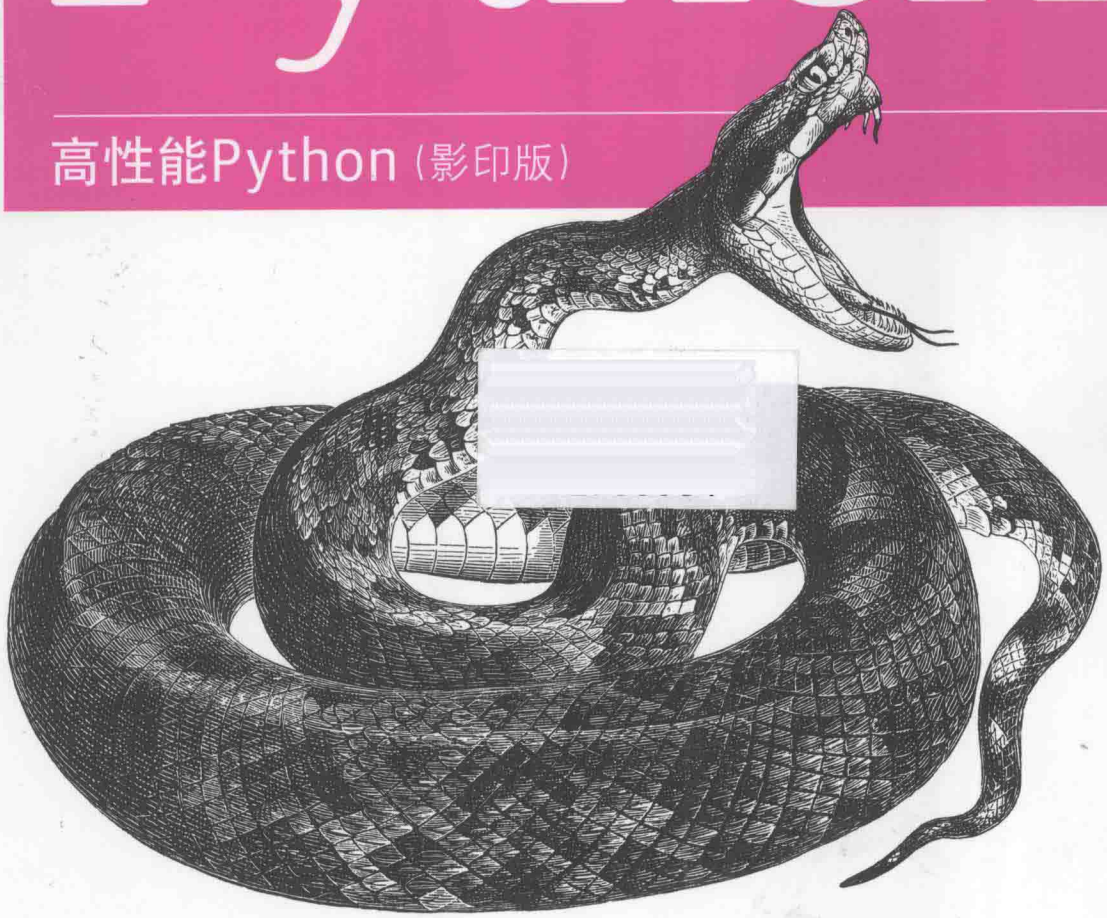
O'REILLY®

High Performance Python

高性能Python（影印版）



東南大學出版社

Micha Gorelick, Ian Ozsvald 著

高性能Python（影印版）

High Performance Python

Micha Gorelick, Ian Ozsvald 著

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo O'REILLY®

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

高性能 Python：英文/(美)戈雷利克(Gorelick, M.)，(英)欧日沃尔德(Ozsvald, I.)著. —影印本. —南京：东南大学出版社，2015.2

