将显式告诉计算机如何解释我们的值，但我们不会告诉它某个名字只与某种编码关联。像Java和C++那样的语言是强类型（`strongly typed`）的，在那些语言中名字与特定的类型或编码牢固地关联起来。它们要求你指明这个名字只与整数关联，那个名字只与浮点数关联。Python仍然拥有类型（可通过名字来引用的编码方式），但不像Java和C++那么明确。Python也有保留字（`reserved words`），保留字是一些词语，你不能用它们给自己的东西命名，因为它们在语言中已有特定含义。

文件和文件名

编程语言不是计算机关联名字和值的唯一场所。计算机的操作系统负责管理磁盘上的文件，并把它们与名字关联起来。你熟悉或使用的操作系统可能包括Windows 95、Windows 98（Windows ME、NT、XP、Vista……）、MacOS和Linux。文件是位于硬盘（计算机在关电后保存信息的部分）上的一组值（字节）的集合。如果你知道一个文件的名字并把它告诉操作系统，那么你就可以得到这个名字所关联的值。

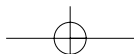
你可能在想：“我使用计算机好多年了，从来没有向操作系统提供文件名字。”或许你在提供的时候没有意识到，实际上当你从Photoshop的文件选择对话框中选取一个文件，或者从目录窗口（或Explorer、Finder）中双击一个文件时，你都在要求某个软件将被选取或被双击的名字提供给操作系统并取回相关的值。然而，当你自己编写程序时，就要显式地获得文件名字并取得它们的值。

文件对媒体计算非常重要。磁盘上可以存储大片大片的信息。还记得我们讨论过的摩尔定律吗？每一元钱能买到的磁盘容量比每一元钱能买到运算速度增长得还要快！今天的计算机磁盘可以存储整部电影、数小时（或数日？）的声音，以及与几百卷胶卷等量的图片。

这些媒体的体积并不小，即使以压缩形式存放，一张屏幕大小的图片也会超过百万字节，一首歌曲可能有300万字节甚至更多。你需要把它们存放在计算机关电后仍然存在而且有大量空间的地方。

与磁盘不同，计算机的内存是不持久的（其内容会随着电源的中断而消失），而且空间相对较少。内存一直在变大，但与磁盘空间相比仍然是小巫见大巫。使用媒体时，你需要将它们从磁盘加载到内存，但工作完成之后你不会愿意把它们留在内存中。它们太大了。

可以把计算机的内存想象成一间宿舍。你可以方便地取用宿舍里的东西——它们就



边，伸手即得，用起来也方便。但你不会把自己拥有的一切（或者你希望拥有的一切）都放在那一间宿舍里。你所有的财产？你的滑雪板？你的汽车？你的小船？那太傻了。相反，你把大东西放在存放大东西的地方。你知道需要时如何得到它们（而且在你需要或条件允许时可将它们放回宿舍）。

你把信息带进内存时，就需要为数值命名，以便于之后取用它。从这个意义上，编程有点儿像代数（algebra）。为编写出通用（即对任意数字或数值都成立）的方程或函数，你会使用变量：就像 $PV = nRT$ ， $e = Mc^2$ 或者 $f(x) = \sin(x)$ 。这些 P 、 V 、 R 、 T 、 e 、 M 、 c 和 x 就是值的名字。当你计算 $f(30)$ 的值，知道自己在计算 f 且 x 是“30”的名字。在程序中使用媒体时，我们将采用（与数值）相同的方式来命名它们。

2.2 Python编程

本书将使用的语言称为Python。它是由Guido van Rossum发明的一门语言。Guido以著名的英国喜剧剧团Monty Python来命名自己的语言。许多未经正规计算机科学训练的人已经使用Python多年——Pythobn的设计目标就是简单易用。本书将要使用的具体实现是Jython，因为它可以实现跨平台的多媒体编程。

实际上，你将使用一种称为JES（Jython Environment for Students）的工具来编写程序。JES是个简单的编辑器（输入程序文本的工具）和交互式工具，你可以在JES中尝试新事物，或者在其中创建新程序。本书所讨论的与媒体相关的名字（函数、变量、编码）将在JES中运行（也就是说，它们不是标准Python发布包的一部分，但我们使用的基础语言是标准Python）。

可以到<http://mediacomputation.org>上阅读JES的安装说明。那上面描述的过程将指导你安装Java、Jython和JES。安装之后，计算机上会有个漂亮的图标，双击它就能启动JES。JES有分别面向Windows、Macintosh和Linux的版本。



调试技巧：需要时别忘了安装Java

对于大多数人来说，只需将JES的文件夹拖到硬盘上就可以使用了。不过，如果你已经安装了Java，而且是个不能运行JES的老版本，那么启动JES时还是会有问题。倘若确实有问题，可以到Sun的网站<http://www.java.sun.com>上获取一份新版本的Java[⊖]。

2.3 JES编程

如何启动JES取决于你的平台。在Windows和Macintosh上，你会看到一个JES图标，双击它就能启动JES。在Linux上，你可能需要在Jython目录下输入一条命令，比如./JES.sh。你可以到网站上查阅相关说明，以确定自己的计算机应采用哪种启动方法。



常见bug：JES启动起来可能较慢

JES可能需要较长时间来加载。不必担心——你会看到长时间的启动画面，但只要你看到了启动画面，它就会加载。通常在第一次使用之后它会启动得更快。

⊖ 如今的新链接是：<http://www.oracle.com/technetwork/java/index.html>。——译者注



常见bug：让JES运行得更快

运行JES实际是在运行Java，这一点后面还会详细讨论。如果发现JES运行得有些慢，那就给它更多内存。你可以退出其他程序，从而达到这一效果。邮件程序、即时通信软件和数字音乐播放器都会占用内存——有时占用得还不少呢！退出这些程序后，JES就能运行得快一些。

启动之后的JES就如图2.1所示，其中有两块主要的区域（它们之间的分隔条可以移动，以便重新调整两块区域的大小）：

- 上面的部分是程序区。这是你编写程序（创建的程序和它们的名字）的地方。这一区域只是个文本编辑器——可以理解成用于编程的Microsoft Word。在你按下Load Program按钮之前，计算机不会真正解释你在程序区中输入的名字，而且，在程序保存之前你也按不了Load Program按钮（保存程序可以使用File菜单下的Save菜单项）。

不必担心按下了Load Program却忘记了保存程序。JES在程序保存之前不会加载它，会提供机会让你保存的。

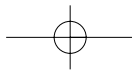
- 底下的部分是命令区。这里是你就一句一句地命令计算机做事的地方。在“>>>”提示符之后输入命令，然后当按下<Return>（Apple）或<Enter>（Windows）键的时候，计算机就会解释你输入的词句（即对这些词句应用Python编程语言的含义和编码）并执行你让它做的事情。解释的内容还会包含你在程序区输入和加载的内容。



图2.1 标注了不同区域的JES

- 右边的区域是帮助区。你可以选择一项内容，然后点击相应的“Explain setRed”按钮来获得帮助。

图2.1中还可以看到JES的其他特性。但目前我们还不想用它们做太多事情。Watcher按钮可以打开一个观察器（调试器），观察器是一个窗口，通过里面的工具可以观察计算机怎样执行你的程序。Stop按钮允许你终止运行中的程序（比如你觉得它运行的时间太长，或者意识到它并没有做你想让它做的事情）。



实践技巧：了解你的助手

首先需要研究的一项重要特性就是**Help**菜单。这个菜单下面有丰富的帮助信息，都是关于编程和使用**JES**的。现在就开始研究它吧，这样当你开始编写自己的程序时，就大致知道那里有什么信息了。

2.4 JES媒体计算

我们将从在命令区输入命令的简单任务开始——暂时不定义新的名字，而只在**JES**中使用计算机已经知道的名字。

print是我们需要知道的一个重要名字。使用时，它后面总是跟着其他东西。**print**的含义是：“以一种可读的形式显示后面的东西，不管它是什么。”后面跟着的可能是个计算机知道的名字，也可能是个表达式（**expression**）（与代数学意义上的表达式差不多）。试着点击命令区并输入命令：**print 34 + 56**，然后输入**Enter**键——就像这样：

```
>>> print 34 + 56
90
```

34 + 56是**Python**所能理解的一个数字表达式。很明显，它由两个数字和一种**Python**知道如何完成的操作（就是我们所说的名字）组成。“+”的意思是“加”。**Python**还理解其他类型的表达式，不一定是数字表达式。

```
>>> print 34.1/46.5
0.7333333333333334
>>> print 22 * 33
726
>>> print 14 - 15
-1
>>> print "Hello"
Hello
>>> print "Hello" + "Mark"
HelloMark
```

Python理解很多标准数学操作。它也知道如何识别不同种类或类型的数字：整数和浮点数。浮点数有小数点，整数没有小数点。**Python**还知道如何识别以双引号（"）开始和结束的字符串（字符序列）。它甚至知道如何将两个字符串“相加”：无非是把一个字符串放到另一个之后。

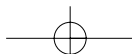
常见bug：Python的类型会产生奇特的结果

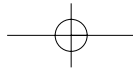


Python对待类型是严肃的。如果它看到你使用整数，那么它认为你想从表达式中得出一个整数结果。如果它看到你使用浮点数，那么它认为你想得到浮点数结果。听上去很合理，不是吗？那下面这种情况呢？

```
>>> print 1.0/2.0
0.5
>>> print 1/2
0
```

1/2结果为0？嗯，当然是的！1和2是整数。没有等于1/2的整数，因此结果只能是0！给整数加上一个“.0”就能向**Python**表明我们讨论的是浮点数，于是结果也就成了浮点数形式。





Python还理解函数（function）。还记得代数中讲的函数吗？它们是一种“盒子”，放进去一个值，出来另一个值。Python认识的函数当中有一个接收单个字符作输入值（放进盒子的值），返回或输出（从盒子里出来的值）一个数字，该数字是那个输入字符的ASCII映射码。这个函数的名字叫ord（ordinal），你可以用print显示ord函数返回的值：

```
>>> print ord("A")
65
```

另一个Python内置的函数叫abs——绝对值函数，它返回输入值的绝对值：

```
>>> print abs(1)
1
>>> print abs(-1)
1
```



调试技巧：常见的拼写错误

如果你输入了Python根本不理解的东西，那么就会得到一条语法错误提示。

```
>>> pint "Hello"
Your code contains at least one syntax
error, meaning it is not legal Jython.
```

如果你试图访问一个Python不知道的词语，Python会说它不认识那个名字。

```
>>> print a
A local or global name could not be found. You need to define
the function or variable before you try to use it in any way.
```

局部名字（local name）是函数内部定义的名字，全局名字（global name）是所有函数都能访问的名字（如pickAFile）。

JES认识的另一个函数允许你从磁盘上选择文件。你可能注意到了，不再说“Python认识”，而改说“JES认识”。print是所有Python实现都认识的东西，pickAFile却是为JES开发的。通常，你可以忽略这种区别，但如果你要尝试使用另一种Python，了解哪些部分是通用的、哪些部分不是通用的就很重要了。与ord不同，这个函数不需要输入，但会返回一个字符串，该字符串是你磁盘上某个文件的名称。这个函数的名字叫pickAFile。Python对大小写非常挑剔——写成pickafile或者Pickafile都不正确。试着用一下print pickAFile()，运行时你会看到图2.2所示的窗口。

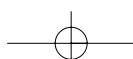
关于如何使用文件选择器或者文件对话框，你应该很熟悉了：

- 通过双击文件夹/目录打开它们。
- 通过点击选择文件，然后点击Open按钮；也可以直接双击文件。

选择了文件之后，pickAFile返回一个字符串（一系列字符）作为全路径文件名（full filename）。（如果单击了Cancel，pickAFile就返回空字符串——一个不包含任何字符的字符串，比如""。）尝试一下：运行print pickAFile()并Open一个文件。

```
>>> print pickAFile()
C:\ip-book\mediasources\beach.jpg
```

选择文件时最终得到的字符串形式与你使用的操作系统有关。在Windows中，文件名可能以C:开头且含有反斜杠（\）。在Linux或MacOS中则可能是这个样子：/Users/guzdial/ip-



book/medi asources/beach.jpg。这个文件名实际包含了两部分：

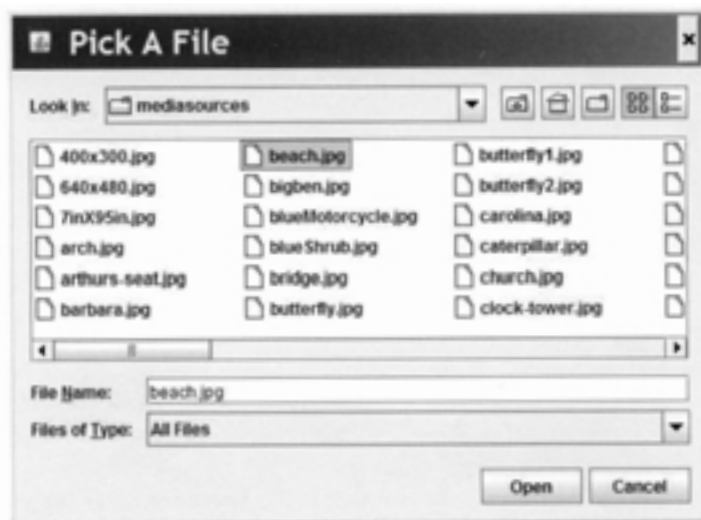


图2.2 文件选择器

- 各单词之间的字符（比如“Users”和“guzdial”之间的/）称为路径分隔符。从文件名开头到最后一个路径分隔符之间的所有部分称为到达文件的路径。它精确地描述了文件存在于磁盘上的位置（即存在于哪个目录中）。
- 文件名的最后一部分（如“beach.jpg”）叫做基本文件名（base filename）。当你在Finder、Explorer或Directory窗口（取决于你的操作系统）中查看文件时，看到的就是这一部分。最后三个字符（.之后的）叫做文件扩展名（file extension）。它们标识了文件使用的编码。扩展名为“.jpg”的文件是JPEG文件。这些文件包含图片内容。（更严格地讲，它们包含可以按图片的表示格式来解释的数据——不过，说它们“包含图片内容”也很接近这个意思了。）JPEG是一种标准编码（表示），可用于编码各种图像。我们还将经常使用的另外一种文件是“.wav”文件（如图2.3所示）。“.wav”扩展名表明它们是WAV文件。它们包含声音内容。

