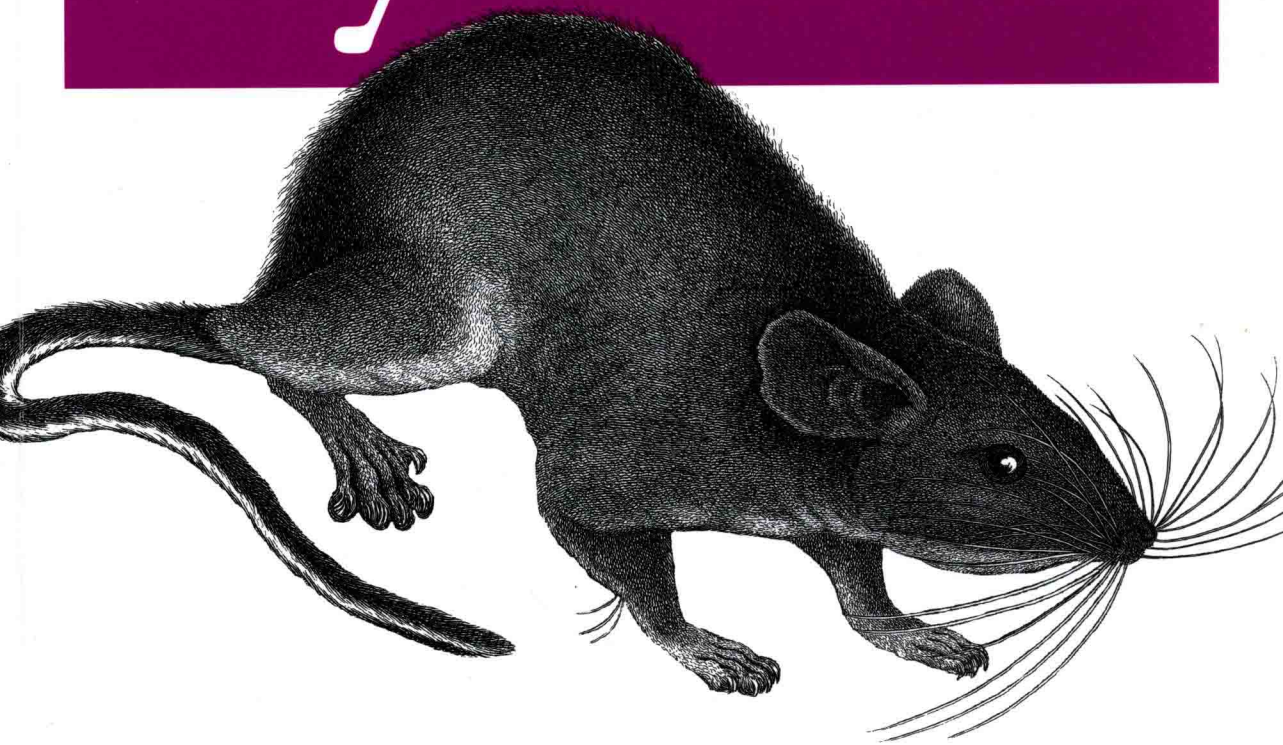
学习Python（影印版）

第五版
上册

Learning
Python



O'REILLY®
東南大學出版社

Mark Lutz 著

（第五版／上册）

学习Python（影印版）

Learning Python

Mark Lutz 著

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

南京　东南大学出版社

图书在版编目（CIP）数据

学习 Python：第 5 版：英文 /（美）鲁特兹（Lutz, M.）著 . —影印本 . —南京：东南大学出版社，2014.1

书名原文：Learning Python, 5E

ISBN 978-7-5641-4597-2

I. ① 学… II. ①鲁… III. ①软件工具－程序设计－英文 IV. ① TP311.56

中国版本图书馆 CIP 数据核字（2013）第 246109 号

江苏省版权局著作权合同登记

图字：10-2013-369 号

学习 Python 第五版（影印版）

出版发行：东南大学出版社
地　　址：南京四牌楼 2 号　　邮编：210096
出 版 人：江建中
网　　址：http://www.seupress.com
电子邮件：press@seupress.com
印　　刷：扬中市印刷有限公司
开　　本：787 毫米 × 980 毫米　16 开本
印　　张：101.75
字　　数：1993 千字
版　　次：2014 年 1 月第 1 版
印　　次：2014 年 1 月第 1 次印刷
书　　号：ISBN 978-7-5641-4597-2
定　　价：148.00 元（上下册）

本社图书若有印装质量问题，请直接与营销部联系。电话（传真）：025-83791830

To Vera.

