The rust programming language

The Rust Programming Language

3rd Edition

by Steve Klabnik, Carol Nichols, and Chris Krycho, with Contributions from The Rust Community

No Starch Press

The Rust Programming Language, 3rd Edition. Copyright © 2026 by the Rust Foundation and the Rust Project Developers.

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

30 29 28 27 26     1 2 3 4 5

ISBN-13: 978-1-7185-0444-8 (print)  
ISBN-13: 978-1-7185-0445-5 (ebook)

COMP: insert nsp logo to the left of the address block; ® is 10pts

Published by No Starch Press®, Inc.  
245 8th Street, San Francisco, CA 94103  
phone: +1.415.863.9900  
www.nostarch.com; info@nostarch.com

Publisher: William Pollock  
Managing Editor: Jill Franklin  
Production Manager: Sabrina Plomitallo-González  
Production Editor: Allison Felus  
Developmental Editor: ???  
Cover Illustrator: Karen Rustad Tölva  
Interior Design: Octopod Studios  
Technical Reviewer: Sophia Turner  
Copyeditor: Audrey Doyle  
Proofreader: ???  
Indexer: ???

The Library of Congress has cataloged the first edition as follows:

Names: Klabnik, Steve, author. | Nichols, Carol, 1983- eauthor.

Title: The Rust programming language / by Steve Klabnik and Carol Nichols ; with contributions from the Rust Community.

Description: San Francisco : No Starch Press, Inc., 2018. | Includes index. Identifiers: LCCN 2018014097 (print) | LCCN 2018019844 (ebook) | ISBN 9781593278519 (epub) | ISBN 1593278519 (epub) | ISBN 9781593278281 (paperback) | ISBN 1593278284 (paperback)

Subjects: LCSH: Rust (Computer programming language) | BISAC: COMPUTERS / Programming / Open Source. | COMPUTERS / Programming Languages / General. | COMPUTERS / Programming / General.

Classification: LCC QA76.73.R87 (ebook) | LCC QA76.73.R87 K53 2018 (print) | DDC 005.13/3--dc23

LC record available at https://lccn.loc.gov/2018014097

For customer service inquiries, please contact info@nostarch.com. For information on distribution, bulk sales, corporate sales, or translations: sales@nostarch.com. For permission to translate this work: rights@nostarch.com. To report counterfeit copies or piracy: counterfeit@nostarch.com. The authorized representative in the EU for product safety and compliance is EU Compliance Partner, Pärnu mnt. 139b-14, 11317 Tallinn, Estonia, hello@eucompliancepartner.com, +3375690241.

No Starch Press and the No Starch Press iron 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.

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 authors 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.

[S]

Start dedication.

Example: This one goes out to the one I love.

About the Authors

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.

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. She has also organized the Rust Belt Rust Conference.

Chris Krycho is a software engineering leader with experience in avionics, developer tools, web frameworks, and more. In addition to his open source software contributions and regular public speaking, he created the New Rustacean podcast (2015–2019).

About the Technical Reviewer

Sophia Turner has been a long-time advocate of better programming languages and tools. She helped create TypeScript and later joined Mozilla to work on compiler errors, IDE support, and the larger Rust ergonomics initiative. She also served on the Rust core team and helped create the Rust Leadership Council.

Brief Contents

COMP: run brief toc in first pages.

Contents in Detail

COMP: run contents in detail in first pages.

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

Foreword

The Rust programming language has come a long way in a few short years, from its creation and incubation by a small and nascent community of enthusiasts, to becoming one of the most loved and in-demand programming languages in the world. Looking back, it was inevitable that the power and promise of Rust would turn heads and gain a foothold in systems programming. What was not inevitable was the global growth in interest and innovation that permeated through open source communities and catalyzed wide-scale adoption across industries.

At this point in time, it is easy to point to the wonderful features that Rust has to offer to explain this explosion in interest and adoption. Who doesn’t want memory safety, and fast performance, and a friendly compiler, and great tooling, among a host of other wonderful features? The Rust language you see today combines years of research in systems programming with the practical wisdom of a vibrant and passionate community. This language was designed with purpose and crafted with care, offering developers a tool that makes it easier to write safe, fast, and reliable code.

