1. Introduction & Setup

A. What is Rust? — System-level programming language overview
B. Rust's history, features and motivations (speed, safety, no garbage collection)
C. Difference vs C/C++, Java, and other languages
D. Installing Rust toolchain (rustup)
E. Using Cargo (build tool and package manager)
F. IDE setup (VSCode, IntelliJ Rust, Rust Analyzer)

2. Basic Language Concepts

A. Rust syntax and program structure
B. Comments, formatting, compiling
C. Variables, mutability, scope and shadowing
D. Data types — scalar and compound
E. Constants and static variables
F. Expressions vs statements
G. Printing and basic input/output

3. Control Flow

A. Conditional statements (if, else)
B. match expression and pattern matching
C. Loops — loop, while, for
D. Early exit and conditional expressions
E. Flow control with enums and patterns

4. Functions & Modules

A. Defining functions
B. Parameters and return types
C. Modules, packages, crates
D. Visibility (pub)
E. Cargo workspace and project structure

5. Ownership, Borrowing & Lifetimes (Core Rust Concept)

A. Ownership rules and memory model
B. Borrowing & references
C. Mutable and immutable references
D. Lifetimes and lifetime elision rules
E. Move vs Copy semantics

F. Slices and string basics

6. Structs and Enums

A. Defining and using structs
B. Tuple structs and unit structs
C. Enums and match with enums
D. Using Option and Result types
E. Data modeling with enums

7. Traits and Generics

A. Generic functions and types
B. Trait definitions and implementations
C. Trait bounds
D. Associated types
E. Default implementations
F. Using derive for common traits

8. Error Handling

A. Panic! and error handling theory
B. Result type and pattern matching
C. Error propagation (? operator)
D. Custom error types
E. Logging and debugging basics

9. Collections & Iterators

A. Vectors, Strings, HashMap
B. Iterators and Iterator trait
C. Functional style with iterators
D. Closures and anonymous functions

10. Smart Pointers & Memory Management

A. Box pointers
B. Reference counting (Rc, Arc)
C. Interior mutability (RefCell)
D. Avoiding reference cycles
E. Comparison with manual memory management

11. Concurrency

A. Safe concurrency principles in Rust
B. Threads and std::thread
C. Shared state concurrency
D. Mutex and message passing
E. Send and Sync traits

12. Advanced Topics

A. Asynchronous programming (async/await)
B. Macros
C. Unsafe Rust and FFI (Foreign Function Interface)
D. Testing & benchmarking
E. Cargo features and build profiles
F. Documentation with rustdoc

13. Project Work / Applications

A. Command-line application with arguments
B. Building a systems utility tool
C. Web or network service using Rust (basic)
D. Performance benchmarking and optimization
E. Using third-party crates (e.g., Serde, Tokio)

Practical:

Hands-on exercises include:

- Rust environment setup & first programs
- Debugging and using Cargo commands
- Ownership and borrowing practice
- Structs & enums projects
- Generic & trait based exercises
- Error handling implementations
- Multithreaded sample applications
- Mini project integrating multiple modules