You are my life.

Preface

If you're standing in a bookstore looking for the short story on this book, try this:

- *Python* is a powerful multiparadigm computer programming language, optimized for programmer productivity, code readability, and software quality.
- *This book* provides a comprehensive and in-depth introduction to the Python language itself. Its goal is to help you master Python fundamentals before moving on to apply them in your work. Like all its prior editions, this book is designed to serve as a single, all-inclusive learning resource for all Python newcomers, whether they will be using Python 2.X, Python 3.X, or both.
- *This edition* has been brought up to date with Python releases 3.3 and 2.7, and has been expanded substantially to reflect current practice in the Python world.

This preface describes this book's goals, scope, and structure in more detail. It's optional reading, but is designed to provide some orientation before you get started with the book at large.

This Book's "Ecosystem"

Python is a popular open source programming language used for both standalone programs and scripting applications in a wide variety of domains. It is free, portable, powerful, and is both relatively easy and remarkably fun to use. Programmers from every corner of the software industry have found Python's focus on developer productivity and software quality to be a strategic advantage in projects both large and small.

Whether you are new to programming or are a professional developer, this book is designed to bring you up to speed on the Python language in ways that more limited approaches cannot. After reading this book, you should know enough about Python to apply it in whatever application domains you choose to explore.

By design, this book is a tutorial that emphasizes the *core Python language* itself, rather than specific applications of it. As such, this book is intended to serve as the first in a two-volume set:

- *Learning Python*, this book, teaches Python itself, focusing on language fundamentals that span domains.
- *Programming Python*, among others, moves on to show what you can do with Python after you've learned it.

This division of labor is deliberate. While application goals can vary per reader, the need for useful language fundamentals coverage does not. Applications-focused books such as *Programming Python* pick up where this book leaves off, using realistically scaled examples to explore Python's role in common domains such as the Web, GUIs, systems, databases, and text. In addition, the book *Python Pocket Reference* provides reference materials not included here, and it is designed to supplement this book.

Because of this book's focus on foundations, though, it is able to present Python language fundamentals with more depth than many programmers see when first learning the language. Its bottom-up approach and self-contained didactic examples are designed to teach readers the entire language one step at a time.

The core language skills you'll gain in the process will apply to every Python software system you'll encounter—be it today's popular tools such as Django, NumPy, and App Engine, or others that may be a part of both Python's future and your programming career.

Because it's based upon a three-day Python training class with quizzes and exercises throughout, this book also serves as a self-paced introduction to the language. Although its format lacks the live interaction of a class, it compensates in the extra depth and flexibility that only a book can provide. Though there are many ways to use this book, linear readers will find it roughly equivalent to a semester-long Python class.

About This Fifth Edition

The prior *fourth edition* of this book published in 2009 covered Python versions 2.6 and 3.0.[1] It addressed the many and sometimes incompatible changes introduced in the Python 3.X line in general. It also introduced a new OOP tutorial, and new chapters on advanced topics such as Unicode text, decorators, and metaclasses, derived from both the live classes I teach and evolution in Python "best practice."

