Perl 6 Fundamentals

A Primer with Examples, Projects, and Case Studies

Moritz Lenz

Foreword by Larry Wall, creator of Perl

Apress®

Perl 6 Fundamentals

A Primer with Examples, Projects, and Case Studies

Moritz Lenz

Foreword by Larry Wall, creator of Perl

Apress®

Perl 6 Fundamentals: A Primer with Examples, Projects, and Case Studies

Moritz Lenz
Fürth, Bayern, Germany

ISBN-13 (pbk): 978-1-4842-2898-2　　ISBN-13 (electronic): 978-1-4842-2899-9
DOI 10.1007/978-1-4842-2899-9

Library of Congress Control Number: 2017948406

Cover image by Freepik (www.freepik.com)

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Technical Reviewer: Massimo Nardone
Coordinating Editor: Mark Powers
Copy Editor: Brendan Frost

Contents at a Glance

Contents

About the Author

Moritz Lenz is a software engineer and architect. In the Perl community, he is well known for his contributions to the Perl 6 programming language, the Rakudo Perl 6 compiler, related test suite, infrastructure, and tools. At his employer, noris network AG, he introduced Continuous Delivery for many in-house-developed applications, and now wants to share his experience with the wider world.

About the Technical Reviewer

Massimo Nardone has more than 22 years of experiences in Security, Web/Mobile development, and Cloud and IT Architecture. His true IT passions are Security and Android.

He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years.

He holds a Master of Science degree in Computing Science from the University of Salerno, Italy.

He has worked as a Project Manager, Software Engineer, Research Engineer, Chief Security Architect, Information Security Manager, PCI/SCADA Auditor, and Senior Lead IT Security/Cloud/SCADA Architect for many years.

Massimo's technical skills include Security, Android, Cloud, Java, MySQL, Drupal, Cobol, Perl, Web and Mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, and Scratch.

He currently works as Chief Information Security Officer (CISO) for Cargotec Oyj.

He worked as visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (PKI, SIP, SAML, and Proxy areas).

Massimo has reviewed more than 40 IT books for different publishing companies, and he is the coauthor of *Pro Android Games* (Apress, 2015).

Acknowledgments

They say it takes a village to raise a child. Similar things can be said about writing a book. It is only possible through the effort of many people, often unpaid volunteers who contribute just to see the project succeed, and out of kindness of heart.

I am very grateful for the review by and feedback from Paul Cochrane, Will Coleda, Elizabeth Mattijsen, Ryan Erwin, Claudio Ramirez, and Massimo Nardone.

Special thanks go to Larry Wall for creating Perl, for the great foreword, and for shaping the community to be friendly, welcoming, and a second home to me.

Finally, thanks go to my parents, for kindling my love both for books and for engineering. And most importantly to my family: to Signe, my wife, for constant support; and to my daughters Ida and Ronja for keeping me grounded in the real world, and bringing joy to my life.

Foreword

The reason I'm writing this (and perhaps why you're reading it), is that people just give me way too much credit. Yeah, sure, I invented Perl 30 years ago, and I coded the first five versions all by myself, pretty much. But for the last 20 years, the vast majority of the work has been done by other members of the industrious Perl community, who get far too little credit. To be sure, I don't mind getting extra credit: I'm human enough to enjoy the undue adulation, and I understand how communities want—and possibly even need—to have a figurehead who represents the whole.

I will gladly take credit, however, for the idea that a computer language must have a vibrant community in order to thrive. From the beginning, that was the intent of Perl. It all comes down to linguistics: Perl was designed to work like a natural language on many levels, not just the syntactic level. In particular, every living language is symbiotic with the culture that conveys it forward into the future. More generally, natural languages are responsive to context on every level, and some of those levels are anthropological. People provide context to Perl, which in turn is designed to respond productively to that context.

This may seem simple, but it's a surprisingly tricky concept to bake into a programming language and its culture. Just look at how many computer languages fail at it. In most programming cultures, you are a slave to the computer language. Rarely, if ever, do you get the feeling that the computer language is there to work for you.

We're trying to change all that. So when the Perl community, back in 2000, decided to do a major redesign of Perl 5 to clean up the cruftier bits, we not only wanted to fix things that we already knew were suboptimal, but we also wanted to do a better job of responding to cultural change, because we simply don't know what we'll want in the future. So we thought about how best to future-proof a computer language; much of the current design is about maintaining careful control of identity, mutability, dimensionality, typology, and extensibility over time, so we could isolate changes to minimize collateral damage. Other than worrying about that, my main contribution as language designer was to unify the community's contradictory desires into a coherent whole.

