+33 (0)1 48 10 09 46
formation@ambient-it.net
140 Avenue Jean Lolive 93500 Pantin - France

Updated 06/03/2025

Sign up

Rust training

4 days (28 hours)

Presentation

Our Rust training course will introduce you to this new Open Source programming language, designed to help developers build ultra-fast, secure applications.

With this course, you'll learn to use all the resources Rust has to offer, such as the high level of abstractions, guaranteed memory security, and templates like trait-based generics, pattern matching and type inference.

With our Rust training course, learn how to avoid segmentation errors while ensuring Thread safety. You'll be able to create programs that have the performance and control of a low-level language like C, with the power of a high-level language.

As with all our training courses, this one will introduce you to the latest version of Rust Programming Language, Rust 1.85.

Objectives

- Discover, understand and use the Rust language
- Using concurrent programming and error handling with Rust
- Interfacing Rust with another language (Python, for example)
- Control flows based on pattern matching

Target audience

- Developers
- Architects

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 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 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
  - Blocks and service life
  - 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 testing: the cargo test command

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 from 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

It's no longer a question of 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
  - Type of conflict
  - Limitations of type inference
- References, borrowing and pointers
  - References
  - Implicit references and syntactic sugar
  - Mutability and exclusivity
  - Raw pointers
- Introduction to service life
  - Implied useful lives
  - 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 features
  - 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 Iry
  - Introduction
  - Syntax det eet
- Closures
  - Difference with a function, capturing the environment
  - Capture by reference
  - Transfer of possession
- What is the functional approach
  - Introducing Iterateurs
  - Internal principle of the iterator
  - 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

- Defining a line
- Static polymorphism
- Orphan rule

FFI

- Calling 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 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 finalized, 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 for the training to come, within the limits imposed by the selected format. This questionnaire also enables us to anticipate any connection or security difficulties within the company (intra-company or virtual classroom) which 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, 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.