But what makes Rust truly special is its roots in empowering you, the user, to achieve your goals. This is a language that wants you to succeed, and the principle of empowerment runs through the core of the community that builds, maintains, and advocates for this language. Since the previous edition of this definitive text, Rust has further developed into a truly global and trusted language. The Rust Project is now robustly supported by the Rust Foundation, which also invests in key initiatives to ensure that Rust is secure, stable, and sustainable.

This edition of The Rust Programming Language is a comprehensive update, reflecting the language’s evolution over the years and providing valuable new information. But it is not just a guide to syntax and libraries—it’s an invitation to join a community that values quality, performance, and thoughtful design. Whether you’re a seasoned developer looking to explore Rust for the first time or an experienced Rustacean looking to refine your skills, this edition offers something for everyone.

The Rust journey has been one of collaboration, learning, and iteration. The growth of the language and its ecosystem is a direct reflection of the vibrant, diverse community behind it. The contributions of thousands of developers, from core language designers to casual contributors, are what make Rust such a unique and powerful tool. By picking up this book, you’re not just learning a new programming language—you’re joining a movement to make software better, safer, and more enjoyable to work with.

Welcome to the Rust community!

Bec Rumbul, Executive Director of the Rust Foundation

Preface

This version of the text assumes you’re using Rust 1.85.0 (released 2025-02-17) or later with edition = "2024" in the Cargo.toml file of all projects to configure them to use Rust 2024 Edition idioms. See “Installation” on page XX for instructions on installing or updating Rust, and see Appendix E for information on editions.

The 2024 Edition of the Rust language includes a number of improvements that make Rust more ergonomic and that 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:

* A new Chapter 17 introduces async programming in Rust, including async and await along with the Future and Stream types. Later chapters have been lightly adapted and renumbered to account for this update.
* Chapter 20 (Chapter 19 in previous editions) now includes an introduction to Miri, Rust’s dynamic analysis tool for unsafe code. It also includes updates for some of the more significant updates to Rust in the 2024 Edition.
* 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 guarantee 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 Sophia Turner for providing technical review, and to Karen Rustad Tölva for the cover art. Thank you to our team at No Starch Press, 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.

Chris is profoundly grateful to Tim McNamara, Will Crichton, James Munns, Nick Cameron, and Tyler Mandry for dedicated feedback on the new chapter on async programming in Rust. He would also like to thank Integer 32 for making this update for the 2024 Edition happen. Finally, and most of all, he is thankful to his wife, Jaimie, and his daughters, Elayne and Katherine, for supporting him, including many a “How is the book going, daddy?” over family dinners.

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 only be caught 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 its 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 students’ 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 it 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. Chapter 2, Chapter 12, and Chapter 21 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. 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, which covers Rust features that are similar to those of other programming languages; then, you can return to Chapter 2 when you’d like to work on a project applying the details you’ve learned.

In Chapter 4, you’ll learn about Rust’s ownership system. Chapter 5 discusses structs and methods. Chapter 6 covers enums, match expressions, and the if let and let...else control flow constructs. You’ll use structs and enums to make custom types.

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: 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 that 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 program in multiple threads fearlessly. In Chapter 17, we build on that by exploring Rust’s async and await syntax, along with tasks, futures, and streams, and the lightweight concurrency model they enable.

Chapter 18 looks at how Rust idioms compare to object-oriented programming principles you might be familiar with. Chapter 19 is a reference on patterns and pattern matching, which are powerful ways of expressing ideas throughout Rust programs. Chapter 20 contains a smorgasbord of advanced topics of interest, including unsafe Rust, macros, and more about lifetimes, traits, types, functions, and closures.

In Chapter 21, we’ll complete a project in which we’ll implement a low-level multithreaded web server!

Finally, some appendixes 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://nostarch.com/rust-programming-language-3rd-edition.