

Updated 05/17/2024

Sign up

Rust training

4 days (28 hours)

Presentation

Our Rust training course will introduce you to this new Open Source programming language created by Mozilla Research, designed to help developers build ultra-fast, secure applications. It avoids segmentation errors and ensures [thread](#) safety, all with elegant syntax. It offers excellent performance thanks to compiler-level memory management and native compilation.

During this course, you'll learn all the capabilities Rust has to offer, such as a high level of abstraction, guaranteed memory security, concurrency-free threads, templates: trait-based generics, pattern matching, type inference, and efficient C binding, all with minimal runtime!

Its design will enable you to create programs that have the performance and control of a low-level language such as C, but with the powerful abstractions of a high-level language (C#, Java...). These properties make Rust suitable for programmers with experience in languages like C/C++ or Python, who are looking for a safer alternative to write code that works more efficiently without sacrificing expressiveness.

Rust's motto is: Performance, Safety, Productivity - choose three!

In this training course, you'll learn how to use all the concepts of Rust: borrowing, ownership, lifetimes, mutability, generics and much more to develop your next projects efficiently!

As with all our training courses, this one will introduce you to the latest version of Rust Programming Language, [Rust 1.78](#).

Objectives

- Discover, understand and use the Rust language and ecosystem in your future projects

- Using concurrent programming and error handling with Rust
- Interfacing Rust with another language (Python, for example)

Target audience

Developers.

Prerequisites

- Basic knowledge of a Unix system
- Programming skills (C, C#, Java)

Software requirements

- Installing Docker and Docker Compose
- Installing gnuplot

Recommendations for pre- and post-course reading

- [An article](#) that shows a case study on writing a Rust application with only a Minimal API.
- This [article introduces](#) the Rust programming language and explains why it is so popular.
- A video explaining in just a few minutes the Rust programming language created by Fireship

Rust training program

Each chapter contains its own exercises, and each part is followed by one or more practical exercises.

Introduction to Rust

- Introducing the Rust language
- Downloading and installing the Rust environment
 - rustup
 - freighter
 - rustc
 - crates.io
- My first program
- Find documentation
- Editors for Rust

THE BASICS OF RUST

- Basic types
 - Numeric, integer and ottant types
 - Booleans
 - Characters
 - Character strings
- Compound types
 - Tables
 - Couples
- Variables and mutability
 - Notion of mutability
 - Constants
 - Variable redefinition and shadowing
 - Static variables
- Functions
 - Function syntax
 - Instructions and expressions
- Blocks and spans
 - Blocks required
 - Life and blocks
 - Assignment block
- Control structures
 - if, else and else if
 - Loop loop
 - While loop
 - For loop
- Name space
 - Notion of crate
 - Imports from std
 - Type alias
- Unit tests: cargo tes

Composite data types

- Structures
 - Definition
 - Assignment
 - Use
 - Tuple structures
 - Unitary structures
- Listings
 - Form common to other languages
 - Enumeration with associated data
- Trait implementations
 - Derivative traits
 - Manual implementations
- Methods
 - Manufacturers
 - Non-mutable methods
 - Mutable methods

Flow controls based on pattern matching

- if let
 - Structure pattern test
 - Testing a variant of an enumeration
 - OR operator and multiple patterns
 - Range testing
 - Incomplete patterns: introduction to destructuring
- while let
- match
 - Difference with switch case
 - Using destructuring
 - Add Boolean expressions

Types for error handling

- Option
 - Option prototype
 - Creating and using an Option
 - Option std methods
 - Going further with Options
- Result
 - Result prototype
 - Creating and using a Result
 - Error management

Key concepts of the Rust language

Here, we're no longer talking about the basics; this module will be used to deepen or introduce concepts specific to the Rust language.

