Cloud Contraptions
TRAINING & CONSULTING

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

Rust Refresher

Class Duration

14 hours of intensive training with live instruction delivered over two to four days to accommodate varied scheduling needs

Student Prerequisites

- Software engineers with some Rust programming experience

Target Audience

Software engineers who find Rust's memory-management model, particularly its lifetime system, challenging, as well as those who struggle with its error-handling patterns and data-structure design, can benefit from a focused refresher class.

Description

This refresher course equips software engineers with a concise, hands-on review of Rust's essential concepts, from scalar types and variable ownership to advanced topics such as lifetimes, pattern matching, and zero-cost generics. Participants will deepen their understanding of Rust's memory-safe model, master idiomatic use of enums, structs, and collections, and learn how to leverage traits and monomorphized generics to write performant, allocation-free code. Ideal for developers who have dabbled in Rust and want to solidify their skills for production-grade projects.

Learning Outcomes

- Explain ownership and borrowing rules and demonstrate idiomatic Rust usage that prevents data races and dangling references.
- Construct and manipulate tuples, enums, structs, and vectors, applying standard patterns for each data structure's typical use case.
- Use idiomatic pattern-matching techniques (match, if let, while let) to safely decompose and process complex data.
- Write generic functions and types that are monomorphized at compile time, showcasing strategies for eliminating heap allocations.
- Evaluate when dynamic traits and heap allocation are appropriate versus using static dispatch and stack-only data, and justify the choice.

Cloud Contraptions
TRAINING & CONSULTING

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 Code Extensions. If students are unable to configure a local environment, a cloud-based environment can be provided.

Training Topics

Scalar Types and Variables

- Primitive Types: Integers, Floats, Booleans, and Characters
- Constants and Static Values
- Immutability by Default
- Mutable Bindings with `mut`
- Variable Shadowing
- Scope and Block Expressions

Memory Management

- Pitfalls of Manual Memory Management
- Overhead of Garbage Collection
- Rust’s Ownership Model
- Move Semantics
- Borrowing and References
- Lifetime Annotations
- The Borrow Checker in Practice

Strings and String Slices

- Heap-Allocated `String` vs Stack-Based `&str`
- Creating and Manipulating String Slices
- Building and Modifying Owned Strings
- Converting Between `String` and `&str`
- Parsing Strings into Numeric Types

- Trimming, Splitting, and Joining
- String Formatting and Interpolation

Tuples

- Tuple Fundamentals and Use Cases
- Storing Heterogeneous Types
- Indexing and Accessing Elements
- Destructuring into Named Bindings
- Unit Tuples and Empty Returns
- Tuples as Lightweight Data Containers

Enums

- Defining Enums and Variants
- Variants with Associated Data
- Implementing Methods on Enums
- The `Option<T>` Type for Nullable Values
- The `Result<T, E>` Type for Error Handling
- When to Use Enums vs Structs

Structs

- Defining Named-Field Structs
- Creating and Initializing Instances
- Field Init Shorthand Syntax
- Updating Instances with Struct Update Syntax
- Tuple Structs for Positional Data
- Unit-Like Structs as Markers
- Struct Ownership and Borrowed Fields
- Implementing Methods with `impl`
- Associated Functions and Constructors

Vectors

- The `Vec<T>` Growable Array Type
- Creating Vectors with `vec!` and `Vec::new()`
- Pushing, Popping, and Inserting Elements
- Safe Access with `get()` vs Direct Indexing
- Iterating with `iter()`, `iter_mut()`, and `into_iter()`
- Slicing, Length, and Capacity Management
- Memory Layout and Reallocation Behavior

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

- Ownership and Borrowing with Vector Elements

Pattern Matching

- Pattern Matching Fundamentals
- The `match` Expression and Exhaustiveness
- Concise Matching with `if let`
- Looping with `while let`
- Destructuring Structs, Tuples, and Enums
- Guards and Bindings in Patterns
- Ownership and Borrowing in Patterns
- Refutable vs Irrefutable Patterns

Generics and Traits

- Generic Type Parameters in Functions and Structs
- Defining and Implementing Traits
- Trait Bounds and `where` Clauses
- Static Dispatch via Monomorphization
- Binary Size Implications of Generics
- Dynamic Dispatch with Trait Objects (`dyn Trait`)
- Choosing Between Static and Dynamic Dispatch