🦀 RUST PARTICIPANTS — Prerequisites & Struggles

Learning Mindset

- Be ready to:
    - **Read error messages carefully**
    - **Unlearn** certain habits from Python or JavaScript
    - **Think with precision** (Rust forces explicit, safe choices)

Mental Model: Stack vs Heap

- At least a **conceptual** idea of how memory works:
    - What's stored in the **stack** vs the **heap**
    - What is a **pointer/reference** in high-level terms

🔸 **Why?** Rust emphasizes memory safety, ownership, and borrowing. These concepts are rooted in how memory is managed.

Rust has a steeper learning curve, so aim to reduce friction and fear of failure.

✅ Prerequisites (Mindset & Skills)

Category	What to Know or Prepare
Basic Programming	At least 6 months experience in any language
Memory Concepts	Basic stack vs heap knowledge
Error Handling	Understand try/catch, panic vs recover ideas
CLI Usage	Comfortable with terminal, basic cargo usage
Ownership/References	Read or watch an intro on "Rust Ownership" model

⚠️ Common Weaknesses / Struggles

Weakness Area	We Overcome it
Ownership & Borrowing	Offer visual diagrams + real-world analogies
Lifetimes	Avoid too early; introduce only with functions
Tooling Complexity	Walk through cargo, clippy, fmt slowly
Fear of Compiler Errors	Teach how to read and appreciate them
No Garbage Collector	Explain with memory safety demos

Rust _Content

Duration: 6 Days (48 hrs)

Content:

Day 1

Introduction

- Java/C++ to Rust Comparison
- Why Rust

- **Hands on code comparison**

MEMORY SEGMENTS

- STACK
- HEAP
- DATA
- CODE/TEXT

RAII (Resource acquisition is initialization)

- Ownership and moves
- Borrowing
- Lifetimes

Hands on code.

Program on all topics

Case studies.

Where to apply this topic.

RUST -Function

- Closures
- Capturing of data in closures
- Closures as input to functions and output parameters
- Higher order functions

Hands on code.

Program on all topics

Case studies.

Where to apply this topics .

Day 2

RUST BASICS

OOPS

- OOP Concepts in RUST VS CPP /Java.

Hands on code.

Program on all topics

Case studies.

Where to apply this topics .

MODULE

- Purpose of modules in rust code
- Defining own custom module
- private and public visibility of members in a module
- use keyword for deep modules
- super and self keyword

ARRAY TYPES

C++ VS RUST
Tuple data type

- Array data type
- Slice data type

Error Handling

panic. Panic values `abort` and `unwind`

- Option and unwrap
- Result. Iterating over Results
- Multiple error types

Day 3

Introduction to generics

- Defining generic functions
- Making implementation generic
- Defining Bounds for generic type
- The newtype idiom
- Item association
- Phantom type parameters

Introduction to traits in Rust

- Derivable traits
- `dyn` keyword
- Operator overloading using traits
- Drop trait
- Iterator trait
- impl trait
- Clone trait
- Supertraits

Hands on code.

Program on all topics

Case studies.

Where to apply this topics .

DATA STRUCTURES

Box type

- Vectors
- Strings
- Option
- Result
- HashMap
- Rc and Arc

Similar to C FFI, I suggest to cover C++ FFI. since C++ is based on class, objects, STL, exceptions and templates, which requires a different thought process to work with it.

Day 4

What are smart pointers

- Box<T> for data on the heap
- The Deref and Drop trait
- Rc<T> and RefCell<T>

Hands on code.

Program on all topics

Case studies.

Where to apply this topics .

Introduction to concept of crates in rust

- Custom library creation and linking to another crate
- Cargo as Rust package management tool
- Managing dependencies using the cargo tool
- Using cargo to run unit and integration tests
- Cargo build scripts

Fearless concurrency: Rust's Approach

- Threads and thread safety
- Shared state concurrency:
  - Mutexes
  - Atomic types
- Message passing with channels
- Async/Await and the Tokio runtime
- Lock-free programming techniques

Hands on code.

Program on all topics

Case studies.

Where to apply this topics .

Difference between threading and async programming

- async/.await syntax
- Future trait
- Pinning
- The Stream trait
- join!, select!

- Shortcomings of the async programming model in Rust

Hands on code.

Program on all topics

Case studies.

Where to apply this topics .

Concurrency pattern

- Worker poll implementation

Day 5

Rust for block chain

🧱 1. Contract Architecture & State Management

- **Persistent Storage**: Understand how to manage on-chain state using data structures like HashMap, Vec, and custom structs.
- **Serialization**: Master serialization and deserialization with serde or borsh, as these are essential for storing and retrieving contract state.
- **Entry Points**: Define clear and secure public methods for contract interaction.

🔐 2. Security Best Practices

- **Ownership & Access Control**: Implement role-based access controls to restrict function access appropriately.
- **Error Handling**: Utilize Rust's Result and Option types for robust error management.
- **Common Vulnerabilities**: Be aware of issues like unchecked arithmetic operations, reentrancy attacks, and improper input validation.

Working with Databases

🔗 Interacting with APIs

Smart contracts often need to communicate with external services to fetch data or trigger actions. In Rust, this is typically achieved through:

- **HTTP Clients**: Libraries like reqwest or hyper are used to make HTTP requests to external APIs.

- **Blockchain APIs**: For Ethereum, crates like ethers-rs or web3 allow interaction with the blockchain, enabling functionalities such as querying contract states or sending transactions. CoinsBench

- **Custom APIs**: Building RESTful APIs using frameworks like axum or actix-web enables your dApp to serve data to front-end applications or other services

- Working with a SQL Database
- Serving a JSON API
- Testing and Building

Day 6

Rust for Linux

🧩 Writing Kernel Modules in Rust

- **Creating Basic Kernel Modules: Start by writing simple "Hello World" modules in Rust to understand the kernel module infrastructure.**

- **Interfacing with C Code: Learn how to interface Rust code with existing C kernel APIs, which is essential for driver development.**

HANDSON PROJECT

🛠️ Project: Build a Simple Blockchain in Rust

Objective: Create a basic blockchain that includes blocks, hashing, proof-of-work mining, and validation

Key Concepts:

- **Blocks**: Each block contains data, a timestamp, and a hash of the previous block.

- **Hashing**: Securely link blocks using cryptographic hashes.

- **Mining**: Implement a simple proof-of-work algorithm to add new blocks.
- **Validation**: Ensure the integrity of the blockchain by verifying the chain's validity.

Tools:

- **Rust**: Programming language for building the blockchain.
- **SHA-256**: Cryptographic hash function for hashing block data.
- **Cargo**: Rust's package manager and build system.

Best Practices and Design Patterns

- Rust idioms and coding style
- Common design patterns in Rust
- Optimizing Rust code for embedded systems | backend
- Code organization for large-scale projects
- Rust-specific anti-patterns to avoid

Tools and Ecosystem (Some parts can be included)

- Advanced Cargo usage
- Debugging Rust programs
- Profiling and benchmarking
- Continuous Integration for Rust projects
- Useful crates for automotive development