This *fifth edition* completed in 2013 is a revision of the prior, updated to cover both *Python 3.3 and 2.7*, the current latest releases in the 3.X and 2.X lines. It incorporates

1. And 2007's short-lived third edition covered Python 2.5, and its simpler—and *shorter*—single-line Python world. See *http://www.rmi.net/~lutz* for more on this book's history. Over the years, this book has grown in size and complexity in direct proportion to Python's own growth. Per Appendix C, Python 3.0 alone introduced 27 additions and 57 changes in the language that found their way into this book, and Python 3.3 continues this trend. Today's Python programmer faces two incompatible lines, three major paradigms, a plethora of advanced tools, and a blizzard of feature redundancy—most of which do not divide neatly between the 2.X and 3.X lines. That's not as daunting as it may sound (many tools are variations on a theme), but all are fair game in an inclusive, comprehensive Python text.

all language changes introduced in each line since the prior edition was published, and has been polished throughout to update and sharpen its presentation. Specifically:

- *Python 2.X* coverage here has been updated to include features such as dictionary and set comprehensions that were formerly for 3.X only, but have been back-ported for use in 2.7.
- *Python 3.X* coverage has been augmented for new `yield` and `raise` syntax; the *__pycache__* bytecode model; 3.3 namespace packages; PyDoc's all-browser mode; Unicode literal and storage changes; and the new Windows launcher shipped with 3.3.
- *Assorted* new or expanded coverage for JSON, `timeit`, PyPy, `os.popen`, generators, recursion, weak references, `__mro__`, `__iter__`, `super`, `__slots__`, metaclasses, descriptors, `random`, Sphinx, and more has been added, along with a general increase in 2.X compatibility in both examples and narrative.

This edition also adds a new *conclusion* as Chapter 41 (on Python's evolution), two new *appendixes* (on recent Python changes and the new Windows launcher), and one new *chapter* (on benchmarking: an expanded version of the former code timing example). See Appendix C for a concise summary of *Python changes* between the prior edition and this one, as well as links to their coverage in the book. This appendix also summarizes initial differences between 2.X and 3.X in general that were first addressed in the prior edition, though some, such as new-style classes, span versions and simply become mandated in 3.X (more on what the X's mean in a moment).

Per the last bullet in the preceding list, this edition has also experienced some growth because it gives fuller coverage to more *advanced language features*—which many of us have tried very hard to ignore as optional for the last decade, but which have now grown more common in Python code. As we'll see, these tools make Python more powerful, but also raise the bar for newcomers, and may shift Python's scope and definition. Because you might encounter any of these, this book covers them head-on, instead of pretending they do not exist.

Despite the updates, this edition retains most of the structure and content of the prior edition, and is still designed to be a comprehensive learning resource for both the 2.X and 3.X Python lines. While it is primarily focused on users of Python 3.3 and 2.7—the latest in the 3.X line and the likely last in the 2.X line—its historical perspective also makes it relevant to *older* Pythons that still see regular use today.

Though it's impossible to predict the future, this book stresses fundamentals that have been valid for nearly two decades, and will likely apply to *future* Pythons too. As usual, I'll be posting Python updates that impact this book at the book's website described ahead. The "What's New" documents in Python's manuals set can also serve to fill in the gaps as Python surely evolves after this book is published.

The Python 2.X and 3.X Lines

Because it bears heavily on this book's content, I need to say a few more words about the Python 2.X/3.X story up front. When the *fourth edition* of this book was written in 2009, Python had just become available in two flavors:

- Version 3.0 was the first in the line of an emerging and incompatible mutation of the language known generically as *3.X*.
- Version 2.6 retained backward compatibility with the vast body of existing Python code, and was the latest in the line known collectively as *2.X*.

While 3.X was largely the same language, it ran almost no code written for prior releases. It:

- Imposed a Unicode model with broad consequences for strings, files, and libraries
- Elevated iterators and generators to a more pervasive role, as part of fuller functional paradigm
- Mandated new-style classes, which merge with types, but grow more powerful and complex
- Changed many fundamental tools and libraries, and replaced or removed others entirely

The mutation of `print` from statement to function alone, aesthetically sound as it may be, broke nearly every Python program ever written. And strategic potential aside, 3.X's mandatory Unicode and class models and ubiquitous generators made for a different programming experience.