书名原文：High Performance Python

ISBN 978-7-5641-5385-4

Ⅰ.①高… Ⅱ.①戈… ②欧… Ⅲ.①软件工具—程序设计—英文 Ⅳ.①TP311.56

中国版本图书馆 CIP 数据核字(2014)第 294385 号

江苏省版权局著作权合同登记

图字：10-2013-366 号

高性能 Python(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：常州市武进第三印刷有限公司

开　　本：787 毫米×980 毫米　　16 开本

印　　张：23.25

字　　数：455 千字

版　　次：2015 年 2 月第 1 版

印　　次：2015 年 2 月第 1 次印刷

书　　号：ISBN 978-7-5641-5385-4

定　　价：78.00 元

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

Preface

Python is easy to learn. You're probably here because now that your code runs correctly, you need it to run faster. You like the fact that your code is easy to modify and you can iterate with ideas quickly. The trade-off between *easy to develop* and *runs as quickly as I need* is a well-understood and often-bemoaned phenomenon. There are solutions.

Some people have serial processes that have to run faster. Others have problems that could take advantage of multicore architectures, clusters, or graphics processing units. Some need scalable systems that can process more or less as expediency and funds allow, without losing reliability. Others will realize that their coding techniques, often borrowed from other languages, perhaps aren't as natural as examples they see from others.

In this book we will cover all of these topics, giving practical guidance for understanding bottlenecks and producing faster and more scalable solutions. We also include some war stories from those who went ahead of you, who took the knocks so you don't have to.

Python is well suited for rapid development, production deployments, and scalable systems. The ecosystem is full of people who are working to make it scale on your behalf, leaving you more time to focus on the more challenging tasks around you.

Who This Book Is For

You've used Python for long enough to have an idea about why certain things are slow and to have seen technologies like Cython, `numpy`, and PyPy being discussed as possible solutions. You might also have programmed with other languages and so know that there's more than one way to solve a performance problem.

While this book is primarily aimed at people with CPU-bound problems, we also look at data transfer and memory-bound solutions. Typically these problems are faced by scientists, engineers, quants, and academics.

We also look at problems that a web developer might face, including the movement of data and the use of just-in-time (JIT) compilers like PyPy for easy-win performance gains.

It might help if you have a background in C (or C++, or maybe Java), but it isn't a prerequisite. Python's most common interpreter (CPython—the standard you normally get if you type `python` at the command line) is written in C, and so the hooks and libraries all expose the gory inner C machinery. There are lots of other techniques that we cover that don't assume any knowledge of C.

You might also have a lower-level knowledge of the CPU, memory architecture, and data buses, but again, that's not strictly necessary.

Who This Book Is Not For

This book is meant for intermediate to advanced Python programmers. Motivated novice Python programmers may be able to follow along as well, but we recommend having a solid Python foundation.

We don't cover storage-system optimization. If you have a SQL or NoSQL bottleneck, then this book probably won't help you.

What You'll Learn

Your authors have been working with large volumes of data, a requirement for *I want the answers faster!* and a need for scalable architectures, for many years in both industry and academia. We'll try to impart our hard-won experience to save you from making the mistakes that we've made.

At the start of each chapter, we'll list questions that the following text should answer (if it doesn't, tell us and we'll fix it in the next revision!).

We cover the following topics:

- Background on the machinery of a computer so you know what's happening behind the scenes
- Lists and tuples—the subtle semantic and speed differences in these fundamental data structures
- Dictionaries and sets—memory allocation strategies and access algorithms in these important data structures
- Iterators—how to write in a more Pythonic way and open the door to infinite data streams using iteration
- Pure Python approaches—how to use Python and its modules effectively

- Matrices with `numpy`—how to use the beloved `numpy` library like a beast
- Compilation and just-in-time computing—processing faster by compiling down to machine code, making sure you're guided by the results of profiling
- Concurrency—ways to move data efficiently
- `multiprocessing`—the various ways to use the built-in `multiprocessing` library for parallel computing, efficiently share `numpy` matrices, and some costs and benefits of interprocess communication (IPC)
- Cluster computing—convert your `multiprocessing` code to run on a local or remote cluster for both research and production systems
- Using less RAM—approaches to solving large problems without buying a humungous computer
- Lessons from the field—lessons encoded in war stories from those who took the blows so you don't have to

Python 2.7

Python 2.7 is the dominant version of Python for scientific and engineering computing. 64-bit is dominant in this field, along with *nix environments (often Linux or Mac). 64-bit lets you address larger amounts of RAM. *nix lets you build applications that can be deployed and configured in well-understood ways with well-understood behaviors.

