Course Outline

# Introduction to Rust Programming for Developers

## Course Summary

### Description

Rust is “a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.” System programming languages, like C, must to be able to exert fine grained control over system resources, run fast with small memory footprints; however, this makes easy for programmers to write code that causes run time errors.

Rust is designed to produce code that runs as fast as C, but Rust’s syntax and semantics make it impossible to write code that would result in any form of run time memory error. As a second-generation systems language, Rust is able to work with modern CPU architectures to provide advanced features like low level, safe concurrency, and parallelism.

Even though the syntax and semantics are similar to C++, Rust introduces a range of novel program language constructs – such as owning, borrowing and loaning memory – that can make it difficult for developers to transition to Rust from other more standard languages.
This course is designed to help experience developers make the transition to Rust as seamless and painless as possible. Students are shown how to leverage their existing programming skills in learning Rust, but also emphasizes where they have to rethink how they code to take advantage of Rust innovative features .

As students work through the features of Rust in an instructor led, hands on manner, the underlying concepts behind Rust and what is happening “under the hood” are explored so that students don’t only how to write Rust code but, even more importantly, what Rust code to write and why it should be written that way. By the end of the course, students will be able to write Rust code that is consistent with Rust best coding and design practices.

The class is designed to be about 50% hands on labs and exercises, about 25% theory and 25% instructor led hands on learning where students code along with the instructor. Because the class will use the most recent version of Rust, some of the newer features of Rust that may be covered in class may not be listed in this outline.

### Topics

- Why Rust? Design goals of Rust – fast, efficient and safe systems programming.
- The Cargo build system and LLVM back-ends.
- Zero-cost abstractions and the compilation cycle.
- Data types, variables, pointers, slices and references.
- Ownership, mutability, references and borrowing.
- Rust flow of control constructs.
- Rust functions and macros.
- Defining and using structs.
- Enums and pattern matching.
- Collections – vectors, strings and hash maps.
- Recoverable errors and unrecoverable panics.
- Generic data types and shared behavior with Traits.
- Concurrent programming
- Usafe Rust and integration with C and C++

### Audience

This course is designed for programmers who want to get up to speed fast in Rust.

### Prerequisite

Before taking this course, an intermediate level of skill in and a solid knowledge of a high level programming language like Java, Python or C is essential. Students who do not have this prerequisite may struggle with the content and pace of the course.

### Duration

Three Days

Course Outline

# Introduction to Rust Programming for Developers

## Course Outline

*I. The World of Rust*
- A. Design goals of Rust
- B. System programming
- C. High-level overiew of Rust and the Rust toolchain
- D. Installing Rust, Cargo and Rustup
- E. Transitioning to Rust as a developer
- F. Status and future development of Rust

*II. Rust Language Basics*
- A. Variables and data types
- B. Structs and enums
- C. Arrays, slices and vec
- D. Strings
- E. Blocks, mutability, assignment and types
- F. Expressions and control flow
- G. Using the “match” statement
- H. Working with tuples
- I. Method chaining

*III. Ownership and Borrowing*
- A. Standard memory management
- B. Rust memory management and allocation
- C. Copyable values and moves
- D. Ownership and lifetimes, explicit lifetimes
- E. Mutable and immutable references
- F. Rust borrow checker

*IV. Rust Software Engineering*
- A. Error handling: panic, option and result
- B. Error handling: unwrap and expect
- C. Handling potential errors: Error combinators
- D. Crates and modules
- E. Libraries and binary crates
- F. Creating, using and deploying crates
- G. Using “pub”
- H. Semantic versioning
- I. Documentation and tests
- J. Test Driven Development with Rust

*V. More Structs and Enums*
- A. Empty structs, tuple structs and single-element structs
- B. Rust struct syntax
- C. Ownership and lifetimes of structs
- D. Using impl and deriving traits
- E. Match and pattern matching
- F. Using Rust enums
- G. Non-value carrying enums
- H. Void enums

*VI. Traits and Generics*
- A. Generics, type inference
- B. Parameterization
- C. Phantom types
- D. Traits as interface types
- E. Defining traits, bound object and orphan rules
- F. Subtraits
- G. Operator overloading
- H. Fundamental traits: copy, clone, marker traits, etc
- I. Using traints in sharing and borrowing

*VII. Closures and Iterators*
- A. First class functions and closures
- B. Understanding how closures work “under the hood”
- C. Iterators and adaptors
- D. Functional programming in Rust

*VIII. Collections, Strings and I/O*
- A. Basic collections in Rust: Vec, array, slice, etc
- B. Implementing advanced data structures
- C. Heaps, maps, hashing, Btrees, sets, linked lists, queues etc
- D. Third party collections
- E. Strings, characters and string operations
- F. Std I/O, buffers, reader and writers, network I/O

*IX. Concurrent Programming*
- A. Approaches to concurrent programming
- B. Multi-threading and message passing
- C. Shared-state concurrency
- D. Sync and Send Traits
- E. Futures, async and await
- F. Implementing concurrent programming
- G. Choosing a concurrency implementation

*X. Macros*
- A. Writing and debugging Rust macros
- B. Macro gotchas
- C. Procedural macros

*XI. Unsafe Rust and Legacy Integration*
- A. The “usafe” keyword
- B. Calling C and C++ from Rust
- C. Common data representations
- D. Raw pointers
- E. Unsafe functions, blocks and other code