Although many viewed Python 3.X as both an improvement and the future of Python, Python 2.X was still very widely used and was to be supported in parallel with Python 3.X for years to come. The majority of Python code in use was 2.X, and migration to 3.X seemed to be shaping up to be a slow process.

The 2.X/3.X Story Today

As this *fifth edition* is being written in 2013, Python has moved on to versions 3.3 and 2.7, but this 2.X/3.X story is still largely *unchanged*. In fact, Python is now a dual-version world, with many users running *both* 2.X and 3.X according to their software goals and dependencies. And for many newcomers, the choice between 2.X and 3.X remains one of existing software versus the language's cutting edge. Although many major Python packages have been ported to 3.X, many others are still 2.X-only today.

To some observers, Python 3.X is now seen as a *sandbox* for exploring new ideas, while 2.X is viewed as the *tried-and-true* Python, which doesn't have all of 3.X's features but is still more pervasive. Others still see Python 3.X as the future, a view that seems supported by current core developer plans: Python 2.7 will continue to be supported but is to be the last 2.X, while 3.3 is the latest in the 3.X line's continuing evolution.

On the other hand, initiatives such as *PyPy*—today a still 2.X-only implementation of Python that offers stunning performance improvements—represent a 2.X future, if not an outright faction.

All opinions aside, almost five years after its release, 3.X has yet to supersede 2.X, or even match its user base. As one metric, 2.X is still downloaded more often than 3.X for Windows at python.org today, despite the fact that this measure would be naturally skewed to *new* users and the *most recent* release. Such statistics are prone to change, of course, but after five years are indicative of 3.X uptake nonetheless. The existing 2.X software base still trumps 3.X's language extensions for many. Moreover, being last in the 2.X line makes 2.7 a sort of *de facto standard*, immune to the constant pace of change in the 3.X line—a positive to those who seek a stable base, and a negative to those who seek growth and ongoing relevance.

Personally, I think today's Python world is large enough to accommodate *both* 3.X and 2.X; they seem to satisfy different goals and appeal to different camps, and there is precedence for this in other language families (C and C++, for example, have a long-standing coexistence, though they may differ more than Python 2.X and 3.X). Moreover, because they are so similar, the skills gained by learning either Python line transfer almost entirely to the other, especially if you're aided by dual-version resources like this book. In fact, as long as you understand how they diverge, it's often possible to write code that runs on both.

At the same time, this split presents a substantial *dilemma* for both programmers and book authors, which shows no signs of abating. While it would be easier for a book to pretend that Python 2.X never existed and cover 3.X only, this would not address the needs of the large Python user base that exists today. A vast amount of existing code was written for Python 2.X, and it won't be going away anytime soon. And while some newcomers to the language can and should focus on Python 3.X, anyone who must use code written in the past needs to keep one foot in the Python 2.X world today. Since it may still be years before many third-party libraries and extensions are ported to Python 3.X, this fork might not be entirely temporary.

Coverage for Both 3.X and 2.X

To address this dichotomy and to meet the needs of all potential readers, this book has been updated to cover *both* Python 3.3 and Python 2.7, and should apply to later releases in both the 3.X and 2.X lines. It's intended for programmers using Python 2.X, programmers using Python 3.X, and programmers stuck somewhere between the two.

That is, you can use this book to learn *either* Python line. Although 3.X is often emphasized, 2.X differences and tools are also noted along the way for programmers using older code. While the two versions are largely similar, they diverge in some important ways, and I'll point these out as they crop up.

For instance, I'll use 3.X `print` calls in most examples, but will also describe the 2.X `print` statement so you can make sense of earlier code, and will often use portable printing techniques that run on both lines. I'll also freely introduce new features, such as the `nonlocal` statement in 3.X and the string `format` method available as of 2.6 and 3.0, and will point out when such extensions are not present in older Pythons.

By proxy, this edition addresses other Python version 2.X and 3.X releases as well, though some older version 2.X code may not be able to run all the examples here. Although class decorators are available as of both Python 2.6 and 3.0, for example, you cannot use them in an older Python 2.X that did not yet have this feature. Again, see the change tables in Appendix C for summaries of recent 2.X and 3.X changes.