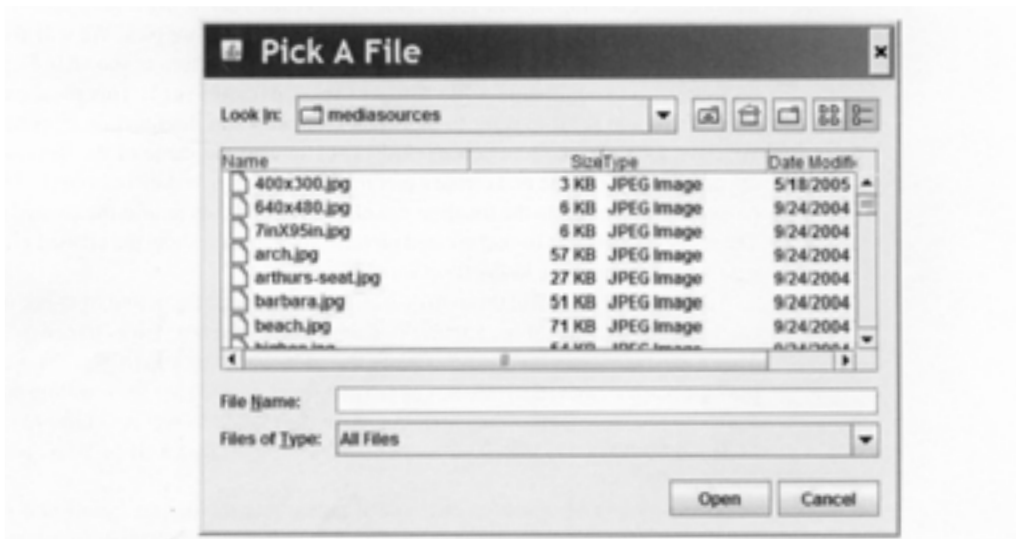
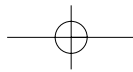


图2.3 标识出媒体类型的文件选择器



WAV是声音的标准编码。还有很多其他类型的文件扩展名，甚至许多其他类型的媒体文件扩展名。举例来说，还有表示图像的GIF（“.gif”）文件和表示声音的AIFF（“.aif”或“.aiff”）文件。为简单起见，本书只考察JPEG和WAV。

2.4.1 显示图片

现在，我们知道了如何获取一个完整的文件名：路径和基本名。但这不表明我们把文件本身加载到内存中了。为了把文件加载到内存中，我们必须告诉JES如何解释它。我们知道JPEG文件是图片，但我们必须显式地告诉JES读入文件并从中构造一幅图片。针对这一功能也有一个函数，叫做makePicture。

makePicture也需要一个参数（parameter）——即函数的输入。与ord一样，在括号中指定函数的输入。makePicture函数接受一个文件名。运气不错——我们知道如何获取一个文件名。

```
>>> print makePicture(pickAFile())
Picture, filename C:\ip-book\mediasources\barbara.jpg
height 294 width 222
```

print函数的结果显示：基于指定的文件名，以及高度、宽度，我们确实构造了一幅图片。成功！哦，你想直接看到真正的图片？那我们需要另一个函数（计算机很蠢的，我们有没有在别处提过？）显示图片的函数叫show。show同样接受一个参数——一个Picture。

但现在我们有问题了。我们没有为刚刚创建的图片取个名字，因此无法再次引用它。我们不可说“显示一秒钟前创建但没有命名的那幅图片”。除非我们给它命名，否则计算机根本记不住任何东西。为一个值创建名字的过程也称为声明变量。

让我们重新来一次，这一次首先为选取的文件命名。我们还会为创建的图片命名，然后我们就能显示命名的图片，如图2.4所示。



图2.4 选取、构造并显示一幅图片，所有的值都被命名

你可以使用 `file = pickAFile()` 来选择并命名一个文件。其含义是创建一个有名字的 `file` 并使之引用 `pickAFile()` 函数返回的值。我们已经知道 `pickAFile()` 函数能返回文件的名字（包括路径）。我们可以使用 `pict = makePicture(file)` 来创建一幅图片并为之命名，它把文件名传给 `makePicture` 函数，并返回创建的图片。名字 `pict` 引用创建的图片。然后我们可以把名字 `pict` 传给函数 `show`，以此来显示创建出来的图片。

另外一种方法是用一个函数完成所有动作，因为一个函数的输出可以作为另一个函数的输入：`show(makePicture(pickAFile()))`。在图2.5中可以看到这种用法。它让你从选择一个文件开始，把文件的名字传给 `makePicture` 函数，然后再把结果图片传给 `show` 函数。但这次我们没给图片命名，因此无法再次引用它了。

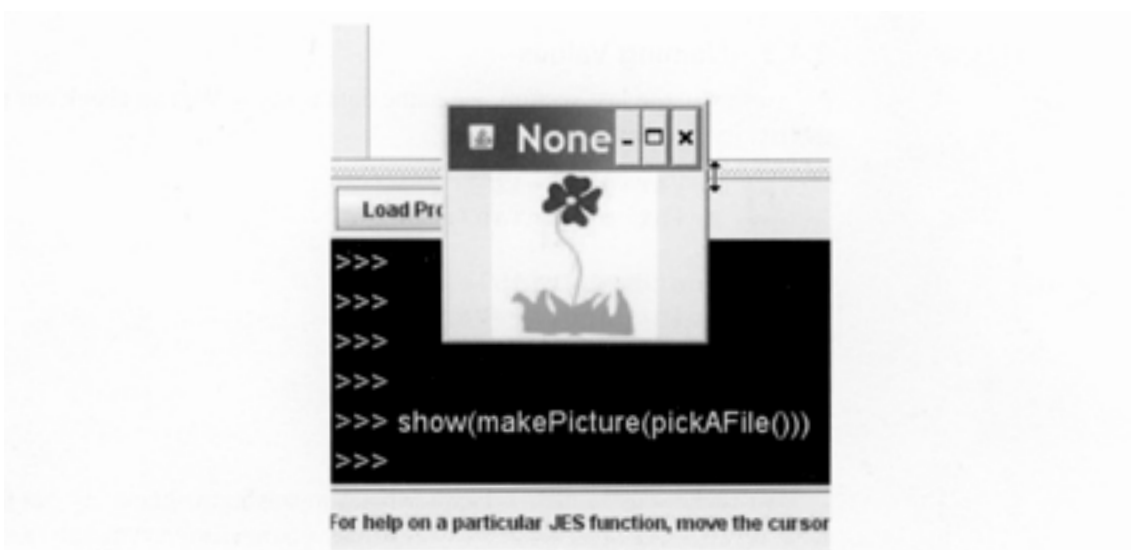


图2.5 选取、构造并显示一幅图片，每个函数直接用做下一个函数的输入

自己动手把两种方法都试一下吧。恭喜！你已经完成了第一次媒体计算！

倘若试一下 `print show(pict)`，你会发现 `show` 输出的是 `None`。与实际的数学函数不同，Python 中的函数不一定要有返回值。只要函数能完成一些功能（比如在窗口中打开一幅图片），即便没有返回值，它也是有用的。计算机科学家使用术语副作用（side-effect）来描述一个函数完成其他计算，而不是从输入到返回值计算的情况。

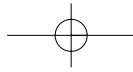
2.4.2 播放声音

我们可以用声音媒体来重复整个过程：

- 仍然用 `pickAFile` 找到想要的文件并取得文件名。这次选取的是以 `.wav` 结尾的文件。
- 这次用 `makeSound` 来构造声音。如你所想，`makeSound` 接受一个文件名作为输入。
- 将使用 `play` 来播放声音。`play` 接收声音值作为输入，返回 `None`。

以下的步骤与我们前面显示图片时是一致的：

```
>>> file = pickAFile()
>>> print file
C:\ip-book\mediasources\hello.wav
>>> sound = makeSound(file)
```



```
>>> print sound

Sound of length 54757
>>> print play(sound)
None
```

(我们将在下一章解释声音长度的含义。)请运行一下这些命令,使用自己的计算机上自己制作的,或者从<http://www.mediacomputation.org>下载的JPEG文件和WAV文件。(关于到哪里获取媒体,以及如何创建它们,后续章节会有更多讨论。)

2.4.3 数值命名

从上一节可以看到,使用“=”来命名数据。我们可以用**print**来检查命名,正如前面做的那样。

```
>>> myVariable=12
>>> print myVariable
12
>>> anotherVariable=34.5
>>> print anotherVariable
34.5
>>> myName="Mark"
>>> print myName
Mark
```

不要把“=”读作“等于”,那是它在数学中的含义。在这里,我们用的不是这个含义。应该把它读作“成为……的名字”。于是, `myVariable = 12` 的意思是:“`myVariable`成为12的名字”。因此反过来写(表达式放左边,名字放右边)是无意义的:这样一来, `12 = myVariable` 岂不成了“12成为`myVariable`的名字”了。

```
>>> x = 2 * 8
>>> print x
16
>>> 2 * 8 = x
Your code contains at least one syntax error, meaning
it is not legal Jython.
```

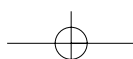
可以多次使用一个名字。

```
>>> print myVariable
12
>>> myVariable="Hello"
>>> print myVariable
Hello
```

名字和相应数据之间的绑定(或关联)仅在以下情况之前存在:(a)名字被赋予其他数据;(b)退出JES。名字和数据(甚至名字和函数)之间的关系仅存在于一次JES会话(session)中。

记住:数据拥有编码和类型。数据在表达式中的行为方式部分地取决于它的类型。留意一下整数12和字符串“12”在下面的操作中会有怎样的不同。针对其类型来说,它们都完成了合理的动作,但却是截然不同的动作。

```
>>> myVariable=12
>>> print myVariable*4
```



```

48
>>> myOtherVariable="12"
>>> print myOtherVariable*4
12121212

```

也可以把名字赋予函数执行的结果。如果为`pickAFile`函数命名结果，那么每次打印这个名字时都将得到同样的结果。但没有重新执行`pickAFile`。另外，为代码命名以便重新执行它是定义函数时所做的事情，几页之后便会讲到。

```

>>> file = pickAFile()
>>> print file
C:\ip-book\mediasources\640x480.jpg
>>> print file
C:\ip-book\mediasources\640x480.jpg

```

在下面的例子中，把名字赋予文件名和图片。

```

>>> myFilename = pickAFile()
>>> print myFilename
C:\ip-book\mediasources\barbara.jpg
>>> myPicture = makePicture(myFilename)
>>> print myPicture
Picture, filename barbara.jpg
height 294 width 222

```

注意，代数的替换（substitution）和求值（evaluation）概念在这里同样有效。`myPicture = makePicture(myFilename)`与`makePicture(pickAFile())`创建的图片是完全一样的[⊖]，因为我们让`myFilename`等于`pickAFile()`的结果。名字在表达式求值的时候被替换为相应的值。`makePicture(myFilename)`表达式求值时被展开为`makePicture("C:/ip-book/mediasources/barbara.jpg")`，因为“C:/ip-book/mediasources/barbara.jpg”是`pickAFile()`求值时我们选取的文件，而返回值被命名为`myFilename`。

我们还可以用函数返回的值来替换函数调用（invocation或call）而保持结果不变。`pickAFile`返回一个字符串——双引号括起来的一串字符。我们可以把上一个例子改写成下面这样，效果仍然不变。

```

>>> myFilename = "C:/ip-book/mediasources/barbara.jpg"
>>> print myFilename
C:/ip-book/mediasources/barbara.jpg
>>> myPicture = makePicture(myFilename)
>>> print myPicture
Picture, filename C:/ip-book/mediasources/barbara.jpg
height 294 width 222

```

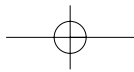
甚至替换掉名字。

```

>>> myPicture = makePicture("C:/ip-book/mediasources/
    barbara.jpg")
>>> print myPicture
Picture, filename C:/ip-book/mediasources/barbara.jpg
height 294 width 222

```

⊖ 当然，假定选取的是同一个文件。



计算机科学思想：名字、值和函数可以相互替换

值、赋予该值的名字和返回相同值的函数可以相互替换。计算机关心的是值，而不关心它究竟来自字符串、名字还是函数调用。关键在于计算机要对值、名字和函数求值（**evaluate**）。只要这些表达式可以求出相同的字符串，那么它们就可以相互替换。

事实上，我们不需要每次让计算机做点什么的时候都使用**print**。如果调用一个不返回任何东西的函数（这样的函数也**print**不出有用的东西），那么我们可以输入函数的名字及其输入（如果有的话）来调用它，然后直接按**Enter**键。

```
>>> show(myPicture)
```

这些让计算机做事的语句，我们常常称之为命令。**print myPicture**就是一条命令。**myFilename = pickAFile()**和**show(myPicture)**也是命令。它们不仅是表达式：它们让计算机做事情。

2.5 构建程序

我们已经会用名字来表示值。表达式求值的时候名字被替换成相应的值。对程序也可以做同样的事情。我们可以给一系列命令取个名字，然后每次想执行这些命令时使用这个名字即可。在**Python**中，我们为命令定义的名字将是一个函数。于是，**Python**中的程序就是由一个或多个执行有用任务的函数组成的集合。我们仍将使用术语菜谱（**recipe**）来描述执行有用媒体计算的程序（或程序的某些部分），虽然有时候它所涵盖的东西本身不足以成为有用程序。

还记得我们之前说过的吗？计算机上几乎所有的东西都可以命名。我们已经见过了为数值命名，现在看一下为菜谱命名。



实践技巧：尝试每一份菜谱

要真正理解每一份菜谱做了什么，就应该输入、加载并执行书中的每一份菜谱。强调一下，是每一份。它们都不长，但在确信程序能正常工作、开发编程技能和理解程序为什么能够工作等方面，这些实践大有裨益。

针对定义新菜谱，**Python**所理解的名字是**def**。**def**不是函数，它是一个命令，就像**print**一样。**def**用于定义新的函数。不过，单词**def**之后还必须跟上特定的内容。与**def**命令相关的内容结构称为命令的语法（**syntax**），为了让**Python**理解这里是什么以及这些内容的次序而必须出现的单词和字符。

def之后需要在同一行上有三样东西：

- 要定义的菜谱名字，比如**showMyPicture**。
- 菜谱接受的所有输入。菜谱可以是接受输入的函数，像**abs**和**makePicture**。各个输入要有名字，且置于括号中以逗号（,）分隔。如果没有输入，只需输入一对括号“**()**”来表示。
- 整行定义以冒号（:）结尾。

def行之后便是每次执行菜谱时将会执行的命令，一条接着一行。通过定义命令块，我们可以创建由一组命令组成的集合。函数的名字所关联的命令就包含在**def**命令（或语句）之后的命令块中。

