

Rust Programming

Audience Course Rust Programming

The course Rust Programming is intended for developers who want to learn how to program in Rust and others who want to understand Rust code.

Prerequisites Course Rust Programming

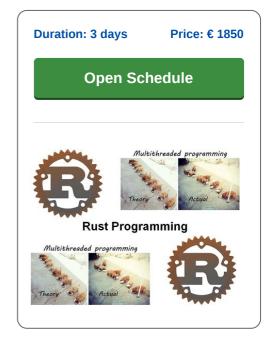
Experience programming in a modern programming language is desirable and beneficial to a good understanding.

Realization Training Rust Programming

The theory is treated on the basis of presentation slides. The concepts are illustrated with demos. The theory is interspersed with exercises. The course times are from 9.30 to 16.30.

Certification Rust Programming

The participants receive an official certificate Rust Programming after successful completion of the course.



Content Course Rust Programming

In the course Rust Programming participants learn to develop software with the latest version of the innovative programming language Rust.

Rust Fundamentals

Rust is a new, practical system programming language that produces lightning fast code. Rust is community driven. With Rust you prevent almost all crashes and data races.

Imperative and Functional

Rust builds on a rich history of programming languages. It is low-level language with multiple paradigms, both imperative and functional.

Concurrency and High Performance

Rust focuses on safe, high-performance, concurrent applications. Rust began to gain momentum in the industry before the official 1.0 version in May 2015, because there is a clear need for a new low-level system language.

Traits, Borrowing en Lifetimes

This course deals with what makes Rust so unique and applies this to practical problems of system programming. Topics that will be discussed are: traits, generics, memory safety, move semantics, borrowing and lifetimes.

Closures en Concurrency

And also the rich macro-system of Rust, closures and concurrency are discussed.

Houten, Amsterdam, Rotterdam, Eindhoven, Zwolle, Online



Modules Course Rust Programming

Module 2 : Data Types	Module 3 : Flow Control
Primitives	Expressions
Tuples and Arrays	Flow Control
Slices	if else
Custom Types	loop
Enums	Nesting and labels
Constants	while
Variable Bindings	for and range
Scope	match
Shadowing	Guards
Casting	Binding
Inference	if let
Alias	while let
Module 5 : Modules	Module 6 : Generics
Visibility	Functions
Struct Visibility	Implementations
use Declaration	Parametrization over Types
Using super	Traits
Using self	Bounds
File Hierarchy	Multiple Bounds
Crates	Where Clauses
Attributes	Associated Items
Extern crate	Associated Types
Dead Code	Phantom Type Parameters
Custom	Unit Clarification
Module 8 : Traits	Module 9 : Standard Library
Zero cost Abstraction	Box, stack, heap
Traits are interfaces	Data Structures
Derive	Vectors
Operator Overloading	Strings
Drop	Hashmap
Iterators	Threads Channels
Clone	Path
Designators	File I/O
Overload and Repeat	Pipes
Unsafe Operations	Wait
Static dispatch	Arguments
	Primitives Tuples and Arrays Slices Custom Types Enums Constants Variable Bindings Scope Shadowing Casting Inference Alias Module 5: Modules Visibility Struct Visibility use Declaration Using super Using self File Hierarchy Crates Attributes Extern crate Dead Code Custom Module 8: Traits Zero cost Abstraction Traits are interfaces Derive Operator Overloading Drop Iterators Clone Designators Overload and Repeat Unsafe Operations