Which Python Should I Use?

Version choice may be mandated by your organization, but if you're new to Python and learning on your own, you may be wondering which version to install. The answer here depends on your goals. Here are a few suggestions on the choice.

When to choose 3.X: new features, evolution
: If you are learning Python for the first time and don't need to use any existing 2.X code, I encourage you to begin with Python 3.X. It cleans up some longstanding warts in the language and trims some dated cruft, while retaining all the original core ideas and adding some nice new tools. For example, 3.X's seamless Unicode model and broader use of generators and functional techniques are seen by many users as assets. Many popular Python libraries and tools are already available for Python 3.X, or will be by the time you read these words, especially given the continual improvements in the 3.X line. All new language evolution occurs in 3.X only, which adds features and keeps Python relevant, but also makes language definition a constantly moving target—a tradeoff inherent on the leading edge.

When to choose 2.X: existing code, stability
: If you'll be using a system based on Python 2.X, the 3.X line may not be an option for you today. However, you'll find that this book addresses your concerns, too, and will help if you migrate to 3.X in the future. You'll also find that you're in large company. Every group I taught in 2012 was using 2.X only, and I still regularly see useful Python software in 2.X-only form. Moreover, unlike 3.X, 2.X is no longer being changed—which is either an asset or liability, depending on whom you ask. There's nothing wrong with using and writing 2.X code, but you may wish to keep tabs on 3.X and its ongoing evolution as you do. Python's future remains to be written, and is largely up to its users, including you.

When to choose both: version-neutral code
: Probably the best news here is that Python's fundamentals are the same in both its lines—2.X and 3.X differ in ways that many users will find minor, and this book is designed to help you learn both. In fact, as long as you understand their differences, it's often straightforward to write version-neutral code that runs on both

Pythons, as we regularly will in this book. See Appendix C for pointers on 2.X/3.X migration and tips on writing code for both Python lines and audiences.

Regardless of which version or versions you choose to focus on first, your skills will transfer directly to wherever your Python work leads you.

About the Xs: Throughout this book, "3.X" and "2.X" are used to refer collectively to all releases in these two lines. For instance, *3.X* includes 3.0 through 3.3, and future 3.X releases; *2.X* means all from 2.0 through 2.7 (and presumably no others). More specific releases are mentioned when a topic applies to it only (e.g., 2.7's set literals and 3.3's launcher and namespace packages). This notation may occasionally be too broad —some features labeled 2.X here may not be present in early 2.X releases rarely used today—but it accommodates a 2.X line that has already spanned 13 years. The 3.X label is more easily and accurately applied to this younger five-year-old line.

This Book's Prerequisites and Effort

It's impossible to give absolute prerequisites for this book, because its utility and value can depend as much on reader motivation as on reader background. Both true beginners and crusty programming veterans have used this book successfully in the past. If you are motivated to learn Python, and willing to invest the time and focus it requires, this text will probably work for you.

Just how much time is required to learn Python? Although this will vary per learner, this book tends to work best when *read*. Some readers may use this book as an on-demand reference resource, but most people seeking Python mastery should expect to spend at least *weeks* and probably *months* going through the material here, depending on how closely they follow along with its examples. As mentioned, it's roughly equivalent to a full-semester course on the Python language itself.

That's the estimate for learning just Python itself and the software skills required to use it well. Though this book may suffice for basic scripting goals, readers hoping to pursue software development at large as a career should expect to devote additional time after this book to large-scale project experience, and possibly to follow-up texts such as *Programming Python*.[2]

2. The standard disclaimer: I wrote this and another book mentioned earlier, which work together as a set: *Learning Python* for language fundamentals, *Programming Python* for applications basics, and *Python Pocket Reference* as a companion to the other two. All three derive from 1995's original and broad *Programming Python*. I encourage you to explore the many Python books available today (I stopped counting at 200 at Amazon.com just now because there was no end in sight, and this didn't include related subjects like Django). My own publisher has recently produced Python-focused books on instrumentation, data mining, App Engine, numeric analysis, natural language processing, MongoDB, AWS, and more—specific domains you may wish to explore once you've mastered Python language fundamentals here. The Python story today is far too rich for any one book to address alone.