All that being said, it's still all about the community: nearly all the implementation work was done by others, and most of the features that ended up in Perl 6 can be traced back through various revisions to the community's original RFCs. True, many of those original designs we deemed inadequate, but we never lost sight of the pain points those original suggestions were trying to address. As a result, even though Perl 6 ended up to be quite a different language than Perl 5, it is still essentially Perl in spirit. We now think of Perl 6 as the "younger sister" to Perl 5, and we expect the sisters will get along well in the future. You're allowed to be friends with either or both. They only squabble occasionally, as family do.

Since 2000, we've had over 800 contributors to the Perl 6 effort, one way or another. Some folks come and go, and that's fine. We welcome the occasional contributor. On the other hand, we also honor those who strove greatly but paid the price of burnout. And we deeply revere those who have already passed on, who contributed, in some cases, knowing they would never see the final result.

But then there are those who have stuck with the Perl 6 effort through thick and thin, through joy and frustration, who have patiently (or at least persistently!) risen to the challenge of building a better Perl community around the revised Perl language, and who have gladly taken on the hard work of making other people's lives easy.

One such is my friend Moritz Lenz, your author, and a much-respected member of our not-so-secret Perl 6 Cabal. Well, some days it's more like the Perl 6 Comedy Club.

While thinking about this foreword, I guessed (and Moritz confirmed) that he has a background in the performance arts. One can tell, because he seems to have a natural feel for when to blend in as part of the ensemble, when to step forward and take a solo lead, and when to step back again and let someone else come to the fore. In many ways, the Perl 6 effort has been like a jazz jam session, or like improv comedy, the kind of art where part of it is showing how cleverly we learn to work together and trade off roles on the fly.

I've had to learn some of that myself. Good leaders don't try to lead all the time. That's what bad leaders try to do. Often, a good leader is just "following out in front," sensing when the group behind wants a change of direction, and then pretending to lead the group in that direction. Moritz knows how to do that too.

Hence, this book. It's not just a reference, since you can always find such materials online. Nor is it just a cookbook. I like to think of it as an extended invitation, from a well-liked and well-informed member of our circle, to people like you who might want to join in on the fun. Because joy is what's fundamental to Perl. The essence of Perl is an invitation to love, and to be loved by, the Perl community. It's an invitation to be a participant of the gift economy, on both the receiving and the giving end.

Since Herr Doktor Professor Lenz is from Deutschland, I think it's appropriate to end with one of my favorite German sayings:

> Liebe ist arm und reich,
>
> Fordert und gibt zugleich.

Oder auf Englisch:

> Love is poor and rich,
>
> Taking and giving as one.

Larry Wall, May 2017

CHAPTER 1

What Is Perl 6?

Perl 6 is a programming language. It is designed to be easily learned, read, and written by humans, and is inspired by natural language. It allows the beginner to write in "baby Perl," while giving the experienced programmer freedom of expression, from concise to poetic.

Perl 6 is gradually typed. It mostly follows the paradigm of dynamically typed languages in that it accepts programs whose type safety it can't guarantee during compilation. However, unlike many dynamic languages, it accepts and enforces type constraints. Where possible, the compiler uses type annotations to make decisions at compile time that would otherwise only be possible at runtime.

Many programming paradigms have influenced Perl 6. You can write imperative, object-oriented, and functional programs in Perl 6. Declarative programming is supported through features like multiple-dispatch, sub-typing, and the regex and grammar engine.

Most lookups in Perl 6 are lexical, and the language avoids global state. This makes parallel and concurrent execution of programs easier, as does Perl 6's focus on high-level concurrency primitives. When you don't want to be limited to one CPU core, instead of thinking in terms of threads and locks, you tend to think about promises and message queues.

Perl 6 as a language is not opinionated about whether Perl 6 programs should be compiled or interpreted. Rakudo Perl 6—the main implementation—precompiles modules on the fly and interprets scripts.

1.1 Perl 5, the Older Sister

Around the year 2000, Perl 5 development faced major strain from the conflicting desires to evolve and to keep backward compatibility.

Perl 6 was the valve to release this tension. All the extension proposals that required a break in backward compatibility were channeled into Perl 6, leaving it in a dreamlike state where everything was possible and nothing was fixed. It took several years of hard work to get into a more solid state.

M. Lenz, *Perl 6 Fundamentals*, DOI 10.1007/978-1-4842-2899-9_1