Book Half title

The Rust Programming Language

Steve Klabnik and Carol Nichols

Book Title. Copyright (c)2023 by Steve Klabnik and Carol Nichols.

PROD: insert FSC FPO in layout

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Printed in the United States of America

First printing

24 23 22 21 20     1 2 3 4 5 6 7 8 9

ISBN-13: 978-1-xxxx-xxxx-x (print)  
ISBN-13: 978-1-xxxx-xxxx-x (ebook)

PROD: hyphenation of ISBN should be 978-1-59327-xxx-x or 978-1-7185-xxxx-x depending on which publisher code is used

Publisher: William Pollock

Managing Editor: Jill Franklin

Production Editor: ???

Developmental Editor: Liz Chadwick

Cover Design: Octopod Studios

Interior Design: Octopod Studios

Technical Reviewer: JT

Copyeditor: ???

Compositor: Happenstance Type-O-Rama

Proofreader: ???

Indexer: ???

PROD: Compositor and Interior Design should be checked and potentially changed if it is not a yellow and black series book

The following images are reproduced with permission: Figure x-x from *example.com*; Figure x-x courtesy

of Organization; Figure x-x photo by Photographer Name

PROD: delete image credits or change as needed

For information on book distributors or translations, please contact No Starch Press, Inc. directly:  
No Starch Press, Inc.  
245 8th Street, San Francisco, CA 94103  
phone: 1-415-863-9900; info@nostarch.com  
www.nostarch.com

*Library of Congress Cataloging-in-Publication Data*

<CIP DATA GOES HERE.>

PROD: hide or remove summary from cip block before copying here

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

PROD: only include this lego paragraph in lego books (and leave out parts, for example mindstorms, if not relevant)

LEGO®, MINDSTORMS®, the brick configuration, and the minifigure are trademarks of the LEGO Group, which does not sponsor, authorize, or endorse this book.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

PROD: only include one of these fiction paragraphs if it is a work of fiction (and choose which one based on the use of real figures).

All characters in this publication are fictitious or are used fictitiously.

All characters in this publication are fictitious, and any resemblance to real persons, living or dead, is purely coincidental.

Start dedication.

About the Author

Carol Nichols is a member of the Rust Crates.io Team and a former member of the Rust Core Team. She’s a co-founder of Integer 32, LLC, the world’s first Rust-focused software consultancy. Nichols has also organized the Rust Belt Rust Conference.

Steve Klabnik was the lead for the Rust documentation team and was one of Rust’s core developers. A frequent speaker and a prolific open source contributor, he previously worked on projects such as Ruby and Ruby on Rails.

About the Technical Reviewer

JT is a Rust core team member and the co-creator of the Rust error message format, Rust Language Server (RLS), and Nushell. They first started using Rust in 2011, and in 2016 joined Mozilla to work on Rust full-time, helping to shape its direction for widespread use. These days, they are a freelance Rust trainer and advocate for safe systems programming.

Brief Contents

TK after correction pass.

Contents in Detail

TK after correction pass.

A frontmatter could have a foreword and/or a Preface.

Foreword

It wasn’t always so clear, but the Rust programming language is fundamentally about \*empowerment\*: no matter what kind of code you are writing now, Rust empowers you to reach farther, to program with confidence in a wider variety of domains than you did before.

Take, for example, “systems-level” work that deals with low-level details of memory management, data representation, and concurrency. Traditionally, this realm of programming is seen as arcane, accessible only to a select few who have devoted the necessary years learning to avoid its infamous pitfalls. And even those who practice it do so with caution, lest their code be open to exploits, crashes, or corruption.

Rust breaks down these barriers by eliminating the old pitfalls and providing a friendly, polished set of tools to help you along the way. Programmers who need to “dip down” into lower-level control can do so with Rust, without taking on the customary risk of crashes or security holes, and without having to learn the fine points of a fickle toolchain. Better yet, the language is designed to guide you naturally towards reliable code that is efficient in terms of speed and memory usage.

Programmers who are already working with low-level code can use Rust to raise their ambitions. For example, introducing parallelism in Rust is a relatively low-risk operation: the compiler will catch the classical mistakes for you. And you can tackle more aggressive optimizations in your code with the confidence that you won’t accidentally introduce crashes or vulnerabilities.

But Rust isn’t limited to low-level systems programming. It’s expressive and ergonomic enough to make CLI apps, web servers, and many other kinds of code quite pleasant to write — you’ll find simple examples of both later in the book. Working with Rust allows you to build skills that transfer from one domain to another; you can learn Rust by writing a web app, then apply those same skills to target your Raspberry Pi.