That may not be welcome news to people looking for instant proficiency, but programming is not a trivial skill (despite what you may have heard!). Today's Python, and software in general, are both challenging and rewarding enough to merit the effort implied by comprehensive books such as this. Here are a few pointers on using this book for readers on both sides of the experience spectrum:

To experienced programmers
: You have an initial advantage and can move quickly through some earlier chapters; but you shouldn't skip the core ideas, and may need to work at letting go of some baggage. In general terms, exposure to any programming or scripting before this book might be helpful because of the analogies it may provide. On the other hand, I've also found that prior programming experience can be a handicap due to expectations rooted in other languages (it's far too easy to spot the Java or C++ programmers in classes by the first Python code they write!). Using Python well requires adopting its mindset. By focusing on key core concepts, this book is designed to help you learn to code Python in Python.

To true beginners
: You can learn Python here too, as well as programming itself; but you may need to work a bit harder, and may wish to supplement this text with gentler introductions. If you don't consider yourself a programmer already, you will probably find this book useful too, but you'll want to be sure to proceed slowly and work through the examples and exercises along the way. Also keep in mind that this book will spend more time teaching Python itself than programming basics. If you find yourself lost here, I encourage you to explore an introduction to programming in general before tackling this book. Python's website has links to many helpful resources for beginners.

Formally, this book is designed to serve as *a first Python text for newcomers of all kinds*. It may not be an ideal resource for someone who has never touched a computer before (for instance, we're not going to spend any time exploring what a computer is), but I haven't made many assumptions about your programming background or education.

On the other hand, I won't insult readers by assuming they are "dummies," either, whatever that means—it's easy to do useful things in Python, and this book will show you how. The text occasionally contrasts Python with languages such as C, C++, Java, and others, but you can safely ignore these comparisons if you haven't used such languages in the past.

This Book's Structure

To help orient you, this section provides a quick rundown of the content and goals of the major parts of this book. If you're anxious to get to it, you should feel free to skip

this section (or browse the table of contents instead). To some readers, though, a book this large probably merits a brief roadmap up front.

By design, each *part* covers a major functional area of the language, and each part is composed of *chapters* focusing on a specific topic or aspect of the part's area. In addition, each chapter ends with *quizzes* and their answers, and each part ends with larger *exercises*, whose solutions show up in Appendix D.

Practice matters: I strongly recommend that readers work through the quizzes and exercises in this book, and work along with its examples in general if you can. In programming, there's no substitute for practicing what you've read. Whether you do it with this book or a project of your own, actual coding is crucial if you want the ideas presented here to stick.

Overall, this book's presentation is *bottom-up* because Python is too. The examples and topics grow more challenging as we move along. For instance, Python's classes are largely just packages of functions that process built-in types. Once you've mastered built-in types and functions, classes become a relatively minor intellectual leap. Because each part builds on those preceding it this way, most readers will find a *linear reading* makes the most sense. Here's a preview of the book's main parts you'll find along the way:

Part I

We begin with a general overview of Python that answers commonly asked initial questions—why people use the language, what it's useful for, and so on. The first chapter introduces the major ideas underlying the technology to give you some background context. The rest of this part moves on to explore the ways that both Python and programmers run programs. The main goal here is to give you just enough information to be able to follow along with later examples and exercises.

Part II

Next, we begin our tour of the Python language, studying Python's major built-in object types and what you can do with them in depth: numbers, lists, dictionaries, and so on. You can get a lot done with these tools alone, and they are at the heart of every Python script. This is the most substantial part of the book because we lay groundwork here for later chapters. We'll also explore dynamic typing and its references—keys to using Python well—in this part.

Part III

The next part moves on to introduce Python's *statements*—the code you type to create and process objects in Python. It also presents Python's general syntax model. Although this part focuses on syntax, it also introduces some related tools (such as the PyDoc system), takes a first look at iteration concepts, and explores coding alternatives.