大多数完成有用功能的实际程序，特别是那些创建用户界面的程序，都需要定义多个函数。想象一下程序区中有多条**def**命令的情况。你觉得Python会如何确定上一个函数的结束和下一个函数的开始呢？（特别要考虑函数内部可以定义其他函数的可能。）Python需要一种确定函数体（**function body**）结束的方法，即确定哪些语句是这个函数的一部分，哪些是下一个函数的一部分。

答案是缩进（**indentation**）。属于一个函数定义的所有语句相对于**def**语句都要缩进一点。我们建议使用不多不少两个空格来缩进——足以看清，便于记忆且简单易行。（在JES中，你还可以使用跳格键（按一次**Tab**键）来缩进。）你可以像下面这样在程序区输入函数（这里的“**□**”表示单个空格，即按一次空格键输入的字符）：

```
def hello():  
    print "Hello"
```

现在，我们可以定义自己的第一份菜谱了。可以把下面的程序输入JES程序区。输完之后保存文件，使用扩展名“.py”来表示这是个Python文件。（我们自己使用的名字是pickAndShow.py。）



程序1：选取并显示图片

```
def pickAndShow():  
    myFile = pickAFile()  
    myPict = makePicture(myFile)  
    show(myPict)
```

输入程序的时候，你会注意到函数体的四周出现的一个浅蓝色框。这个蓝色框显示的就是程序中的块（如图2.6所示）。与包含光标（接受输入处的竖杠）的语句处于同一块中的所有命令都圈在同一个蓝框中。如果你希望处于同一块的所有命令都出现在同一个蓝框中，那么你能知道自己的缩进是正确的。

输入完菜谱并保存之后，你就可以加载它了。点击Load Program按钮。

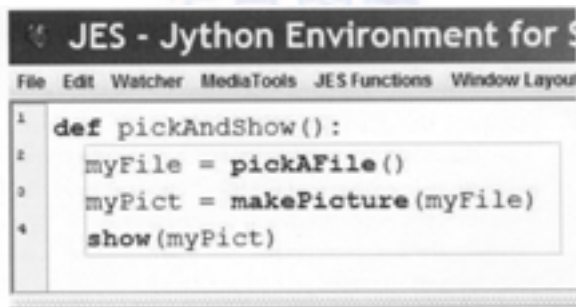


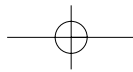
图2.6 JES中的可视化命令块



调试技巧：别忘了加载

使用JES时，最常见的错误就是输入并保存了函数，然后还没有加载它就在命令区尝试使用函数。你需要点击Load Program按钮把函数加载进来，这样命令区才能使用它。

现在你可以执行自己的菜谱。点击一下命令区。你的菜谱既不接受输入也不返回值（也就是说，这不是严格数学意义上的函数），所以只需要把它的名字作为一条命令输入就可以了：



```
>>> pickAndShow()
>>>
```

可以用类似的方法定义第二份菜谱，选取并播放一段声音。



程序2：选取并播放声音

```
def pickAndPlay():
    myFile = pickAFile()
    mySound = makeSound(myFile)
    play(mySound)
```



实践技巧：选用自己喜欢的名字

在上一节中，我们使用了名字`myFilename`和`myPicture`。这份菜谱中，我们使用了`myFile`和`myPict`。这有关系吗？对计算机来说无所谓。我们可以把图片命名为`myGlyph`甚至`myThing`。计算机并不关心你使用什么名字——名字完全是为你服务的。你应该选用这样的名字：(a) 对你有意义（这样你可以阅读并理解自己的程序）；(b) 对别人有意义（这样当你把程序展示给别人时，别人也能理解）；(c) 易于输入。像`myPictureThatIAmGoingToOpenAfterThis`这种包含30多个字符的名字虽然有意义且易于阅读，但输入的时候太痛苦了。

作为程序而言，这些菜谱可能没什么实际用处。如果你想显示同一幅图片，那么一遍遍地选取文件很烦人。既然有定义菜谱的能力，那么我们就可以定义新的菜谱来执行自己想要的任何动作。让我们定义一个打开特定图片的菜谱和另一个打开特定声音的菜谱。

使用`pickAFile`来获得你想要的声音或图片的文件名。我们在定义播放那段声音或显示那幅图片的菜谱时，将需要这个文件名。在这个菜谱中，我们直接把文件名字符串用双引号引起来，然后直接设置`myFile`的值，而不是使用`pickAFile`的结果。



程序3：显示特定图片

别忘了使用你自己图片文件的完整路径名替换下面程序中的`FILENAME`。比如“`C:/ip-book/mediasources/barbara.jpg`”。

```
def showPicture():
    myFile = "FILENAME"
    myPict = makePicture(myFile)
    show(myPict)
```

程序原理

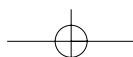
变量`myFile`接受了文件名的值——它与`pickAFile`函数返回的是同一个值（如果你选择那个文件的话）。然后，我们基于这个文件构造了一幅图片并命名为`myPict`。最后，我们显示了`myPict`中的图片。



程序4：播放特定声音

别忘了使用你自己声音文件的完整路径名替换下面程序中的`FILENAME`。比如“`C:/ip-book/mediasources/hello.wav`”。

```
def playSound():
    myFile = "FILENAME"
    mySound = makeSound(myFile)
    play(mySound)
```





常见bug: Windows文件名和反斜杠

Windows使用反斜杠（'\'）作为文件名分隔符。Python为某些反斜杠-字符的组合赋予了特殊含义，这一点我们后面会讲到更多。比如，'\n'的意思与Enter或Return键的输入是一样的。这些组合有可能自然地出现在Windows的文件名中。为避免Python错误地解读这些字符，你可以改用正斜杠（'/'），就像“C:\ip-book\mediasources\barbara.jpg”这样，或者你可以在文件名前输入一个“r”，就像：

```
>>> myFile = r"C:\ip-book\mediasources\barbara.jpg"
>>> print myFile
C:\\ip-book\\mediasources\\barbara.jpg
```



实践技巧：复制和粘贴

可以在程序区和命令区之间复制和粘贴文本。你可以用`print pickAFile()`来打印一个文件名，然后选中并复制（Copy，从Edit菜单中）它。然后点击命令区再粘贴（Paste）它。类似地，你可以从命令区把完整的命令复制到上面的程序区。这是一种非常便利的测试程序的方法：先测试各条命令，当次序和结果都正确的时候再放在一起做成一份菜谱。你还可以在命令区内部复制文本。选中一条命令，复制它，将它粘贴到最后一行（确定光标位于行末！），然后输入Enter键执行就可以了，不必每次运行同样的命令时都重新输入一遍。

可变菜谱：真正接受输入的一类数学函数

如何创建一个真正的函数呢？就像数学中接受输入的函数那样，比如`ord`和`makePicture`。我们又因何需要这样的函数呢？

使用变量来指定菜谱输入的一个重要原因是让程序更加通用。考虑程序3：`showPicture`。该程序针对一个特定的文件名。如果能有一个能接受任意文件名并从中构造、显示图片的函数会不会更有用呢？这种函数可以处理构造并显示图片的一般情形。我们将这种一般化的过程称为抽象（abstraction）。抽象可以带来适应多种情形的一般解决方案。

定义一个接受输入的菜谱非常容易。这依然是个替换与求值的问题。我们在`def`行的括号中放入一个名字。这个名字有时称为形参（parameter）或输入变量。

当你指定函数的名字，并在括号中给出输入值，也称为实参（argument），就像`makePicture(myFilename)`或`show(myPicture)`，在函数求值的时候，输入值就被赋值给输入变量。我们说输入变量接受（take on）了输入值。在函数（菜谱）执行期间，输入值将替换这个输入变量。

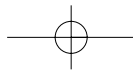
下面就是一个接受文件名作为输入变量的菜谱：



程序5：显示图片文件，文件的名称作为输入

```
def showNamed(myFile):
    myPict = makePicture(myFile)
    show(myPict)
```

点击Load Program按钮，让JES把函数读入程序区。如果函数中有错误，那么你需要改正它们，然后重新点击Load Program按钮。一旦成功加载了函数，那么你就可以在命令区使用它们。



26 · 第一部分 导 论

当你在命令区输入:

```
showNamed("C:/ip-book/mediasources/barbara.jpg")
```

并按Enter(回车)键, `showNamed`函数中的变量`myFile`就接收了该值:

```
"C:/ip-book/mediasources/barbara.jpg"
```

然后, `myPict`变量引用了读取并解释文件所得的结果, 然后图片就显示出来了。

可以用相同的方法创建一个播放声音的函数。我们可以在程序区输入多个函数。试着在前面的函数之后增加如下函数, 然后再次点击Load Program按钮。



程序6: 播放声音文件, 文件的名字作为输入

```
def playNamed(myFile):
    mySound = makeSound(myFile)
    play(mySound)
```

可在命令区输入以下命令来试用这个函数。

```
>>> playNamed("C:/ip-book/mediasources/croak.wav")
```

也可以创建接受多个参数的函数, 用逗号把各参数隔开即可。



程序7: 播放声音文件的同时显示图片

```
def playAndShow(sFile, pFile):
    mySound = makeSound(sFile)
    myPict = makePicture(pFile)
    play(mySound)
    show(myPict)
```

还可以编写接收图片或声音作为输入值的程序。下面是一个显示图片的程序, 但它接收图片对象作为输入, 而不是文件名。



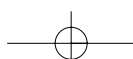
程序8: 显示作为输入传入的图片

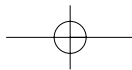
```
def showPicture(myPict):
    show(myPict)
```

这时你可能想把这些函数存入文件以便再次使用它们。点击File, 然后点击Save Program。这时会有一个文件对话框显示出来, 你可以指定文件的名字和存放位置。之后如果你退出并重新启动了JES, 那么你就可以用File菜单中的Open Program重新打开文件, 并点击Load Program来加载函数以供使用。

`showPicture`函数与JES内置的`show`函数有什么区别呢? 没有任何区别。我们当然可以创建一个函数来给另一个函数提供新名称。如果那样更便于你理解自己的代码, 那么它就是个好主意。

对函数来说, 恰当的输入值应该是什么呢? 输入文件名更好还是输入图片对象更好? 而这里的“更好”又是什么意思呢? 关于这些问题后面会有更多讨论。但这里先给一个简单答案: 编写对自己最有用的函数。如果对你来说定义`showPicture`比使用`show`的可读性更好, 那么这





样的定义就是有用的。如果你真正想要的是一个全权负责把图片构造出来并显示的函数，那么你可能觉得`showNamed`是最好用的。

编程摘要

本章讨论了以下几种数据（或对象）的编码：

整数（如：3）	没有小数点的数字——不能表示分数
浮点数（如：3.0、3.01）	可包含小数点的数字——可以表示分数
字符串（如："Hello!"）	一系列字符（包括空格、标点符号等），两端各有一个双引号
文件名	一个字符串，其中的字符表示了一个路径和一个基本文件名
图片	图像的编码，通常来自JPEG文件
声音	声音的编码，通常来自WAV文件

以下是本章介绍的程序片段：

<code>print</code>	以文本形式显示表达式（变量、值、算式等）的值
<code>def</code>	定义函数及其输入变量（如果有的话）
<code>ord</code>	返回与输入字符等价的数字值（根据ASCII标准）
<code>abs</code>	接受一个数字并返回其绝对值
<code>pickAFile</code>	让用户选取一个文件，并以字符串形式返回其完整文件名
<code>makePicture</code>	接受路径名作为输入，读入文件并基于文件内容构造图片，返回新构造的图片对象
<code>show</code>	显示作为输入传入的图片。不返回任何东西
<code>makeSound</code>	接受路径名作为输入，读入文件并基于文件内容构造声音，返回新构造的声音对象
<code>play</code>	接受声音对象并播放它。不返回任何东西

习题

2.1 计算机科学概念问题：

- 什么是算法？
- 什么是编码？
- 计算机如何以数字形式表示图片？
- 摩尔定律是什么？

2.2 `def`的含义是什么？语句`def someFunction(x, y):`做什么事情？

2.3 `print`的含义是什么？语句`print a`做什么事情？

2.4 `print 1 / 3`会输出什么结果？为什么会输出这样的结果？

2.5 `print 1.3 * 3`会输出什么结果？为什么会输出这样的结果？

2.6 `print 1.0 / 3`会输出什么结果？为什么会输出这样的结果？

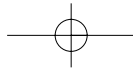
2.7 `print 10 + 3 * 7`会输出什么结果？为什么会输出这样的结果？

2.8 `print (10 + 3) * 7`会输出什么结果？为什么会输出这样的结果？

2.9 `print "Hi" + "there"`会输出什么结果？为什么会输出这样的结果？

2.10 以下命令的输出是什么？

```
>>> a = 3
>>> b = 4
>>> x = a * b
>>> print x
```

28 · 第一部分 导 论

2.11 以下命令的输出是什么？

```
>>> a = 3
>>> b = -5
>>> x = a * b
>>> print x
```

2.12 以下命令的输出是什么？

```
>>> a = 4
>>> b = 2
>>> x = a / b
>>> print x
```

2.13 以下命令的输出是什么？

```
>>> a = 4
>>> b = 2
>>> x = b - a
>>> print x
```

2.14 以下命令的输出是什么？

```
>>> a = -4
>>> b = 2
>>> c = abs(a)
>>> x = a * c
>>> print x
```

2.15 以下命令的输出是什么？

```
>>> name = "Barb"
>>> name = "Mark"
>>> print name
```

2.16 以下命令的输出是什么？

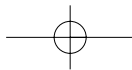
```
>>> a = ord("A")
>>> b = 2
>>> x = a * b
>>> print x
```

2.17 下面的代码导致了它后面的错误消息，试着改正它。

```
>>> pickafile()
The error was:pickafile
Name not found globally.
A local or global name could not be foun . You need
to define the function or variable before you try
to use it in any way.
```

2.18 下面的代码导致了它后面的错误消息，试着改正它。

```
>>> a = 3
>>> b = 4
>>> c = d * a
The error was:d
```



```
Name not found globally.  
A local or global name could not be found. You need  
to define the function or variable before you try  
to use it in any way.
```

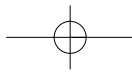
- 2.19 `show(p)`是做什么的？（提示：该问题的答案不止一个。）
- 2.20 试着在JES中使用其他一些字符串操作。如果把数字与字符串相乘，比如`3 * "Hello"`，结果会怎样？字符串乘以字符串，比如`"a" * "b"`，结果又会怎样？
- 2.21 当我们要执行名为`pickAFile`的函数时，会对表达式`pickAFile()`求值。那名字`pickAFile`本身又是什么呢？如果执行`print pickAFile`，会得到什么结果？执行`print makePicture`呢？打印出的内容是什么？你又是如何理解其含义的呢？

