# Mastering Rust

### Table of Contents for Learning Rust

1. **Introduction to Rust**
   * What is Rust?
   * Why Rust? (Memory Safety, Concurrency, Speed)
   * Setting up the Rust Environment
   * First Rust Program: "Hello, World!"
   * Rust Toolchain (Cargo, Rustup, rustc)
   * Rust Documentation and Community Resources

* **Basic Rust Syntax**
  + Variables and Constants
  + Data Types (Scalar, Compound)
  + Functions and Control Flow
  + Ownership, Borrowing, and Lifetimes
  + References and Mutable References
  + Pattern Matching (if, else, match)
  + Error Handling (Result, Option, unwrap, expect)
* **Working with Data Structures**
  + Strings (String, &str)
  + Arrays, Slices, and Tuples
  + Vectors
  + HashMaps
  + Enums and Pattern Matching with Enums
  + Structs and Methods
  + Option and Result Types in Detail
* **Memory Management and Ownership System**
  + Understanding Ownership
  + Move Semantics
  + Borrowing and Lifetimes
  + Copy vs. Clone Traits
  + Stack vs. Heap Memory Allocation
  + Managing Memory with Box, Rc, and Arc

* **Advanced Rust Features**
  + Traits and Generics
  + Lifetimes in Functions and Structs
  + Closures and Higher-Order Functions
  + Iterators and the Iterator Trait
  + Advanced Pattern Matching
  + Using Modules and Crates (Managing Dependencies with Cargo)
  + Using Cargo for Project Management (workspaces, testing)
* **Concurrency in Rust**
  + Introduction to Rust's Concurrency Model
  + Threads and Threading with std::thread
  + Shared State Concurrency with Mutex and RwLock
  + Channels and Message Passing Concurrency
  + Understanding Send and Sync Traits
  + The async/await Model for Asynchronous Programming
  + Futures and Async Ecosystem (tokio, async-std)
* **Rust and System-Level Programming**
  + Interfacing with C (FFI: Foreign Function Interface)
  + Unsafe Rust and when to use it
  + Working with Raw Pointers
  + Memory Layout and Alignment
  + Writing Memory-Efficient Code
  + Using the std::fs Module for File I/O
* **Error Handling and Robust Code**
  + Unrecoverable Errors (panic!)
  + Recoverable Errors with Result and Error Handling Patterns
  + Error Propagation with ? Operator
  + Custom Error Types
  + Logging and Debugging with Rust
* **Testing and Benchmarking in Rust**
  + Writing Unit Tests
  + Integration Testing
  + Using Cargo to Run Tests
  + Test Organization and Structuring
  + Benchmarking Code with criterion crate
  + Profiling Rust Programs
* **Macros and Metaprogramming**
  + Introduction to Macros (macro\_rules!)
  + Declarative Macros
  + Procedural Macros (Custom Derive, Attribute-like, Function-like)
  + Writing and Using Macros Effectively
  + Best Practices for Macros in Production Code
* **Building and Packaging Rust Applications**
  + Using Cargo for Builds and Releases
  + Understanding Build Profiles (debug, release)
  + Cross-compiling Rust for Different Platforms
  + Packaging Rust Projects for Distribution (Binaries, Libraries)
  + Publishing Crates to <crates.io>
  + Rust Bindings for Other Languages (C, Python)
* **Web Development with Rust**
  + Introduction to Web Frameworks (Rocket, Actix)
  + Building REST APIs with Rust
  + Working with Asynchronous Programming in Web Servers
  + WebAssembly with Rust (Wasm)
  + Building Frontend Apps with Yew or Seed
  + Connecting Rust with Databases (Diesel, SQLx)
* **Performance Optimization in Rust**
  + Profiling and Optimizing Rust Code
  + Inlining and Loop Unrolling
  + Memory Management and Heap vs. Stack Optimizations
  + Concurrency vs. Parallelism in Rust
  + Performance Tools and Crates (perf, flamegraph)
* **Rust in Embedded Systems**
  + Overview of Rust for Embedded Development
  + Setting Up Rust for Embedded Platforms
  + Interacting with Hardware in Rust
  + Real-Time Systems with Rust
  + Microcontroller Programming
  + Case Studies in Rust Embedded
* **Rust in the Blockchain**
  + Introduction to Blockchain Development in Rust
  + Popular Rust-based Blockchain Projects (Substrate, Solana)
  + Writing Smart Contracts in Rust
  + Interfacing Rust with Web3
  + Using Cryptography Libraries in Rust
* **Contributing to Open-Source Rust**
  + How to Contribute to Rust Core and Crates
  + Understanding Rust’s RFC Process
  + Best Practices for Writing Rust Libraries
  + Building Your Own Crates and Libraries
* **Becoming a Rust Expert**
  + Advanced Type System Features (HRTB, Associated Types)
  + Memory Management in Depth (RefCell, Cell)
  + Building Domain-Specific Languages (DSLs)
  + Advanced Lifetime Management
  + Zero-Cost Abstractions in Rust
  + Design Patterns in Rust
  + Keeping Up with Rust’s Evolving Ecosystem

This table of contents offers a structured pathway to mastering Rust from the basics through advanced topics.

#software/languages/rust