|  |
| --- |
| Learning Rust |
|  |
| August, 2025The Rust Programming Language (AKA The Rust book) | |

* The non-experimental version of the book is available offline with installations of Rust made with rustup; run rustup doc --book to open.
* Foreword:
  + It wasn’t always so clear, but the Rust programming language is fundamentally about *empowerment.* Rust empowers you to reach farther, **to program with confidence in a wider variety of domains than you did before.**
  + See foreword for how.
* Introduction:
  + Helps you write faster, more reliable software.
  + Rust gives you the option to control low-level details (such as memory usage) without all the hassle traditionally associated with such control.
  + Safety *and* productivity, speed *and* ergonomics.
  + Cargo -> the included dependency manager and build tool.
* Getting started:
  + rustup, a command line tool for managing Rust versions and associated tools.
  + Also need a linker to join compiled outputs into one file. Typically included in C compilers. A C compiler is also useful because some common Rust packages depend on C code and will need a C compiler.
  + rustc -> Rust compiler
  + Check version: rustc –version
  + rustc <filename>.rs to compile a Rust program
  + The main function in a program is special: it is always the first code that runs in every executable Rust program.
* fn *main*() {
* *println!*("Hello, world!");
* }
  + println! calls a Rust macro. If it had called a function instead, it would be entered as println (without the !). Rust macros are a way to write code that generates code to extend Rust syntax. For now, you just need to know that using a ! means that you’re calling a macro instead of a normal function and that macros don’t always follow the same rules as functions.
  + Cargo is Rust’s build system and package manager. Most Rustaceans use this tool to manage their Rust projects because Cargo handles a lot of tasks for you, such as building your code, downloading the libraries your code depends on, and building those libraries. (We call the libraries that your code needs *dependencies*.)
  + As you write more complex Rust programs, you’ll add dependencies, and if you start a project using Cargo, adding dependencies will be much easier to do. Having multiple files will also be easier. Thus, the vast majority of Rust projects use cargo, and hence we shall too!
    - $ cargo new hello\_cargo
    - $ cd hello\_cargo
  + Cargo.toml file (TOML = *Tom’s Obvious, Minimal Language).* Contains config info to compile our program + dependencies, which are referred to as crates.
  + Cargo enforced a specific folder structure too. Cargo helps you organize your projects. There’s a place for everything, and everything is in its place.
  + cargo build
    - Places executable in ./*target/debug/ since the default build is a debug build.*
  + To compile and run in one command: cargo run
  + cargo check
    - Quickly checks your code to make sure it compiles but doesn’t produce an executable.
    - Why would you not want an executable? Often, cargo check is much faster than cargo build because it skips the step of producing an executable.
  + cargo build --release to compile it with optimizations.
    - But it lengthens the time it takes for your program to compile.