深入学习

最好（最深入、最具体、最优雅）的计算机科学教程是Abelson、Sussman和Sussman（这里不是重复，一个是Gerald Jay Sussman，另一个是Julie Sussman。参阅书后的“参考书目”。——译者注）合著的《Structure and Interpretation of Computer Programs》（中文版《计算机程序的构造和解释》，机械工业出版社。——译者注）[2]。读完这本书是一项颇具挑战性的任务，但绝对值得。另一本较新的书《How to Design Programs》（中文版《程序设计方法》，人民邮电出版社。——译者注）[12]更注重面向编程新手的内容，但主体思想与上一本是一致的。

然而，这两本书针对的都不是因为兴趣而编程或者因为要做一些小事而编程的学生。它们针对的都是未来的专业软件开发人员。对探索计算机的学生来说，最好的书是Brian Harvey编写的，《Simply Scheme》，该书使用的编程语言与Abelson等人的书一样，但更加易懂。这类书中，我们最喜欢的却是Harvey的三卷本《Computer Science Logo Style》（这里的“Logo”指的是一门编程语言。——译者注）[23]，这本书把有益的计算机科学与富有创造性的有趣项目有机地结合了起来。

华章图书



第3章

Introduction to Computing and Programming in Python: A Multimedia Approach, 2E

使用循环修改图片

本章学习目标

本章媒体学习目标：

- 理解如何利用人类的视觉限制把图片数字化。
- 识别不同的颜色模型，包括计算机中最常用的RGB。
- 处理图片中的颜色值，比如增加或减少红色值。
- 通过多种方法将彩色图片转换为灰度图片。
- 图片的颜色反转。

本章计算机科学学习目标：

- 使用矩阵表示查找图片中的像素。
- 使用图片对象和像素对象。
- 使用（基于for循环的）迭代改变图片中的像素颜色值。
- 代码块嵌套。
- 在有返回值的函数和仅提供副作用的函数之间做选择。
- 确定变量名的作用域。

3.1 图片的编码

图片（或图像）是所有媒体通信的重要组成部分。本章讨论图片在计算机上的表示方式（主要讲位图图像——单独表示每个点或像素），以及如何处理它们。下一章将介绍其他的图像表示法，比如向量图像（vector image）。

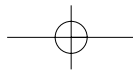
图片是二维的像素数组。这一节将描述所有这些术语。

就我们的目标而言，图片就是存储在JPEG文件中的图像。JPEG是关于如何以较少空间存储高品质图像的国际标准。JPEG是一种有损压缩（lossy compression）格式。这意味着它是压缩的，尺寸更小，但并不具有原始格式100%的品质。当然，通常舍弃掉的是你看不到或者注意不到的东西。就大多数应用而言，使用JPEG图像效果都很不错。

一维数组是类型相同的一系列元素。可以为数组命名，然后用下标来访问数组中的元素。数组中第一个元素的下标为0。在图3.1中，下标0处的值为15，下标1处的值为12，下标2处的值为13，下标3处的值为10。

二维数组也称为矩阵（matrix）。矩阵是以行和列的形式排列的一组元素的集合。这意味着要访问矩阵中的值可以同时指定行下标和列下标。

图3.2就是一个矩阵的例子。在坐标（0，1）（横，纵）处可以找到值为9的矩阵元素。（0，0）是15，（1，0）是12，（2，0）是13。我们会经常以（x，y）（横，纵）的形式来引用这些坐标。



0	1	2	3
15	12	13	10

图3.1 数组示例

	0	1	2	3
0	15	12	13	10
1	9	7	2	1
2	6	3	9	10

图3.2 矩阵示例

图片的每个元素中存储的是一个像素 (pixel)。pixel是“picture element”的缩写。它实际上就是一个点，而整幅图片就是由大量这样的点组成的。你有没有试过拿一个放大镜看报纸或杂志上的图片，或者看电视甚至显示器？当你看着杂志或电视上的图片时，它似乎并没有被分解成几百万个离散小点，但实际上它是的。

使用图片工具可以得到类似的一个个像素的视觉效果，如图3.3所示。这一工具能让你将图片放大到500%，于是每个像素都是可见的。要使用图片工具，方法之一是像下面的程序这样浏览图片。

```
>>> file = "c:/ip-book/mediasources/caterpillar.jpg"
>>> pict = makePicture(file)
>>> explore(pict)
```

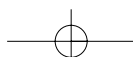


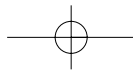
图3.3 显示在JES图片工具中的图像：左侧的显示比例是100%，右侧的显示比例是500%

我们人类的感官无法从事物的整体中分辨出极其细小的部分（除非凭借放大设备或其他特殊设备）。人类的视觉敏锐度 (acuity) 很低——比如说，我们看到的细节不如老鹰多。实际上，我们大脑和眼睛中的视觉系统不止一种。处理彩色的系统和处理黑白色（或者亮度）的系统是不一样的。比如，我们通过亮度来检测运动和物体的尺寸。实际上，眼睛的边缘部分比中心部分更适于拾取亮度。这是一种进化优势，它能让你提取到龇着利齿的老虎正从右边的灌木丛中悄悄地向你走来。

人类视觉分辨力的限制使图片的数字化成为可能。有些动物（比如，鹰和猫）能观察到比人类更多的细节，实际上，这些动物能看清图片上的单个像素。我们把图片分解成更小的元素（像素），但这些像素足够多且足够小，于是人们从整体上看时不会觉得断断续续。如果你能看清数字化的效果（比如，你可以看到某些位置的小矩形），那么我们把这种效果称为像素化 (pixelization)——即数字化过程显而易见的效果。

图片编码比声音编码更加复杂。声音本来是线性的——它沿着时间一直向前。图片却有两个维度：宽度和高度。





可见光是连续的——可见光的波长在370~730纳米(0.000 000 37~0.000 000 73米)之间。但我们对光的感知却受到色感器官(color sensor)工作方式的限制。我们的眼睛拥有几个色感器官,它们分别在感受到波长为425纳米(蓝)、550纳米(绿)和560纳米(红)左右的光时被触发(达到峰值)。我们的大脑基于眼睛中这三种色感的反馈来确定一种颜色。有些动物(比如狗)只有两种色感。这些动物仍然能感知颜色。但看到的颜色或感知的方式与人类不一样。关于我们能力有限的视觉感官,一个有趣的结果是:我们实际上能感知两种橙色。一种是光谱橙色——自然橙色的特定波长的光。另一种是红、黄的某种混合,当它冲击我们的色感器官时,使我们恰好感知成相同的橙色。

根据人类感知颜色的方式,只要能把冲击三种颜色感官的光线进行编码,我们就能记录人类所感知的颜色。于是,我们把每个像素编码成三个数字。第一个数字表示像素中的红色分量,第二个是绿色分量,第三个是蓝色分量。通过组合红、绿和蓝色的光,我们能构造出所有人类能看到的颜色(如图3.4所示)。三种颜色的光全部组合起来可以得到纯白色。三种全部关闭则是黑色。我们把这称为RGB颜色模型。

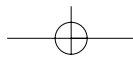
除了RGB颜色模型之外,还有其他定义和编码颜色的模型。有一种叫HSV颜色模型,它编码了色调(Hue)、饱和度(Saturation)和颜色值(Value)(有时也称为HSB颜色模型,三个字母分别表示色调、饱和度和亮度Brightness)。HSV模型的优点是,像颜色“调亮”或“调暗”这样的概念可以清晰地反映到这种模型上——比如,可以只调整饱和度(如图3.5所示)。另一种模型是CMYK颜色模型,它编码了青色(Cyan)、紫红色(Magenta)、黄色(Yellow)和黑色(black)(使用字母K而不是B是因为B会与“蓝色”(Blue)混淆)。CMYK是打印机使用的模型——4种颜色对应打印机混合产生颜色的4种墨水。然而,使用4种元素意味着需要在计算机上编码更多内容,因此,这种模型在媒体计算中不那么流行。RGB是计算机上最流行的模型。

像素的每种颜色分量(有时也称为颜色通道)通常都以单个字节表示,一个字节包含8位。8位可以表示256种模式(2^8):从00000000, 00000001, ..., 11111111。通常,我们使用这些模式来表示数值0~255。于是,每个像素就使用24位表示颜色。24位总共可以有 2^{24} 种可能的0/1组合模式,这意味着使用RGB模型的标准颜色编码可以表示16 777 216种颜色。实际上,我们确实能感知超过1 600万种颜色,但这并不重要。要完全复制我们能够看到的颜色空间,还没有哪种技术能接近这一目标。我们确实有能够表示1 600万种不同颜色的设备,但这1 600万种颜色也不能涵盖所有我们能够感知的颜色(或亮度)空间。因此,在技术取得进展之前,24位的RGB模型就足够了。



图3.4 融合红、绿、蓝产生新颜色

在有些计算机模型中,每个像素会使用更多的位。比如,有一种32位的模型使用另外的8位来表示透明度(transparency),即给定图像“底下”的颜色应该有多少与图像本身的颜色混合起来。这额外的8位有时称为alpha通道(alpha channel)。也有红、绿、蓝三个通道各使用超过8位的模型,但并不常见。



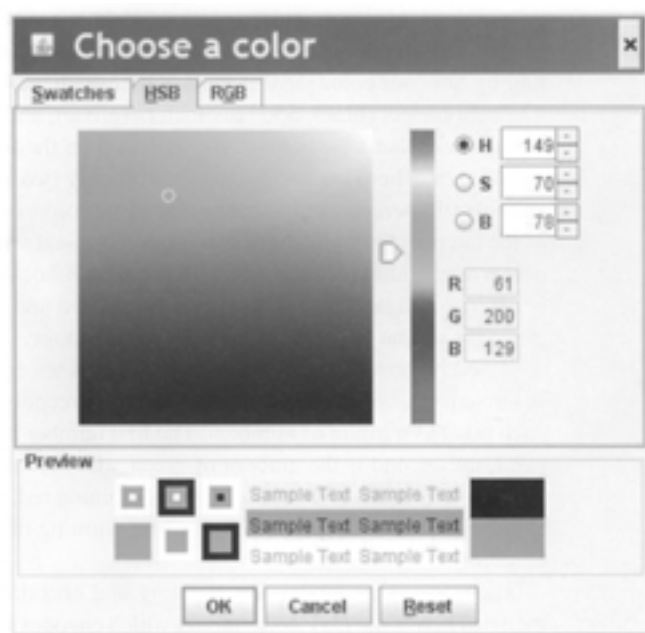


图3.5 基于HSB颜色模型选取颜色

实际上，我们有单独的一套视觉系统来感知物体的边缘、运动和厚度。我们通过一套系统来感知色彩，通过另一套系统来感知亮度（luminance，东西有多亮或多暗）。亮度实际上不是光量（amount of light），而是我们对光量的感知。我们可以度量光量（比如，基于颜色反射的光子数目）并验证一个红点与一个蓝点反射的光量是一样的，但我们还是感觉蓝色更暗。我们的亮度感基于物体与周围环境的对比。图3.6中的视觉错觉突出展现了我们是如何感知灰度级别的。左右两端实际上是同一灰度级，但因为中间的部分亮度和暗亮反差很大，所以我们感觉一端比另一端更暗。



图3.6 此图两端是同样的灰色，但中间的部分对比明显，因此左端看上去比右端更暗

大多数让用户选取颜色的工具都允许用户以RGB分量的形式指定颜色。JES中的颜色选取器（实际是Java Swing的标准颜色选取器）提供了一组滑块，可用于控制每种颜色的分量（如图3.7所示）。可以在命令区输入以下命令来选取颜色。

```
>>> pickAColor()
```

前面提过，三元组（0，0，0）（分别对应红、绿、蓝分量）是黑色，而（255，255，255）是白色。（255，0，0）是红色，（100，0，0）也是红色——只是更暗一些。（0，100，0）是中等亮度的绿色，而（0，0，100）是中等亮度的蓝色。当红、绿、蓝三种分量相同的时候，结果就是灰色。（50，50，50）是相当暗的灰色，而（100，100，100）更亮一些。

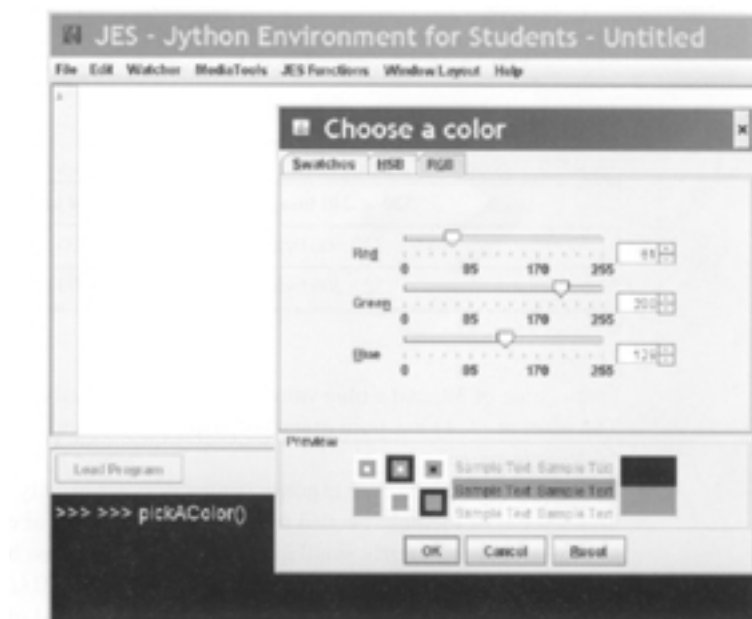


图3.7 JES中使用RGB滑块选取颜色

图3.8用一个矩阵表示像素RGB三元组。在图3.8中，(1, 0)处的像素颜色是(30, 30, 255)，意味着它有红色值30，绿色值30和蓝色值255——基本上它是一种蓝色，只是不纯。(2, 1)处的像素颜色是(150, 255, 150)，它有纯绿的颜色值，但也有更多的红和蓝，因此是一种很淡的绿色。

磁盘上甚至计算机内存中的图像通常都以压缩形式存储。即使一幅小图片，要表示它的每个像素也需要数量可观的内存（如表3.1所示）。一幅很小的 320×240 像素，每像素24位的图片占用的内存也有230 400字节——大约230 KB或1/4 MB。一台宽1024像素、高768像素的计算机显示器，每像素32位，要表示整屏画面需要3 MB。