Part IV

This part begins our look at Python's higher-level program structure tools. *Functions* turn out to be a simple way to package code for reuse and avoid code redundancy. In this part, we will explore Python's scoping rules, argument-passing techniques, the sometimes-notorious lambda, and more. We'll also revisit iterators from a functional programming perspective, introduce user-defined generators, and learn how to time Python code to measure performance here.

Part V

Python *modules* let you organize statements and functions into larger components, and this part illustrates how to create, use, and reload modules. We'll also look at some more advanced topics here, such as module packages, module reloading, package-relative imports, 3.3's new namespace packages, and the `__name__` variable.

Part VI

Here, we explore Python's object-oriented programming tool, the *class*—an optional but powerful way to structure code for customization and reuse, which almost naturally minimizes redundancy. As you'll see, classes mostly reuse ideas we will have covered by this point in the book, and OOP in Python is mostly about looking up names in linked objects with a special first argument in functions. As you'll also see, OOP is optional in Python, but most find Python's OOP to be much simpler than others, and it can shave development time substantially, especially for long-term strategic project development.

Part VII

We conclude the language fundamentals coverage in this text with a look at Python's exception handling model and statements, plus a brief overview of development tools that will become more useful when you start writing larger programs (debugging and testing tools, for instance). Although exceptions are a fairly lightweight tool, this part appears after the discussion of classes because user-defined exceptions should now all be classes. We also cover some more advanced topics, such as context managers, here.

Part VIII

In the final part, we explore some advanced topics: Unicode and byte strings, managed attribute tools like properties and descriptors, function and class decorators, and metaclasses. These chapters are all optional reading, because not all programmers need to understand the subjects they address. On the other hand, readers who must process internationalized text or binary data, or are responsible for developing APIs for other programmers to use, should find something of interest in this part. The examples here are also larger than most of those in this book, and can serve as self-study material.

Part IX

The book wraps up with a set of four appendixes that give platform-specific tips for installing and using Python on various computers; present the new Windows

launcher that ships with Python 3.3; summarize changes in Python addressed by recent editions and give links to their coverage here; and provide solutions to the end-of-part exercises. Solutions to end-of-chapter quizzes appear in the chapters themselves.

See the table of contents for a finer-grained look at this book's components.

What This Book Is Not

Given its relatively large audience over the years, some have inevitably expected this book to serve a role outside its scope. So now that I've told you what this book is, I also want to be clear on what it isn't:

- This book is a tutorial, *not* a reference.
- This book covers the language itself, *not* applications, standard libraries, or third-party tools.
- This book is a comprehensive look at a substantial topic, *not* a watered-down overview.

Because these points are key to this book's content, I want to say a few more words about them up front.

It's Not a Reference or a Guide to Specific Applications

This book is a *language tutorial*, not a reference, and not an applications book. This is by design: *today's Python*—with its built-in types, generators, closures, comprehensions, Unicode, decorators, and blend of procedural, object-oriented, and functional programming paradigms—makes the core language a substantial topic all by itself, and a prerequisite to all your future Python work, in whatever domains you pursue. When you are ready for other resources, though, here are a few suggestions and reminders:

Reference resources
: As implied by the preceding structural description, you can use the index and table of contents to hunt for details, but there are no reference appendixes in this book. If you are looking for Python reference resources (and most readers probably will be very soon in their Python careers), I suggest the previously mentioned book that I also wrote as a companion to this one—*Python Pocket Reference*—as well as other reference books you'll find with a quick search, and the standard Python reference manuals maintained at *http://www.python.org*. The latter of these are free, always up to date, and available both on the Web and on your computer after a Windows install.

