# High Performance Python

*Micha Gorelick and Ian Ozsvald*

Beijing • Cambridge • Farnham • Koln • Sebastopol • Tokyo O'REILLY$^{0}$

# Table of Contents

Preface ix

1. Understanding Performant Python 1

The Fundamental Computer System 1

Computing Units 2

Memory Units 5

Communications Layers 7

Putting the Fundamental Elements Together 9

Idealized Computing Versus the Python Virtual Machine 10

So Why Use Python? 13

2. Profiling to Find Bottlenecks 17

Profiling Efficiently 18

Introducing the Julia Set 19

Calculating the Full Julia Set 23

Simple Approaches to Timing—print and a Decorator 26

Simple Timing Using the Unix time Command 29

Using the cProfile Module 31

Using runsnakerun to Visualize cProfile Output 36

Using line_profiler for Line-by-Line Measurements 37

Using memory_profiler to Diagnose Memory Usage . 42

Inspecting Objects on the Heap with heapy 48

Using dowser for Live Graphing of Instantiated Variables 50

Using the dis Module to Examine CPython Bytecode 52

Different Approaches, Different Complexity 54

Unit Testing During Optimization to Maintain Correctness 56

No-op @profile Decorator 57

Strategies to Profile Your Code Successfully 59

Wrap-Up 60

3. Lists and Tuples 61
A More Efficient Search 64
Lists Versus Tuples 66
Lists as Dynamic Arrays 67
Tuples As Static Arrays 70
Wrap-Up 72

4. Dictionaries and Sets 73
How Do Dictionaries and Sets Work? 77
Inserting and Retrieving 77
Deletion 80
Resizing 81
Hash Functions and Entropy 81
Dictionaries and Namespaces 85
Wrap-Up 88

5. Iterators and Generators 89
Iterators for Infinite Series 92
Lazy Generator Evaluation 94
Wrap-Up 98

6. Matrix and Vector Computation 99
Introduction to the Problem 100
Aren't Python Lists Good Enough? 105
Problems with Allocating Too Much 106
Memory Fragmentation 109
Understanding perf 111
Making Decisions with perf s Output 113
Enter numpy 114
Applying numpy to the Diffusion Problem 117
Memory Allocations and In-Place Operations 120
Selective Optimizations: Finding What Needs to Be Fixed 124
numexpr: Making In-Place Operations Faster and Easier 127
A Cautionary Tale: Verify "Optimizations" (scipy) 129
Wrap-Up 130

7. Compiling to C 135
What Sort of Speed Gains Are Possible? 136
JIT Versus AOT Compilers 138
Why Does Type Information Help the Code Run Faster? 138
Using a C Compiler 139
Reviewing the Julia Set Example 140
Cython 140

Compiling a Pure-Python Version Using Cython 141
Cython Annotations to Analyze a Block of Code 143
Adding Some Type Annotations 145
Shed Skin 150
Building an Extension Module 151
The Cost of the Memory Copies 153
Cython and numpy 154
Parallelizing the Solution with OpenMP on One Machine 155
Numba 157
Pythran 159
PyPy 160
Garbage Collection Differences 161
Running PyPy and Installing Modules 162
When to Use Each Technology 164
Other Upcoming Projects 165
A Note on Graphics Processing Units (GPUs) 166
A Wish for a Future Compiler Project 166
Foreign Function Interfaces 167
ctypes 167
cffi 170
f2py 173
CPython Module 175
Wrap-Up 179

8. Concurrency 181
Introduction to Asynchronous Programming 182
Serial Crawler 185
gevent 187
tornado 192
AsyncIO 196
Database Example 198
Wrap-Up 201

9. The multiprocessing Module 203
An Overview of the Multiprocessing Module 206
Estimating Pi Using the Monte Carlo Method 208
Estimating Pi Using Processes and Threads 210
Using Python Objects 210
Random Numbers in Parallel Systems 217
Using numpy 218
Finding Prime Numbers 221
Queues of Work 227
Verifying Primes Using Interprocess Communication 232

Serial Solution 236
Naive Pool Solution 236
A Less Naive Pool Solution 238
Using Manager.Value as a Flag 239
Using Redis as a Flag 241
Using RawValue as a Flag 243
Using mmap as a Flag 244
Using mmap as a Flag Redux 245
Sharing numpy Data with multiprocessing 248
Synchronizing File and Variable Access 254
File Locking 255
Locking a Value 258
Wrap-Up 261

10. Clusters and Job Queues 263
Benefits of Clustering 264
Drawbacks of Clustering 265
$462 Million Wall Street Loss Through Poor Cluster Upgrade Strategy 266
Skype's 24-Hour Global Outage 267
Common Cluster Designs 268
How to Start a Clustered Solution 268
Ways to Avoid Pain When Using Clusters 269
Three Clustering Solutions 270
Using the Parallel Python Module for Simple Local Clusters 271
Using IPython Parallel to Support Research 273
NSQ for Robust Production Clustering 277
Queues 277
Pub/sub 278
Distributed Prime Calculation 280
Other Clustering Tools to Look At 284
Wrap-Up 285

11. Using Less RAM 287
Objects for Primitives Are Expensive 288
The Array Module Stores Many Primitive Objects Cheaply 289
Understanding the RAM Used in a Collection 292
Bytes Versus Unicode 294
Efficiently Storing Lots of Text in RAM 295
Trying These Approaches on 8 Million Tokens 296
Tips for Using Less RAM 304
Probabilistic Data Structures 305
Very Approximate Counting with a 1-byte Morris Counter 306
K-Minimum Values 308

Bloom Filters 312
LogLog Counter 317
Real-Worid Example 321

12. Lessons from the Field 325
Adaptive Labs Social Media Analytics (SoMA) 325
Python at Adaptive Lab 326
SoMA's Design 326
Our Development Methodology 327
Maintaining SoMA 327
Advice for Fellow Engineers 328
Making Deep Learning Fly with RadimRehurek.com 328
The Sweet Spot 328
Lessons in Optimizing 330
Wrap-Up 332
Large-Scale Productionized Machine Learning at Lyst.com 333
Pythons Place at Lyst 333
Cluster Design 333
Code Evolution in a Fast-Moving Start-Up 333
Building the Recommendation Engine 334
Reporting and Monitoring 334
Some Advice 335
Large-Scale Social Media Analysis at Smesh 335
Python's Role at Smesh . 335
The Platform 336
High Performance Real-Time String Matching 336
Reporting, Monitoring, Debugging, and Deployment . 338
PyPy for Successful Web and Data Processing Systems 339
Prerequisites 339
The Database 340
The Web Application 340
OCR and Translation 341
Task Distribution and Workers 341
Conclusion 341
Task Queues at Lanyrd.com 342
Python's Role at Lanyrd 342
Making the Task Queue Performant 343
Reporting, Monitoring, Debugging, and Deployment 343
Advice to a Fellow Developer 343

Index 345