

# Comprehensive Rust

Download Whitepaper: Accelerate Your Modernization Efforts with a Cloud-Native Strategy  
Get Your Free Copy Now

**Course Number: RUST-108**

**Duration: 5 days**

## Overview

### Course Description

This Comprehensive Rust training course teaches attendees to master Rust's unique features, including its focus on safety, robust concurrency capabilities, and ability to optimize performance for even the most demanding applications.

After the basics, participants learn advanced topics, including macros, metaprogramming, and FFI integration while building real-world applications that showcase Rust's versatility and power. By the end of this Rust course, students have the confidence to tackle complex Rust projects, write efficient code, and successfully implement Rust in their software development projects.

This class is appropriate for seasoned developers seeking to expand their skills and those new to programming.

### Skills Gained

- Understand the Rust philosophy
- Set up and navigate the Rust environment

- Grasp basic Rust syntax and semantics
- Use control flow and logic
- Understand ownership and borrowing concepts
- Use tuples, enums, structs, and vectors
- Implement pattern matching
- Work with Rust's concurrency model
- Create custom macros
- Write Rust tests
- Create documentation with Rustdoc

## Prerequisites

- Software development experience; this is not a general introduction to programming course.
- Basic understanding of programming concepts such as variables, expressions, functions, and control flow.

## Training Materials

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub through documentation and extensive code samples.

NOTE: These materials are not related to Google's [Comprehensive Rust](#) training material.

## Software Requirements

- A free, personal GitHub account to access the courseware
- Permission to install Rust and Visual Studio Code on their computers
- Permission to install Rust Crates and Visual Studio Extensions

If students cannot configure a local environment, a cloud-based environment can be provided.

## Audience

## Course Details

## Outline

- Introduction
- What is Rust?
  - Rust's Philosophy and Goals

- History and motivation
- Rust Community
- The Rust Playground
- Install Rust
  - Script
  - macOS Homebrew
  - Platform Installers
- Rust Editors
  - VSCode with Extensions
  - Rust Rover
  - Debug Rust in VSCode
  - GitHub Copilot
- Hello World
  - Create a new Project
  - Main Function
  - Print to the Console
  - Comments
- Cargo
  - What is Cargo?
  - Run Command
  - Build Command
  - Build Release Command
  - Install Third-Party Crates
- Scalar Types and Data
  - Rust Types
  - Constants
  - Immutable Variables
  - Mutable Variables
- Code Logic
  - If Statement
  - Loop with Break

- While Loop
- Functions
  - Define a Function
  - Call a Function
  - Parameter Types
  - Return Types
  - Closure Functions
- Modules
  - Import Modules from Standard Library
  - Import Modules from Third-Party Crates
  - Define Custom Modules
  - Import Custom Modules
- Built-In Macros
  - `print!` and `println!`
  - `format!`
  - `assert!`, `assert_eq!`, and `assert_ne!`
  - `vec!`
  - `include_str!` and `include_bytes!`
  - `cfg!` and `env!`
  - `panic!`
- Memory Management
  - Problems with Manual Management
  - Problems with Garbage Collection
  - Ownership & Borrowing
  - References
  - Lifetimes
- Strings and String Slices
  - What is a String and a String Slice?
  - String Slices
  - String Objects
  - Convert Between Slices and Strings
  - Parse Number from String

- Trim String
- Print Strings with Interpolation
- Tuples
  - What is a Tuple?
  - Heterogeneous Elements
  - Access Elements
  - Destructuring
  - Immutable
- Enums
  - What is an Enum?
  - Define an Enum
  - Using Enums
  - Enum Variants
  - Enum Methods
  - Enums and Pattern Matching
  - Result Enum
  - Option Enum
  - Enums vs Structs
- Structs
  - What is a Struct?
  - Create Instance
  - Field Init Shorthand
  - Struct Update Syntax
  - Tuple Structs
  - Unit-Like Structs
  - Ownership of Struct Data
  - Function Implementation
  - Associated Functions
  - Struct Methods
  - Constructor Pattern
- Vectors
  - What is a Vector?

- Create a Vector
- Add and Remove Elements
- Access Elements
- Iterate over Elements
- Slicing, Length, and Capacity
- Common Vector Operations
- Understand Memory Management
- Ownership and Borrowing Rules
- Collections and Iterators
  - Vectors, arrays, and slices
  - HashMaps and hash sets
  - Iteration and iterators
- Traits
  - What is a trait?
  - How does a trait related to traditional OOP interfaces?
  - Defining a trait
  - Implementing a trait
  - Default implementations
  - Traits as parameters
  - Traits as return types
  - Traits as bounds
- Generics
  - What is a generic?
  - How does a generic related to traditional OOP generics?
  - Defining a generic
  - Implementing a generic
  - Generic bounds
  - Multiple generic types
  - Where clauses
- Pattern Matching
  - What is Pattern Matching?
  - Match Statement

- If Let Statement
- While Let Statement
- Destructuring Structs and Tuples
- Pattern Matching with Enums
- Pattern Matching with Functions
- Pattern Matching and Ownership
- Refutability and Irrefutability
- Concurrent Programming
  - What is Concurrent Programming?
  - Using Multiple Threads
  - Mutex, RwLock, and Arc
  - Message Passing with Channels
  - Sync and Send Traits
  - Futures and Async/Await
- Unsafe Rust
  - What is Unsafe Rust?
  - Raw Pointers
  - Dereferencing Raw Pointers
  - Calling Unsafe Functions
  - Creating Safe Abstractions
  - Unsafe Traits
  - Unsafe Blocks
  - Unsafe Superpowers
- Macros and Metaprogramming
  - What is a Macro?
  - Define a Macro with `macro_rules!`
  - Using Pattern Matching
  - Define Expansion
  - Use the Custom Macro
- Tests
  - What is a Test?
  - Test Functions

- Test Organization
- Test Attributes
- Test Coverage
- assert!, assert\_eq!, and assert\_ne!
- Documentation with Rustdoc
  - What is Rustdoc?
  - Add Documentation to Rust Code
  - Triple-Slash Comments and the #[doc] Attribute
  - Generate Documentation
  - Linking and Cross-Referencing Documentation
- Conclusion