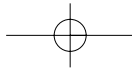
	0	1	2	3
0	 255, 30, 30	 30, 30, 255	 30, 255, 30	 0, 0, 0
1	 255, 150, 150	 150, 150, 255	 150, 255, 150	 200, 200, 200

图3.8 以矩阵表示的RGB三元组

表3.1 存储不同尺寸和格式的图片像素时所需的字节数

	320×240图像	640×480图像	1024×768图像
24位彩色	230 400字节	921 600字节	2 359 296字节
32位彩色	307 200字节	1 228 800字节	3 145 728字节



3.2 处理图片

在JES中，我们从JPEG文件中构造出图片对象，然后改变图片中各像素的颜色，以此来处理图片。为改变像素的颜色，我们需要处理像素的红、绿、蓝分量。

我们用makePicture函数来构造图片，使用show来显示图片。

```
>>> file = pickAFile()
>>> print file
C:\ip-book\mediasources\beach.jpg
>>> myPict = makePicture(file)
>>> show(myPict)
>>> print myPict
Picture, filename C:\ip-book\mediasources\beach.jpg
      height 480 width 640
```

makePicture函数完成的工作是：接收输入文件名中的所有字节，将它们放入内存，稍微再次调整格式，然后为它们打上一个标记，宣称“这是一幅图片！”执行myPict = makePicture(filename)命令就等于说：“那个图片对象（注意它上面的标记）现在叫做myPict”。

图片知道自己的宽度和高度，可以用getWidth和getHeight来查询。

```
>>> print getWidth(myPict)
640
>>> print getHeight(myPict)
480
```

有了图片对象，再给出目标像素的坐标，我们就可以用getPixel函数来获取图片中的任何一个像素。我们还可以用getPixels来获得包含所有像素的一维数组。一维数组从第一行的所有像素开始，后面跟着第二行的所有像素，以此类推。我们使用“[下标]”的形式引用数组元素。

```
>>> pixel = getPixel(myPict,0,0)
>>> print pixel
Pixel red=2 green=4 blue=3
>>> pixels = getPixels(myPict)
>>> print pixels[0]
Pixel red=2 green=4 blue=3
```

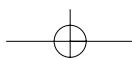


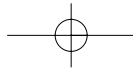
常见bug：不要输出像素数组，它太大了

getPixels毫无保留地返回了全部像素组成的数组。如果你试图输出getPixels函数的返回值，就会输出所有的像素。图片中的像素有多少呢？好吧，告诉你“beach.jpg”的宽640、高480。那会输出多少行呢？ $640 \times 480 = 307\,200$ ！307 200行的输出非常庞大。很可能你根本没有耐心等它结束。如果不小心做了这种事情，直接退出JES重启吧。

像素知道自己来自何处。可以用getX和getY询问它们的x和y坐标。

```
>>> print getX(pixel)
0
>>> print getY(pixel)
0
```





各个像素还知道如何getRed、setRed。(绿和蓝类似。)

```
>>> print getRed(pixel)
2
>>> setRed(pixel,255)
>>> print getRed(pixel)
255
```

也可以用getColor询问像素的颜色。还可以用setColor设置它的颜色。颜色对象知道自己的红、绿、蓝颜色分量。可以用函数makeColor来构造新的颜色。

```
>>> color = getColor(pixel)
>>> print color
color r=255 g=4 b=3
>>> newColor = makeColor(0,100,0)
>>> print newColor
color r=0 g=100 b=0
>>> setColor(pixel,newColor)
>>> print getColor(pixel)
color r=0 g=100 b=0
```

如果像素的颜色改变了，像素所属的图片也会改变。

```
>>> print getPixel(mypicture,0,0)
Pixel, color=color r=0 g=100 b=0
```



常见bug: 观看图片的变化

如果你显示了图片，然后改变了像素，你可能会疑惑为什么没看出任何不同。图片显示是不会自动更新的。对图片执行repaint，比如repaint(picture)，它才会更新。

也可以用pickAColor构造颜色，这个函数提供了多种选取颜色的方法。

```
>>> color2=pickAColor()
>>> print color2
color r=255 g=51 b=51
```

处理完一幅图片之后，可以用writePictureTo函数把它写到文件中去。

```
>>> writePictureTo(myPict,"C:/temp/changedPict.jpg")
```



常见bug: 以“.jpg”结尾

一定要用“.jpg”作为文件名的结尾，以便操作系统把它识别为JPEG文件。

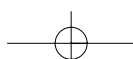


常见bug: 快速保存文件——以及怎样再次找到它

如果不知道所选目录的完整路径该怎么办呢？你可以仅指定文件基本名。

```
>>> writePictureTo(myPict,"new-picture.jpg")
```

问题在于如何再次找到这个文件。它保存到哪个目录里去了？这是个很容易解决的bug。默认目录（未指定路径时使用的目录）是JES所在的目录。如果不久前使用过pickAFile()，默认目录就是从中选取文件的目录。如果你有一个保存媒体并从中选取文件的标准媒体文件夹



(比如, Mediasources), 那么, 当未指定完整路径时, 它就是保存文件的目录。

我们无须编写新的函数来处理图片, 可以在命令区直接使用前面描述的函数。

```
>>> file="C:/ip-book/mediasources/caterpillar.jpg"
>>> pict=makePicture(file)
>>> show(pict)
>>> pixels = getPixels(pict)
>>> setColor(pixels[0],black)
>>> setColor(pixels[1],black)
>>> setColor(pixels[2],black)
>>> setColor(pixels[3],black)
>>> setColor(pixels[4],black)
>>> setColor(pixels[5],black)
>>> setColor(pixels[6],black)
>>> setColor(pixels[7],black)
>>> setColor(pixels[8],black)
>>> setColor(pixels[9],black)
>>> repaint(pict)
```

结果如图3.9所示, 图片的左上方显示了一条短短的黑线, 这条黑线的长度为10个像素。

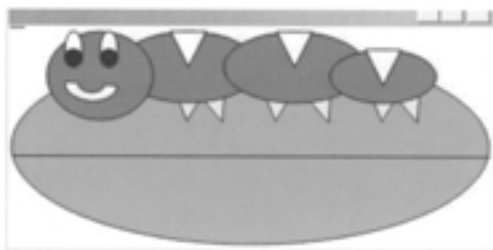


图3.9 使用命令直接修改像素颜色: 注意左上角的短小黑线



实践技巧: 使用JES帮助

JES拥有极好的帮助系统。忘记了自己需要哪个函数? 直接从JES的Help菜单上选择一项就行了(图3.10——全是超链接, 需要什么自己搜寻一下吧)。忘记了自己用过的某个函数是干什么的? 选中它, 然后选择Help菜单中的Explain就可以得到针对选中内容的解释(如图3.11所示)。

浏览图片

可以到<http://mediacomputation.org>上找到MediaTools应用程序以及关于如何使用它的文档。还可以在JES中找到MediaTools菜单。两种MediaTools都有一组图片浏览工具, 对研究图片非常有用。MediaTools应用程序如图3.12所示。JES图片工具如图3.13所示。

JES图片工具基于在命令区定义并命名的图片对象来工作。如果没有给图片命名, 就不能用JES图片工具来查看它。`p = makePicture(pickAFile())`定义一个图片对象并将它命名为p。然后你就能浏览图片了, 可以用`explore(p)`, 也可以用MediaTools菜单中的Picture Tool, 浏览的时候会有一个菜单弹出, 上面有当前可用的图片对象(基于其变量名), 可以选择其中的一个并点击OK按钮(如图3.14所示)。

JES图片工具允许你浏览图片。可以从Zoom菜单中选择一个级别来放大或缩小图片。在鼠标移过图片时按下鼠标按钮可以显示当前所指像素的(x, y)(横, 纵)坐标和RGB值(如

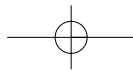


图3.13所示)。

Table of Contents > Understanding Pictures in JES > Picture Objects in JES	
Picture Objects in JES	
To understand how to work with pictures in JES, you must first understand the objects (or encodings) that represent pictures.	
You can imagine that each picture is made up of a collection of pixels , which is made up of pixel 1 , pixel 2 , pixel 3 , etc, and that each pixel has it's own particular color .	
Pictures	Pictures are encodings of images, typically coming from a JPEG file.
Pixels	Pixels are a sequence of Pixel objects. They flatten the two dimensional nature of the pixels in a picture and give you instead an arraylike sequence of pixels. pixels[1] returns the leftmost pixel in a picture.
Pixel	A pixel is a dot in the Picture. It has a color and an (x, y) position associated with it. It remembers its own Picture so that a change to the pixel changes the real dot in the picture.
Color	It's a mixture of red, green, and blue values, each between 0 and 255.
jump to top	

图3.10 JES帮助条目示例

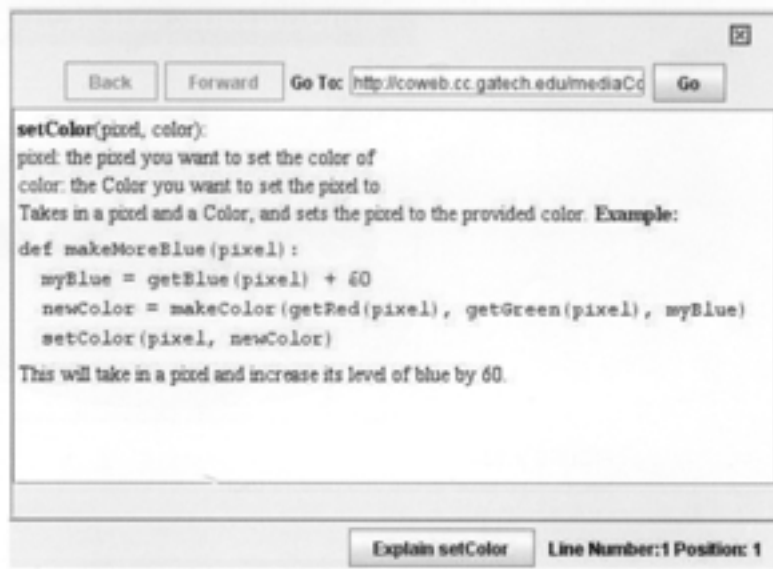


图3.11 JES Explain条目示例

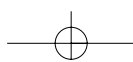




图3.12 使用MediaTools图像浏览工具

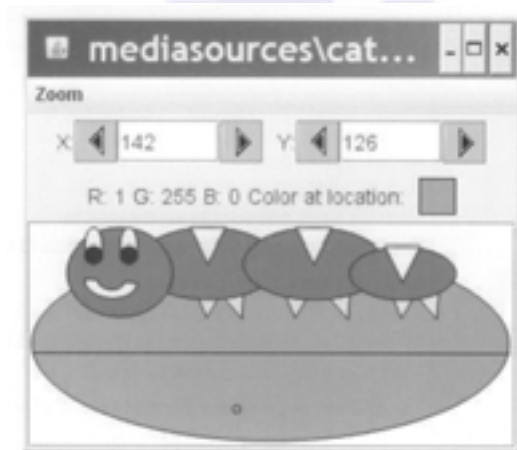


图3.13 在JES图片工具中选择像素

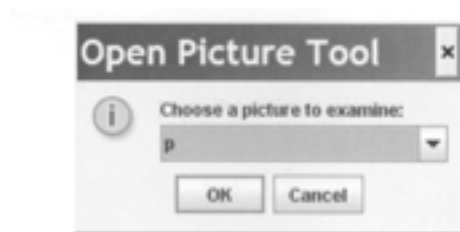
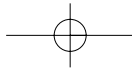


图3.14 在JES图片工具中选择图片

MediaTools应用程序基于磁盘文件来工作，而不是基于命令区中命名的图片来工作。如果你想在文件加载到JES之前先检查一下，那么可以用MediaTools应用程序做这件事。点击MediaTools中的Picture Tools就可以打开图片工具，可以用Open按钮打开一个文件选择框——



从左边点击想要浏览的目录，从右边点击想要浏览的文件，然后点击OK按钮。图像显示出来以后，还有多种不同的工具可供选用。把鼠标移到图片上并用鼠标按钮按下：

- 鼠标当前所指像素的红、绿、蓝颜色值将显示出来。如果你想对图片如何映射成红、绿、蓝颜色数值有一些感性认识，那么这一功能非常有用。如果你想对像素做些计算并检查结果值，这也很有帮助。
- 鼠标当前所指像素的x和y坐标将显示出来。当你想知道屏幕上一块区域的坐标范围时（比如，你只想处理图片的一部分）它很有用。如果你想处理某块区域而且知道这块区域的x和y坐标范围，那么就可以适当地安排一个for循环，只处理这一部分。
- 最后，还会显示出一个放大镜，让你看到像素逐渐放大的效果。（放大镜可以点击、拖曳。）

3.3 改变颜色值

最简单的图片处理是通过改变像素的红、绿、蓝分量来改变图片中像素的颜色值。只需对这些值做简单调整，就能得到迥然不同的图片效果。Adobe Photoshop的滤镜（filter）所做的工作就是我们在这一节要做的事情。

我们将要使用的颜色处理方法是基于原始颜色计算一个百分比。如果想得到原始图片中50%的红色数量，我们会把红色通道设置为当前值的0.5倍。如果想把红色增加25%，则可以把红色设为当前值的1.25倍。别忘记Python中的乘法运算符是星号（*）。

3.3.1 在图片上运用循环

我们可以获得图片中的各个像素并将它们的红、绿或蓝分量设置为新值。比如，我们想把红色减少50%，肯定可以这样编写代码：

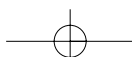
```
>>> file="C:/ip-book/mediasources/barbara.jpg"
>>> pict=makePicture(file)
>>> show(pict)
>>> pixels = getPixels(pict)
>>> setRed(pixels[0],getRed(pixels[0]) * 0.5)
>>> setRed(pixels[1],getRed(pixels[1]) * 0.5)
>>> setRed(pixels[2],getRed(pixels[2]) * 0.5)
>>> setRed(pixels[3],getRed(pixels[3]) * 0.5)
>>> setRed(pixels[4],getRed(pixels[4]) * 0.5)
>>> setRed(pixels[5],getRed(pixels[5]) * 0.5)
>>> repaint(pict)
```

但这么写太枯燥了，特别当针对所有像素的时候，哪怕对小图片也是一样。我们需要的是让计算机反反复复做同样一件事情的方法。当然，并非完全同样的事情——我们想以一种明确定义的方法来调整正在进行的过程。每次只运行一步，或者说只处理一个像素。

我们可以用for循环来达到这一目的。for循环针对（你提供的）数组中的每个项执行（你指定的）某个命令块，每次执行命令时，一个（你命名的）特别的变量将持有数组中不同元素的值。数组是数据的顺序集合。getPixels返回的是包含输入图片中所有像素对象的数组。

