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 a variety of 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 more 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.

Rustfmt 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. Those tasks include 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 the speed of the programs that you can create with Rust and the speed at which Rust lets you write them. The Rust compiler’s checks ensure stability through feature additions and refactoring as opposed to brittle legacy code in languages without these checks that developers are 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.

Although we’ve not provided a complete list of everyone the Rust language hopes to support, those we have mentioned are some of the biggest stakeholders. Overall, Rust’s greatest ambition is to eliminate the dichotomy of the trade-offs that programmers have accepted for decades: 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 topic; we typically 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.

Additionally, Chapter 2 is a hands-on introduction to the Rust language. We’ll 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 one for that. At first, you might even want to skip Chapter 3, which covers Rust features similar to other programming language features, and head straight to Chapter 4 to learn about Rust’s ownership system. However, if you’re a particularly meticulous learner who prefers to learn every detail before moving onto 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 those details.

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 is still necessary even with Rust’s safety guarantees 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 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 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 macros.

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 of code 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.

Contributing to the 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/master/CONTRIBUTING.md> for more details.