

**Rust for C# Developers**

**Course Number:** RUST-112  
**Duration:** 5 days

**Overview**

This Rust training course gives C# developers a smooth transition from C# to Rust by leveraging their existing object-oriented programming knowledge. Participants learn how to use Rust's innovative features, including its ownership model, memory safety, and fine-grained control over system resources.

**Prerequisites**

All students must be proficient in Python programming.

**Materials**

All 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 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.

**Objectives**

* Understand the Rust philosophy
* Set up and navigate the Rust environment
* Explore Rust within the context of C#
* Understand basic Rust syntax and semantics
* Use control flow and logic
* Understand ownership and borrowing concepts
* Use tuples, enums, structs, and vectors
* Employ pattern matching
* Implement Rust’s concurrency model
* Create custom macros
* Write Rust tests
* Create documentation with Rustdoc

**Outline**

* Introduction
* What is Rust?
  + Rust’s Philosophy and Goals
  + History and motivation
  + Rust vs. C# & .NET
  + 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?
  + How does Cargo compare to Pip and Conda?
  + Rust Crates compared to Python Packages
  + Run Command
  + Build Command
  + Build Release Command
  + Install Third-Party Crates
* Rust and C# Differences
  + Memory Management
  + Error Handling
  + Sequence, Selection, and Iteration
  + Structs vs Classes
  + Traits vs Protocols
  + Generics
  + Concurrency
* Scalar Types and Data
  + Rust Types vs C# Types
  + Constants
  + Immutable Variables
  + Mutable Variables
* Code Logic
  + If Statement
  + Loop with Break
  + While Loop
* Functions
  + Define a Function
  + Call a Function
  + Paramter 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
  + Rust vs C#
  + References
  + Lifetimes
* Strings
  + String Slices
  + String Objects
  + Convert Between Slices and Strings
  + Parse Number from String
  + Trim String
  + Print Strings with Interpolation
* Tuples
  + What is a Tuple?
  + Rust Tuples vs. C# Tuples
  + Heterogeneous Elements
  + Access Elements
  + Destructuring
  + Immutable
* Enums
  + What is an Enum?
  + Rust Enums vs. C# Enums
  + 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?
  + Rust Structs vs. C# Structs
  + 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?
  + Rust Vectors vs. C# Lists
  + 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 C# 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 C# 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