If you're a Windows user, then you'll have to buckle up. Most of what we show will work just fine, but some things are OS-specific, and you'll have to research a Windows solution. The biggest difficulty a Windows user might face is the installation of modules: research in sites like StackOverflow should give you the solutions you need. If you're on Windows, then having a virtual machine (e.g., using VirtualBox) with a running Linux installation might help you to experiment more freely.

Windows users should definitely look at a packaged solution like those available through Anaconda, Canopy, Python(x,y), or Sage. These same distributions will make the lives of Linux and Mac users far simpler too.

Moving to Python 3

Python 3 is the future of Python, and everyone is moving toward it. Python 2.7 will nonetheless be around for many years to come (some installations still use Python 2.4 from 2004); its retirement date has been set at 2020.

The shift to Python 3.3+ has caused enough headaches for library developers that people have been slow to port their code (with good reason), and therefore people have been slow to adopt Python 3. This is mainly due to the complexities of switching from a mix

of string and Unicode datatypes in complicated applications to the Unicode and byte implementation in Python 3.

Typically, when you want reproducible results based on a set of trusted libraries, you don't want to be at the bleeding edge. High performance Python developers are likely to be using and trusting Python 2.7 for years to come.

Most of the code in this book will run with little alteration for Python 3.3+ (the most significant change will be with `print` turning from a statement into a function). In a few places we specifically look at improvements that Python 3.3+ provides. One item that might catch you out is the fact that `/` means *integer* division in Python 2.7, but it becomes *float* division in Python 3. Of course—being a good developer, your well-constructed unit test suite will already be testing your important code paths, so you'll be alerted by your unit tests if this needs to be addressed in your code.

`scipy` and `numpy` have been Python 3–compatible since late 2010. `matplotlib` was compatible from 2012, `scikit-learn` was compatible in 2013, and `NLTK` is expected to be compatible in 2014. Django has been compatible since 2013. The transition notes for each are available in their repositories and newsgroups; it is worth reviewing the processes they used if you're going to migrate older code to Python 3.

We encourage you to experiment with Python 3.3+ for new projects, but to be cautious with libraries that have only recently been ported and have few users—you'll have a harder time tracking down bugs. It would be wise to make your code Python 3.3+-compatible (learn about the `__future__` imports), so a future upgrade will be easier.

Two good guides are "Porting Python 2 Code to Python 3" (*http://bit.ly/pyporting*) and "Porting to Python 3: An in-depth guide." (*http://python3porting.com/*) With a distribution like Anaconda or Canopy, you can run both Python 2 and Python 3 simultaneously—this will simplify your porting.

License

How to Make an Attribution

The Creative Commons license requires that you attribute your use of a part of this book. Attribution just means that you should write something that someone else can follow to find this book. The following would be sensible: "*High Performance Python* by Micha Gorelick and Ian Ozsvald (O'Reilly). Copyright 2014 Micha Gorelick and Ian Ozsvald, 978-1-449-36159-4."

Errata and Feedback

We encourage you to review this book on public sites like Amazon—please help others understand if they'd benefit from this book! You can also email us at *feedback@highperformancepython.com*.

We're particularly keen to hear about errors in the book, successful use cases where the book has helped you, and high performance techniques that we should cover in the next edition. You can access the page for this book at *http://bit.ly/High_Performance_Python*.