我们将写出下面这样的语句：

```
for pixel in getPixels(picture):
```



我们来详细讨论一下这里的每一部分。

- 首先是命令的名字**for**。
- 接下来是你想在寻址（**addressing**）（并处理）序列元素的代码中使用的变量名字。我们在这里使用单词**pixel**，因为我们想处理图片中的每个像素。
- 单词**in**是需要的——你一定要输入这个单词！输入它，命令的可读性更好，所以，多敲这4下键盘（空格-i-n-空格）是有益的。
- 接下来需要一个数组。在每一轮循环中，变量**pixel**将赋给数组中的每个元素：一个元素循环迭代一次。使用**getPixels**函数来产生这个数组。
- 最后，需要有个冒号（“:”）。这个冒号很重要——它表示后续内容是一个块（**block**）（你应该还记得上一章介绍的有关“块”的知识）。

接下来便是你想针对各个像素执行的命令。每次执行命令时，变量（在这个例子中是**pixel**）会引用数组中的不同元素。这些命令（称为循环体）作为一个块指定。这意味着它们应该跟在**for**语句之后，每条占一行，而且要比**for**语句多缩进两个空格！例如，下面就是把各像素的红色通道设置为原值一半的**for**循环。

```
for pixel in getPixels(picture):  
    value = getRed(pixel)  
    setRed(pixel,value * 0.5)
```

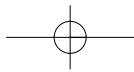
我们把这段代码查走一遍。

- 第一条语句表明我们将拥有一个**for**循环，它会把变量**pixel**设置为**getPixels(picture)**所输出数组的每个元素。
- 下一条语句向右缩进，因此它是**for**语句循环体的一部分——每次变量**pixel**拥有新值时将执行的语句之一（不论图片中的下一个像素是什么）。它取得当前像素的当前红色值并把它保存在变量**value**中。
- 第三条语句仍然向右缩进，因此它仍是循环体的一部分。在这一行，把名为**pixel**的像素的红色通道值设置为变量**value**乘以0.5的结果（使用**setRed()**函数）。这将使原值减少一半。

别忘了用来定义函数的**def**语句之后也是一个块。如果某函数中有一个**for**循环，那么**for**语句已经缩进两个空格了，因此**for**的循环体（循环中执行的命令）必须缩进4个空格。**for**循环的块位于函数块的内部。这叫做内嵌块（**nested block**）——一个块嵌套在另一个块中。下面的例子把循环语句变成了一个函数。

```
def decreaseRed(picture):  
    for pixel in getPixels(picture):  
        value = getRed(pixel)  
        setRed(pixel,value * 0.5)
```

实际上，不需要把循环放到函数中使用。可以在JES的命令区直接输入它们。JES很智能，如果输入的是循环命令，它能猜出需要输入多条命令，因此会将提示符“>>>”变为“...”。当然，它无法猜出什么时候结束，因此，需要直接按Enter键（不输入其他任何东西），以此告诉JES循环体已经结束。你可能意识到了，其实不需要变量**value**——可以用能求出相同值的函数调用来替代这个变量。在命令区这样输入：



```
>>> for pixel in getPixels(picture):
...     setRed(pixel, getRed(pixel) * 0.5)
```

我们已经明白了如何在不用编写上千行代码的情况下让计算机执行上千条命令，那就试试吧。

3.3.2 增/减红（绿、蓝）

处理数字图片的一种常见需求就是改变图片的红色度（或者绿色度、蓝色度——但更常见的是红色度）。可以把红色度提高，从而“暖化”图片，也可以降低它来“冷却”图片，或者应对红色过度的数码相机。

下面的菜谱将输入图片的红色数量减少50%。它用变量`p`代表当前像素。我们在上一个函数中使用了变量`pixel`。这无关紧要——名字可以随意取。



程序9：将图片中的红色数量减少50%

```
def decreaseRed(picture):
    for p in getPixels(picture):
        value=getRed(p)
        setRed(p, value*0.5)
```

把上面这段程序直接输入JES的程序区。点击Load Program按钮，以便让Python处理这个函数（一定要保存它，比如保存为DECREASERED.PY，或类似的名字），从而使名字`decreaseRed`代表这个函数。可以一步步执行下面的示例，进一步弄清这一切的原理。

这份菜谱接受一幅图片作为输入——我们将从这幅图片中取得像素。为了得到图片，需要一个文件名，然后需要基于文件构造一幅图片。也可以使用函数`explore(picture)`基于图片对象打开JES图片工具。它会将当前图片复制一份并在JES图片工具中显示它。对图片应用`decreaseRed`函数之后，可以再次浏览它并让两幅图片并排在一起进行比较。因此，菜谱可以这样使用：

```
>>> file="C:/ip-book/mediasources/Katie-smaller.jpg"
>>> picture=makePicture(file)
>>> explore(picture)
>>> decreaseRed(picture)
>>> explore(picture)
```



常见bug：耐心一点——`for`循环总会结束的

对这种代码来说，最常见的bug就是不等它自己结束就按下了Stop按钮。如果你使用的是`for`循环，程序总会结束的。但对于我们执行的一些操作，它可能需要运行整整一分钟（或者两分钟！）——特别是源图像很大的时候。

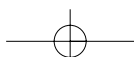
原始图片及其减少红色的版本如图3.15所示。50%显然是个不小的比例。图片看上去像是经过蓝色滤镜拍下的。注意第一个像素的红色量原来是133，后来变成了66。



跟踪程序：程序是如何执行的

计算机科学思想：跟踪是最重要的技能

编程实践中你能练就的最重要的一项技能就是跟踪程序（有时也称为程序的单



步跟踪)。跟踪程序就是一行一行地执行它，弄清楚每一行发生了什么。看一个程序时，你能预知它的功能吗？你应该能够逐行想出它的功能。

程序原理

让我们跟踪一下这个减少红色的函数，看看它的工作原理。我们想从刚刚调用 `decreaseRed` 的地方打断并开始跟踪。

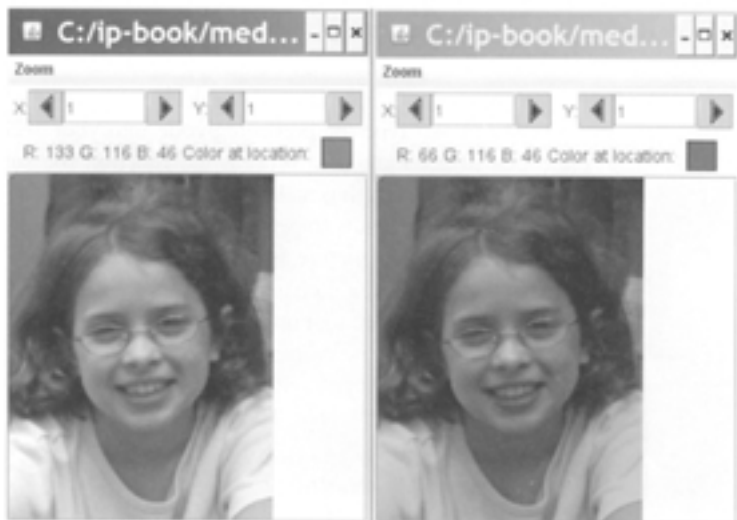


图3.15 原来的图片（左）和减少红色的版本（右）

```
>>> file="C:/ip-book/mediasources/Katie-smaller.jpg"
>>> picture=makePicture(file)
>>> explore(picture)
>>> decreaseRed(picture)
>>> explore(picture)
```

这时发生了什么？名字 `decreaseRed` 的确代表了之前看到的函数，于是它开始执行。

```
def decreaseRed(picture):
    for p in getPixels(picture):
        value=getRed(p)
        setRed(p,value*0.5)
```

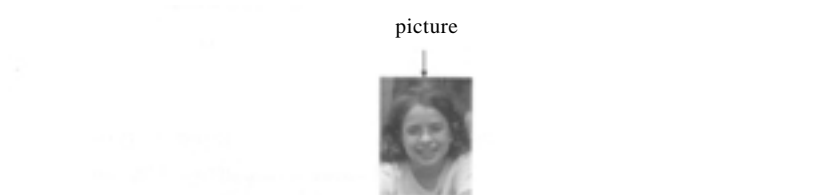
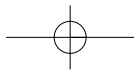
第一行执行的是 “`def decreaseRed(picture):`”。这一行指明了函数接受某种输入，而且在函数执行过程中输入将命名为 `picture`。



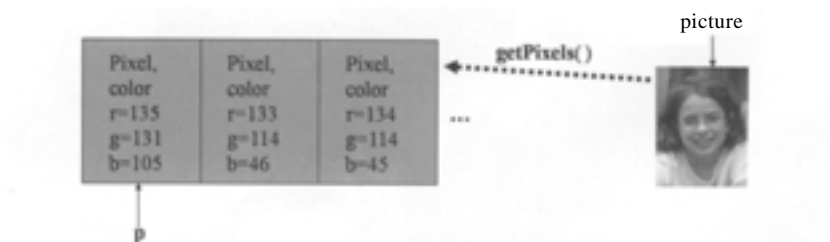
计算机科学思想：函数内部的名字与函数外面的名字是不一样的

函数内部的名字（如 `decreaseRed` 例子中的 `picture`、`p` 和 `value`）与命令区中的名字以及任何其他函数中的名字是完全不同的。我们说：它们有不同的作用域（`scope`）。

可以想象，这个时候计算机内部的样子：在单词 `picture` 和我们输入的图片对象之间存在某种关联。

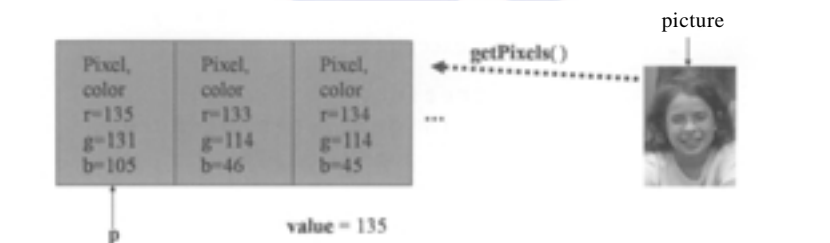


现在我们执行到了下一行：“`for p in getPixels(picture):`”。这一行的意思是：来自图片的所有像素都（在计算机内部）排成一个序列（位于数组中），且变量`p`应该赋予（关联到）第一个元素。可以想象，此时计算机内部就像这个样子：

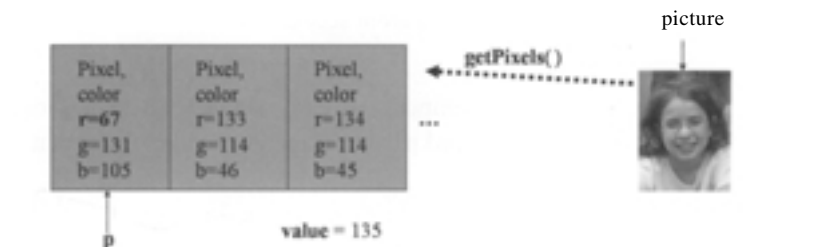


每个像素都有自己的RGB值。`p`指向第一个元素。注意变量`picture`还在那儿——我们用它或者不用它，它都在那里。

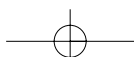
现在到了`value = getRed(p)`这一行。它只是把另一个名字加进了计算机正为我们跟踪的名字集合中，并为这个名字提供了一个简单的数字值。

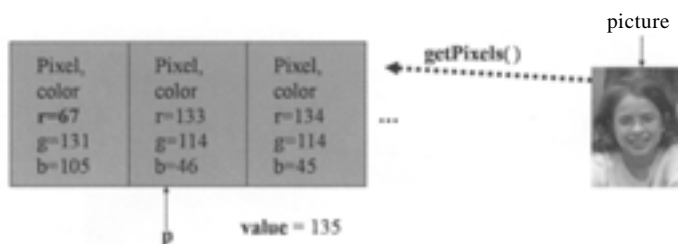


最后，我们到了循环的末尾。计算机执行`setRed(p, value * 0.5)`，把像素`p`的红色通道改为`value`的50%。`p`的值是奇数，乘以0.5会得到67.5，但我们把结果放进一个整数中，因此小数部分就被丢弃了。原先的红色值135变成了67。

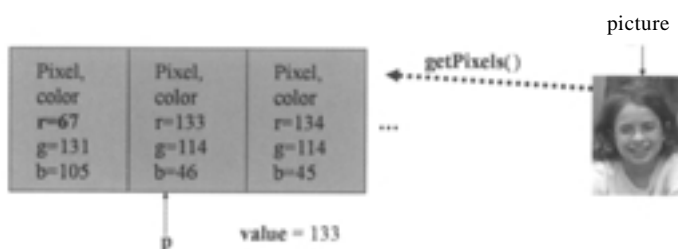


接下来发生的事情非常重要：循环又重新开始了！我们又回到了`for`循环并取出了数组中的下一个值。名字`p`与下一个值关联了起来。

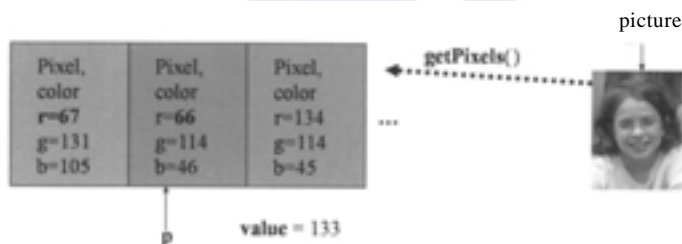




在`value = getRed(p)`这一步，变量`value`又获得了新值，现在它是133，而不是上次来自第一个像素的135。



然后，我们又改变了这个像素的红色通道。



最终，我们得到了图3.15。我们遍历了序列中的所有像素并修改了全部红色值。

3.3.3 测试程序：它真的能运行吗

我们如何知道自己所做的一切真正有效呢？当然，图片发生了一些变化，但我们真的把红色减少了吗？而且是50%？



实践技巧：不要轻易相信自己的程序

人们很容易误认为自己的程序已经正常工作了。毕竟，你让计算机做一些事，如果它做了你让它做的，那么你就不会觉得奇怪。但计算机真的很愚蠢——它们猜不出你想要什么。它们只是做你让它们做的事。你很容易得到“几乎正确”的结果。一定要检查一下。