- Expressions
 - Conditional initialization of a variable
 - Using pattern matching and expressions
 - Expression block
- Possession
 - Transfer of possession or copy
 - Reclaiming possession
 - The problem of local modification
- Type inference
 - Different cases of inference
 - Conits type
 - Limitations of type inference
- References, borrowing and pointers
 - References
 - Implicit references and syntactic sugar
 - Mutability and exclusivity
 - Raw pointers
- Introduction to life
 - s de vie implicites
 - When you need to be explicit
 - Lifetime static

- Type slice
 - Definition
 - Use
 - Iteration
 - Methods
 - Mutable access
- Genericity
 - Standard and generic libraries
 - Generic type in a structure or enumeration
- Features
 - From line
 - Implementing From for an enumeration
- Panic, Safe and Unsafe Rust
 - Unsafe code
 - Panic prototype

Dynamic allocation

- Box
 - Introduction, why allocate dynamically?
 - Box and Trait Deref
 - Box use case
- Collections - Vector and HashMap
 - Vectors : Dynamically allocated array
 - Hashmap : Dictionary
- Type String
 - A dynamically allocated character string
 - String creation
 - String methods
 - String and &str type
 - About From and Into
- Arc, atomic reference counters
 - Arc characteristics
 - Proof of concept
 - About the Clone trait
 - Clone derivation
 - Recursive drift
 - Mandatory implementation of Clone for Copy

Functional Programming

Rust combines aspects of functional programming, such as closures and immutability, enriching its imperative and concurrent model.

- Operator lry
 - Introduction
 - Syntax det eet
- Closures
 - Difference with a function, capturing the environment
 - Capture by reference
 - Transfer of possession

- What is the functional approach
 - Introducing Iterators
 - Internal iterator principle
 - Example of a custom Iterator implementation
 - Iterator methods
 - A few examples of how to use iterators
- Iterators, mutability, exclusivity and RefCell
 - A borrow checker problem
 - Bypassing with RefCell

Concurrent programming

Rust excels in concurrent programming thanks to its property and borrowing system, ensuring thread safety without explicit locking.

- Guaranteed thanks to Send and Sync features
- Resource sharing between several threads
 - Using Arc
 - Mutex, lock and unlock
- MPSC or Multiple Producer Single Consumer
 - What is an MPSC
 - Initialization
 - Thread creation and use

Line creation

- Line definition
- Static polymorphism
- Orphan rule

FFI

- Calling up Rust from C/C++
- Calling up C/C++ from Rust

Project tree and crate creation

- Using a crate
 - Consulting crates.io
 - Integrating a crate into the project
 - Documentation of crates
 - Using crates in your Rust projects
- Creating a bookshop
 - Creating a default boilerplate with cargo
 - Fill in the cargo.toml file
 - Using the library

- Files, Folders and Visibility
 - Add file
 - Subfolders, presentation of the two techniques

Additional module (+1 day) : Advanced

The basics of concurrent programming

- Definition: Mandelbrot
- Parsing pair: command-line arguments
- Mapping pixels: complex numbers
- Tracer : Plotting the set
- Writing an image file
- Mandelbrot: Concurrent program
- Run: Mandelbrot plotter
- Security

Unsafe code

- Unsafe blocks
- Example: ASCII string type
- Unsafe functions, traits, Raw pointers
- Example: RefWithFlag
- Nullable pointers
- Type sizes & alignments
- Pointer arithmetic
- Moving into and out of memory
- Example: GapBuffer
- Interop: Foreign functions, calling function lib C & C++
- Common data representations
- Declaration: foreign functions & variables
- Raw interface libgit2, Safe interface libgit2

Further information

Companies concerned

This training course is aimed at both individuals and companies, large or small, wishing to train their teams in a new advanced computer technology, or to acquire specific business knowledge or modern methods.

Positioning on entry to training

Positioning at the start of training complies with Qualiopi quality criteria. As soon as registration is confirmed, the learner receives a self-assessment questionnaire which enables us to assess his or her estimated level of proficiency in different types of technology, as well as his or her expectations and personal objectives with regard to the training to come, within the limits imposed by the format selected. This

The questionnaire also enables us to anticipate any connection or internal security problems (intra-company or virtual classroom) that could be problematic for the follow-up and smooth running of the training session.

Teaching methods

Practical course: 60% Practical, 40% Theory. Training material distributed in digital format to all participants.

Organization

The course alternates theoretical input from the trainer, supported by examples, with brainstorming sessions and group work.

Validation

At the end of the session, a multiple-choice questionnaire verifies the correct acquisition of skills.

Sanction

A certificate will be issued to each trainee who completes the course.