Document Generated: 11/17/2026

Learning Style: Virtual Classroom

Technology:

Difficulty: Beginner

Course Duration: 5 Days

Hands-on Rust Programming: From Basics to Proficiency (TTRS2103)



About This Course:

Rust is a cutting-edge programming language designed for high performance, safety, and concurrent systems. Known for its focus on memory safety without garbage collection and seamless concurrency, Rust empowers developers to create reliable, efficient applications at scale. Whether you're an experienced developer looking to enhance your systems programming skills or exploring Rust for the first

time, this five-day immersive course will help you master the language and build confidence in tackling complex projects.

Hands-on Rust Programming: From Basics through Proficiency dives deep into Rust's unique features, such as ownership and borrowing, pattern matching, and its powerful concurrency model. Led by an expert instructor, the course blends engaging lectures with practical labs to ensure you gain both theoretical knowledge and applied experience. You'll explore topics ranging from Rust's syntax, modules, and traits to advanced concepts like unsafe Rust, macros, and FFI integration, equipping you with the skills needed to build high-performance, safe, and scalable software.

The workshop-style format dedicates half the class time to hands-on exercises, where you'll work on real-world applications, such as writing tests, managing memory effectively, and deploying concurrent systems. By the end of the course, you'll be proficient in Rust, capable of creating robust, well-documented, and efficient programs, and ready to harness Rust's full potential in your projects.

Course Objectives:

- Working in a hands-on learning environment led by our expert instructor you will explore:
- Understand Rust's Philosophy and Setup: Learn the principles behind Rust's design, install and configure the Rust environment, and navigate tools like Cargo and Rust Playground.
- Master Rust Syntax and Semantics: Write efficient code using Rust's unique syntax for control flow, functions, and modules.
- Implement Effective Memory Management: Master ownership, borrowing, and lifetimes to ensure memory safety and efficiency in your applications.
- Harness Rust's Concurrency Model: Build concurrent applications with Rust's tools, including threads, Mutex, RwLock, and async/await for multitasking.
- Utilize Rust's Ecosystem and Tools: Leverage Cargo for dependency management and explore popular crates to enhance your projects.
- Apply Advanced Rust Features: Develop complex solutions using traits, generics, macros, and unsafe Rust for optimized performance and flexibility.
- Build and Document Applications: Write robust tests using Rust's testing framework, and create clear, professional documentation with Rustdoc.

Audience:

- This course is designed for experienced software developers, system

programmers, and engineers eager to adopt Rust for performance-critical and concurrent applications. Whether you're transitioning to systems programming or enhancing your current skill set, this training is ideal for those aiming to leverage Rust for its unique combination of performance, safety, and concurrency.

Prerequisites:

- Basic Programming Knowledge: Familiarity with core programming concepts such as variables, loops, conditionals, and functions, gained from experience with any programming language.
- Basic Systems Knowledge: Understanding of memory management principles and general software development practices.

Course Outline:

1. Introduction

2. What is Rust?

    - Rust's Philosophy and Goals
    - History and motivation
    - Rust Community
    - The Rust Playground

3. Install Rust

    - Rustup Script
    - macOS Homebrew
    - Platform Installers

4. Rust Editors & AI Tools

- VSCode with Extensions
- Rust Rover
- Debug Rust in VSCode
- GitHub Copilot & ChatGPT

5. Hello World
    - Create a new Project
    - Main Function
    - Print to the Console
    - Comments

6. Cargo
    - What is Cargo?
    - Run Command
    - Build Command
    - Build Release Command
    - Install Third-Party Crates

7. Scalar Types and Data
    - Rust Types
    - Constants
    - Immutable Variables
    - Mutable Variables

8. Code Logic
    - If & If-Let
    - Match

- Loop with Break & Continue
- While & While-Let Loop
- For-In Loop
- Control-Flow as Expressions

9. Functions

- Define a Function
- Call a Function
- Parameter Types
- Return Types
- Closure Functions

10. Modules

- Import Modules from Standard Library
- Import Modules from Third-Party Crates
- Define Custom Modules
- Import Custom Modules
- Nested Modules

11. Built-In Macros

- print! and println!
- format!
- assert!, assert_eq!, and assert_ne!
- vec!
- panic!

12. Memory Management

- Challenges with Manual Management
- Challenges with Garbage Collection
- Ownership & Borrowing
- Immutable & Mutable References
- Lifetimes

13. Strings and String Slices

- What is a String and a String Slice?
- String Slices
- Strings
- Convert Between Slices and Strings
- Parse Number from String
- Trim String
- Print Strings with Interpolation

14. Tuples

- What is a Tuple?
- Heterogeneous Elements
- Access Elements
- Destructuring
- Immutable

15. 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

16. 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
- Stuct Methods
- Constructor Pattern

17. 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

18. Collections and Iterators
    - Vectors, arrays, and slices
    - HashMaps and hash sets
    - Iteration and iterators

19. 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

20. 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

21. Pattern Matching

- What is Pattern Matching?
- Match Statement
- If-Let Statement
- While-Let Statement
- Destructuring Stucts and Tuples
- Pattern Matching with Enums
- Pattern Matching with Functions
- Pattern Matching and Ownership
- Refutability and Irrefutability

22. Error Handling

- Result Enum
- Unwrap & Expect
- Map Error
- ? Operator
- Handle Multiple Error Types with Box Dyn Error
- Handle Multiple Error Types with Custom Enum
- Handle Multiple Error Types with Anyhow

23. 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

24. Unsafe Rust

- What is Unsafe Rust?
- Raw Pointers
- Dereferencing Raw Pointers
- Calling Unsafe Functions
- Creating Safe Abstractions
- Unsafe Traits
- Unsafe Blocks
- Unsafe Superpowers
- C/C++ FFI

25. Macros and Metaprogramming

- What is a Macro?
- Define a Macro with macro_rules!
- Using Pattern Matching
- Define Expansion
- Use the Custom Macro

26. Tests

- What is a Test?
- Test Functions
- Test Organization
- Test Attributes
- Test Coverage
- assert!, assert_eq!, and assert_ne!

27. 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