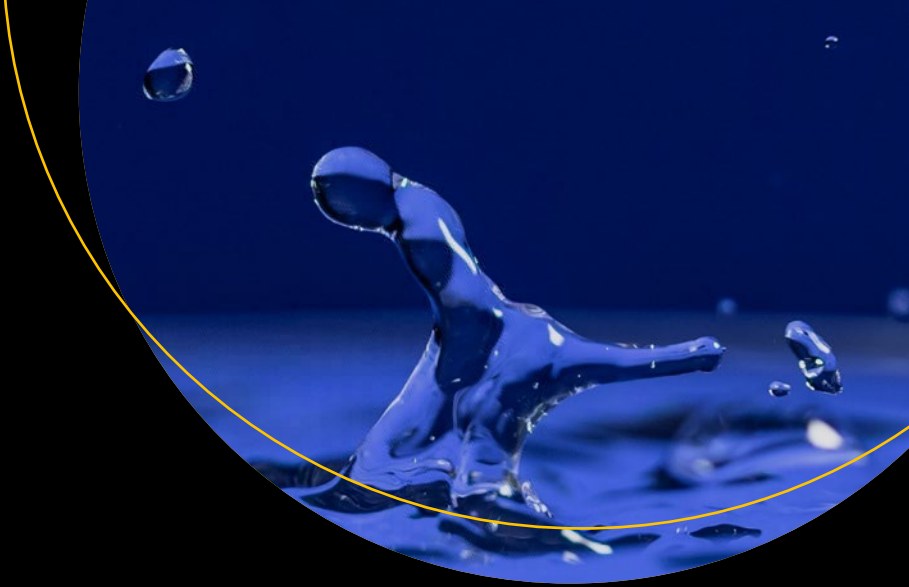




Raku Fundamentals

A Primer with Examples, Projects,
and Case Studies

—

Second Edition

—

Moritz Lenz

Foreword by Larry Wall, creator of Raku

Apress®

Raku Fundamentals

**A Primer with Examples,
Projects, and Case Studies**

Second Edition

Moritz Lenz

Foreword by Larry Wall, creator of Raku

Apress®

Raku Fundamentals: A Primer with Examples, Projects, and Case Studies

Moritz Lenz
Fürth, Bayern, Germany

ISBN-13 (pbk): 978-1-4842-6108-8
<https://doi.org/10.1007/978-1-4842-6109-5>

ISBN-13 (electronic): 978-1-4842-6109-5

Copyright © 2020 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 author 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.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image by Stephen Hume on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Springer Science+Business Media, 1 New York Plaza, New York, NY 10004, U.S.A.. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit 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 booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

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/9781484261088. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

Table of Contents

About the Author	ix
About the Technical Reviewer	xi
Acknowledgments	xiii
Foreword	xv
Chapter 1: What Is Raku?	1
1.1 Intended Audience	2
1.2 Perl 5: The Older Sister	3
1.3 Library Availability.....	4
1.4 Why Should I Use Raku?	4
1.5 Summary.....	6
Chapter 2: Running Rakudo.....	7
2.1 Installers	8
2.2 Docker.....	9
2.3 Building from Source	10
2.4 Testing Your Rakudo Star Installation.....	11
2.5 Documentation.....	12
2.6 Summary.....	12

