CLOUD CONTRAPTIONS
COMPUTER PROGRAMMING & TRAINING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Mastering Rust

Duration

5 days

Description

The Mastering Rust course is a comprehensive deep dive into the world of Rust programming. Whether you're a seasoned developer or new to systems programming, this training will equip you with the skills and knowledge to become proficient with Rust. Discover the language's unique features, including its focus on safety, concurrency, and performance optimization. Explore advanced topics like macros, metaprogramming, and FFI integration while building real-world applications. By the end of this course, you'll have the mastery and confidence to tackle complex projects, write efficient code, and harness Rust's full potential in your software development endeavors.

Objectives

- Understand the Rust Philosophy
- Set Up and Navigate the Rust Environment
- Grasp Basic Rust Syntax and Semantics
- Learn Control Flow and Logic
- Learn Ownership and Borrowing Concepts
- Utilize Tuples, Enums, Structs, and Vectors
- Employ Pattern Matching
- Harness 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.

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Training Materials

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub in the form of documentation and extensive code samples. Students practice the topics covered through challenging hands-on lab exercises.

Software Requirements

Students will need a free, personal GitHub account to access the courseware. Student will need permission to install Rust and Visual Studio Code on their computers. Also, students will need permission to install Rust Crates and Visual Studio Extensions. If students are unable to configure a local environment, a cloud-based environment can be provided.

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
  - Stuct 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 Stucts 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