



# 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

Copyright © 2017 by Moritz Lenz

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image by Freepik ([www.freepik.com](http://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

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [rights@apress.com](mailto:rights@apress.com), or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/9781484228982](http://www.apress.com/9781484228982). For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

# Contents at a Glance

<b>About the Author .....</b>	<b>xi</b>
<b>About the Technical Reviewer .....</b>	<b>xiii</b>
<b>Acknowledgments .....</b>	<b>xv</b>
<b>Foreword .....</b>	<b>xvii</b>
<b>■ Chapter 1: What Is Perl 6? .....</b>	<b>1</b>
<b>■ Chapter 2: Running Rakudo Perl 6 .....</b>	<b>5</b>
<b>■ Chapter 3: Formatting a Sudoku Puzzle .....</b>	<b>9</b>
<b>■ Chapter 4: Datetime Conversion for the Command Line .....</b>	<b>23</b>
<b>■ Chapter 5: Testing say() .....</b>	<b>39</b>
<b>■ Chapter 6: Silent-Cron, a Cron Wrapper .....</b>	<b>43</b>
<b>■ Chapter 7: Stateful Silent-Cron .....</b>	<b>61</b>
<b>■ Chapter 8: Review of the Perl 6 Basics .....</b>	<b>67</b>
<b>■ Chapter 9: Parsing INI Files Using Regexes and Grammars .....</b>	<b>73</b>
<b>■ Chapter 10: A File and Directory Usage Graph .....</b>	<b>95</b>
<b>■ Chapter 11: A Unicode Search Tool .....</b>	<b>113</b>
<b>■ Chapter 12: Plotting Using Inline::Python and Matplotlib .....</b>	<b>119</b>
<b>■ Chapter 13: What's Next? .....</b>	<b>135</b>
<b>Index .....</b>	<b>139</b>

# Contents

<b>About the Author .....</b>	<b>xi</b>
<b>About the Technical Reviewer .....</b>	<b>xiii</b>
<b>Acknowledgments .....</b>	<b>xv</b>
<b>Foreword .....</b>	<b>xvii</b>
<b>■ Chapter 1: What Is Perl 6? .....</b>	<b>1</b>
1.1 Perl 5, the Older Sister .....	1
1.2 Library Availability .....	2
1.3 Why Should I Use Perl 6? .....	2
1.4 Summary.....	3
<b>■ Chapter 2: Running Rakudo Perl 6 .....</b>	<b>5</b>
2.1 Installers .....	5
2.2 Docker .....	6
2.3 Building from Source .....	7
2.4 Testing Your Rakudo Star Installation.....	8
2.5 Documentation.....	8
2.6 Summary.....	8
<b>■ Chapter 3: Formatting a Sudoku Puzzle.....</b>	<b>9</b>
3.1 Making the Sudoku Playable.....	12
3.2 Shortcuts, Constants, and More Shortcuts.....	16

3.3	I/O and Other Tragedies.....	18
3.4	Get Creative! .....	20
3.5	Summary.....	21
■	<b>Chapter 4: Datetime Conversion for the Command Line.....</b>	<b>23</b>
4.1	Libraries to the Rescue .....	23
4.2	DateTime Formatting .....	26
4.3	Looking the Other Way .....	28
4.4	Dealing with Time .....	30
4.5	Tighten Your Seat Belt.....	31
4.6	MAIN Magic.....	33
4.7	Automated Tests.....	34
4.8	Summary.....	38
■	<b>Chapter 5: Testing say() .....</b>	<b>39</b>
5.1	Summary.....	42
■	<b>Chapter 6: Silent-Cron, a Cron Wrapper .....</b>	<b>43</b>
6.1	Running Commands Asynchronously .....	43
6.2	Implementing Timeouts.....	46
6.3	More on Promises .....	48
6.4	Possible Extensions .....	51
6.5	Refactoring and Automated Tests .....	51
6.5.1	Refactoring.....	51
6.5.2	Mocking and Testing.....	53
6.5.3	Improving Reliability and Timing .....	58
6.5.4	Installing a Module .....	58
6.6	Summary.....	59

■ <b>Chapter 7: Stateful Silent-Cron</b>	<b>61</b>
7.1 Persistent Storage	61
7.2 Developing the Storage Back End	62
7.3 Using the Storage Back End	65
7.4 Room for Expansion	66
7.5 Summary	66
■ <b>Chapter 8: Review of the Perl 6 Basics</b>	<b>67</b>
8.1 Variables and Scoping	67
8.2 Subroutines	67
8.3 Classes and Objects	69
8.4 Concurrency	71
8.5 Outlook	71
■ <b>Chapter 9: Parsing INI Files Using Regexes and Grammars</b>	<b>73</b>
9.1 Regex Basics	74
9.1.1 Character Classes	75
9.1.2 Quantifiers	75
9.1.3 Alternatives	76
9.2 Parsing the INI Primitives	76
9.3 Putting Things Together	79
9.4 Backtracking	80
9.5 Grammars	82
9.6 Extracting Data from the Match	83
9.7 Generating Good Error Messages	88
9.7.1 Failure Is Normal	88
9.7.2 Detecting Harmful Failure	89
9.7.3 Providing Context	90
9.7.4 Shortcuts for Parsing Matching Pairs	92

9.8	Write Your Own Grammars .....	93
9.9	Summary.....	93
■	<b>Chapter 10: A File and Directory Usage Graph .....</b>	<b>95</b>
10.1	Reading File Sizes.....	95
10.2	Generating a Tree Map .....	97
10.3	Flame Graphs .....	101
10.4	Functional Refactorings .....	103
10.5	More Language Support for Functional Programming.....	109
10.6	More Improvements .....	110
10.7	Explore! .....	111
10.8	Summary.....	112
■	<b>Chapter 11: A Unicode Search Tool.....</b>	<b>113</b>
11.1	Code Points, Grapheme Clusters, and Bytes .....	115
11.2	Numbers.....	116
11.3	Other Unicode Properties .....	117
11.4	Collation .....	117
11.5	Summary.....	118
■	<b>Chapter 12: Plotting Using Inline::Python and Matplotlib.....</b>	<b>119</b>
12.1	Extracting the Stats.....	119
12.2	Plotting with Python .....	120
12.3	Bridging the Gap .....	122
12.4	Using the Bridge to Plot .....	123
12.5	Stacked Plots .....	125



12.6	Idiomatic Use of Inline::Python.....	129
12.6.1	Types of Python APIs.....	129
12.6.2	Mapping the Function API.....	130
12.6.3	An Object-Oriented Interface .....	132
12.7	Summary.....	134
■	<b>Chapter 13: What's Next? .....</b>	<b>135</b>
13.1	Scaling Your Code Base.....	135
13.2	Packaging Your Application .....	136
13.2.1	Packaging as a Traditional Perl 6 Module.....	136
13.2.2	Deploying with Docker .....	137
13.2.3	Windows Installers .....	137
13.3	Closing Thoughts.....	137
	<b>Index.....</b>	<b>139</b>

# 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 Mattijssen, 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.