检查程序的方法可以有多种。一种方法是使用JES的图片工具。用图片工具可以检查之前和之后的图片在同一 x 和 y 坐标处的RGB值。在之前的图片中点击并拖动光标到某个点来检查，然后在之后的图片中输入相同的 x 和 y 坐标并按Enter键。这样你可以同时看到之前和之后的图片在 x 和 y 坐标处的颜色值（如图3.16所示）。

我们也可以在命令区使用之前已知的函数来检查各个像素的红色数量。

```

>>> file = pickAFile()
>>> pict = makePicture(file)
>>> pixel = getPixel(pict,0,0)
>>> print pixel
Pixel, color=color r=168 g=131 b=105
>>> decreaseRed(pict)
>>> newPixel = getPixel(pict,0,0)
>>> print newPixel
Pixel, color=color r=84 g=131 b=105
>>> print 168 * 0.5
84.0

```



图3.16 使用JES图片工具让自己确信红色已经减少

3.3.4 一次修改一种颜色

现在让我们增加图片中的红色。如果将红色分量乘以0.5能使之减少，那么乘以一个大于1.0的数就能使之增加。



程序10：将红色分量增加20%

```

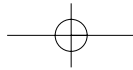
def increaseRed(picture):
    for p in getPixels(picture):
        value=getRed(p)
        setRed(p,value*1.2)

```

程序原理

就与使用`decreaseRed`一样，我们将使用同样的命令区（Command Area）语句来使用`increaseRed`。当我们输入`increaseRed(picture)`这样的命令时，会发生同样的过程。我们获得了输入图片`picture`的所有像素（不论它是什么样的图片），然后将变量`p`赋予列表中的第一个像素。我们取得它的红色值（假如说是100）并命名为`value`。我们将名字`p`当前所代表像素的红色值赋为 $1.2 * 100$ ，或者说120。然后我们针对输入图片中的每一个像素`p`重复这一过程。

如果一幅图片中的红色数量很多，增加它会导致某些红色结果值超出255，那么这时的情



况会怎样呢？针对这种情况有两种选择。结果可以截断为最大值255，也可以使用模（余数）运算符来回绕（wrap around）。例如，如果当前值为200而你试图将它倍增至400，那么它可以保持为255，或者绕回为144（ $400 - 256$ ）。尝试一下，看看会有什么效果。

JES提供了一个选项来控制截断颜色值为255还是回绕它。要改变此选项可以点击Edit菜单，然后点击Options。需要改变的选项是：Mouulo pixel color values by 256。

我们甚至可以完全消除一种颜色分量。下面的菜谱清除了一幅图片的蓝色分量。



程序11：清除图片的蓝色分量

```
def clearBlue(picture):  
    for p in getPixels(picture):  
        setBlue(p,0)
```

3.4 制作日落效果

我们当然可以同时完成多种图片处理动作。有一次，Mark想基于一幅海滩风景产生一种日落效果。他的第一次尝试是增加红色量，但没有达到效果。在一幅给定的图片中，某些像素的红色量已经很高了。如果某个通道的值超出255，那么默认结果是回绕。比如，通过setRed()函数把某个像素设置成256，实际会得到0。因此，增加红色会产生明亮的蓝、绿（没有红色）点。

Mark的第二种想法是：或许日落中的效果是蓝色和绿色更少，从而突出了红色，而不是实际增加了红色。下面是他针对这种想法编写的程序：



程序12：制作日落效果

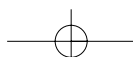
```
def makeSunset(picture):  
    for p in getPixels(picture):  
        value=getBlue(p)  
        setBlue(p,value*0.7)  
        value=getGreen(p)  
        setGreen(p,value*0.7)
```

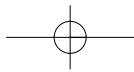
程序原理

与前面的例子一样，我们接受picture输入并用变量p表示输入图片中的各个像素。我们取得各个像素的蓝色分量并重设其值为原值乘以0.70的积。然后对绿色做同样的操作。结果，我们同时修改了绿色和蓝色通道——分别减少30%。效果非常不错，如图3.17所示。



图3.17 原来的海滩风景（左）和（虚假的）日落时的效果（右）





理解函数

此时此刻，关于函数你可能有许多问题要问。我们为什么以这种方式编写这些函数？我们为什么能同时在函数和命令区中重用`picture`这样的变量名？编写这些函数还有其它方法吗？函数有好坏之分吗？

由于我们总是先选取一个文件（或输入一个文件名），然后在调用某个文件处理函数之前先构造一幅图片，然后再显示或浏览图片，你自然会问：为什么不把这些固定下来？为什么不让每个函数直接包含`pickAFile()`和`makePicture()`？

我们使用函数的方式是使之更加通用或可重用。我们想让每个函数做一件事且只做一件事，于是我们可以在另一个需要做这件事情的上下文中再次使用这个函数。举个例子会更清楚。考虑一下制作日落效果的程序（程序12）。它通过将绿色和蓝色分别减少30%来达到效果。如果我们重写这个函数，让它调用两个更小的、分别只完成这两种操作的函数，那么结果会怎样呢？我们可能编写像程序13这样的代码。



程序13：使用三个函数制作日落效果

```
def makeSunset2(picture):  
    reduceBlue(picture)  
    reduceGreen(picture)  
  
def reduceBlue(picture):  
    for p in getPixels(picture):  
        value=getBlue(p)  
        setBlue(p,value*0.7)  
  
def reduceGreen(picture):  
    for p in getPixels(picture):  
        value=getGreen(p)  
        setGreen(p,value*0.7)
```

程序原理

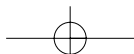
首先要弄清楚的是：这确实能达到了效果。这里的`makeSunset2`做的事情与之前的菜谱一样。`makeSunset2`函数接受一幅输入图片，然后使用同一幅输入图片调用了`reduceBlue`。`reduceBlue`使用`p`表示输入图片中的各个像素，并将每个像素的蓝色减少了30%（乘以0.7）。然后`reduceBlue`结束了，程序的控制流（即，接下来要执行的语句）返回`makeSunset2`函数并执行下一条语句。下一条语句是用同一幅输入图片调用函数`reduceGreen`。与前面一样，`reduceGreen`遍历每个像素并将绿色值减少30%。



实践技巧：使用多个函数

在同一程序区中使用多个函数并将它们保存在同一文件中完全没有问题。这将使函数的阅读和重用更加容易。

让一个函数（本例中是`makeSunset2`）使用程序员在同一文件中编写的其他函数（`reduceBlue`和`reduceGreen`）完全没有问题。你可以与之前一样使用`makeSunset2`。它还是同一份菜谱（让计算机做同样的事情），但使用了不同的函数。实际上，你也可以直接使用`reduceBlue`和`reduceGreen`——在命令区构造一幅图片并作为输入传给两个函数。它们用起来就与`decreaseRed`一样。



不同的是，`makeSunset2`的可读性更好一些，它清晰地表达了“制作日落效果就是减少蓝绿两种颜色”的意思。方便阅读真的很重要。



计算机科学思想：程序是以人为本的

计算机对程序的外表根本不关心。程序写出来是为了与人交流。程序易阅读易理解意味着它们易修改易重用，而且能更有效地将过程传达给其他人。

如果我们在`reduceBlue`和`reduceGreen`中加入`pickAFile`、`show`和`repaint`又会怎样？那样我们需要将图片提供两次——每个函数一次。把函数写成只减少蓝色或减少绿色（做且只做一件事），我们就可以在`makeSunset2`这样的新函数中使用它们。

现在，假如我们把`pickAFile`和`makePicture`放进`makeSunset`中。`reduceBlue`和`reduceGreen`函数再次变得完全灵活且可重用了，这很重要吗？没有，如果你只关心一个能给单幅图片带来日落效果的函数，那它就不重要。但如果以后你想制作一段几百帧的电影，为每一帧加入日落效果呢？你会愿意把那几百帧图像中的每一帧都取出来处理一遍？还是编写一个循环遍历它们（后面章节会学到），并将每一帧都作为输入传给更通用的`makeSunset`更好呢？这就是为什么我们要把函数做得通用且可重用——你永远不知道自己什么时候会在更大的上下文中再次使用一个函数。



实践技巧：开始时不要急着编写应用程序

新程序员常常想编写可供非技术用户使用的完整应用程序。你可能考虑编写一个`makeSunset`应用程序，运行时帮用户取得一幅图片并生成日落效果。构建大家都可以使用的一套良好用户界面并非易事。刚开始还是慢慢来比较好。编写一个接收图片输入的可重用函数已经够难的了。你可以将来再考虑用户界面。

也可以使用显式的文件名来编写这些函数，方法是在程序开头写：

```
file="C:/ip-book/mediasources/bridge.jpg"
```

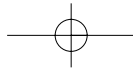
这样，程序就不会每次都提示我们提供一个文件，但函数也只能处理一个文件了。如果想让它处理其他文件，那么我们必须修改程序。你会愿意每次使用一个函数时都改动它一下吗？还是让函数保持独立，每次只改动传给它的图片更方便。

当然，我们可以把任何一个函数都改成传入文件名而不是传入图片。比如，我们可以写：

```
def makeSunset3(filename):
    reduceBlue(filename)
    reduceGreen(filename)

def reduceBlue(filename):
    picture = makePicture(filename)
    for p in getPixels(picture):
        value=getBlue(p)
        setBlue(p,value*0.7)

def reduceGreen(filename):
    picture=makePicture(filename)
    for p in getPixels(picture):
        value=getGreen(p)
        setGreen(p,value*0.7)
```



比起之前的代码，这样做好还是不好呢？从某种层面来说，这无所谓——只要合理，我们可以使用图片，也可以使用文件名。不过以文件名作为输入的版本确实有几个缺点：首先，它不能达到目的！`picture`分别在`reduceGreen`和`reduceBlue`中构造出来，但后来没有保存，因此函数结束时它就丢失了。之前的`makeSunset2`版本（以及它调用的子函数）是通过副作用来生效的——函数没返回任何东西，但它直接改变了输入对象。

我们可以在每个函数结束时将文件保存到磁盘，从而修正图片丢失的问题，但这样一来函数就不是“做且只做一件事”了。另外，两次构造图片还有低效的问题。如果我们再添加保存动作，那就保存了图片两次，更低效了。还是那句话，最好的函数“做且只做一件事”。

像`makeSunset2`这样的大函数同样“做且只做一件事”。`makeSunset2`制作了一张看似日落时分的图片。它是通过减少绿色和蓝色来实现的。我们最终得到的是一个目标的层次结构——目标就是“做且只做一件事”。`makeSunset2`让其他两个函数各自完成一件事，从而完成了自己的一件事。我们把这种过程称为层次式分解（**hierarchical decomposition**）（将一个程序分解为更小的部分，然后继续分解更小的部分，直到得出一项容易实现的任务），这一机制非常强大，有了它你可以基于自己理解的部分来创建复杂的程序。

函数中的名字与命令区中的名字是完全隔离的。函数从命令区获得数据的唯一方法是把数据作为输入传递给函数。在函数内部，可以用任何想用的名字。首次在函数内部定义的名字（如上一例中的`picture`）和用来表示输入数据的名字（如`filename`）仅存在于函数执行的过程中。函数结束后这些变量名就不复存在了。

这实际是个优点。先前我们说过，命名对计算机科学家非常重要：从数据到函数，我们给各种各样的东西命名。但如果每个名字永远表示且仅表示一样东西，那么我们说不定会把名字用光。在自然语言中，单词在不同的上下文中可以表示不同的意思。（比如，“**What do you mean?**”是问“你要表达什么意思？”而“**You are being mean!**”的意思是“你太坏了！”）不同的函数就是不同的上下文——其中的名字可以与函数外的同一个名字含义不同。

有时你会把函数里面计算出来的结果返回给命令区或调用方。我们已经见过函数输出值的情况，比如`pickAFile`输出一个文件名。如果在函数内部执行`makePicture`，那么你可能考虑将函数中创建的图片输出到外面去。这可以使用`return`来实现，后面我们会详细讨论它。

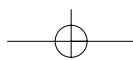
为函数的输入取的名字可以理解成占位符（**placeholder**）。看到占位符的时候，把它想象成输入数据就可以了。因此，像下面这样的函数：

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p,value*0.5)
```

调用的时候我们将使用`decreaseRed(myPicture)`这样的语句。不论`myPicture`里面是什么图片，在`decreaseRed`执行期间它的名字就是`picture`。在这几秒钟内，`decreaseRed`中的`picture`和命令区中的`myPicture`表示的是同一幅图片。改变一幅图片的像素也就改变了另一幅图片的像素。

我们刚刚讨论过了用不同方法实现功能相同的函数——有的方法好一点，有的方法差一点。编写函数还有其他方法，有些与前面的差不多，有些则好得多。让我们再来考察几种编写函数的方法。

我们可以一次传入多个输入。考虑下面这个`decreaseRed`版本：




```
def decreaseRed(picture, amount):
    for p in getPixels(picture):
        value=getRed(p)
        setRed(p,value*amount)
```

我们可以用`decreaseRed(myPicture, 0.25)`这样的命令来使用这个函数，它将把红色减少75%。我们可以用`decreaseRed(myPicture, 1.25)`把红色增加25%。或许，这个函数还是叫`changeRed`更好，因为它现在就是这样的——一种通用的改变图片中红色总量的方法。这是个很有用也很强大的函数。

还记得程序11中的这段代码吗？

```
def clearBlue(picture):
    for p in getPixels(picture):
        setBlue(p,0)
```

我们也可以用下面的方法写出同样的菜谱：

```
def clearBlue(picture):
    for p in getPixels(picture):
        value = getBlue(p)
        setBlue(p,value*0)
```

需要注意的是：这个函数与之前的菜谱做了完全一样的事情，它们都把所有像素的蓝色通道设置成0。后一个函数的优点之一在于它与我们看到的其他改变颜色的函数在形式上别无二致，从而更易于理解，这是有益的。效率上它略逊一筹——将蓝色值设置为0之前没必要先取得原来的值，要得到一个0也没必要把另一个值乘以0。这个函数的确做了一些没必要做的事情——它没有“做且只做一件事”。

3.5 亮化和暗化

将图片加亮或变暗非常容易。模式与之前的代码完全相同，但不是改变一种颜色分量，而是改变整体颜色。以下是图片亮化和暗化的菜谱。图3.18显示了原始图片和更暗的版本。



程序14：亮化图片