TABLE OF CONTENTS

Chapter 3: Formatting a Sudoku Puzzle13

 3.1 Making the Sudoku Playable..... 17

 3.2 Shortcuts, Constants, and More Shortcuts.....22

 3.3 I/O and Other Tragedies.....24

 3.4 Get Creative!26

 3.5 Summary.....28

Chapter 4: Datetime Conversion for the Command Line.....31

 4.1 Libraries to the Rescue32

 4.2 DateTime Formatting36

 4.3 Looking the Other Way38

 4.4 Dealing with Time41

 4.5 Tighten Your Seat Belt.....42

 4.6 MAIN Magic.....45

 4.7 Automated Tests.....46

 4.8 Summary.....51

Chapter 5: Testing say()53

 5.1 Summary.....57

Chapter 6: Silent-Cron: A Cron Wrapper59

 6.1 Running Commands Asynchronously.....60

 6.2 Implementing Timeouts64

 6.3 More on Promises66

 6.4 Possible Extensions69

 6.5 Refactoring and Automated Tests69

6.5.1 Refactoring	70
6.5.2 Mocking and Testing.....	72
6.5.3 Improving Reliability and Timing	78
6.5.4 Installing a Module	79
6.6 Summary.....	80
Chapter 7: Stateful Silent-Cron.....	81
7.1 Persistent Storage	81
7.2 Developing the Storage Back End.....	83
7.3 Using the Storage Back End.....	87
7.4 Room for Expansion	88
7.5 Summary.....	89
Chapter 8: Review of the Raku Basics.....	91
8.1 Variables and Scoping.....	91
8.2 Subroutines.....	92
8.3 Classes and Objects.....	94
8.4 Concurrency	96
8.5 Outlook.....	97
Chapter 9: Parsing INI Files Using Regexes and Grammars	99
9.1 Regex Basics.....	100
9.1.1 Character Classes.....	102
9.1.2 Quantifiers	103
9.1.3 Alternatives.....	104
9.2 Parsing the INI Primitives.....	104
9.3 Putting Things Together	108
9.4 Backtracking	109
9.5 Grammars	112

TABLE OF CONTENTS

9.6 Extracting Data from the Match	113
9.7 Generating Good Error Messages	120
9.7.1 Failure Is Normal	120
9.7.2 Detecting Harmful Failure	122
9.7.3 Providing Context	123
9.7.4 Shortcuts for Parsing Matching Pairs.....	125
9.8 Write Your Own Grammars	126
9.9 Summary.....	126
Chapter 10: A File and Directory Usage Graph	129
10.1 Reading File Sizes.....	129
10.2 Generating a Tree-Map.....	132
10.3 Flame Graphs	137
10.4 Functional Refactorings	140
10.5 More Language Support for Functional Programming.....	148
10.6 More Improvements.....	149
10.7 Explore!	151
10.8 Summary.....	152
Chapter 11: A Unicode Search Tool.....	153
11.1 Code Points, Grapheme Clusters, and Bytes	156
11.2 Numbers	158
11.3 Other Unicode Properties	158
11.4 Collation	159
11.5 Summary.....	160

Chapter 12: Creating a Web Service and Declarative APIs	161
12.1 Getting Started with Cro.....	161
12.2 Expanding the Service	165
12.3 Testing.....	167
12.4 Adding a Web Page	170
12.5 Declarative APIs	173
12.6 Summary.....	176
Chapter 13: What's Next?	177
13.1 Scaling Your Code Base	177
13.2 Packaging Your Application.....	178
13.2.1 Packaging As a Traditional Raku Module.....	179
13.2.2 Deploying with Docker	180
13.2.3 Windows Installers	180
13.3 Closing Thoughts.....	181
Index.....	183

About the Author



Moritz Lenz is a software engineer and architect. In the Raku community, he is well known for his contributions to the Raku programming language, the Rakudo compiler, related test suite, infrastructure, and tools. He is also a prolific Perl and Python programmer.

About the Technical Reviewer

Matt Wade is a programmer, database developer, and system administrator. He works for a large financial firm running a production support organization where he gets to solve puzzles all day long. Matt resides in Jacksonville, Florida, with his wife, Michelle, and their children, Matthew and Jonathan.

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, Alexander Kiryuhin, Aleks-Daniel Jakimenko-Aleksejev, Matt Wade, and Massimo Nardone.

Special thanks go to Larry Wall for creating Perl and Raku, 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 Perl¹ 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.

¹Raku started its life as Perl 6; it was renamed after Mr. Wall has written this foreword.

FOREWORD

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 the 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 Raku?

Raku 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 Raku,” while giving the experienced programmer freedom of expression, from concise to poetic.

Raku 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 Raku. It has started its life under the name “Perl 6” but has been renamed in 2019 to break the illusion that it is just another version of Perl. Besides the obvious Perl influence, it contains inspirations from Ruby, Haskell, Smalltalk, and many other languages.

You can write imperative, object-oriented, and functional programs in Raku. We will see object-oriented programming starting from Chapter 5 and a refactoring with functional approaches in Sections 10.4 and 10.5. Declarative programming is supported through features like multiple dispatch, subtyping, and the regex and grammar engine (explored in Chapter 9).

Most lookups in Raku are lexical, and the language avoids global state. This makes parallel and concurrent execution of programs easier, as does Raku'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.

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

1.1 Intended Audience

To get the most out of this book, you should be interested in learning the Raku programming language and have some basic familiarity with programming.

You should know what variables, if statements, and loops are and have used some mechanisms for structuring code, be it through functions, subroutines, methods, or similar constructs. A basic understanding of object-oriented concepts such as classes, objects or instances, and methods helps but is not required.