This book fully embraces the potential of Rust to empower its users. It’s a friendly and approachable text intended to help you level up not just your knowledge of Rust, but also your reach and confidence as a programmer in general. So dive in, get ready to learn—and welcome to the Rust community!

Nicholas Matsakis and Aaron Turon

Preface

This version of the text assumes you’re using Rust 1.62.0 (released 2022-06-30) or later with edition="2021" in Cargo.toml of all projects to use Rust 2021 Edition idioms. See “Installation” on page 1 for instructions on installing or update Rust, and see Appendix E for information on editions.

The 2021 Edition of the Rust language includes a number of improvements that make Rust more ergonomic and correct some inconsistencies. On top of a general update to reflect these improvements, this rendition of the book has a number of improvements to address specific feedback:

* Chapter 7 contains a new quick reference section on organizing your code into multiple files with modules.
* Chapter 13 has new and improved closure examples that more clearly illustrate captures, the move keyword, and the Fn traits.
* We fixed a number of small errors and imprecise wording throughout the book. Thank you to the readers who reported them!

Note that any code from earlier renditions of this book that compiled will continue to compile with the relevant edition in the project’s Cargo.toml, even as you update the Rust compiler version you’re using. That’s Rust’s backward compatibility guarantees at work!

ACKNOWLEDGMENTS

We would like to thank everyone who has worked on the Rust language for creating an amazing language worth writing a book about. We’re grateful to everyone in the Rust community for being welcoming and creating an environment worth welcoming more folks into.

We’re especially thankful for everyone who read early versions of this book online and provided feedback, bug reports, and pull requests. Special thanks to Eduard-Mihai Burtescu, Alex Crichton, and JT for providing technical review and Karen Rustad Tölva for the cover art. Thank you to our team at No Starch, including Bill Pollock, Liz Chadwick, and Janelle Ludowise, for improving this book and bringing it to print.

Carol is grateful for the opportunity to work on this book. She thanks her family for their constant love and support, especially her husband Jake Goulding and her daughter Vivian.

Introduction

Welcome to The Rust Programming Language, an introductory book about Rust. The Rust programming language helps you write faster, more reliable software. High-level ergonomics and low-level control are often at odds in programming language design; Rust challenges that conflict. Through balancing powerful technical capacity and a great developer experience, Rust gives you the option to control low-level details (such as memory usage) without all the hassle traditionally associated with such control.

Who Rust Is For

Rust is ideal for many people for a variety of reasons. Let’s look at a few of the most important groups.

Teams of Developers

Rust is proving to be a productive tool for collaborating among large teams of developers with varying levels of systems programming knowledge. Low-level code is prone to various subtle bugs, which in most other languages can be caught only through extensive testing and careful code review by experienced developers. In Rust, the compiler plays a gatekeeper role by refusing to compile code with these elusive bugs, including concurrency bugs. By working alongside the compiler, the team can spend their time focusing on the program’s logic rather than chasing down bugs.

Rust also brings contemporary developer tools to the systems programming world:

* Cargo, the included dependency manager and build tool, makes adding, compiling, and managing dependencies painless and consistent across the Rust ecosystem.
* The Rustfmt formatting tool ensures a consistent coding style across developers.
* The Rust Language Server powers Integrated Development Environment (IDE) integration for code completion and inline error messages.

By using these and other tools in the Rust ecosystem, developers can be productive while writing systems-level code.

Students

Rust is for students and those who are interested in learning about systems concepts. Using Rust, many people have learned about topics like operating systems development. The community is very welcoming and happy to answer student questions. Through efforts such as this book, the Rust teams want to make systems concepts more accessible to more people, especially those new to programming.

Companies

Hundreds of companies, large and small, use Rust in production for a variety of tasks, including command line tools, web services, DevOps tooling, embedded devices, audio and video analysis and transcoding, cryptocurrencies, bioinformatics, search engines, Internet of Things applications, machine learning, and even major parts of the Firefox web browser.

Open Source Developers

Rust is for people who want to build the Rust programming language, community, developer tools, and libraries. We’d love to have you contribute to the Rust language.

People Who Value Speed and Stability

Rust is for people who crave speed and stability in a language. By speed, we mean both how quickly Rust code can run and the speed at which Rust lets you write programs. The Rust compiler’s checks ensure stability through feature additions and refactoring. This is in contrast to the brittle legacy code in languages without these checks, which developers are often afraid to modify. By striving for zero-cost abstractions, higher-level features that compile to lower-level code as fast as code written manually, Rust endeavors to make safe code be fast code as well.