Complaints are welcomed through the instant-complaint-transmission-service `> /dev/null`.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to commands, modules, and program elements such as variable or function names, databases, datatypes, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a question or exercise.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/mynameisfiber/high_performance_python*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM

Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

To comment or ask technical questions about this book, send email to *bookques tions@oreilly.com.*

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com.*

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Thanks to Jake Vanderplas, Brian Granger, Dan Foreman-Mackey, Kyran Dale, John Montgomery, Jamie Matthews, Calvin Giles, William Winter, Christian Schou Oxvig, Balthazar Rouberol, Matt "snakes" Reiferson, Patrick Cooper, and Michael Skirpan for invaluable feedback and contributions. Ian thanks his wife Emily for letting him disappear for 10 months to write this (thankfully she's terribly understanding). Micha thanks Elaine and the rest of his friends and family for being so patient while he learned to write. O'Reilly are also rather lovely to work with.

Our contributors for the "Lessons from the Field" chapter very kindly shared their time and hard-won lessons. We give thanks to Ben Jackson, Radim Řehůřek, Sebastjan Trebca, Alex Kelly, Marko Tasic, and Andrew Godwin for their time and effort.

Table of Contents

CHAPTER 1

Understanding Performant Python

Questions You'll Be Able to Answer After This Chapter

- What are the elements of a computer's architecture?
- What are some common alternate computer architectures?
- How does Python abstract the underlying computer architecture?
- What are some of the hurdles to making performant Python code?
- What are the different types of performance problems?

Programming computers can be thought of as moving bits of data and transforming them in special ways in order to achieve a particular result. However, these actions have a time cost. Consequently, high performance programming can be thought of as the act of minimizing these operations by either reducing the overhead (i.e., writing more efficient code) or by changing the way that we do these operations in order to make each one more meaningful (i.e., finding a more suitable algorithm).

Let's focus on reducing the overhead in code in order to gain more insight into the actual hardware on which we are moving these bits. This may seem like a futile exercise, since Python works quite hard to abstract away direct interactions with the hardware. However, by understanding both the best way that bits can be moved in the real hardware and the ways that Python's abstractions force your bits to move, you can make progress toward writing high performance programs in Python.

The Fundamental Computer System

The underlying components that make up a computer can be simplified into three basic parts: the computing units, the memory units, and the connections between them. In

addition, each of these units has different properties that we can use to understand them. The computational unit has the property of how many computations it can do per second, the memory unit has the properties of how much data it can hold and how fast we can read from and write to it, and finally the connections have the property of how fast they can move data from one place to another.

Using these building blocks, we can talk about a standard workstation at multiple levels of sophistication. For example, the standard workstation can be thought of as having a central processing unit (CPU) as the computational unit, connected to both the random access memory (RAM) and the hard drive as two separate memory units (each having different capacities and read/write speeds), and finally a bus that provides the connections between all of these parts. However, we can also go into more detail and see that the CPU itself has several memory units in it: the L1, L2, and sometimes even the L3 and L4 cache, which have small capacities but very fast speeds (from several kilobytes to a dozen megabytes). These extra memory units are connected to the CPU with a special bus called the *backside bus*. Furthermore, new computer architectures generally come with new configurations (for example, Intel's Nehalem CPUs replaced the front-side bus with the Intel QuickPath Interconnect and restructured many connections). Finally, in both of these approximations of a workstation we have neglected the network connection, which is effectively a very slow connection to potentially many other computing and memory units!

To help untangle these various intricacies, let's go over a brief description of these fundamental blocks.

Computing Units

The computing unit of a computer is the centerpiece of its usefulness—it provides the ability to transform any bits it receives into other bits or to change the state of the current process. CPUs are the most commonly used computing unit; however, graphics processing units (GPUs), which were originally typically used to speed up computer graphics but are becoming more applicable for numerical applications, are gaining popularity due to their intrinsically parallel nature, which allows many calculations to happen simultaneously. Regardless of its type, a computing unit takes in a series of bits (for example, bits representing numbers) and outputs another set of bits (for example, representing the sum of those numbers). In addition to the basic arithmetic operations on integers and real numbers and bitwise operations on binary numbers, some computing units also provide very specialized operations, such as the "fused multiply add" operation, which takes in three numbers, `A,B,C`, and returns the value `A * B + C`.

The main properties of interest in a computing unit are the number of operations it can do in one cycle and how many cycles it can do in one second. The first value is measured