A basic knowledge of data types such as numbers, strings (text), arrays or lists, and hashes (often also called hash maps, dictionaries, or maps) is also assumed.

If you lack this knowledge, *Think Perl 6* by Allen Downey and Laurent Rosenfeld (2017, O'Reilly Media) is good introduction.

Finally, this book is not a reference, so it assumes you are at least somewhat comfortable with looking things up, usually through the search engine of your choice or the official documentation at <https://docs.raku.org/>.

1.2 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.

During this time, Perl 5 also evolved, and the two languages are different enough that most Perl 5 developers don't consider Perl 6 a natural upgrade path anymore, to the point that Perl 6 does not try to obsolete Perl 5 (at least not more than it tries to obsolete any other programming language :-), and the first stable release of Perl 6 in 2015 does not indicate any lapse in support for Perl 5. The rename of Perl 6 to Raku solidified the intention of both communities to continue development of Perl and Raku separately but with collaboration through shared workshops and conferences.

Raku provides several features that have not found their way into Perl 5, mostly because they seem to require backward incompatible changes or changes too big for the fairly conservative Perl 5 developers:

- An easy-to-use, powerful object model, includes a meta-object model, built into the language.
- A rich collection of built-in types.
- A clear distinction between binary data and strings.
- A solid approach to concurrent execution with threads.
- Built-in grammars and a cleaned-up regex syntax.

On the other hand, Perl scores with maturity and an excellent track record of backward compatibility, a huge ecosystem of libraries, and predictable (and often, but not always) superior performance.

1.3 Library Availability

Being a relatively young language, Raku lacks the mature module ecosystem that languages such as Perl 5 and Python provide.

Nonetheless, some excellent, open source modules exist for Raku. One example is the [Cro](https://cro.services/)¹ HTTP framework for both client- and serverside HTTP applications, including support for HTTP/2 and reactive programming. Another is Red,² a cross-database object-relation mapper that makes use of Raku's extensive meta-programming capabilities to provide a smooth interface.

If you still find yourself missing libraries, interfaces exist that allow you to call into libraries written in C, Python, Perl 5, and Ruby. The Perl 5 and Python interfaces are sophisticated enough that you can write a Raku class that subclasses a class written in either language and the other way around.

So if you like a particular Python library, for example, you can simply load it into your Raku program through the `Inline::Python` module.

1.4 Why Should I Use Raku?

If you like the quick prototyping experience from dynamically typed programming languages, but you also want enough safety features to build big, reliable applications, Raku is a good fit for you. Its gradual typing allows you to write code without having a full picture of the types involved, and later introduce type constraints to guard against future misuse of your internal and external APIs.

Perl has a long history of making text processing via regular expressions (*regexes*) very easy, but more complicated regexes have acquired a reputation of being hard to read and maintain. Raku solves this

¹<https://cro.services/>

²<https://modules.raku.org/dist/Red:cpan:FCO>

by putting regexes on the same level as code, allowing you to name them like subroutines and even to use object-oriented features such as class inheritance and role composition to manage code and regex reuse. The resulting grammars are very powerful and easy to read. In fact, the Rakudo compiler parses Raku source code with a grammar!

Speaking of text, Raku has amazing Unicode support. If you ask your user for a number, and they enter it with digits that don't happen to be the Arabic digits from the ASCII range, Raku still has you covered. And if you deal with graphemes that cannot be expressed as a single Unicode code point, Raku still presents it as a single character.

There are more technical benefits that I could list, but more importantly, the language is designed to be fun to use. An important aspect of that is good error messages. Have you ever been annoyed at Python for typically giving just `SyntaxError: invalid syntax` when something's wrong? This error could come from forgetting a closing parenthesis, for example. In this case, Rakudo says

```
Unable to parse expression in argument list; couldn't find
final ')'
```

which actually tells you what's wrong. But this is just the tip of the iceberg. The compiler catches common mistakes and points out possible solutions and even suggests fixes for spelling mistakes. The Raku community considers error messages that are *less than awesome*, short LTA, to be worthy of bug reports, and much effort is spent into raising the bar for error messages.

Finally, Raku gives you the freedom to express your problem domain and solution in different ways and with different programming paradigms. And if the options provided by the core language are not enough, it is designed with extensibility in mind, allowing you to introduce both new semantics for object-oriented code and new syntax.

1.5 Summary

Raku is a flexible programming language that offers many cool and convenient features to both beginners and experts. It offers flexibility, type checking, and powerful Unicode and text processing support.