

**Introduction to Rust Programming**

**Course Number:** RUST-102  
**Duration:** 3 days

**Overview**

This Introduction to Rust training course gives attendees a foundational understanding of the Rust programming language. Participants learn Rust fundamentals, including how to create custom data structures, work with pattern matching, and master modern features. This Rust course is appropriate for both beginners and experienced software developers.

**Prerequisites**

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

**Materials**

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

**Software Needed on Each Student PC**

A complete, remote virtual environment is provided for training and is accessible via the Internet from any modern web browser.

**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
* Use pattern matching

**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!
  + 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
  + Iteration and iterators
  + Iterate over Elements
  + Slicing, Length, and Capacity
  + Common Vector Operations
  + Understand Memory Management
  + Ownership and Borrowing Rules
* 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
* Conclusion