The Rust language hopes to support many other users as well; those mentioned here are merely some of the biggest stakeholders. Overall, Rust’s greatest ambition is to eliminate the trade-offs that programmers have accepted for decades by providing safety and productivity, speed and ergonomics. Give Rust a try and see if its choices work for you.

Who This Book Is For

This book assumes that you’ve written code in another programming language but doesn’t make any assumptions about which one. We’ve tried to make the material broadly accessible to those from a wide variety of programming backgrounds. We don’t spend a lot of time talking about what programming is or how to think about it. If you’re entirely new to programming, you would be better served by reading a book that specifically provides an introduction to programming.

How to Use This Book

In general, this book assumes that you’re reading it in sequence from front to back. Later chapters build on concepts in earlier chapters, and earlier chapters might not delve into details on a particular topic but will revisit the topic in a later chapter.

You’ll find two kinds of chapters in this book: concept chapters and project chapters. In concept chapters, you’ll learn about an aspect of Rust. In project chapters, we’ll build small programs together, applying what you’ve learned so far. Chapters 2, 12, and 20 are project chapters; the rest are concept chapters.

Chapter 1 explains how to install Rust, how to write a “Hello, world!” program, and how to use Cargo, Rust’s package manager and build tool. Chapter 2 is a hands-on introduction to writing a program in Rust, having you build up a number guessing game. Here we cover concepts at a high level, and later chapters will provide additional detail. If you want to get your hands dirty right away, Chapter 2 is the place for that. Chapter 3 covers Rust features that are similar to those of other programming languages, and in Chapter 4 you’ll learn about Rust’s ownership system. If you’re a particularly meticulous learner who prefers to learn every detail before moving on to the next, you might want to skip Chapter 2 and go straight to Chapter 3, returning to Chapter 2 when you’d like to work on a project applying the details you’ve learned.

Chapter 5 discusses structs and methods, and Chapter 6 covers enums, match expressions, and the if let control flow construct. You’ll use structs and enums to make custom types in Rust.

In Chapter 7, you’ll learn about Rust’s module system and about privacy rules for organizing your code and its public Application Programming Interface (API). Chapter 8 discusses some common collection data structures that the standard library provides, such as vectors, strings, and hash maps. Chapter 9 explores Rust’s error-handling philosophy and techniques.

Chapter 10 digs into generics, traits, and lifetimes, which give you the power to define code that applies to multiple types. Chapter 11 is all about testing, which even with Rust’s safety guarantees is necessary to ensure your program’s logic is correct. In Chapter 12, we’ll build our own implementation of a subset of functionality from the grep command line tool that searches for text within files. For this, we’ll use many of the concepts we discussed in the previous chapters.

Chapter 13 explores closures and iterators: features of Rust that come from functional programming languages. In Chapter 14, we’ll examine Cargo in more depth and talk about best practices for sharing your libraries with others. Chapter 15 discusses smart pointers that the standard library provides and the traits that enable their functionality.

In Chapter 16, we’ll walk through different models of concurrent programming and talk about how Rust helps you to program in multiple threads fearlessly. Chapter 17 looks at how Rust idioms compare to object-oriented programming principles you might be familiar with.

Chapter 18 is a reference on patterns and pattern matching, which are powerful ways of expressing ideas throughout Rust programs. Chapter 19 contains a smorgasbord of advanced topics of interest, including unsafe Rust, macros, and more about lifetimes, traits, types, functions, and closures.

In Chapter 20, we’ll complete a project in which we’ll implement a low-level multithreaded web server!

Finally, some appendices contain useful information about the language in a more reference-like format. Appendix A covers Rust’s keywords, Appendix B covers Rust’s operators and symbols, Appendix C covers derivable traits provided by the standard library, Appendix D covers some useful development tools, and Appendix E explains Rust editions.

There is no wrong way to read this book: if you want to skip ahead, go for it! You might have to jump back to earlier chapters if you experience any confusion. But do whatever works for you.

An important part of the process of learning Rust is learning how to read the error messages the compiler displays: these will guide you toward working code. As such, we’ll provide many examples that don’t compile along with the error message the compiler will show you in each situation. Know that if you enter and run a random example, it may not compile! Make sure you read the surrounding text to see whether the example you’re trying to run is meant to error. In most situations, we’ll lead you to the correct version of any code that doesn’t compile.

Resources and How to Contribute to This Book

This book is open source. If you find an error, please don’t hesitate to file an issue or send a pull request on GitHub at <https://github.com/rust-lang/book/>. Please see CONTRIBUTING.md at <https://github.com/rust-lang/book/blob/main/CONTRIBUTING.md> for more details.

The source code for the examples in this book, errata, and other information are available at <https://www.nostarch.com/Rust2021/>.