```
def lighten(picture):
    for px in getPixels(picture):
        color = getColor(px)
        color = makeLighter(color)
        setColor(px,color)
```

程序原理

变量`px`用于表示输入图片中的各个像素。（这次不是`p`！这重要吗？计算机无所谓——如果你觉得`p`表示像素，那就用`p`，但`px`、`pxl`甚至`pixel`也都可以，随便使用。）`color`接受了像素`px`的颜色值。`makeLighter`返回更亮的新颜色。`setColor`方法（面向对象术语中，与类或对象关联的函数常称为“成员函数”，也叫“方法”。作者这里冷不丁冒出“方法”一词，可能有些突兀。第16章将介绍面向对象编程。——译者注）将像素设成了更亮的新颜色。

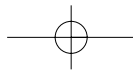


图3.18 原始图片（左）和更暗的版本（右）



程序15：暗化图片

```
def darken(picture):
    for px in getPixels(picture):
        color = getColor(px)
        color = makeDarker(color)
        setColor(px, color)
```

3.6 制作底片

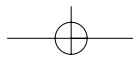
为图片制作一张底片像（negative image）比你想象的要容易得多。我们先来仔细考虑一下。想要达到的目标是把各像素的红、绿、蓝当前值都反一下。最容易理解的是边缘情形。倘若某个红色分量为0，我们想得到的是255。如果为255，我们则想把它变成0。

现在考虑中间地带。如果红色分量为淡红（比如50），我们想要的是一种近乎完全的红色——而“近乎”的程度应等同于原始图片中的红色数量。我们想要的就是比最大的红色值（255）小50的那种颜色，即 $255 - 50 = 205$ 的红色分量。一般地，反色应该是255减去原色。我们需要计算每一个红、绿、蓝分量的反色分量，然后创建一种新的反颜色，并把像素的值设置成这种颜色。

下面是完成这一功能的菜谱，你可以看到它真正达到了目标（如图3.19所示）。



图3.19 底片像





程序16: 创建原始图片的底片

```
def negative(picture):
    for px in getPixels(picture):
        red=getRed(px)
        green=getGreen(px)
        blue=getBlue(px)
        negColor=makeColor( 255-red, 255-green, 255-blue)
        setColor(px,negColor)
```

程序原理

用`px`表示输入图片的各个像素。对每个像素`px`，用变量`red`、`green`和`blue`来命名像素颜色的红、绿、蓝分量。用`makeColor`构造一种新颜色。它的红色分量是`255 - red`，绿色是`255 - green`，蓝色是`255 - blue`。也就是说，新颜色是原始颜色的反色。最后，把像素`px`的颜色设置成新的反色（`negColor`）并转向下一个像素。

3.7 转换到灰度

彩色转灰度是一个有趣的菜谱。它很短，理解起来也不难，视觉效果却很不错。这是个很好的例子：通过对像素颜色值的处理，我们能方便地实现强大的功能。

还记得吗？当红、绿、蓝三种颜色分量的值相同时，结果就是灰色。这意味着我们的RGB编码支持256个灰度级。从（0，0，0）（黑色）、（1，1，1）……（100，100，100）直到最后的（255，255，255）（白色）。需要技巧的地方在于计算出应该用哪一级来复制。

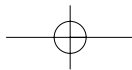
我们想要的是一种称为亮度（**luminance**）的色彩强度（**intensity**）。事实上，有一种很简单的方法可以计算这个结果：求出三种颜色的平均值。因为有三个分量，所以我们使用的强度公式为

$$\frac{(red + green + blue)}{3}$$

这引出了如下的简单菜谱以及图3.20。



图3.20 彩色图片转换成了灰度图片



程序17: 转换成灰度

```
def grayScale(picture):
    for p in getPixels(picture):
        intensity = (getRed(p)+getGreen(p)+getBlue(p))/3
        setColor(p,makeColor(intensity,intensity,intensity))
```

实际上, 这里的灰度概念过于简单了。下面的菜谱考虑了人眼如何感知亮度的因素。还记得吗? 即使反射的光量相同, 我们也觉得蓝色比红色更暗。因此, 在计算平均数时, 我们可以给蓝色更低的权值 (weight), 而给红色更高的权值。



程序18: 加权法转换为灰度

```
def grayScaleNew(picture):
    for px in getPixels(picture):
        newRed = getRed(px) * 0.299
        newGreen = getGreen(px) * 0.587
        newBlue = getBlue(px) * 0.114
        luminance = newRed+newGreen+newBlue
        setColor(px,makeColor(luminance,luminance,luminance))
```

程序原理

我们用`px`表示图片的各个像素。然后, 根据人们感知红、绿、蓝三种颜色亮度的相关实证研究, 我们为三种颜色分配不同的权重。注意, $0.299 + 0.587 + 0.114 = 1.0$ 。最终我们仍会得到0~255之间的一个值。但我们让亮度值较多地来自绿色部分, 较少来自红色, 更少来自蓝色 (我们已经讲过, 蓝色是我们感觉最暗的)。我们把三个加权值加在一起得到新的亮度值。最后我们构造了新的颜色值并把它设置为像素`px`的新颜色。

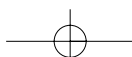
编程摘要

本章讨论了以下几种数据 (或对象) 的编码。

图片	图像的编码, 通常从JPEG文件中创建出来
像素数组	像素对象的一维数组 (序列)。 <code>pixels[0]</code> 返回位于图片左上角的像素
像素	图片中的一个点。它拥有颜色及与之关联的 (x, y) 位置。它记录着自己所属的图片, 因此改变像素也会改变图片中那个真实的点
颜色	红、绿、蓝三个值的混合, 每个值都在0~255之间

图片程序片段

<code>getPixels</code>	接收图片输入, 返回图片中像素对象组成的一维数组, 数组中首先是来自第一行的像素, 然后是来自第二行的像素, 以此类推
<code>getPixel</code>	接收一个图片对象和两个位置值 <code>x</code> 和 <code>y</code> (两个数字), 返回图片中相应点的像素对象
<code>getWidth</code>	接收图片输入, 返回以像素个数表示的图片宽度
<code>getHeight</code>	接收图片输入, 返回以像素个数表示的图片高度



(续)

<code>writePictureTo</code>	接收图片和文件名(字符串)作为输入,然后基于JPEG编码将图片写入文件(要保证文件名以“.jpg”结尾,从而操作系统能正确理解它)
<code>getRed</code> , <code>getGreen</code> , <code>getBlue</code>	这三个函数接受一个像素对象,分别返回像素中红色、绿色和蓝色数量(0~255之间)
<code>setRed</code> , <code>setGreen</code> , <code>setBlue</code>	这三个函数接收一个像素对象和一个值(0~255之间),分别将像素的红色、绿色和蓝色数量设置成给定的值
<code>getColor</code>	接收一个像素对象,返回此像素处的颜色对象
<code>setColor</code>	接收一个像素对象和一个颜色对象,为此像素设置颜色
<code>getX</code> , <code>getY</code>	接收一个像素对象,分别返回像素在图片中所处位置的x和y坐标

颜色程序片段

<code>makeColor</code>	接收三项输入,依次为红、绿和蓝色分量,返回一个颜色对象
<code>pickAColor</code>	不需要输入,显示一个颜色选择器,在上面找出你想要的颜色,函数会返回它
<code>makeDarker</code> , <code>makeLighter</code>	这两个函数接受一个颜色对象,分别返回比它稍暗一点或稍亮一点的版本

本章出现了许多有用的常量(`constant`)。常量是一些具有预定义值的变量。这些值都是颜色,如:`black`、`white`、`blue`、`red`、`green`、`gray`、`darkGray`、`lightGray`、`yellow`、`orange`、`pink`、`magenta`和`cyan`。

习题

3.1 图片概念问题:

- 为什么我们看不到图片中各个位置上的红、绿、蓝点?
- 层次式分解是什么?它适合做什么?
- 亮度是什么?
- 为什么任何颜色分量(红、绿或蓝)的最大值都是255?
- 我们使用的颜色编码是RGB,从表示图片所需的内存数量来看,这意味着什么?我们能够表示的颜色数量有限制吗?RGB能够表示的颜色数目够用吗?

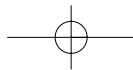
3.2 程序9显然减少了太多红色。试着重写一个只把红色减少10%的版本,然后再试着减少20%。你能找到分别适合用这些不同版本来处理图片吗?注意,你肯定可以反复不断地减少一幅图片中的红色,但你不会愿意重复得次数太多。

3.3 写出消减红色函数(程序9)的蓝色和绿色版本。

3.4 下面的两个函数都等价于增加红色的函数(程序10)。试着测试一下,确保它们是正确的。你更喜欢哪一个?为什么?

```
def increaseRed2(picture):
    for p in getPixels(picture):
        setRed(p, getRed(p)*1.2)

def increaseRed3(picture):
    for p in getPixels(picture):
        redComponent = getRed(p)
        greenComponent = getGreen(p)
```



```

blueComponent = getBlue(p)
newRed=int(redComponent*1.2)
newColor = makeColor
            (newRed,greenComponent,blueComponent)
setColor(p,newColor)

```

3.5 如果打开回绕选项并不断增加红色值，最后某些像素会变成光亮的绿色和蓝色。如果使用图片工具查看那些像素，你会发现其红色值非常低。你认为这是怎么回事呢？它们为何变得这么小？回绕的机制的工作原理是什么？

3.6 编写一个函数交换两种颜色值，比如交换红色值和蓝色值。

3.7 编写一个函数将红色、绿色和蓝色值都变成0，结果会怎样？

3.8 编写一个函数将红色、绿色和蓝色值都变成255，结果会怎样？

3.9 下面的函数实现了什么功能？

```

def test1 (picture):
    for p in getPixels(picture):
        setRed(p,getRed(p) * 0.3)

```

3.10 下面的函数实现了什么功能？



```

def test2 (picture):
    for p in getPixels(picture):
        setBlue(p,getBlue(p) * 1.5)

```

3.11 下面的函数实现了什么功能？

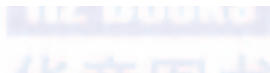


```

def test3 (picture):
    for p in getPixels(picture):
        setGreen(p,0)

```

3.12 下面的函数实现了什么功能？



```

def test4 (picture):
    for p in getPixels(picture):
        red = getRed(p)
        green = getGreen(p)
        blue = getBlue(p)
        color = makeColor
                (red + 10, green + 10, blue + 10)
        setColor(p,color)

```

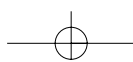
3.13 下面的函数实现了什么功能？

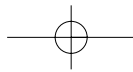
```

def test5 (picture):
    for p in getPixels(picture):
        red = getRed(p)
        green = getGreen(p)
        blue = getBlue(p)
        color = makeColor
                (red - 20, green - 20, blue - 20)
        setColor(p,color)

```

3.14 下面的函数实现了什么功能？





```
def test6 (picture):  
    for p in getPixels(picture):  
        red = getRed(p)  
        green = getGreen(p)  
        blue = getBlue(p)  
        color = makeColor(blue, red, green)  
        setColor(p,color)
```

3.15 下面的函数实现了什么功能?

```
def test7 (picture):  
    for p in getPixels(picture):  
        red = getRed(p)  
        green = getGreen(p)  
        blue = getBlue(p)  
        color = makeColor  
            (red / 2, green / 2, blue / 2)  
        setColor(p,color)
```

3.16 编写函数将一幅图片变成灰度图,然后再将它制成底片。

3.17 编写三个函数,一个清除蓝色(程序11),一个清除红色,另一个清除绿色。在实际应用中,哪一个会是最有用的?它们的组合呢?

3.18 重写清除蓝色的函数(程序11),这次把蓝色最大化(即设置成255),而不是清除它。这个函数有用吗?将红色和绿色最大化的版本呢?有用吗?什么情况下有用呢?

3.19 编写函数changeColor,接受三个输入:一幅图片、一个增加或减少的量度以及一个用1、2或3分别表示红、绿、蓝的数字。量度值将在-0.99~0.99之间。

- changeColor(pict, -0.10, 1)应把图片中的红色数量减少10%。
- changeColor(pict, 0.30, 2)应把图片中的绿色数量增加30%。
- changeColor(pict, 0, 3)应当不影响图片中的蓝色数量(红或绿也不影响)。

深入学习

关于视觉的原理以及艺术家是怎样学着处理它,有一本极好的书,那就是Margaret Livingstone编写的《Vision and Art: the Biology of Seeing》[28]。

Python计算与编程实践 多媒体方法 (原书第2版)

Introduction to Computing and Programming in Python A Multimedia Approach Second Edition

计算机程序设计课程往往是枯燥乏味的,而本书根据教育理论,开发了一种新的教学方法,介绍如何通过多媒体编程来学习计算机程序设计,将趣味性和实用性融于枯燥的编程课程的教学当中。本书方法独特新颖,实例通俗易懂,可帮助读者快速入门并深入掌握编程技能,是一本理论联系实际的程序设计教程。

本书使用的编程语言是Python,这是因为Python强大实用(比如网站开发)、易于入门,计算机专业和非专业人士都可以学习。本书以Python数字多媒体编程为主线,从图像、声音、文本和电影这些学生已经熟知的内容开始,讲解它们的处理方法,其中穿插介绍计算机科学与程序设计的基础知识。在讲解知识点的时候也独具匠心,先介绍容易理解的基本概念,如赋值、顺序操作、迭代、条件式、函数定义等,逐步涉及抽象内容,如算法复杂度、程序效率、计算机组成、层次式分解、递归、面向对象等。本书还提供了大量例题和习题,方便教学。

作者简介

Mark Guzdial 是佐治亚理工学院计算机学院交互式计算专业的教授。他是ACM国际计算机教育研究系列研讨会的创立者之一,ACM教育委员会副主席,“Journal of the Learning Sciences”和“Communications of the ACM”编委会委员。Guzdial博士主要关注计算机教育方面的研究。他的第一本著作论述Squeak语言及其在教育中的应用。他是Swiki(Squeak Wiki)的早期开发者,Swiki是第一个专门用于学校的wiki。他出版了多本关于利用多媒体编程环境学习计算机编程的著作,影响了世界各地的计算机本科生教学。

Barbara Ericson 是佐治亚理工学院计算机学院“计算机普及”课程的主管和研究人员。她从2004年开始就致力于改善计算机基础教育,现在是计算机科学教师协会的师范教育代表,美国女性信息技术中心K-12联盟的合作主席,计算机科学AP考试开发委员会成员。她的研究兴趣涉及计算机图形学、人工智能和面向对象编程等多个领域。

客服热线: (010) 88378991, 88361066
购书热线: (010) 68326294, 88379649, 68995259
投稿热线: (010) 88379604
读者信箱: hzsj@hzbook.com

华章网站 <http://www.hzbook.com>

网上购书: www.china-pub.com

封面设计: 包逸 杨杉

PEARSON

www.pearson.com

上架指导: 计算机/程序设计

ISBN 978-7-111-38738-1



9 787111 387381

定价: 69.00元