Applications and libraries
: As also discussed earlier, this book is not a guide to specific *applications* such as the Web, GUIs, or systems programming. By proxy, this includes the libraries and

tools used in applications work; although some *standard libraries* and tools are introduced here—including `timeit`, `shelve`, `pickle`, `struct`, `json`, `pdb`, `os`, `urllib`, `re`, `xml`, `random`, *PyDoc* and *IDLE*—they are not officially in this book's primary scope. If you're looking for more coverage on such topics and are already proficient with Python, I recommend the follow-up book *Programming Python*, among others. That book assumes this one as its prerequisite, though, so be sure you have a firm grasp of the core language first. Especially in an engineering domain like software, one must walk before one runs.

It's Not the Short Story for People in a Hurry

As you can tell from its size, this book also doesn't skimp on the details: it presents the *full Python language*, not a brief look at a simplified subset. Along the way it also covers *software principles* that are essential to writing good Python code. As mentioned, this is a multiple-week or -month book, designed to impart the skill level you'd acquire from a full-term class on Python.

This is also deliberate. Many of this book's readers don't need to acquire full-scale software development skills, of course, and some can absorb Python in a piecemeal fashion. At the same time, because *any* part of the language may be used in code you will encounter, no part is truly optional for most programmers. Moreover, even casual scripters and hobbyists need to know basic principles of software development in order to code well, and even to use precoded tools properly.

This book aims to address both of these needs—*language and principles*—in enough depth to be useful. In the end, though, you'll find that Python's more advanced tools, such as its object-oriented and functional programming support, are relatively easy to learn once you've mastered their prerequisites—and you will, if you work through this book one chapter at a time.

It's as Linear as Python Allows

Speaking of *reading order*, this edition also tries hard to minimize *forward references*, but Python 3.X's changes make this impossible in some cases (in fact, 3.X sometimes seems to assume you already know Python while you're learning it!). As a handful of representative examples:

- Printing, sorts, the string `format` method, and some `dict` calls rely on function *keyword* arguments.
- Dictionary key lists and tests, and the `list` calls used around many tools, imply *iteration* concepts.
- Using `exec` to run code now assumes knowledge of *file objects* and interfaces.
- Coding new *exceptions* requires *classes* and OOP fundamentals.

- And so on—even basic *inheritance* broaches advanced topics such as *metaclasses* and *descriptors*.

Python is still best learned as a progression from simple to advanced, and a *linear reading* here still makes the most sense. Still, some topics may require nonlinear jumps and random lookups. To minimize these, this book will point out forward dependencies when they occur, and will ease their impacts as much as possible.

But if your time is tight: Though depth is crucial to mastering Python, some readers may have limited time. If you are interested in starting out with a *quick Python tour*, I suggest Chapter 1, Chapter 4, Chapter 10, and Chapter 28 (and perhaps 26)—a short survey that will hopefully pique your interest in the more complete story told in the rest of the book, and which most readers will need in today's Python software world. In general, this book is intentionally *layered* this way to make its material easier to absorb—with introductions followed by details, so you can start with overviews, and dig deeper over time. You don't need to read this book all at once, but its gradual approach is designed to help you tackle its material eventually.

This Book's Programs

In general, this book has always strived to be agnostic about both Python versions and platforms. It's designed to be useful to all Python users. Nevertheless, because Python changes over time and platforms tend to differ in pragmatic ways, I need to describe the specific systems you'll see in action in most examples here.

Python Versions

This fifth edition of this book, and all the program examples in it, are based on Python versions *3.3 and 2.7*. In addition, many of its examples run under prior 3.X and 2.X releases, and notes about the history of language changes in earlier versions are mixed in along the way for users of older Pythons.

Because this text focuses on the core language, however, you can be fairly sure that most of what it has to say won't change very much in *future* releases of Python, as noted earlier. Most of this book applies to *earlier* Python versions, too, except when it does not; naturally, if you try using extensions added after a release you're using, all bets are off. As a rule of thumb, the latest Python is the best Python if you are able to upgrade.

Because this book focuses on the core language, most of it also applies to both *Jython* and *IronPython*, the Java- and .NET-based Python language implementations, as well as other Python implementations such as *Stackless* and *PyPy* (described in Chapter 2). Such alternatives differ mostly in usage details, not language.